|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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»\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 4**

|  |  |
| --- | --- |
| **Тема** Программно- алгоритмическая реализация моделей на основе дифференциальных уравнений в частных производных с краевыми условиями II и III рода  **Студент** Белоусова Ю.С.  **Группа** ИУ7-61Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Градов В.М. |  |

Москва.

2020 г

**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 

 (1)

Краевые условия



В обозначениях уравнения (14.1) лекции №14

.

2. Разностная схема с разностным краевым условием при  получена в Лекции №14 (14.6),(14.7) и может быть использована в данной работе. Самостоятельно надо получить интегро -интерполяционным методом разностный аналог краевого условия при , точно так же, как это сделано при  (формула (14.7)). Для этого надо проинтегрировать на отрезке [xN-1/2, xN] выписанное выше уравнение (1) и учесть, что поток , а 

3. Значения параметров для отладки (все размерности согласованы)

, Вт/см К,

, Дж/см3К.

=0.0134, =1, =4.35 10-4, =1,

=2.049, =0.563 10-3, =0.528 105, =1.

,

0.05 Вт/см2 К,

0.01 Вт/см2 К,

 10 см,

300К,

0.5 см,

50 Вт/см2 (для отладки принять постоянным).

**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 , зависящее от координаты *x* и меняющееся во времени.

2. Свойства материала стержня привязаны к температуре, т.е. теплоемкость и коэффициент теплопроводности зависят от , тогда как в работе №3 зависит от координаты, а  = 0.

3. При  цилиндр нагружается тепловым потоком , в общем случае зависящим от времени, а в работе №3 поток был постоянный.

**Результаты работы**

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

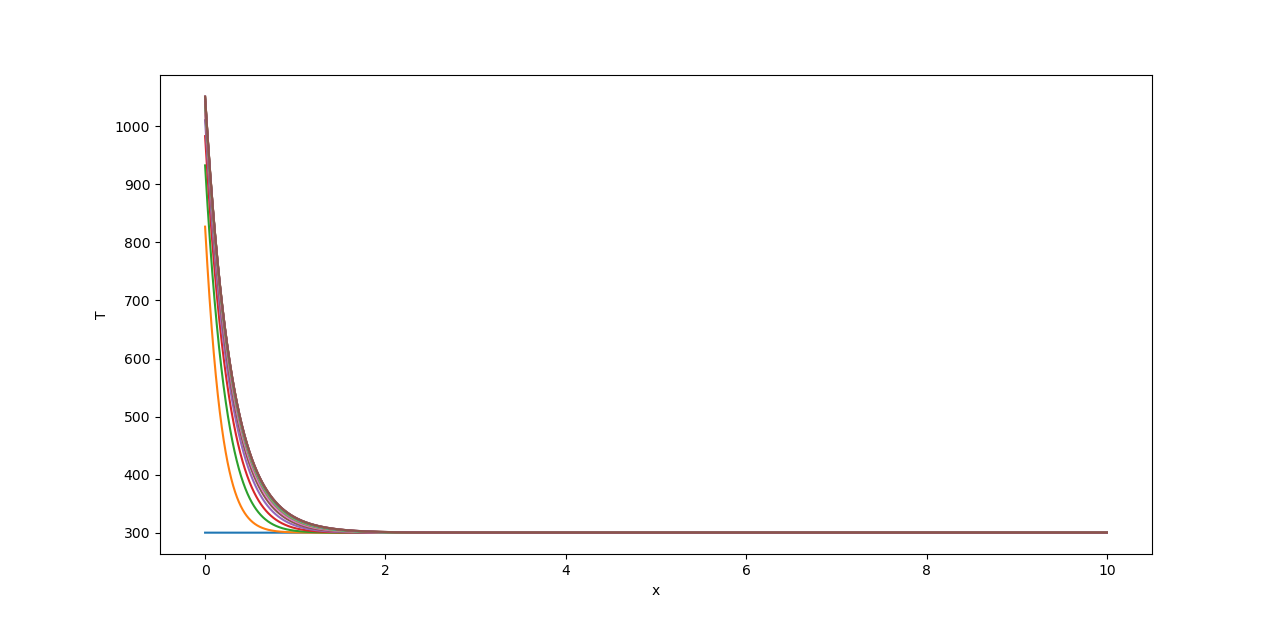
Учитывая, что , проинтегрируем уравнение (1) на отрезке [xN-1/2, xN]:

Вычисляя внутренние интегралы получаем:

Используя метод трапеций для 1, 3 и 4 интегралов и метод правых прямоугольников для 2, получим:

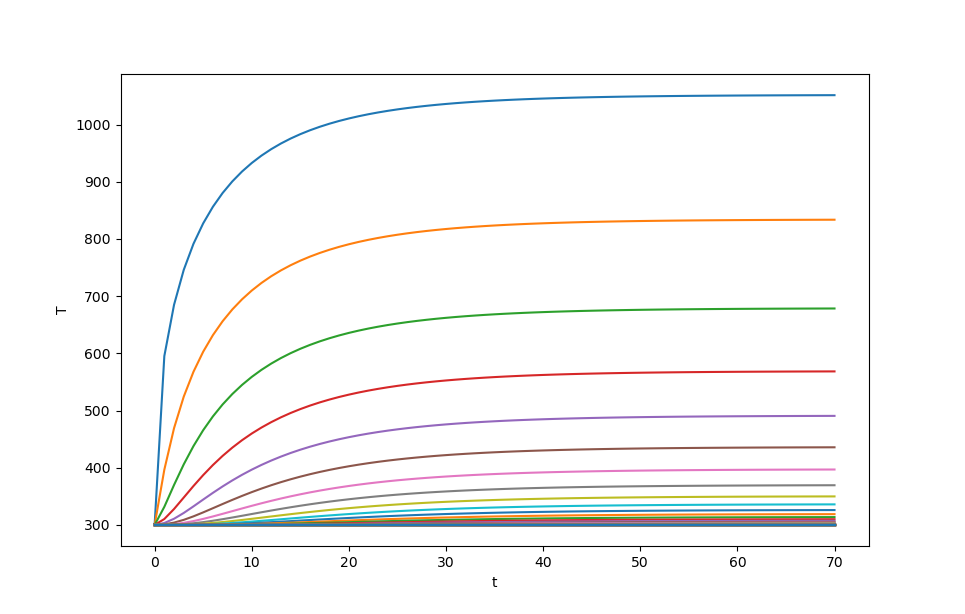
Учитывая, что , , , и приводя подобные слагаемые получаем разностный аналог:

1. График зависимости температуры  от координаты  при нескольких фиксированных значениях времени  (аналогично рисунку в лекции №14) при заданных выше параметрах.



Самый верхний график представляет распределение  в момент времени, соответствующий установившемуся режиму, когда поле перестает меняться с точностью , т.е. имеет место выход на стационарный режим.

1. График зависимости  при нескольких фиксированных значениях координаты .



Самый верхний график представляет случай *n*=0, т.е..

**Текст программы**

*Листинг 1. Файл с расчетами.*

from math import fabs  
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  
T0 = 300  
R = 0.5  
F0 = 50  
h = 1e-3  
t = 1  
  
def get\_consts():  
 cc = (-alpha0\*alphaN\*l)/(alphaN-alpha0)  
 dd = (alphaN\*l)/(alphaN-alpha0)  
 return cc, dd  
def alpha(x):  
 return cc/(x-dd)  
  
def k(T):  
 return a1 \* (b1 + c1 \* T \*\* m1)  
  
def c(T):  
 return a2 + b2 \* T \*\* m2 - (c2 / T \*\* 2)  
  
def A(T):  
 return t / h \* count\_minus\_half(k, T, t)  
  
def D(T):  
 return t / h \* count\_plus\_half(k, T, t)  
  
