**Московский авиационный институт**

**(национальный исследовательский университет)**

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №8**

по курсу “Численные методы”

Студент: Пясковский Н.Л.

Группа: М8О-409Б-19

Руководитель: Пивоваров Д.Е.

Оценка: ______________

Москва 2022

**1) Задание**

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 $\tau, h_x, h_y$.

**2) Вариант**

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

**3) Теория**

Общая поставка двумерной задачи параболического типа выглядит следующим образом:

$$\frac{\partial u}{\partial t} = a\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + f(x,y,t), \quad x \in (0, \ell_1), \quad y \in (0, \ell_2), \quad t > 0 \;;$$

$$u(x,0,t) = \varphi_1(x,t),\ x \in [0, \ell_1],\ \ y = 0,\ t > 0 \;;$$

$$u(x, \ell_2, t) = \varphi_2(x,t),\ x \in [0, \ell_1],\ \ y = \ell_2,\ t > 0 \;;$$

$$u(0, y, t) = \varphi_3(y,t),\ x = 0,\ y \in [0, \ell_2], t > 0;$$

$$u(\ell_1, y, t) = \varphi_4(y,t),\ x = \ell_1, y \in [0, \ell_2],\ \ t > 0;$$

$$u(x,y,0) = \psi(x,y),\ x \in [0, \ell_1],\ y \in [0, \ell_2],\ \ t = 0.$$

Далее вводится пространственно-временная сетка с шагами h1, h2, τ соответственно по переменным x, y, t:

$$\omega^{\tau}_{h_1 h_2} = \left\{x_i = ih_1, i = \overline{0, I};\ x_j = jh_2, j = \overline{0, J}\ ;\ t^k = k\tau\ ,\ k = 0,1,2,\ldots\right\}$$

На этой сетке будет аппроксимироваться дифференциальная задача, приведенная выше, методом конечных разностей.

Рассмотрим подробнее методы решения поставленной двумерной задачи параболического типа.

**Метод переменных направлений**

Шаг по времени τ разбивается на два. На каждом временном полуслое первый из пространственных дифференциальных операторов аппроксимируется неявно, а второй явно. На следующем дробном шаге соответственно первый – явно, второй – неявно.

$$\frac{u_{ij}^{k+1/2}-u_{ij}^{k}}{\tau/2}=\frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2}-2u_{ij}^{k+1/2}+u_{i-1j}^{k+1/2}\right)+\frac{a}{h_2^2}\left(u_{ij+1}^{k}-2u_{ij}^{k}+u_{ij-1}^{k}\right)+f_{ij}^{k+1/2},$$

$$\frac{u_{ij}^{k+1}-u_{ij}^{k+1/2}}{\tau/2}=\frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2}-2u_{ij}^{k+1/2}+u_{i-1j}^{k+1/2}\right)+\frac{a}{h_2^2}\left(u_{ij+1}^{k+1}-2u_{ij}^{k+1}+u_{ij-1}^{k+1}\right)+f_{ij}^{k+1/2}$$

Т.е. здесь оператор $a\frac{\partial^2}{\partial x^2}$ на первом временном полуслое аппроксимируется неявно, $a\frac{\partial^2}{\partial y^2}$ – явно.

На втором временном полуслое наоборот.

С помощью скалярных прогонок в количестве J – 1 в направлении переменной x получаем значение $u_{ij}^{k+1/2}$ на первом временном полуслое.

Уже на втором шаге с помощью скалярных прогонок в количестве I – 1 в направлении переменной y получаем значение $u_{ij}^{k+1}$ на следующем временном слое k+1.

К достоинствам метода переменных направлений можно отнести высокую точность, т.к. метод имеет второй порядок точности по времени.

**Метод дробных шагов**

В отличие от МПН (метода переменных направлений) в методе дробных шагов используются только неявная схема аппроксимации.

$$\frac{u_{ij}^{k+1/2}-u_{ij}^{k}}{\tau}=\frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2}-2u_{ij}^{k+1/2}+u_{i-1j}^{k+1/2}\right)+\frac{f_{ij}^{k}}{2},$$

$$\frac{u_{ij}^{k+1}-u_{ij}^{k+1/2}}{\tau}=\frac{a}{h_2^2}\left(u_{i\,j+1}^{k+1}-2u_{ij}^{k+1}+u_{i\,j-1}^{k+1}\right)+\frac{f_{ij}^{k+1}}{2}.$$

С помощью скалярных прогонок в количестве J – 1 в направлении переменной x получаем значение $u_{ij}^{k+1/2}$ на первом временном полуслое.

