МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. УЛЬЯНОВА (ЛЕНИНА)
Кафедра алгоритмической математики

КУРСОВАЯ РАБОТА

по дисциплине «Дифференциальные уравнения»

Тема: Использование численных методов решения дифференциальных уравнений для решения задачи диффузии в твердых телах

Студенты гр. 8382 _____ Кобенко В.П.
Черницын П.А.

Преподаватель _____ Павлов Д.А.

Санкт-Петербург
2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Кобенко В.П.
Студент Черницын П.А.
Группа 8382
Тема работы: Использование численных методов решения дифференциальных уравнений для решения задачи диффузии в твердых телах

Исходные данные:
Диффузия в твердых телах

Содержание пояснительной записки:
«Содержание», «Введение», «Уравнение диффузии», «FTCS - Dirichlet problem», «BTCS - Dirichlet problem», «BTCS-Neumann problem», «CN-Neumann problem», «Графический интерфейс», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 09.05.2021

Дата сдачи курсовой работы: 29.05.2021

Дата защиты курсовой работы: 29.05.2021

Студенты		Кобенко В.П. Черницын П.А.
Преподаватель		Павлов Д.А.

АННОТАЦИЯ

В курсовой работе рассмотрены различные способы решения физической задачи “Диффузия в твердых телах” с использованием численных методов решения дифференциальных уравнений. Для этого было выделено произвольное уравнение диффузии. Для решения данного уравнения были использованы методы: «FTCS - Dirichlet problem», «BTCS - Dirichlet problem», «BTCS-Neumann problem», «CN-Neumann problem». Результаты решения данного уравнения были представлены в виде графиков в графическом интерфейсе.

SUMMARY

In the course work, various ways of solving the physical problem “Diffusion in solids” are considered using numerical methods for solving differential equations. For this, an arbitrary diffusion equation was selected. To solve this equation, the following methods were used: “FTCS - Dirichlet problem”, “BTCS - Dirichlet problem”, “BTCS-Neumann problem”, “CN-Neumann problem”. The results of solving this equation were presented in the form of graphs in the graphical interface.

СОДЕРЖАНИЕ

Введение

Дифференциальное уравнение является одним из фундаментальных понятий математики, широко применяемое в различных областях современных наук. Оно также применимо в физических процессах, один из которых мы собираемся рассмотреть в данной курсовой работе. Диффузия в твердых телах может происходить годами без внешнего воздействия, поэтому целью данной курсовой работы является ознакомление с влиянием внешнего воздействия на процесс диффузии. С помощью численных методов решения дифференциальных уравнений можно выбрать время, которое будет подразумеваться влияниям внешней среды на процесс. В зависимости от этого мы получим различные результаты на графиках и проанализируем их.

Уравнение диффузии

Одномерное уравнение диффузии:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + \alpha u$$

,

где **u(x,t)** – это функция, зависящая от пространства и времени. Физическое объяснение функции **u** зависит от процесса диффузии. Например, **u** - это концентрация вещества, если уравнение диффузии моделирует перенос этого вещества путем диффузии. Процессы диффузии имеют особое значение на микроскопическом уровне в биологии, например диффузионный перенос определенных типов ионов в клетке, вызванный столкновениями молекул. Также существует диффузия атомов в жидкости, например, диффузия чернил в стакане с водой.

Один из самых популярных примеров уравнения диффузии - перенос тепла в твердых телах. Тогда **u** - температура, и уравнение предсказывает, как температура изменяется в пространстве и времени внутри твердого тела. Для таких примеров уравнение известно как уравнение теплопроводности. Отметим, что на температуру жидкости влияет не только диффузия, но и течение жидкости. Если последний эффект присутствует, требуется дополнительный член в уравнении (известный как член адвекции или конвекции).

Следует также отметить, что уравнение диффузии может появиться после упрощения более сложных уравнений в частных производных. Например, течение вязкой жидкости между двумя плоскими и параллельными пластинами описывается одномерным уравнением диффузии, где **u** - скорость жидкости.

Термин αu известен как исходный член и представляет собой выделение или потерю тепла (каким-либо механизмом) внутри тела. Для диффузионного переноса αu моделирует инъекцию или экстракцию вещества.

Уравнение в частных производных решается в некоторой области **Ω** в пространстве и за промежуток времени **[0, T]**. Решение уравнения не является единственным, если мы также не зададим начальные и граничные условия. Тип и количество таких условий зависят от типа уравнения. Для уравнения диффузии нам нужно одно начальное условие **u(x, 0)**, указывающее, что такое **u**, в точке времени **t = 0**. Кроме того, уравнение диффузии требует одного граничного условия в каждой точке границы **∂Ω** области **Ω**. Это условие может заключаться в том, что известно **u** или известна производная по нормали, **∇u·n = ∂u / ∂n** (**n** обозначает единичный вектор внешней нормали к **∂Ω**).

Граничные и начальные условия являются обязательными. Начальные и граничные условия чрезвычайно важны, без них решение не является

уникальным, и никакой численный метод работать не будет. К сожалению, во многих физических задачах одно или несколько начальных или граничных условий неизвестны.

Forward Time Centered Space – Dirichlet problem

Уравнение в частных производных вида:

$$Au_{xx} + Bu_{xy} + Cu_{yy} = f(x, y, u, u_x, u_y)$$

где A, B и C - константы, называется квазилинейным. Существуют три типа квазилинейных уравнений:

$$\begin{aligned} &\text{Elliptic,} && \text{if } B^2 - 4AC < 0 \\ &\text{Parabolic,} && \text{if } B^2 - 4AC = 0 \\ &\text{Hyperbolic,} && \text{if } B^2 - 4AC > 0 \end{aligned}$$

Уравнение теплопроводности / диффузии является примером параболического дифференциального уравнения. Общая форма одномерного уравнения теплопроводности выглядит так:

$$u_t = D\,u_{xx} + f(u, x, t)$$

должно сопровождаться начальными и граничными условиями, чтобы уравнение имело единственное решение.

Мы аппроксимируем производные по времени и пространству отдельно. Используя явный или прямой метод Эйлера, формула конечных разностей производных по времени выглядит так:

$$u_t = \frac{u(x_i, t_{j+1}) - u(x_i, t_j)}{\Delta t} + \mathcal{O}(\Delta t)$$

А формула конечных разностей производных по пространству:

$$u_{xx} = \frac{u(x_{i-1}, t_j) - 2u(x_i, t_j) + u(x_{i+1}, t_j)}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$

Рассмотрим простое уравнение теплопроводности / диффузии вида:

$$u_t = D\,u_{xx}$$

которое мы хотим решить в одномерной области **0 <= x <= L** за временной интервал **0 < t < T**.

Начальные и граничные условия задаются формулами

$$u(x,0) = f(x) \qquad u(0,t) = g_1(t)$$
$$u(L,t) = g_2(t)$$

Используя обозначения $U_{i,j} = u(x_i, t_j)$ и формулы конечных разностей уравнение диффузии становится:

$$\frac{U_{i,j+1} - U_{i,j}}{\Delta t} = D\,\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{\Delta x^2}$$

Где $i = 0, 1, 2, \ldots, M$ и $j = 0, 1, 2, \ldots, N$.

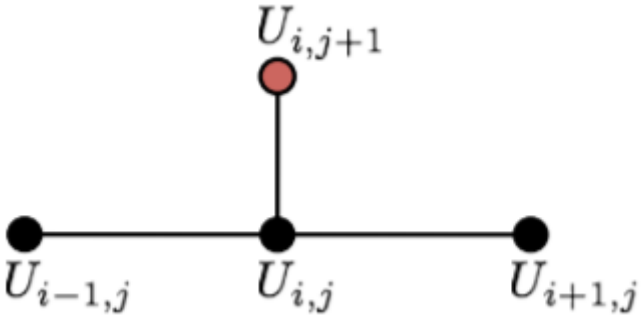
Здесь M и N номера точек сетки, Δt и Δx - размеры сетки вдоль оси t и x.

Так как: $r = \frac{D\,\Delta t}{\Delta x^2}$, то формулу выше можно представить в виде:

$$U_{i,j+1} = \frac{\Delta t D}{\Delta x^2}\left(U_{i-1,j} - 2U_{i,j} + U_{i+1,j}\right) + U_{i,j}$$

$$U_{i,j+1} = rU_{i-1,j} + (1-2r)U_{i,j} + rU_{i+1,j}$$

Отсюда можно вывести значение $U_{i,j+1}$ с помощью $U_{i-1,j}$, $U_{i,j}$ и $U_{i+1,j}$:



Эта формула верна при условии:

$$r = \frac{\Delta t D}{\Delta x^2} \leq \frac{1}{2}$$

Это означает, что размер сетки должен удовлетворять:

$$\Delta t \leq \frac{\Delta x^2}{2D}$$

Если это условие не выполняется, то ошибки, совершенные в $\{U_{i,j}\}$, могут быть увеличены в последующих $\{U_{i,p}\}$ для некоторых $p > j$.

Backward Time Centered Space – Dirichlet problem

Следующий метод называется неявным или обратным методом Эйлера.

