Лабораторная работа №3 учебного года 2023-2024 по курсу «Численные методы»

Выполнил: Зубко Д. В.
Группа: М8О-408Б-20
Преподаватель: Пивоваров Д.Е.
Вариант по списку группы: 8

Условие лабораторной работы

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, h_y.

Вариант 8

8.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2\frac{\partial u}{\partial x} - 3u,$$

$$u(0, y) = \cos y,$$

$$u(\frac{\pi}{2}, y) = 0,$$

$$u(x, 0) = \exp(-x)\cos x,$$

$$u(x, \frac{\pi}{2}) = 0.$$

Аналитическое решение: $U(x, y) = \exp(-x)\cos x \cos y$.

Программа

main.py

```python
import copy

import numpy as np

from functions import phi1, phi2, phi3, phi4
from show import show_inaccuracy, show_result
from task import bx, by, c, count_x, count_y, eps, hx, hy, lx,
max_iterations


def liebmann_method():
    u = np.zeros((count_x + 1, count_y + 1))

    u[0, 0] = phi1(0)
    u[-1, -1] = phi2(-hy)

    for i in range(1, count_x):
        u[i, 0] = phi3(i * hx)
        u[i, -1] = phi4(i * hx)

    for j in range(1, count_y):
        u[0, j] = phi1(j * hy)
        u[-1, j] = phi2(j * hy)
        for i in range(1, count_x):
            u[i, j] = u[0, j] + (u[-1, j] - u[0, j]) / lx * i * hx

    k = 0
    while True:
        k += 1
        if k > max_iterations:
            print("Достигнуто максимальное число итераций!")
            break

        u_prev = copy.deepcopy(u)

        for j in range(1, count_y):
            for i in range(1, count_x):
                u[i, j] = (
                    -(u_prev[i + 1, j] + u_prev[i - 1, j])
                    - hx**2 * (u_prev[i, j + 1] + u_prev[i, j - 1]) / (hy**2)
                    - bx * hx * (u_prev[i + 1, j] - u_prev[i - 1, j]) / 2
                    - by * hx**2 * (u_prev[i, j + 1] - u_prev[i, j - 1]) / (2 * hy)
```

```
            ) / (c * hx**2 - 2 * (hy * hy + 1 * hx**2) / (hy**2))

        norm = np.linalg.norm(u - u_prev, np.inf)
        if norm <= eps:
            break

    print("liebmann_method: k =", k)
    return u


def sor_method(omega):
    u = np.zeros((count_x + 1, count_y + 1))

    u[0, 0] = phi1(0)
    u[-1, -1] = phi2(-hy)

    for i in range(1, count_x):
        u[i, 0] = phi3(i * hx)
        u[i, -1] = phi4(i * hx)

    for j in range(1, count_y):
        u[0, j] = phi1(j * hy)
        u[-1, j] = phi2(j * hy)
        for i in range(1, count_x):
            u[i, j] = u[0, j] + (u[-1, j] - u[0, j]) / lx * i * hx

    k = 0
    while True:
        k = k + 1
        if k > max_iterations:
            print("Достигнуто максимальное число итераций!")
            break

        u_prev = copy.deepcopy(u)

        for j in range(1, count_y):
            for i in range(1, count_x):
                u[i, j] = (
                    (
                        -(u_prev[i + 1, j] + u[i - 1, j])
                        - 1 * hx**2 * (u_prev[i, j + 1] + u[i, j - 1]) / (hy**2)
                        - bx * hx * (u_prev[i + 1, j] - u[i - 1, j]) / 2
```

```
                - by * hx**2 * (u_prev[i, j + 1] - u[i, j - 1]) / (2 * hy)
            )
            / (c * hx**2 - 2 * (hy**2 + 1 * hx**2) / (hy**2))
        ) * omega + (1 - omega) * u_prev[i, j]

    norm = np.linalg.norm(u - u_prev, np.inf)
    if norm <= eps:
        break

  if omega == 1:
    print("seildel_method: k =", k)
  else:
    print("sor_method: k =", k)

  return u


def get_axis_np(count, mul):
  axis = np.zeros(count)
  for i in range(count):
    axis[i] = mul * i
  return axis


def main():
  u1 = liebmann_method()
  u2 = sor_method(1)
  u3 = sor_method(1.5)

  y_axis = get_axis_np(count_y + 1, hy)
  x_axis = get_axis_np(count_x + 1, hx)

  show_result(y_axis, x_axis, u1, u2, u3)
  show_inaccuracy(y_axis, x_axis, u1)


if __name__ == "__main__":
  main()
```

show.py

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

from functions import analytic_solution
from task import count_x, count_y


def show_result(y_axis, x_axis, u1, u2, u3):
    fig, ax = plt.subplots(2)
    fig.suptitle("Сравнение численных решений ДУ с
аналитическим")
    fig.set_figheight(15)
    fig.set_figwidth(16)
    y = 0
    for i in range(2):
        ax[i].plot(x_axis, u1[:, y], label="Liebmann method")
        ax[i].plot(x_axis, u2[:, y], label="Seidel method")
        ax[i].plot(x_axis, u3[:, y], label="Successive over-relaxation")
        ax[i].plot(
            x_axis, [analytic_solution(x, y_axis[y]) for x in x_axis],
label="Analytic"
        )
        ax[i].grid(True)
        ax[i].set_xlabel("x")
        ax[i].set_ylabel("u")
        ax[i].set_title(f"Решения при y = {y / count_y}")
        y += count_y - 1

    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()
```

```
def show_inaccuracy(y_axis, x_axis, u):
    inaccuracy = np.zeros(count_x + 1)
    for i in range(count_x + 1):
        inaccuracy[i] = np.max(
            np.abs(u[i] - np.array([analytic_solution(x_axis[i], y) for y in
y_axis]))
        )

    plt.figure(figsize=(14, 8))
    plt.plot(x_axis[1:], inaccuracy[1:], "violet", label="Ошибка")
    plt.legend(bbox_to_anchor=(1.05, 1), loc="upper right",
borderaxespad=0.0)
    plt.title("График изменения ошибки")
    plt.xlabel("y")
    plt.ylabel("error")
    plt.grid(True)
    plt.show()
```

task.py

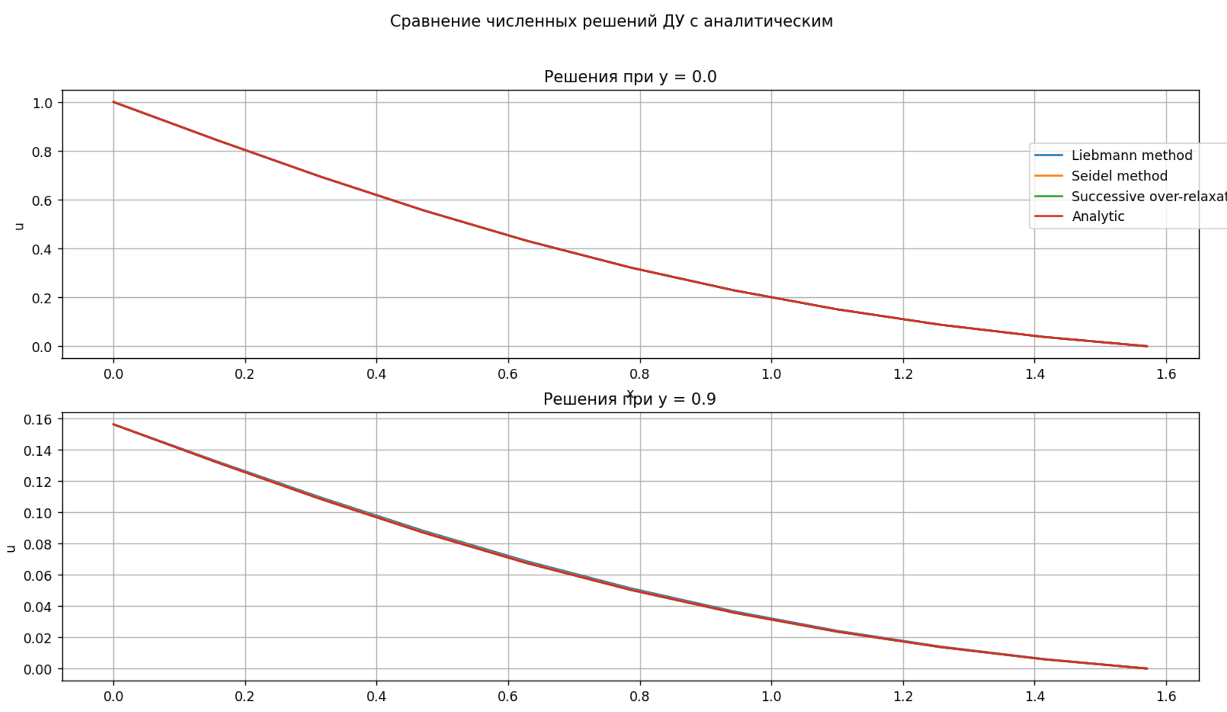
```
import numpy as np