def B(x, T):  
 return A(T) + D(T) + h \* c(T) + h \* t \* 2 \* alpha(x) / R  
  
def F(x, T):  
 return h \* t \* 2 \* T0 \* alpha(x) / R + T \* h \* c(T)  
  
def count\_plus\_half(function, n, step):  
 return (function(n) + function(n + step)) / 2  
  
def count\_minus\_half(function, n, step):  
 return (function(n) + function(n - step)) / 2  
  
def find\_koef\_with\_left(T):  
 chalf = count\_plus\_half(c, T[0], t)  
 khalf = count\_plus\_half(k, T[0], t)  
 c0 = c(T[0])  
 K0 = h / 8 \* chalf + h / 4 \* c0 + t / h \* khalf + t \* h / 4 \* alpha(h / 2) / R + t \* h / 2 \* alpha(0) / R  
 M0 = h / 8 \* chalf - t / h \* khalf + t \* h / 4 \* alpha(h / 2) / R  
 P0 = h / 8 \* chalf \* (T[0] + T[1]) + h / 4 \* c0 \* T[0] + F0 \* t + t \* h / 4 \* T0 / R \* (3 \* alpha(0) + alpha(h))  
 return K0, M0, P0  
  
def find\_koef\_with\_right(T):  
 chalf = count\_minus\_half(c, T[-1], t)  
 khalf = count\_minus\_half(k, T[-1], t)  
 cN = c(T[-1])  
 KN = h / 8 \* chalf + h / 4 \* cN + t / h \* khalf + t \* alphaN + t \* h \* alpha(l - h / 2)/R/4 + t \* h \* alpha(l)/R/4  
 MN = h / 8 \* chalf - t / h \* khalf + t \* h \* alpha(l - h / 2)/R/4  
 PN = h / 8 \* chalf \* (T[-1] + T[-2]) + h / 4 \* cN \* T[-1] + t \* alphaN \* T0 + t \* h / 2 / R \* T0 \* (alpha(l) + alpha(l - h / 2))  
 return KN, MN, PN  
  
def find\_next (y):  
 K0, M0, P0 = find\_koef\_with\_left(y)  
 KN, MN, PN = find\_koef\_with\_right(y)  
 ksi = [0, -M0 / K0]  
 eta = [0, P0 / K0]  
 x = h  
 n = 1  
 while (x + h < l):  
 znam = (B(x, y[n]) - A(y[n]) \* ksi[n])  
 ksi.append(D(y[n]) / znam)  
 eta.append((F(x, y[n]) + A(y[n]) \* eta[n]) / znam)  
 n += 1  
 x += h  
 ynext = [0] \* (n + 1)  
 ynext[n] = (PN - MN \* eta[n]) / (KN + MN \* ksi[n])  
 for i in range(n - 1, -1, -1):  
 ynext[i] = ksi[i + 1] \* ynext[i + 1] + eta[i + 1]  
 return ynext  
  
def iterations():  
 N = int(l / h + 1)  
 ti = 0  
 y = [T0]\*N  
 res = []  
 res.append(y)  
 ynext = [0]\*N  
 while 1:  
 ycur = y  
 while 1:  
 ynext = find\_next(ycur)  
 maxfault = fabs((y[0] - ynext[0]) / ynext[0])  
 for i in range(1, N):  
 fault = fabs((y[i] - ynext[i]) / ynext[i])  
 if fault > maxfault:  
 maxfault = fault  
 if maxfault < 1:  
 break  
 ycur = ynext  
 res.append(ynext)  
 ti += t  
 flag = 0  
 for i in range(N):  
 if fabs((y[i] - ynext[i]) / ynext[i]) < 1e-4:  
 flag = 1  
 if flag:  
 break  
 y = ynext  
 return res, ti  
  
def work():  
 global cc, dd  
 cc, dd = get\_consts()  
 res, ti = iterations()  
 return res, ti

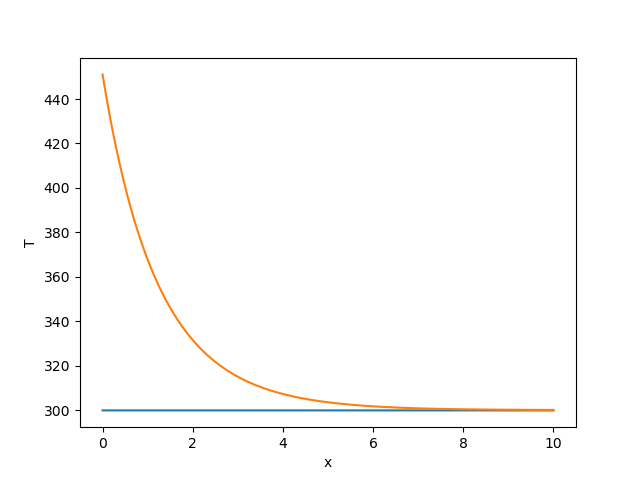
*Листинг 2. Main, вывод графиков*

import matplotlib.pyplot as plt  
import numpy as np  
from lab4 import work, l, h, t  
  
res, ti = work()  
length = len(res)  
print(length)  
te = list(np.arange(0, ti + t, t))  
x = list(np.arange(0, l + h, h))  
for i in range(0, len(x), 100):  
 graph = [y[i] for y in res]  
 plt.plot(te, graph)  
plt.xlabel("t")  
plt.ylabel("T")  
plt.show()  
for i in range(0, length, 5):  
 plt.plot(x, res[i])  
plt.plot(x, res[-1])  
plt.xlabel("x")  
plt.ylabel("T")  
plt.show()

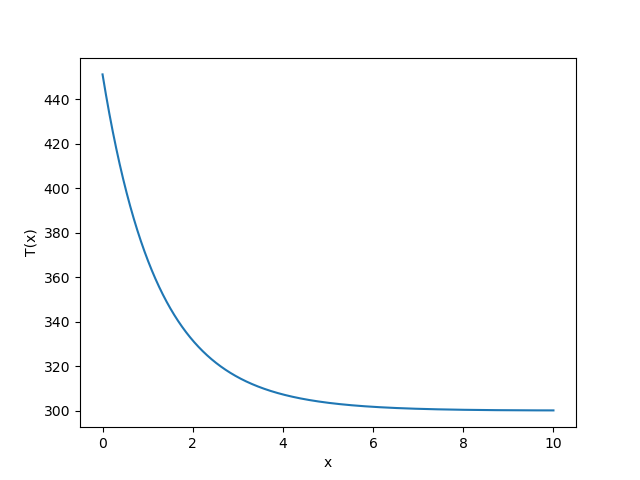
**Ответы на вопросы**

1. Приведите результаты тестирования программы (графики, общие соображения, качественный анализ).

