Московский авиационный институт
(национальный исследовательский университет)

Институт №8 "Информационные технологии и прикладная математика"
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №8

по дисциплине:

Численные методы

Вариант №3

Выполнил: студент группы М8О-409Б-20
Чибисов М.Р.
Принял: Пивоваров Е.Д.
Оценка:____

Москва, 2023 г.

1. Задание

Используя схемы переменных направлений и дробных шагов, решить двумерную начальнокраевую 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 h_x, h_y .

Уравнение:

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + a\frac{\partial^2 u}{\partial y^2}, \quad a > 0,$$

$$u(0, y, t) = \cosh(y)\exp(-3at),$$

$$u(\frac{\pi}{4}, y, t) = 0,$$

$$u(x, 0, t) = \cos(2x)\exp(-3at),$$

$$u(x, \ln 2, t) = \frac{5}{4}\cos(2x)\exp(-3at),$$

$$u(x, y, 0) = \cos(2x)\cosh(y).$$

Аналитическое решение:

$$U(x, y, t) = \cos(2x)\cosh(y)\exp(-3at)$$

2. Решение

- Метод переменных направлений:
  - 1 этап:

$$\frac{u_{i,j}^{k+0.5} - u_{i,j}^{k}}{\frac{\tau}{2}} = a\frac{u_{i+1,j}^{k+0.5} - 2u_{i,j}^{k+0.5} + u_{i-1.j}^{k+0.5}}{h_x^2} + a\frac{u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i.j-1}^{k}}{h_y^2}$$

  - 2 этап:

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k+0.5}}{\frac{\tau}{2}} = a\frac{u_{i+1,j}^{k+0.5} - 2u_{i,j}^{k+0.5} + u_{i-1.j}^{k+0.5}}{h_x^2} + a\frac{u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i.j-1}^{k+1}}{h_y^2}$$

- Метод дробных шагов:
  - 1 дробный шаг:

$$\frac{u_{i,j}^{k+0.5} - u_{i,j}^{k}}{\tau} = a\frac{u_{i+1,j}^{k+0.5} - 2u_{i,j}^{k+0.5} + u_{i-1.j}^{k+0.5}}{h_x^2}$$

  - 2 дробный шаг:

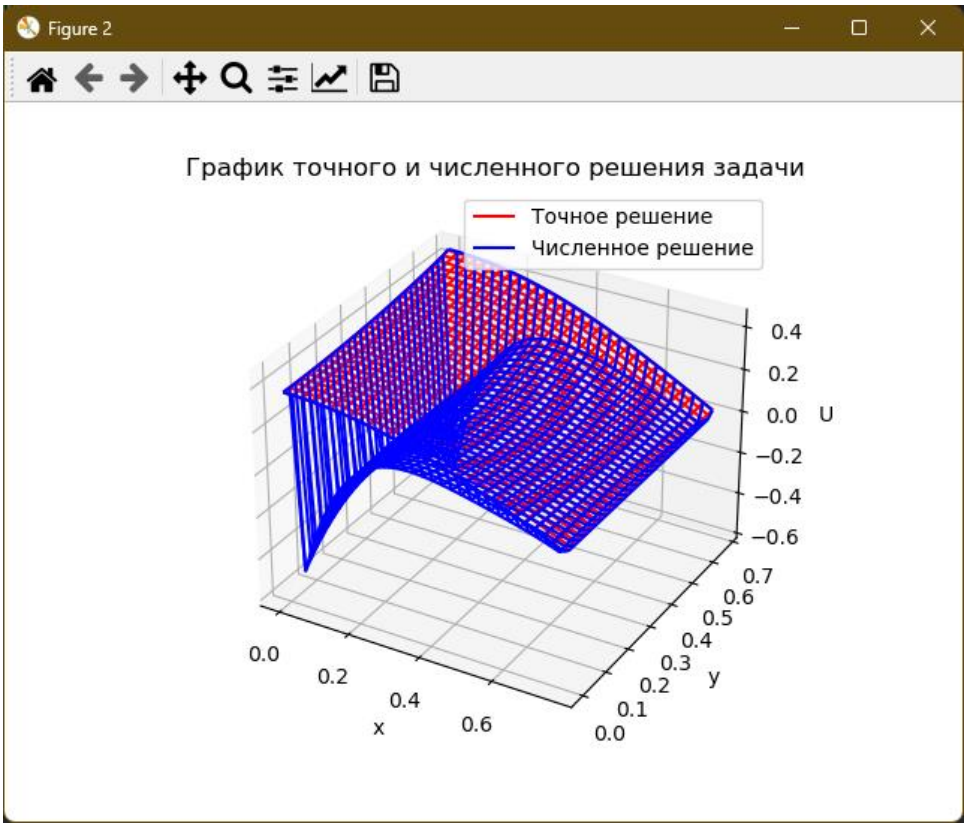
$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k+0.5}}{\tau} = a\frac{u_{i,j+1}^{k+1} - 2u_{i,j}^{k+1} + u_{i.j-1}^{k+1}}{h_y^2}$$

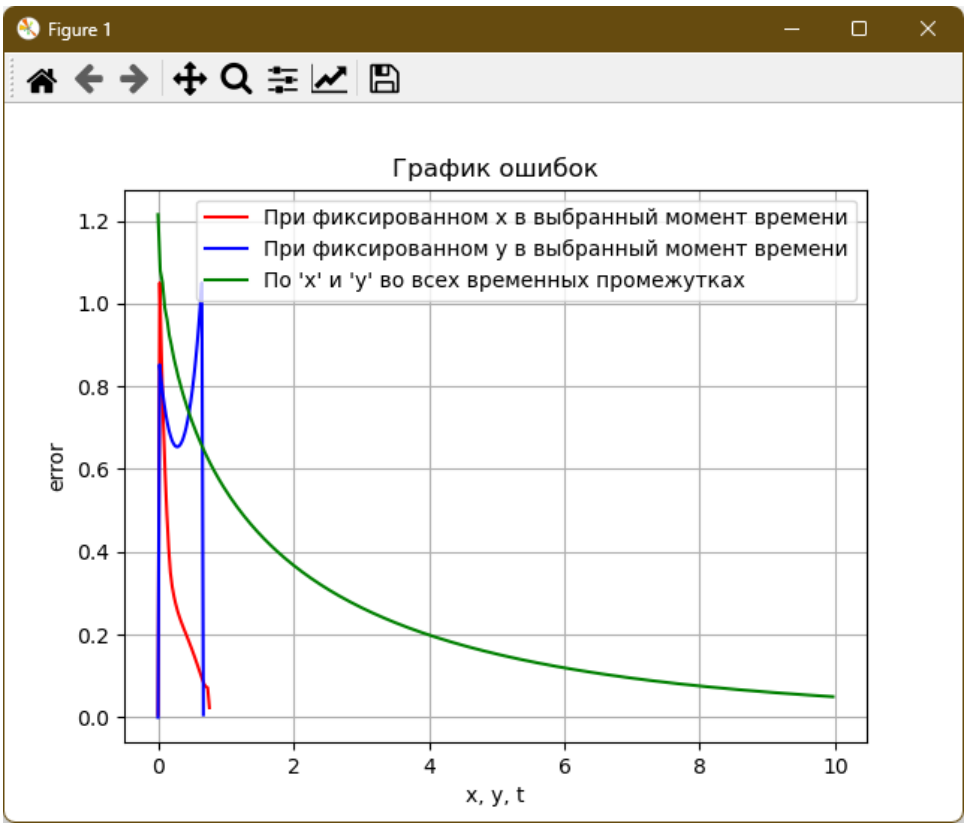
Погрешность между численным и аналитическим решением рассчитывается как абсолютная.

