Пищик Е.С. М8О-406Б-19

Лабораторная работа №8

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 τ, h_x и h_y.

Вариант №2

Уравнение:

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

Граничные условия:

$$\begin{cases} u(0,\ y,\ t) = cos(\mu_2 y)e^{(-(\mu_1^2+\mu_2^2)at)} \\ u(\frac{\pi}{2}\mu_1,\ y,\ t) = 0 \\ u(x,\ 0,\ t) = cos(\mu_1 x)e^{(-(\mu_1^2+\mu_2^2)at)} \\ u(x,\ \frac{\pi}{2}\mu_2,,\ t) = 0 \\ u(x,\ y,\ 0) = cos(\mu_1 x)cos(\mu_2 y) \end{cases}$$

Аналитическое решение:

$$U(x,y,t) = cos(\mu_1 x)cos(\mu_2 y)e^{(-(\mu_1^2+\mu_2^2)at)}$$

Параметры μ_1 и μ_2:

1. $\mu_1 = 1, \mu_2 = 1$ \
2. $\mu_1 = 2, \mu_2 = 1$ \
3. $\mu_1 = 1, \mu_2 = 2$ \

Файл **tools.py** содержит функции для граничных условий (**u0yt**, **u1yt**, **u0xt**, **u1xt**, **u0xy**) аналитического решения (**analitic_solution**), аналитической сетки (**analitic_grid**), ошибки (**error**).

In [ ]:

```
import numpy as np
```

In [ ]:

```
@np.vectorize
def u0yt(y, t, a, mu1, mu2):
    return np.cos(mu2 * y) * np.exp(-(mu1 * mu1 + mu2 * mu2) * a * t)
```

In [ ]:

```
@np.vectorize
def u1yt(y, t, a, mu1, mu2):
    return 0.0
```

In [ ]:

```
@np.vectorize
def u0xt(x, t, a, mu1, mu2):
    return np.cos(mu1 * x) * np.exp(-(mu1 * mu1 + mu2 * mu2) * a * t)
```

In [ ]:

```
@np.vectorize
def u1xt(x, t, a, mu1, mu2):
    return 0.0
```

In [ ]:

```
@np.vectorize
def u0xy(x, y, a, mu1, mu2):
    return np.cos(mu1 * x) * np.cos(mu2 * y)
```

In [ ]:

```
def analitic_solution(a, x, y, t, mu1, mu2):
    p1 = np.cos(mu1 * x)
    p2 = np.cos(mu2 * y)
    p3 = np.exp(-(mu1 * mu1 + mu2 * mu2) * a * t)
    return p1 * p2 * p3
```

In [ ]:

```
def analitic_grid(a, x, y, t, f):
    grid = np.zeros((len(t), len(y), len(x)))

    for i in range(len(t)):
        for j in range(len(y)):
            for k in range(len(x)):
                grid[i, j, k] = f(a, x[k], y[j], t[i])

    return grid
```

In [ ]:

```
def error(approx, analytic):
    return np.max(np.abs(approx - analytic))
```

Файл **plot.py** содержит функцию для отрисовки графиков (**plot**).

In [ ]:

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

In [ ]:

```
def plot(approx,
         analytic,
         x,
         y,
         tp,
         t,
         filename,
         label):
    fig = plt.figure(figsize=plt.figaspect(0.7))
    plt.subplots_adjust(left=0.1,
                        right=0.9,
                        bottom=0.1,
                        top=0.9,
                        hspace=0.5,
                        wspace=0.5)

    x_mesh, y_mesh = np.meshgrid(x, y)

    for i, ti in enumerate(tp):
        ax = fig.add_subplot(len(tp),
                             2,
                             (len(tp) - 1) * i + 1,
                             projection='3d')

        plt.title(f'{label} t={t[ti]}')

        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_zticks([])

        ax.plot_surface(x_mesh,
                        y_mesh,
                        approx[ti],
                        cmap=cm.BrBG)

        ax = fig.add_subplot(len(tp),
                             2,
                             (len(tp) - 1) * i + 2,
                             projection='3d')

        plt.title(f'analytic t={t[ti]}')

        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_zticks([])

        ax.plot_surface(x_mesh,
                        y_mesh,
                        analytic[ti],
                        cmap=cm.BrBG)

    plt.savefig(filename)
```

Файл **methods.py** - основная точка запуска.

