Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Методы численного анализа

ОТЧЁТ

к лабораторной работе
на тему

Метод сеток решения одномерного нестационарного уравнения теплопроводности

Выполнил: студент группы 153501
Тимофеев Кирилл Андреевич

Проверил: Анисимов Владимир Яковлевич

Минск 2023

Цели выполнения задания

- Изучить метод разностных аппроксимаций для уравнения теплопроводности;
- Составить алгоритмы решения уравнения теплопроводности методом сеток, применимыми для организации вычислений на ПЭВМ;
- Составить программы решения уравнения теплопроводности по разработанным алгоритмам;
- Выполнить тестовые примеры и проверить правильность работы программ;
- Получить численное решение заданного уравнения теплопроводности.

Краткие теоретические сведения

Требуется найти непрерывную на замкнутом прямоугольнике $\overline{D}$ функцию $u(x, t)$, которая на D' удовлетворяет уравнению теплопроводности

$$Lu \equiv \frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = f(x,t), \qquad (2.10)$$

которое при $t = 0$ удовлетворяет начальному условию

$$u(x, 0) = s(x), \qquad (2.11)$$

а при $x = 0$ и $x = 1$ подчиняется краевым условиям

$$u(0, t) = p(t), \;\; u(1, t) = q(t), \qquad (2.12)$$

где $f(x, t)$, $s(x)$, $p(t)$, $q(t)$ − заданные достаточно гладкие функции, причем $s(0) = p(0)$, $s(l) = q(l)$.

Задача (2.10) − (2. 12) называется *смешанной задачей*, поскольку она содержит как *начальные условия*, так и *краевые условия*. Известно, [11] что у поставленной задачи существует единственное решение $u(x, t)$. Мы будем предполагать, что это решение имеет на замкнутом прямоугольнике $\overline{D}$ непрерывные частные производные $\partial u/\partial t$, $\partial^2 u/\partial t^2$, $\partial^2 u/\partial x^2$, $\partial^4 u/\partial x^4$.

Сетки и нормы.

Пусть $h = 1/N$, $\tau = T/M$ − шаги по x и t, где N, M – натуральные числа, а $x_k = kh$, $t_\nu = \nu\tau$, $u_k^\nu = u(x_k, t_\nu)$. Построим сетки (рис. 2.1)

$\omega_h = \{(x_k, t_\nu):\ k = 0, 1, \ldots, N,\ \nu = 0, 1, \ldots, M\}$,

$\omega_h' = \{(x_k, t_\nu):\ k = 1, 2, \ldots, N\text{-}1,\ \nu = 1, 2, \ldots, M\}$,

$\omega_h^* = \omega_h \setminus \omega_h'$.

Сетка ω_h^* состоит из узлов сетки ω_h, обозначенных на рис. 2.1 крестиками. Эти узлы расположены на трех сторонах прямоугольника $\overline{D}$, на которых заданы начальное и краевые условия. Сетка ω_h' состоит из остальных узлов сетки ω_h. Зададим для сеточных функции определенных на ω_h или на ω_h', следующие нормы:

$$\|y\|_h = \max_{\omega_h}|y_k^\nu|, \quad \|y\|_h' = \max_{\omega_h'}|y_k^\nu|. \qquad (2.13)$$

Разностные схемы.

Введем разностный оператор Λ:

$$\Lambda y_k^\nu = -\frac{y_{k-1}^\nu - 2y_k^\nu + y_{k+1}^\nu}{h^2}. \qquad (2.14)$$

Здесь под выражением Λy_k^ν подразумевается значение сеточной функции Λy в точке с координатами (x_k, t_ν), т. е. $(\Lambda y)_k^\nu$.

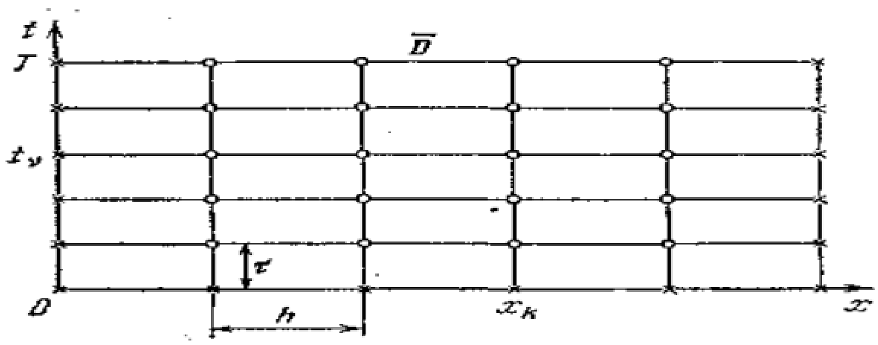


Рис. 2.1

Скобки в выражении (2.14) опущены для упрощения записи. Аналогичные упрощения в записи будем допускать и при введении других операторов. Зададим на сетке ω_h^* тождественный оператор

$$l^h y \equiv y \tag{2.15}$$

и сеточную функцию

$$g = \begin{cases} s(x_k), & x = x_k, \quad t = 0, \quad k = 1, \ 2, \ldots, N-1 \\ p(t_\nu), & x = 0, \quad t = t_\nu, \quad \nu = 0, \ 1, \ldots, M, \\ q(t_\nu), & x = 1, \quad t = t_\nu, \quad \nu = 0, \ 1, \ldots, M. \end{cases} \tag{2.16}$$

Рассмотрим две разностные схемы:

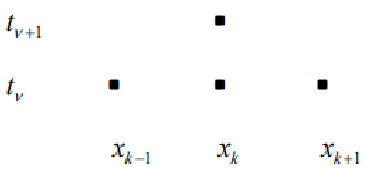
$$L_1^h y_k^\nu \equiv \frac{y_k^\nu - y_k^{\nu-1}}{\tau} + \Lambda y_k^{\nu-1} = f_k^{\nu-1}, \tag{2.17}$$

$$l^h y = g, \tag{2.18}$$

$$L_2^h y_k^\nu \equiv \frac{y_k^\nu - y_k^{\nu-1}}{\tau} + \Lambda y_k^\nu = f_k^\nu, \tag{2.19}$$

$$l^h y = g. \tag{2.20}$$

Здесь и далее индекс k изменяется от 1 до N-1, $\nu = 1, 2, \ldots, M$. Шаблоны разностных уравнений (2.17) и (2.19) представлены соответственно на рис. (2.2) и (2.3). Обе разностные



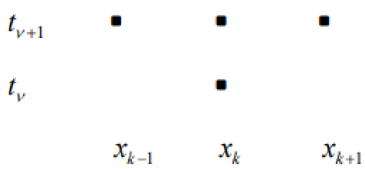


Рис. 2.2 Рис. 2.3

схемы (2.17)–(2.18) и (2.19)– (2.20) называются *двухслойными*, так как шаблоны разностных уравнений (2.17) и (2.19) содержат узлы, лежащие только на двух временных *слоях* − подмножествах сетки ω_h, отвечающих значениям времени $t = t_{\nu-1}$ и $t = t_\nu$. Слой, находящийся на горизонтальной прямой $t = t_{\nu-1}$, называется *нижним*, а слой, находящийся на горизонтальной прямой $t = t_\nu$, − *верхним*. Разностные схемы (2.17), (2.18) и (2.19), (2.20) отличаются тем, что в уравнении (2.17) оператор Λ действует на нижнем слое, а в уравнении (2.19) оператор Λ вынесен на верхний слой, и, кроме того, значения правой части $f_k^{\nu-1} = f(x_k, t_{\nu-1})$ и $f_k^\nu = f(x_k, t_\nu)$ берутся на разных слоях. Ограничимся пока предложенным формальным описанием двух разностных схем. Их качественное различие будет описано далее в данной лабораторной работе.

Аппроксимация.

Сопоставляя, с одной стороны, дифференциальное уравнение (2.10), а, с другой стороны, разностные уравнения (2.17) и (2.19), видим, что частной производной u_t' отвечает разностная производная $\dfrac{y_k^\nu - y_k^{\nu-1}}{\tau}$, а частной производной $-u_{xx}''$ соответствует разностная производная второго порядка в направлении *x*, образуемая с противоположным знаком с помощью оператора Λ (2.14).

Пусть $u(x,t)$ − решение задачи (2.10) − (2.12). Поскольку частные производные $\partial^2 u / \partial t^2$ и $\partial^4 u / \partial x^4$ по предположению непрерывны и, следовательно, ограничены на замкнутом прямоугольнике $\overline{D}$, то согласно (2.14),

$$\Lambda y_k^{\nu-1} = -u_{xx}''(x_k, t_{\nu-1}) + r_k^\nu, \tag{2.21}$$

$$\frac{u_k^\nu - u_k^{\nu-1}}{\tau} = u_t'(x_k, t_{\nu-1}) + \rho_k^\nu, \tag{2.22}$$

где $k = 1, 2, \ldots, M\text{-}1$, $\nu = 1, 2, \ldots, N$,

$$\left|r_k^\nu\right| \le c_1 h^2, \ \left|\rho_k^\nu\right| \le c_2 \tau, \tag{2.23}$$

а c_1, c_2 − некоторые постоянные, не зависящие от h, τ, k, ν. В силу непрерывности частных производных u'_t и u''_{xx}, на $\overline{D}$ решение задачи (2.10) − (2.12) удовлетворяет уравнению (2.10) на замкнутом прямоугольнике $\overline{D}$.

Следовательно, выполняется равенство

$$u'_t(x_k,t_{\nu-1}) - u''_{xx}(x_k,t_{\nu-1}) = f_k^{\nu-1} \tag{2.24}$$

для $k = 1, 2, \ldots, N-1$, $\nu = 1, 2, \ldots, M$, т. е., в частности, и для $t_{\nu-1} = 0$.

Согласно (2.21), (2.22), (2.23) невязка ψ_1 решения u задачи (2.10) — (2.12) для разностного уравнения (2.17) имеет следующее выражение:

$$\psi_{1k}^{\nu} = L_1^h y_k^{\nu} - f_k^{\nu-1} = \frac{u_k^{\nu} - y_k^{\nu-1}}{\tau} + \Lambda y_k^{\nu-1} - f_k^{\nu-1} =$$

$$= u'_t(x_k,t_{\nu-1}) + \rho_k^{\nu} - u''_{xx}(x_k,t_{\nu-1}) + r_k^{\nu} - f_k^{\nu-1} = r_k^{\nu} + \rho_k^{\nu}.$$

Отсюда с учетом (2. 23) получаем

$$\|\Psi_1\|'_h = \max_{\omega_h^{\prime}} \left|\Psi_{1k}^{\nu}\right| = \max_{1\le\nu\le M} \max_{1\le k\le N-1} \left|r_k^{\nu} + \rho_k^{\nu}\right| = O(h^2+\tau). \tag{2.25}$$

Аналогично находим

$$\|\Psi_2\|'_h = O(h^2+\tau), \tag{2.26}$$

где Ψ_2 − невязка решения u задачи (2.10) − (2.12) для разностного уравнения (2.19).

Таким образом, оба разностных уравнения (2.17) и (2.19) аппроксимируют дифференциальное уравнение (2.10) на решении u задачи (2.10) — (2.12) со вторым порядком по h и с первым порядком по τ.

Дополнительные условия, т. е. начальное условие (2.11) и краевые условия (2.12), аппроксимируются на сетке ω_h^* с помощью тождественного оператора l^h условием (2.18) или соответственно условием (2.20) точно, т. е. невязка решения u задачи (2.10) − (2.12) для условий (2.18) и (2.20) равна нулю на сетке ω_h^*.

Итак, обе разностные схемы (2.17), (2.18) и (2.19), (2.20), с точки зрения аппроксимации задачи (2.10) − (2.12), обладают одинаковой по порядку относительно h и τ гарантируемой точностью.

Вычислительные алгоритмы

Разрешив разностное уравнение (2.17) относительно y_k^ν, получим

$$y_k^\nu = \frac{\tau}{h^2} y_{k-1}^{\nu-1} + (1 - \frac{2\tau}{h^2}) y_k^{\nu-1} + \frac{\tau}{h^2} y_{k+1}^{\nu-1} + \tau f_k^{\nu-1}. \qquad (2.27)$$

Поскольку y_k^0, y_0^ν, y_N^ν, k = 1, 2, . . . , N–1, ν = 0, 1, . . . , M известны (они задаются на ω_h^* условием (2.18)), решение разностной схемы (2.17), (2.18) находится по формуле (2.27) явно, слой за слоем. Поэтому разностная схема (2.17), (2.18) называется *явной*.

Разностное уравнение (2.19) с учетом (2.14) может быть записано в виде

$$\frac{\tau}{h^2} y_{k-1}^\nu - (1 + \frac{2\tau}{h^2}) y_k^\nu + \frac{\tau}{h^2} y_{k+1}^\nu = -y_k^{\nu-1} - \tau f_k^\nu. \qquad (2.28)$$

Согласно (2.15), (2.16), (2.20) имеем также

$$y_0^\nu = \rho_\nu, \quad y_N^\nu = q_\nu \qquad (2.29)$$

Таким образом, если $y_k^{\nu-1}$, k = 1, 2, . . . , N–1, известны (в частности, y_k^0, k = 1, 2, . . . , N–1, заданы условием (2.20)), то для нахождения решения разностной схемы (2.19), (2.20) на следующем ν-м слое нужно решить трехточечное разностное уравнение (2.28) с краевыми условиями первого рода (2.29), т. е. разностную краевую задачу. Поэтому разностная схема (2.19), (2.20) называется *неявной*.

Для нахождения разностного решения на ν-м слое может быть применен метод прогонки, [2] поскольку для задачи (2.28), (2.29) достаточные условия выполнены (проверьте, положив $k = j$, $y_k^\nu = z_j$, $y_{k\pm1}^\nu = z_{j\pm1}$, $-y_k^{\nu-1} - \tau f_k^\nu = F_j$). При этом число выполняемых арифметических действии для нахождения разностного решения на одном слое имеет величину порядка $O(N)$, т. е. по порядку относительно N не больше, чем при применении явной формулы (2.27) для схемы (2.17), (2.18).

Устойчивость и сходимость

Так как дополнительные условия (2.11), (2.12) аппроксимируются в разностных схемах (2.17), (2.18) и (2.19), (2.20) на сетке ω_h^* точно, то нам будет достаточно исследовать устойчивость только по правой части.

Остановимся сначала на разностной схеме (2.17), (2.18). Для исследования ее устойчивости по правой части нужно рассмотреть решение z вспомогательной разностной задачи:

$$L_1^h z_k^\nu \equiv \frac{z_k^\nu - z_k^{\nu-1}}{\tau} + \Lambda z_k^{\nu-1} = \xi_k^\nu, \qquad (2.30)$$

$$l^h z = 0, \qquad (2.31)$$

где ξ − произвольная заданная на ω_h' сеточная функция.

Разрешив разностное уравнение (2.30) относительно z_k^ν, аналогично (2.27) получаем

$$z_k^\nu = \frac{\tau}{h^2} z_{k-1}^{\nu-1} + \left(1 - \frac{2\tau}{h^2}\right) z_k^{\nu-1} + \frac{\tau}{h^2} z_{k+1}^{\nu-1} + \tau \xi_k^\nu, \qquad (2.32)$$

где $K = 1, 2, \ldots, N-1,\ \ \nu = 1, 2, \ldots, M.$

Кроме того, в соответствии с (2.22) имеем

$$\begin{aligned} &z_k^0 = 0,\ k = 1, 2, \ldots, N-1; \\ &z_0^\nu = z_N^\nu = 0,\ \nu = 0, 1, \ldots, M. \end{aligned} \qquad (2.33)$$

Предположим, что τ и h удовлетворяют следующему условию:

$$\frac{\tau}{h^2} \le \frac{1}{2}. \qquad (2.34)$$

Тогда очевидно, что

$$\frac{\tau}{h^2} + \left|1 - \frac{2\tau}{h^2}\right| + \frac{\tau}{h^2} = 1.$$

Отсюда и из (2.32), (2.33) вытекает неравенство

$$\max_{0 \le k \le N} \left|z_k^\nu\right| \le \max_{0 \le k \le N} \left|z_k^{\nu-1}\right| + \tau \max_{1 \le k \le N-1} \left|\xi_k^\nu\right|, \qquad (2.35)$$

и поскольку $\max\limits_{0 \le k \le N} \left|z_k^0\right| = 0$, то

$$\max_{0 \le k \le N} \left|z_k^\nu\right| \le \nu\tau \left\|\xi\right\|'_h.$$

Следовательно,

$$\|z\|_h = \max_{0\le\nu\le M}\ \max_{0\le k\le N}\left|z_k^\nu\right| \le M\tau\ \|\xi\|'_h = T\ \|\xi\|'_h$$

или, окончательно,

$$\|z\|_h \le T\ \|\xi\|'_h\ . \tag{2.36}$$

Полученное неравенство (2.36) для решения задачи (2.30), (2.31), в котором постоянная T не зависит от h и τ, а также от функции ξ, означает устойчивость разностной схемы (2.17), (2.18) по правой части при условии (2.34). Можно доказать, что нарушение условия (2.34) может привести к нарушению устойчивости разностной схемы (2.17), (2.18). В частности, если $h\to 0$, $\tau\to 0$, а $\tau/h^2 \ge$ const $> 1/2$, то разностная схема (2.17), (2.18) будет неустойчива.

Для исследования устойчивости разностной схемы (2.19), (2.20) зададим на ω'_h произвольную сеточную функцию ξ и рассмотрим разностную задачу

$$L_2^h z_k^\nu \equiv \frac{z_k^\nu - z_k^{\nu-1}}{\tau} + \Lambda z_k^\nu = \xi_k^\nu\,, \tag{2.37}$$

$$l^h z = 0\,, \tag{2.38}$$

не накладывая никаких ограничений на соотношение шагов τ и h. Задачу (2.37), (2.38) можно аналогично (2.28), (2.29) записать в следующем виде:

$$\frac{\tau}{h^2} z_{k-1}^\nu - (1+\frac{2\tau}{h^2}) z_k^\nu + \frac{\tau}{h^2} z_{k-1}^\nu = - z_k^{\nu-1} - \tau\xi_k^\nu\,, \tag{2.39}$$

$$z_0^\nu = 0\,,\ z_N^\nu = 0\,. \tag{2.40}$$

Если $z_k^{\nu-1}$, $k = 1, 2, \ldots, N-1$, известны (в частности, по условию (2.38) $z_k^0 = 0$, $k = 0, 1, \ldots, N$), то, как отмечалось ранее, для разностной задачи (2.39), (2.40), где ν фиксировано, выполнены условия (2.12). Следовательно, по лемме эта задача однозначно разрешима на ν-м слое.

