Пищик Е.С. М8О-406Б-19

Лабораторная работа №7

Задание: Решить краевую задачу для дифференциального уравнения эллиптического типа. Аппроксимацию уравнения произвести с использованием центрально-разностной схемы. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x, y)$. Исследовать зависимость погрешности от сеточных параметров h_x и h_y.

Вариант №2

Уравнение:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Граничные условия:

$$\begin{cases} u_x(0,\ y) = 0 \\ u(1,\ y) = 1 - y^2 \\ u_y(x,\ 0) = 0 \\ u(x,\ 1) = x^2 - 1 \end{cases}$$

Аналитическое решение:

$$U(x, y) = x^2 - y^2$$

Файл **tools.py** содержит функции для граничных условий (**ux0y**, **u1y**, **uyx0**, **ux1**) аналитического решения (**analitic_solution**), аналитической сетки (**analitic_grid**), ошибки (**error**).

In [ ]:

```
import numpy as np
```

In [ ]:

```
def ux0y(y):
    return np.zeros(len(y))
```

In [ ]:

```
def u1y(y):
    return 1.0 - y * y
```

In [ ]:

```
def uyx0(x):
    return np.zeros(len(x))
```

In [ ]:

```
def ux1(x):
    return x * x - 1.0
```

In [ ]:

```
def analitic_solution(x, y):
    return x * x - y * y
```

In [ ]:

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

    for i in range(len(y)):
        for j in range(len(x)):
            grid[i, j] = analitic_solution(x[j], y[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, filename):
    fig = plt.figure(figsize=plt.figaspect(0.7))
    x_mesh, y_mesh = np.meshgrid(x, y)

    ax = fig.add_subplot(1, 2, 1, projection='3d')

    plt.title('approx')
    ax.set_xlabel('x', fontsize=14)
    ax.set_ylabel('y', fontsize=14)
    ax.set_zlabel('u', fontsize=14)

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

    ax = fig.add_subplot(1, 2, 2, projection='3d')

    plt.title('analytic')
    ax.set_xlabel('x', fontsize=14)
    ax.set_ylabel('y', fontsize=14)
    ax.set_zlabel('u', fontsize=14)

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

    plt.savefig(filename)
```

Файл **libman.py** содержит реализацию метода Либмана.

In [ ]:

```
import numpy as np
from copy import deepcopy

from tools import ux0y, u1y, uyx0, ux1, analitic_grid, error
from plot import plot
```

In [ ]:

```
def libman(params, x, y):
    h = params['h']
    eps = params['eps']
    epochs = params['epochs']
    diverge = params['diverge']

    u = np.zeros((len(y), len(x)))
    prev_u = np.zeros((len(y), len(x)))

    u[:, -1] = u1y(y)
    u[-1, :] = ux1(x)

    for i in range(len(x) - 2, -1, -1):
        for j in range(len(y) - 2, -1, -1):
            u[j, i] = u[j + 1, i] * x[i] / (x[i] + y[j] + eps)
            u[j, i] += u[j, i + 1] * y[j] / (x[i] + y[j] + eps)

    diff_u = np.max(np.abs(u - prev_u))

    for _ in range(epochs):
        if diff_u <= eps:
            break

        prev_diff_u = diff_u
        prev_u = deepcopy(u)

        val1 = -2.0 * h * uyx0(x[:-1])
        val2 = 4.0 * prev_u[1, :-1]
        val3 = prev_u[2, :-1]

        u[0, :-1] = (val1 + val2 - val3) / 3.0

        val1 = -2.0 * h * ux0y(y[:-1])
        val2 = 4.0 * prev_u[:-1, 1]
        val3 = prev_u[:-1, 2]

        u[:-1, 0] = (val1 + val2 - val3) / 3.0

        for i in range(1, len(y) - 1):
            for j in range(1, len(x) - 1):
                sumv = prev_u[i - 1, j] + prev_u[i + 1, j]
                sumv += prev_u[i, j - 1] + prev_u[i, j + 1]
                u[i, j] = sumv / 4.0

        diff_u = np.max(np.abs(u - prev_u))

        if diff_u > diverge * prev_diff_u:
            break

    return u
```

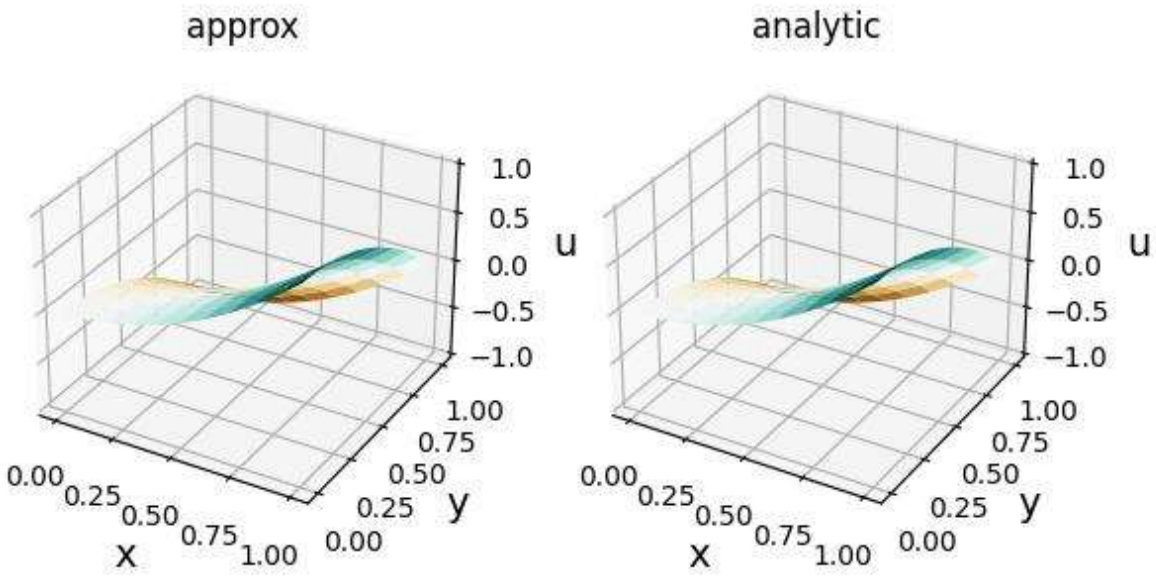
In [ ]:

```
def main(params):
    h = params['h']
    eps = params['eps']
    epochs = params['epochs']
    diverge = params['diverge']

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

    x = np.arange(xlhs, xrhs + h / 2.0, step=h)
    y = np.arange(ylhs, yrhs + h / 2.0, step=h)

    analytic = analitic_grid(x, y)
    approx = libman(params, x, y)

    err = error(approx, analytic)
    print(f'error={err:.3f}')

    filename = f'images/libman_h={h}_xlhs={xlhs}_xrhs={xrhs}'
    filename += f'_ylhs={ylhs}_yrhs={yrhs}_eps={eps}_epochs={epochs}'
    filename += f'_diverge={diverge}.jpg'

    plot(approx, analytic, x, y, filename)
```

In [ ]:

```
if __name__ == '__main__':
    params = {
        "h": 0.1,
        "xlhs": 0.0,
        "xrhs": 1.0,
        "ylhs": 0.0,
        "yrhs": 1.0,
        "eps": 0.0001,
        "epochs": 100,
        "diverge": 1.5,
    }

    main(params)
```

График сравнения аналитического решения и численного метода.



Файл **relaxation.py** содержит реализацию метода простых итераций с верхней релаксацией.

In [ ]:

```
import numpy as np

from tools import phi, psi0, psi1
from tools import tridiagonal_matrix_algorithm, error, analitic_grid

from plot import plot
```

In [ ]:

```
def relaxation(params, x, y):
    h = params['h']
    eps = params['eps']
    epochs = params['epochs']
    diverge = params['diverge']
    w = params['w']

    u = np.zeros((len(y), len(x)))
    prev_u = np.zeros((len(y), len(x)))

    u[:, -1] = u1y(y)
    u[-1, :] = ux1(x)

    for i in range(len(x) - 2, -1, -1):
        for j in range(len(y) - 2, -1, -1):
            u[j, i] = u[j + 1, i] * x[i] / (x[i] + y[j] + eps)
            u[j, i] += u[j, i + 1] * y[j] / (x[i] + y[j] + eps)

    diff_u = np.max(np.abs(u - prev_u))

    for _ in range(epochs):
        if diff_u <= eps:
            break

        prev_diff_u = diff_u
        prev_u = deepcopy(u)

        val1 = -2.0 * h * uyx0(x[:-1])
        val2 = 4.0 * u[1, :-1]
        val3 = u[2, :-1]

        u[0, :-1] += w * ((val1 + val2 - val3) / 3.0 - u[0, :-1])

        val1 = -2.0 * h * ux0y(y[:-1])
        val2 = 4.0 * u[:-1, 1]
        val3 = u[:-1, 2]

        u[:-1, 0] += w * ((val1 + val2 - val3) / 3.0 - u[:-1, 0])

        for i in range(1, len(y) - 1):
            for j in range(1, len(x) - 1):
                sumv = u[i - 1, j] + prev_u[i + 1, j]
                sumv += u[i, j - 1] + prev_u[i, j + 1]

                u[i, j] += w * (sumv / 4.0 - prev_u[i, j])

        diff_u = np.max(np.abs(u - prev_u))

        if diff_u > diverge * prev_diff_u:
            break

    return u
```

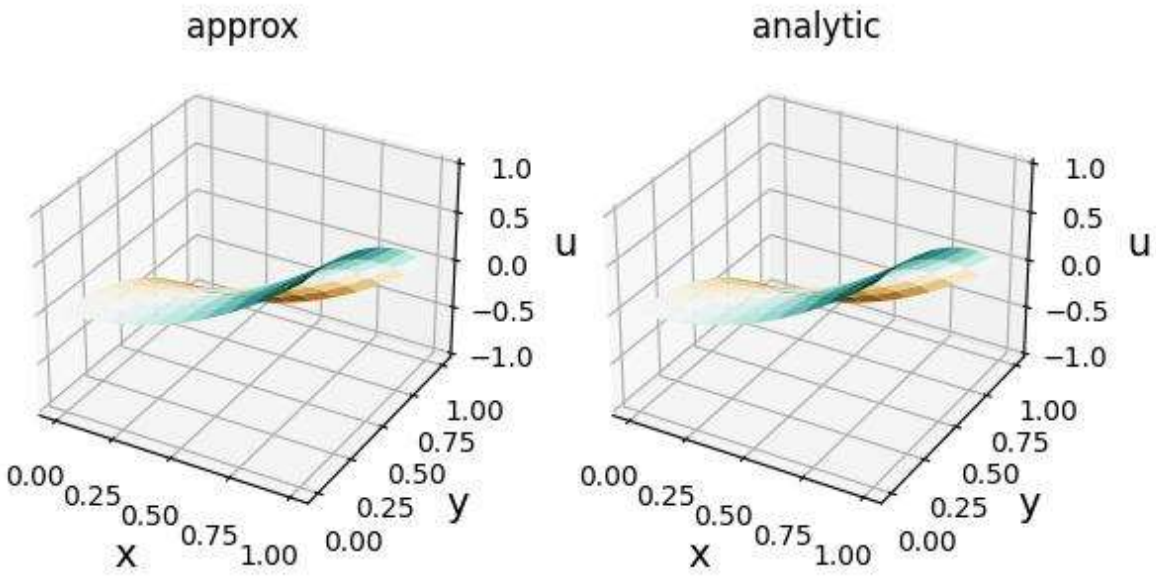
In [ ]:

```
def main(params):
    h = params['h']
    eps = params['eps']
    epochs = params['epochs']
    diverge = params['diverge']
    w = params['w']

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

    x = np.arange(xlhs, xrhs + h / 2.0, step=h)
    y = np.arange(ylhs, yrhs + h / 2.0, step=h)

    analytic = analitic_grid(x, y)
    approx = relaxation(params, x, y)

    err = error(approx, analytic)
    print(f'error={err:.3f}')

    filename = f'images/relaxation_h={h}_xlhs={xlhs}_xrhs={xrhs}'
    filename += f'_ylhs={ylhs}_yrhs={yrhs}_eps={eps}_epochs={epochs}'
    filename += f'_diverge={diverge}_w={w}.jpg'

    plot(approx, analytic, x, y, filename)
```

In [ ]:

```
if __name__ == '__main__':
    params = {
        "h": 0.1,
        "xlhs": 0.0,
        "xrhs": 1.0,
        "ylhs": 0.0,
        "yrhs": 1.0,
        "eps": 0.0001,
        "epochs": 100,
        "diverge": 1.5,
        "w": 1.5
    }

    main(params)
```

График сравнения аналитического решения и численного метода.



Файл **zeidel.py** содержит реализацию метода Зейделя.

In [ ]:

```
import numpy as np

from tools import phi, psi0, psi1
from tools import tridiagonal_matrix_algorithm, error, analitic_grid

from plot import plot
```

In [ ]:

```
def zeidel(params, x, y):
    h = params['h']
    eps = params['eps']
    epochs = params['epochs']
    diverge = params['diverge']
    w = 1.0

    u = np.zeros((len(y), len(x)))
    prev_u = np.zeros((len(y), len(x)))

    u[:, -1] = u1y(y)
    u[-1, :] = ux1(x)

    for i in range(len(x) - 2, -1, -1):
        for j in range(len(y) - 2, -1, -1):
            u[j, i] = u[j + 1, i] * x[i] / (x[i] + y[j] + eps)
            u[j, i] += u[j, i + 1] * y[j] / (x[i] + y[j] + eps)

    diff_u = np.max(np.abs(u - prev_u))

    for _ in range(epochs):
        if diff_u <= eps:
            break

        prev_diff_u = diff_u
        prev_u = deepcopy(u)

        val1 = -2.0 * h * uyx0(x[:-1])
        val2 = 4.0 * u[1, :-1]
        val3 = u[2, :-1]

        u[0, :-1] += w * ((val1 + val2 - val3) / 3.0 - u[0, :-1])

        val1 = -2.0 * h * ux0y(y[:-1])
        val2 = 4.0 * u[:-1, 1]
        val3 = u[:-1, 2]

        u[:-1, 0] += w * ((val1 + val2 - val3) / 3.0 - u[:-1, 0])

        for i in range(1, len(y) - 1):
            for j in range(1, len(x) - 1):
                sumv = u[i - 1, j] + prev_u[i + 1, j]
                sumv += u[i, j - 1] + prev_u[i, j + 1]

                u[i, j] += w * (sumv / 4.0 - prev_u[i, j])

        diff_u = np.max(np.abs(u - prev_u))

        if diff_u > diverge * prev_diff_u:
            break

    return u
```

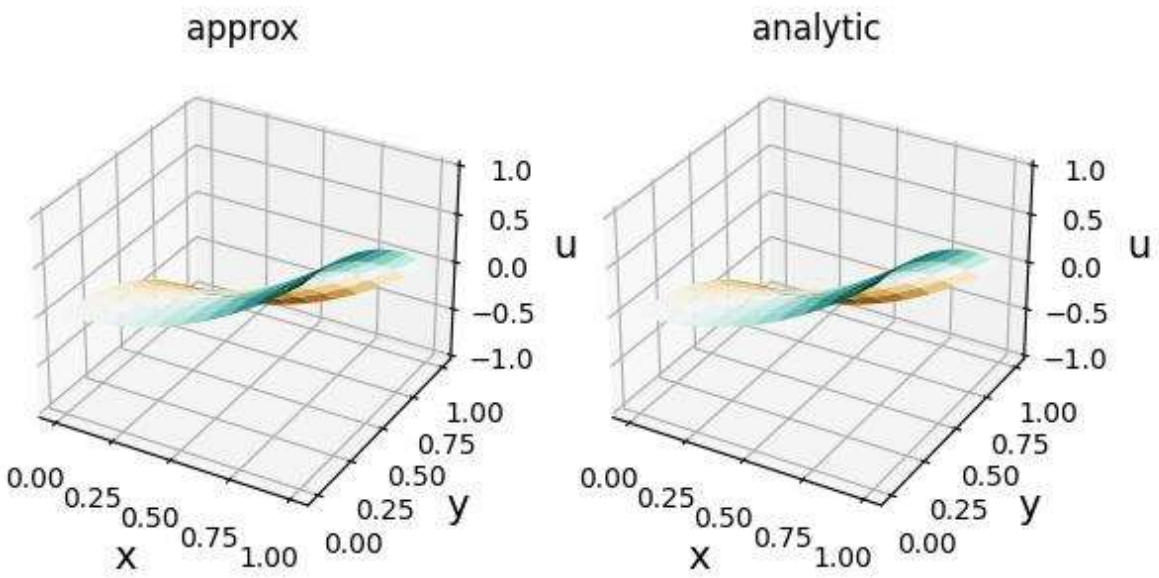
In [ ]:

```
def main(params):
    h = params['h']
    eps = params['eps']
    epochs = params['epochs']
    diverge = params['diverge']

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

    x = np.arange(xlhs, xrhs + h / 2.0, step=h)
    y = np.arange(ylhs, yrhs + h / 2.0, step=h)

    analytic = analitic_grid(x, y)
    approx = zeidel(params, x, y)

    err = error(approx, analytic)
    print(f'error={err:.3f}')

    filename = f'images/zeidel_h={h}_xlhs={xlhs}_xrhs={xrhs}'
    filename += f'_ylhs={ylhs}_yrhs={yrhs}_eps={eps}_epochs={epochs}'
    filename += f'_diverge={diverge}.jpg'

    plot(approx, analytic, x, y, filename)
```

In [ ]:

```
if __name__ == '__main__':
    params = {
        "h": 0.1,
        "xlhs": 0.0,
        "xrhs": 1.0,
        "ylhs": 0.0,
        "yrhs": 1.0,
        "eps": 0.0001,
        "epochs": 100,
        "diverge": 1.5,
    }

    main(params)
```

График сравнения аналитического решения и численного метода.



In [ ]: