Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Лабораторная работа № 13

«Метод сеток решения одномерного нестационарного уравнения теплопроводности»

Выполнил:

студент гр. 953506

Кондрашов И.Д.

Проверил:

Анисимов В.Я.

Минск, 2021

**Цель работы**

1. Изучить метод разностных аппроксимаций для уравнения теплопроводности;

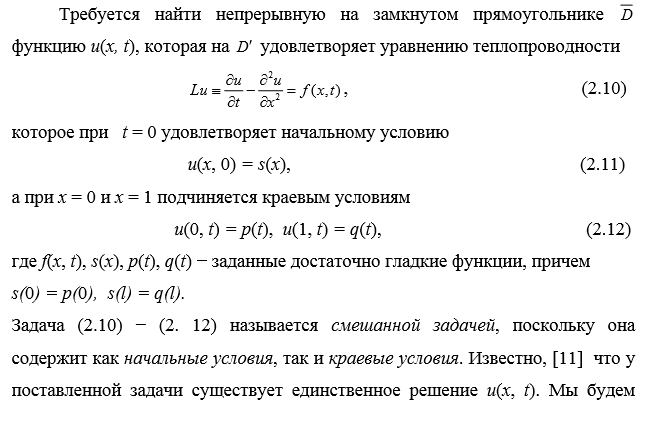
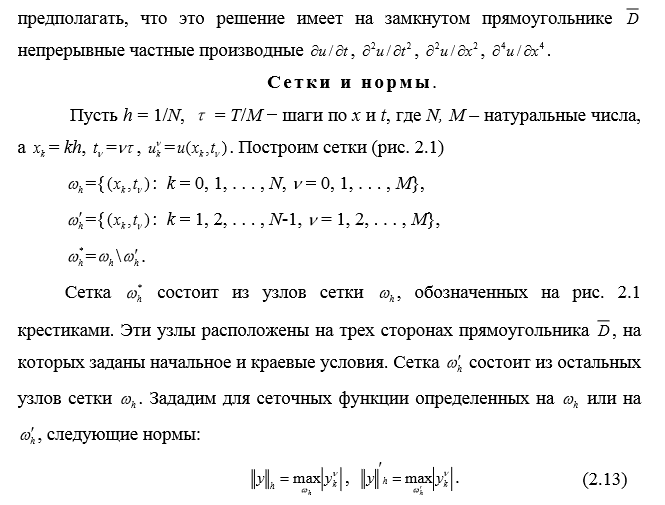
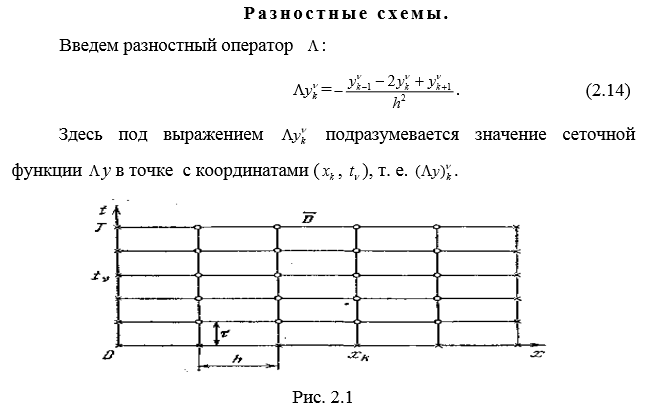
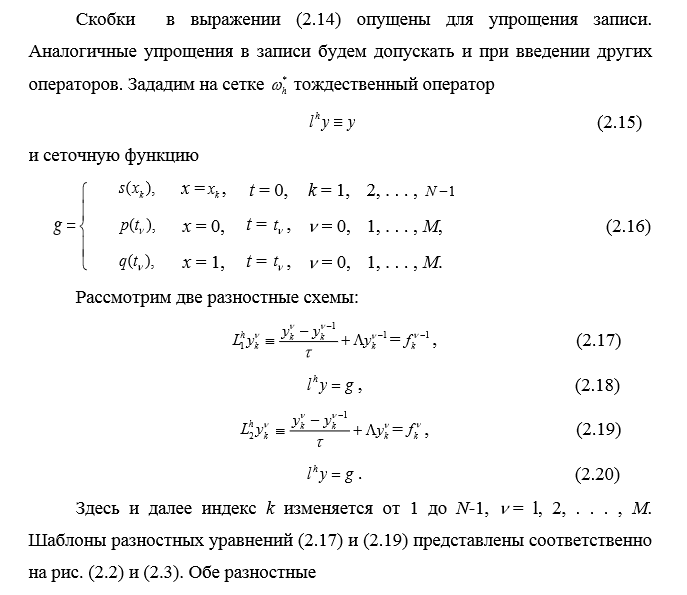
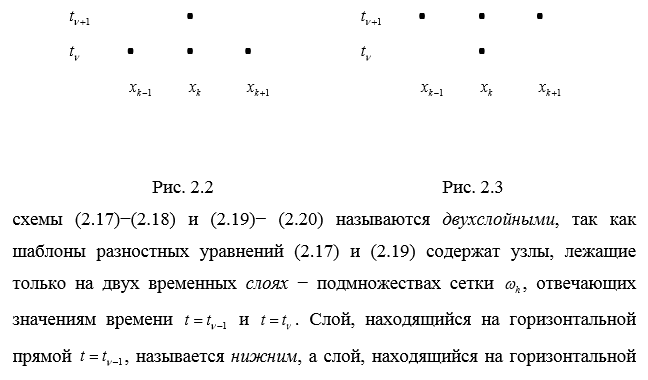
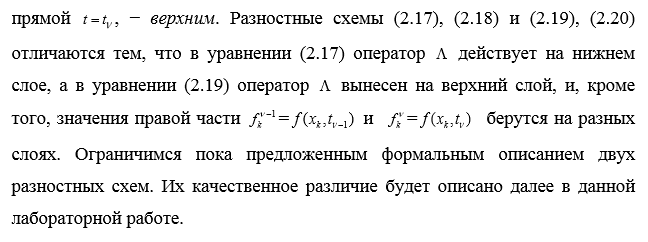
2. Составить алгоритмы решения уравнения теплопроводности методом сеток, применимыми для организации вычислений на ПЭВМ;