Уже на втором шаге с помощью скалярных прогонок в количестве I – 1 в направлении переменной y получаем значение $u_{ij}^{k+1}$ на следующем временном слое k+1.

Схема метода дробных шагов имеет первый порядок точности по времени и второй порядок точности по пространству.

Шаблоны приведенных схем выглядят следующим образом:

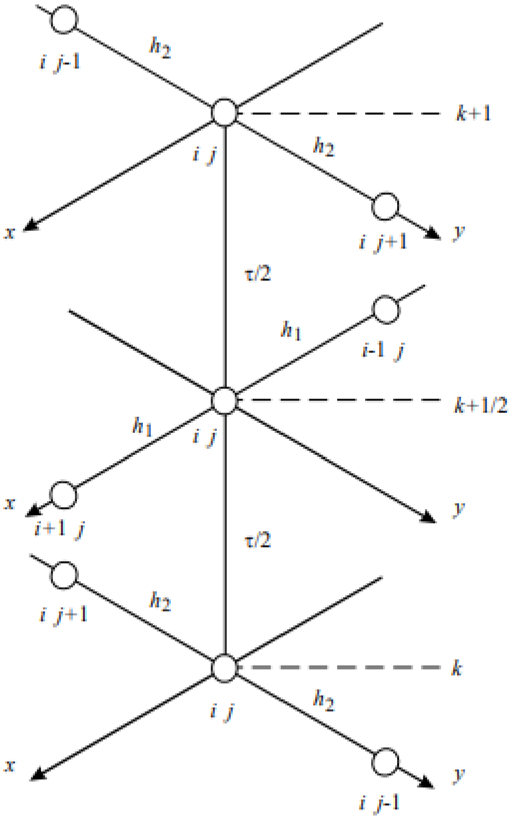



Рис. 5.7 Шаблон схемы метода переменных направлений

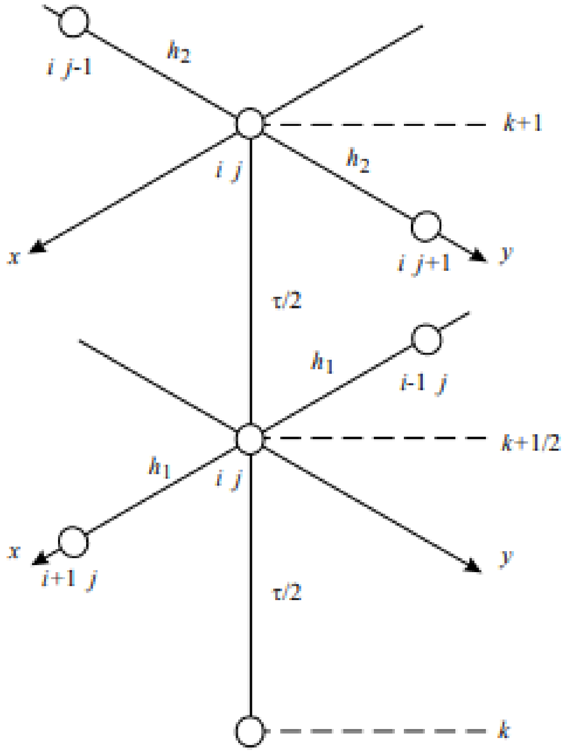



Рис.5.8. Шаблон схемы метода дробных шагов

**4) Программа**

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

T = 1  # верхняя граница времени
K = 100  # количество отрезков t
tau = T / K  # шаг по времени
Nx = 10  # количество отрезков x
Ny = 10  # количество отрезков y
lx = 1  # правая граница координат x
ly = 1  # правая граница координат y
hx = lx / Nx  # шаг сетки по x
hy = ly / Ny  # шаг сетки по y


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

```python
# аналитическое решение
def analytic_solution(x, y, t):
    return x * y * np.cos(t)


# условия
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


# схема переменных направлений
def alternative_directions_scheme():
    u = np.zeros((Nx + 1, Ny + 1, K + 1))
    u_1 = np.zeros((Nx + 1, Ny + 1))
    u_2 = np.zeros((Nx + 1, Ny + 1))

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

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

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

        for j in range(Ny):
            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, Nx):
```

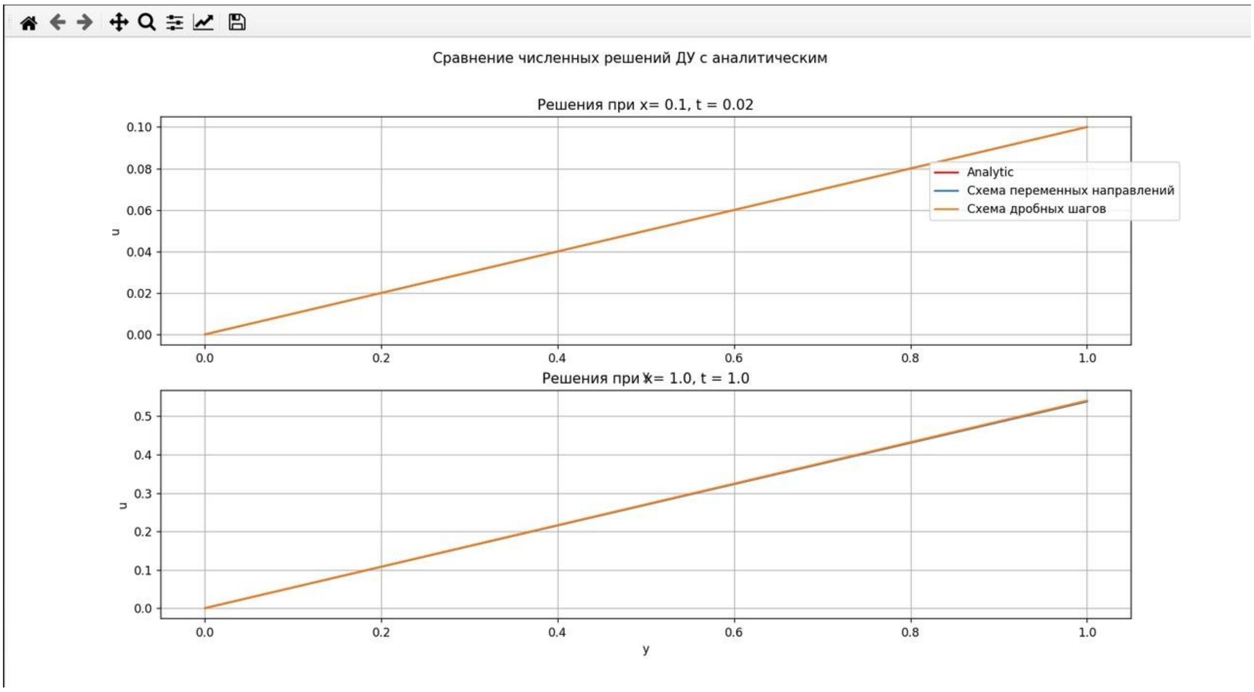
```
            ai[i] = 1
            bi[i] = -2 * (hx ** 2) / tau - 2
            ci[i] = 1
            di[i] = -2 * (hx ** 2) * u_prev[i, j] / tau - (hx ** 2) * (
                    u_prev[i, j + 1] - 2 * u_prev[i, j] + u_prev[i, j - 1]) / (
                        hy ** 2) - (hx ** 2) * f(i * hx, j * hy, t_step)

        ta = thomas_algorithm(ai, bi, ci, di)
        for i in range(Nx + 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(Ny + 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(Nx):
        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, Ny):
            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(Ny + 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(Nx + 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(Ny + 1):
            u[i, j, k] = u_2[i, j]

    return u


# схема дробных шагов
def fractional_steps_scheme():
    u = np.zeros((Nx + 1, Ny + 1, K + 1))
    u_1 = np.zeros((Nx + 1, Ny + 1))
    u_2 = np.zeros((Nx + 1, Ny + 1))
```

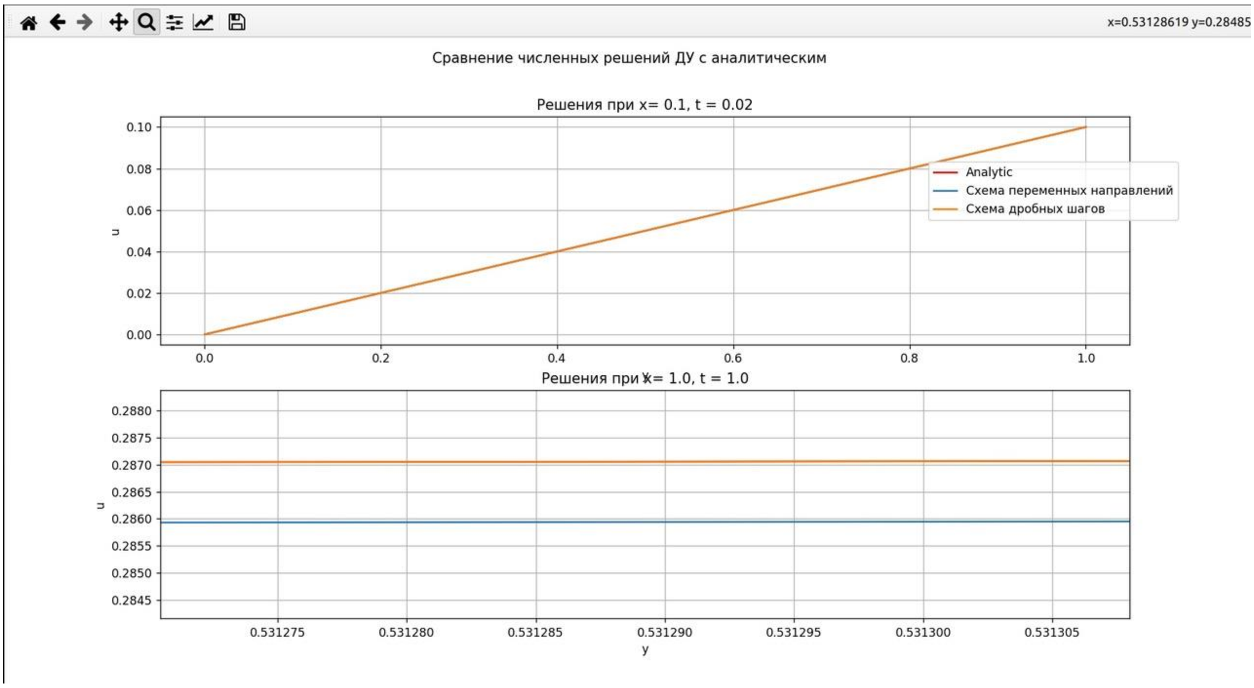
```
ai = np.zeros(Nx + 1)
bi = np.zeros(Nx + 1)
ci = np.zeros(Nx + 1)
di = np.zeros(Nx + 1)

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

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

    for j in range(Ny):
        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, Nx):
            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(Nx + 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(Ny + 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(Nx):
        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, Ny):
            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(Ny + 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)
```

