Лабораторная работа №2 учебного года 2023-2024 по курсу «Численные методы»

Выполнил: Зубко Д. В.
Группа: М8О-408Б-20
Преподаватель: Пивоваров Д.Е.
Вариант по списку группы: 8

Условие лабораторной работы

Используя явную схему крест и неявную схему, решить начально-краевую задачу для дифференциального уравнения гиперболического типа. Аппроксимацию второго начального условия произвести с первым и со вторым порядком. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x,t)$. Исследовать зависимость погрешности от сеточных параметров τ, h.

Вариант 8

8.

$$\frac{\partial^2 u}{\partial t^2} + 2\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + 2\frac{\partial u}{\partial x} - 3u,$$

$$u(0,t) = 0,$$

$$u(\pi,t) = 0,$$

$$u(x,0) = 0,$$

$$u_t(x,0) = 2\exp(-x)\sin x .$$

Аналитическое решение: $U(x,t) = \exp(-t-x)\sin x \sin(2t)$

Программа

main.py

```
import numpy as np
```

```
from functions import phi0, phi1, psi1, psi2
from show import show_inaccuracy, show_result
from task import a, b, c, count_t, count_x, e, h, sigma, tau


def explicit_scheme(bound_condition, initial_condition):
    u = np.zeros((count_t, count_x))

    for i in range(0, count_x):
        u[0][i] = psi1(i * h)

        if initial_condition == 1:
            u[1][i] = psi1(i * h) + psi2(i * h) * tau + 0
        elif initial_condition == 2:
            u[1][i] = (
                psi1(i * h)
                + tau * psi2(i * h)
                + 0.5 * (tau**2) * (-e * psi2(i * h) + c * psi1(i * h))
            )
        else:
            print("Approximation type not found")

    for k in range(2, count_t):
        for i in range(1, count_x - 1):
            u[k][i] = (
                4 * u[k - 1][i]
                + (e * tau - 2) * u[k - 2][i]
                + 2
                * a
                * (tau**2)
                * (u[k - 1][i - 1] - 2 * u[k - 1, i] + u[k - 1][i + 1])
                / (h**2)
                + b * (tau**2) * (u[k - 1][i + 1] - u[k - 1][i - 1]) / h
                + 2 * (tau**2) * (c * u[k - 1][i])
            ) / (2 + e * tau)
        if bound_condition == 1:
            u[k][0] = 0
            u[k][-1] = 0
        elif bound_condition == 2:
            u[k][0] = (phi0(k * tau) + u[k][2] / (2 * h) - 2 * u[k][1] / h) * 2
* h / -3
            u[k][-1] = phi1(k * tau)
```

```
        elif bound_condition == 3:
            u[k][0] = 2 * h * phi1(k * tau)
            u[k][-1] = 2 * h * phi1(k * tau)
        else:
            print("Условие не найдено")

    return u


def implicit_scheme(bound_condition, initial_condition):
    u = np.zeros((count_t, count_x))

    ai = np.zeros(count_x)
    bi = np.zeros(count_x)
    ci = np.zeros(count_x)
    di = np.zeros(count_x)

    for i in range(0, count_x):
        u[0][i] = psi1(i * h)

        if initial_condition == 1:
            u[1][i] = psi1(i * h) + psi2(i * h) * tau
        elif initial_condition == 2:
            u[1][i] = (
                psi1(i * h)
                + tau * psi2(i * h)
                + 0.5 * (tau**2) * (-e * psi2(i * h) + c * psi1(i * h))
            )
        else:
            print("Условие не найдено")

    for k in range(2, count_t):
        for i in range(1, count_x - 1):
            ai[i] = 2 * a - h * b
            bi[i] = 2 * (h**2) * (-e / (2 * tau) - 1 / (tau**2) + c) - 4 * a
            ci[i] = h * b + 2 * a
            di[i] = (
                -4 * (h**2) * u[k - 1][i] / (tau**2)
                + (2 * (h**2) / (tau**2) - e * (h**2) / tau) * u[k - 2][i]
            )

        if bound_condition == 1:
```

```
        bi[0] = h
        ci[0] = 0
        di[0] = h * phi0(k * tau)
        ai[-1] = 2 * sigma
        bi[-1] = -(1 + 2 * sigma - c * tau)
        di[-1] = -phi1(k * tau)
    elif bound_condition == 2:
        bi[0] = -(1 + 2 * sigma - c * tau)
        ci[0] = 2 * sigma
        di[0] = -(u[k - 1][0] - 2 * a * tau * phi0(k * tau) / h)
        ai[-1] = 2 * sigma
        bi[-1] = -(1 + 2 * sigma - c * tau)
        di[-1] = -phi1(k * tau)
    elif bound_condition == 3:
        bi[0] = -(1 + 2 * sigma - c * tau)
        ci[0] = 2 * sigma
        di[0] = -(
            (1 - sigma) * u[k - 1][1] + sigma / 2 * u[k - 1][0]
        ) - sigma * phi0(k * tau)
        ai[-1] = 2 * sigma
        bi[-1] = -(1 + 2 * sigma - c * tau)
        di[-1] = -phi1(k * tau)
    else:
        print("Условие не найдено")

    u[k] = thomas_algorithm(ai, bi, ci, di)

return u


def thomas_algorithm(a, b, c, d):
    size = len(a)
    p = np.zeros(size)
    q = np.zeros(size)
    p[0] = -c[0] / b[0]
    q[0] = d[0] / b[0]

    for i in range(1, size):
        s = b[i] + a[i] * p[i - 1]
        p[i] = -c[i] / s
        q[i] = (d[i] - a[i] * q[i - 1]) / s
```

```
    result = np.zeros(size)
    result[-1] = q[-1]

    for i in range(size - 2, -1, -1):
        result[i] = p[i] * result[i + 1] + q[i]

    return result


def get_axis_np(count, mul):
    axis = np.zeros(count)
    for i in range(count):
        axis[i] = mul * i
    return axis


def main():
    res1 = explicit_scheme(1, 1)
    res2 = implicit_scheme(1, 1)

    t_axis = get_axis_np(count_t, tau)
    x_axis = get_axis_np(count_x, h)

    show_result(t_axis, x_axis, res1, res2)
    show_inaccuracy(t_axis, x_axis, res1)


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

show.py

```
import numpy as np
from matplotlib import pyplot as plt

from functions import analytic_solution
from task import count_t


def show_result(t_axis, x_axis, u1, u2):
    fig, ax = plt.subplots(2)
    fig.suptitle("Сравнение численных решений ДУ с
```

```
аналитическим")
    fig.set_figheight(15)
    fig.set_figwidth(16)
    time = 0
    for i in range(2):
        ax[i].plot(x_axis, u1[time, :], label="Explicit scheme")
        ax[i].plot(x_axis, u2[time, :], label="Implicit scheme")
        ax[i].plot(
            x_axis,
            [analytic_solution(x, t_axis[time]) for x in x_axis],
            label="Analytic",
        )
        ax[i].grid(True)
        ax[i].set_xlabel("x")
        ax[i].set_ylabel("u")
        ax[i].set_title(f"Решения при t = {time / count_t}")
        time += count_t - 1

    plt.legend(bbox_to_anchor=(1.05, 2), loc="upper right",
borderaxespad=0)
    plt.show()

    fig = plt.figure(num=1, figsize=(19, 12), clear=True)
    ax = fig.add_subplot(1, 1, 1, projection="3d")
    fig.suptitle("Аналитическое решение")
    xgrid, tgrid = np.meshgrid(x_axis, t_axis)
    ax.plot_surface(xgrid, tgrid, analytic_solution(xgrid, tgrid))
    ax.set(xlabel="x", ylabel="t", zlabel="u")
    fig.tight_layout()
    plt.show()


def show_inaccuracy(t_axis, x_axis, u):
    inaccuracy = np.zeros(count_t)
    for i in range(count_t):
        inaccuracy[i] = np.max(
            np.abs(u[i] - np.array([analytic_solution(x, t_axis[i]) for x in
x_axis]))
        )

    plt.figure(figsize=(14, 8))
    plt.plot(t_axis[1:], inaccuracy[1:], "violet", label="Ошибка")
```

```
    plt.legend(bbox_to_anchor=(1.05, 1), loc="upper right",
borderaxespad=0.0)
    plt.title("График изменения ошибки во времени")
    plt.xlabel("t")
    plt.ylabel("error")
    plt.grid(True)
    plt.show()
```

task.py

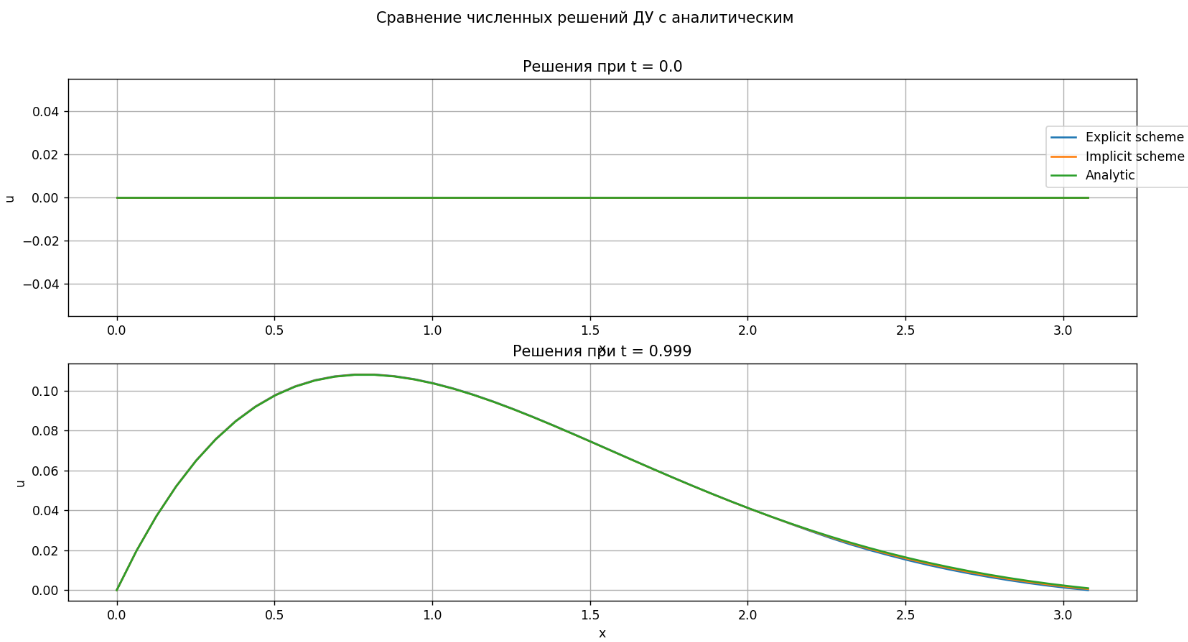
```
import numpy as np

t_max = 1
count_x = 50
count_t = 1000
r_coord = np.pi
a = 1
b = 2
c = -3
e = 2
h = r_coord / count_x
tau = t_max / count_t
# число Куранта
sigma = a * tau / (h**2)
```

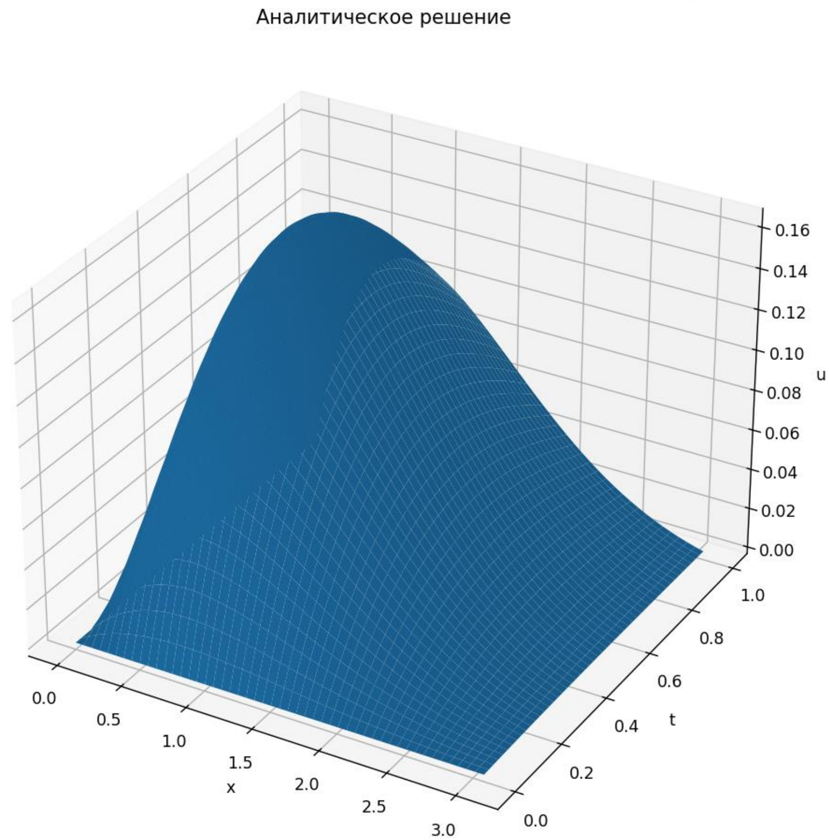
functions.py

```
import numpy as np


def analytic_solution(x, t):
    return np.exp(-t - x) * np.sin(x) * np.sin(2 * t)


def phi0(t):
    return 0


def phi1(t):
    return 0
```

```
def psi1(x):
    return 0


def psi2(x):
    return 2 * np.exp(-x) * np.sin(x)
```

Результаты работы




Аналитическое решение
u
t
x
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

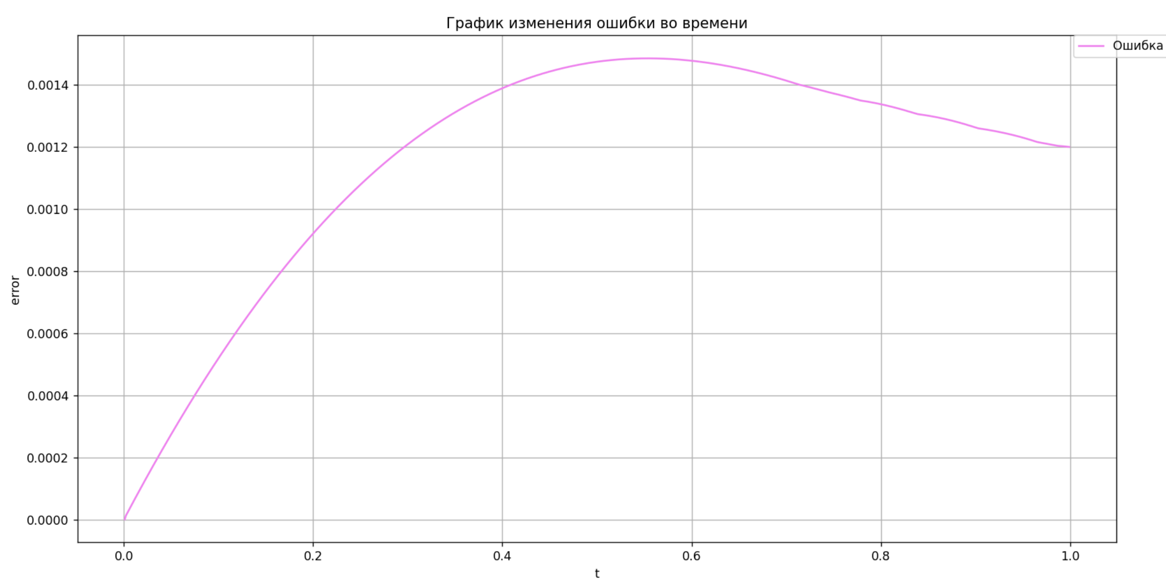


График изменения ошибки во времени
Ошибка
error
t
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0.0
0.2
0.4
0.6
0.8
1.0

Вывод по лабораторной работе

После выполнения лабораторной работы, я успешно применил два разных метода для решения начально-краевой задачи дифференциального уравнения гиперболического типа и проанализировал погрешности получившихся результатов вычислений.