Лабораторная работа №4. Задача E5

Денис Слынко, Б01-818

2021

Задача:

$$
\begin{cases}
y_1' = -Ay_1 - By_1y_3,\\
y_2' = Ay_1 - MCy_2y_3,\\
y_3' = Ay_1 - By_1y_3 - MCy_2y_3 + Cy_4,\\
y_4' = By_1y_3 - Cy_4.
\end{cases}
$$

Начальные условия: $y_1 = 1.76 \cdot 10^{-3}$, а все остальные переменные равны 0. Значения коэффициентов модели следующие: $A = 7.89 \cdot 10^{-10}, B = 1.1 \cdot 10^7, C = 1.13 \cdot 10^3, M = 10^6$. Первоначально задача ставилась на отрезке $T_k = 1000$, но впоследствии было обнаружено, что она обладает нетривиальными свойствами вплоть до времени $T_k = 10^{13}$ (подробнее см. [2]). Обратить особое внимание, что в процессе расчетов приходится иметь дело с очень малыми концентрациями реагентов (малы значения y_2, y_3 и y_4). Как «подправить» постановку задачи E5?

1 Используемые методы

В задаче необходимо воспользоваться тем фактом, что имеется инвариант $y_2' - y_3' - y_4' = 0$, а из начального условия $y_2 = y_3 = y_4 = 0$ справедливо также соотношение $y_2 - y_3 - y_4 = 0$. Эти два выражения упрощают вычисления и позволяют избежать потери значащих разрядов при умножении.

В качестве основного метода был выбран полу-неявный метод Рунге-Кутты четвертого порядка сходимости со следующей таблицей Бутчера:

$$
\begin{array}{c|cccc}
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
\frac{2}{3} & \frac{1}{6} & \frac{1}{2} & 0 & 0 \\
\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \\
1 & \frac{3}{2} & -\frac{3}{2} & \frac{1}{2} & \frac{1}{2} \\
\hline
 & \frac{3}{2} & -\frac{3}{2} & \frac{1}{2} & \frac{1}{2}
\end{array}
$$

Для нахождения вспомогательных коэффициентов k_i через решение нелинейной системы уравнений был применён метод Ньютона с критерием останова $epsilon = 10^{-20}$. Вычисления матрицы Якоби в методе Ньютона производились без учёта шага итерации, что ускоряло процесс вычислений (на каждый шаг y_n обратная матрица к матрице Якоби вычислялась один раз, а не $s = 4$ раз).

В целях сравнения результатов была произведена попытка решить данную задачу методом Розенброка, основанном на такой же таблице Бутчера. Однако, данный метод оказался расходящимся на предложенной задаче.

2 Результаты вычислений

Вычисления численного решения системы ОДУ проводились в промежутке $[0, 10^6]$ с шагом 1.0. Вычисления заняли порядка 220 секунд.