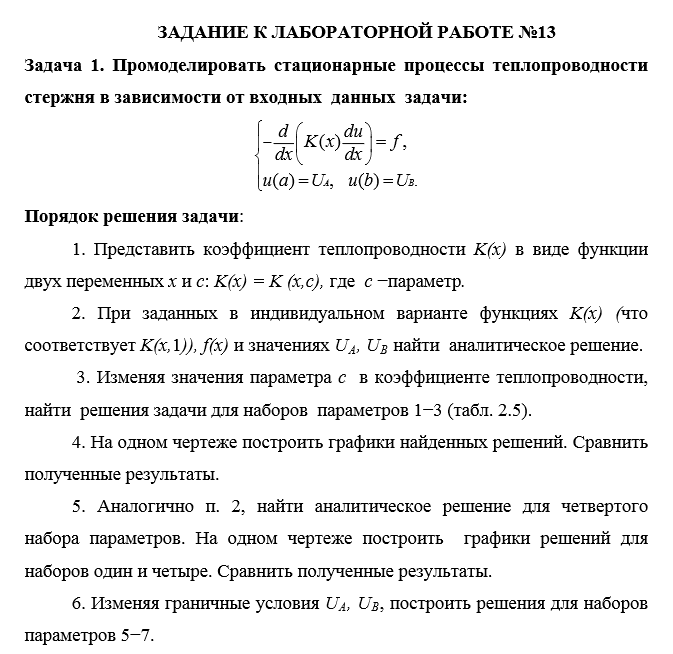
3. Составить программы решения уравнения теплопроводности по разработанным алгоритмам;