В этом методе формула для производной по времени имеет вид:

$$u_t = \frac{u(x_i, t_j) - u(x_i, t_{j-1})}{\Delta t} + \mathcal{O}(\Delta t)$$

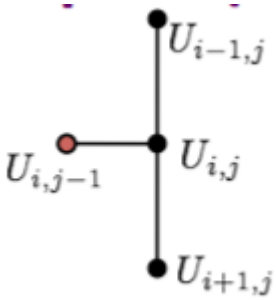
В то время, как производная по пространству такая же как в методе выше. Аппроксимация уравнения теплопроводности/диффузии выглядит так:

$$\frac{U_{i,j} - U_{i,j-1}}{\Delta t} = \frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{\Delta x^2}$$

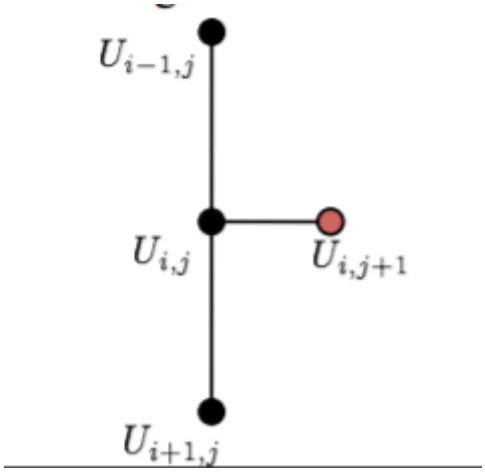
Пусть $r = \frac{D\,\Delta t}{\Delta x^2}$, тогда

$$U_{i,j-1} = -rU_{i-1,j} + (1+2r)U_{i,j} - rU_{i+1,j}$$

Вычислительный шаблон, представляющий собой неявный метод выглядит так:



Для прямого метода Эйлера шаблон выглядит так:



Если значения в конечных точках x даны из граничного условия типа Дирихле, то уравнение выше превращается в систему уравнений:

$$\begin{bmatrix} 1+2r & -r & & & \\ -r & 1+2r & -r & & \\ & \ddots & & & \\ & & \ddots & & \\ & & -r & 1+2r & -r \\ & & & -r & 1+2r \end{bmatrix} \begin{bmatrix} U_{1,j} \\ U_{2,j} \\ \vdots \\ \vdots \\ U_{M-3,j} \\ U_{M-2,j} \end{bmatrix} = \begin{bmatrix} rU_{0,j} & + & U_{1,j-1} \\ 0 & + & U_{2,j-1} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ 0 & + & U_{M-3,j-1} \\ rU_{M-1,j} & + & U_{M-2,j-1} \end{bmatrix}$$

Backward Time Centered Space – Neumann problem

Если применяется граничное условие Неймана, где $\partial_x u = g$, x = 0, то границы аппроксимируются

$$-\left(\frac{U_{i+1,j} - U_{i-1,j}}{2\Delta x}\right) = g_{i,j}$$

При x = 0 или i = 0 формула преобразуется в

$$U_{-1,j} - U_{1,j} = 2\Delta x\, g_{i,j}$$

Следовательно, по оси x = 0 приближение становится

$$U_{0,j-1} = (1+2r)U_{0,j} - 2rU_{1,j} - 2r\Delta x\, g_{0,j}$$

Система уравнений в матрично-векторном виде:

$$\begin{bmatrix} 1+2r & -2r & & & \\ -r & 1+2r & -r & & \\ & \ddots & & & \\ & & \ddots & & \\ & -r & 1+2r & -r & \\ & & -r & 1+2r & \end{bmatrix} \begin{bmatrix} U_{0,j} \\ U_{1,j} \\ \vdots \\ \vdots \\ U_{M-2,j} \\ U_{M-1,j} \end{bmatrix} = \begin{bmatrix} U_{0,j-1} + 2r\Delta x g_{0,j} \\ U_{1,j-1} \\ U_{2,j-1} \\ \vdots \\ U_{M-2,j-1} \\ U_{M-1,j-1} + rU_{M,j} \end{bmatrix}$$

Граничные условия Неймана, при x = 0 и x = L:

$$\partial_x u = g \quad \text{at} \quad x = 0$$
$$\partial_x u = f \quad \text{at} \quad x = L$$

система уравнений в матрично-векторной записи принимает вид:

$$\begin{bmatrix} 1+2r & -2r & & & \\ -r & 1+2r & -r & & \\ & \ddots & & & \\ & & \ddots & & \\ & -r & 1+2r & -r & \\ & & -2r & 1+2r & \end{bmatrix} \begin{bmatrix} U_{0,j} \\ U_{1,j} \\ \vdots \\ \vdots \\ U_{M-2,j} \\ U_{M-1,j} \end{bmatrix} = \begin{bmatrix} 2r\Delta x g_{0,j} & + & U_{0,j-1} \\ 0 & + & U_{1,j-1} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ 0 & + & U_{M-2,j-1} \\ 2r\Delta x f_{M-1,j} & + & U_{M-1,j-1} \end{bmatrix}$$

Crank-Nicolson Method – Neumann problem

Неявная схема, изобретенная Джоном Крэнком и Филлисом Николсоном, основана на численных аппроксимациях решений дифференциального уравнения в точке $\left(x_i, t_{j+\frac{\Delta t}{2}}\right)$, лежащей между строками сетки.

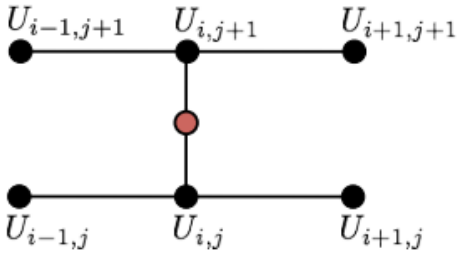
Формула аппроксимации для производной по времени имеет вид:

$$u_t\left(x_i, t_{j+\frac{\Delta t}{2}}\right) = \frac{u(x_i, t_{j+1}) - u(x_i, t_j)}{\Delta t} + \mathcal{O}(\Delta t^2)$$

а для производной по пространству:

$$u_{xx}\left(x_i, t_{j+\frac{\Delta t}{2}}\right) = \frac{1}{2\Delta x^2}\left(u(x_{i-1}, t_{j+1}) - 2u(x_i, t_{j+1}) + u(x_{i+1}, t_{j+1})\right.$$
$$\left. + u(x_{i-1}, t_j) - 2u(x_i, t_j) + u(x_{i+1}, t_j)\right) + \mathcal{O}(\Delta x^2)$$

Вычислительный шаблон показан на рисунке ниже:



Аналогично предыдущему выводу, уравнение конечных разностей для метода Кранка-Николсона имеет вид:

$$\frac{U_{i,j+1} - U_{i,j}}{\Delta t} = \frac{D}{2\Delta x^2}\left(U_{i-1,j+1} - 2U_{i,j+1} + U_{i+1,j+1}\right.$$
$$\left. + U_{i-1,j} - 2U_{i,j} + U_{i+1,j}\right)$$

Умножая обе части на Δt и подставляя получим:

$$r = \frac{D\Delta t}{2\Delta x^2}$$

После подстановки получим:

$$-rU_{i-1,j+1} + (1+2r)U_{i,j+1} - rU_{i+1,j+1} =$$
$$rU_{i-1,j} + (1-2r)U_{i,j} + rU_{i+1,j}$$

Уравнение, описанное выше может быть представлено в матрично-векторном виде:

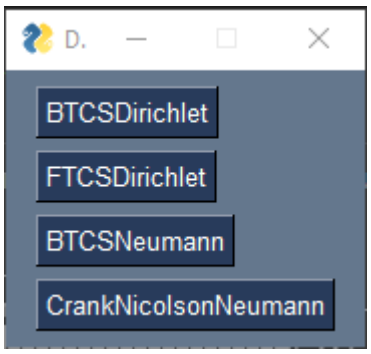
$$\begin{bmatrix} 1+2r & -r & & \\ -r & 1+2r & -r & \\ \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ & -r & 1+2r & -r \\ & & -r & 1+2r \end{bmatrix} \begin{bmatrix} U_{1,j+1} \\ U_{2,j+1} \\ \vdots \\ \vdots \\ U_{M-3,j+1} \\ U_{M-2,j+1} \end{bmatrix} = \begin{bmatrix} 1-2r & r & & \\ r & 1-2r & r & \\ \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ & r & 1-2r & r \\ & & r & 1-2r \end{bmatrix} \begin{bmatrix} U_{1,j} \\ U_{2,j} \\ \vdots \\ \vdots \\ U_{M-3,j} \\ U_{M-2,j} \end{bmatrix}$$

$$+ \begin{bmatrix} rU_{0,j} + rU_{0,j+1} \\ 0 \\ \vdots \\ \vdots \\ 0 \\ rU_{M-1,j} + rU_{M-1,j+1} \end{bmatrix}$$

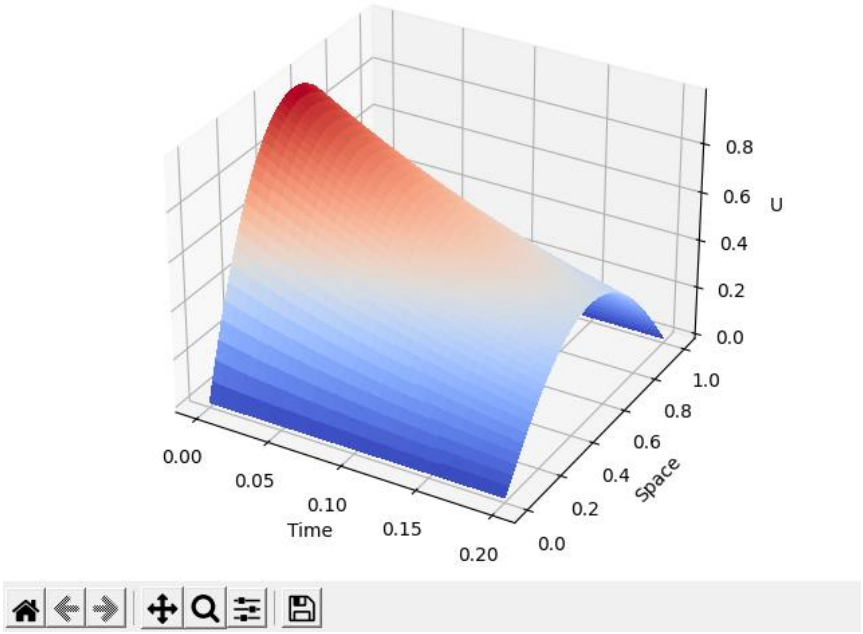
GUI

Был реализован графический интерфейс на языке Python с помощью библиотеки PySimpleGUI. Код программы представлен в приложении А.

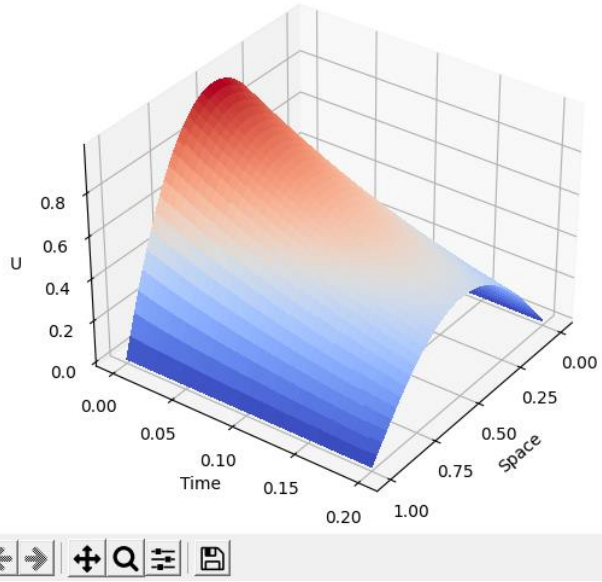
Интерфейс программы включает в себя 4 кнопки:



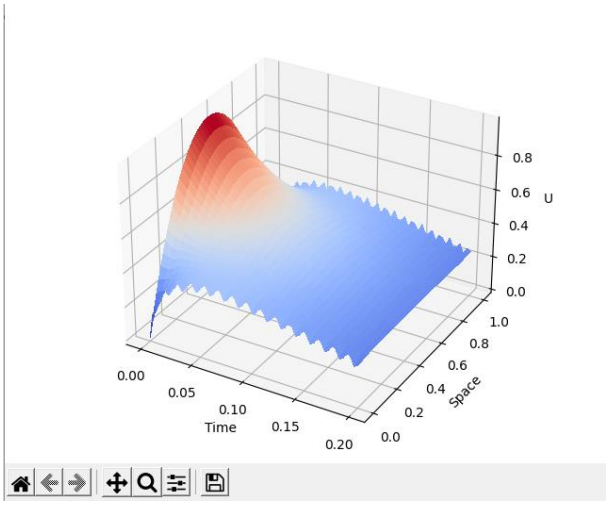
BTCSDirichlet:



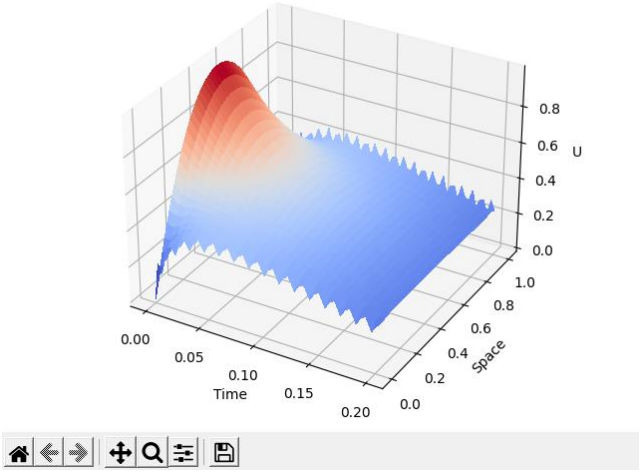
FTCSDirichlet:



BTCSNeumann:



Crank-Nicolson-Neumann:



Вывод

В курсовой работе был рассмотрен процесс диффузии. Для решения задачи были использованы такие методы, как: “FTCS - Dirichlet problem”, “BTCS - Dirichlet problem”, “BTCS-Neumann problem”, “CN-Neumann problem”. Было написано приложение на языке Python, решающее поставленную задачу данными методами. Графический интерфейс позволяет увидеть отличия между методами на графиках.