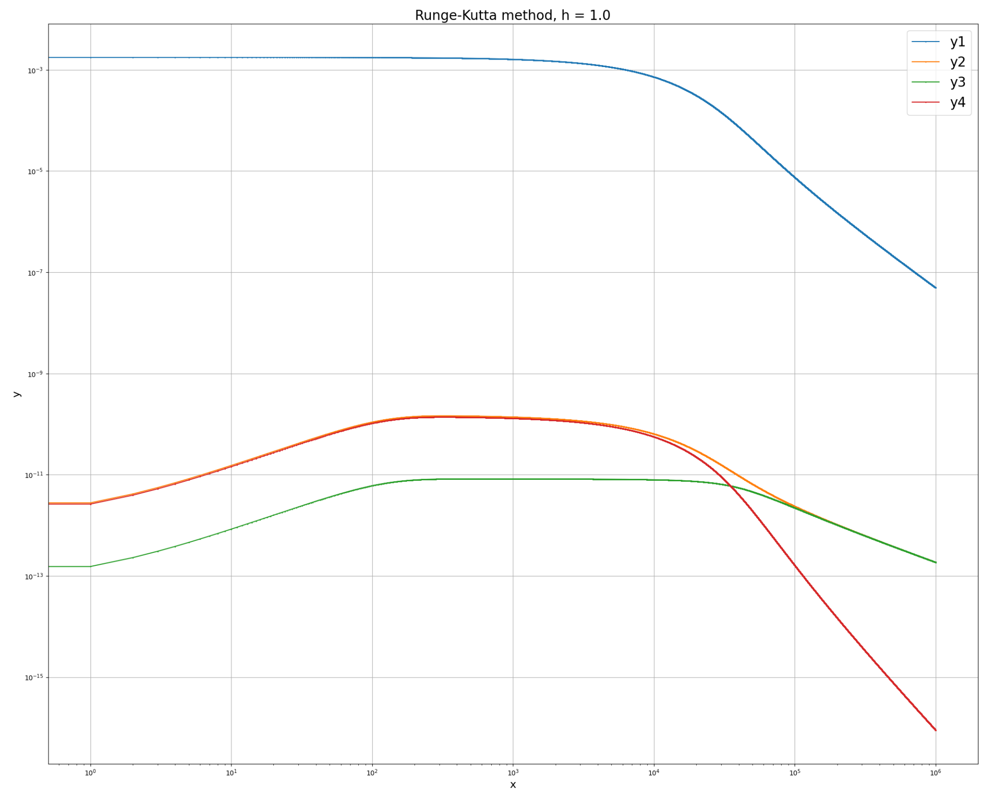

Рис. 1: Решение неявным методом Рунге-Кутты; обе шкалы логарифмические

3 Код программы

// Main.py

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

```python
import equation
import argparse
import time
import sys
import os


class Main:
    system = None
    right_end = 0
    debug = False

    def __init__(self, right_end=1.0e+03):
        self.system = equation.Equation()
        self.right_end = right_end

        parser = argparse.ArgumentParser()
        parser.add_argument("--debug", action="store_true",
                            help="Debug mode, print step results")
        cmd_args = parser.parse_args()
        self.debug = cmd_args.debug

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

    @staticmethod
    def __initiate__method__():
        num_of_steps = 4

        alpha = np.zeros(num_of_steps * num_of_steps,
            dtype=np.float64).reshape(num_of_steps, num_of_steps)
        for i in range(num_of_steps):
            alpha[i][i] = np.float64(0.5)
        alpha[1][0] = np.float64(1.0 / 6.0)
        alpha[2][0] = np.float64(-0.5)
        alpha[2][1] = alpha[3][2] = np.float64(0.5)
```

```
        alpha[3][0] = np.float64(1.5)
        alpha[3][1] = np.float64(-1.5)
        c = np.array([0.5, 2.0 / 3.0, 0.5, 1.0], dtype=np.float64)
        b = np.array([1.5, -1.5, 0.5, 0.5], dtype=np.float64)

        return num_of_steps, alpha, b, c

    def __solve_runge_kutte__(self, y_n, h):
        num_of_steps, alpha, b, c = self.__initiate__method__()

        k = np.zeros(num_of_steps * self.system.num_of_equations,
            dtype=np.float64) \
            .reshape(num_of_steps, self.system.num_of_equations)

        # alpha[i][i] is the same for all i
        # we calculate derivative matrix in y_n for the sake of fast
            execution
        k_matrix =
            np.linalg.inv(np.identity(self.system.num_of_equations,
            dtype=np.float64) -\
                                h * alpha[0][0] *
                                    self.system.calculate_der_f(y_n))
        for i in range(num_of_steps):
            k[i] = ssolver.solve_newton(k_matrix, y_n + h *
                sum(alpha[i][j] * k[j] for j in range(i)), h,
                alpha[i][i],
                                          self.system.calculate_f)

        tmp = y_n + b.dot(k)
        tmp[2] = tmp[1] - tmp[3]
        return tmp

    # doesn't work for the task
    def __solve_rosenbrock__(self, y_n, h):
        num_of_steps, alpha, b, c = self.__initiate__method__()

        k = np.zeros(num_of_steps * self.system.num_of_equations,
            dtype=np.float64)\
            .reshape(num_of_steps, self.system.num_of_equations)
        k[0] = h * self.system.calculate_f(y_n)
```

```
    # alpha[i][i] is the same for all i
    tmp_matrix = np.identity(self.system.num_of_equations,
        dtype=np.float64) -\
                h * alpha[0][0] * self.system.calculate_der_f(y_n)
    for i in range(1, num_of_steps):
        tmp = sum(alpha[i][j] * k[j] for j in range(i))
        k[i] = ssolver.solve_seidel(tmp_matrix, h *
            self.system.calculate_f(y_n + tmp), debug=self.debug)

    tmp = y_n + b.dot(k)
    tmp[2] = tmp[1] - tmp[3]
    return tmp

def solve(self, h, p=1):
    x_axis = []
    y_values = ([], [], [], [])

    x = np.float64(0.0)
    y = self.system.initial_conditions
    while x <= self.right_end:
        try:
            if p == 1:
                y_tmp = self.__solve_runge_kutte__(y, h)
            else:
                y_tmp = self.__solve_rosenbrock__(y, h)
            y = y_tmp
        except np.linalg.LinAlgError as e:
            print(f"An error occurred while calculating
                Rosenbrock iteration for x = {x}")
            print(f"Previous y = {y}")
            print(e)
            exit(1)

        x_axis.append(x)
        x += h
        for i in range(self.system.num_of_equations):
            y_values[i].append(y[i])

        if self.debug:
```

```
                print("\n", x, y, "\n")

        return x_axis, y_values

    # p = 1 stands for Runge-Kutta method implication; Rosenbrock
        otherwise
    def main(self, h=0.5, p=1):
        Main.create_plots_dir()

        fig = plt.figure()

        start = time.time()
        x_axis, y_values = self.solve(h, p)
        end = time.time()
        print(f"Elapsed time = {end - start}")

        ax = fig.add_subplot(111)
        for i in range(self.system.num_of_equations):
            ax.plot(x_axis, y_values[i], label=f"y{i + 1}",
                marker='o', markersize=1)

        if p == 1:
            ax.set_title(f"Runge-Kutta method, h = {h}", fontsize=20)
        else:
            ax.set_title(f"Rosenbrock method, h = {h}", fontsize=20)

        ax.set_xlabel("x", fontsize=16)
        ax.set_ylabel("y", fontsize=16)
        ax.set_xscale('log')
        ax.set_yscale('log')
        ax.legend(fontsize=20)
        ax.grid()

        fig.set_figheight(20)
        fig.set_figwidth(25)
        fig.savefig(os.path.join("Plots", "Result.png"))


if __name__ == '__main__':
    main = Main(right_end=np.float64(1.0e+06))
```

```
main.main(h=np.float64(1.0), p=1)
```