count_x = 10
count_y = 10
lx = np.pi / 2
ly = np.pi / 2
hx = lx / count_x
hy = ly / count_y
bx = 2
by = 0
c = 3
eps = 0.001
max_iterations = 1000
```

functions.py

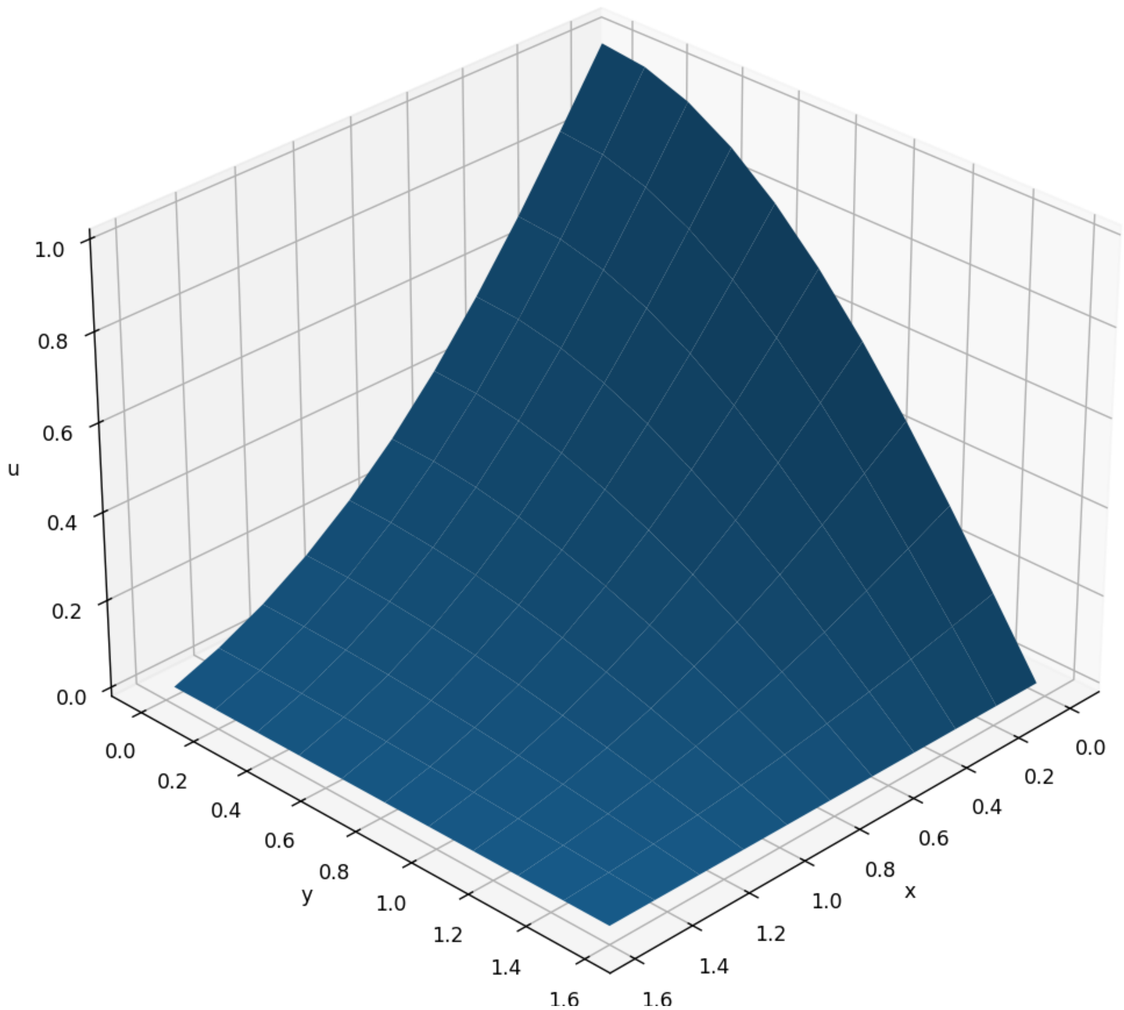
```
import numpy as np


def analytic_solution(x, y):
```

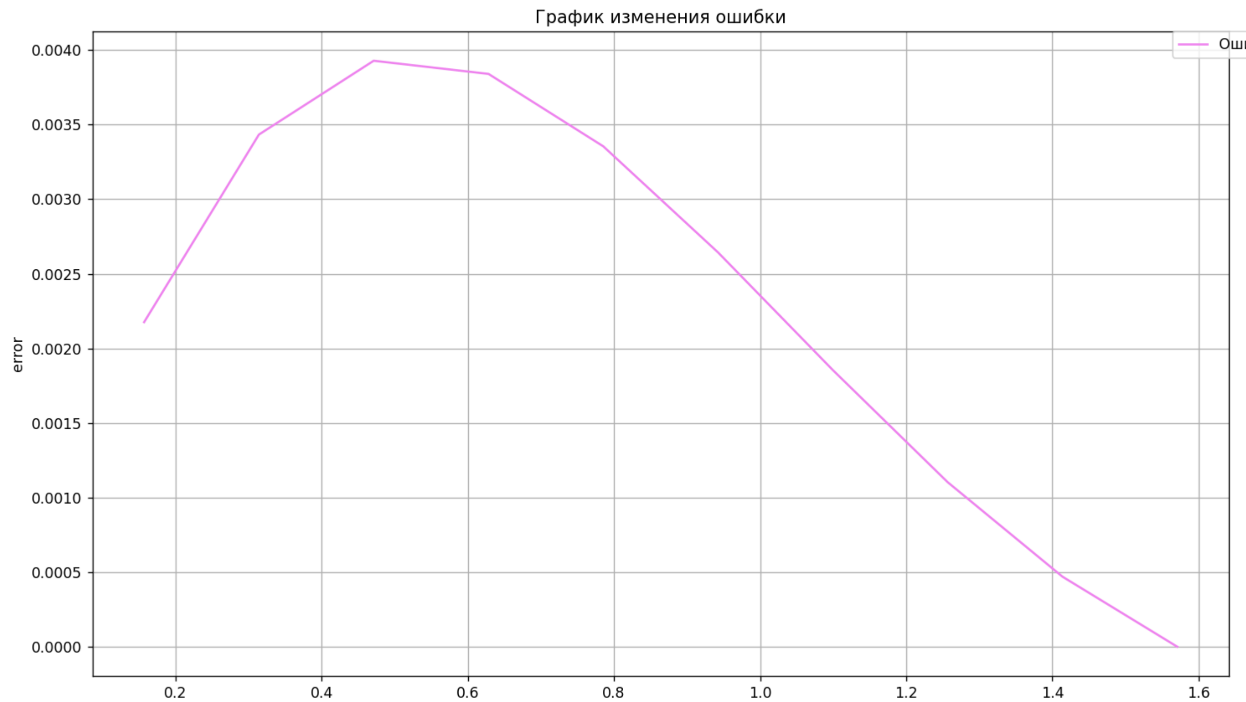
```
    return np.exp(-x) * np.cos(x) * np.cos(y)


def phi1(y):
    return np.cos(y)


def phi2(y):
    return 0


def phi3(x):
    return np.exp(-x) * np.cos(x)


def phi4(x):
    return 0
```

Результаты работы



Аналитическое решение





Вывод по лабораторной работе

После выполнения лабораторной работы, я успешно нашел решение начально-краевой задачи для дифференциального уравнения эллиптического типа. В рамках этого процесса, я применил три разных метода для решения системы линейных алгебраических уравнений и провел проверку на наличие ошибок в полученных вычислениях.