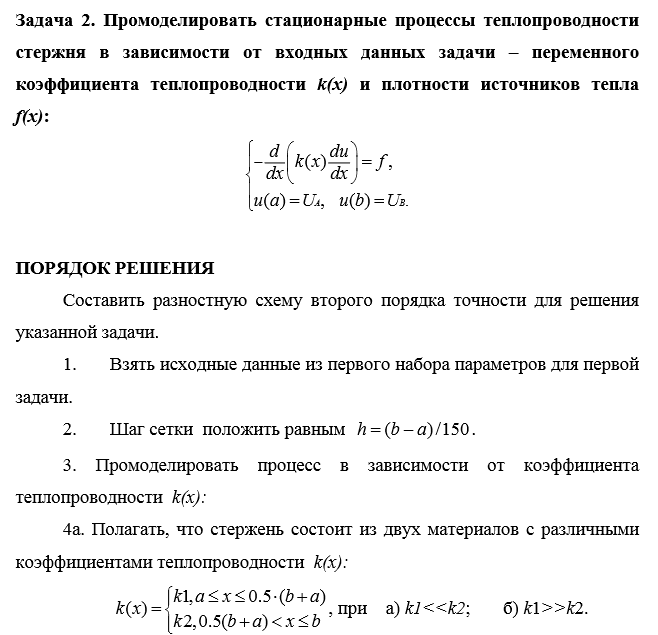
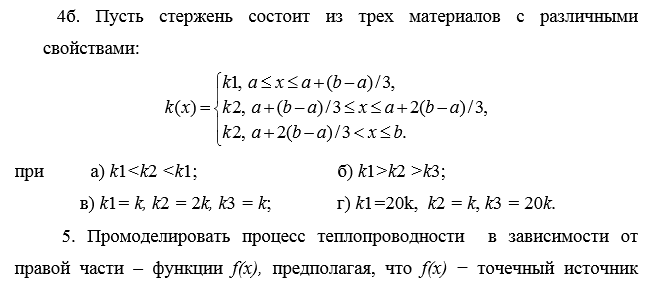
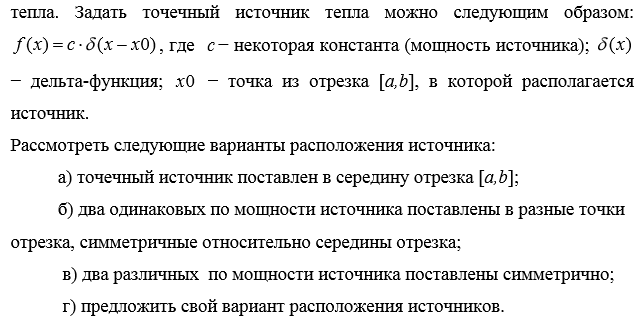
4. Выполнить тестовые примеры и проверить правильность работы программ;

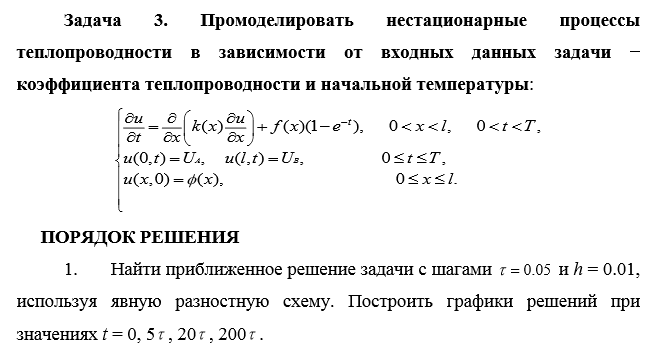
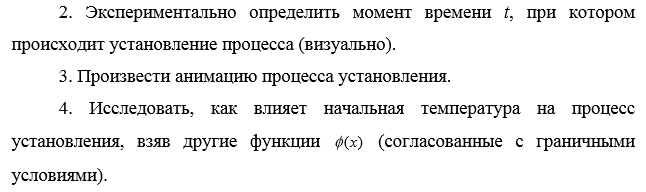
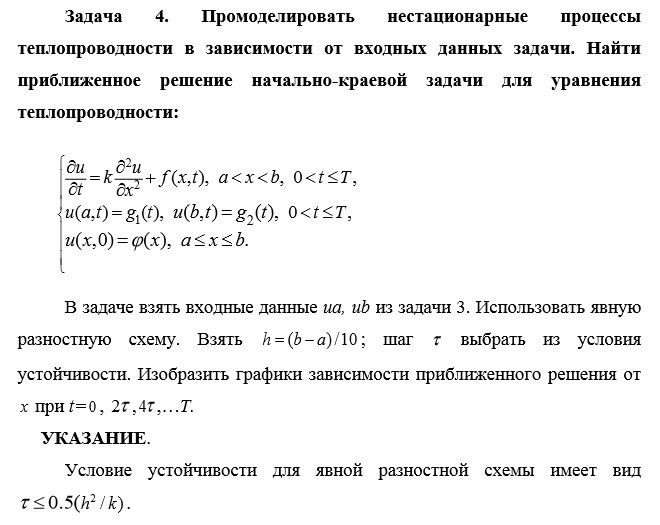
5. Получить численное решение заданного уравнения теплопроводности.