Параметр задачи: a = 5

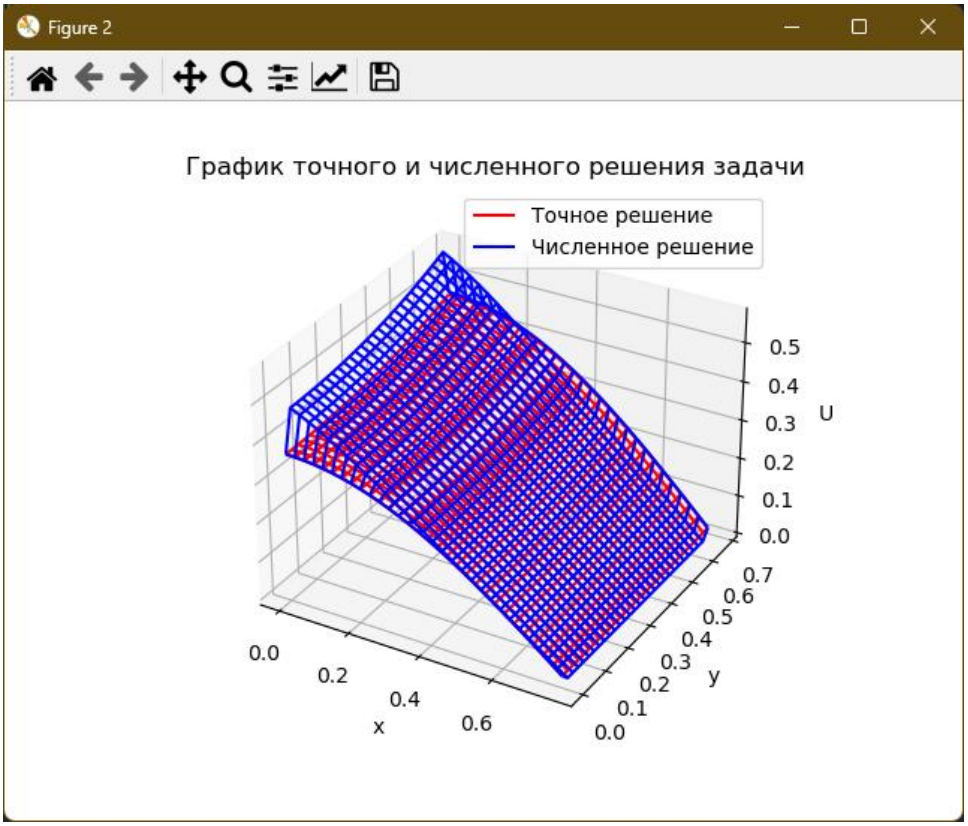
3. Вывод программы

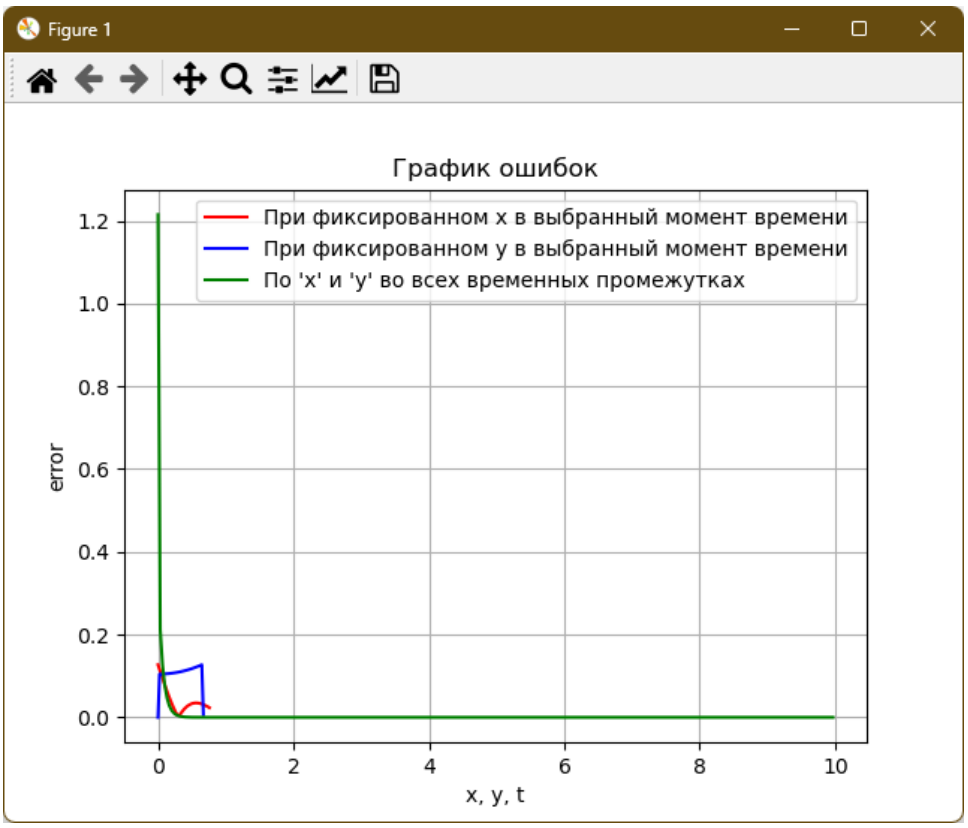
- Метод переменных направлений





- Метод дробных шагов





4. Листинг

```
import numpy as np, matplotlib.pyplot as plt

def analyt_func(ap, x, y, t):
    return np.cos(2*x) * np.cosh(y) * np.exp(-3*ap*t)


def func_border1(ap, y, t):
    return np.cosh(y) * np.exp(-3*ap*t)


def func_border2(ap, y, t):
    return 0


def func_border3(ap, x, t):
    return np.cos(2*x) * np.exp(-3*ap*t)


def func_border4(ap, x, t):
    return 1.25 * np.cos(2*x) * np.exp(-3*ap*t)


def norm(U1, U2):
    max = 0
```

```
    for i in range(U1.shape[0]):
        for j in range(U1.shape[1]):
            if abs(U1[i, j] - U2[i, j]) > max:
                max = abs(U1[i, j] - U2[i, j])

    return max


def run_through(a, b, c, d, s):
    P = np.zeros(s + 1)
    Q = np.zeros(s + 1)

    P[0] = -c[0] / b[0]
    Q[0] = d[0] / b[0]

    k = s - 1
    for i in range(1, s):
        P[i] = -c[i] / (b[i] + a[i] * P[i - 1])
        Q[i] = (d[i] - a[i] * Q[i - 1]) / (b[i] + a[i] * P[i - 1])
    P[k] = 0
    Q[k] = (d[k] - a[k] * Q[k - 1]) / (b[k] + a[k] * P[k - 1])

    x = np.zeros(s)
    x[k] = Q[k]

    for i in range(s - 2, -1, -1):
        x[i] = P[i] * x[i + 1] + Q[i]

    return x


def variable_directions(ap, x, y, hx, hy, K, tau, t):
    U = np.zeros((K, len(x), len(y)))
    sigma_a = (ap * tau) / (2 * hx ** 2)

    for k in range(K):
        for i in range(len(x)):
            U[k, i, 0] = func_border3(ap, x[i], t)
            U[k, i, -1] = func_border4(ap, x[i], t)
    for k in range(K):
        for j in range(len(y)):
            U[k, 0, j] = func_border1(ap, y[j], t)
            U[k, -1, j] = func_border2(ap, y[j], t)

    for k in range(1, K):
        U_temp = np.zeros((len(x), len(y)))
        a = np.zeros(len(y))
        b = np.zeros(len(y))
```

```
        c = np.zeros(len(y))
        d = np.zeros(len(y))
        t += tau / 2
        for i in range(1, len(x) - 1):
            for j in range(1, len(y) - 1):
                a[j] = sigma_a
                b[j] = -1 - 2 * sigma_a
                c[j] = sigma_a
                d[j] = -sigma_a * (U[k - 1, i, j + 1] - 2 * U[k - 1,
i, j] + U[k - 1, i, j - 1]) + U[k - 1, i, j]

            alpha = 0
            betta = 1
            gamma = 0
            delta = 1
            b[0] = betta - alpha / hy
            c[0] = alpha / hy
            d[0] = func_border3(ap, x[i], t - tau / 2)
            a[-1] = -gamma / hy
            b[-1] = delta + gamma / hy
            d[-1] = func_border4(ap, x[i], t - tau / 2)

            u_new = run_through(a, b, c, d, len(d))
            U_temp[i] = u_new
            for j in range(len(y)):
                U_temp[0, j] = func_border1(ap, y[j], t)
                U_temp[-1, j] = func_border2(ap, y[j], t)

        a = np.zeros(len(x))
        b = np.zeros(len(x))
        c = np.zeros(len(x))
        d = np.zeros(len(x))
        t += tau / 2
        for j in range(1, len(y) - 1):
            for i in range(1, len(x) - 1):
                a[i] = sigma_a
                b[i] = -1 - 2 * sigma_a
                c[i] = sigma_a
                d[i] = sigma_a * (U_temp[i + 1, j] - 2 * U_temp[i, j]
+ U_temp[i - 1, j]) + U_temp[i, j]

            alpha = 0
            betta = 1
            gamma = 0
            delta = 1
            b[0] = betta - alpha / hx
            c[0] = alpha / hx
            d[0] = func_border1(ap, y[j], t)
```

```
            a[-1] = -gamma / hx
            b[-1] = delta + gamma / hx
            d[-1] = func_border2(ap, y[j], t)

            u_new = run_through(a, b, c, d, len(d))
            for i in range(len(u_new)):
                U[k, i, j] = u_new[i]
            for i in range(len(x)):
                U[k, i, 0] = func_border3(ap, x[i], t)
                U[k, i, -1] = func_border4(ap, x[i], t)

    return U.transpose()


def fractional_step(ap, x, y, hx, hy, K, tau, t):
    U = np.zeros((K, len(x), len(y)))
    sigma_a = (ap * tau) / hx ** 2

    for k in range(K):
        for i in range(len(x)):
            U[k, i, 0] = func_border3(ap, x[i], t)
            U[k, i, -1] = func_border4(ap, x[i], t)
    for k in range(K):
        for j in range(len(y)):
            U[k, 0, j] = func_border1(ap, y[j], t)
            U[k, -1, j] = func_border2(ap, y[j], t)

    for k in range(1, K):
        U_temp = np.zeros((len(x), len(y)))
        a = np.zeros(len(y))
        b = np.zeros(len(y))
        c = np.zeros(len(y))
        d = np.zeros(len(y))
        t += tau / 2
        for i in range(1, len(x) - 1):
            for j in range(1, len(y) - 1):
                a[j] = sigma_a
                b[j] = -1 - 2 * sigma_a
                c[j] = sigma_a
                d[j] = -U_temp[i, j]

            alpha = 0
            betta = 1
            gamma = 0
            delta = 1
            b[0] = betta - alpha / hy
            c[0] = alpha / hy
            d[0] = func_border3(ap, x[i], t)
```

```
            a[-1] = -gamma / hy
            b[-1] = delta + gamma / hy
            d[-1] = func_border4(ap, x[i], t)

            u_new = run_through(a, b, c, d, len(d))
            U_temp[i] = u_new
            for j in range(len(y)):
                U_temp[0, j] = func_border1(ap, y[j], t)
                U_temp[-1, j] = func_border2(ap, y[j], t)

        a = np.zeros(len(x))
        b = np.zeros(len(x))
        c = np.zeros(len(x))
        d = np.zeros(len(x))
        t += tau / 2
        for j in range(1, len(y) - 1):
            for i in range(1, len(x) - 1):
                a[i] = sigma_a
                b[i] = - 1 - 2 * sigma_a
                c[i] = sigma_a
                d[i] = -U[k - 1][i][j]

            alpha = 0
            betta = 1
            gamma = 0
            delta = 1
            b[0] = betta - alpha / hx
            c[0] = alpha / hx
            d[0] = func_border1(ap, y[j], t - tau / 2)

            a[-1] = - gamma / hx
            b[-1] = delta + gamma / hx
            d[-1] = func_border2(ap, y[j], t - tau / 2)

            u_new = run_through(a, b, c, d, len(d))
            for i in range(len(u_new)):
                U[k, i, j] = u_new[i]
            for i in range(len(x)):
                U[k, i, 0] = func_border3(ap, x[i], t)
                U[k, i, -1] = func_border4(ap, x[i], t)

    return U.transpose()

hx = (np.pi/4) / Nx
hy = np.log(2) / Ny
tau = time / K
Nx = 30
```

```
Ny = 30
K = 300
time = 10
```

5. Вывод

В ходе выполнения лабораторной работы были изучены методы переменных направлений и дробных шагов решений двумерной начально-краевой задачи для дифференциального уравнения параболического типа.Была применена двухточечная аппроксимация краевого условия первого порядка. Были получены результаты в графическом представлении и подсчитаны погрешности для каждого варианта решения.