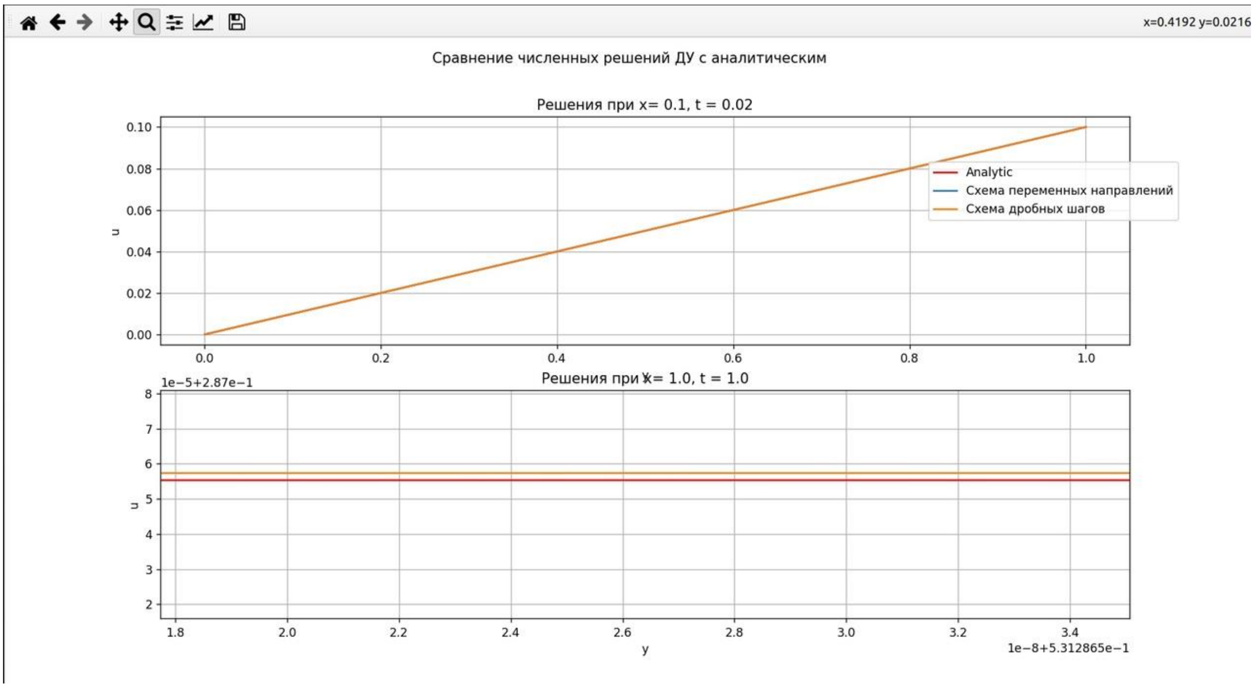
```
        for i in range(Nx + 1):
            u_2[i, 0] = phi3(i * hx, k * tau)
            u_2[i, -1] = (phi4(i * hx, k * tau) - u_2[i, -2] / hy) / (1 - 1 / hy)
            for j in range(Ny + 1):
                u[i, j, k] = u_2[i, j]

    return u


# вывод решения в виде графиков
def show_result(u1, u2):
    x = np.arange(0, lx + hx, hx)
    y = np.arange(0, ly + hy, hy)
    t = np.arange(0, T + 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 = Nx
        t_i = K

    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, ygrid = np.meshgrid(x_axis, y_axis)
    # ax.plot_surface(xgrid, ygrid, analytic_solution(xgrid, ygrid))
    # ax.set(xlabel='x', ylabel='y', zlabel='u')
    # fig.tight_layout()
    # plt.show()


# вывод изменения ошибки со временем
def show_inaccuracy(u1, u2):
    x = np.arange(0, lx + hx, hx)
    y = np.arange(0, ly + hy, hy)
    t = np.arange(0, T + tau, tau)
    x_i = 1
    t_i = 2
    plt.title(f'Погрешность по y при x= {x[x_i]}, t = {t[t_i]}')
```

