Лабораторная работа №4 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 схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. 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_x, h_y.

Вариант 8

8.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - xy\sin t,$$

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

$$u(1,y,t) - u_x(1,y,t) = 0,$$

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

$$u(x,1,t) - u_y(x,1,t) = 0,$$

$$u(x,y,0) = xy.$$

Аналитическое решение: $U(x,y,t) = xy\cos t$.

Программа

main.py

```
import numpy as np

from functions import f, phi1, phi2, phi3, phi4, psi
from show import show_inaccuracy, show_result
```

```
from task import count_x, count_y, hx, hy, t_count, tau


def alternative_directions_scheme():
    u = np.zeros((count_x + 1, count_y + 1, t_count + 1))
    u_1 = np.zeros((count_x + 1, count_y + 1))
    u_2 = np.zeros((count_x + 1, count_y + 1))

    ai = np.zeros(count_x + 1)
    bi = np.zeros(count_x + 1)
    ci = np.zeros(count_x + 1)
    di = np.zeros(count_x + 1)

    for i in range(count_x + 1):
        for j in range(count_y + 1):
            u[i, j, 0] = psi(i * hx, j * hy)

    for k in range(1, t_count + 1):
        u_prev = u[:, :, k - 1]
        t_step = tau * (k - 0.5)

        for j in range(count_y):
            bi[0] = hx
            bi[-1] = hx - 1
            ci[0] = 0
            ai[-1] = 1
            di[0] = phi1(j * hy, t_step) * hx
            di[-1] = phi2(j * hy, t_step) * hx
            for i in range(1, count_x):
                ai[i] = 1
                bi[i] = -2 * (hx**2) / tau - 2
                ci[i] = 1
                di[i] = (
                    -2 * (hx**2) * u_prev[i, j] / tau
                    - (hx**2)
                    * (u_prev[i, j + 1] - 2 * u_prev[i, j] + u_prev[i, j - 1])
                    / (hy**2)
                    - (hx**2) * f(i * hx, j * hy, t_step)
                )

            ta = thomas_algorithm(ai, bi, ci, di)
            for i in range(count_x + 1):
```

```
            u_1[i, j] = ta[i]
            u_1[i, 0] = phi3(i * hx, t_step)
            u_1[i, -1] = (phi4(i * hx, t_step) - u_1[i, -2] / hy) / (1 - 1 /
hy)

    for j in range(count_y + 1):
        u_1[0, j] = phi1(j * hy, t_step)
        u_1[-1, j] = (phi2(j * hy, t_step) - u_1[-2, j] / hx) / (1 - 1 / hx)

    for i in range(count_x):
        bi[0] = hy
        bi[-1] = hy - 1
        ci[0] = 0
        ai[-1] = 1
        di[0] = phi3(i * hx, k * tau) * hy
        di[-1] = phi4(i * hx, k * tau) * hy

        for j in range(1, count_y):
            ai[j] = 1
            bi[j] = -2 * (hy**2) / tau - 2
            ci[j] = 1
            di[j] = (
                -2 * (hy**2) * u_1[i, j] / tau
                - (hy**2)
                * (u_1[i + 1, j] - 2 * u_1[i, j] + u_1[i - 1, j])
                / (hx**2)
                - (hy**2) * f(i * hx, j * hy, k * tau)
            )

        ta = thomas_algorithm(ai, bi, ci, di)
        for j in range(count_y + 1):
            u_2[i, j] = ta[j]
            u_2[0, j] = phi1(j * hy, k * tau)
            u_2[-1, j] = (phi2(j * hy, k * tau) - u_2[-2, j] / hx) / (1 - 1 /
hx)

    for i in range(count_x + 1):
        u_2[i, 0] = phi3(i * hx, k * tau)
        u_2[i, -1] = (phi4(i * hx, k * tau) - u_2[i, -2] / hy) / (1 - 1 / hy)
        for j in range(count_y + 1):
            u[i, j, k] = u_2[i, j]
```

```
    return u


# схема дробных шагов
def fractional_steps_scheme():
    u = np.zeros((count_x + 1, count_y + 1, t_count + 1))
    u_1 = np.zeros((count_x + 1, count_y + 1))
    u_2 = np.zeros((count_x + 1, count_y + 1))

    ai = np.zeros(count_x + 1)
    bi = np.zeros(count_x + 1)
    ci = np.zeros(count_x + 1)
    di = np.zeros(count_x + 1)

    for i in range(count_x + 1):
        for j in range(count_y + 1):
            u[i, j, 0] = psi(i * hx, j * hy)

    for k in range(1, t_count + 1):
        u_prev = u[:, :, k - 1]
        t_step = tau * (k - 1)

        for j in range(count_y):
            bi[0] = hx
            bi[-1] = hx - 1
            ci[0] = 0
            ai[-1] = 1
            di[0] = phi1(j * hy, t_step) * hx
            di[-1] = phi2(j * hy, t_step) * hx
            for i in range(1, count_x):
                ai[i] = 1
                bi[i] = -(hx**2) / tau - 2
                ci[i] = 1
                di[i] = (
                    -(hx**2) * u_prev[i, j] / tau
                    - (hx**2) * f(i * hx, j * hy, t_step) / 2
                )

            ta = thomas_algorithm(ai, bi, ci, di)
            for i in range(count_x + 1):
                u_1[i, j] = ta[i]
                u_1[i, 0] = phi3(i * hx, t_step)
```

```
            u_1[i, -1] = (phi4(i * hx, t_step) - u_1[i, -2] / hy) / (1 - 1 /
hy)

        for j in range(count_y + 1):
            u_1[0, j] = phi1(j * hy, t_step)
            u_1[-1, j] = (phi2(j * hy, t_step) - u_1[-2, j] / hx) / (1 - 1 / hx)

        for i in range(count_x):
            bi[0] = hy
            bi[-1] = hy - 1
            ci[0] = 0
            ai[-1] = 1
            di[0] = phi3(i * hx, k * tau) * hy
            di[-1] = phi4(i * hx, k * tau) * hy

            for j in range(1, count_y):
                ai[j] = 1
                bi[j] = -(hy**2) / tau - 2
                ci[j] = 1
                di[j] = (
                    -(hy**2) * u_1[i, j] / tau
                    - (hy**2) * f(i * hx, j * hy, k * tau) / 2
                )
            ta = thomas_algorithm(ai, bi, ci, di)
            for j in range(count_y + 1):
                u_2[i, j] = ta[j]
                u_2[0, j] = phi1(j * hy, k * tau)
                u_2[-1, j] = (phi2(j * hy, k * tau) - u_2[-2, j] / hx) / (1 - 1 /
hx)

        for i in range(count_x + 1):
            u_2[i, 0] = phi3(i * hx, k * tau)
            u_2[i, -1] = (phi4(i * hx, k * tau) - u_2[i, -2] / hy) / (1 - 1 / hy)
            for j in range(count_y + 1):
                u[i, j, k] = u_2[i, j]

    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 main():
    u1 = alternative_directions_scheme()
    u2 = fractional_steps_scheme()

    show_result(u1, u2)
    show_inaccuracy(u1, u2)


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

<u>show.py</u>

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

from functions import analytic_solution
from task import count_x, count_y, hx, hy, lx, ly, t_count, t_max,
tau


def show_result(u1, u2):
```

```
    x = np.arange(0, lx + hx, hx)
    y = np.arange(0, ly + hy, hy)
    t = np.arange(0, t_max + tau, tau)
    x_i = 1
    t_i = 2
    fig, ax = plt.subplots(2)
    fig.suptitle("Сравнение численных решений ДУ с
аналитическим")
    fig.set_figheight(15)
    fig.set_figwidth(16)

    for i in range(2):
        ax[i].plot(
            y, analytic_solution(x[x_i], y, t[t_i]), color="red",
label="Analytic"
        )
        ax[i].plot(y, u1[x_i, :, t_i], label="Схема переменных
направлений")
        ax[i].plot(y, u2[x_i, :, t_i], label="Схема дробных шагов")
        ax[i].grid(True)
        ax[i].set_xlabel("y")
        ax[i].set_ylabel("u")
        ax[i].set_title(f"Решения при x= {x[x_i]}, t = {t[t_i]}")
        x_i = count_x
        t_i = t_count

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


def show_inaccuracy(u1, u2):
    x = np.arange(0, lx + hx, hx)
    y = np.arange(0, ly + hy, hy)
    t = np.arange(0, t_max + tau, tau)
    x_i = 1
    t_i = 2
    plt.title(f"Погрешность по y при x= {x[x_i]}, t = {t[t_i]}")

    for i in range(2):
        inaccuracy = []
        if i == 0:
```

```
        u_tfix = u1[:, :, t_i]
    else:
        u_tfix = u2[:, :, t_i]

    for j in range(count_y + 1):
        a = analytic_solution(x, y[j], t[t_i]) - u_tfix[:, j]
        inaccuracy = np.append(inaccuracy, np.linalg.norm(a))
    plt.plot(y, inaccuracy)

  plt.xlabel("y")
  plt.ylabel("error")
  plt.xlim((0, ly))
  plt.grid(True)
  plt.show()
```

