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

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

Лабораторная работа № 4
Дисциплина: «Моделирование»

Тема: «Программно-алгоритмическая реализация моделей на основе дифференциальных уравнений в частных производных с краевыми условиями II и III рода»

Студент	Овчинникова А. П.
Группа	ИУ7-65Б
Оценка (баллы)	
Преподаватель	Градов В.М.

Москва, 2020 г.

Цель работы

Целью данной работы является 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 $T(x)$:

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

Краевые условия

$$\begin{cases} t = 0, T(x,0) = T_0 \\ x = 0, -k(T(0))\frac{\delta T}{\delta x} = F_0 \\ x = l, -k(T(l))\frac{\delta T}{\delta x} = \alpha_N(T(l) - T_0) \end{cases} \tag{2}$$

Примем

$$p(x) = \frac{2}{R}\alpha(x) \tag{3}$$

$$f(u) = f(x) = \frac{2T_0}{R}\alpha(x) \tag{4}$$

$$F = -k(T)\frac{\delta T}{\delta x} \tag{5}$$

2. Разностная схема:

$$\widehat{A}_n\widehat{y}_{n-1} - \widehat{B}_n\widehat{y}_n + \widehat{D}_n\widehat{y}_{n+1} = -\widehat{F}_n, 1 \le n \le N-1 \tag{6}$$

где

$$\widehat{A}_n = \widehat{\chi}_{n-1/2}\,\frac{\tau}{h}, \tag{7}$$

$$\widehat{D}_n=\widehat{\chi}_{n+1/2}\,\frac{\tau}{h}, \tag{8}$$

$$\widehat{B}_n=\widehat{A}_n+\widehat{D}_n+\widehat{c}_n\,h+\widehat{p}_n\,h\tau, \tag{9}$$

$$\widehat{F}_n=f_nh\tau+\widehat{c}_n\,y_nh. \tag{10}$$

Разностный аналог краевого условия при $x=0$ (получена в Лекции №14 (14.6),(14.7)):

$$\begin{aligned}\left(\frac{h}{8}\,\widehat{c}_{1/2}+\frac{h}{4}\,\widehat{c}_0+\widehat{\chi}_{1/2}\,\frac{\tau}{h}+\frac{\tau h}{8}p_{1/2}+\frac{\tau h}{4}p_0\right)\widehat{y}_0+\\ +\left(\frac{h}{8}\,\widehat{c}_{1/2}-\widehat{\chi}_{1/2}\,\frac{\tau}{h}+\frac{\tau h}{8}p_{1/2}\right)\widehat{y}_1=\\ =\frac{h}{8}\,\widehat{c}_{1/2}\,(y_0+y_1)+\frac{h}{4}\,\widehat{c}_0\,y_0+\widehat{F}\,\tau+\frac{\tau h}{4}(\widehat{f}_{1/2}+\widehat{f}_0)\end{aligned} \tag{11}$$

Самостоятельно надо получить интегро-интерполяционным методом разностный аналог краевого условия при $x=l$. При этом учесть, что поток

$$\widehat{F}_n=\alpha_N(\widehat{y}_N-T_0), \tag{12}$$

$$\widehat{F}_{N-1/2}=\widehat{\chi}_{N-1/2}\,\frac{\widehat{y}_{N-1}-\widehat{y}_N}{h}. \tag{13}$$

3. Значения параметров для отладки (все размерности согласованы).

Таблица 1: Значения параматров.

$k(T) =$	$a_1(b_1 + c_1 T^{m_1})$ Вт/см К
$c(T) =$	$a_2 + b_2 T^{m_2} - \frac{c_2}{T^2}$ Дж/см^3 К
$a_1 =$	0.0134
$b_1 =$	1
$c_1 =$	$4.35 \cdot 10^{-4}$
$m_1 =$	1
$a_2 =$	2.049
$b_2 =$	$0.563 \cdot 10^{-3}$
$c_2 =$	$0.528 \cdot 10^5$
$m_2 =$	1
$\alpha(x) =$	$\frac{c}{x-d}$
$\alpha_0 =$	0.05 Вт/см^2 К
$\alpha_N =$	0.01 Вт/см^2 К
$l =$	10 см
$T_0 =$	300 К
$R =$	0.5 см
$F(t) =$	50 Вт/см^2

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 меняющееся во времени.

2. Свойства материала стержня привязаны к температуре, т.е. теплоемкость 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 зависящим от времени, а в работе №3 поток был постоянный.

Если в настоящей работе задать поток постоянным, т.е. $F(t) = const$, 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.

Задание

1. Представить разностный аналог краевого условия при и его краткий вывод интегро -интерполяционным методом.

2. График зависимости температуры $T(x, t_m)$ от координаты x при нескольких фиксированных значениях времени t_m (аналогично рисунку в лекции №14) при заданных выше параметрах. Обязательно представить распределение $T(x, t)$ в момент времени, соответствующий установившемуся режиму, когда поле перестает меняться с некоторой точностью, т.е. имеет место выход на стационарный режим. На этой стадии левая часть дифференциального уравнения близка к нулю, и на самом деле решается уравнение из лабораторной работы №3 (отличие только в том, что там было линейное уравнение).

3. График зависимости $T(x_n, t)$ при нескольких фиксированных значениях координаты x_n. Обязательно представить случай $n = 0$, т.е. $x = x_0 = 0$.

Предварительные вычисления

1. Найдем c, d**.**

$c = -\alpha_0 d$

$d = \frac{\alpha_N l}{\alpha_N - \alpha_0}$.