**Краткие теоретические сведения**



**Листинг кода**

Файл Lr13.py(основной)

import sympy  
from sympy.solvers import solve  
import matplotlib.pyplot as plt  
import numpy as np  
from matplotlib import cm  
import math  
import support  
  
import pylab  
from mpl\_toolkits.mplot3d import Axes3D  
  
from numpy import linspace  
from collections import namedtuple  
  
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  
  
  
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 = function\_1  
  
 func = -k \* (yk\_p1 - 2 \* yk + yk\_m1) / h \*\* 2 - func(x)  
 return func  
  
  
# this function returns final thermal conductivity function  
def solve\_thermal\_conductivity\_equation(f, k, c, a, b, u\_a, u\_b):  
 c1, c2 = sympy.Symbol('c1'), sympy.Symbol('c2')  
 true\_eq = 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 = 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 = support.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)  
  
  
# TASK 1  
def task\_1():  
 print("Задача №1")  
  
 var\_1 = solve\_thermal\_conductivity\_equation(function\_1, k\_1, 1, a, b, u\_a, u\_b)  
 var\_2 = solve\_thermal\_conductivity\_equation(function\_1,  
 lambda x, c: c \* k\_1(x, c),  
 2, a, b, u\_a, u\_b)  
 var\_3 = solve\_thermal\_conductivity\_equation(function\_1,  
 lambda x, c: c \* 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')  
 plt.plot(D, func\_y2, c='green', label='c=2')  
 plt.plot(D, func\_y3, c='blue', label='c=0.1')  
  
 plt.legend()  
 plt.show()  
  
 var\_4 = solve\_thermal\_conductivity\_equation(function\_1,  
 lambda x, c: 1 / 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 2')  
  
 plt.legend()  
 plt.show()  
  
 var\_5 = solve\_thermal\_conductivity\_equation(function\_1, k\_1, 1, a, b, -u\_a,  
 u\_b)  
 var\_6 = solve\_thermal\_conductivity\_equation(function\_1, k\_1, 1, a, b, u\_a,  
 -u\_b)  
 var\_7 = solve\_thermal\_conductivity\_equation(function\_1, 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 task\_2():  
 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  
 # TASK 2  
 print("Задача №2")  
 # a  
 data\_a1 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, func\_for\_partition\_2,  
 [(0.5 \* (b + a), 0.01), (b, 150)])  
 print('h=', data\_a1.step)  
 data\_a2 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, 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('задача 2 пункт 3.1')  
 plt.grid(True)  
 plt.legend()  
 plt.show()  
  
 # b  
 data\_b1 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, func\_for\_partition\_2,  
 [(a + (1 / 3) \* (b - a), 0.2),  
 (a + (2 / 3) \* (b - a), 0.6),  
 (b, 0.9)])  
 data\_b2 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, func\_for\_partition\_2,  
 [(a + (1 / 3) \* (b - a), 0.9),  
 (a + (2 / 3) \* (b - a), 0.6),  
 (b, 0.2)])  
 data\_b3 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, func\_for\_partition\_2,  
 [(a + (1 / 3) \* (b - a), my\_k),  
 (a + (2 / 3) \* (b - a), 2 \* my\_k),  
 (b, my\_k)])  
 data\_b4 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, 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('задача 2 пункт 3.2')  
 plt.grid(True)  
 plt.legend()  
 plt.show()  
  
 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  
  
 # c  
 data\_c1 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, func\_for\_partition\_2,  
 [(b, my\_k)], delta\_1)  
  
 data\_c2 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, func\_for\_partition\_2,  
 [(b, my\_k)], delta\_2)  
 data\_c3 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, func\_for\_partition\_2,  
 [(b, my\_k)], delta\_3)  
 data\_c4 = differences\_method(ITER\_COUNT, a, b, u\_a, u\_b, 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('задача №2 пункт 3.4: ')  
 plt.legend()  
 plt.grid(True)  
 plt.show()  
  
  
def task\_3():  
 def specified(a, b, t\_0, t\_n, h, tetta, fi, g1, g2, k, f, func\_for\_partition):  
 # iteration count  
 columns = int((b - a) / h) + 1  
 rows = int((t\_n - t\_0) / tetta) + 1  
 solving = np.zeros((rows, columns))  
  
 # boundary conditions initialization  
 for index, value in enumerate(j for j in np.arange(a, b + h, h)):  
 solving[0][index] = fi(value)  
 for index, value in enumerate(j for j in np.arange(t\_0, t\_n, tetta)):  
 solving[index][0] = g1(value)  
 solving[index][-1] = g2(value)  
 alfa = tetta / (h \*\* 2)  
 a\_k = [k \* alfa] \* columns  
 b\_k = [-(1 + 2 \* k \* alfa)] \* columns  
 c\_k = [alfa] \* columns  
 x\_points = linspace(a, b, columns).tolist()  
 for index\_t, time in enumerate((j for j in np.arange(t\_0, t\_n, tetta)), 1):  
 d\_k = []  
 for index\_x, coord in enumerate((j for j in x\_points)):  
 d\_k.append(-tetta \* f(coord, time) - solving[index\_t - 1][index\_x])  
 d\_k[0] = d\_k[0] - a\_k[0] \* g1(time)  
 d\_k[-1] = d\_k[-1] - c\_k[-1] \* g2(time)  
 solving[index\_t] = support.sweep\_method(a\_k, b\_k, c\_k, d\_k)  
 return solving  
  
 # TASK 3  
 print('Задача № 3')  
 a = 0.1  
 ua = 2  
 b = 1.1  
 ub = 4  
 h = 0.01  
 tetta = 0.05  
 k = 1  
 fi = lambda x: 0  
 T1 = 0  
 T = 10  
 f = lambda x, t: (x \*\* 3) + 2  
  
 def func\_for\_partition(coord, time, T\_prev, T\_cur, T\_next, h, tetta, f):  
 return tetta / (h \*\* 2) \* T\_prev + (1 - 2 \* tetta / (h \*\* 2)) \* T\_cur + tetta / (h \*\* 2) \* T\_next + tetta \* f(  
 coord, time)  
  
 solving = specified(a, b, T1, T, h, tetta, fi, lambda t: ua, lambda t: ub, k, f, func\_for\_partition)  
 points = np.arange(a, b + h, h)  
  
 plt.plot(points, solving[0], color='red', label='t=0')  
 plt.legend()  
  
 plt.plot(points, solving[5], color='green', label='t=5t')  
 plt.legend()  
  
 plt.plot(points, solving[20], color='blue', label='t=20t')  
 plt.legend()  
  
 plt.plot(points, solving[199], color='black', label='t=200t')  
 plt.title('задача 3: ')  
 plt.legend()  
 plt.grid(True)  
 plt.show()  
  
 def plot\_3d():  
 x = np.arange(a, b + h, h)  
 y = np.arange(T1, T + tetta, tetta)  
  
 xgrid, ygrid = np.meshgrid(x, y)  
 fig = pylab.figure()  
 axes = Axes3D(fig)  
 axes.plot\_surface(xgrid, ygrid, solving, rstride=4, cstride=4, cmap=cm.jet)  
 pylab.show()  
  
 plot\_3d()  
 solving = specified(a, b, 0, T, h, tetta, lambda x: math.sin(x) \* (2 - x) / 5, lambda t: ua, lambda t: ub,  
 k, f, func\_for\_partition)  
  
 plt.plot(points, solving[0], color='red', label='t=0')  
 plt.legend()  
  
 plt.plot(points, solving[5], color='green', label='t=5t')  
 plt.legend()  
  
 plt.plot(points, solving[20], color='blue', label='t=20t')  
 plt.legend()  
  
 plt.plot(points, solving[199], color='black', label='t=200t')  
 plt.title('')  
 plt.legend()  
 plt.grid(True)  
 plt.show()  
  
 plot\_3d()  
  
  
def task\_4():  
 # TASK 4  
 print('Задача № 4')  
 a = -1  
 b = 1  
 ua = lambda t: 0  
 ub = lambda t: 0  
 u = lambda x: 0  
 T = 10  
 h = 0.1  
 tetta = (h \*\* 2) / 4  
 f = lambda x, t: 0  
  
 def specified(a, b, t\_0, t\_n, h, tetta, u, g1, g2, f, func\_for\_partition):  
 # iteration count  
 columns = int((b - a) / h) + 1  
 rows = int((t\_n - t\_0) / tetta) + 1  
 solving = np.zeros((rows, columns))  
  
 # boundary conditions initialization  
 for index, value in enumerate(j for j in np.arange(a, b + h, h)):  
 solving[0][index] = u(value)  
 for index, value in enumerate(j for j in np.arange(t\_0, t\_n, tetta)):  
 solving[index][0] = g1(value)  
 solving[index][-1] = g2(value)  
  
 # all values calculation  
 for index\_x, coord in enumerate((j for j in np.arange(a + h, b - h, h)), 1):  
 for index\_t, time in enumerate((j for j in np.arange(t\_0 + tetta, t\_n, tetta)), 1):  
 solving[index\_t][index\_x] = func\_for\_partition(coord,  
 time - tetta,  
 solving[index\_t - 1][index\_x - 1],  
 solving[index\_t - 1][index\_x],  
 solving[index\_t - 1][index\_x + 1],  
 h,  
 tetta,  
 f)  
  
 return solving  
  
 def func3\_for\_partition\_2(coord, time, T\_prev, T\_cur, T\_next, h, tetta):  
 return 2 \* tetta / (h \*\* 2) \* (T\_next - 2 \* T\_cur + T\_prev) + T\_cur  
  
 def func\_for\_partition\_2(coord, time, T\_prev, T\_cur, T\_next, h, tetta, f):  
 return 0.05 \* tetta / (h \*\* 2) \* (T\_next - 2 \* T\_cur + T\_prev) + T\_cur + tetta \* f(  
 coord, time)  
  
 solving = specified(a, b, 0, T, h, tetta, u, ua, ub, f, func\_for\_partition\_2)  
  
  
 def plot\_3d():  
 x = np.arange(a, b + h, h)  
 y = np.arange(0, T, tetta)  
  
 xgrid, ygrid = np.meshgrid(x, y)  
 fig = pylab.figure()  
 axes = Axes3D(fig)  
 print(len(x))  
 print(len(y))  
 print(len(solving[0]))  
 print(len(solving))  
  
 axes.plot\_surface(xgrid, ygrid, solving, rstride=4, cstride=4, cmap=cm.jet)  
 pylab.show()  
  
 plot\_3d()  
  
  
task\_1()  
task\_2()  
task\_3()  
task\_4()

Файл support.py(вспомогательный)

import math  
LINEWIDTH = 0.5  
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 check\_eps(current, prev, eps, debuging=True):  
 eps\_t = max([math.fabs(current[i\*2]-prev[i]) for i in range(len(prev))])  
 if debuging:  
 print('Check: ', eps\_t)  
 if eps\_t > eps:  
 return False  
 return True  
  
  
def deep\_copy(system):  
 return [item for item in system]  
  
  
def get\_h(a, b, n):  
 return (b - a) / n

**Результаты**

Задача 1.

**Chart, line chart

Description automatically generated**

**Chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

Задача 2.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Задача 3.

Chart, line chart

Description automatically generated

Chart, surface chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, surface chart

Description automatically generated

**Выводы**

В ходе выполнения лабораторной работы:

* Изучен метод разностных аппроксимаций для уравнения теплопроводности
* Составлены алгоритмы решения уравнения теплопроводности методом сеток, применимыми для организации вычислений на ПЭВМ
* Составлены программы решения уравнения теплопроводности по разработанным алгоритмам
* Выполнены тестовые примеры и проверена правильность работы программ
* Получено численное решение заданного уравнения теплопроводности

**Задания**

* Первое задание было направлено на изучение функций, описывающих функцию теплопроводности.
* Вторая задача направлена на изучение поведения решения в зависимости от функции теплопроводности.
* Третья и четвертая задачи исследуют нестационарное уравнение теплопроводности.