task.py

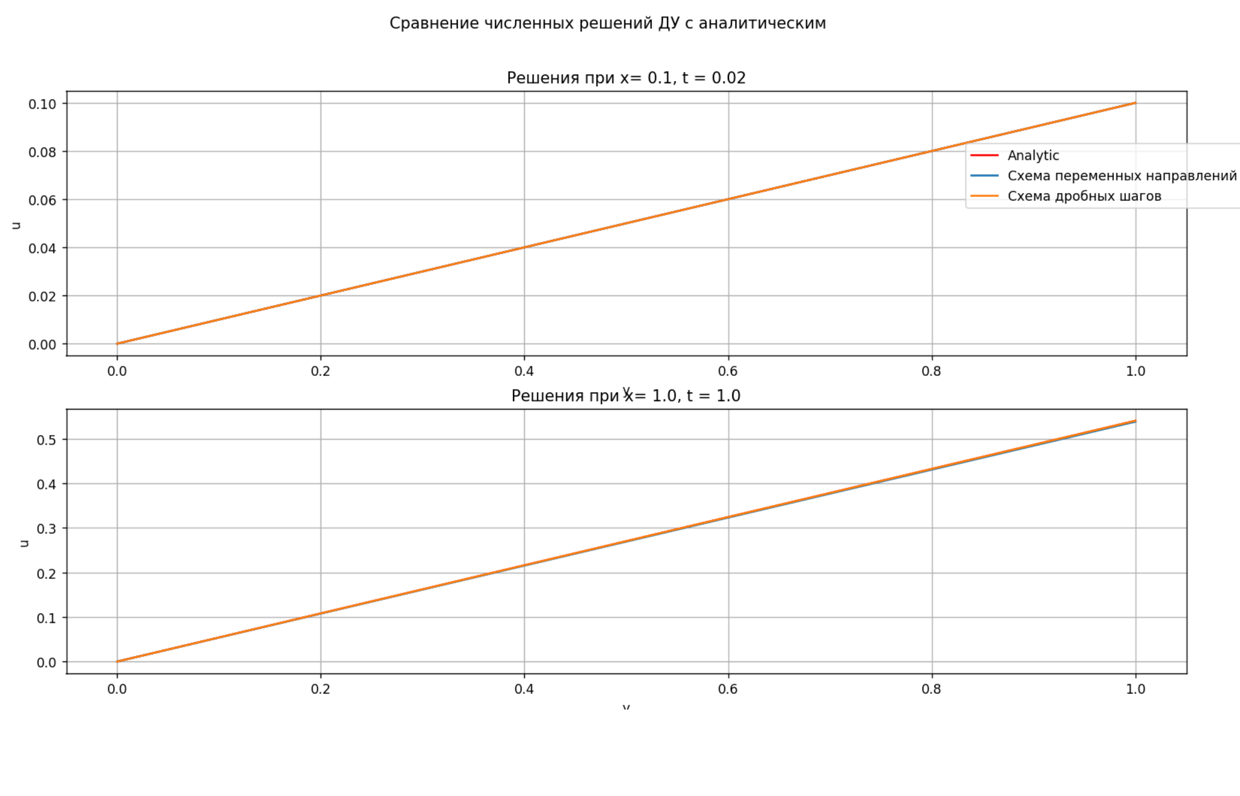
```
t_max = 1
t_count = 100
tau = t_max / t_count
count_x = 10
count_y = 10
lx = 1
ly = 1
hx = lx / count_x
hy = ly / count_y
```

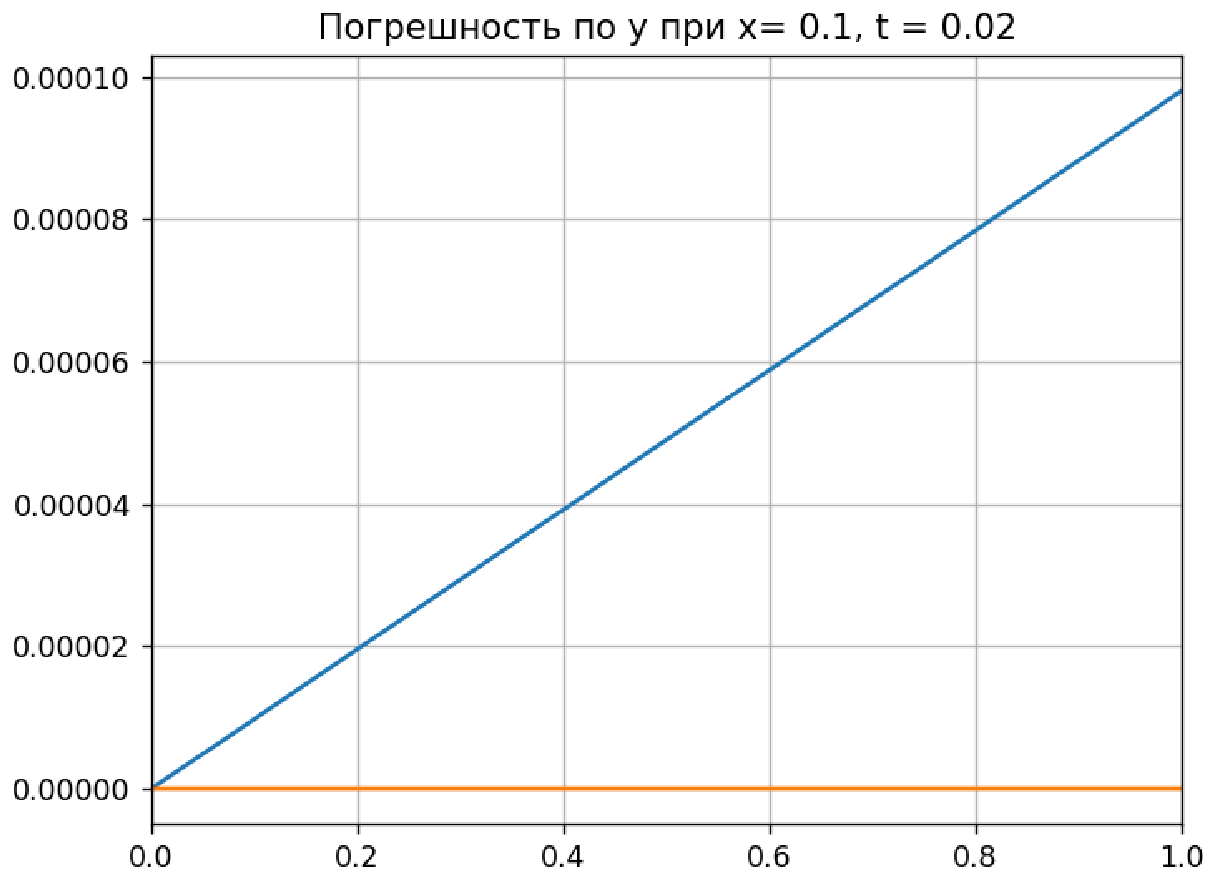
functions.py

```
import numpy as np


def f(x, y, t):
    return -x * y * np.sin(t)


def analytic_solution(x, y, t):
    return x * y * np.cos(t)


def phi1(y, t):
    return 0
```

```
def phi2(y, t):
    return 0


def phi3(x, t):
    return 0


def phi4(x, t):
    return 0


def psi(x, y):
    return x * y
```

Результаты работы





Вывод по лабораторной работе

В ходе выполнения этой лабораторной работы, я получил навыки и знания в численных методах решения дифференциальных уравнений параболического типа. Я успешно применил необходимые численные методы, оценил ошибки в зависимости от шага и времени, построил графики, демонстрирующие соответствующие зависимости этих ошибок, а также создал графики функции U от x.