2. Получим интегро-интерполяционным методом разностный аналог краевого условия при $x = l$.

Запишем уравнение (1) с учетом (3), (4) и (5)

$$c(T)\frac{\delta T}{\delta x} = -\frac{\delta F}{\delta x} - p(x)T + f(T) \tag{14}$$

Проводим интегрирование уравнения (14) на отрезке $[x_{N-1/2}; x_N]$ и на временном интервале $[t_m; t_{m+1}]$.

$$\int\limits_{x_{N-1/2}}^{x_N} dx \int\limits_{t_m}^{t_{m+1}} c(T)\frac{\delta T}{\delta t}dt =$$

$$= -\int\limits_{t_m}^{t_{m+1}} dt \int\limits_{x_{N-1/2}}^{x_N} \frac{\delta F}{\delta x}dx - \int\limits_{x_{N-1/2}}^{x_N} dx \int\limits_{t_m}^{t_{m+1}} p(x)Tdt + \int\limits_{x_{N-1/2}}^{x_N} dx \int\limits_{t_m}^{t_{m+1}} f(T)dt \tag{15}$$

или

$$\int\limits_{x_{N-1/2}}^{x_N} \widehat{c}\ (\widehat{T} - T)dx = \int\limits_{t_m}^{t_{m+1}} (F_N - F_{N-1/2})dt - \int\limits_{x_{N-1/2}}^{x_N} p\ \widehat{T}\ \tau dx +$$

$$+ \int\limits_{x_{N-1/2}}^{x_N} \widehat{f}\ \tau dx \tag{16}$$

Здесь при вычислении внутренних интегралов по t справа в уравнении (15) применен метод правых прямоугольников, тем самым следует ожидать порядок точности $O(\tau)$ по переменной t.

Вычисляем интегралы. Первый интеграл справа,как и ранее, находим методом правых прямоугольников, а остальные – методом трапеций.

$$\frac{h}{4}\left(\widehat{c}_N\ (\widehat{y}_N - y_N) + \widehat{c}_{N-1/2}\ (\widehat{y}_{N-1/2} - y_{N-1/2})\right) = -(\widehat{F}_N - \widehat{F}_{N-1/2})\tau -$$

$$-(p_N\ \widehat{y}_N + p_{N-1/2}\ \widehat{y}_{N-1/2})\tau\frac{h}{4} + (\widehat{f}_N + \widehat{f}_{N-1/2})\tau\frac{h}{4}. \tag{17}$$

Подставляя в данное уравнение (12) и (13), и заменяя $\widehat{y}_{N-1/2}=\frac{\widehat{y}_{N-1}+\widehat{y}_N}{2}$, $y_{N-1/2}=\frac{y_{N-1}+y_N}{2}$, найдем разностный аналог краевого условия.

$$\widehat{y}_{N-1}\left(\frac{h}{8}\,\widehat{c}_{N-1/2}-\frac{\tau\,\widehat{\chi}_{N-1/2}}{h}+\frac{h}{8}p_{N-1/2}\right)+$$
$$+\,\widehat{y}_N\left(\frac{h}{4}\,\widehat{c}_N+\frac{h}{8}\,\widehat{c}_{N-1/2}+\tau\alpha_N+\frac{\tau\,\widehat{\chi}_{N-1/2}}{h}+\frac{h}{4}\tau p_N+\frac{h}{8}\tau p_{N-1/2}\right)=$$
$$=\frac{h}{4}\,\widehat{c}_N\,y_N+\frac{h}{8}\,\widehat{c}_{N-1/2}\,y_N+$$
$$+\frac{h}{8}\,\widehat{c}_{N-1/2}\,y_{N-1}+\tau\alpha_N T_0+\frac{h}{4}\tau\left(\widehat{f}_N+\widehat{f}_{N-1/2}\right) \quad (18)$$

Отсюда ищутся начальные значения прогоночных коэффициентов.

В итоге система квазилинейных разностных уравнений примет канонический вид

$$\begin{cases}\widehat{K}_0\widehat{y}_0+\widehat{M}_0\widehat{y}_1=\widehat{P}_0,\\ \widehat{A}_n\widehat{y}_{n-1}-\widehat{B}_n\widehat{y}_n+\widehat{D}_n\widehat{y}_{n+1}=-\widehat{F}_n, 1\le n\le N-1,\\ \widehat{K}_N\widehat{y}_N+\widehat{M}_{N-1}\widehat{y}_{N-1}=\widehat{P}_N\ .\end{cases} \quad (19)$$

Система (19) решается методом простых итераций. В методе простых итераций система (19) решается многократно на каждом шаге по времени, т.е. для получения решения $\widehat{y}_n, n=0...N$ в момент времени $t=t_{m+1}$ итерационная процедура организуется по схеме (8.3) из лекции №8.

$$\widehat{A}_n^{s-1}\widehat{y}_{n+1}^{s}-\widehat{B}_n^{s-1}\widehat{y}_n^{s}+\widehat{D}_n^{s-1}\widehat{y}_{n-1}^{s}=-\widehat{F}_n^{s-1}, \quad (20)$$

здесь s – номер итерации.

В качестве начального приближения $\widehat{y}_n^0$ задается сошедшееся решение $\widehat{y}_n$ с предыдущего шага $t=t_m$, то есть $\widehat{y}_n^0=\widehat{y}_n$.

Прекращение итераций происходит при условиями

$$max\left|\frac{\widehat{y}_n^{s}-\widehat{y}_n^{s-1}}{\widehat{y}_n^{s}}\right|\le\varepsilon. \quad (21)$$

В итоге для каждого момента времени $t = t_m, m = 1, 2, ...M$ получаем разностное решение $\widehat{y}_n$. Набор таких решений для всех $t = t_m, m = 1, 2, ...M$ оответствуетфункции двух переменных $T(x_n, t_m)$, являющейся решением исходного дифференциального уравнения (1).

Код программы

Код программы представлен в листингах 1-2.

Листинг 1: Класс MyApp.

```
import sys

from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QMessageBox

from Ui_mainwindow import Ui_MainWindow

from Modeller import Modeller

class MyApp(QtWidgets.QMainWindow):
    def __init__(self):
        super(MyApp, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.ui.set_def_button.clicked.connect(self.
            set_defaults)
        self.ui.run_button.clicked.connect(self.run)

        self.defaults = {
            "a1"     : 0.0134,
            "b1"     : 1,
            "c1"     : 4.35e-4,
            "m1"     : 1,
            "a2"     : 2.049,
            "b2"     : 0.563e-3,
            "c2"     : 0.528e5,
            "m2"     : 1,
            "alpha0" : 0.05,
            "alphaN" : 0.01,
            "l"      : 10,
```

```
31                  "T0"      : 300,
32                  "R"       : 0.5,
33                  "F0"      : 50,
34                  "h"       : 0.001,
35                  "t"       : 1
36              }
37
38              self.data = {
39                  "a1"      : None,
40                  "b1"      : None,
41                  "c1"      : None,
42                  "m1"      : None,
43                  "a2"      : None,
44                  "b2"      : None,
45                  "c2"      : None,
46                  "m2"      : None,
47                  "alpha0"  : None,
48                  "alphaN"  : None,
49                  "l"       : None,
50                  "T0"      : None,
51                  "R"       : None,
52                  "F0"      : None,
53                  "h"       : None,
54                  "t"       : None
55              }
56
57              self.set_defaults()
58
59          def set_defaults(self):
60              self.ui.lineEdit_a1.setText(str(self.defaults.get("a1"
                    )))
61              self.ui.lineEdit_b1.setText(str(self.defaults.get("b1"
                    )))
62              self.ui.lineEdit_c1.setText(str(self.defaults.get("c1"
                    )))
63              self.ui.lineEdit_m1.setText(str(self.defaults.get("m1"
                    )))
64
65              self.ui.lineEdit_a2.setText(str(self.defaults.get("a2"
                    )))
66              self.ui.lineEdit_b2.setText(str(self.defaults.get("b2"
```

```
            )))
67          self.ui.lineEdit_c2.setText(str(self.defaults.get("c2"
            )))
68          self.ui.lineEdit_m2.setText(str(self.defaults.get("m2"
            )))
69
70          self.ui.lineEdit_alpha0.setText(str(self.defaults.get(
            "alpha0")))
71          self.ui.lineEdit_alphaN.setText(str(self.defaults.get(
            "alphaN")))
72
73          self.ui.lineEdit_l.setText(str(self.defaults.get("l"))
            )
74          self.ui.lineEdit_T0.setText(str(self.defaults.get("T0"
            )))
75          self.ui.lineEdit_R.setText(str(self.defaults.get("R"))
            )
76          self.ui.lineEdit_F0.setText(str(self.defaults.get("F0"
            )))
77
78          self.ui.lineEdit_h.setText(str(self.defaults.get("h"))
            )
79          self.ui.lineEdit_t.setText(str(self.defaults.get("t"))
            )
80
81      def get_data(self):
82          try:
83              self.data["a1"] = float(self.ui.lineEdit_a1.text()
                )
84              self.data["b1"] = float(self.ui.lineEdit_b1.text()
                )
85              self.data["c1"] = float(self.ui.lineEdit_c1.text()
                )
86              self.data["m1"] = float(self.ui.lineEdit_m1.text()
                )
87
88              self.data["a2"] = float(self.ui.lineEdit_a2.text()
                )
89              self.data["b2"] = float(self.ui.lineEdit_b2.text()
                )
90              self.data["c2"] = float(self.ui.lineEdit_c2.text()
```

```
                )
91              self.data["m2"] = float(self.ui.lineEdit_m2.text()
                )
92
93              self.data["alpha0"] = float(self.ui.
                lineEdit_alpha0.text())
94              self.data["alphaN"] = float(self.ui.
                lineEdit_alphaN.text())
95
96              self.data["l"] = float(self.ui.lineEdit_l.text())
97              self.data["T0"] = float(self.ui.lineEdit_T0.text()
                )
98              self.data["R"] = float(self.ui.lineEdit_R.text())
99              self.data["F0"] = float(self.ui.lineEdit_F0.text()
                )
100
101             self.data["h"] = float(self.ui.lineEdit_h.text())
102             self.data["t"] = float(self.ui.lineEdit_t.text())
103
104         except ValueError:
105             return False
106         return True
107
108     def run(self):
109         if self.get_data():
110             print("Computing...")
111             mdlr = Modeller(self.data)
112             mdlr.compute()
113             print("Finish.")
114         else:
115             self.msg_box("Error!", "Error! Incorrect input!",
                QMessageBox.Critical)
116
117     def msg_box(self, title, message, type):
118         msg = QMessageBox(self)
119         msg.setIcon(type)
120         msg.setWindowTitle(title)
121         msg.setText(message)
122         msg.addButton('Ok', QMessageBox.AcceptRole)
123         msg.exec()
124
```

```
125
126     def main():
127         app = QtWidgets.QApplication(sys.argv)
128         window = MyApp()
129         window.show()
130         app.exec_()
131
132
133     if __name__ == '__main__':
134         main()
```

Листинг 2: Класс Modeller.

```
1     import matplotlib.pyplot as plt
2     import numpy as np
3     from math import fabs
4
5     class Modeller():
6         def __init__(self, data):
7             self.data = data
8
9             self.a1 = self.data.get("a1")
10            self.b1 = self.data.get("b1")
11            self.c1 = self.data.get("c1")
12            self.m1 = self.data.get("m1")
13
14            self.a2 = self.data.get("a2")
15            self.b2 = self.data.get("b2")
16            self.c2 = self.data.get("c2")
17            self.m2 = self.data.get("m2")
18
19            self.alpha0 = self.data.get("alpha0")
20            self.alphaN = self.data.get("alphaN")
21
22            self.l = self.data.get("l")
23            self.T0 = self.data.get("T0")
24            self.R = self.data.get("R")
25            self.F0 = self.data.get("F0")
26
27            self.h = self.data.get("h")
28            self.t = self.data.get("t")
29
```

```
30            self.d = (self.alphaN * self.l) / (self.alphaN - self.
                  alpha0)
31            self.c_koef = - self.alpha0 * self.d
32
33            self.eps = 1e-2
34
35        def c(self, T):
36            res = self.a2 + self.b2 * (T ** self.m2) - (self.c2 /
                  (T ** 2))
37            return res
38            #return 0
39
40        def f_plus_half(self, x, h, func):
41            res = (func(x) + func(x + h)) / 2
42            return res
43
44        def f_minus_half(self, x, h, func):
45            res = (func(x) + func(x - h)) / 2
46            return res
47
48        def old_k(self, x):
49            res = self.a1 / (x - self.b1)
50            return res
51
52        def k(self, T):
53            #res = self.old_k(self.x)
54            res = self.a1 * (self.b1 + self.c1 * (T ** self.m1))
55            return res
56
57        def alpha(self, x):
58            return self.c_koef / (x - self.d)
59
60        def p(self, x):
61            return 2 * self.alpha(x) / self.R
62
63        def f(self, x):
64            return 2 * self.alpha(x) * self.T0 / self.R
65
66        def __left_boundary_condition(self, T):
67            h8 = self.h / 8
68            h4 = self.h / 4
```

```
69              h2 = self.h / 2
70              c_p12 = self.f_plus_half(T[0], self.t, self.c)
71              chi_p12 = self.f_plus_half(T[0], self.t, self.k)
72              t_over_h = self.t / self.h
73
74              K0 = h8 * c_p12 + \
75                   h4 * self.c(T[0]) + chi_p12 * t_over_h + \
76                   self.t * h8 * self.p(h2) + (self.t * self.h) / 4
                        * self.p(0)
77
78              M0 = h8 * c_p12 - chi_p12 * t_over_h + \
79                   self.t * h8 * self.p(h2)
80
81              P0 = h8 * c_p12 * (T[0] + T[1]) + \
82                   h4 * self.c(T[0]) * T[0] + self.F0 * self.t + \
83                   self.t * h8 * (3 * self.f(0) + self.f(self.h))
84                   #self.t * h4 * (self.f(0) + self.f_plus_half(T
                        [0], self.t, self.f)) # ?
85              return K0, M0, P0
86
87          def __right_boundary_condition(self, T):
88              h8 = self.h / 8
89              h4 = self.h / 4
90              h2 = self.h / 2
91              c_m12 = self.f_minus_half(T[-1], self.t, self.c)
92              chi_m12 = self.f_minus_half(T[-1], self.t, self.k)
93
94              h8_cm12 = h8 * c_m12
95
96              KN = h4 * self.c(T[-1]) + h8_cm12 + \
97                   self.t * self.alphaN + self.t * chi_m12 / self.h +
                        \
98                   h4 * self.t * self.p(self.l) + h8 * self.t * self.
                        p(self.l - h2)
99
100             MN = h8_cm12 - \
101                  self.t * chi_m12 / self.h + \
102                  h8 * self.p(self.l - h2)
103
104             PN = h4 * self.c(T[-1]) * T[-1] + h8_cm12 * T[-1] +\
105                  h8_cm12 * T[-2] + \
```

```
106                 self.t * self.alphaN * self.T0 + \
107                 h4 * self.t * (self.f(self.l) + self.f(self.l -
                        h2))
108
109             return KN, MN, PN
110
111         def A(self, T):
112             return self.t / self.h * self.f_minus_half(T, self.t,
                    self.k)
113
114
115         def D(self, T):
116             return self.t / self.h * self.f_plus_half(T, self.t,
                    self.k)
117
118         def B(self, x, T):
119             return self.A(T) + self.D(T) + self.c(T) * self.h + \
120                  self.p(x) * self.h * self.t
121
122         def F(self, x, T):
123             return self.f(x) * self.h * self.t + self.c(T) * T *
                    self.h
124
125         def progon(self, T, K0, M0, P0, KN, MN, PN):
126             epsilon = [0, - M0 / K0]
127             eta = [0, P0 / K0]
128
129             x = self.h
130             n = 1
131             while x + self.h < self.l:
132                 An = self.A(T[n])
133                 Bn = self.B(x, T[n])
134
135                 newEps = self.D(T[n]) / (Bn - An * epsilon[n])
136                 epsilon.append(newEps)
137
138                 newEta = (self.F(x, T[n]) + An * eta[n]) / (Bn -
                        An * epsilon[n])
139                 eta.append(newEta)
140
141                 x += self.h
```

```
142                 n += 1
143
144             T = [0] * (n + 1)
145             T[n] = (PN - MN * eta[n]) / (KN + MN * epsilon[n])
146
147             for i in range(n-1, -1, -1):
148                 T[i] = epsilon[i + 1] * T[i + 1] + eta[i+1]
149
150             return T
151
152         def check_iter(self, T, T_next):
153             #если нагрев замедлился и изменение
                    температуры за
154             #шаг по времени достаточно мало
155             max = fabs((T[0] - T_next[0]) / T_next[0])
156             for i, j in zip(T, T_next):
157                 d = fabs(i - j) / j
158                 if d > max:
159                     max = d
160             return max < 1
161
162         def check_epsilon(self, T, T_next):
163             #если максимально изменившийся элемент
164             #изменился меньше, чем на eps
165             for i, j in zip(T, T_next):
166
167                 if fabs((i - j) / j) > self.eps:
168                     return True
169             return False
170
171         def __iterate(self):
172             res = []
173             n = int(self.l / self.h)
174             print(n)
175             T = [self.T0] * (n + 1)
176             T_new = [0] * (n + 1)
177             ti = 0
178             res.append(T)
179             while True:
180                 tmp = T
181                 self.x = self.h
```

```
182                 while True:
183                     K0, M0, P0 = self.__left_boundary_condition(
                            tmp)
184                     KN, MN, PN = self.__right_boundary_condition(
                            tmp)
185                     T_new = self.progon(tmp, K0, M0, P0, KN, MN,
                            PN)
186                     self.x += self.h
187 
188                     if self.check_iter(tmp, T_new):
189                         break
190                     tmp = T_new
191 
192                 res.append(T_new)
193                 ti += self.t
194                 if not self.check_epsilon(T, T_new):
195                     break
196                 T = T_new
197             return res, ti
198 
199         def compute(self):
200             res, ti = self.__iterate()
201 
202             xes = [i for i in np.arange(0, self.l, self.h)]
203             #ts = [i for i in range(0, int(ti), int(self.t))]
204             ts = [i for i in np.arange(0, ti, self.t)]
205 
206             s = 0
207             for i in res:
208                 if s % 2 == 0:
209                     plt.plot(xes, i[:-1])
210                 s += 2
211 
212             plt.plot(xes, res[-1][:-1], 'r')
213             plt.xlabel("x, см")
214             plt.ylabel("T, K")
215             plt.grid()
216             plt.show()
217 
218             s = 0
219             while s < self.l / 3:
```

```
220                 p = [j[int(s / self.h)] for j in res]
221                 plt.plot(ts, p[:-1])
222                 s += 0.1
223             plt.xlabel("t, sec")
224             plt.ylabel("T, K")
225             plt.grid()
226             plt.show()
```

Результаты работы программы

1. Представить разностный аналог краевого условия при и его краткий вывод интегро-интерполяционным методом.

Вывод был проведен выше в разделе «Предварительные вычисления».

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

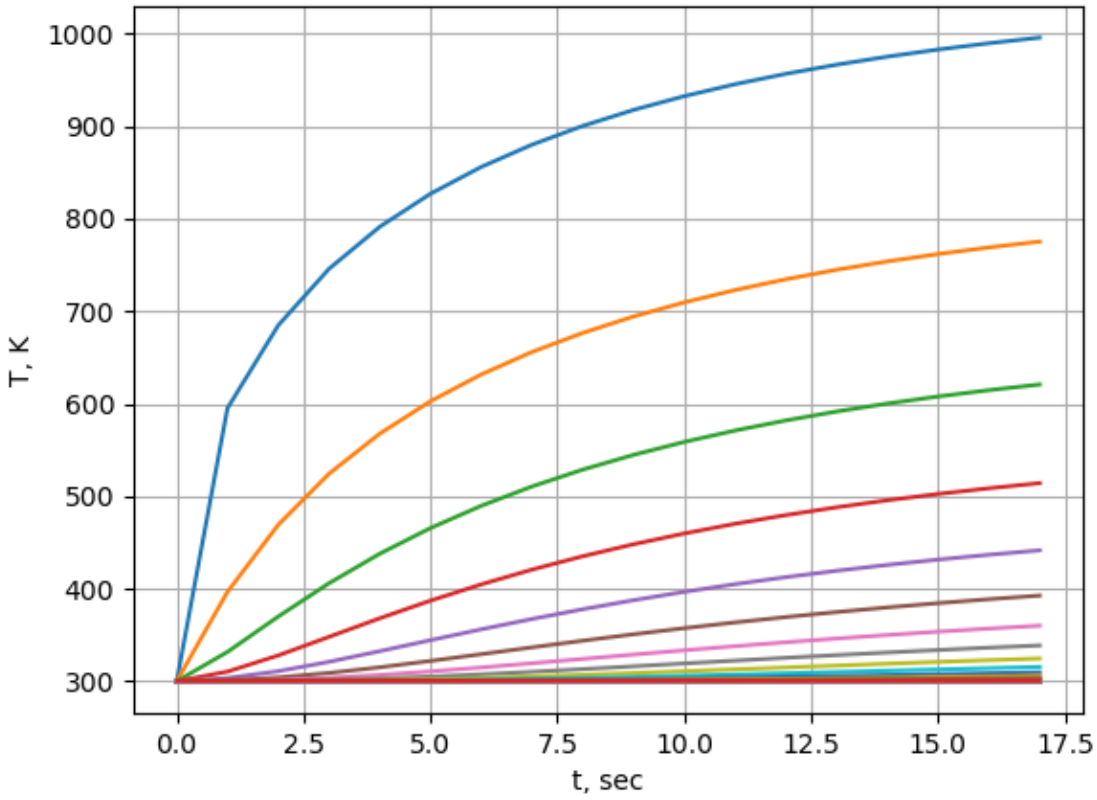

Рис. 1: График зависимости $T(x_n, t)$ при нескольких фиксированных значениях координаты x_n.

3. График зависимости температуры $T(x, t_m)$ от координаты x при нескольких фиксированных значениях времени t_m.

На рисунке 2 красным цветом представлено распределение $T(x, t_m)$ в момент времени, соответсвующий установившемуся режиму, когда поле перестает меняться с некоторой точностью (1e-2), т.е. имеет место выход на стационарный режим.

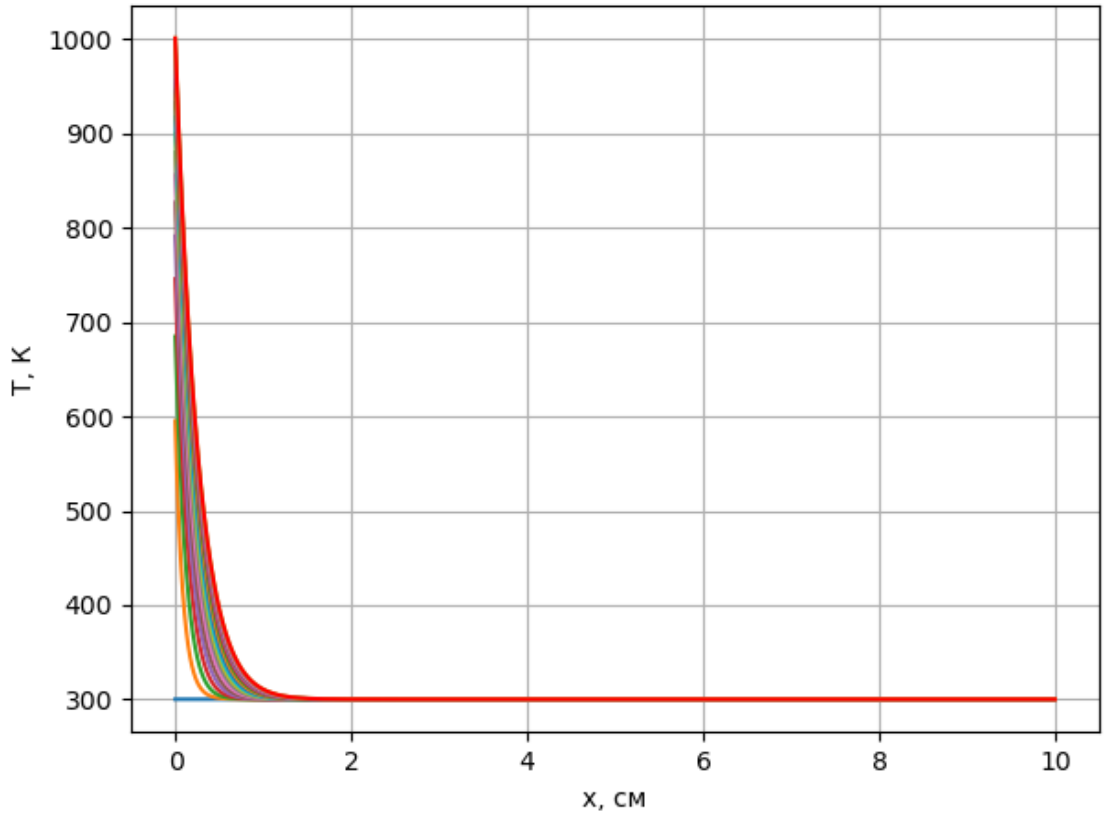

Рис. 2: График зависимости температуры $T(x, t_m)$ от координаты x при нескольких фиксированных значениях времени t_m.

Ответы на вопросы

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

1. 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 (рисунок 3).

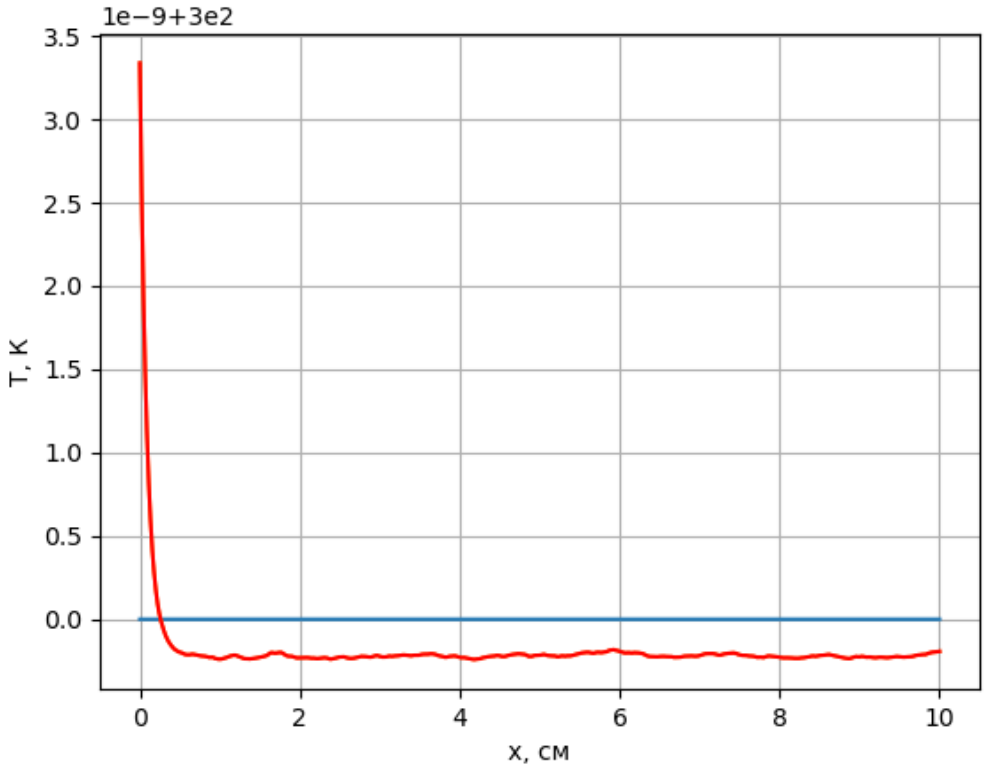

Рис. 3: $F(t) = 0$.

2. Если в настоящей работе задать поток постоянным, т.е. $F(t) = const$, то будет происходить формирование температурного поля от начальной-температуры $T0$ 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.

Необходимо задать $a_1 = 1.333$ и $b_1 = -3.333$, как в лабораторной №3.

На рисунке 4 представлен полученный график.

График из 3 лабораторной для сравнения приведен на рисунке 5.

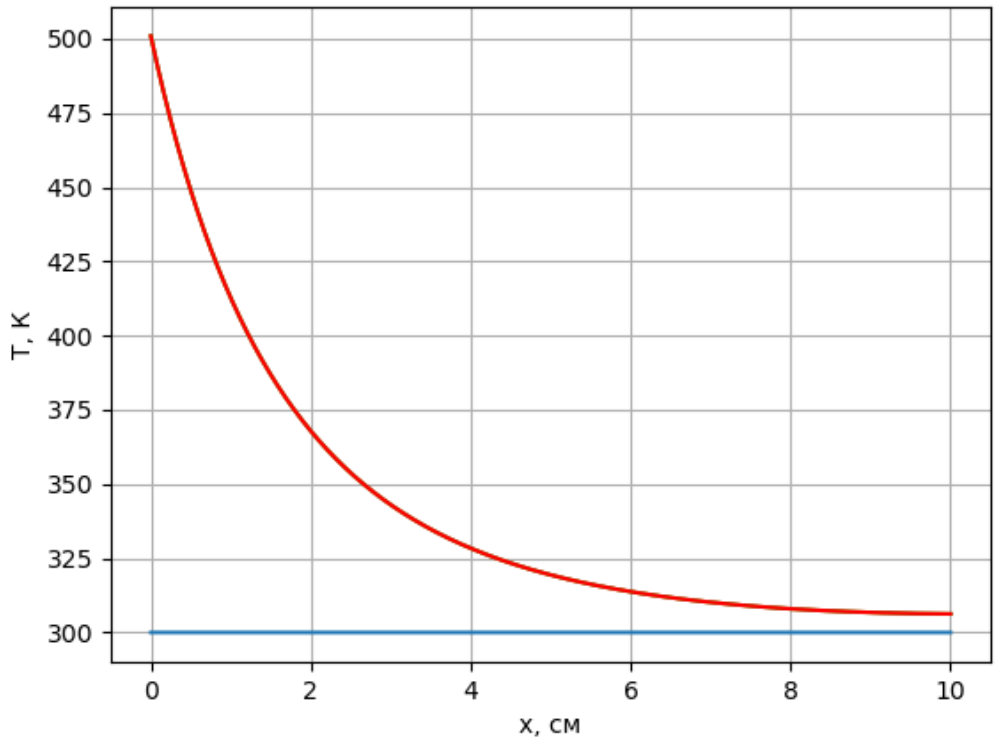

Рис. 4: $k(x)$ вместо $k(T)$.

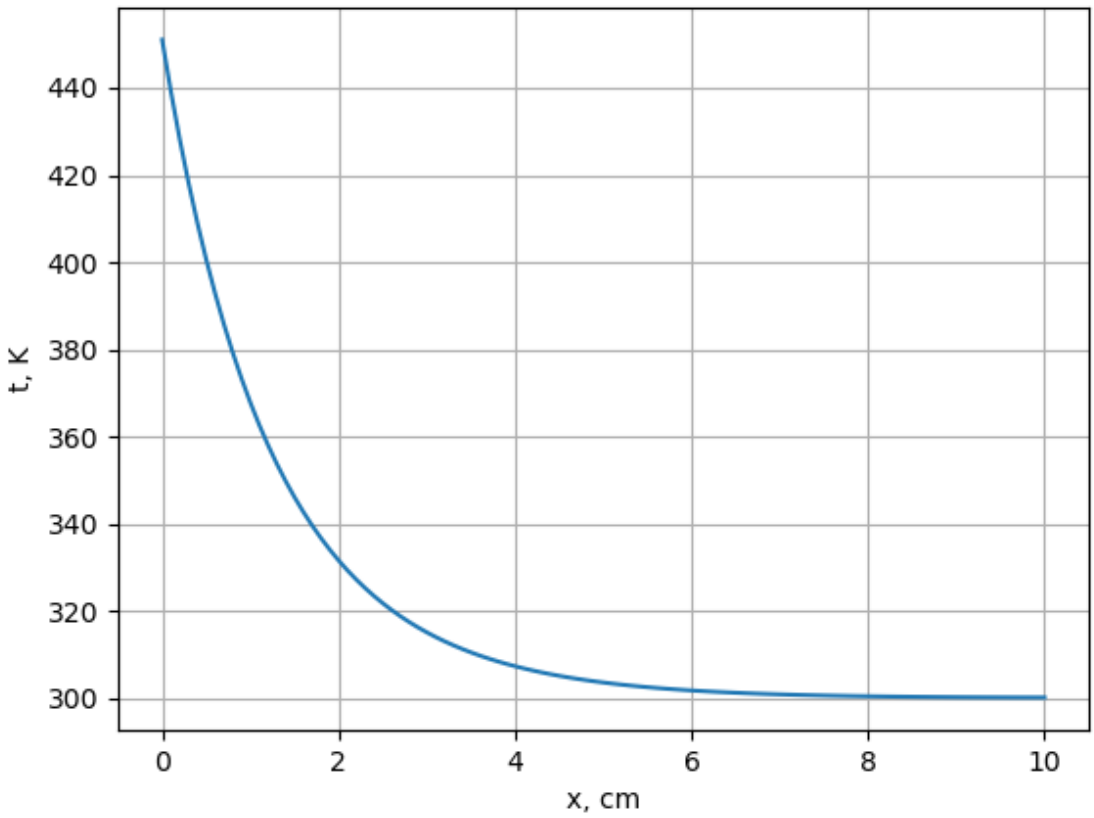

Рис. 5: График из лабораторной №3.

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.

Имеем:

$$\begin{cases} \widehat{K}_0\widehat{y}_0 + \widehat{M}_0\widehat{y}_1=\widehat{P}_0, \\ \widehat{A}_n\widehat{y}_{n-1} - \widehat{B}_n\widehat{y}_n + \widehat{D}_n\widehat{y}_{n+1}= - \widehat{F}_n, 1 \le n \le N-1, \\ \widehat{K}_N\widehat{y}_N + \widehat{M}_{N-1}\widehat{y}_{N-1}=\widehat{P}_N\ . \end{cases} \tag{22}$$

Выполняя линеаризацию по Ньютону последовательно по неизвестным

$$\begin{aligned} &\left(\widehat{A}_n\widehat{y}_{n-1} - \widehat{B}_n\widehat{y}_n + \widehat{D}_n\widehat{y}_{n+1} + \widehat{F}_n\right)\Big|_{(s-1)} + \\ &+ \widehat{A}_n^{s-1}\ \Delta y_{n-1}^s + \\ &+ \left(\frac{\delta\ \widehat{A}_n}{\delta\ \widehat{y}_n}\ \widehat{y}_{n-1} - \frac{\delta\ \widehat{B}_n}{\delta\ \widehat{y}_n}\ \widehat{y}_n - \widehat{B}_n + \frac{\delta\ \widehat{D}_n}{\delta\ \widehat{y}_n}\ \widehat{y}_{n+1} + \frac{\delta\ \widehat{F}_n}{\delta\ \widehat{y}_n}\right)\Big|_{(s-1)}\ \Delta\ \widehat{y}_n^s + \\ &+ \widehat{D}_n^{s-1}\ \Delta\ \widehat{y}_{n+1}^s = 0 \end{aligned} \tag{23}$$

Пусть

$$A_n = \widehat{A}_n^{s-1}, \tag{24}$$

$$B_n = \left(\frac{\delta\ \widehat{A}_n}{\delta\ \widehat{y}_n}\ \widehat{y}_{n-1} - \frac{\delta\ \widehat{B}_n}{\delta\ \widehat{y}_n}\ \widehat{y}_n - \widehat{B}_n + \frac{\delta\ \widehat{D}_n}{\delta\ \widehat{y}_n}\ \widehat{y}_{n+1} + \frac{\delta\ \widehat{F}_n}{\delta\ \widehat{y}_n}\right)\Big|_{(s-1)}, \tag{25}$$

$$D_n = \widehat{D}_n^{s-1} \tag{26}$$

$$F_n = \left(\widehat{A}_n\widehat{y}_{n-1} - \widehat{B}_n\widehat{y}_n + \widehat{D}_n\widehat{y}_{n+1} + \widehat{F}_n\right)\Big|_{(s-1)} \tag{27}$$

Тогда

$$A_n\Delta\ \widehat{y}^s_{n-1} + B_n\Delta\ \widehat{y}^s_n + D_n\Delta\ \widehat{y}^s_{n+1} = -F_n \tag{28}$$

Уравнение (28) решается методом прогонки, в результате находятся все $\Delta\ \widehat{y}^s_n$, после чего определяются значения искомой функции в узлах на s-итерации $\widehat{y}^s_n = \widehat{y}^s_{n-1} + \Delta\ \widehat{y}^s_n$. Итерационный процесс заканчивается при выполнении условия $max \left| \frac{\Delta \widehat{y}^s_n}{\widehat{y}^s_n} \right| \leq \varepsilon$, для всех $n = 0, 1, ..., N$.

Вывод

Таким образом, в ходе данной работы были получены навыки разработки алгоритмов решения смешанной краевой задачи при реализации моделей, построенных на квазилинейном уравнении параболического типа.