In [ ]:

```
import numpy as np
from functools import partial

from tools import u0yt, u1yt, u0xt, u1xt, u0xy
from tools import analitic_solution, analitic_grid, error

from plot import plot
```

In [ ]:

```
def solve(matrix, target):
    p, q = np.zeros(matrix.shape[1]), np.zeros(matrix.shape[1])

    for i in range(matrix.shape[1]):
        if i == matrix.shape[1] - 1:
            p[i] = 0
        elif i == 0:
            p[i] = -matrix[i, i + 1] / matrix[i, i]
        else:
            p[i] = -matrix[i, i + 1] / \
                (matrix[i, i] + matrix[i, i - 1] * p[i - 1])

        if i == 0:
            q[i] = target[i] / matrix[i, i]
        else:
            q[i] = ((target[i] - matrix[i, i - 1] * q[i - 1]) /
                    (matrix[i, i] + matrix[i, i - 1] * p[i - 1]))

    res = np.zeros(matrix.shape[1] + 1)

    for i in range(matrix.shape[1] - 1, -1, -1):
        res[i] = p[i] * res[i + 1] + q[i]

    return res[:-1]
```

In [ ]:

```
def _common(u0yt,
            u1yt,
            u0xt,
            u1xt,
            u0xy,
            a,
            hx,
            hy,
            tau,
            xlhs,
            xrhs,
            ylhs,
            yrhs,
            tlhs,
            trhs,
            coef):
    x = np.arange(xlhs, xrhs + hx / 2.0, step=hx)
    y = np.arange(ylhs, yrhs + hy / 2.0, step=hy)
    t = np.arange(tlhs, trhs + tau / 4.0, step=tau / 2.0)
    u = np.zeros((len(t), len(y), len(x)))

    for i, yi in enumerate(y):
        for j, xj in enumerate(x):
            u[0, i, j] = u0xy(xj, yi)

    for i, ti in enumerate(t):
        for j, yj in enumerate(y):
            u[i, j, 0] = u0yt(yj, ti)
            u[i, j, -1] = u1yt(yj, ti)

    for i, ti in enumerate(t):
        for j, xj in enumerate(x):
            u[i, 0, j] = u0xt(xj, ti)
            u[i, -1, j] = u1xt(xj, ti)

    for k in range(0, len(t) - 2, 2):
        for i in range(1, len(y) - 1):
            matrix = np.zeros((len(x) - 2, len(x) - 2))

            m1 = -(2.0 * a * tau * hy * hy + (1.0 + coef) * hx * hx * hy * hy)
            m2 = a * tau * hy * hy

            matrix[0] += np.array([m1, m2] + [0.0] * (len(matrix) - 2))

            t1 = -a * tau * hx * hx * coef * u[k, i - 1, 1]

            t2_add1 = (1.0 + coef) * hx * hx * hy * hy
            t2_add2 = 2.0 * a * tau * hx * hx * coef
            t2 = (t2_add1 - t2_add2) * u[k, i, 1]

            t3 = a * tau * hx * hx * coef * u[k, i + 1, 1]
            t4 = a * tau * hy * hy * u[k + 1, i, 0]

            target = [t1 - t2 - t3 - t4]

            for j in range(1, len(matrix) - 1):
                m1 = [0.0] * (j - 1)

                m21 = a * tau * hy * hy
```

```
            m22 = -(2.0 * a * tau * hy * hy +
                    (1.0 + coef) * hx * hx * hy * hy)
            m23 = a * tau * hy * hy
            m2 = [m21, m22, m23]

            m3 = [0.0] * (len(matrix) - j - 2)

            matrix[j] += np.array(m1 + m2 + m3)

            t1 = -a * tau * hx * hx * coef * u[k, i - 1, j + 1]

            t2_add1 = (1.0 + coef) * hx * hx * hy * hy
            t2_add2 = 2.0 * a * tau * hx * hx * coef
            t2 = (t2_add1 - t2_add2) * u[k, i, j + 1]

            t3 = a * tau * hx * hx * coef * u[k, i + 1, j + 1]

            target += [t1 - t2 - t3]

        m1 = [0.0] * (len(matrix) - 2)

        m21 = a * tau * hy * hy
        m22 = -(2.0 * a * tau * hy * hy + (1.0 + coef) * hx * hx * hy * hy)
        m2 = [m21, m22]

        matrix[-1] += np.array(m1 + m2)

        t1 = -a * tau * hx * hx * coef * u[k, i - 1, -2]

        t2_add1 = (1.0 + coef) * hx * hx * hy * hy
        t2_add2 = 2.0 * a * tau * hx * hx * coef
        t2 = (t2_add1 - t2_add2) * u[k, i, -2]

        t3 = a * tau * hx * hx * coef * u[k, i + 1, -2]
        t4 = a * tau * hy * hy * u[k + 1, i, -1]

        target += [t1 - t2 - t3 - t4]

        sol = solve(matrix, np.array(target)).tolist()
        u[k + 1, i] += np.array([0.0] + sol + [0.0])

    for j in range(1, len(x) - 1):
        matrix = np.zeros((len(y) - 2, len(y) - 2))

        m11 = -(2.0 * a * tau * hx * hx + (1.0 + coef) * hx * hx * hy * hy)
        m12 = a * tau * hx * hx
        m1 = [m11, m12]

        m2 = [0.0] * (len(matrix) - 2)

        matrix[0] += np.array(m1 + m2)

        t1 = -a * tau * hy * hy * coef * u[k + 1, 1, j - 1]

        t2_add1 = (1.0 + coef) * hx * hx * hy * hy
        t2_add2 = 2.0 * a * tau * hy * hy * coef
        t2 = (t2_add1 - t2_add2) * u[k + 1, 1, j]

        t3 = a * tau * hy * hy * coef * u[k + 1, 1, j + 1]
        t4 = a * tau * hx * hx * u[k + 2, 0, j]
```

```
            target = [t1 - t2 - t3 - t4]

            for i in range(1, len(matrix) - 1):
                m1 = [0.0] * (i - 1)

                m21 = a * tau * hx * hx
                m22 = -(2.0 * a * tau * hx * hx +
                        (1.0 + coef) * hx * hx * hy * hy)
                m23 = a * tau * hx * hx
                m2 = [m21, m22, m23]

                m3 = [0.0] * (len(matrix) - i - 2)

                matrix[i] += np.array(m1 + m2 + m3)

                t1 = -a * tau * hy * hy * coef * u[k + 1, i + 1, j - 1]

                t2_add1 = (1.0 + coef) * hx * hx * hy * hy
                t2_add2 = 2.0 * a * tau * hy * hy * coef
                t2 = (t2_add1 - t2_add2) * u[k + 1, i + 1, j]

                t3 = a * tau * hy * hy * coef * u[k + 1, i + 1, j + 1]

                target += [t1 - t2 - t3]

            m1 = [0.0] * (len(matrix) - 2)

            m21 = a * tau * hx * hx
            m22 = -(2.0 * a * tau * hx * hx + (1.0 + coef) * hx * hx * hy * hy)
            m2 = [m21, m22]

            matrix[-1] += np.array(m1 + m2)

            t1 = -a * tau * hy * hy * coef * u[k + 1, -2, j - 1]

            t2_add1 = (1.0 + coef) * hx * hx * hy * hy
            t2_add2 = 2.0 * a * tau * hy * hy * coef
            t2 = (t2_add1 - t2_add2) * u[k + 1, -2, j]

            t3 = a * tau * hy * hy * coef * u[k + 1, -2, j + 1]
            t4 = a * tau * hx * hx * u[k + 2, -1, j]

            target += [t1 - t2 - t3 - t4]

            sol = solve(matrix, np.array(target)).tolist()
            u[k + 2, :, j] += np.array([0.0] + sol + [0.0])

    return u[::2]
```

In [ ]:

```
def fract_steps(*args):
    return _common(coef=0.0, *args)
```

In [ ]:

```
def alter_dirs(*args):
    return _common(coef=1.0, *args)
```

In [ ]:

```
def _main(params):
    a = params['a']
    hx, hy = params['hx'], params['hy']
    tau = params['tau']

    u0yt = params['u0yt']
    u1yt = params['u1yt']
    u0xt = params['u0xt']
    u1xt = params['u1xt']
    u0xy = params['u0xy']

    xlhs, xrhs = params['xlhs'], params['xrhs']
    ylhs, yrhs = params['ylhs'], params['yrhs']
    tlhs, trhs = params['tlhs'], params['trhs']

    mu = params['mu']
    method = params['method']

    x = np.arange(xlhs, mu[0] * np.pi / 2.0 + hx / 2.0, step=hx)
    y = np.arange(ylhs, mu[1] * np.pi / 2.0 + hy / 2.0, step=hy)
    t = np.arange(tlhs, trhs + tau / 2.0, step=tau)

    analytic = analitic_grid(a,
                             x,
                             y,
                             t,
                             partial(analitic_solution,
                                     mu1=mu[0],
                                     mu2=mu[1]))

    approx = method(partial(u0yt, a=a, mu1=mu[0], mu2=mu[1]),
                    partial(u1yt, a=a, mu1=mu[0], mu2=mu[1]),
                    partial(u0xt, a=a, mu1=mu[0], mu2=mu[1]),
                    partial(u1xt, a=a, mu1=mu[0], mu2=mu[1]),
                    partial(u0xy, a=a, mu1=mu[0], mu2=mu[1]),
                    a,
                    hx,
                    hy,
                    tau,
                    xlhs,
                    xrhs,
                    ylhs,
                    yrhs,
                    tlhs,
                    trhs)

    err = error(approx, analytic)
    method_str = str(method).split()[1]

    print(f'method={method_str}, mu1={mu[0]}, mu2={mu[1]}, error={err:.3f}')

    filename = f'images/{method_str}_mu1={mu[0]}_mu2={mu[1]}.jpg'
    tp = [0, len(t) // 2, len(t) - 1]

    plot(approx, analytic, x, y, tp, t, filename, method_str)
```

In [ ]:

```
def main(method):
    mu1, mu2, mu3 = (1.0, 1.0), (2.0, 1.0), (1.0, 2.0)

    params1 = {
        "u0yt": u0yt,
        "u1yt": u1yt,
        "u0xt": u0xt,
        "u1xt": u1xt,
        "u0xy": u0xy,
        "a": 1.0,
        "hx": 0.157,
        "hy": 0.157,
        "tau": 0.1,
        "xlhs": 0.0,
        "xrhs": mu1[0] * np.pi / 2.0,
        "ylhs": 0.0,
        "yrhs": mu1[1] * np.pi / 2.0,
        "tlhs": 0.0,
        "trhs": 1.0,
        "mu": mu1,
        "method": method
    }

    params2 = {
        "u0yt": u0yt,
        "u1yt": u1yt,
        "u0xt": u0xt,
        "u1xt": u1xt,
        "u0xy": u0xy,
        "a": 1.0,
        "hx": 0.157,
        "hy": 0.157,
        "tau": 0.1,
        "xlhs": 0.0,
        "xrhs": mu2[0] * np.pi / 2.0,
        "ylhs": 0.0,
        "yrhs": mu2[1] * np.pi / 2.0,
        "tlhs": 0.0,
        "trhs": 1.0,
        "mu": mu2,
        "method": method
    }

    params3 = {
        "u0yt": u0yt,
        "u1yt": u1yt,
        "u0xt": u0xt,
        "u1xt": u1xt,
        "u0xy": u0xy,
        "a": 1.0,
        "hx": 0.157,
        "hy": 0.157,
        "tau": 0.1,
        "xlhs": 0.0,
        "xrhs": mu3[0] * np.pi / 2.0,
        "ylhs": 0.0,
        "yrhs": mu3[1] * np.pi / 2.0,
        "tlhs": 0.0,
        "trhs": 1.0,
```

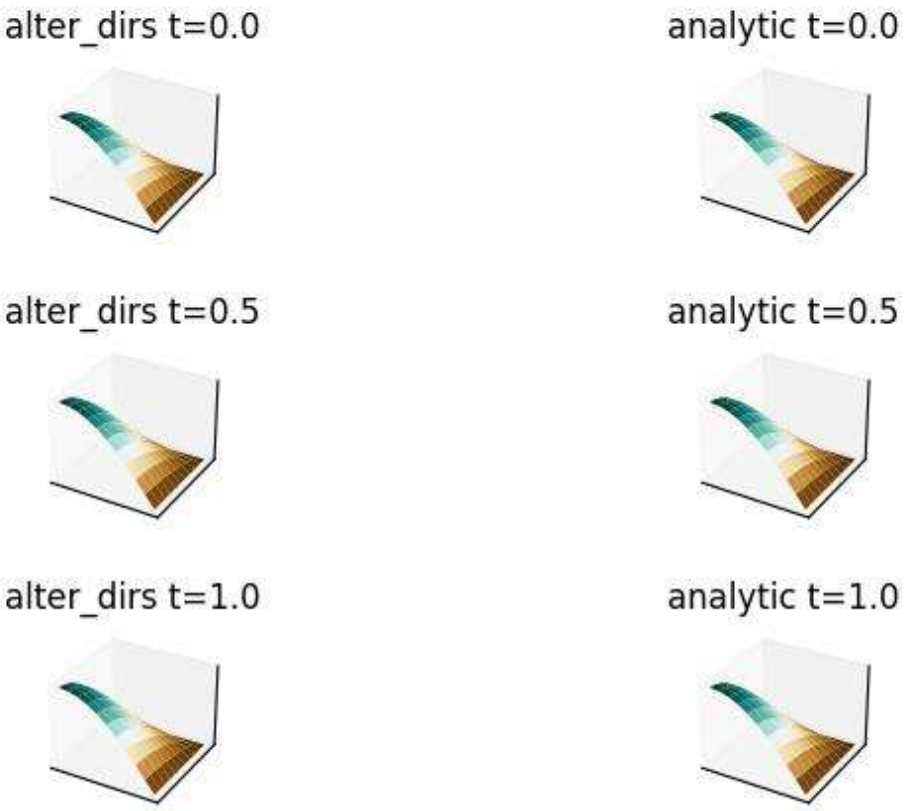
```
        "mu": mu3,
        "method": method
    }

    _main(params=params1)
    _main(params=params2)
    _main(params=params3)
```

In [ ]:

```
if __name__ == '__main__':
    main(fract_steps)
    main(alter_dirs)
```

Графики.

$\mu_1 = 1, \mu_2 = 1$



$\mu_1 = 1, \mu_2 = 2$

alter_dirs t=0.0

analytic t=0.0

alter_dirs t=0.5

analytic t=0.5

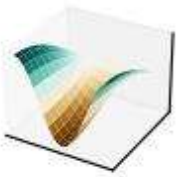
alter_dirs t=1.0

analytic t=1.0

$\mu_1 = 2, \mu_2 = 1$

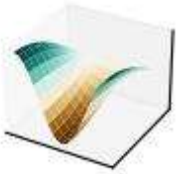
alter_dirs t=0.0

analytic t=0.0

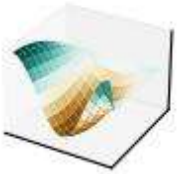


alter_dirs t=0.5

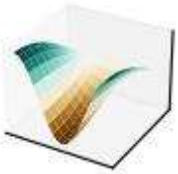
analytic t=0.5



alter_dirs t=1.0



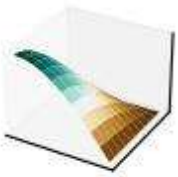
analytic t=1.0



$\mu_1 = 1, \mu_2 = 1$

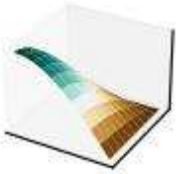
fract_steps t=0.0

analytic t=0.0



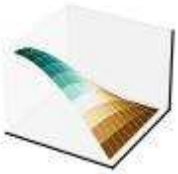
fract_steps t=0.5

analytic t=0.5



fract_steps t=1.0

analytic t=1.0



$\mu_1 = 1, \mu_2 = 2$

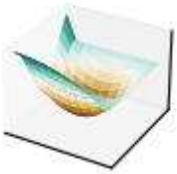
fract_steps t=0.0

analytic t=0.0

fract_steps t=0.5

analytic t=0.5

fract_steps t=1.0

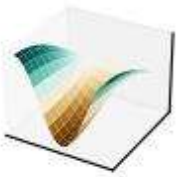


analytic t=1.0

$\mu_1 = 2, \mu_2 = 1$

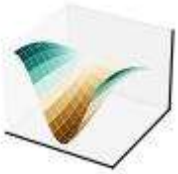
fract_steps t=0.0

analytic t=0.0

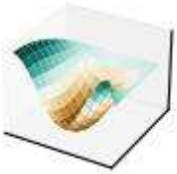


fract_steps t=0.5

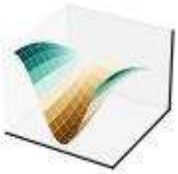
analytic t=0.5



fract_steps t=1.0



analytic t=1.0



In [ ]: