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
Банденков Даниил Викторович

____

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 уравнение в частных производных может быть представлено в виде

$$F(x, y, \dots;\ u_x, u_y, \dots; u_{xx}, u_{xy}, \dots) = 0$$

где x,y,… – независимые переменные; u = u(x,y,…) – искомая функция;

Любое дифференциальное уравнение в частных производных имеет бесконечное множество решений. Наибольший интерес представляют решения, удовлетворяющие краевыми условиями, заключающимся в указании поведения решения на некоторой граничной линии (поверхности) или в ее непосредственной окрестности. 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.

Существует два вида методов решения уравнений математической физики:

1. аналитический, при котором результат выводится различными математическими преобразованиями;
2. численный, при котором полученный результат соответствует действительному с заданной точностью, но который требует много рутинных вычислений и поэтому выполним только при помощи вычислительной техники (ЭВМ).

Поскольку нахождение аналитического решения даже простого уравнения в сложной области не всегда возможно, то было разработано множество методов решения уравнений математической физики.

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.

Описание математической задачи

Для решения задачи нам необходимо составить неявную разностную схему.

Возьмем вместо функции *b(x)* :

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

а вместо начальной функции $\varphi(x)$ следующую функцию И заполним нулевой слой разностной схемы значениями, которые получаются из нее:

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

Для заполнения последующих слоев нам необходимо посчитать интеграл в точке. Перед вычислением каждого следующего слоя находим интеграл для значений последнего известного 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.

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

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

И составим трехточечные разностные производные для краевых условий в виде:

$$\frac{y_K^{\mathrm{j+1}} - y_{K-1}^{\mathrm{j+1}}}{h} = 0$$

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. Получаем СЛАУ вида:

$$\begin{cases} y_0^{\mathrm{j+1}} = 0 \\ y_{\mathrm{N}}^{\mathrm{j+1}} = y_{\mathrm{N-1}}^{\mathrm{j+1}} + \ \mathrm{ht_{j+1}} \\ \frac{\tau y_{\mathrm{i\text{-}1}}^{\mathrm{j+1}}}{h^2} \text{-}(1 + \frac{2\tau}{h^2}) y_{\mathrm{i}}^{\mathrm{j+1}} + \frac{\tau y_{\mathrm{i+1}}^{\mathrm{j+1}}}{h^2} = \text{-}(y_{\mathrm{i}}^{\mathrm{j}} + \tau \mathrm{x_i}) \end{cases}$$

Матрицу коэффициентов системы линейных уравнений мы преобразовываем к виду трехдиагональной,матрицы:

$$A = \begin{pmatrix} C_1 & B_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ A_2 & C_2 & B_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & A_3 & C_3 & B_3 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & A_{n-1} & C_{n-1} & B_{n-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & A_n & C_n \end{pmatrix}.$$

После чего решаем 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:

$$y_k = \alpha_{k+1} y_{k+1} + \beta_{k+1}, \ \ k = \overline{K-1, 0}$$

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

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

Отсюда

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

Так как $A_0 = 0$,

$$\begin{cases} \alpha_1 = \dfrac{-C_0}{B_0} \\ \beta_1 = \dfrac{F_0}{B_0} \end{cases}$$

Теперь можно найти все прогоночные коэффициенты.

Последняя компонента решения:

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

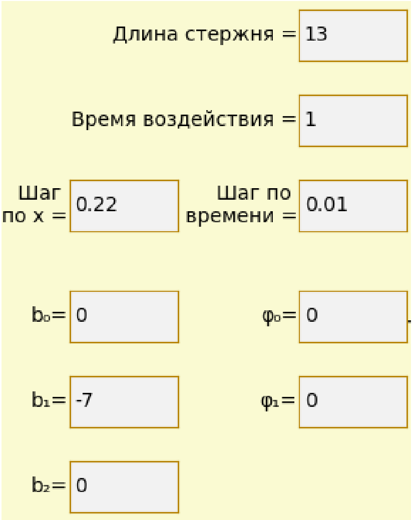
Остальные находим из рекуррентного соотношения.

Выбор языка

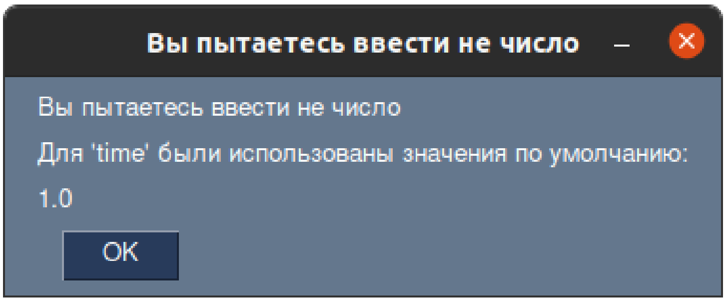
Для реализации данной программы был выбран язык программирования python, из-за огромного количества библиотек для визуализации графики и универсальности языка. В качестве gui используется стандартная форма matplotlib. Дополнительная библиотека PySimpleGUI упрощает работу пользователя с интерфейсом, например, позволяет создать строку прогресса, таблицу параметров и окона с ошибками. Коллекция команд pyplot, заставляет работать matplotlib как библиотеку matlab. Также благодаря возможностям данной библиотеки, можно сохранять полученные результаты менять масштаб и цвета графиков, а благодаря добавленным на формам кнопкам можно менять параметры, шаги, длину стержня, время воздействия, удалять полученные графики и добавлять новые.

Реализация программы

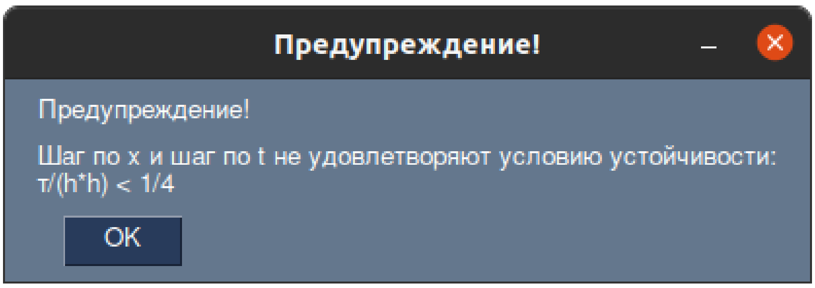
В начале работы пользователь может ввести значения или использовать значения по умолчанию для следующих полей: длина стержня=13, время воздействия =1, шаг по x= 0.22, шаг по времени = 0.01, параметры функций b0=0 , b1=-7 , b2=0, $\varphi 1 = 0, \varphi 2 = 0$.



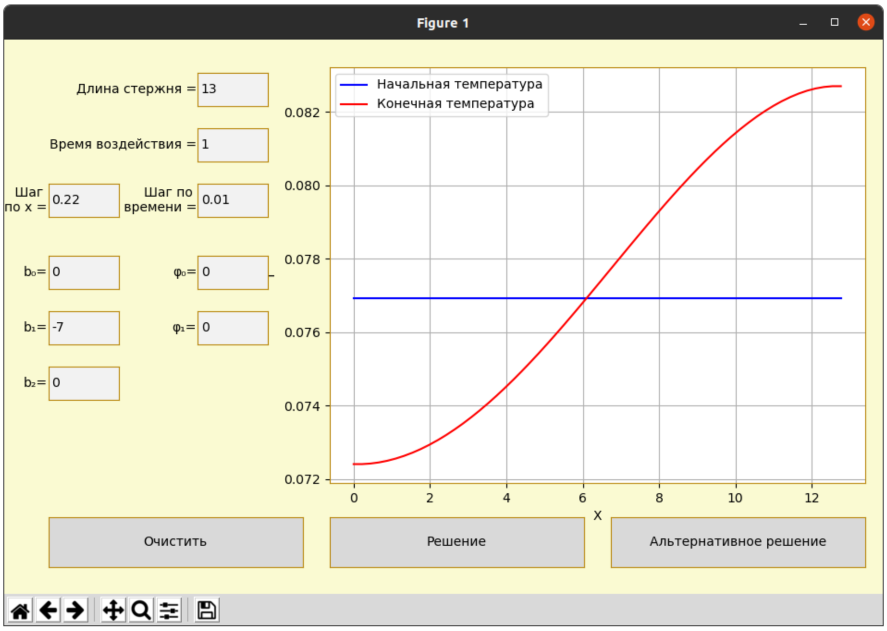
Если пользователь пытается ввести в поле не целочисленные или десятичные значения, то срабатывает предупреждение и используется значение по умолчанию для данного поля:



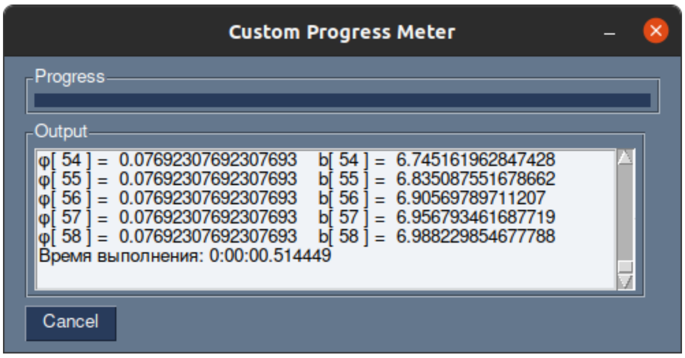
Если пользователь вводит значения в поля чисел шагов, нарушающие условие устойчивости, по появляется сообщение. При этом никакие графики не строятся, форма остается открытой и готовой к дальнейшей работе.



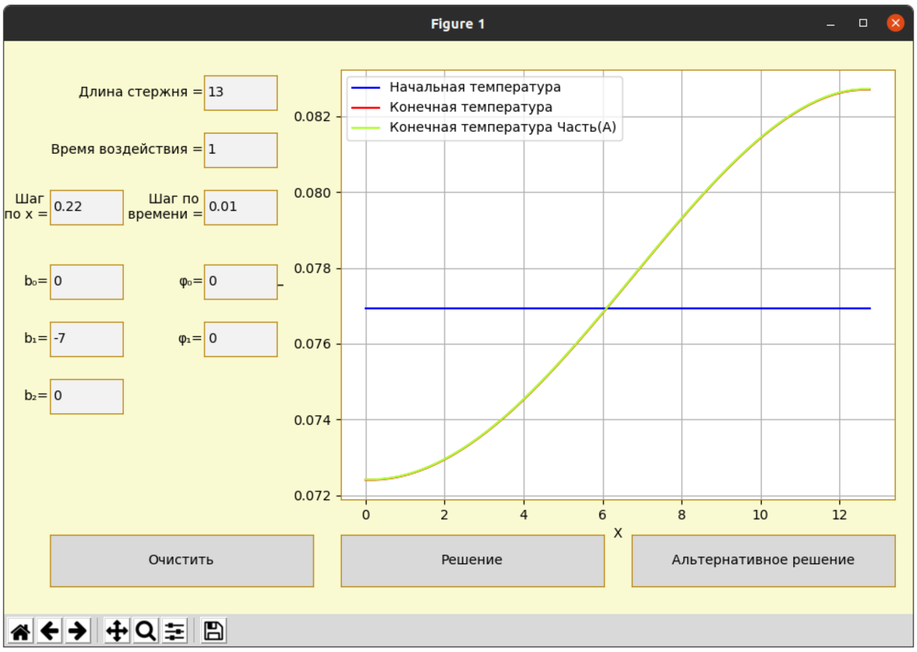
При нажатии на кнопку «Решение» строится два графика $\varphi(x)$- синим цветом; график функции $y(x,T\,)$ - красным цветом:



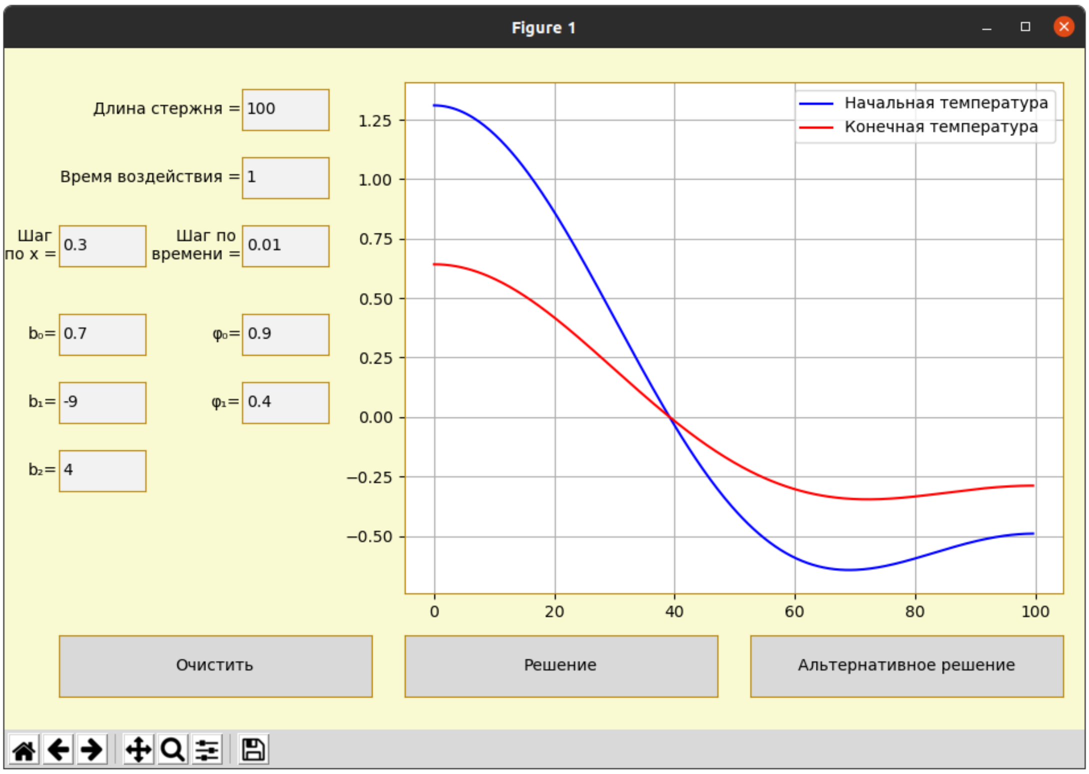
А также появляется дополнительное окно, содержащее строку прогресса и выводящее таблицей значения двух функций в точках:



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



При нажатии на кнопку «Очистить» графики удаляются с формы без ее закрытия. После этого можно ввести новые значения и строить графики



Заключение

В процессе выполнения лабораторной работы была решена начально- краевая задача для интегро-дифференциального уравнения нагревания стержня. Были получены практические навыки программирования.

Литература

1. «Численные методы» Самарский А.А., Гулин А.В.
2. Учебно-методическое пособие - Лабораторная работа «Численное ре-шение начально-краевой задачи для интегро-дифференциального урав-нения в частных производных».

Листинг

Ниже приведены основные функции работы программы (остальная программа в репозитории на GitHub):

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


#Метод прогонки
def tridiagAlg(a, b, c, func, count):
    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


#Обработчик события. Отрисовка графика, решения части Б
def onButtonAddClicked(event):
    global graph_axesm, delta_t, delta_x
    pogreshnostPython = 0.0000001
    k = delta_t / (delta_x * delta_x) + pogreshnostPython
    if (k < 0.25):
        solution(graph_axes)
    else:
        sg.Popup('Предупреждение!', 'Шаг по x и шаг по t не удовлетворяют условию
устойчивости:\nτ/(h*h) < 1/4')


#Обработчик события. Решения части А
def onButtonCreateCliked(event):
    global graph_axes
    alternativeSolution(graph_axes)

#Обработчик события. Отчистка формы графиков
def onButtonClearClicked(event):
    global graph_axes
    graph_axes.clear()
    graph_axes.grid()
    plt.draw()


#Вспомогательная функция,отрисовка графика (Часть А)
def alternativeSolution(graph_axes):
    global x_val, resA, func_val
    graph_axes.plot(x_val, resA, label = 'Конечная температура Часть(А)', color
='greenyellow')
    graph_axes.legend()
    plt.draw()
```

```
#Вспомогательная функция вызова строки состояния, и таблицы значений
def progressBarSolution():
    global resA, resB, func_val, bfunc_val
    start_time = datetime.now()
    progressbar = [[sg.ProgressBar(len(func_val), orientation='h', size=(41, 10),
key='progressbar')]]
    outputwin = [[sg.Output(size=(60,7))]]

    layout = [
        [sg.Frame('Progress',layout= progressbar)],
        [sg.Frame('Output', layout = outputwin)],
        [sg.Cancel()]
    ]

    window = sg.Window('Custom Progress Meter', layout)
    progress_bar = window['progressbar']

    event, values = window.read(timeout=10)
    for i,item in enumerate(func_val):
        print("φ[", i, "] = ", item, "   b[", i, "] = ", bfunc_val[i] )
        sleep(0.003)
        progress_bar.UpdateBar(i + 1)
    end_time = datetime.now()
    print('Время выполнения: {}'.format(end_time - start_time))
    if event == 'Cancel'  or event is None:
        window.close()



#Вспомогательная функция. Решение частей А и Б. Отрисовка графика (Часть Б)
def solution(graph_axes):
    global x_val, resA, resB, func_val, bfunc_val #глобальные переменные
    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 = integrate(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 = tridiagAlg(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 = tridiagAlg(coeff_a, coeff_b, coeff_c, fu2, count_N)
        for j in range(0, count_N):
            slices2[i].append(res2[j])
    I = integrate(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)
    #Отрисовка графиков, вызов меню строки состояния
    progressBarSolution()
    graph_axes.plot(x_val, func_val, label = 'Начальная температура', color ='b')
    graph_axes.plot(x_val, slices1[count_T - 1], label = 'Конечная температура', color
='r')
    graph_axes.legend()
    plt.draw()
```