Очевидно, имеется такое k', $0<k'<N$, что

$$\left|z_{k'}^{\nu}\right| = \max_{0 \le k \le N}\left|z_{k}^{\nu}\right|. \qquad (2.41)$$

Так как $\left|z_{k'-1}^{\nu}\right| \le \left|z_{k'}^{\nu}\right|$, $\left|z_{k'+1}^{\nu}\right| \le \left|z_{k'}^{\nu}\right|$, то

$$\left|z_{k'}^{\nu}\right| \le \left|z_{k'}^{\nu}(1+\frac{2\tau}{h^2}) - \frac{\tau}{h^2}\left(z_{k'-1}^{\nu} + z_{k'+1}^{\nu}\right)\right|$$

и, следовательно, согласно (2.39)

$$\left|z_{k'}^{\nu}\right| \le \left|z_{k'}^{\nu-1}\right| + \tau\left|\xi_{k'}^{\nu}\right|.$$

Из полученного неравенства с учетом (2.41) вытекает неравенство (2.35) и, в конечном счете, оценка (2.36), что и означает устойчивость по правой части разностной схемы (2.19), (2.20) при любом соотношении шагов τ и h.

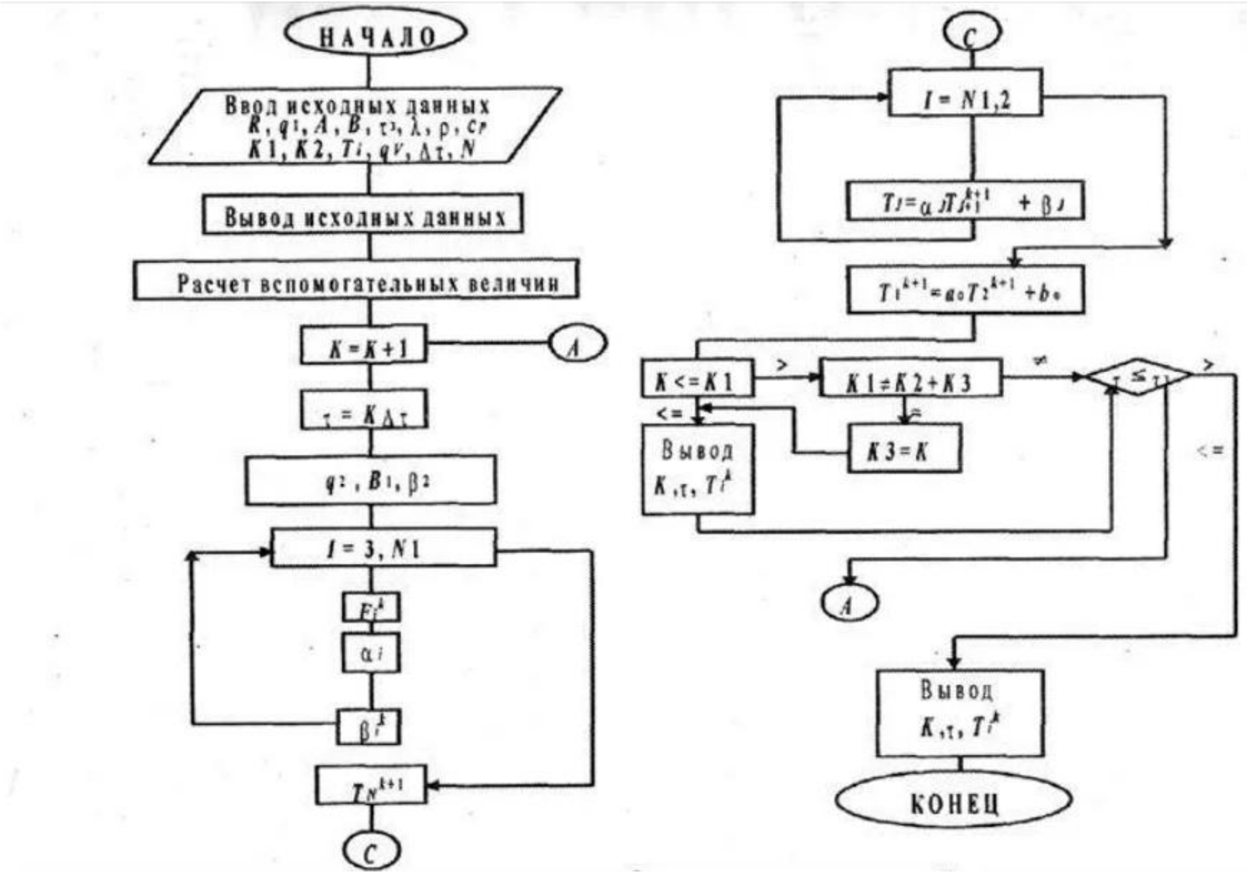
Итак, поскольку дополнительные условия (2.11), (2.12) аппроксимируются на $\overset{*}{\omega}_h$ точно, то из аппроксимации (см. (2.25), (2.26)) и установленной устойчивости по правой части в силу основной теоремы теории разностных схем вытекает сходимость решений разностных схем (2.17), (2.18) и (2.19), (2.20) к решению задачи (2.10) − (2.12) со вторым порядком по h и с первым порядком по τ, т. е.

$$\|u-y\|_h = O(h^2+\tau). \qquad (2.42)$$

При этом в случае явной схемы (2.17), (2.18) предполагается выполнение условия (2.34).

Определение. Разностная схема, устойчивая при любом соотношении шагов τ и h, называется *абсолютно устойчивой*, а устойчивая при ограничениях на τ и h – *условно устойчивой*.

Недостатком разностной схемы (2.17), (2.18) является ее условная устойчивость (ограничение (2.34) является жестким для шага τ по времени). Преимущество − простота счета по явной формуле (2.27) и возможность распространения на задачу Коши (когда условие (2.11) задано на всей оси x, а краевые условия (2.12) отсутствуют). В случае смешанной задачи (2.10) − (2.12) предпочтение отдают неявной абсолютно устойчивой разностной схеме (2.19), (2.20). Разностная краевая задача (2.28), (2.29) при переходе на каждый следующий слой решается методом прогонки весьма эффективно.



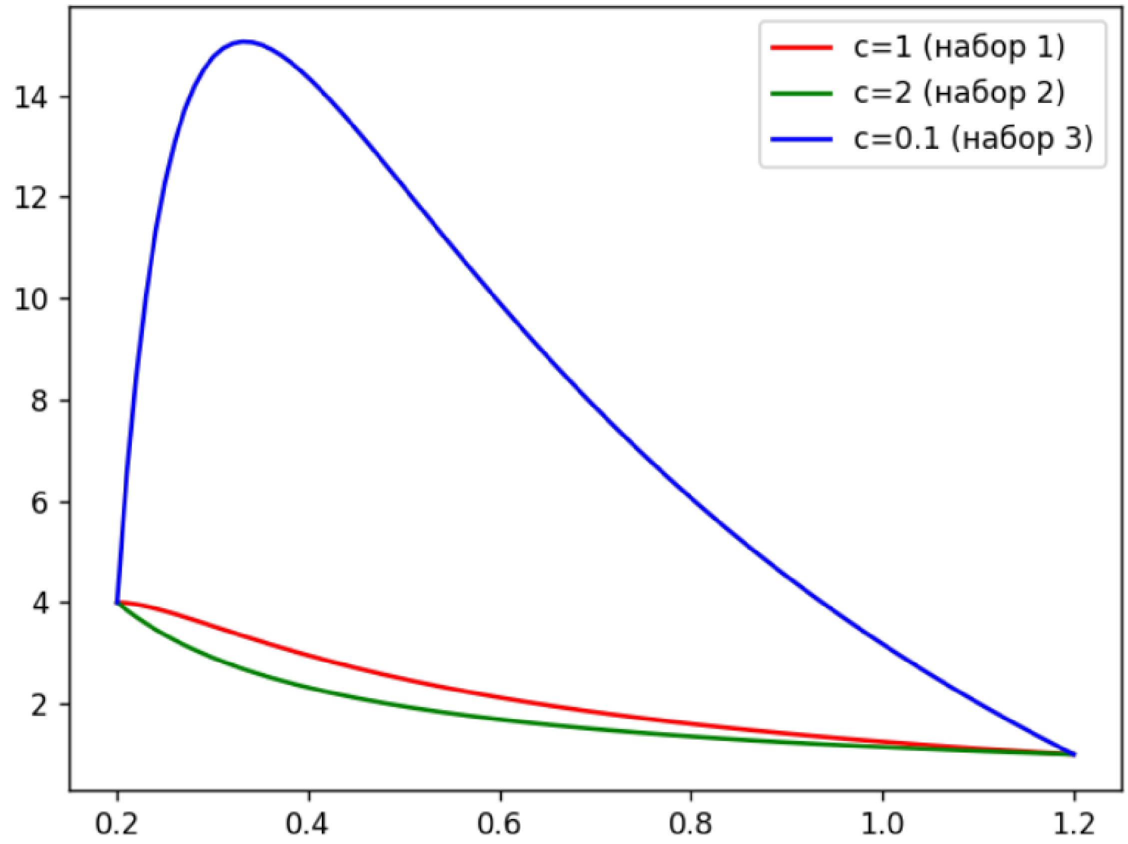
Тестовые задания

Промоделировать стационарные процессы теплопроводности стержня в зависимости от входных данных задачи:

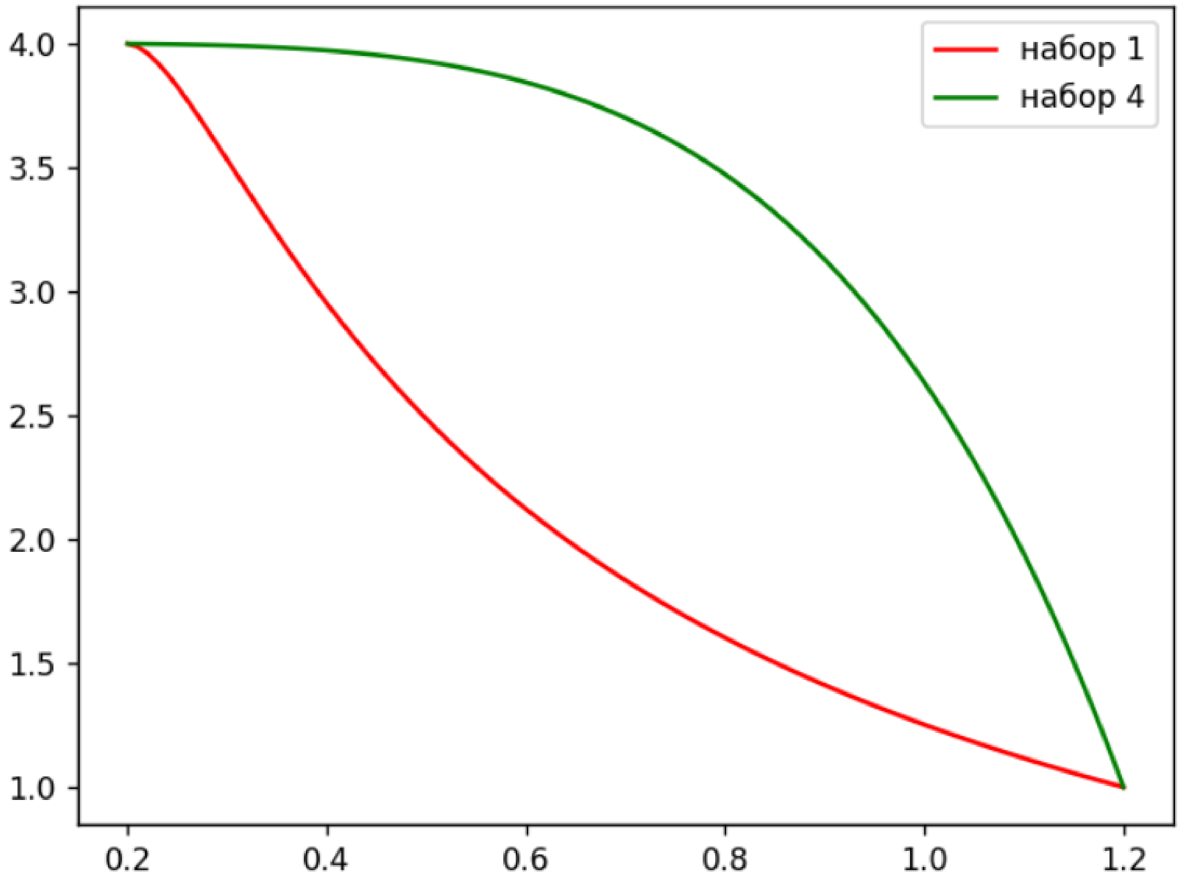
$$\{-\frac{d}{dx}\left(K(x)\frac{du}{dx}\right) = f\ u(a) = Ua, u(b) = Ub$$

$$N = 100, A = 0.2, B = 1.2, Ua = 4, Ub = 1$$

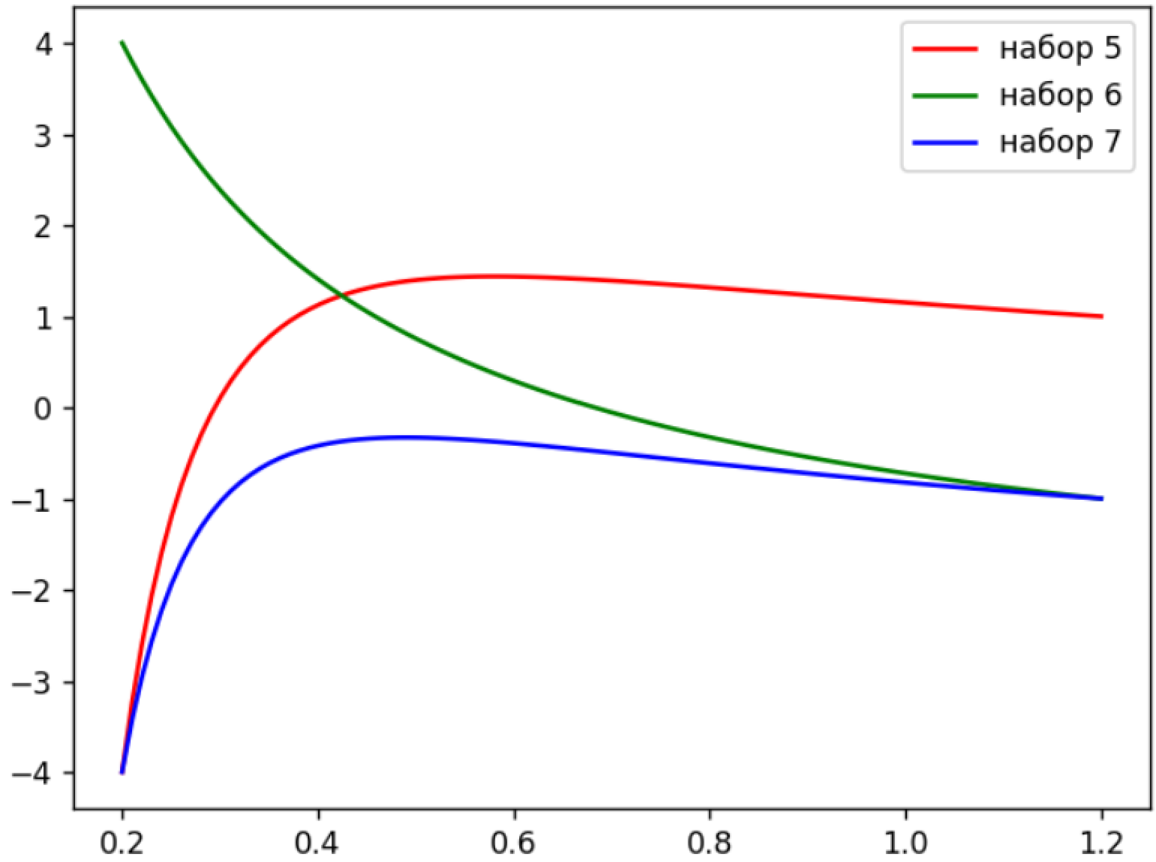
Решения задачи для наборов параметров 1-3.



Решения задачи для наборов параметров 1 и 4.



Решения задачи для наборов параметров 5-7.



Программная реализация

```
import numpy
import numpy as np
import sympy as sp
import sympy
from scipy.misc import derivative
from scipy import sparse
from scipy import integrate
import matplotlib
import matplotlib.pyplot as plt
import scipy
from sympy.solvers import solve
from numpy import linspace
from collections import namedtuple
import math
from mpl_toolkits import mplot3d


POWER = 5
POWER5 = 25
POWER_1 = 35
POWER_2 = 15
POWER_3 = 10
POWER_4 = 30
ITER_COUNT = 150
LINSPACE_SIZE = 100
```

```
x = sympy.Symbol('x')

a = 0.2
b = 1.2
u_a = 4
u_b = 1
my_k = 1



class Util:
    def sweep_method(a, b, c, d):
        AlphaS = [-c[0] / b[0]]
        BetaS = [d[0] / b[0]]
        GammaS = [b[0]]
        n = len(d)
        result = [0 for i in range(n)]

        for i in range(1, n - 1):
            GammaS.append(b[i] + a[i] * AlphaS[i - 1])
            AlphaS.append(-c[i] / GammaS[i])
            BetaS.append((d[i] - a[i] * BetaS[i - 1]) / GammaS[i])

        GammaS.append(b[n - 1] + a[n - 1] * AlphaS[n - 2])
        BetaS.append((d[n - 1] - a[n - 1] * BetaS[n - 2]) / GammaS[n - 1])

        result[n - 1] = BetaS[n - 1]
        for i in reversed(range(n - 1)):
            result[i] = AlphaS[i] * result[i + 1] + BetaS[i]

        return result

    def k_1(x, c):
        return x ** 3


    def function_1(x):
        return 1.0 + x ** (1 / 3)


    def build_final_equation(f, k, c, c1, c2):
        first_iter = (-(sympy.integrate(f(x), x) + c1) / k(x, c)).expand()
        second_iter = sympy.integrate(first_iter, x) + c2
        return second_iter


    def func_for_partition_2(yk_m1, yk, yk_p1, h, k=1, func=None):
        if not func:
            func = Util.function_1

        func = -k * (yk_p1 - 2 * yk + yk_m1) / h ** 2 - func(x)
        return func


    def solve_thermal_conductivity_equation(f, k, c, a, b, u_a, u_b):
        c1, c2 = sympy.Symbol('c1'), sympy.Symbol('c2')
        true_eq = Util.build_final_equation(f, k, c, c1, c2)
        c2_val = solve(true_eq.subs(x, b) - u_b, c2)[0]
        true_eq = true_eq.subs(c2, c2_val)

        c1_val = solve(true_eq.subs(x, a) - u_a, c1)[0]
```

```
        true_eq = true_eq.subs(c1, c1_val)
        print(true_eq)
        return true_eq
    def differences_method(start_variables_count,
                           a,
                           b,
                           y_a,
                           y_b,
                           func_for_partition,
                           points_k,
                           func=None):
        h = (b - a) / start_variables_count
        points = linspace(a + h, b - h, start_variables_count).tolist()

        a_k = []
        b_k = []
        c = []
        d = []
        if func is None:
            func = Util.function_1
        # print(func)
        selected_k = 0
        for i in range(start_variables_count):
            if points[i] > points_k[selected_k][0]:
                selected_k += 1
            a_k.append(-points_k[selected_k][1] / (h ** 2))
            b_k.append(2 * points_k[selected_k][1] / h ** 2)
            c.append(-points_k[selected_k][1] / h ** 2)
            d.append(func(points[i]))
        d[0] = d[0] - a_k[0] * y_a
        d[-1] = d[-1] - c[-1] * y_b
        answer = Util.sweep_method(a_k, b_k, c, d)
        data_type = namedtuple('data',
                               ('points', 'answer', 'step'))
        points.insert(0, a)
        points.append(b)

        answer.insert(0, y_a)
        answer.append(y_b)
        return data_type(points, answer, h)
    def func_for_partition_2(yk_m1, yk, yk_p1, h, k=1, func=None):
        if not func:
            func = Util.function_1

        func = -k * (yk_p1 - 2 * yk + yk_m1) / h ** 2 - func(x)
        return func


# a = 0
# b = 1
# h = (b - a) / 150
# ua = 0
# ub = 0

def task1():

    print('Набор 1:')
    var_1 = Util.solve_thermal_conductivity_equation(Util.function_1,
Util.k_1, 1, a, b, u_a, u_b)
    print('\nНабор 2:')
    var_2 = Util.solve_thermal_conductivity_equation(Util.function_1,
```

```
                                                        lambda x, c: c *
Util.k_1(x, c),
                                                        2, a, b, u_a, u_b)
    print('\nНабор 3:')
    var_3 = Util.solve_thermal_conductivity_equation(Util.function_1,
                                                        lambda x, c: c *
Util.k_1(x, c),
                                                        0.1, a, b, u_a, u_b)

    D = linspace(a, b, LINSPACE_SIZE)
    func_y1, func_y2, func_y3, func_y4 = [], [], [], []
    for i in range(len(D)):
        func_y1.append(var_1.subs(x, D[i]))
        func_y2.append(var_2.subs(x, D[i]))
        func_y3.append(var_3.subs(x, D[i]))

    fig, _ = plt.subplots()
    plt.plot(D, func_y1, c='red', label='c=1 (набор 1)')
    plt.plot(D, func_y2, c='green', label='c=2 (набор 2)')
    plt.plot(D, func_y3, c='blue', label='c=0.1 (набор 3)')

    plt.legend()
    plt.show()

    print('\nНабор 4:')
    var_4 = Util.solve_thermal_conductivity_equation(Util.function_1,
                                                        lambda x, c: 1 /
Util.k_1(x, c),
                                                        1, a, b, u_a, u_b)

    for i in range(len(D)):
        func_y4.append(var_4.subs(x, D[i]))

    fig, _ = plt.subplots()
    plt.plot(D, func_y1, c='red', label='набор 1')
    plt.plot(D, func_y4, c='green', label='набор 4')

    plt.legend()
    plt.show()

    print('\nНабор 5:')
    var_5 = Util.solve_thermal_conductivity_equation(Util.function_1,
Util.k_1, 1, a, b, -u_a,
                                                        u_b)
    print('\nНабор 6:')
    var_6 = Util.solve_thermal_conductivity_equation(Util.function_1,
Util.k_1, 1, a, b, u_a,
                                                        -u_b)
    print('\nНабор 7:')
    var_7 = Util.solve_thermal_conductivity_equation(Util.function_1,
Util.k_1, 1, a, b, -u_a,
                                                        -u_b)

    func_y5, func_y6, func_y7 = [], [], []
    for i in range(len(D)):
        func_y5.append(var_5.subs(x, D[i]))
        func_y6.append(var_6.subs(x, D[i]))
        func_y7.append(var_7.subs(x, D[i]))

    fig, _ = plt.subplots()
    plt.plot(D, func_y5, c='red', label='набор 5')
```

```
    plt.plot(D, func_y6, c='green', label='набор 6')
    plt.plot(D, func_y7, c='blue', label='набор 7')

    plt.legend()
    plt.show()

def A(x, k1, k2, k3, xr1, xr2):
    if x < xr1:
        return k1
    elif x-h < xr1 < x:
        return h / ((xr1 - x + h) / k1 + (x - xr1) / k2)
    elif x <= xr2:
        return k2
    elif x-h < xr2 < x:
        return h / ((xr2 - x + h) / k2 + (x - xr2) / k3)
    else:
        return k3
def B(x, k1, k2, k3, xr1, xr2):
    if x < xr1:
        return k1
    elif x < xr1 < x+h:
        return h / ((xr1 - x) / k1 + (x+h - xr1) / k2)
    elif x+h <= xr2:
        return k2
    elif x < xr2 < x+h:
        return h / ((xr2 - x) / k2 + (x+h - xr2) / k3)
    else:
        return k3
def phi(x, x0, C):
    if abs(x - x0) - h/2 < 1e-5:
        return C/2
    elif x - h/2 < x0 < x + h/2:
        return C
    else:
        return 0
def solve2(a, b, ua, ub, h, k1, k2, k3, xr1, xr2, phi_conds):
    n = int((b - a) / h) + 1
    M = np.zeros(shape=(n, n))
    Y = np.zeros(n)
    M[0, 0] = 1
    M[-1, -1] = 1
    Y[0] = ua
    Y[-1] = ub
    for i in range(1, n - 1):
        xi = a + h * i
        M[i, i - 1] = A(xi, k1, k2, k3, xr1, xr2)
        M[i, i] = -(A(xi, k1, k2, k3, xr1, xr2) + B(xi, k1, k2, k3, xr1,
xr2))
        M[i, i + 1] = B(xi, k1, k2, k3, xr1, xr2)
        Y[i] = - h * sum(phi(xi, x0, C) for x0, C in phi_conds)

    return np.linspace(a, b, n), np.linalg.solve(M, Y)


def task2():
    POWER = 200
    POWER_1 = 100
    POWER_2 = 150
    POWER_3 = 100
    POWER_4 = 300
```

```
    a = 0.2
    b = 1.2
    u_a = 4
    u_b = 1
    my_k = 1
    eps = 0.007

    # 4a
    data_a1 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                  [(0.5 * (b + a), 0.01), (b, 150)])
    print('h=', data_a1.step)
    data_a2 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                  [(0.5 * (b + a), 150), (b, 0.01)])

    D1, y1 = data_a1.points, data_a1.answer
    D2, y2 = data_a2.points, data_a2.answer

    plt.figure()
    plt.plot(D1, y1, color='red', label='k1 << k2')
    plt.plot(D2, y2, label='k1 >> k2')
    plt.title('Задание 2 пункт 4а')
    plt.grid(True)
    plt.legend()
    plt.show()

    # 4б
    data_b1 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                  [(a + (1 / 3) * (b - a), 0.2),
                                   (a + (2 / 3) * (b - a), 0.6),
                                   (b, 0.9)])
    data_b2 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                  [(a + (1 / 3) * (b - a), 0.9),
                                   (a + (2 / 3) * (b - a), 0.6),
                                   (b, 0.2)])
    data_b3 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                  [(a + (1 / 3) * (b - a), my_k),
                                   (a + (2 / 3) * (b - a), 2 * my_k),
                                   (b, my_k)])
    data_b4 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                  [(a + (1 / 3) * (b - a), 20 * my_k),
                                   (a + (2 / 3) * (b - a), my_k),
                                   (b, 20 * my_k)])

    D1, y1 = data_b1.points, data_b1.answer
    D2, y2 = data_b2.points, data_b2.answer
    D3, y3 = data_b3.points, data_b3.answer
    D4, y4 = data_b4.points, data_b4.answer
    plt.figure()
    plt.plot(D1, y1, color='red', label='k1<k2<k3')
    plt.plot(D2, y2, color='green', label='k1>k2>k3')
    plt.plot(D3, y3, color='blue', label='k1=k, k2=2k, k3=k')
    plt.plot(D4, y4, color='yellow', label='k1=20k, k2=k, k3=20k')
    plt.title('Задание 2 пункт 4б')
    plt.grid(True)
    plt.legend()
```

```
    plt.show()

    # 5
    def delta_1(x):
        if x > (a + (b - a) * 0.5):
            return POWER
        return 0

    def delta_2(x):
        p = 0
        if x > (a + (b - a) / 3):
            p = POWER5
        if x > (a + 2 * (b - a) / 3):
            p += POWER5
        if p != 0:
            return p
        return 0

    def delta_3(x):

        p = 0
        if x > (a + (b - a) / 3):
            p += POWER_1
        if x > (a + 2 * (b - a) / 3):
            p += POWER_2
        if p != 0:
            return p
        return 0

    def delta_4(x):
        p = 0
        if x > (a + (b - a) * 0.2):
            p = POWER_3
        if x > (a + (b - a) * 0.8):
            p += POWER_4
        if p != 0:
            return p
        return 0

    data_c1 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                 [(b, my_k)], delta_1)

    data_c2 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                 [(b, my_k)], delta_2)
    data_c3 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                 [(b, my_k)], delta_3)
    data_c4 = Util.differences_method(ITER_COUNT, a, b, u_a, u_b,
Util.func_for_partition_2,
                                 [(b, my_k)], delta_4)

    D1, y1 = data_c1.points, data_c1.answer
    D2, y2 = data_c2.points, data_c2.answer
    D3, y3 = data_c3.points, data_c3.answer
    D4, y4 = data_c4.points, data_c4.answer
    plt.figure()
    plt.plot(D1, y1, color='red', label='источник в середине')
    plt.plot(D2, y2, color='green', label='одинаковые источники
симметрично')
```

```
    plt.plot(D3, y3, color='yellow', label='разные источники симметрично')
    plt.plot(D4, y4, label='Разные источники на 0.2 и 0.8')
    plt.title('Задание 2 пункт 5')
    plt.legend()
    plt.grid(True)
    plt.show()

x, t = sympy.symbols('x, t')

def task3(x_h, t_h, a, b, k, T, g1, g2, phi, f):
    x_h_step_amount = int((b - a) / x_h) + 1
    t_h_step_amount = int(T / t_h) + 1
    x_hs = np.linspace(a, b, x_h_step_amount)
    t_hs = np.linspace(0, T, t_h_step_amount)

    matrix = np.zeros(shape=(t_h_step_amount, x_h_step_amount))
    matrix[0, 1:-1] = np.array([phi(x_hs[i]) for i in range(1,
x_h_step_amount-1)])
    matrix[:, 0] = np.array([g1(x_hs[0], t_hs[i]) for i in
range(t_h_step_amount)])
    matrix[:, -1] = np.array([g2(x_hs[-1], t_hs[i]) for i in
range(t_h_step_amount)])
    for i in range(1, t_h_step_amount):
        for j in range(1, x_h_step_amount-1):
            matrix[i,j] = sum([
                k(x_hs[j] - x_h/2) * t_h / x_h**2 * matrix[i-1, j-1],
                (1 - (k(x_hs[j] - x_h / 2) + k(x_hs[j] + x_h / 2)) * t_h /
x_h**2) * matrix[i-1, j],
                k(x_hs[j] + x_h/2) * t_h / x_h**2 * matrix[i-1, j+1],
                t_h * f(x_hs[j], t_hs[i]) * (1 - math.exp(-t_hs[i]))
            ])
    ax = plt.axes(projection='3d')
    ax.set_ylabel('$T$ time axis')
    ax.set_xlabel('$X$ spatial axis')
    ax.set_zlabel('$Y$ function value axis')


    for i in range(0, t_h_step_amount, 100):
        ax.plot3D(x_hs, np.array([t_hs[i]]*x_h_step_amount), matrix[i,:])
    plt.title('Задание 3')
    plt.show()



def task4(a, b, k, T, phi, g1, g2, f):
    x_h = (b - a) / 50
    x_h_step_amount = int((b - a) / x_h) + 1
    t_h = 0.5 * x_h**2 / k
    t_h_step_amount = int(T / t_h) + 1
    x_hs = np.linspace(a, b, x_h_step_amount)
    t_hs = np.linspace(0, T, t_h_step_amount)

    matrix = np.zeros(shape=(t_h_step_amount, x_h_step_amount))

    # initial condition
    matrix[0, 1:-1] = np.array([phi(x_hs[i], t_hs[0]) for i in range(1,
x_h_step_amount-1)])

    # bounds condition
    matrix[:, 0] = np.array([g1(x_hs[0], t_hs[i]) for i in
range(t_h_step_amount)])
```

```
    matrix[:, -1] = np.array([g2(x_hs[-1], t_hs[i]) for i in
range(t_h_step_amount)])

    coef = np.array([k * t_h / x_h**2, 1 - 2 * k * t_h / x_h**2, k * t_h /
x_h**2])

    for i in range(1, t_h_step_amount):
        for j in range(1, x_h_step_amount-1):
            matrix[i][j] = matrix[i-1, j-1:j+2].dot(coef) + t_h *
f(x_hs[j], t_hs[i-1])

    # plotting
    ax = plt.axes(projection='3d')
    ax.set_ylabel('$T$ time axis')
    ax.set_xlabel('$X$ spatial axis')
    ax.set_zlabel('$Y$ function value axis')


    for i in range(0, t_h_step_amount, 10):
        ax.plot3D(x_hs, np.array([t_hs[i]]*x_h_step_amount), matrix[i,:])
    plt.title('Задание 4')
    plt.show()

if __name__=='__main__':
    task1()
    task2()


    a, b = 1.5, 2.5
    g1 = sp.lambdify((x, t), 3)
    g2 = sp.lambdify((x, t), 3)
    f = sp.lambdify((x, t), x + x**0.5)
    k = sp.lambdify(x, x**(-1/3))
    phi = sp.lambdify(x, 12*(x-2)**2)
    h_x, h_t = 0.05, 0.001
    T = 500 * h_t
    task3(h_x, h_t, a, b, k, T, g1, g2, phi, f)

    k = 0.5
    T = 0.4
    a, b = -1, 1
    phi = sp.lambdify((x, t), 1-x*x)
    g1 = sp.lambdify((x, t), 0)
    g2 = sp.lambdify((x, t), 0)
    f = sp.lambdify((x, t), x)
    task4(a, b, k, T, phi, g1, g2 , f)
```

Полученные результаты

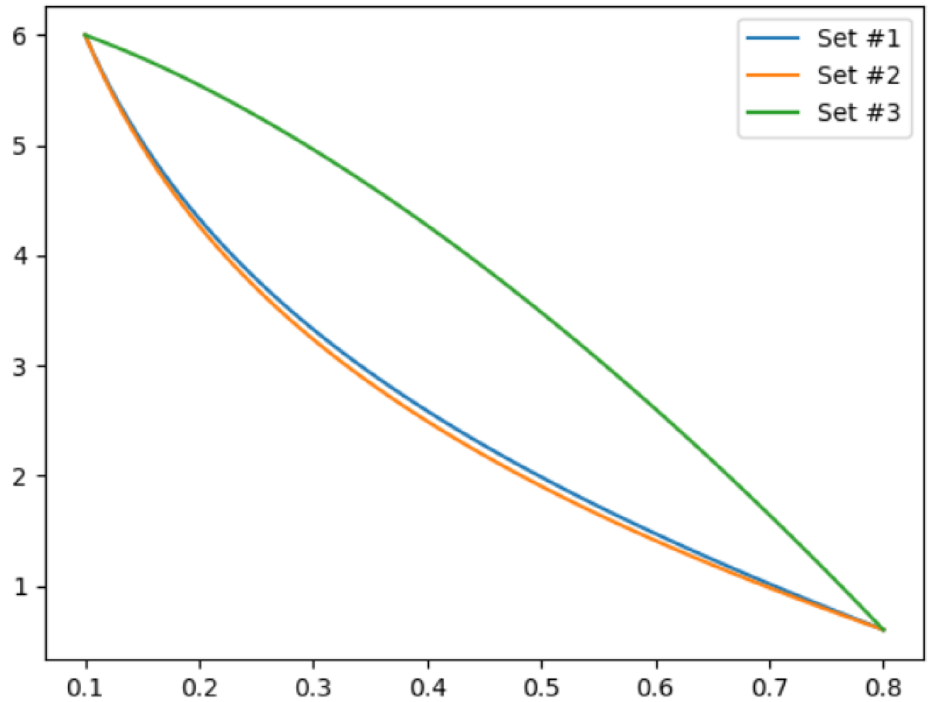
Задание 1.

Промоделировать стационарные процессы теплопроводности стержня в зависимости от входных данных задачи:

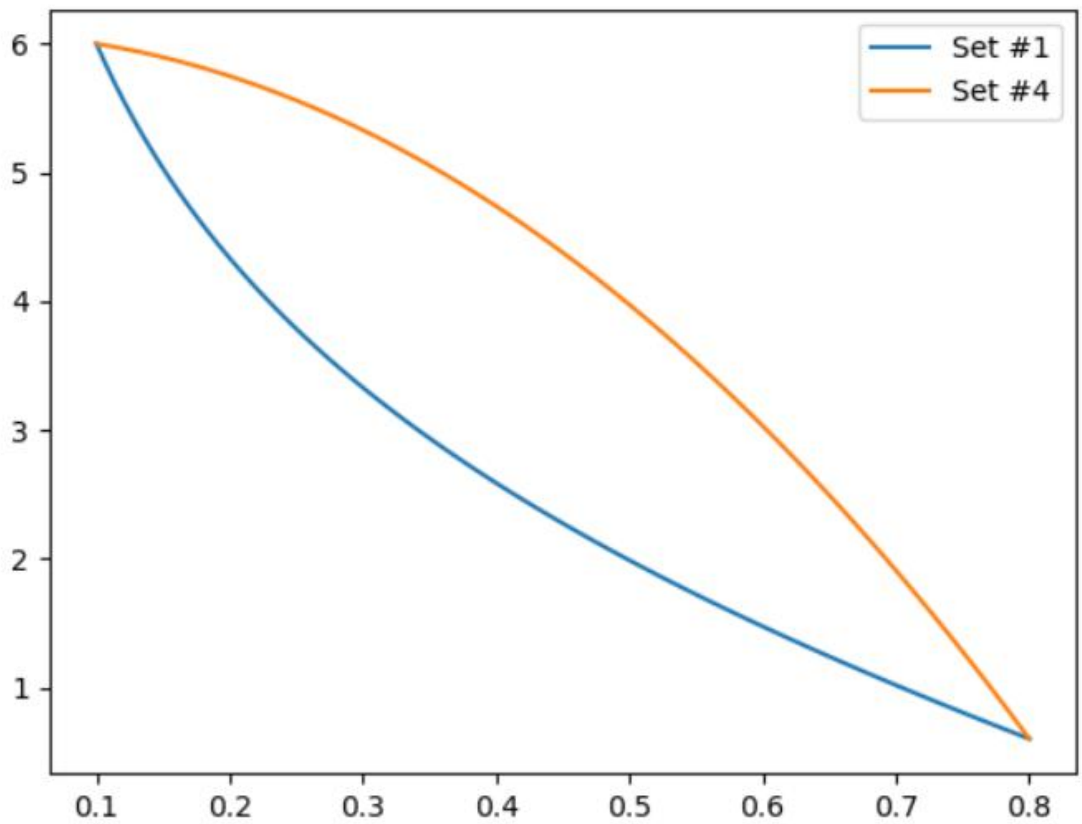
$$\{\frac{-d}{dx}\left(K(x)\frac{du}{dx}\right) = f\ u(a) = Ua, u(b) = Ub$$

$$N = 100, A = 0.1, B = 0.8, Ua = 6, Ub = 0.6$$

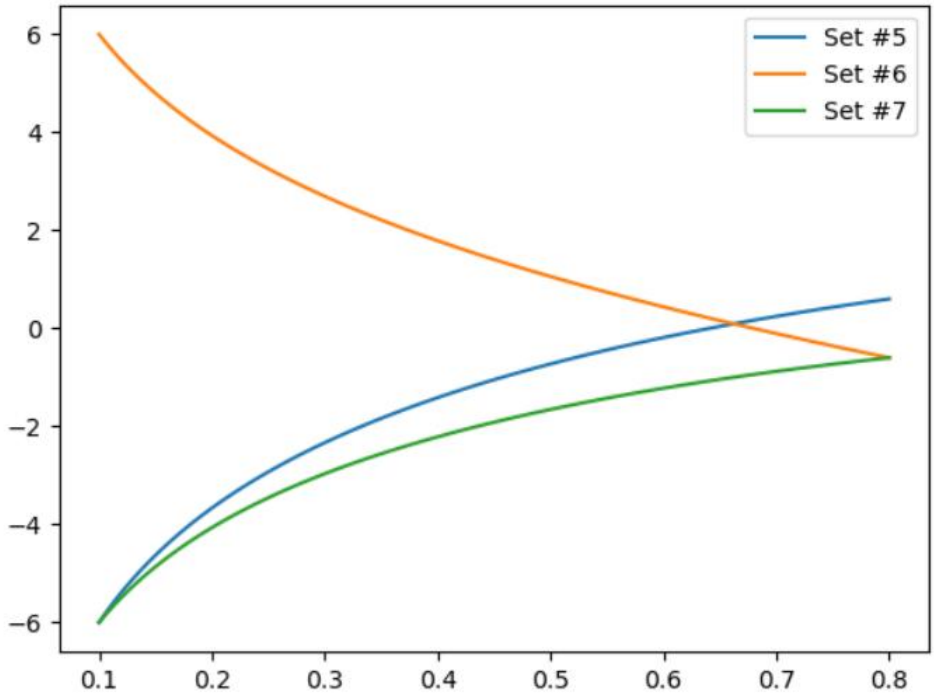
Решения задачи для наборов параметров 1-3.



Решения задачи для наборов параметров 1 и 4.



Решения задачи для наборов параметров 5-7.



Задание 2.

Промоделировать стационарные процессы теплопроводности стержня в зависимости от входных данных задачи – переменного коэффициента теплопроводности $k(x)$ и плотности источников тепла $f(x)$:

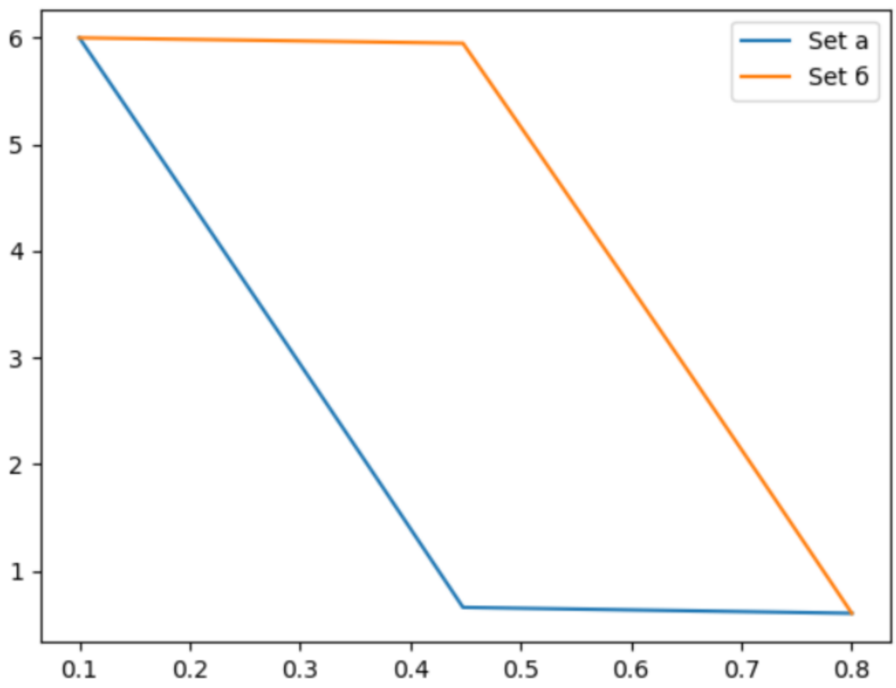
$$\{\frac{-d}{dx}\left(k(x)\frac{du}{dx}\right) = f,\ u(a) = Ua, u(b) = Ub$$

$$N = 150, A = 0.1, B = 0.8, Ua = 6, Ub = 0.6$$

Решение задачи, положив, что стержень состоит из двух материалов с различными коэффициентами теплопроводности $k(x)$:

$$k(x) = \{k1, a \le x \le a + \frac{b-a}{3}\ k2, 0.5(b + a) < x \le b$$

при: а) $k1(1) \ll k2(100)$, б) $k1(100) \gg k2(1)$:



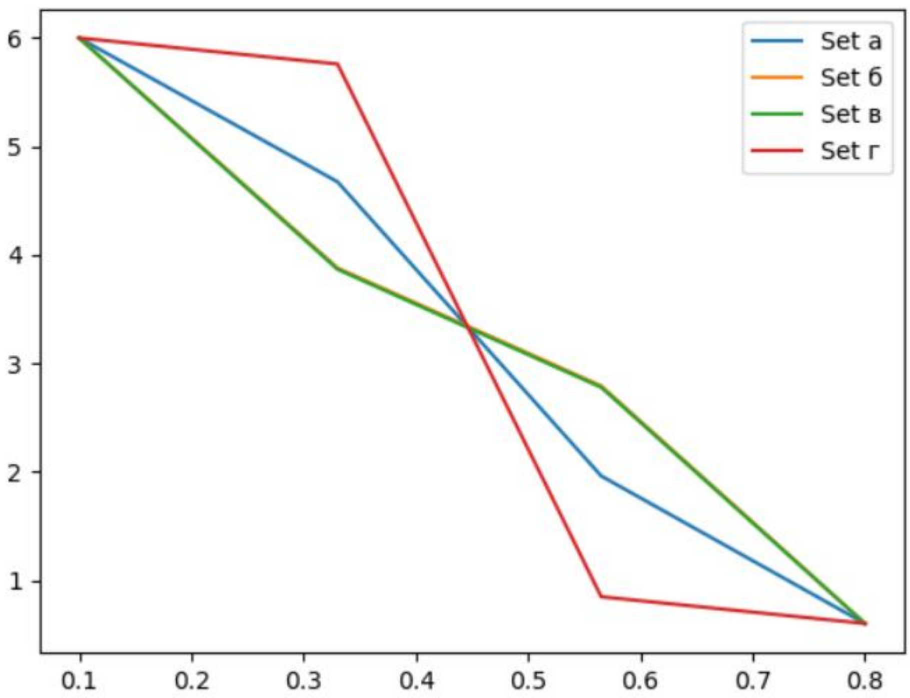
Решение задачи, положив, что стержень состоит из трёх материалов с различными свойствами:

$k(x) = \{k1, a \le x \le a + \frac{b-a}{3}\ k2, a + \frac{b-a}{3} \le x \le a + \frac{2(b-a)}{3}\ k3, a + \frac{2(b-a)}{3} < x \le b$

при а) $k1(5) < k2(10) < k3(20)$; б$k1(20) > k2(10) > k3(5)$;

в) $k1(100) = k, k2(200) = 2k, k3(100) = k$;

г) $k1(100) = 20k, k2(5) = k, k3(100) = 20k$.



