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 №5

Выполнил: студент группы М8О-409Б-20
Искренкова А.В.
Принял: Пивоваров Е.Д.
Оценка:____

Москва, 2023 г.

# 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 $\tau, h_x, h_y$.

Уравнение:

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

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

$$u\left(\frac{\pi}{2}, y, t\right) = -\sinh(y)\exp(-3at)$$

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

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

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

Аналитическое решение:

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

# 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:

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

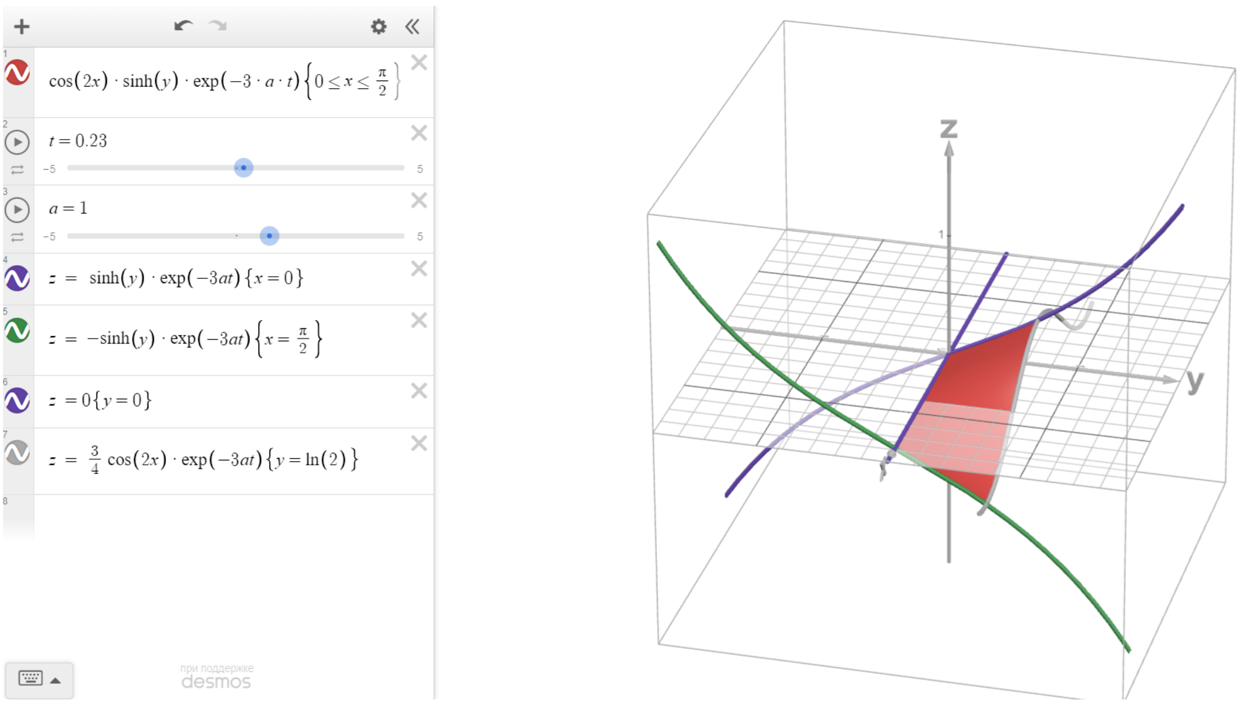
Аппроксимация граничного условия $u_y(x, 0, t) = \cos(2x)\exp(-3at) = \varphi_3(x, 0, t)$:

$$\frac{u_{i,1}^{k} - u_{i,0}^{k}}{h_y} = \varphi_3(x_i, 0, t^k) \ => \ u_{i,0}^{k} = u_{i,1}^{k} - h_y * \varphi_3(x_i, 0, t^k)$$

Погрешность между численным и аналитическим решением рассчитывается как абсолютная.

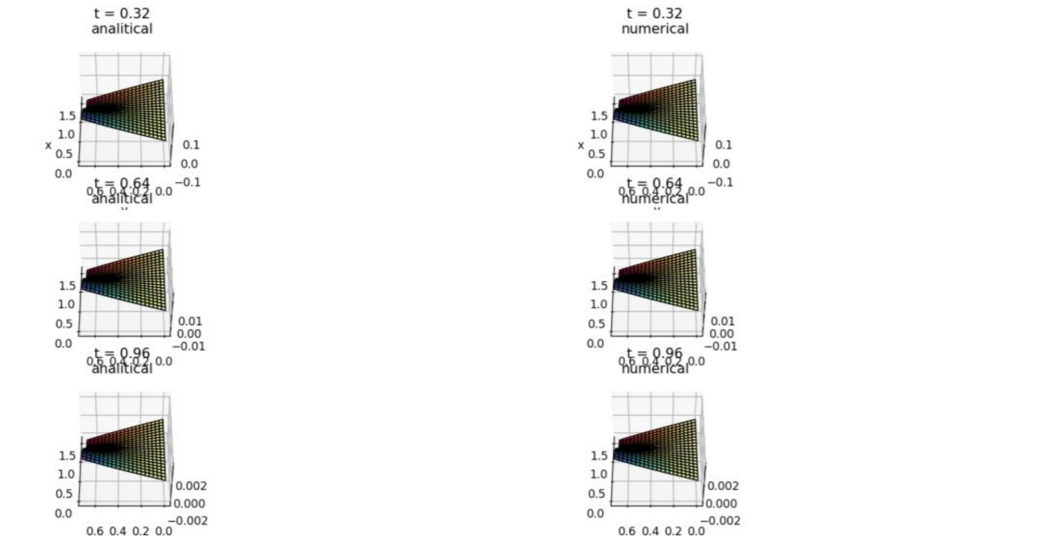
Параметры задачи: a = 2, разбиение по x и по y = 20, T = 1, разбиение по времени = 50

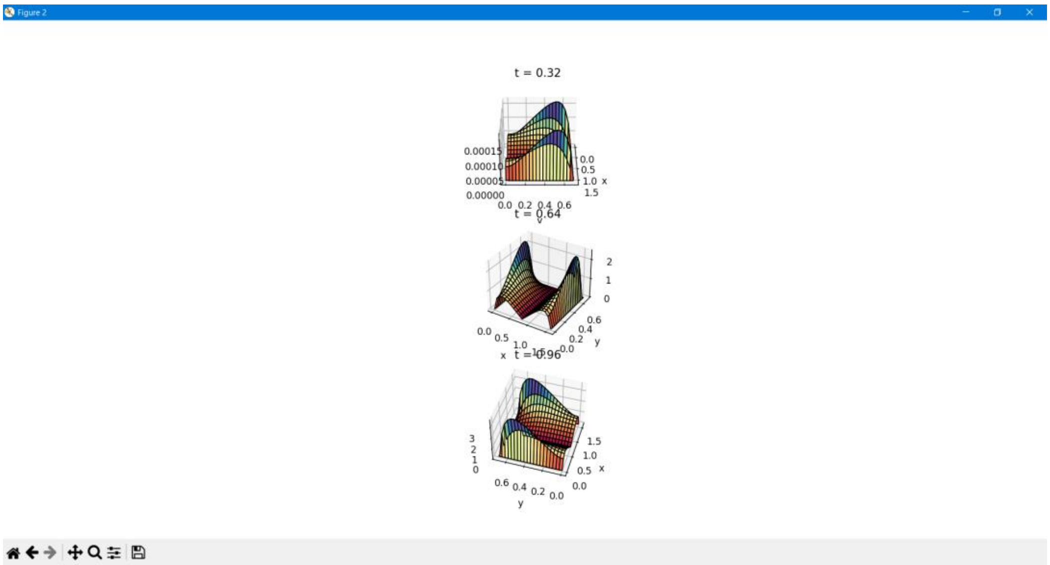
Для наглядности приведен график точного решения в системе Desmos 3D:

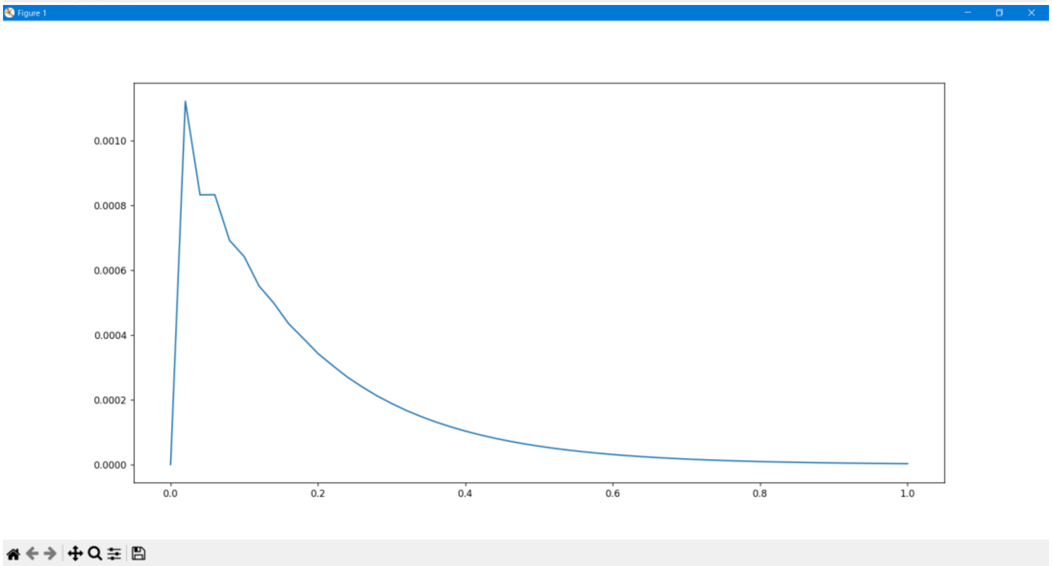



# 3. Вывод программы

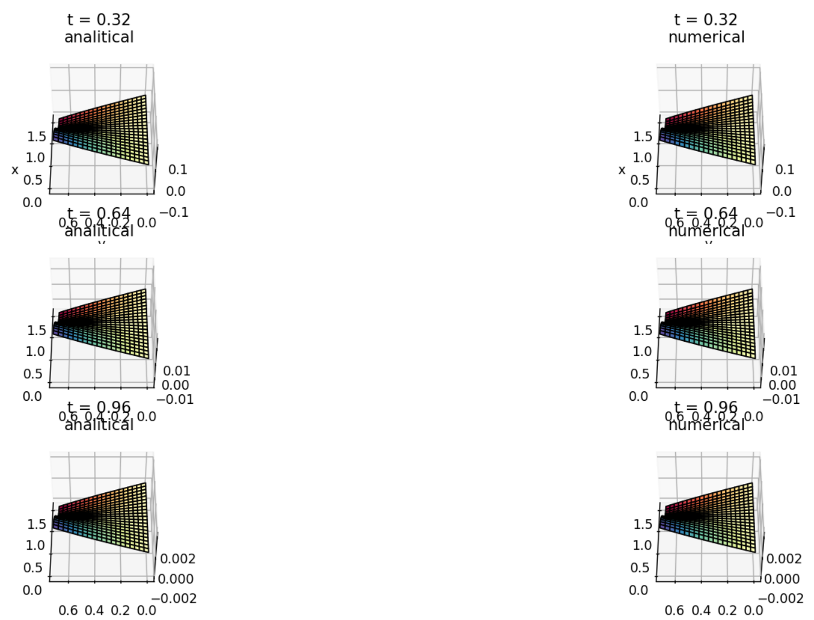
- Метод переменных направлений




- Метод дробных шагов


t = 0.32
analitical
t = 0.32
numerical
t = 0.64
analitical
t = 0.64
numerical
t = 0.96
analitical
t = 0.96
numerical

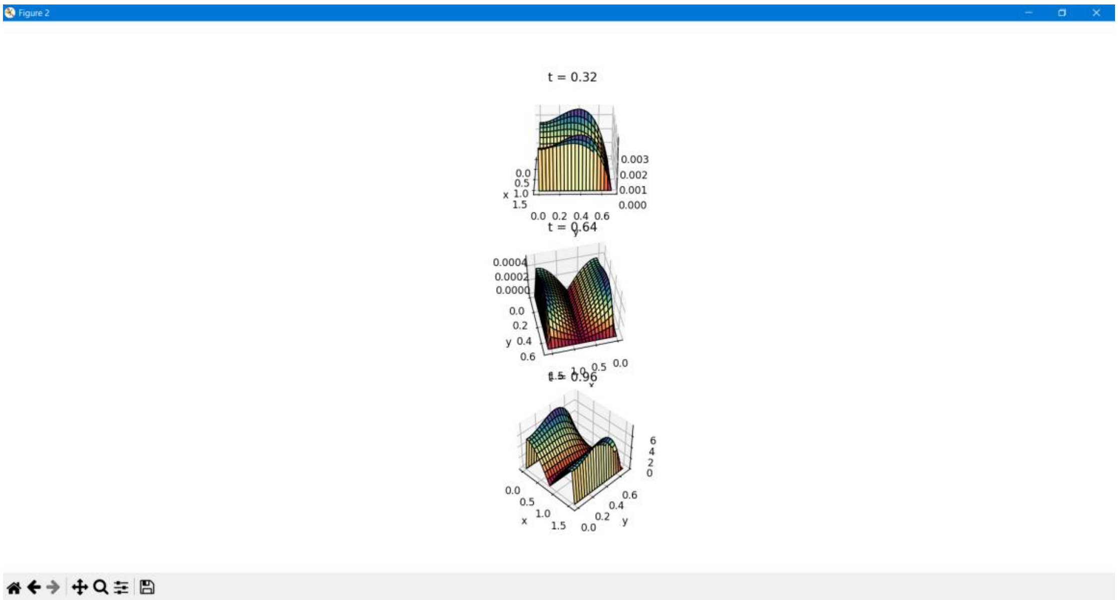


Figure 2
t = 0.32
t = 0.64
t = 0.96

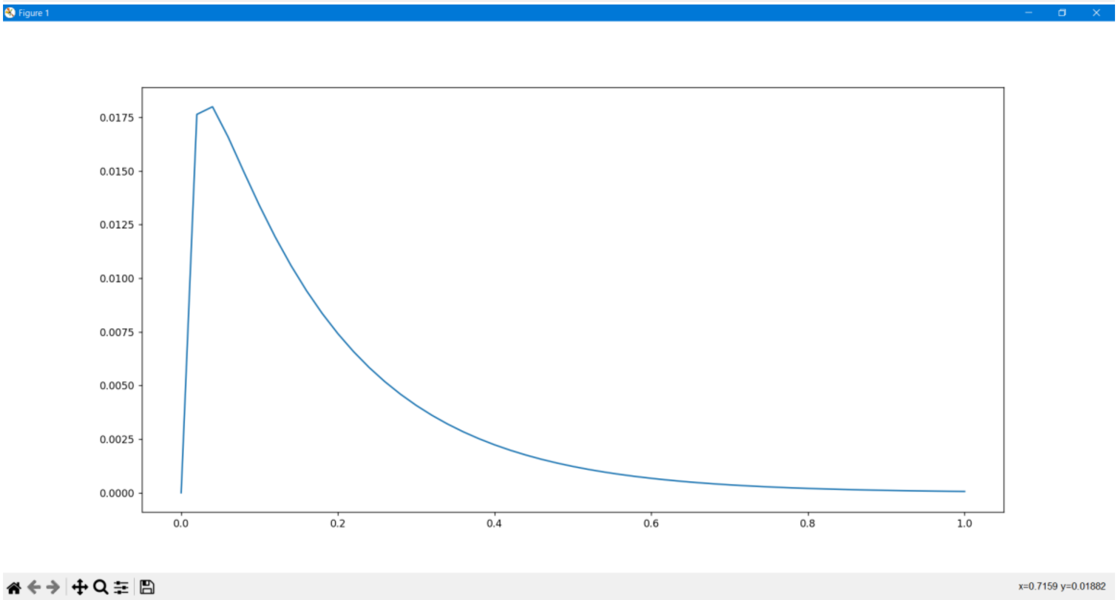


Figure 1
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0.0
0.2
0.4
0.6
0.8
1.0
x=0.7159 y=0.01882

# 4. Листинг

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

# вариант 5

def true_fval(x, y, t, a):
    return np.cos(2*x) * np.sinh(y) * np.exp(-3*a* t)

def phi1(y, t, a):
    return np.sinh(y) * np.exp(-3*a* t)

def phi2(y, t, a):
    return -np.sinh(y) * np.exp(-3*a* t)

def phi3(x, t, a):
    return np.cos(2*x) * np.exp(-3*a* t)

def phi4(x, t, a):
    return np.cos(2*x) * np.exp(-3*a* t)*0.75

def psi(x, y):
    return np.cos(2*x) * np.sinh(y)


def tridiagonal(a, b, c, d):
    n = len(d)
    x = np.zeros(n)
    p = [-c[0] / b[0]]
    q = [d[0] / b[0]]
    for i in range(1, n):
        p.append(-c[i] / (b[i] + a[i] * p[i - 1]))
        q.append((d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]))
    x[-1] = q[-1]
    for i in reversed(range(n - 1)):
        x[i] = p[i] * x[i + 1] + q[i]
    return x

def method1(a, lbx, ubx, nx, lby, uby, ny, T, K):
    hx = (ubx - lbx) / nx
    x = np.arange(lbx, ubx + hx, hx)

    hy = (uby - lby) / ny
    y = np.arange(lby, uby + hy, hy)

    tau = T / K
    t = np.arange(0, T + tau, tau)

    UU = np.zeros((len(x), len(y), len(t)))
```

```python
    for i in range(len(x)):
        for j in range(len(y)):
            UU[i, j, 0] = psi(x[i], y[j])

    for k in range(1, len(t)):
        U1 = np.zeros((len(x), len(y)))
        t2 = t[k] - tau / 2
        # первый дробный шаг
        L = np.zeros((len(x), len(y)))
        L = UU[:, :, k - 1]
        for j in range(len(y) - 1):
            aa = np.zeros(len(x))
            bb = np.zeros(len(x))
            cc = np.zeros(len(x))
            dd = np.zeros(len(x))
            bb[0] = hx
            bb[-1] = hx
            cc[0] = 0
            aa[-1] = 0
            dd[0] = phi1(y[j], t2, a) * hx
            dd[-1] = phi2(y[j], t2, a) * hx
            for i in range(1, len(x) - 1):
                aa[i] = a
                bb[i] = -2 * (hx**2) / tau - 2*a
                cc[i] = a
                dd[i] = -2 * (hx**2) * L[i,j] / tau - a*(hx**2) * (L[i,j+1] - 2*L[i,j] + L[i,j-1]) / (hy**2)
            xx = tridiagonal(aa, bb, cc, dd)
            for i in range(len(x)):
                U1[i, j] = xx[i]
                U1[i, 0] = (phi3(x[i], t2, a) - U1[i, 1] / hy) / (-1/hy)
                U1[i, -1] = phi4(x[i], t2, a)
        for j in range(len(y)):
            U1[0, j] = phi1(y[j], t2, a)
            U1[-1, j] = phi2(y[j], t2, a)
            # второй дробный шаг
        U2 = np.zeros((len(x), len(y)))

        for i in range(len(x) - 1):
            aa = np.zeros(len(x))
            bb = np.zeros(len(x))
            cc = np.zeros(len(x))
            dd = np.zeros(len(x))
            bb[0] = -1
            bb[-1] = hy
            cc[0] = 1
            aa[-1] = 0
            dd[0] = phi3(x[i], t[k], a) * hy
            dd[-1] = phi4(x[i], t[k], a) * hy
            for j in range(1, len(y) - 1):
```

```
                aa[j] = a
                bb[j] = - 2 * (hy**2) / tau - 2*a
                cc[j] = a
                dd[j] = -2* (hy**2) * U1[i,j] / tau - a*(hy**2)*(U1[i + 1,j] - 2*U1[i, j] + U1[i - 1, j]) / (hx**2)
            xx = tridiagonal(aa, bb, cc, dd)
            for j in range(len(y)):
                U2[i, j] = xx[j]
                U2[0, j] = phi1(y[j], t[k], a)
                U2[-1, j] = phi2(y[j], t[k], a)
        for i in range(len(x)):
            U2[i, 0] = (phi3(x[i], t[k], a) - U2[i, 1] / hy) / (-1/ hy)
            U2[i, -1] = phi4(x[i], t[k], a)
            # print(U2)
        for i in range(len(x)):
            for j in range(len(y)):
                UU[i, j, k] = U2[i, j]
    return UU

def method2(a, lbx, ubx, nx, lby, uby, ny, T, K):
    hx = (ubx - lbx) / nx
    x = np.arange(lbx, ubx + hx, hx)

    hy = (uby - lby) / ny
    y = np.arange(lby, uby + hy, hy)

    tau = T / K
    t = np.arange(0, T + tau, tau)

    UU = np.zeros((len(x), len(y), len(t)))
    for i in range(len(x)):
        for j in range(len(y)):
            UU[i, j, 0] = psi(x[i], y[j])

    for k in range(1, len(t)):
        U1 = np.zeros((len(x), len(y)))
        t2 = t[k] - tau / 2
        # первый дробный шаг
        L = np.zeros((len(x), len(y)))
        L = UU[:, :, k - 1]

        for j in range(len(y) - 1):
            aa = np.zeros(len(x))
            bb = np.zeros(len(x))
            cc = np.zeros(len(x))
            dd = np.zeros(len(x))
            bb[0] = hx
            bb[-1] = hx
            cc[0] = 0
            aa[-1] = 0
            dd[0] = phi1(y[j], t2, a) * hx
```

```
            dd[-1] = phi2(y[j], t2, a) * hx
            for i in range(1, len(x) - 1):
                aa[i] = a
                bb[i] = - (hx ** 2) / tau - 2 * a
                cc[i] = a
                dd[i] = -(hx ** 2) * L[i, j] / tau
            xx = tridiagonal(aa, bb, cc, dd)
            for i in range(len(x)):
                U1[i, j] = xx[i]
                U1[i, 0] = (phi3(x[i], t2, a) - U1[i,1] / hy) / (-1 / hy)
                U1[i, -1] = phi4(x[i], t2, a)
        for j in range(len(y)):
            U1[0, j] = phi1(y[j], t2, a)
            U1[-1, j] = phi2(y[j], t2, a)
            # второй дробный шаг
        U2 = np.zeros((len(x), len(y)))

        for i in range(len(x) - 1):
            aa = np.zeros(len(x))
            bb = np.zeros(len(x))
            cc = np.zeros(len(x))
            dd = np.zeros(len(x))
            bb[0] = -1
            bb[-1] = hy
            cc[0] = 1
            aa[-1] = 0
            dd[0] = phi3(x[i], t[k], a) * hy
            dd[-1] = phi4(x[i], t[k], a) * hy
            for j in range(1, len(y) - 1):
                aa[j] = a
                bb[j] = - (hy**2) / tau - 2 * a
                cc[j] = a
                dd[j] = -(hy**2) * U1[i, j] / tau
            xx = tridiagonal(aa, bb, cc, dd)
            for j in range(len(y)):
                U2[i, j] = xx[j]
                U2[0, j] = phi1(y[j], t[k], a)
                U2[-1, j] = phi2(y[j], t[k], a)
        for i in range(len(x)):
            U2[i, 0] = (phi3(x[i], t[k], a) - U2[i, 1] / hy) / (-1 / hy)
            U2[i, -1] = phi4(x[i], t[k], a)
        for i in range(len(x)):
            for j in range(len(y)):
                UU[i, j, k] = U2[i, j]
    return UU


a = 2
```

```
lbx = 0
ubx = np.pi/2
nx = 20
hx = (ubx - lbx) / nx

lby = 0
uby = np.log(2)
ny = 20
hy = (uby - lby) / ny

T = 1
K = 50
tau = T / K

x = np.arange(lbx, ubx + hx, hx)
y = np.arange(lby, uby + hy, hy)
t = np.arange(0, T + tau, tau)

step = len(t) // 3 - 1
yy, xx = np.meshgrid(y, x)
z = []
z.append(true_fval(xx, yy, 0,a))
z.append(true_fval(xx, yy, t[step],a))
z.append(true_fval(xx, yy, t[step * 2],a))
z.append(true_fval(xx, yy, t[step * 3],a))

#u = method1(a, lbx, ubx, nx, lby, uby, ny, T, K)
u = method2(a, lbx, ubx, nx, lby, uby, ny, T, K)

resz = []

for q in range(3):
    resz.append([])
    for i in range(len(x)):
        resz[q].append([])
        for j in range(len(y)):
            resz[q][i].append(u[i][j][step*(q+1)])
resz = np.array(resz)

def plt_res_error(xx, yy, t, z, resz, step):

    error_t = []

    for i in range(len(t)):
        zz = true_fval(xx, yy, t[i],a)
        zz = np.array(zz)
        error_t.append(np.abs(zz - np.array(u[:, :, i])).max(axis =
(0,1)))

    figure = plt.figure(figsize = (20, 10))
```

```python
    plt.plot(t, error_t)

    fig1, ax1 = plt.subplots(3, 1, figsize = (20, 20), subplot_kw = {"projection": "3d"})

    ax1[0].set_title("t = " + str(t[step]))
    ax1[0].view_init(30, 0)
    ax1[0].set(xlabel='x', ylabel='y')
    surf = ax1[0].plot_surface(xx, yy, np.abs(z[1] - resz[0]),
                               edgecolors = ["black"], linewidth = 1,
                               cmap = matplotlib.cm.Spectral, shade = True, antialiased = True)

    ax1[1].set_title("t = " + str(t[step * 2]))
    ax1[1].view_init(30, 0)
    ax1[1].set(xlabel='x', ylabel='y')
    surf = ax1[1].plot_surface(xx, yy, np.abs(z[2] - resz[1]),
                               edgecolors = ["black"], linewidth = 1,
                               cmap = matplotlib.cm.Spectral, shade = True, antialiased = True)

    ax1[2].set_title("t = " + str(t[step * 3]))
    ax1[2].view_init(30, 0)
    ax1[2].set(xlabel='x', ylabel='y')
    surf = ax1[2].plot_surface(xx, yy, np.abs(z[3] - resz[2]),
                               edgecolors = ["black"], linewidth = 1,
                               cmap = matplotlib.cm.Spectral, shade = True, antialiased = True)

    fig, ax = plt.subplots(3, 2, figsize = (20, 20), subplot_kw = {"projection": "3d"})

    ax[0][0].set_title("t = " + str(t[step]) + "\n" + "analitical")
    ax[0][0].view_init(50, 180)
    ax[0][0].set(xlabel='x', ylabel='y')
    surf = ax[0][0].plot_surface(xx, yy, z[1],
                               edgecolors = ["black"], linewidth = 1,
                               cmap = matplotlib.cm.Spectral, shade = True, antialiased = True)

    ax[0][1].set_title("t = " + str(t[step]) + "\n" + "numerical")
    ax[0][1].view_init(50, 180)
    ax[0][1].set(xlabel='x', ylabel='y')
    surf = ax[0][1].plot_surface(xx, yy, resz[0],
                               edgecolors = ["black"], linewidth = 1,
                               cmap = matplotlib.cm.Spectral, shade = True, antialiased = True)

    ax[1][0].set_title("t = " + str(t[step * 2]) + "\n" + "analitical")
    ax[1][0].view_init(50, 180)
    ax[0][1].set(xlabel='x', ylabel='y')
```

```
    surf = ax[1][0].plot_surface(xx, yy, z[2],
                                 edgecolors = ["black"], linewidth = 1,
                                 cmap = matplotlib.cm.Spectral, shade = True, antialiased = True)

    ax[1][1].set_title("t = " + str(t[step * 2]) + "\n" + "numerical")
    ax[1][1].view_init(50, 180)
    ax[0][1].set(xlabel='x', ylabel='y')
    surf = ax[1][1].plot_surface(xx, yy, resz[1],
                                 edgecolors = ["black"], linewidth = 1,
                                 cmap = matplotlib.cm.Spectral, shade = True, antialiased = True)

    ax[2][0].set_title("t = " + str(t[step * 3]) + "\n" + "analitical")
    ax[2][0].view_init(50, 180)
    ax[0][1].set(xlabel='x', ylabel='y')
    surf = ax[2][0].plot_surface(xx, yy, z[3],
                                 edgecolors = ["black"], linewidth = 1,
                                 cmap = matplotlib.cm.Spectral, shade = True, antialiased = True)

    ax[2][1].set_title("t = " + str(t[step * 3]) + "\n" + "numerical")
    ax[2][1].view_init(50, 180)
    ax[0][1].set(xlabel='x', ylabel='y')
    surf = ax[2][1].plot_surface(xx, yy, resz[2],
                                 edgecolors = ["black"], linewidth = 1,
                                 cmap = matplotlib.cm.Spectral, shade = True, antialiased = True)

    plt.show()

def plt_error_t(xx, yy, t, u,a):
    error_t = []

    for i in range(len(t)):
        zz = true_fval(xx, yy, t[i],a)
        zz = np.array(zz)
        error_t.append(np.abs(zz - np.array(u[:, :, i])).max(axis = (0,1)))

    figure = plt.figure(figsize = (20, 10))
    plt.plot(t, error_t)
    plt.show()

plt_res_error(xx, yy, t, z, resz, step)
```

# 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.