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

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 x функцию y(x,t). Функция должна быть решением уравнения:

$$y'_t(x,t) = a^2 y''_{xx}(x,t) + u(x,t)$$

Функция должна удовлетворять однородным условиям второго рода:

$$y'_x(0,t) = y'_x(l,t) = 0$$

Функция должна удовлетворять начальному условию:

$$y(x,0) = \varphi(x),$$

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

$$\int_0^l \varphi(x)dx = 1.$$

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

$$u(x,t) = b(x)y(x,t)$$

$$u(x,t) = b(x)y(x,t) - y(x,t)\int_0^l b(x)y(x,t)\,dx$$

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 $\varphi(x)$:

$$\varphi(x) = \frac{1}{l} + \varphi_1 \cos\frac{\pi x}{l} + \varphi_2 \cos\frac{2\pi x}{l}.$$

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

$$I_j = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_3 + \cdots + 2y_{K-2} + 4y_{K-1} + y_K),$$

Теперь необходимо составить неявную разностную схему с погрешностью $O(\tau + h^2)$:

$$\frac{y_k^{n+1} - y_k^n}{\tau} = \frac{y_{k+1}^{n+1} - 2y_k^{n+1} + y_{k-1}^{n+1}}{h^2} + u_k^n \;. *$$

Схема является устойчивой при $\frac{\tau}{h^2} < \frac{1}{4}$.

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

$$\frac{y_1^{n+1} - y_0^{n+1}}{h} = \frac{y_K^{n+1} - y_{K-1}^{n+1}}{h} = 0; \qquad \frac{\partial^2 y}{\partial x^2} = \frac{\partial y}{\partial \tau} - u(x,\tau)$$

Теперь подставим вторую производную в выражение:

$$\frac{y_1 - y_0}{h} = \frac{\partial y(o,\tau)}{\partial x} + \frac{h}{2}\left(\frac{\partial^2 y}{\partial x^2}\right)_{x=0} + O(h^2)\;; \qquad \frac{y_1^{n+1} - y_0^{n+1}}{h} - \frac{h}{2}\frac{y_0^{n+1} - y_0^n}{\tau} + \frac{h}{2}u_0^n = 0 \;. *$$

$$\frac{y_K^{n+1} - y_{K-1}^{n+1}}{h} + \frac{h}{2}\frac{y_K^{n+1} - y_K^n}{\tau} - \frac{h}{2}u_0^n = 0 \;. *$$

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

Задание А:

$$\begin{cases} (2\tau + h^2)y_0^{n+1} + (-2\tau)y_1^{n+1} = h^2 y_k^n(1+\tau b_k) \\ (-\tau)y_{k-1}^{n+1} + (2\tau + h^2)y_k^{n+1} + (-\tau)y_{k+1}^{n+1} = h^2 y_k^n(1+\tau b_k), \\ (-2\tau)y_{K-1}^{n+1} + (2\tau + h^2)y_K^{n+1} = h^2 y_k^n(1+\tau b_k) \end{cases}$$

Задание В:

$$\begin{cases} (2\tau + h^2)y_0^{n+1} + (-2\tau)y_1^{n+1} = h^2 y_k^n\big(1+\tau(b_k - I_n)\big) \\ (-\tau)y_{k-1}^{n+1} + (2\tau + h^2)y_k^{n+1} + (-\tau)y_{k+1}^{n+1} = h^2 y_k^n\big(1+\tau(b_k - I_n)\big), \\ (-2\tau)y_{K-1}^{n+1} + (2\tau + h^2)y_K^{n+1} = h^2 y_k^n\big(1+\tau(b_k - I_n)\big) \end{cases}$$

Для удобства перепишем систему в другом виде

$A_k y_{k-1}^{n+1} + B_k y_k^{n+1} + C_k y_{k+1}^{n+1} = F_k . y_k = \alpha_{k+1} y_{k+1} + \beta_{k+1}$

Выразим y_k и y_{k-1} через y_{k+1} и подставим в исходную систему:

$$(A_k \alpha_k \alpha_{k+1} + B_k \alpha_{k+1} + C_k) y_{k+1} + A_k \alpha_k \beta_{k+1} + A_k \beta_k + B_k \beta_{k+1} - F_k = 0\,,$$