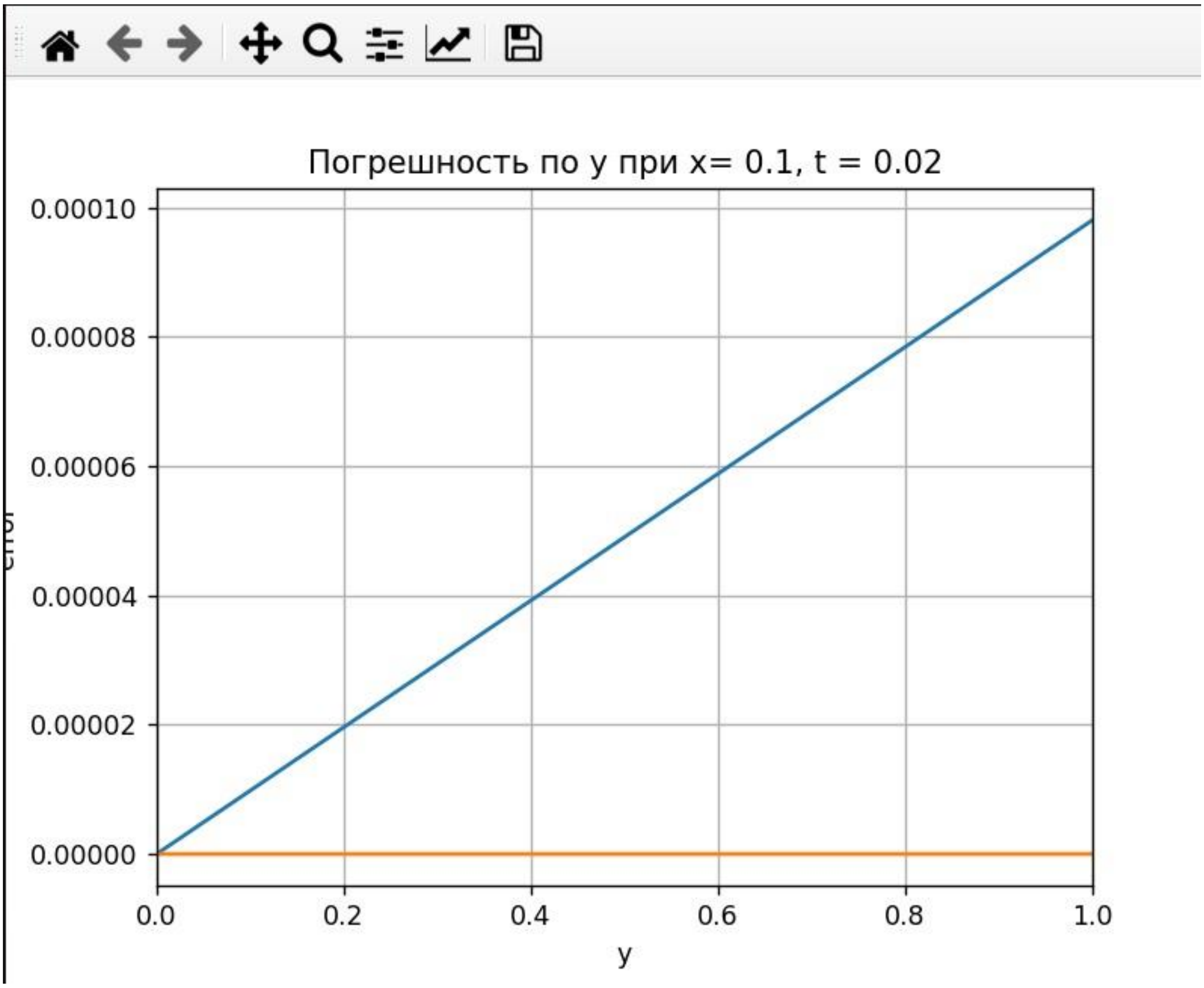
```
    for i in range(2):
        inaccuracy = []
        if i == 0:
            u_tfix = u1[:, :, t_i]
        else:
            u_tfix = u2[:, :, t_i]

        for j in range(Ny + 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()


# метод прогонки
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()
```

# 5) Результат


Сравнение численных решений ДУ с аналитическим
Решения при x= 0.1, t = 0.02
Analytic
Схема переменных направлений
Схема дробных шагов
u
Решения при x= 1.0, t = 1.0
y


x=0.53128619 y=0.28485
Сравнение численных решений ДУ с аналитическим
Решения при x= 0.1, t = 0.02
Analytic
Схема переменных направлений
Схема дробных шагов
u
Решения при x= 1.0, t = 1.0
y


x=0.4192 y=0.0216
Сравнение численных решений ДУ с аналитическим
Решения при x= 0.1, t = 0.02
Analytic
Схема переменных направлений
Схема дробных шагов
u
Решения при x= 1.0, t = 1.0
1e−5+2.87e−1
u
y
1e−8+5.312865e−1


Погрешность по y при x= 0.1, t = 0.02
y