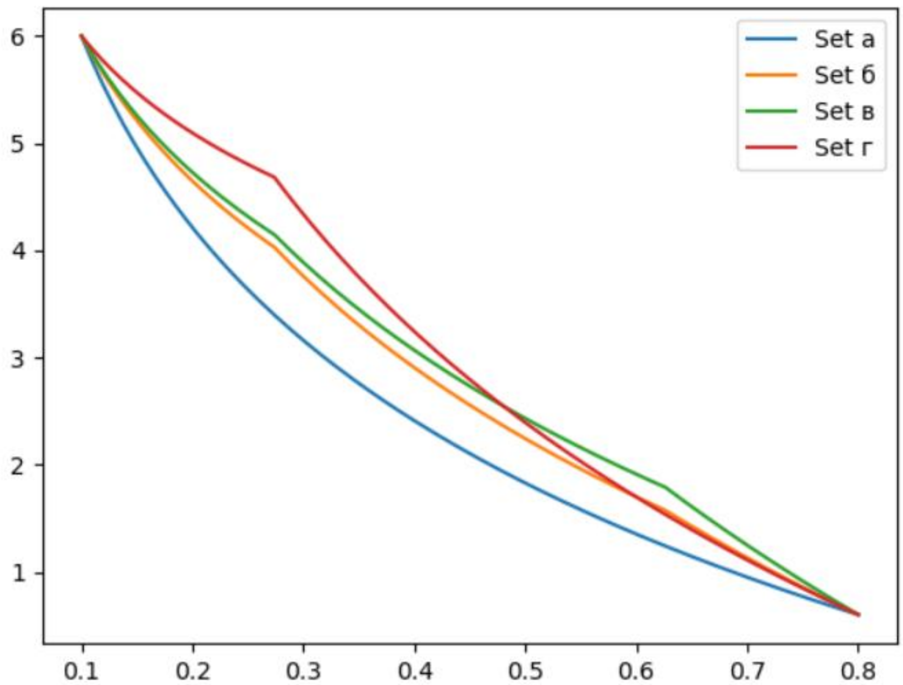
Решение задачи в зависимости от правой части – функции $f(x) = c\delta(x - x0)$ – точечного источника тепла. Взяты следующие варианты расположения источника:

а) точечный источник поставлен в середину отрезка $[a, b]$;

б) два одинаковых источника по мощности поставлены в разные точки отрезка, симметричные относительно середины отрезка;

в) два различных по мощности источника поставлены симметрично;

г) два одинаковых по мощности источника поставлены следующим образом – первый поставлен в середину отрезка $[a, b]$,а второй в середину отрезка между точкой a и первым источником.



Задание 3.

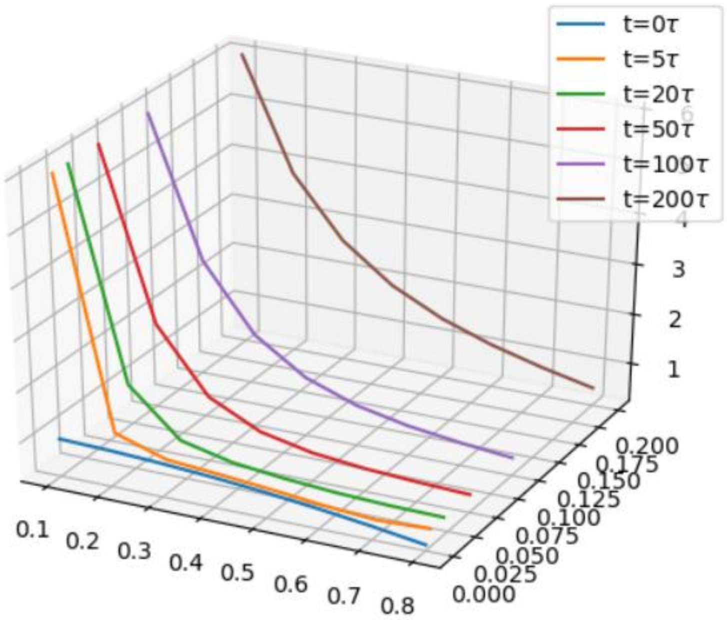
Промоделировать нестационарные процессы теплопроводности в зависимости от входных данных задачи – коэффициента теплопроводности и начальной температуры:

$$\{\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right) + f(x)\left(1 - e^{-t}\right), 0 < x < 1, 0 < t < T,\ u(0,t) = Ua, u(1,t) = Ub,$$

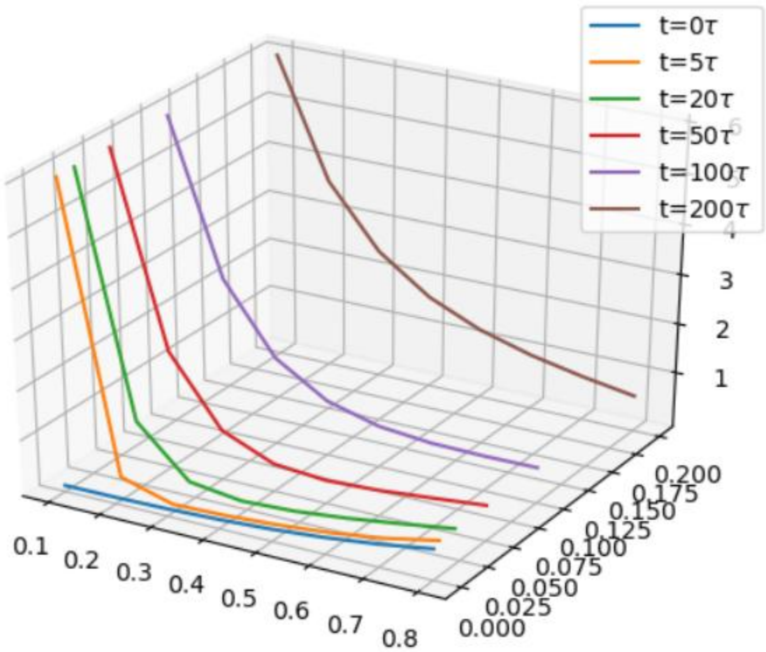
$$\tau \leq 0.5\left(\frac{h^2}{k}\right)$$

$$a = 0.1, b = 0.8, Ua = 6, Ub = 0.6, k(x) = x, f(x) = x + x^{\frac{1}{3}}.$$

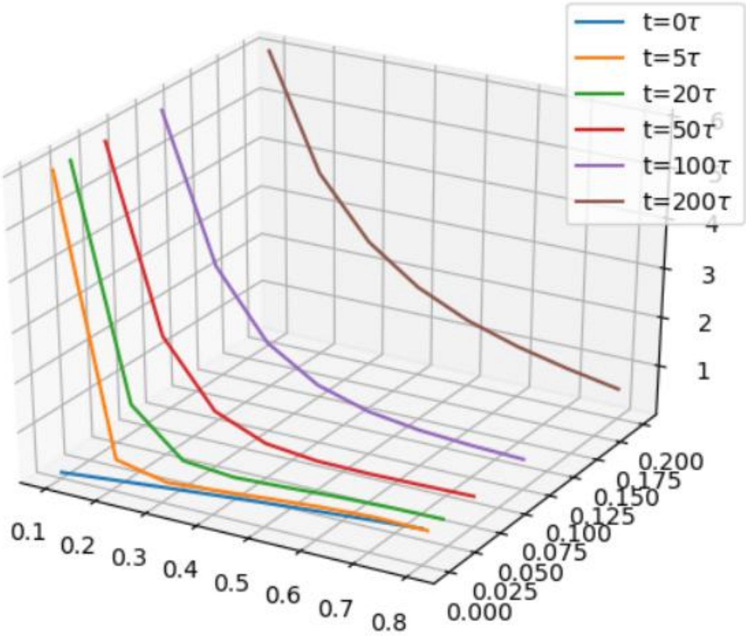
Решение задачи при $\phi(x) = 1 - x^2$:



Решение задачи при $\phi(x) = x^3$:



Решение задачи при $\phi(x) = sin(x)$:



Задание 4.

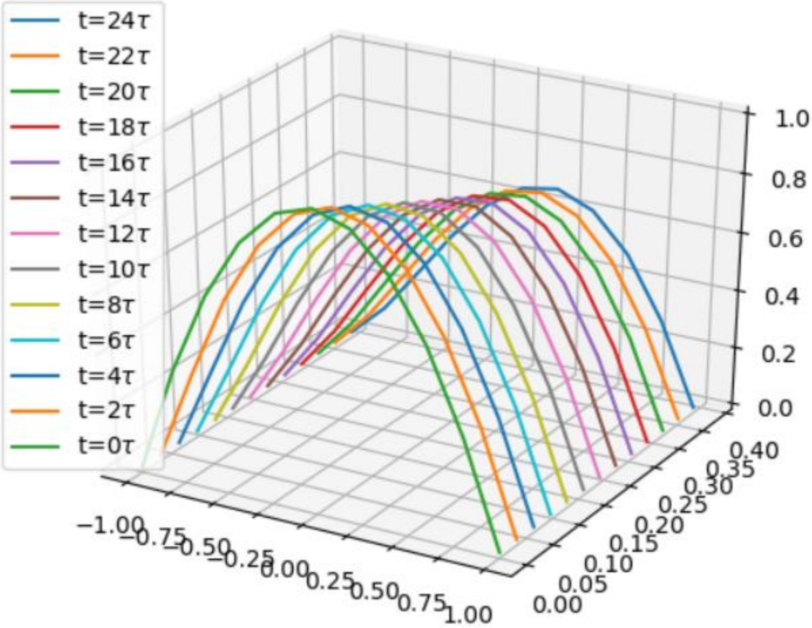
Промоделировать нестационарные процессы теплопроводности в зависимости от входных данных задачи. Найти приближенное решение начально-краевой задачи для уравнения теплопроводности:

$$\{\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} + f(x,t), a < x < b, 0 < t \le T,\ u(a,t) = g_1(t), u(b,t) = g_2(t), 0 < t \le$$

$$N = 10, a = -1, b = 1, k = 0.5, T = 0.4, \varphi(x) = 1 - x^2, g_1(t) = 0,$$

$$g_2(t) = 0, f(x,t) = x.$$

$$\tau \le 0.5\left(\frac{h^2}{k}\right)$$



Выводы

Мы изучили метод разностных аппроксимаций для уравнения теплопроводности, составили алгоритмы решения уравнения теплопроводности методом сеток, применимыми для организации вычислений на ПЭВМ, составили программы решения уравнения теплопроводности по разработанным алгоритмам, также выполнили тестовые примеры и проверили правильность работы программ и получили численное решение заданного уравнения теплопроводности. А также были промоделированы стационарные и нестационарные процессы теплопроводности стержня в зависимости от исходных данных.