Используемая литература

http://hplgit.github.io/primer.html/doc/pub/ode2/._ode2-readable003.html

https://en.wikipedia.org/wiki/FTCS_scheme

http://hplgit.github.io/prog4comp/doc/pub/p4c-sphinx-Python/._pylight006.html

https://services.math.duke.edu/~jtwong/math563-2020/lectures/Lec10-PDEs.pdf

https://folk.ntnu.no/leifh/teaching/tkt4140/._main056.html

https://pysimplegui.readthedocs.io/en/latest/

https://www.python.org/

ПРИЛОЖЕНИЕ А. Код программы

Файл main.py:

```
import gui

def main():
    gui.MainApplication()

if __name__ == "__main__":
    main()
```

Файл gui.py:

```
import PySimpleGUI as sg
import BTCS_DirichletBCs
import FTCS_DirichletBCs
import BTCS_NeumannBCs
import CN_NeumannBCs

class MainApplication():
    def __init__(self):
        layout = [[sg.Button("BTCSDirichlet")], [sg.Button("FTCSDirichlet")],
                  [sg.Button("BTCSNeumann")], [sg.Button("CrankNicolsonNeumann")]
]

        # Create the window
        self.window = sg.Window("Diffusion", layout)

        # Create an event loop
        while True:
            event, values = self.window.read()
            # End program if user closes window
            if event == sg.WIN_CLOSED:
                break

            if event == "BTCSDirichlet":
                sim = BTCS_DirichletBCs.BTCSDirichlet(50, 60)
                sim.plot_()

            if event == "FTCSDirichlet":
                sim = FTCS_DirichletBCs.FTCSDirichlet(40, 70)
                sim.plot_()

            if event == "BTCSNeumann":
                sim = BTCS_NeumannBCs.BTCSNeumann(50, 60)
                sim.plot_()

            if event == "CrankNicolsonNeumann":
                sim = CN_NeumannBCs.CrankNicolsonNeumann(50, 60)
                sim.plot_()
```

```
    def __del__(self):
        self.window.close()
```

Файл BTCS_DirichletBCs.py:

```
import numpy as np
from scipy import sparse
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm


class BTCSDirichlet:

    def __init__(self, M, N):
        self.M = M
        self.N = N
        self.x0 = 0
        self.xL = 1
        self.t0 = 0
        self.tF = 0.2
        self.D = 0.1 # Diffusion coefficient
        self.alpha = -3 # Reaction rate
        self.createGrid()
        self.params()
        self.solve()


    def createGrid(self):
        self.xspan = np.linspace(self.x0, self.xL, self.M)
        self.tspan = np.linspace(self.t0, self.tF, self.N)


    def params(self):
        # ----- Spatial discretization step -----
        dx = (self.xL - self.x0)/(self.M - 1)
        # ----- Time step -----
        dt = (self.tF - self.t0)/(self.N - 1)
        self.r = dt*self.D/dx**2
        self.s = dt*self.alpha


    def solve(self):
        maindiag = (1 + 2*self.r - self.s)*np.ones((1, self.M-2))
        offdiag = -self.r*np.ones((1, self.M-3))
        a = maindiag.shape[1]
        diagonals = [maindiag, offdiag, offdiag]
        A = sparse.diags(diagonals, [0,-1,1], shape=(a,a)).toarray()
```

```
        # ----- Initializes matrix U -----
        self.U = np.zeros((self.M, self.N))

        #----- Initial condition -----
        self.U[:,0] = 4*self.xspan - 4*self.xspan**2

        #----- Dirichlet boundary conditions -----
        self.U[0,:] = 0.0
        self.U[-1,:] = 0.0

        for k in range(1, self.N):
            c = np.zeros((self.M-4, 1)).ravel()
            b1 = np.asarray([self.r*self.U[0,k], self.r*self.U[-1,k]])
            b1 = np.insert(b1, 1, c)
            b2 = np.array(self.U[1:self.M-1, k-1])
            b = b1 + b2  # Right hand side
            self.U[1:self.M-1, k] = np.linalg.solve(A,b)  # Solve x=A\b

        # ----- Checks if the solution is correct:
        g = np.allclose(np.dot(A,self.U[1:self.M-1,self.N-1]), b)
        print(g)


    def plot_(self):
        # ----- Surface plot -----
        X, T = np.meshgrid(self.tspan, self.xspan)

        fig = plt.figure()
        ax = fig.gca(projection='3d')

        ax.plot_surface(X, T, self.U, linewidth=0,
                                cmap=cm.coolwarm, antialiased=False)

        ax.set_xticks([0, 0.05, 0.1, 0.15, 0.2])
        ax.set_xlabel('Time')
        ax.set_ylabel('Space')
        ax.set_zlabel('U')
        plt.tight_layout()
        plt.show()


# def main():
#     sim = BTCSDirichlet(50, 60)
#     sim.plot_()

# if __name__ == "__main__":
#     main()


#M = 50 # GRID POINTS on space interval
#N = 60 # GRID POINTS on time interval
```

Файл BTCS_NeumannBCs.py:

```
import numpy as np
from scipy import sparse
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm


class BTCSNeumann:

    def __init__(self, M, N):
        self.M = M
        self.N = N
        self.x0 = 0
        self.xL = 1
        self.t0 = 0
        self.tF = 0.2
        self.D = 0.5  # Diffusion coefficient
        self.alpha = -5 # Reaction rate
        self.params()
        self.createGrid()
        self.solve()


    def params(self):
        # ----- Spatial discretization step -----
        self.dx = (self.xL - self.x0)/(self.M - 1)
        # ----- Time step -----
        dt = (self.tF - self.t0)/(self.N - 1)
        self.r = dt*self.D/self.dx**2
        self.s = dt*self.alpha


    def createGrid(self):
        self.xspan = np.linspace(self.x0, self.xL, self.M)
        self.tspan = np.linspace(self.t0, self.tF, self.N)


    def solve(self):
        maindiag = (1 + 2*self.r - self.s)*np.ones((1,self.M))
        offdiag = -self.r*np.ones((1, self.M-1))
        a = maindiag.shape[1]
        diagonals = [maindiag, offdiag, offdiag]
        A = sparse.diags(diagonals, [0,-1,1], shape=(a,a)).toarray()
        A[0, 1] = -2*self.r
        A[self.M-1, self.M-2] = -2*self.r

        # ----- Initializes matrix U -----
        self.U = np.zeros((self.M, self.N))
```

```
        #----- Initial condition -----
        self.U[:,0] = 4*self.xspan - 4*self.xspan**2

        #----- Neumann boundary conditions -----
        leftBC = np.arange(1, self.N+1)
        f = np.sin(leftBC*np.pi/2)

        rightBC = np.arange(1, self.N+1)
        g = np.sin(3*rightBC*np.pi/4)

        for k in range(1, self.N):
            c = np.zeros((self.M-2, 1)).ravel()
            b1 = np.asarray([2*self.r*self.dx*f[k], 2*self.r*self.dx*g[k]])
            b1 = np.insert(b1, 1, c)
            b2 = np.array(self.U[0:self.M, k-1])
            b = b1 + b2  # Right hand side
            self.U[0:self.M, k] = np.linalg.solve(A,b)  # Solve x=A\b

        # ----- Checks if the solution is correct:
        gc = np.allclose(np.dot(A,self.U[0:self.M,self.N-1]), b)
        print(gc)


    def plot_(self):
        # ----- Surface plot -----
        X, T = np.meshgrid(self.tspan, self.xspan)

        fig = plt.figure()
        ax = fig.gca(projection='3d')

        ax.plot_surface(X, T, self.U, linewidth=0,
                        cmap=cm.coolwarm, antialiased=False)

        ax.set_xticks([0, 0.05, 0.1, 0.15, 0.2])

        ax.set_xlabel('Time')
        ax.set_ylabel('Space')
        ax.set_zlabel('U')
        plt.tight_layout()
        plt.show()


# def main():
#     sim = BTCSNeumann(50, 60)
#     sim.plot_()

# if __name__ == "__main__":
#     main()
```

Файл CN_NeumannBCs.py:

```
import numpy as np
from scipy import sparse
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm


class CrankNicolsonNeumann:

    def __init__(self, M, N):
        self.M = M
        self.N = N
        self.x0 = 0
        self.xL = 1
        self.t0 = 0
        self.tF = 0.2
        self.D = 0.5  # Diffusion coefficient
        self.alpha = -5.0 # Reaction rate
        self.params()
        self.createGrid()
        self.matrices()
        self.solve()


    def params(self):
        # ----- Spatial discretization step -----
        self.dx = (self.xL - self.x0)/(self.M - 1)
        # ----- Time step -----
        dt = (self.tF - self.t0)/(self.N - 1)
        self.r = dt*self.D/(2*self.dx**2)
        self.s = dt*self.alpha/2
        self.a0 = 1 + 2*self.r - self.s
        self.c0 = 1 - 2*self.r + self.s


    def createGrid(self):
        self.xspan = np.linspace(self.x0, self.xL, self.M)
        self.tspan = np.linspace(self.t0, self.tF, self.N)


    def matrices(self):
        main_diag_a0 = self.a0*np.ones((1, self.M))
        off_diag_a0 = -self.r*np.ones((1, self.M-1))

        main_diag_c0 = self.c0*np.ones((1, self.M))
        off_diag_c0 = self.r*np.ones((1, self.M-1))

        # Left-hand side tri-diagonal matrix
        a = main_diag_a0.shape[1]
```

```
        diagonalsA = [main_diag_a0, off_diag_a0, off_diag_a0]
        self.A = sparse.diags(diagonalsA, [0,-1,1], shape=(a,a)).toarray()
        self.A[0, 1] = -2*self.r
        self.A[self.M-1, self.M-2] = -2*self.r

        # Right-hand side tri-diagonal matrix
        c = main_diag_c0.shape[1]
        diagonalsC = [main_diag_c0, off_diag_c0, off_diag_c0]
        self.A_rhs = sparse.diags(diagonalsC, [0,-1,1], shape=(c,c)).toarray()
        self.A_rhs[0, 1] = 2*self.r
        self.A_rhs[self.M-1, self.M-2] = 2*self.r


    def solve(self):

        # ----- Initializes matrix U -----
        self.U = np.zeros((self.M, self.N))

        #----- Initial condition -----
        self.U[:,0] = 4*self.xspan - 4*self.xspan**2

        #----- Neumann boundary conditions -----
        leftBC = np.arange(1, self.N+1)
        f = np.sin(leftBC*np.pi/2)

        rightBC = np.arange(1, self.N+1)
        g = np.sin(3*rightBC*np.pi/4)

        for k in range(1, self.N):
            ins = np.zeros((self.M-2, 1)).ravel()
            b1 = np.asarray([4*self.r*self.dx*f[k], 4*self.r*self.dx*g[k]])
            b1 = np.insert(b1, 1, ins)
            b2 = np.matmul(self.A_rhs, np.array(self.U[0:self.M, k-1]))
            b = b1 + b2  # Right hand side
            self.U[0:self.M, k] = np.linalg.solve(self.A,b)  # Solve x=A\b

        # ----- Checks if the solution is correct:
        gc = np.allclose(np.dot(self.A,self.U[0:self.M,self.N-1]), b)
        print(gc)


    def plot_(self):
        # ----- Surface plot -----
        X, T = np.meshgrid(self.tspan, self.xspan)

        fig = plt.figure()
        ax = fig.gca(projection='3d')

        ax.plot_surface(X, T, self.U, linewidth=0,
                                cmap=cm.coolwarm, antialiased=False)
```

```
        ax.set_xticks([0, 0.05, 0.1, 0.15, 0.2])
        ax.set_xlabel('Time')
        ax.set_ylabel('Space')
        ax.set_zlabel('U')
        plt.tight_layout()
        plt.show()


# def main():
#     sim = CrankNicolsonNeumann(50, 60)
#     sim.plot_()

# if __name__ == "__main__":
#     main()

#M = 50 # GRID POINTS on space interval
#N = 60 # GRID POINTS on time interval
```

FTCS_DirichletBCs.py:

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm


class FTCSDirichlet:

    def __init__(self, M, N):
        self.M = M
        self.N = N
        self.x0 = 0
        self.xL = 1.0
        self.t0 = 0
        self.tF = 0.2
        self.D = 0.1 # Diffusion coefficient
        self.alpha = -3 # Reaction rate
        self.createGrid()
        self.params()
        self.solve()

    def createGrid(self):
        self.xspan = np.linspace(self.x0, self.xL, self.M)
        self.tspan = np.linspace(self.t0, self.tF, self.N)

    def params(self):
        # ----- Spatial discretization step -----
        dx = (self.xL - self.x0)/(self.M - 1)
        # ----- Time step -----
        dt = (self.tF - self.t0)/(self.N - 1)
```

```
        self.r = dt*self.D/dx**2
        self.s = dt*self.alpha

    def solve(self):
        # ----- Initializes matrix solution U -----
        self.U = np.zeros((self.M, self.N))
        # ----- Initial condition -----
        self.U[:,0] = 4*self.xspan - 4*self.xspan**2
        # ----- Dirichlet Boundary Conditions -----
        self.U[0,:] = 0.0
        self.U[-1,:] = 0.0

        for k in range(0, self.N-1):
            for i in range(1, self.M-1):
                self.U[i, k+1] = self.r*self.U[i-1, k] + (1-
2*self.r+self.s)*self.U[i,k] + self.r*self.U[i+1,k]

        self.T, self.X = np.meshgrid(self.tspan, self.xspan)

    def plot_(self):
        fig = plt.figure()
        ax = fig.gca(projection='3d')

        ax.plot_surface(self.X, self.T, self.U, cmap=cm.coolwarm,
                        linewidth=0, antialiased=False)

        ax.set_xticks([0, 0.25, 0.5, 0.75, 1.0])
        ax.set_yticks([0, 0.05, 0.1, 0.15, 0.2])

        ax.set_xlabel('Space')
        ax.set_ylabel('Time')
        ax.set_zlabel('U')
        ax.view_init(elev=33, azim=36)
        plt.tight_layout()
        plt.show()


# def main():
#     sim = FTCSDirichlet(40, 70)
#     sim.plot_()

# if __name__ == "__main__":
#     main()

#M = 40 # GRID POINTS on space interval
#N = 70 # GRID POINTS on time interval
```