данное условие независимо от y, если:

$$\begin{cases} A_k \alpha_k \alpha_{k+1} + B_k \alpha_{k+1} + C_k = 0 \\ A_k \alpha_k \beta_{k+1} + A_k \beta_k + B_k \beta_{k+1} - F_k = 0 \end{cases} \Rightarrow \begin{cases} \alpha_{k+1} = \frac{-C_k}{A_k \alpha_k + B_k} \\ \beta_{k+1} = \frac{F_k - A_k \beta_k}{A_k \alpha_k + B_k} \end{cases}$$

$$\begin{cases} \alpha_1 = \dfrac{-C_0}{B_0} \\ \beta_1 = \dfrac{F_0}{B_0} \end{cases} ; \qquad y_K = \frac{F_K - A_K \beta_K}{B_K + A_K \alpha_k}$$

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 A и B.

```
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
```

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.

```
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
```

Нулевой слой схемы

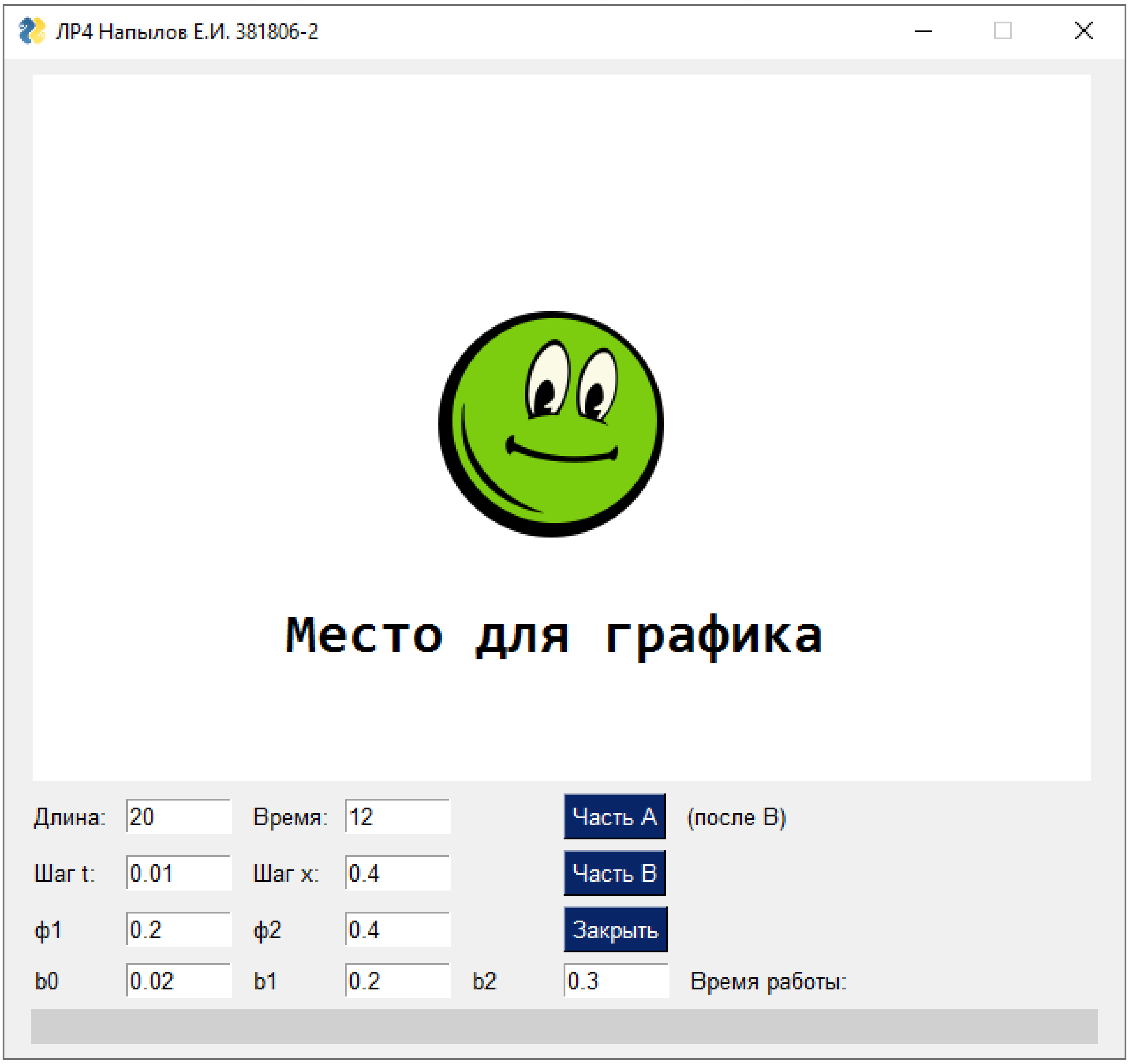
```
for i in range(0, count_L_segm):
    phi_list[i] = Source_Function_Phi(i*h, l, f1, f2)
    b_list[i] = Source_Function_B(i*h, l, b0, b1, b2)

    slices1[0][i] = phi_list[i]
```

Остальные слои схемы

```
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]

    Integral = Calc_Integral(h, y_func)
    right_1 = np.zeros(count_L_segm, float)

    for j in range(1, count_L_segm - 1):
        right_1[j] = -slices1[i - 1][j] * ((b_list[j] - Integral) * tau  + 1.0)

    res = Tridiagonal_Matrix_Algorithm(k_a, k_b, k_c, right_1)
    for j in range(0, count_L_segm):
        slices1[i][j] = res[j]

B_result = np.zeros(count_L_segm, float)

for j in range(count_L_segm):
    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
```

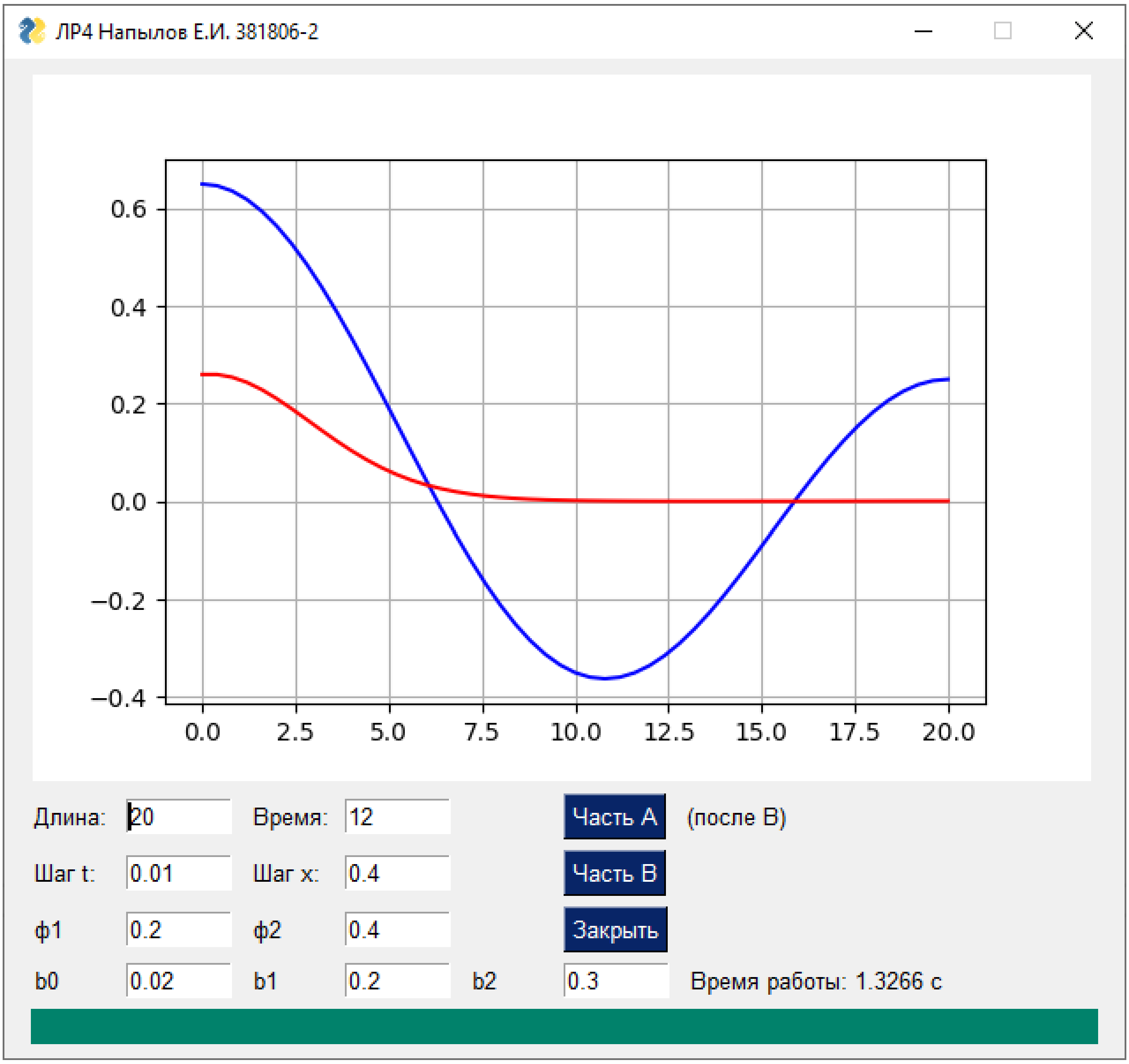
Руководство к программе

Стартовое окно программы выглядит следующим образом



В нижней части находятся поля для изменения параметров. Реализована возможность изменения длины стержня, времени воздействия, шага по оси x 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.

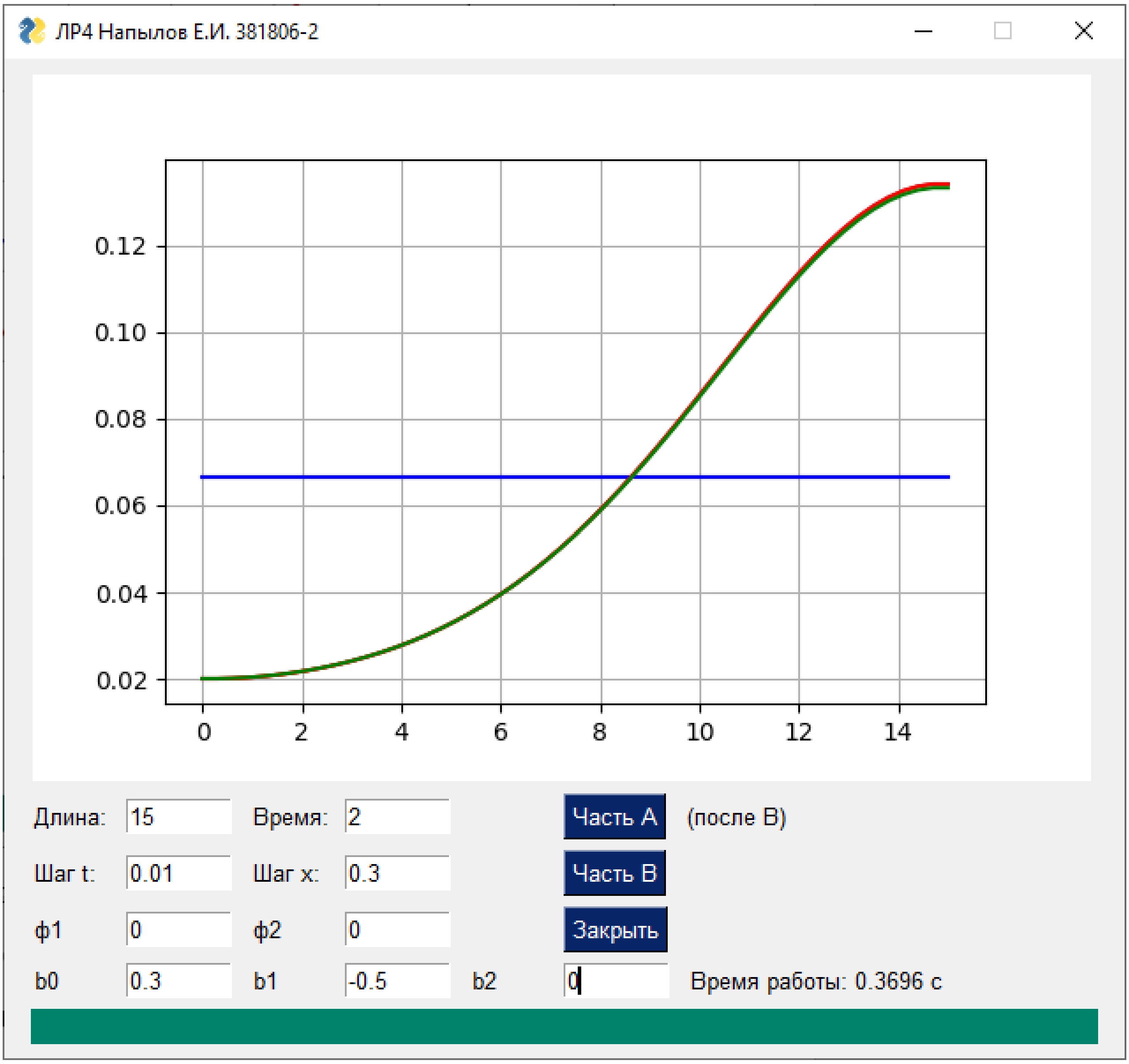
«Часть А» доступна только после выполнения предыдущей, т.к. требовалось чтобы график А лежал на верхнем слое.

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.



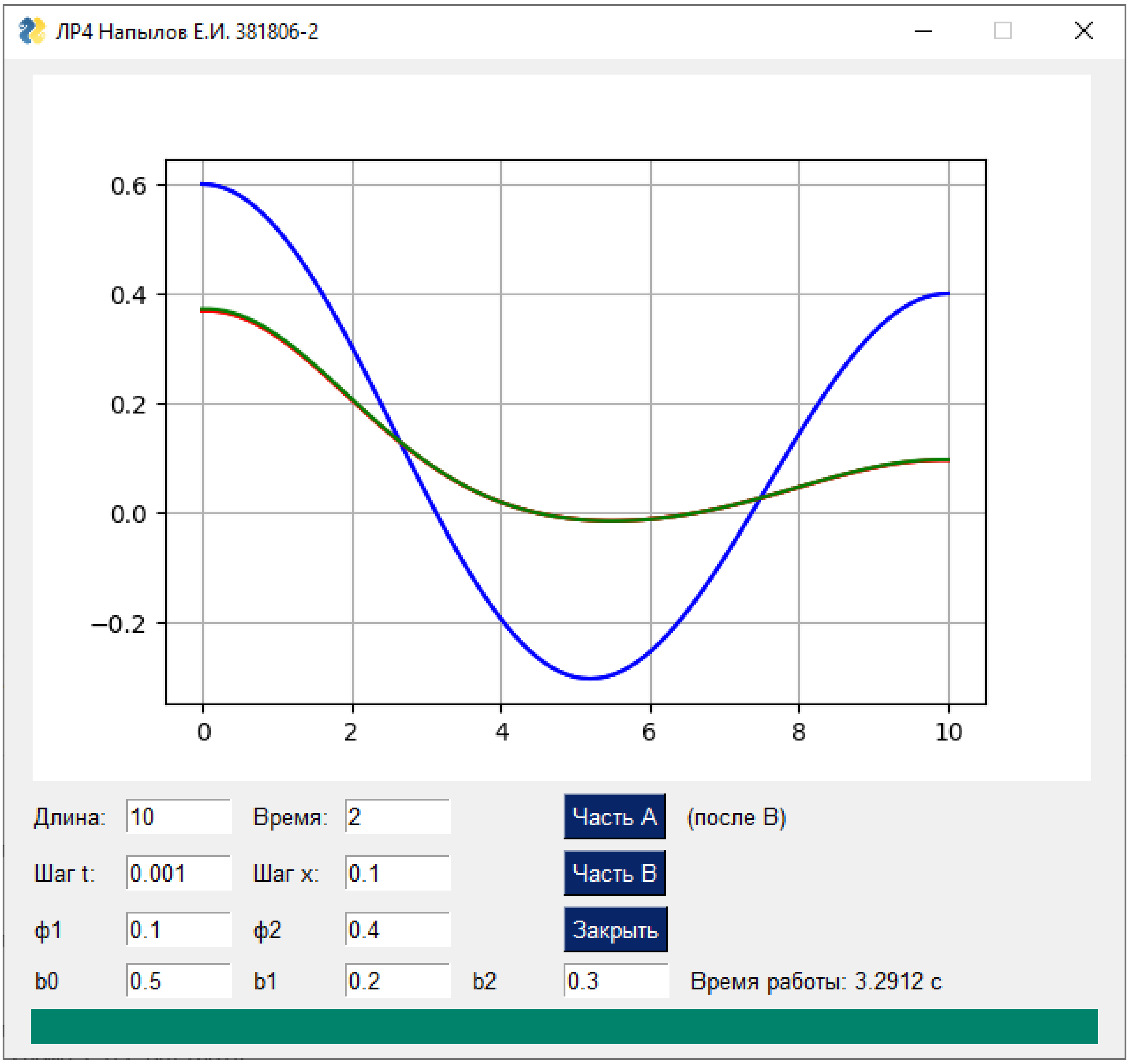
1. На концах отрезка касательные горизонтальные.

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 A'), sg.Text('(после B)', size=(10, 1))],
            [sg.Text('Шаг t:', size=(5, 1)), sg.Input('0.01', size=(8, 1)), sg.Te
xt('Шаг x:', size=(5, 1)), sg.Input('0.4', size=(8, 1)), sg.Text('', size=(5, 1))
,sg.Button('Часть B')],
            [sg.Text('ф1', size=(5, 1)), sg.Input('0.2', size=(8, 1)), sg.Text('ф
2', size=(5, 1)), sg.Input('0.4', size=(8, 1)), sg.Text('', size=(5, 1)),sg.Butto
n('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.I
nput('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 x
# 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])
```

```python
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 A'):
        # 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 B'):
        # 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)} c')
```