Лабораторная работа №7. Метод стрельбы

Денис Слынко, Б01-818

2021

Задача:

Найти решение краевой задачи для одномерного стационарного уравнения теплопроводности

$$\begin{cases} \frac{d}{dx}((x+1)\frac{du}{dx}) - e^x u = -e^{-x^2}, \\ u_x(0) = 0, \\ -2u_x(0) = u(1). \end{cases}$$

в одиннадцати равноудалённых точках отрезка $[0, 1]$ с относительной точностью 0.0001.

1 Результаты вычислений

Вычисления численного решения задачи Штурма-Лиувилля проводились с помощью метода трёхдиагональной прогонки. Критерием достижения необходимой точности было сравнение значений в искомых точках при данной и предыдущей итерации. Искомая точность была достигнута при 8019 точках на отрезке. Полученные численные значения искомой функции в одиннадцати равноудалённых точках отрезка $[0, 1]$:

$[0.3634333394866881; 0.3610022536957464; 0.3544286117066448; 0.34469041829106356; 0.33264312165986204; 0.31904147300057906; 0.3045522967442257; 0.28976234925377564; 0.2751839280402337; 0.2612599174109243; 0.2483692916174675]$

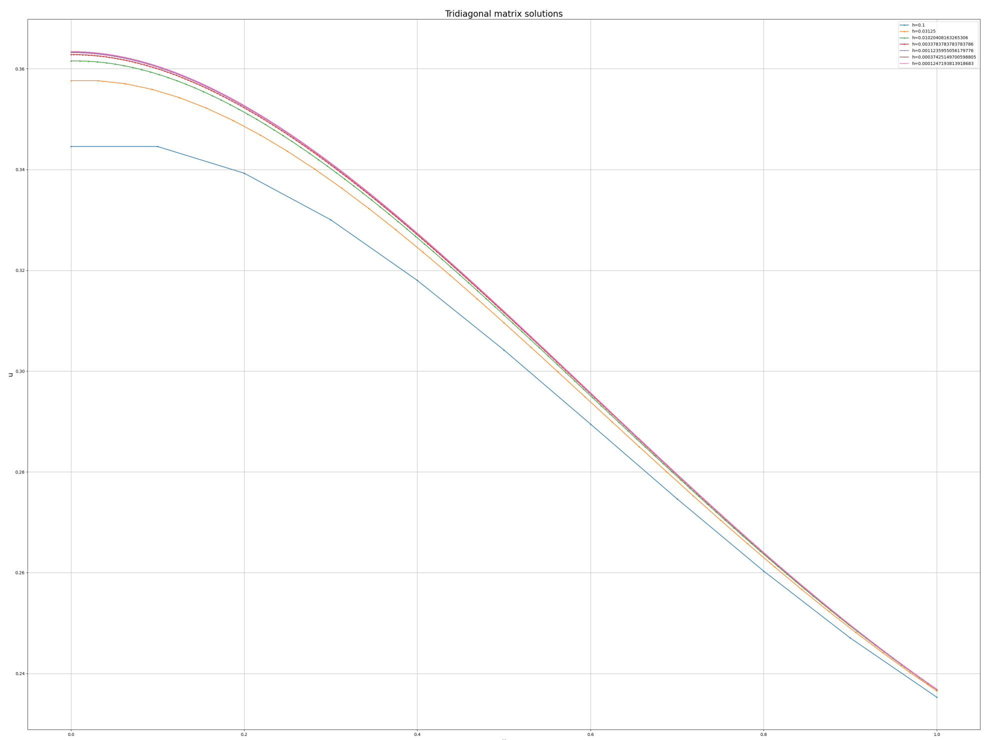

Рис. 1: Решение методом трёхдиагональной прогонки для различных значений шага сетки.

Отдельные графики решения при различных значениях шага сетки представлены в дополнении к работе.

Также была проанализирована ошибка приближений в зависимости от шага сетки. На рисунке 2 представлен график пар последовательных ошибок, аппроксимированный квадратичной зависимостью. Второй порядок погрешности при изменении шага сетки согласуется с вторым порядком аппроксимации метода.

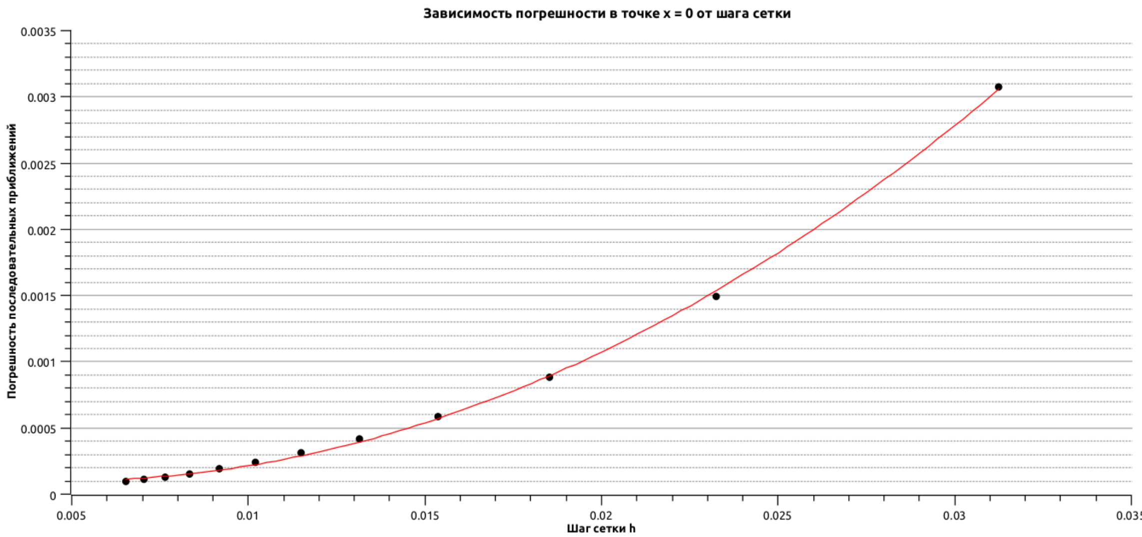

Рис. 2: График зависимости ошибки пар последовательных приближений от величины шага сетки. Точки аппроксимированы полиномом второй степени.

2 Код программы

// main.py

```
#!/usr/bin/python3.8

import matplotlib.pyplot as plt
import numpy as np
import subprocess
import sys
import os
import equation


def create_plots_dir(name="Plots"):
    if not os.path.isdir(os.path.join(os.path.dirname(sys.argv[0]),
        name)):
        subprocess.check_output(f"mkdir {name}", shell=True)
```

```python
epsilon = 0.0001
result_values = np.zeros(11)

dirname = "Plots"
create_plots_dir(dirname)
cumulative_fig = plt.figure()
cumulative_ax = cumulative_fig.add_subplot(111)

N = 11
while True:
    h = (equation.Equation.b - equation.Equation.a) / float(N - 1)

    M, d = equation.get_matrices(N, h)

    p, r = equation.get_auxiliary_matrices(M, d)

    u = np.zeros(N)
    u[N - 1] = r[N - 1]
    for i in range(N - 2, -1, -1):
        u[i] = r[i] - p[i] * u[i + 1]
    x = np.array([h * t for t in range(N)], dtype=float)

    markersz = 2 if N < 300 else 1
    cumulative_ax.plot(x, u, marker='o', markersize=markersz,
        label=f"h={h}")

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(x, u, marker='o', markersize=2)
    ax.set_title(f"Tridiagonal matrix solution for h = {h}",
        fontsize=20)
    ax.set_xlabel("x", fontsize=16)
    ax.set_ylabel("u", fontsize=16)
    ax.grid()

    fig.set_figheight(20)
    fig.set_figwidth(25)
```

```
    fig.savefig(os.path.join(dirname, f"with_{N}_points.png"))
    plt.close(fig)

    step = int(N / 11.0)
    print(h, abs(result_values[0] - u[0]))
    if all([abs(result_values[i] - u[step * i]) < epsilon for i in
        range(11)]):
        print(f"The solution with the error = {epsilon} was gained
            with {N} points")
        print([u[step * i] for i in range(11)])

        cumulative_ax.set_title("Tridiagonal matrix solutions",
            fontsize=20)
        cumulative_ax.set_xlabel("x", fontsize=16)
        cumulative_ax.set_ylabel("u", fontsize=16)
        cumulative_ax.legend()
        cumulative_ax.grid()

        cumulative_fig.set_figheight(30)
        cumulative_fig.set_figwidth(40)

        cumulative_fig.savefig(os.path.join(dirname,
            "cumulative.png"))
        break
    else:
        result_values = np.array([u[step * i] for i in range(11)])
    N *= 3
```

// equation.py

```
import numpy as np


class Equation:
    epsilon = 0.0001
    a = 0.0
    b = 1.0

    @staticmethod
    def k(x):
```

```python
        return x + 1

    @staticmethod
    def q(x):
        return np.exp(x)

    @staticmethod
    def f(x):
        return np.exp(-1.0 * x * x)


def get_matrices(N, h):
    M = np.zeros(N * N).reshape(N, N)
    d = np.zeros(N)

    M[0][0] = -1.0
    M[0][1] = 1.0
    M[N - 1][N - 2] = -1.0 * Equation.k(1.0)
    M[N - 1][N - 1] = h + Equation.k(1.0)
    for i in range(1, N - 1):
        M[i][i - 1] = Equation.k((float(i) - 0.5) * h) / h / h
        M[i][i] = -1.0 * (
                    Equation.k((float(i) - 0.5) * h) +
                        Equation.k((float(i) + 0.5) * h)) / h / h - \
                  Equation.q(i * h)
        M[i][i + 1] = Equation.k((float(i) + 0.5) * h) / h / h
        d[i] = -1.0 * Equation.f(i * h)

    return M, d


def get_auxiliary_matrices(M, d):
    if M.shape[0] != M.shape[1] or M.shape[0] != d.shape[0]:
        raise ValueError("invalid matrices")

    N = d.shape[0]
    p = np.zeros(N)
    r = np.zeros(N)

    p[0] = M[0][1] / M[0][0]
```

```
r[0] = d[0] / M[0][0]

for i in range(1, N - 1):
    p[i] = M[i][i + 1] / (M[i][i] - M[i][i - 1] * p[i - 1])
    r[i] = (d[i] - M[i][i - 1] * r[i - 1]) / (M[i][i] - M[i][i -
        1] * p[i - 1])
r[N - 1] = (d[N - 1] - M[N - 1][N - 2] * r[N - 2]) / (M[N - 1][N
    - 1] - M[N - 1][N - 2] * p[N - 2])

return p, r
```

3 Литература

1 Демченко В.В. Вычислительный практикум по прикладной математике: Учебное пособие. - М.: МФТИ, 2007. - 196 с. ISBN 5-7417-0186-8