Лабораторная работа №7. Метод стрельбы

Денис Слынко, Б01-818

2021

Задача:
Рассмотрите нелинейную сингулярно-возмущенную краевую задачу

$$\begin{cases} \epsilon y'' = (y')^2, \\ y(0) = 1, \\ y(1) = 0, \\ 0 < \epsilon << 1. \end{cases}$$

1. Получите точное решение задачи, см [2]. Для этого сделайте замену $y' = p(y)$.

2. Предложите и реализуйте численный метод решения задачи. Сравните полученное решение с точным. Исследуйте поведение погрешности Вашего численного метода при $\epsilon \to +0$.

1 Точное решение

Произведём замену $y' = p(y)$; тогда:

$$\epsilon p p' = p^2 \tag{1}$$

$$\epsilon \frac{dp}{dy} = p \tag{2}$$

$$p = y' = Ce^{\frac{y}{\epsilon}} \tag{3}$$

$$-\epsilon\frac{d}{dx}(e^{-\frac{y}{\epsilon}}) = C \tag{4}$$

$$-\frac{y}{\epsilon} = ln|-\frac{C}{\epsilon}x + C_1| \tag{5}$$

$$y = -\epsilon ln|-\frac{C}{\epsilon}x + C_1| \equiv -\epsilon ln(x + e^{-\frac{1}{\epsilon}}(1-x)), x \in [0,1]. \tag{6}$$

2 Методика численного решения

Для решения дифференциальной задачи был выбран метод стрельбы. Он подразумевает решение промедуточной задачи

$$\begin{cases} \epsilon y'' = (y')^2, \\ y(0) = 1, \\ y'(0) = \alpha, \\ 0 < \epsilon << 1, \end{cases}$$

где α - варьируемый параметр. Его подбор ведётся таким образом, чтобы удовлетворить правому граничному условию $y(1) = 0$. Поиск ведётся метод половинного деления.

Решение промежуточной задачи при этом производится с помощью следующей схемы 2 порядка аппроксимации:

$$\epsilon\frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} = (\frac{y_{n+1} - y_{n-1}}{2h})^2 \tag{7}$$

Выражая каждый последующий слой через предыдущие, получим выражение на поиск y_{n+1}:

$$y_{n+1}^2 - 2y_{n+1}(y_{n-1} + 2\epsilon) + y_{n-1}^2 + 8\epsilon y_n - 4\epsilon y_{n-1} = 0. \tag{8}$$

Как видно, для решения необходимо разрешить 1 слой. Это делается с помощью разложения по формуле Тейлора:

$$y_1 = y_0 + y_0'h + O(h^2) \equiv y_0 + \alpha h + O(h^2)/ \tag{9}$$

При этом порядок апроксимации задачи остаётся вторым.

3 Результаты вычислений

Вычисления численного решения системы ОДУ проводились для значений ϵ из $[0.07, 0.1, 0.35, 0.7]$. График решений на отрезке $[0, 1]$ представлен на рис. 1. Также для каждого значения параметра была найдена погрешность численного решения:

0.07	0.1	0.35	0.7
0.0045	0.00064	0.000055	0.00024

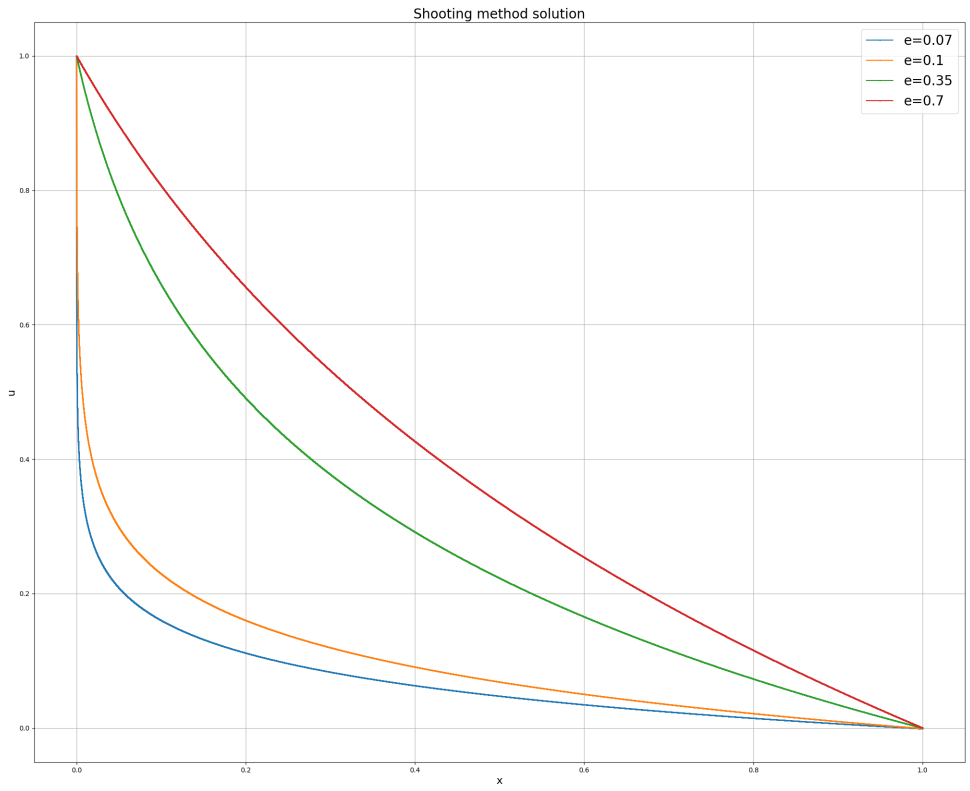

Рис. 1: Решение методом стрельбы со схемой второго порядка аппроксимации; ϵ - параметр уравнения.

4 Код программы

// Main.py

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

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

```
import subprocess
import time
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)


def get_max_deviation(y, exact_y):
    return max(np.abs(np.array(y, dtype=np.float64) - exact_y))


n = 1000000
precision = np.float64(0.001)

h = (equation.Equation.b - equation.Equation.a) / n
x = np.arange(0.0, n * h, step=h, dtype=np.float64)

fig = plt.figure()
ax = fig.add_subplot(111)

eps = [np.float64(0.07), np.float64(0.1), np.float64(0.35),
    np.float64(0.7)]
for i in eps:
    epsilon = i

    exact_solution = np.float64(-1.0) * epsilon * np.log(x +
        np.exp(-1.0 / epsilon) * (np.float64(1.0) - x))

    start_time = time.time()
    u = equation.shooting_method(n, epsilon, precision=precision)
    end_time = time.time()
    print(f"Time elapsed for epsilon = {epsilon}: {end_time -
        start_time}s")
    print("Maximum deviation comparing to the exact solution
```

```
        equals", get_max_deviation(u, exact_solution))
    ax.plot(x, u, marker='o', label=f"e={epsilon}", markersize=1)

ax.set_title(f"Shooting method solution", fontsize=20)
ax.set_xlabel("x", fontsize=16)
ax.set_ylabel("u", fontsize=16)
ax.legend(fontsize=20)
ax.grid()

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

dirname = "Plots"
create_plots_dir(dirname)
fig.savefig(os.path.join(dirname, "graph.png"))
```

// equation.py

```
import numpy as np


class Equation:
    a = np.float64(0.0)
    b = np.float64(1.0)

    @staticmethod
    def get_boundary_values():
        return [(np.float64(0.0), np.float64(1.0)),
            (np.float64(1.0), np.float64(0.0))]

    @staticmethod
    def exact_solution(x, e=np.float64(0.01)):
        return -1.0 * e * np.log(x + np.exp(-1.0 / e) *
            (np.float64(1.0) - x))


def shooting_method(n, epsilon, precision=0.001):
    bv = Equation.get_boundary_values()
    right_bv = bv[1][1]
```

```
    def right_boundary_dev(right_y_value, rbv):
        return right_y_value - rbv

    step = np.float64(0.001)
    while True:
        alpha1 = -1.0 * epsilon * (np.exp(1.0 / epsilon) - 1) + step
        tmp1 = get_auxiliary_solution(n, epsilon, alpha1)
        dev1 = right_boundary_dev(tmp1[n - 1], right_bv)
        if abs(dev1) < precision:
            return tmp1

        alpha2 = -1.0 * epsilon * (np.exp(1.0 / epsilon) - 1) - step
        tmp2 = get_auxiliary_solution(n, epsilon, alpha2)
        dev2 = right_boundary_dev(tmp2[n - 1], right_bv)
        if abs(dev2) < precision:
            return tmp2

        if dev1 * dev2 < 0.0:
            while True:
                alpha = (alpha1 + alpha2) / 2
                tmp = get_auxiliary_solution(n, epsilon, alpha)
                dev = right_boundary_dev(tmp[n - 1], right_bv)
                if abs(dev) < precision:
                    return tmp

                if dev * dev1 < 0.0:
                    alpha2 = alpha
                elif dev * dev2 < 0.0:
                    alpha1 = alpha
                else:
                    print("Unforeseen!")
        else:
            step *= 10.0
            print(f"Step equals {step}")


def get_auxiliary_solution(n, epsilon, alpha):
    h = (Equation.b - Equation.a) / n
    bv = Equation.get_boundary_values()
```

```
    y = np.zeros(n)
    y[0] = bv[0][1]
    y[1] = y[0] + alpha * h
    for i in range(2, n):
        b = y[i - 2] + 2.0 * epsilon
        d = 4.0 * epsilon * (epsilon + 2.0 * y[i - 2] - 2.0 * y[i -
            1])
        y1 = b - np.sqrt(d)
        y2 = b + np.sqrt(d)
        if abs(y[i - 1] - y1) < abs(y[i - 1] - y2):
            y[i] = y1
        else:
            y[i] = y2

    return y
```

5 Литература

1 Практические занятия по вычислительной математике в МФТИ : учеб. пособие / Е. Н. Аристова, А. И. Лобанов. Часть II. – М. : МФТИ, 2015. – 310 с. ISBN 978-5-7417-0568-1 (Ч. II)

2 Чанг К., Хауэс Ф. Нелинейные сингулярно возмущенные краевые задачи. Теория и приложения: Пер. с англ. — М.: Мир, 1988. — 247 с, ил.