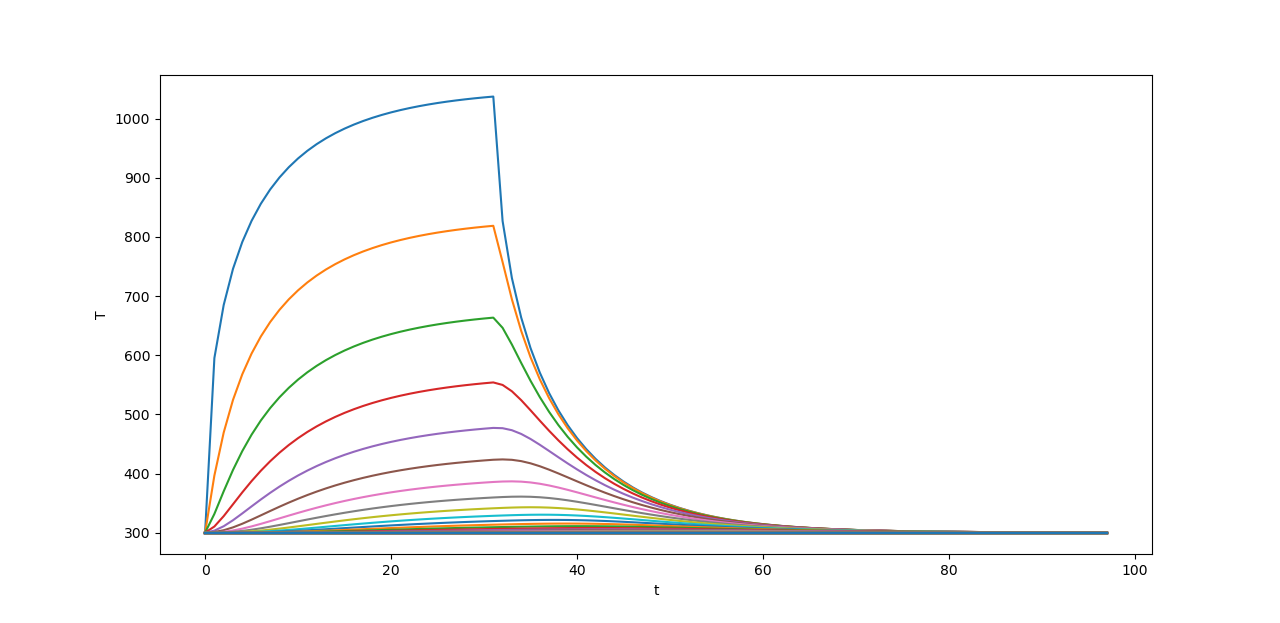
* Если=const, то будет происходить формирование температурного поля от начальной температуры  до некоторого установившегося (стационарного) распределения . Это поле в дальнейшем с течением времени меняться не будет и должно совпасть с температурным распределением , получаемым в лаб. работе №3, если все параметры задач совпадают, а именно для теплопроводности вместо  использованоиз лаб. работы №3, а теплопроводность с = 0. При таких условиях получаем график:



Соответствующий график из работы №3:

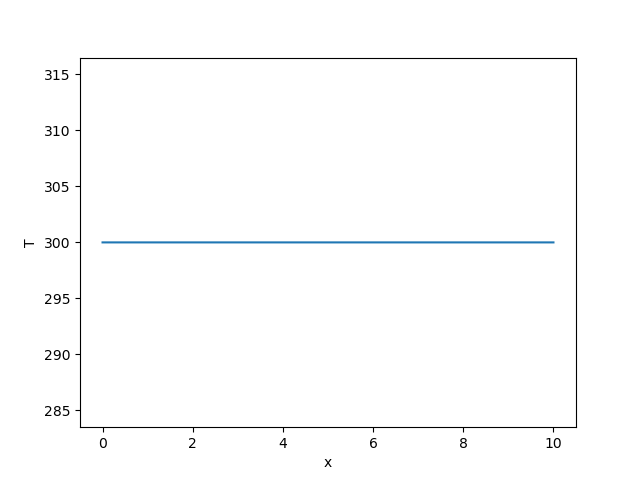


* Если после разогрева стержня положить поток =0, то будет происходить остывание, пока температура не выровняется по всей длине и не станет равной . Это демонстрирует следующий график:



Здесь на 31 секунде F становится равным 0.

* Если изначально положить F = 0, тогда нагрева нет, температура стержня постоянна и равна температуре окружающей среды:



1. Выполните линеаризацию уравнения (14.8) по Ньютону, полагая для простоты, что все коэффициенты зависят только от одной переменной . Приведите линеаризованный вариант уравнения и опишите алгоритм его решения.

Уравнение: , 1≤n≤N-1

Выполняем линеаризацию по Ньютону по переменным :

Учитывая, что все коэффициенты зависят только от , получаем:

Уравнение решается методом прогонки:

Каноническая форма СЛАУ с трехдиагональной матрицей имеет вид:

Тогда для нашего уравнения получаем:

, где

В результате прогонки найдем все , тогда значение функции .

Итерационный процесс заканчивается, если выполняется условие: