**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №9**

**Уравнение параболического типа**

**Вариант 9**

*Выполнил студент 3 курса МОиАИС*

*Ходосевич Данила*

***Цель работы***: усвоить сущность и методы решения ***линейного дифференциального уравнения 2-го порядка параболического типа***.

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.

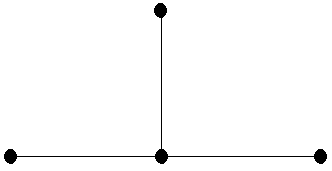
***Задание.***

Решить параболическое уравнение

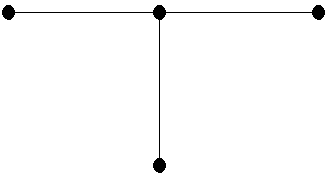
**

явным методом и неявным методом.

Шаблон для явного метода:

**

Шаблон для неявного метода:

**

Вывести результаты в виде двумерных графиков U(x,t).

Неявные схемы решать с помощью прогонки.

**Метод прогонки РАСПИСАТЬ подробно!**

Для всех вариантов [*a*, *b*] = [0; 1], [*c*, *d*] = [0; 10], D=1. Погрешность решения 0,01. Исходя из погрешности, порядка аппроксимации и условий сходимости для явных схем определить шаги по пространству и времени

|  |  |  |
| --- | --- | --- |
| № вар. | Начальные условия | Граничные условия |
| 9 |  |  |

**Явный метод:**

Уравнение теплопроводности в одномерном виде:

Разностное представление уравнения строится по схеме, состоящей из четырех узлов (явная схема):

Где – номер узла

- номер слоя

Для явной схемы необходимо условие устойчивости:

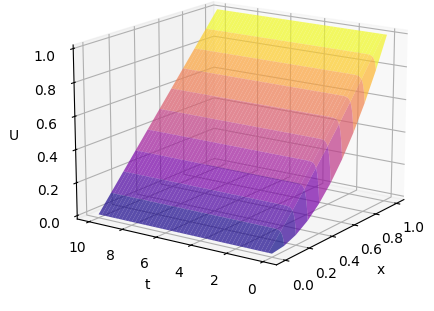
Разрешая уравнение относительно :

Начальные условия:

Граничные условия в сеточном виде:

Возьмем в качестве первого шага разбиения: и .

Итерации завершаются, когда разница значений и становится меньше порогового значения:



**Неявный метод:**

Временная производная аппроксимируется назад:

Пространственная вторая производная аппроксимируется:

Разностное уравнение, построенное по данной схеме:

Где – номер узла

- номер слоя

Из этого разностного соотношения можно получить систему уравнений относительно неизвестных значений сеточной функции на слое:

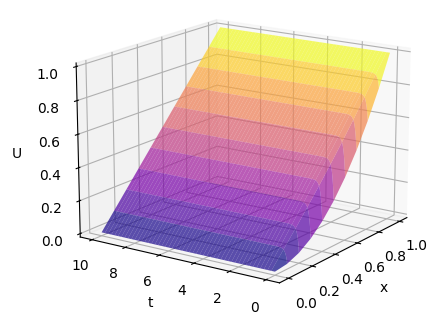
Полученная схема устойчива и сходится со скоростью .

Начальные условия:

Граничные условия в сеточном виде:

Возьмем в качестве первого шага разбиения: и .

Итерации завершаются, когда разница значений и становится меньше порогового значения:



Приложения:

import math

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

def visualize\_results(results):

    rows, cols = np.size(results, 0), np.size(results, 1)

    x\_vals, t\_vals, u\_vals = [0] \* rows \* cols, [0] \* rows \* cols, [0] \* rows \* cols

    for r in range(rows):

        for c in range(cols):

            x\_vals[c + r \* cols] = results[r, c][0]

            t\_vals[c + r \* cols] = results[r, c][1]

            u\_vals[c + r \* cols] = results[r, c][2]

    x, t, u = np.reshape(x\_vals, (rows, cols)), np.reshape(t\_vals, (rows, cols)), np.reshape(u\_vals, (rows, cols))

    fig = plt.figure()

    ax = fig.add\_subplot(111, projection='3d')

    surf = ax.plot\_surface(x, t, u, cmap=cm.plasma, alpha=0.7)

    ax.set\_xlabel('x')

    ax.set\_ylabel('t')

    ax.set\_zlabel('U')

    plt.colorbar(surf, ax=ax, shrink=0.5, aspect=10)

    plt.show()

def explicit\_scheme(init\_x, init\_t0, init\_t1, coeff, step):

    def u0x(x):

        return init\_x(x)

    def ut0(t):

        return init\_t0(t)

    def ut1(t):

        return init\_t1(t)

    tau = step\*\*2 / (2 \* coeff + 3)

    t\_steps = math.floor(10 / tau) + 1

    x\_steps = math.floor(1 / step) + 1

    results = np.zeros((t\_steps, x\_steps), dtype='f,f,f')

    lamb = coeff \* tau / step\*\*2

    for xi in range(x\_steps):

        x\_val = xi \* step

        u\_val = u0x(x\_val)

        results[0, xi] = (x\_val, 0.0, u\_val)

    last\_x\_val = (x\_steps - 1) \* step

    for ti in range(t\_steps):

        t\_val = ti \* tau

        results[ti, 0] = (0.0, t\_val, ut0(t\_val))

        results[ti, x\_steps - 1] = (last\_x\_val, t\_val, ut1(t\_val))

    for ti in range(1, t\_steps):

        print(f"step {ti} of {t\_steps}")

        t\_val = ti \* tau

        for xi in range(1, x\_steps - 1):

            results[ti, xi] = (

                results[ti - 1, xi][0],

                t\_val,

                (1 - 2 \* lamb) \* results[ti - 1, xi][2] +

                lamb \* results[ti - 1, xi - 1][2] +

                lamb \* results[ti - 1, xi + 1][2]

            )

    return results

def implicit\_method(init\_x, init\_t0, init\_t1, coeff, step):

    def u0x(x):

        return init\_x(x)

    def ut0(t):

        return init\_t0(t)

    def ut1(t):

        return init\_t1(t)

    tau = step\*\*2 / (2 \* coeff + 3)

    t\_steps = math.floor(10 / tau) + 1

    x\_steps = math.floor(1 / step) + 1

    results = np.zeros((t\_steps, x\_steps), dtype='f,f,f')

    lamb = coeff \* tau / step\*\*2

    for xi in range(x\_steps):

        x\_val = xi \* step

        u\_val = u0x(x\_val)

        results[0, xi] = (x\_val, 0.0, u\_val)

    last\_x\_val = (x\_steps - 1) \* step

    for ti in range(t\_steps):

        t\_val = ti \* tau

        results[ti, 0] = (0.0, t\_val, ut0(t\_val))

        results[ti, x\_steps - 1] = (last\_x\_val, t\_val, ut1(t\_val))

    coeffs = np.zeros(x\_steps - 1, dtype='f,f,f')

    for ti in range(1, t\_steps):

        print(f"step {ti} of {t\_steps}")

        t\_val = ti \* tau

        d = -results[ti - 1, 1][2] - lamb \* results[ti, 0][2]

        a, b = lamb / (1 + 2 \* lamb), -d / (1 + 2 \* lamb)

        coeffs[0] = (a, b)

        for xi in range(2, x\_steps - 2):

            d = -results[ti - 1, xi][2]

            e = lamb \* coeffs[xi - 2][0] - (1 + 2 \* lamb)

            a = -lamb / e

            b = (d - lamb \* coeffs[xi - 2][1]) / e

            coeffs[xi - 1] = (a, b)

        d = -results[ti - 1, x\_steps - 2][2] - lamb \* results[ti, x\_steps - 1][2]

        u\_val = (d - lamb \* coeffs[x\_steps - 4][1]) / (-(1 + 2 \* lamb) + lamb \* coeffs[x\_steps - 4][0])

        results[ti, x\_steps - 2] = (step \* (x\_steps - 2), t\_val, u\_val)

        for xi in range(x\_steps - 3, 0, -1):

            u\_val = coeffs[xi - 1][0] \* results[ti, xi + 1][2] + coeffs[xi - 1][1]

            results[ti, xi] = (step \* xi, t\_val, u\_val)

    return results

def u0x(x):

    return x\*\*2

def ut0(t):

    return 0

def ut1(t):

    return 1

res\_ = explicit\_scheme(u0x, ut0, ut1, 1, 0.1)

visualize\_results(res\_)