МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий математики и механики**

Направление подготовки: «Фундаментальная информатика и информационные технологии»

**ОТЧЕТ**

По курсу «Вычислительные методы»

На тему:

**Численное решение начально-краевой задачи для интегро-дифференциального уравнения в частных производных.**

**Выполнил**:студент группы 381806-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Напылов Е.И.

Подпись

**Научный руководитель**:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Эгамов А.И.

Подпись

Нижний Новгород

2021

# Содержание

[Содержание 2](#_Toc71937987)

[Введение 3](#_Toc71937988)

[Постановка задачи 4](#_Toc71937989)

[Теоретическое решение 5](#_Toc71937990)

[Реализация 7](#_Toc71937991)

[Руководство к программе 9](#_Toc71937992)

[Заключение 13](#_Toc71937993)

[Список литературы 14](#_Toc71937994)

[Приложение 15](#_Toc71937995)

# Введение

Дифференциальное уравнение в частных производных – это уравнение для функции с двумя и более переменными, в котором имеется хотя бы одна частная производная этой функции. Чаще всего возникает потребность в поиске решений, которые удовлетворяют условиям начально-краевой задачи.

Решение начально-краевой задачи для ДУ заключается в поиске решения, которое удовлетворяет условиям, задающим поведение данного уравнения на границах исследуемой области, а также в начальный момент времени.

Решение таких задач вручную аналитическим способом практически невозможно на практике. Поэтому были разработаны численные методы решения начально-краевой задачи, которые могут быть реализованы на компьютере.

# Постановка задачи

Дан тонкий однородный стержень с теплоизолированными концами длины l. На процесс изменения температуры стержня осуществляется некое воздействие для достижения определённых целей, например, через стержень пропускается электрический ток.

Требуется найти на множестве Q=[0,l]x[0,T[, l>0, T>0 непрерывно-дифференцируемую по времени и дважды по х функцию y(x,t). Функция должна быть решением уравнения:



Функция должна удовлетворять однородным условиям второго рода:



Функция должна удовлетворять начальному условию:



Здесь а – константа, ф(х) >0 задает начальное распределение температуры, дважды непрерывно дифференцируема на [0,l] и удовлетворяет условиям согласования и условию



Непрерывная функция u(x, t) – уравнение с обратной связью, представимая в одном из двух вариантов:





где b(x) – непрерывная на [0,l] управляющая функция.

# Теоретическое решение

Во-первых, необходимо определить нулевой слой разностной схемы. В качестве начальной функции предложено выбрать :

Далее необходимо вычислить интеграл с помощью формулы Симпсона. Эта формула применяется на каждом шаге.

,

Теперь необходимо составить неявную разностную схему с погрешностью :

|  |  |
| --- | --- |
| **\***  Схема является устойчивой при . |  |

Требуется составить трехточечные разностные производные первого порядка для краевых условий с погрешностью второго порядка. Для этого краевые условия запишем в виде разностных производных:

Теперь подставим вторую производную в выражение:

|  |  |
| --- | --- |
| **\*** |  |

|  |  |
| --- | --- |
| **\*** |  |

Уравнения, помеченные \* образуют систему уравнений. Система имеет размерность равную k+1. Для решения системы методом прогонки необходимо привести ее к трехдиагональному виду.

Задание A:

Задание B:

Для удобства перепишем систему в другом виде

|  |  |
| --- | --- |
| . |  |

Выразим и через и подставим в исходную систему:

данное условие независимо от у, если:

⇒

В части В нужно разделить функцию на ее интеграл по длине стержня.

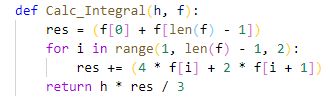
# Реализация

**Используемые технологии.**

Для разработки программы был выбран современный язык программирования Python, который ориентирован на высокую скорость разработки. Для работы с массивами и матрицами использован модуль numpy. Модуль matplotlib был использован для построения графиков. Пользовательский интерфейс реализован с помощью модуля PySimpleGui.

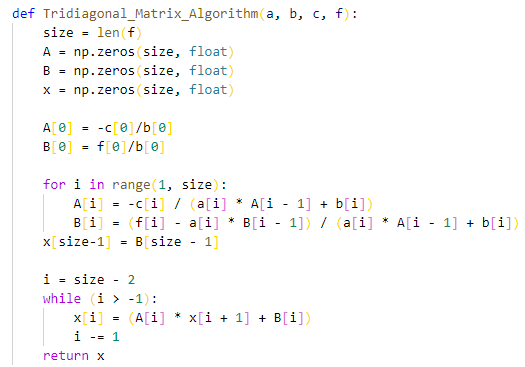
**Метод Симпсона для вычисления интеграла.**

Суть метода заключается в приближении подынтегральной функции на отрезке [a,b] интерполяционным многочленом второй степени. Метод разделен на две задания А и В.