// equation.py

```
import numpy as np


class Equation:
    num_of_equations = 0
    equations = []
    initial_conditions = None
    A = 0.0
    B = 0.0
    C = 0.0
    M = 0.0

    def __init__(self):
        self.num_of_equations = 4

        self.equations.append("y1' = -A * y1 - B * y1 * y3")
        self.equations.append("y2' = A * y1 - M * C * y2 * y3")
        self.equations.append("y3' = A * y1 - B * y1 * y3 - M * C *
            y2 * y3 + C * y4")
        self.equations.append("y4' = B * y1 * y3 - C * y4")

        self.initial_conditions = np.array([1.76e-03, 0.0, 0.0,
            0.0], dtype=np.float64)

        self.A = np.float64(7.89e-10)
        self.B = np.float64(1.1e+07)
        self.C = np.float64(1.13e+03)
        self.M = np.float64(1.0e+06)

    def get_equations(self):
        for eq in self.equations:
            print(eq)

        for i in range(len(self.initial_conditions)):
            print(f"y{i + 1}({self.initial_conditions[i][0]}) =
                {self.initial_conditions[i][1]}")
```

```
    def calculate_f(self, y):
        res = np.zeros(4, dtype=np.float64)
        res[0] = np.float64(-1.0) * self.A * y[0] - self.B * y[0] *
            y[2]
        res[1] = self.A * y[0] - self.M * self.C * y[1] * y[2]
        res[3] = self.B * y[0] * y[2] - self.C * y[3]
        res[2] = res[1] - res[3]

        return res

    def calculate_der_f(self, y):
        res = np.zeros(self.num_of_equations *
            self.num_of_equations,
            dtype=np.float64).reshape(self.num_of_equations,
                                                                                    sel

        res[0][0] = np.float64(-1.0) * self.A - self.B * y[2]
        res[0][2] = np.float64(-1.0) * self.B * y[0]
        res[1][0] = self.A
        res[1][1] = res[2][1] = np.float64(-1.0) * self.M * self.C *
            y[2]
        res[1][2] = np.float64(-1.0) * self.M * self.C * y[1]
        res[2][0] = self.A - self.B * y[2]
        res[2][2] = res[0][2] + res[1][2]
        res[2][3] = self.C
        res[3][0] = self.B * y[2]
        res[3][2] = self.B * y[0]
        res[3][3] = np.float64(-1.0) * self.C

        return res
```

// ssolver.py

```
import numpy as np


def inverse_lower_triangular(A):
    inversed = np.array(A)
    dim = inversed.shape[0]
```

```
    for i in range(dim):
        if A[i][i] == np.float64(0.0):
            raise ValueError("Matrix is singular")
        inversed[i][i] = np.float64(1.0) / A[i][i]

    for i in range(1, dim):
        for j in range(i - 1, -1, -1):
            tmp = np.float64(0.0)
            for k in range(i, j, -1):
                tmp += inversed[i][k] * A[k, j]

            inversed[i][j] = np.float64(-1.0) / A[j][j] * tmp

    return inversed


def solve_seidel(A, f, debug=False):
    if debug:
        print(f"Determinant in Seidel method equals
            {np.linalg.det(A)}")
    epsilon = np.float64(1.0e-20)

    dim = A.shape[0]

    L = np.tril(A, k=-1)
    D = np.zeros(dim * dim, dtype=np.float64).reshape(dim, dim)
    for i in range(dim):
        D[i][i] = A[i][i]
    U = A - L - D
    inversed = inverse_lower_triangular(L + D)

    init = np.zeros(dim, dtype=np.float64)
    res = np.float64(-1.0) * inversed.dot(U).dot(init) +
        inversed.dot(f)
    counter = 0
    while np.linalg.norm(res - init, ord=1) >= epsilon and counter <
        50:
        counter += 1
        init = res
        res = np.float64(-1.0) * inversed.dot(U).dot(init) +
```

```
            inversed.dot(f)

    if debug:
        print(f"Difference equals
            {np.linalg.norm(np.linalg.inv(A).dot(f) - res)}")

    return res


# in our case a = const = (I - h * alpha[0][0] * df/dy)^-1
def solve_newton(a, y_n, h, alpha, calculate_f, maxiter=50):
    if a.shape[0] != a.shape[1]:
        raise ValueError("Calculation matrix of the system must be
            square")

    epsilon = np.float64(1.0e-20)

    dim = a.shape[0]
    initial = np.zeros(dim, dtype=np.float64)
    res = a.dot(calculate_f(y_n))
    counter = 0
    while np.linalg.norm(res - initial) >= epsilon and counter <=
        maxiter:
        counter += 1
        initial = res
        res = initial - a.dot(initial - calculate_f(y_n + h * alpha
            * initial))

    return res
```

4 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 Решение обыкновенных дифференциальных уравнений. Жесткие и дифференциально-алгебраические задачи. Пер. с англ. — М.: Мир,

1999. — 685 с, ил. ISBN 5-03-003117-0