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

ОТЧЕТ

по лабораторной работе
на тему:

«Численное решение начально-краевой задачи для интегро-дифференциального уравнения в частных производных»

Выполнил: студент группы 381706-2
Синицина Мария Сергеевна

____
подпись

Преподаватель:
Ассистент кафедры дифференциальных уравнений, математического и численного анализа ИИТММ
Морозов Кирилл Евгеньевич

____
Подпись

Нижний Новгород
2020

Содержание

Введение

Под дифференциальным уравнением в частных производных понимается уравнение для функции двух или большего числа переменных, содержащее хотя бы одну частную производную этой функции. При этом сама функция и независимые переменные могут и не входить в уравнение явным образом.

Любое дифференциальное уравнение в частных производных имеет бесконечное множество решений. Наибольший интерес представляют решения, удовлетворяющие дополнительным условиям. Эти условия называются краевыми условиями и заключаются в указании поведения решения на некоторой граничной линии (поверхности) или в ее непосредственной окрестности. С этой точки зрения начальные условия представляют собой краевые условия во времени. Краевые условия используются для выбора частного решения из бесконечного множества решений. Практически любая задача, описывающая физический процесс и сформулированная в терминах дифференциальных уравнений в частных производных, включают в себя краевые условия.

Постановка задачи

Описание управляемого процесса

Рассмотрим в качестве примера управляемый процесс нагревания однородного стержня длины l с теплоизолированными концами.

Задача: на множестве $Q = [0, l] \times [0, T], l > 0, T > 0$ найти непрерывно дифференцируемую по t и дважды непрерывно дифференцируемую по x функцию $y(x, t)$ – температуру стержня, являющуюся решением уравнения

$$y'_t(x,t) = a^2 y''_{xx}(x,t) + u(x,t) \tag{1}$$

и удовлетворяющую однородным граничным условиям второго рода

$$y'_x(0,t) = y'_x(l,t) = 0 \tag{2}$$

и начальному условию

$$y(x,0) = \varphi(x), \tag{3}$$

где a – константа, $\varphi(x) > 0$ – дважды непрерывно дифференцируемая на отрезке функция, задающая начальное распределение температуры и удовлетворяющая условиям согласования (3) и условию

$$\int_0^l \varphi(x)\, dx = 1\,, \tag{4}$$

непрерывная функция $u(x, t)$ – управление с обратной связью, представляющаяся в виде

$$u(x,t) = b(x) y(x,t) \tag{5}$$

Или

$$u(x,t) = b(x)y(x,t) - y(x,t)\int_0^l b(x)y(x,t)\, dx\,, \tag{6}$$

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

Задача

Возьмем вместо начальной функции $\varphi(x)$ следующую функцию:

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

а вместо функции $b(x)$:

$$b(x) = b_0 + b_1 \cos\frac{\pi x}{l} + b_2 \cos\frac{2\pi x}{l}$$

Для решения задачи нам необходимо составить неявную разностную схему.

Определим нулевой слой будущей разностной схемы из (3). Заполним его значениями, которые получаются из функции $\varphi(x)$.

Для заполнения последующих слоев нам необходимо посчитать интеграл в точке. Перед вычислением каждого следующего слоя находим интеграл в (6) для значений последнего известного j слоя по формуле Симпсона:

$$I_j = \frac{h}{6}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_3 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n),$$

$n = \frac{l}{h}$ – количество шагов по x, предполагается чётным.

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

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

И составим трехточечные разностные производные первого порядка для краевых условий с погрешностью второго порядка. В виде разностных производных краевые условия выглядят следующим образом:

$$\frac{y_1^{n+1} - y_0^{n+1}}{h} = \frac{y_K^{n+1} - y_{K-1}^{n+1}}{h} = 0$$

Уравнение (1) преобразуем к виду:

$$\frac{\partial^2 y}{\partial x^2} = \frac{\partial y}{\partial \tau} - u(x,\tau)$$

Подставив вторую производную в это выражение, получим два уравнения для правой и левой границы. Эти два уравнения и неявная разностная схема составляют систему линейных уравнений, которую мы преобразовываем к трехдиагональной и решаем методом прогонки.

Требования к графическому выводу

- На одном и том же рисунке вывести оси координат, график функции $\varphi(x)$- синим цветом; график функции $y(x,T)$ - красным цветом.

- Учесть в оконном меню программы возможность изменения:

1. длины стержня l; времени T
2. шага h в разностной схеме по координате x;
3. шага τ в разностной схеме по координате t;
4. константы $b0$, $b1$, $b2$, $\varphi1, \varphi2$.

- Вывести на экран время выполнения данной работы

- Вывести полученный график следующей функции на экран зеленым цветом:

$$\frac{w(x,T)}{\int_0^l w(x,T)dx}$$

- Поскольку красный и зеленый график должны “совпадать”, зеленый график выводится на экран только при дополнительном нажатии специальной кнопки на форме.

Описание методов

Из формы считываются следующие константы:

- _len – длина стержня
- time – время наблюдения
- delta_x – количество шагов по x
- delta_t – количество шагов по времени
- double b0, b1, b2 – параметры управляющей функции b(x)
- double f1, f2 – параметры функции $\varphi(x)$

Основные функции:

- Simpson_integration(h, func) – функция, рассчитывающая интеграл по формуле Симпсона: $I_j = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_3 + \cdots + 2y_{K-2} + 4y_{K-1} + y_K)$,

  $K = \frac{l}{h}$ – количество шагов по x, предполагается чётным.

- sweep_method(a, b, c, func, count) – функция, решающая систему методом прогонки

  Система уравнений Ay=F равносильна соотношению $A_i y_{i-1} + B_i y_i + C_i y_{i+1} = F_i$

  Метод прогонки основывается на предположении, что искомые неизвестные связаны рекуррентным соотношением:

  $$y_k = \alpha_{k+1} y_{k+1} + \beta_{k+1}, \; k = \overline{K-1, 0} \qquad (*)$$

  Выразив y_k и y_{k-1} через y_{k+1} и подставив в исходный вид системы, получаем:

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

  где F_k — правая часть k-го уравнения.

  Это соотношение будет выполняться независимо от y в случае

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

  Так как $A_0 = 0$, $\begin{cases} \alpha_1 = \frac{-C_0}{B_0} \\ \beta_1 = \frac{F_0}{B_0} \end{cases}$

  Теперь можно найти все прогоночные коэффициенты.

  Последняя компонента решения:

  $$y_K = \frac{F_K - A_K \beta_K}{B_K + A_K \alpha_k}$$

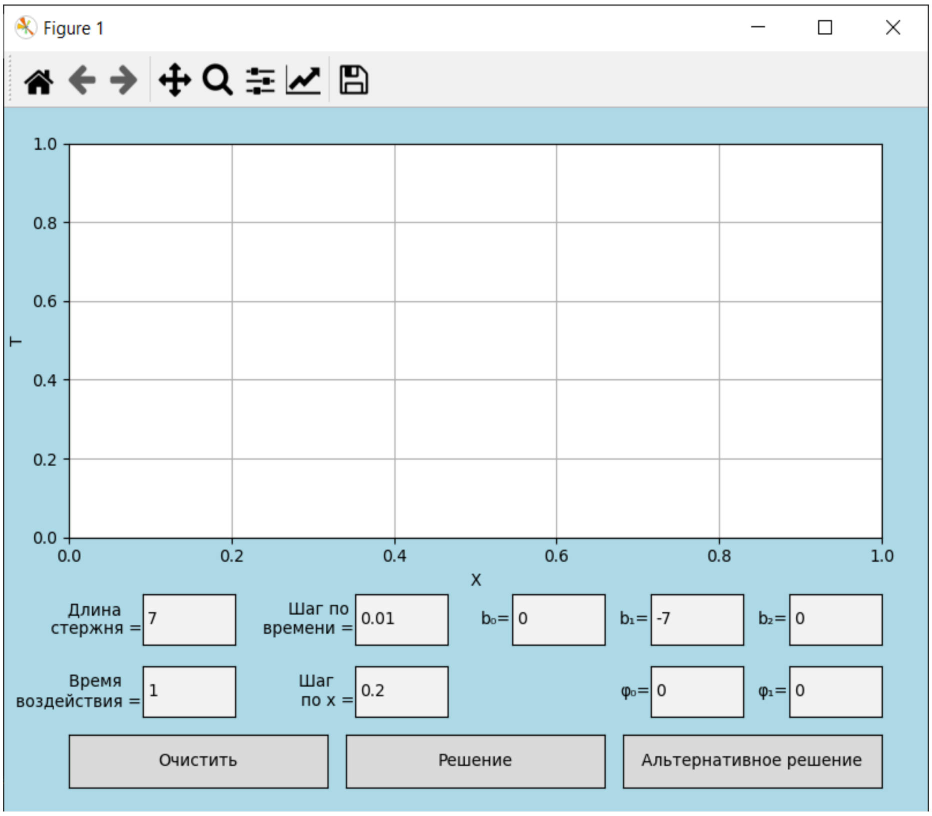
  Остальные находим из (*).

- solution(graph_axes) - функция, выполняющая все основные операции для решения и рисующая графики функций $\varphi(x)$ и $y(x,T)$.

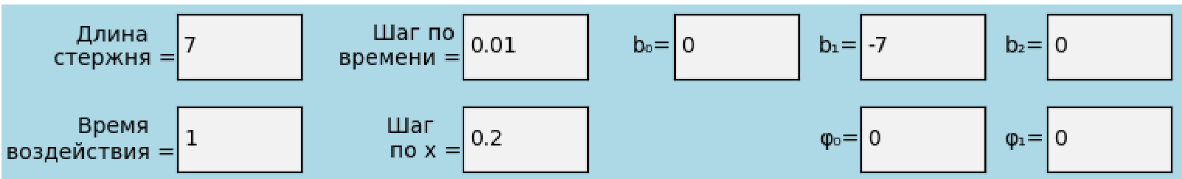
Реализация программы

В ходе работы была реализована программа, решающая предложенную систему

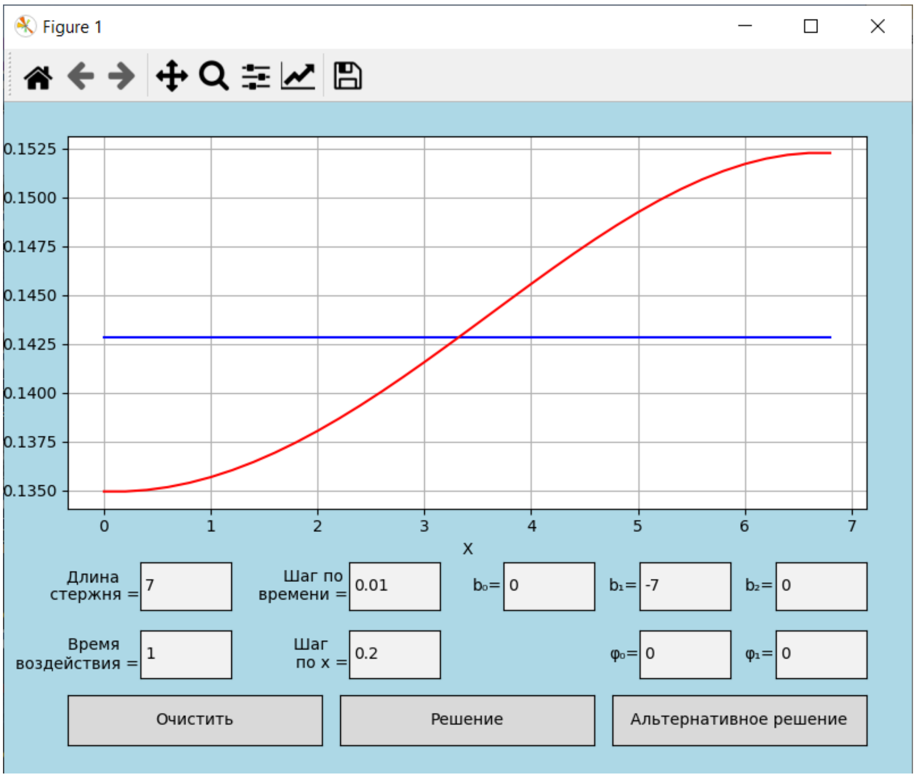
После запуска программы пользователю предлагается ввести длину стержня, время изменения температуры, количество точек, в которых производятся расчеты, а также параметры двух функций. Внутри полей предусмотрен ввод целочисленных значений и цифр, запятой (если пользователь пытается ввести не числа, то в консоль выведется предупреждение и будут взяты значения по умолчанию), а также удаление символов.



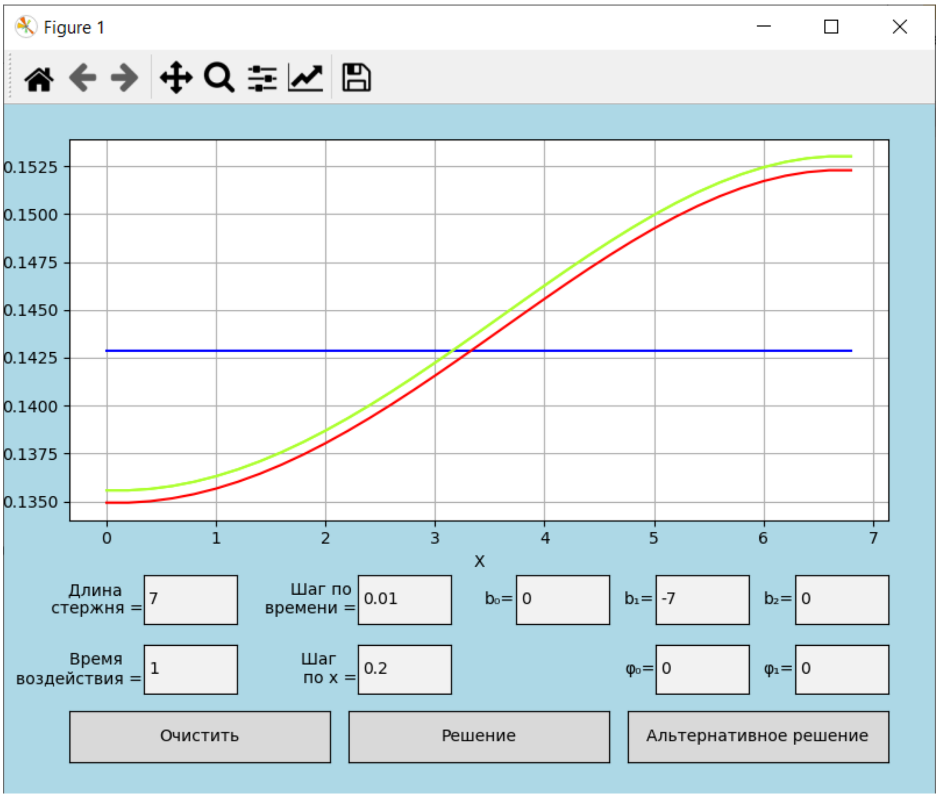
В начале работы программы пользователь может не вводить вручную параметры, а использовать значения по умолчанию: шаг длина стержня =7, время воздействия= 1, шаг по x=0.2, шаг по времени= 0.01, а также параметры двух функций.



Далее, нажав на кнопку «решение», строятся график функций $\varphi(x)$- синим цветом и график функции $y(x,T)$ - красным цветом.



Далее, нажав на кнопку «Альтернативное решение», строится зеленым цветом график функции $\frac{w(x,T)}{\int_0^l w(x,T)dx}$



При нажатии кнопки «Очистить» с формы удаляются все графики.

Заключение

В процессе выполнения лабораторной работы была решена начально- краевая задача для интегро -дифференциального уравнения нагревания стержня. Была написана программа на языке python с дружественным интерфейсом, которая выводит графическую информацию на экран. В работе использовалась библиотека языка python matpotlib, которая очень удобна в использовании для визуализации 2D графиков.

Литература

1. «Численные методы анализа» Демидович Б.П., Марон И.А., Шувалова Э.З.
2. «Численные методы» Самарский А.А., Гулин А.В.
3. Учебно-методическое пособие - Лабораторная работа «Численное ре-шение начально-краевой задачи для интегро-дифференциального урав-нения в частных производных».

Листинг программы

```
import matplotlib.pyplot as plt
from matplotlib.widgets import Button, TextBox
from math import cos
from tqdm import tqdm
from time import *
plt.rcParams.update({'figure.figsize': '8, 6', "figure.facecolor": 'lightblue',
'axes.edgecolor': 'black'})
#import progressbar

pi = 3.1415926535

def func(x, l, f1, f2):
    return 1/l + f1 * cos((pi*x)/l) + f2 * cos(2*(pi*x)/l)

def bfunc(x, l, b0, b1, b2):
    return b0 + b1 * cos((pi*x)/l) + b2 * cos(2*(pi*x)/l)

def Simpson_integration(h, fun): #Функция численного интегрирования
    res = (h/3)*(fun[0] + fun[len(fun) - 1])
    for i in range(1, len(fun) - 1, 2):
        res += (h/3)*(4*fun[i] + 2*fun[i + 1])
    return res

def sweep_method(a, b, c, func, count):#Метод прогонки
    A = []
    B = []
    res = [0] * count
    A.append(-c[0]/b[0])
    B.append(func[0]/b[0])
    for i in range(1, count):
        A.append(-c[i] / (a[i] * A[i - 1] + b[i]))
        B.append((func[i] - a[i] * B[i - 1]) / (a[i] * A[i - 1] + b[i]))
    res[count-1] = B[count - 1]
    for i in range(count - 2, -1, -1):
        res[i] = (A[i] * res[i + 1] + B[i])
    return res

def onButtonAddClicked(event):
    global graph_axes
    pbar = tqdm(total=100)
    for i in range(10):
        sleep(0.1)
        pbar.update(10)
    pbar.close()
    #tim1 = time.time()
    solution(graph_axes)
    #tim2 = time.time()
    #time_ = tim2 - tim1
    #print("Время выполнения: ", time_)

def onButtonCreateCliked(event):
    global graph_axes
    pbar = tqdm(total=100)
    for i in range(10):
        sleep(0.1)
        pbar.update(10)
    pbar.close()
    alternativeSolution(graph_axes)
```

```
def onButtonClearClicked(event):
    global graph_axes
    graph_axes.clear()
    graph_axes.grid()
    plt.draw()

def alternativeSolution(graph_axes):
    global x_val, resA, func_val
    graph_axes.plot(x_val, resA, 'greenyellow')
    plt.draw()

def solution(graph_axes):
    global x_val, resA, func_val
    func_val = []
    bfunc_val = []
    slices1 = [[]]
    slices2 = [[]]
    count_N = int(_len/delta_x)
    count_T = int(time/delta_t)
    # Вычисление значений функции и заполнение первого слоя сетки
    for i in range(0, count_N):
        func_val.append(func(i*delta_x, _len, f1, f2))
        bfunc_val.append(bfunc(i*delta_x, _len, b0, b1, b2))
        slices1[0].append(func_val[i])
        slices2[0].append(func_val[i])
    # Заполнение матрицы коэффициентов для метода прогонки
    coeff_a = [0.0]
    coeff_b = [1.0]
    coeff_c = [-1.0]
    for i in range(1, count_N - 1):
        coeff_a.append(delta_t / (delta_x * delta_x))
        coeff_b.append(-1 - 2*delta_t / (delta_x * delta_x))
        coeff_c.append(delta_t / (delta_x * delta_x))
    coeff_a.append(-1.0)
    coeff_b.append(1.0)
    coeff_c.append(0.0)
    #Вычисление последующих слоев сетки
    for i in range(1, count_T):
        I = Simpson_integration(delta_x, bfunc_val)
        fu = [0]
        fu2 = [0]
        slices1.append([])
        slices2.append([])
        #Вычисляем правую часть системы для прогонки
        for j in range(1, count_N - 1):
            fu.append(-slices1[i - 1][j] * ((bfunc_val[j] - I) * delta_t * delta_t  +
1.0))
            fu2.append(-slices2[i - 1][j] * (bfunc_val[j] * delta_t * delta_t + 1.0))
        fu.append(0)
        fu2.append(0)
        #Метод прогонки для системы из B
        res = sweep_method(coeff_a, coeff_b, coeff_c, fu, count_N)
        for j in range(0, count_N):
            slices1[i].append(res[j])
        #Метод прогонки для системы из A
        res2 = sweep_method(coeff_a, coeff_b, coeff_c, fu2, count_N)
        for j in range(0, count_N):
            slices2[i].append(res2[j])
    I = Simpson_integration(delta_x, slices2[count_T - 1])
    resB = []
    resA = []
    for j in range(0, count_N):
        resA.append(slices2[count_T - 1][j] / I)
        resB.append(slices1[count_T - 1][j])
```

```
    x_val = []
    for i in range(0, count_N):
        x_val.append(i * delta_x)
    graph_axes.plot(x_val, func_val, 'b')
    graph_axes.plot(x_val, slices1[count_T - 1], 'r')
    plt.draw()

if __name__ == "__main__":
    global graph_axes, flag, b0, b1, b2, f0, f1, f2, delta_t, delta_x, time, _len
    global func_val, resA, resB, x_val, func_val, bfunc_val, slices1, slices2
    func_val = []
    resA = []
    resB = []
    x_val = []

    func_val = []
    bfunc_val = []
    slices1 = [[]]
    slices2 = [[]]

    fig, graph_axes = plt.subplots()
    graph_axes.grid()
    fig.subplots_adjust(left=0.07, right=0.95, top=0.95, bottom=0.4)
    graph_axes.set_xlabel('X')
    graph_axes.set_ylabel('T')

    def submitTime(text):
        global time
        try:
            time = float(text)
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для параметра 'time' были использованы значения по умолчанию = ",
time)

    def submit_Len(text):
        global _len
        try:
            _len = float(text)
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для шага '_len' были использованы значения по умолчанию = ", _len)

    def submitB0(text):
        global b0
        try:
            b0 = float(text)
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для начального 'b0' были использованы значения по умолчанию = ", b0)

    def submitB1(text):
        global b1
        try:
            b1 = float(text)
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для начального 'b1' были использованы значения по умолчанию = ", b1)

    def submitB2(text):
        global b2
        try:
            b2 = float(text)
            return b2
```

```
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для колличества точек 'b2' были использованы значения по умолчанию =
", b2)

    def submitF0(text):
        global f0
        try:
            #f0 = float(text)
            f0 = 1 / _len
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для параметра 'f0' были использованы значения по умолчанию = ", f0)

    def submitF1(text):
        global f1
        try:
            f1 = float(text)
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для параметра 'f1' были использованы значения по умолчанию = ", f1)

    def submitF2(text):
        global f2
        try:
            f2 = float(text)
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для параметра 'f2' были использованы значения по умолчанию = ", f2)

    def submitDeltaT(text):
        global delta_t
        try:
            delta_t = float(text)
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для параметра 'delta_t' были использованы значения по умолчанию = ",
delta_t)

    def submitDeltaX(text):
        global delta_x
        try:
            delta_x = float(text)
        except ValueError:
            print("Вы пытаетесь ввести не число")
            print("Для параметра 'delta_x' были использованы значения по умолчанию = ",
delta_x)

    axes_button_add = plt.axes([0.37, 0.05, 0.28, 0.075])
    button_add = Button(axes_button_add, 'Решение')
    button_add.on_clicked(onButtonAddClicked)

    axes_button_clear = plt.axes([0.07, 0.05, 0.28, 0.075])
    button_clear = Button(axes_button_clear, 'Очистить')
    button_clear.on_clicked(onButtonClearClicked)

    axes_button_create = plt.axes([0.67, 0.05, 0.28, 0.075])
    button_create = Button(axes_button_create, 'Альтернативное решение')
    button_create.on_clicked(onButtonCreateCliked)

    axbox = plt.axes([0.15, 0.25, 0.10, 0.07])
    _len_box = TextBox(axbox, 'Длина    \n стержня =', initial="7")
    _len = 7.
    _len_box.on_submit(submit_Len)
```

```
axbox = plt.axes([0.38, 0.25, 0.10, 0.07])
delta_t_box = TextBox(axbox, 'Шаг по \nвремени =', initial="0.01")
delta_t = 0.01
delta_t_box.on_submit(submitDeltaT)

axbox = plt.axes([0.55, 0.25, 0.10, 0.07])
bo_box = TextBox(axbox, 'b₀=', initial="0")
b0 = 0.
bo_box.on_submit(submitB0)

axbox = plt.axes([0.70, 0.25, 0.10, 0.07])
b1_box = TextBox(axbox, 'b₁=', initial= "-7")
b1 = -7
b1_box.on_submit(submitB1)

axbox = plt.axes([0.85, 0.25, 0.10, 0.07])
b2_box = TextBox(axbox, 'b₂=', initial="0")
b2 = 0.
b2_box.on_submit(submitB2)

axbox = plt.axes([0.15, 0.15, 0.10, 0.07])
time_box = TextBox(axbox, 'Время    \nвоздействия =', initial= "1")
time = 1.
time_box.on_submit(submitTime)

axbox = plt.axes([0.38, 0.15, 0.10, 0.07])
deltax_box = TextBox(axbox, 'Шаг    \nпо x =', initial= "0.2")
delta_x = 0.2
deltax_box.on_submit(submitDeltaX)

axbox = plt.axes([0.70, 0.15, 0.10, 0.07])
f1_box = TextBox(axbox, 'ϕ₀=', initial= "0")
f1 = 0.
f1_box.on_submit(submitF1)

axbox = plt.axes([0.85, 0.15, 0.10, 0.07])
f2_box = TextBox(axbox, 'ϕ₁=', initial= "0")
f2 = 0.
f2_box.on_submit(submitF2)

#bar = progressbar.ProgressBar(max_value=progressbar.UnknownLength)
#for i in range(20):
    # time.sleep(0.1)
    # bar.update(i)

plt.show()
```