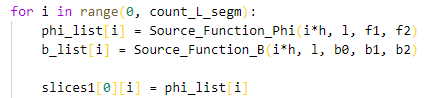


**Метод прогонки для решения СЛАУ.**

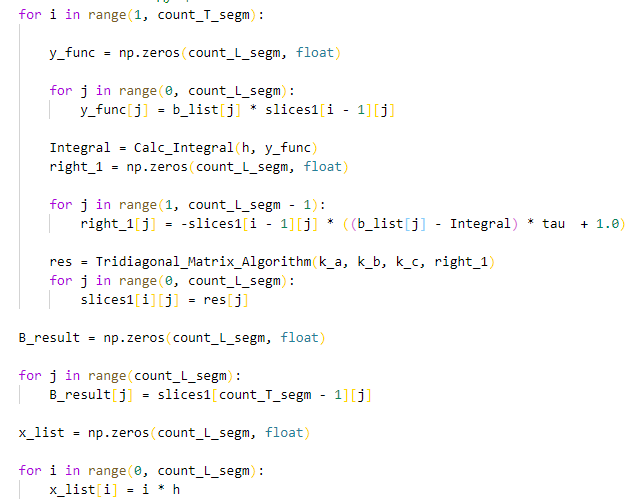
Данный метод подходит для решения СЛАУ, приведенных к трех-диагональному виду. Метод прогонки состоит из двух этапов: прямой прогонки и обратной прогонки. На первом этапе определяются прогоночные коэффициенты, а на втором – находят неизвестные.



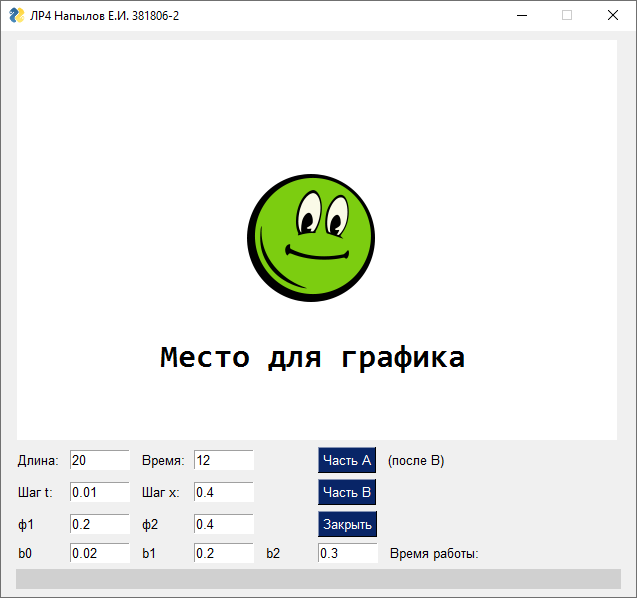
**Нулевой слой схемы**



**Остальные слои схемы**

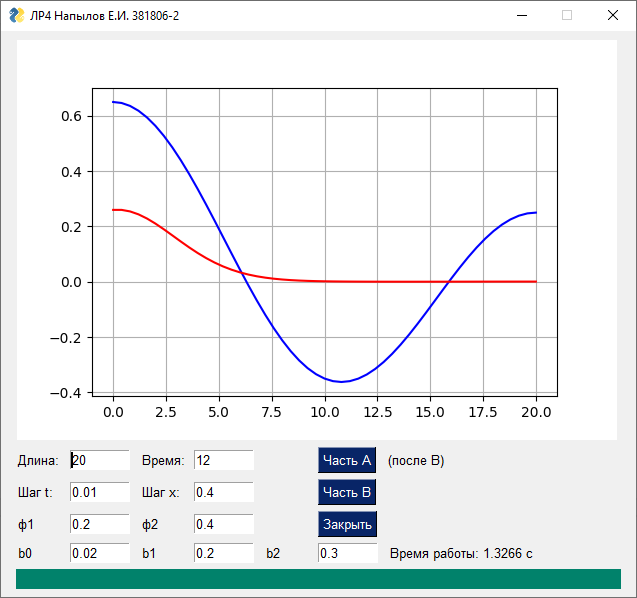


# Руководство к программе

Стартовое окно программы выглядит следующим образом

В нижней части находятся поля для изменения параметров. Реализована возможность изменения длины стержня, времени воздействия, шага по оси х и по времени, коэффициентов функций ф и b.

Для выполнения вычисления необходимо нажать кнопку «Часть В».



Во время вычислений происходит информирование пользователя о прогрессе. После завершения расчета строится график и выводится время работы программы.

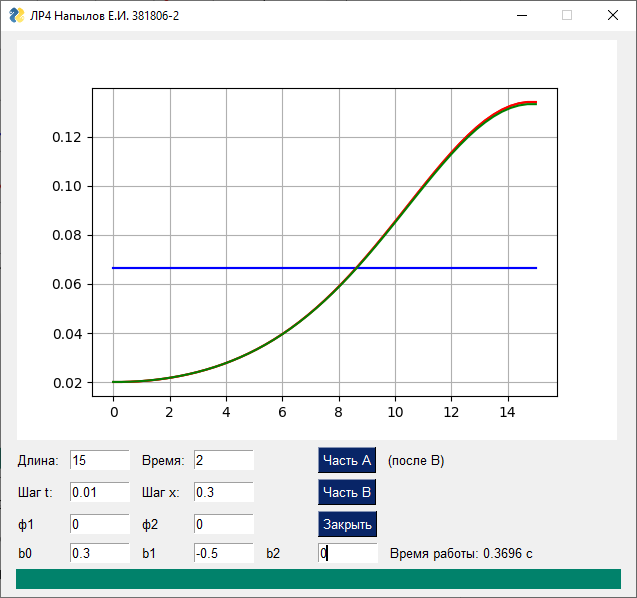
«Часть А» доступна только после выполнения предыдущей, т.к. требовалось чтобы график А лежал на верхнем слое.

Чтобы построить график А необходимо нажать на соответствующую кнопку. В идеале зеленый график должен лежать на красном.

Тестирование

Для тестирования я рассмотрел несколько вариантов входных данных.

**Тест 1.**



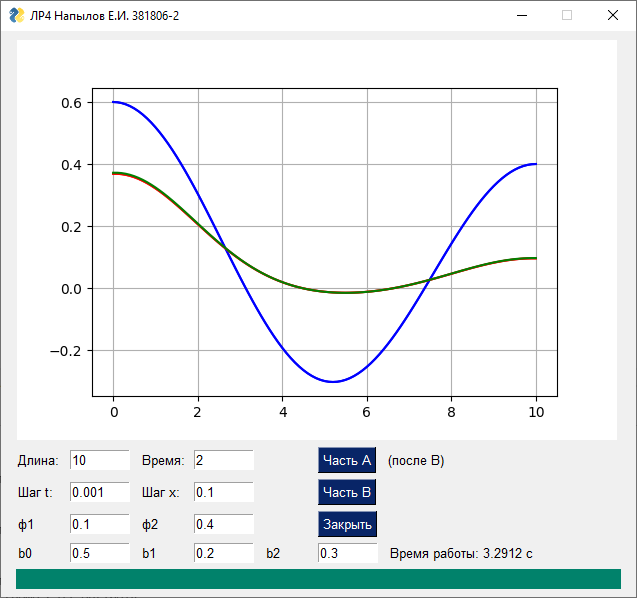
1. На концах отрезка касательные горизонтальные.

2. Площади верхней и нижней части равны.

3. При добавлении константы к функции b(x) результат не меняется

4. Зеленый график, полученный методом А лежит точно на красном графике В.

**Тест 2.**



Критерии из раздела «Методы самоконтроля части В» выполняются, следовательно, программа корректна.

# Заключение

В ходе работы я изучил алгоритмы численного решения начально-краевой задачи для интегро-дифференциального уравнения в частных производных. В качестве предмета исследования была выбрана задача о нагревании тонкого однородного стрежня с теплоизолированными концами. В теоретической части были выведены формулы для разностных схем и решения полученной СЛАУ методом прогонки.

После изучения теории была разработана программа с графическим интерфейсом. В программе была реализована возможность переключения между способами решения, а также возможность изменения начальных условий и коэффициентов начальных функций.

Программа была протестирована с использование различных параметров. Полученное решение удовлетворяет критериям из методического пособия для это лабораторной работы, следовательно, программа работает корректно.

# **Список литературы**

1. Эгамов А.И. Лабораторная работа «Численное решение задачи интегро-дифференциального уравнения в частных производных», 2019.
2. Волков Е.А., «Численные методы», 2008
3. Жидков Е.Н., «Вычислительная математика», 2013
4. Колдаев В.Д. «Численные методы и программирование», 2009
5. Лутц М., «Изучаем Python», том 1, 2019

# Приложение

**Код программы**

import PySimpleGUI as sg

import math

import matplotlib.pyplot as plt

import numpy as np

import time as timelib

sg.theme('Default')

layout = [  # graph.png

            [sg.Image(r'clear.png', key='plot\_image')],

            [sg.Text('Длина:', size=(5, 1)), sg.Input('20', size=(8, 1)), sg.Text('Время:', size=(5, 1)), sg.Input('12', size=(8, 1)), sg.Text('', size=(5, 1)),sg.Button('Часть А'), sg.Text('(после В)', size=(10, 1))],

            [sg.Text('Шаг t:', size=(5, 1)), sg.Input('0.01', size=(8, 1)), sg.Text('Шаг x:', size=(5, 1)), sg.Input('0.4', size=(8, 1)), sg.Text('', size=(5, 1)),sg.Button('Часть В')],

            [sg.Text('ф1', size=(5, 1)), sg.Input('0.2', size=(8, 1)), sg.Text('ф2', size=(5, 1)), sg.Input('0.4', size=(8, 1)), sg.Text('', size=(5, 1)),sg.Button('Закрыть')],

            [sg.Text('b0', size=(5, 1)), sg.Input('0.02', size=(8, 1)), sg.Text('b1', size=(5, 1)), sg.Input('0.2', size=(8, 1)), sg.Text('b2', size=(5, 1)), sg.Input('0.3', size=(8, 1)), sg.Text('Время работы:', size=(25, 1), key='work\_time')],

            [sg.ProgressBar(1000,orientation='h', size=(55, 20), key='br\_bar')]

        ]

window = sg.Window('ЛР4 Напылов Е.И. 381806-2', layout)

# values[0] - длина

# values[1] - время

# values[2] - шаг время

# values[3] - шаг х

# values[4] - ф1

# values[5] - ф2

# values[6] - б0

# values[7] - б1

# values[8] - б2

A\_result = []

B\_result = []

fig, ax = plt.subplots(figsize=(6, 4))

ax.grid()

ax.plot([0], [0])

fig.savefig('graph.png')

def Source\_Function\_Phi(x, l, f1, f2):

    return 1/l + f1 \* math.cos((math.pi\*x)/l) + f2 \* math.cos(2\*(math.pi\*x)/l)

def Source\_Function\_B(x, l, b0, b1, b2):

    return b0 + b1 \* math.cos((math.pi\*x)/l) + b2 \* math.cos(2\*(math.pi\*x)/l)

x\_list = []

phi\_list = []

def Tridiagonal\_Matrix\_Algorithm(a, b, c, f):

    size = len(f)

    A = np.zeros(size, float)

    B = np.zeros(size, float)

    x = np.zeros(size, float)

    A[0] = -c[0]/b[0]

    B[0] = f[0]/b[0]

    for i in range(1, size):

        A[i] = -c[i] / (a[i] \* A[i - 1] + b[i])

        B[i] = (f[i] - a[i] \* B[i - 1]) / (a[i] \* A[i - 1] + b[i])

    x[size-1] = B[size - 1]

    i = size - 2

    while (i > -1):

        x[i] = (A[i] \* x[i + 1] + B[i])

        i -= 1

    return x

def Calc\_Integral(h, f):

    res = (f[0] + f[len(f) - 1])

    for i in range(1, len(f) - 1, 2):

        res += (4 \* f[i] + 2 \* f[i + 1])

    return h \* res / 3

view\_mode = False

while True:

    event, values = window.Read(timeout = 100)

    # print(values)

    if event in (None, 'Закрыть'):

        # print('close')

        break

    if event in (None, 'Часть А'):

        # print('part A')

        if(view\_mode == 0):

            ax.plot(x\_list, phi\_list, 'b')

            ax.plot(x\_list, B\_result, 'r')

            ax.plot(x\_list, A\_result, 'g') # linewidth=5.0

            fig.savefig('graph.png')

            window['plot\_image'].update(r'graph.png')

            view\_mode = 1

        else:

            # print('part A else')

            fig, ax = plt.subplots(figsize=(6, 4))

            ax.grid()

            ax.plot(x\_list, phi\_list, 'b')

            ax.plot(x\_list, B\_result, 'r')

            fig.savefig('graph.png')

            window['plot\_image'].update(r'graph.png')

            view\_mode = 0

    if event in ('Часть В'):

        # print('part b')

        start\_time = timelib.time()

        fig, ax = plt.subplots(figsize=(6, 4))

        ax.grid()

        fig.savefig('graph.png')

        progress\_bar = window['br\_bar']

        window['plot\_image'].update(r'clear.png')

        try:

            l = float(values[0])

            time = float(values[1])

            tau = float(values[2])

            h = float(values[3])

            f1 = float(values[4])

            f2 = float(values[5])

            b0 = float(values[6])

            b1 = float(values[7])

            b2 = float(values[8])

        except:

            print('value parse error')

            continue

        if not tau / (h\*\*2) < 0.25:

            sg.popup\_error(f"tau/h^2={round(tau/(h\*\*2), 3)}")

            continue

        count\_L\_segm = int(l/h) + 1

        count\_T\_segm = int(time/tau) + 1

        slices1 = np.zeros((count\_T\_segm,count\_L\_segm), float)

        slices2 = np.zeros((count\_T\_segm,count\_L\_segm), float)

        prbar\_step = 1000/count\_T\_segm

        phi\_list = np.zeros(count\_L\_segm, float)

        b\_list = np.zeros(count\_L\_segm, float)

        for i in range(0, count\_L\_segm):

            phi\_list[i] = Source\_Function\_Phi(i\*h, l, f1, f2)

            b\_list[i] = Source\_Function\_B(i\*h, l, b0, b1, b2)

            slices1[0][i] = phi\_list[i]

            slices2[0][i] = phi\_list[i]

        k\_a = np.zeros(count\_L\_segm, float)

        k\_b = np.zeros(count\_L\_segm, float)

        k\_c = np.zeros(count\_L\_segm, float)

        k\_b[0] = 1.0

        k\_c[0] = -1.0

        for i in range(1, count\_L\_segm - 1):

            k\_a[i] = (tau / (h \* h))

            k\_b[i] = (-1 - 2\*tau / (h \* h))

            k\_c[i] = (tau / (h \* h))

        k\_a[count\_L\_segm - 1] = -1.0

        k\_b[count\_L\_segm - 1] = 1.0

        for i in range(1, count\_T\_segm):

            y\_func = np.zeros(count\_L\_segm, float)

            for j in range(0, count\_L\_segm):

                y\_func[j] = b\_list[j] \* slices1[i - 1][j]

            I = Calc\_Integral(h, y\_func)

            f = np.zeros(count\_L\_segm, float)

            right\_2 = np.zeros(count\_L\_segm, float)

            for j in range(1, count\_L\_segm - 1):

                f[j] = -slices1[i - 1][j] \* ((b\_list[j] - I) \* tau  + 1.0)

                right\_2[j] = -slices2[i - 1][j] \* (b\_list[j] \* tau + 1.0)

            res = Tridiagonal\_Matrix\_Algorithm(k\_a, k\_b, k\_c, f)

            for j in range(0, count\_L\_segm):

                slices1[i][j] = res[j]

            res2 = Tridiagonal\_Matrix\_Algorithm(k\_a, k\_b, k\_c, right\_2)

            for j in range(0, count\_L\_segm):

                slices2[i][j] = res2[j]

            progress\_bar.UpdateBar(i \* prbar\_step)

        I = Calc\_Integral(h, slices2[count\_T\_segm - 1])

        B\_result = np.zeros(count\_L\_segm, float)

        A\_result = np.zeros(count\_L\_segm, float)

        for j in range(count\_L\_segm):

            A\_result[j] = slices2[count\_T\_segm - 1][j] / I

            B\_result[j] = slices1[count\_T\_segm - 1][j]

        x\_list = np.zeros(count\_L\_segm, float)

        for i in range(0, count\_L\_segm):

            x\_list[i] = i \* h

        ax.plot(x\_list, phi\_list, 'b')

        ax.plot(x\_list, slices1[count\_T\_segm - 1], 'r')

        fig.savefig('graph.png')

        window['plot\_image'].update(r'graph.png')

        progress\_bar.UpdateBar(1000)

        view\_mode = 0

        end\_time = timelib.time()

        window['work\_time'].update(f'Время работы: {round(end\_time - start\_time, 4)} с')