**ОТЧЁТ**

**ЛАБОРАТОРНАЯ РАБОТА №7**

**Квазилинейное уравнение переноса**

**Вариант 9**

*Выполнил студент 3 курса МОиАИС*

Ходосевич Данила

***Цель работы***: усвоить сущность и методы решения ***квазилинейного дифференциального уравнения 1-го порядка в частных производных с разрывными начальными условиями***.

Численное решение дифференциального уравнения в частных производных предполагает получение двумерной числовой таблицы приближенных значений *Uij* искомой функции *U*(*t,x)* с заданной точностью для некоторых значений аргументов

*xj Î* [*a*, *b*], *ti Î* [*c*, *d*]

Численное решение таких дифференциальных уравнений возможно методами конечных разностей.

Погрешность решения, найденного этими методами, оценивается величиной O(*tp,hq*)*,* где *p*, *q*  - порядок метода.

***Задание.***

Решить уравнение переноса

(1)

методом с искусственной вязкостью и консервативной схемы.

***Варианты задания (лабораторная №7)***

Для всех вариантов [*a*, *b*] = [0; 1], [*c*, *d*] = [0; 1]. Погрешность решения 0,01 (определяется сходимостью схемы и величиной шагов).

|  |  |
| --- | --- |
| № вариантов | Начальное условие |
| 9,19,29 | {2, x≥0,54, x<0,52, x≥0,54, x<0,5 |

1. **Решение методом искусственной вязкости**

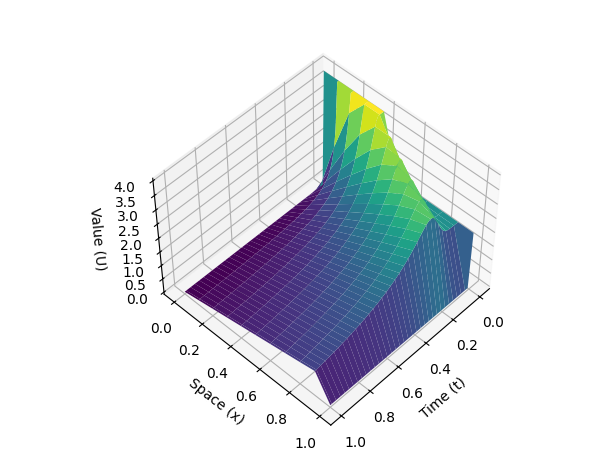
Введем искусственную вязкость, рассмотрим данное уравнение:, ε – параметр вязкости.

Для численного решения используется разностная схема. Пример:

Где uij​ — значение функции U(x,t)U(x,t)U(x,t) в узле сетки с координатами xi=ih и временем tj=jτt

Упрощаем это выражение и разрешаем его относительно неизвестного значения сеточной функции на j+1 слое

Для явной схемы необходимо выполнять условие устойчивости. Эта явная схема условно устойчива при выполнении неравенства



1. **Консервативные схемы**

В консервативной форме рассмотрим уравнение переноса:

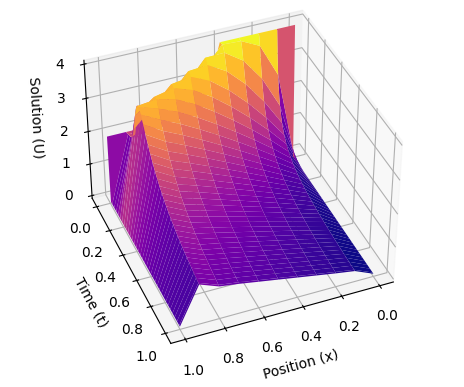
Интегрируем это уравнение по области , получаем

Для численного решения применяется **метод прямоугольников**, в котором интегралы заменяются суммами: ; .

Окончательно получим разностную схему вида:

Отсюда можно найти значение искомой функции на верхнем слое с помощью решения на нижнем слое. Следовательно, это явная схема.

Упрощаем это выражение и разрешаем его относительно неизвестного значения сеточной функции на j+1 слое



Приложения

Graph1.py

import matplotlib.pyplot as plt

import numpy as np

# Parameters

space\_step = 0.1

time\_step = 0.01

diffusion\_coef = 0.01

x\_start, x\_end = 0, 1

t\_start, t\_end = 0, 1

# Grid dimensions

num\_x = int((x\_end - x\_start) / space\_step) + 1

num\_t = int((t\_end - t\_start) / time\_step) + 1

# Initialize the grid

grid = np.zeros((num\_t, num\_x))

# Initial conditions

for idx\_x in range(num\_x):

    x\_value = x\_start + idx\_x \* space\_step

    grid[0, idx\_x] = 2 if x\_value >= 0.5 else 4

# Update values using the scheme

for idx\_t in range(num\_t - 1):

    for idx\_x in range(1, num\_x - 1):

        grid[idx\_t + 1, idx\_x] = (

            grid[idx\_t, idx\_x]

            - time\_step / space\_step \* grid[idx\_t, idx\_x] \* (grid[idx\_t, idx\_x] - grid[idx\_t, idx\_x - 1])

            - diffusion\_coef\*\*2 \* time\_step / (2 \* space\_step\*\*3)

            \* (grid[idx\_t, idx\_x + 1] - 2 \* grid[idx\_t, idx\_x] + grid[idx\_t, idx\_x - 1])

        )

# Stability condition check

for idx\_t in range(num\_t):

    for idx\_x in range(num\_x):

        if grid[idx\_t, idx\_x] > (space\_step / time\_step):

            print("Stability condition violated!")

            break

# Create mesh grid

x\_vals = np.linspace(x\_start, x\_end, num\_x)

t\_vals = np.linspace(t\_start, t\_end, num\_t)

X, T = np.meshgrid(x\_vals, t\_vals)

# Plotting the 3D surface

fig = plt.figure()

ax = fig.add\_subplot(projection="3d")

ax.set\_xlabel("Time (t)")

ax.set\_ylabel("Space (x)")

ax.set\_zlabel("Value (U)")

ax.plot\_surface(T, X, grid, cmap='viridis')

plt.show()

Graph2.py

import matplotlib.pyplot as plt

import numpy as np

# Parameters

dx = 0.1

dt = 0.01

epsilon = 0.01

x\_min, x\_max, t\_min, t\_max = 0, 1, 0, 1

# Grid dimensions

num\_x\_points = int((x\_max - x\_min) / dx) + 1

num\_t\_points = int((t\_max - t\_min) / dt) + 1

# Initialize the solution array

solution = np.zeros((num\_t\_points, num\_x\_points))

# Setting initial conditions

for idx\_x in range(num\_x\_points):

    x\_pos = x\_min + idx\_x \* dx

    solution[0, idx\_x] = 2 if x\_pos >= 0.5 else 4

# Stability condition update

for idx\_t in range(num\_t\_points - 1):

    for idx\_x in range(1, num\_x\_points - 1):

        solution[idx\_t + 1, idx\_x] = (

            solution[idx\_t, idx\_x]

            + dt / (2 \* dx) \* (solution[idx\_t, idx\_x - 1]\*\*2 - solution[idx\_t, idx\_x]\*\*2)

            - epsilon \* (solution[idx\_t, idx\_x + 1] - 2 \* solution[idx\_t, idx\_x] + solution[idx\_t, idx\_x - 1])

        )

# Create the mesh grid for visualization

x\_vals = np.linspace(x\_min, x\_max, num\_x\_points)

t\_vals = np.linspace(t\_min, t\_max, num\_t\_points)

X, T = np.meshgrid(x\_vals, t\_vals)

# Plotting the solution

fig = plt.figure()

ax = fig.add\_subplot(projection="3d")

ax.set\_xlabel("Position (x)")

ax.set\_ylabel("Time (t)")

ax.set\_zlabel("Solution (U)")

ax.plot\_surface(X, T, solution, cmap="plasma")

plt.show()