ЛАБОРАТОРНАЯ РАБОТА №1

«ЧИСЛЕННОЕ РЕШЕНИЕ ЗАДАЧИ КОШИ ДЛЯ ОДУ»

Выполнил

Святослав Артюшкевич, 3 группа, 3 курс

Условие задачи

Написать программу реализующую предиктор-корректор с автоматическим выбором шага интегрирования. Экспериментально сравнить эффективность данного метода с явным методом Эйлера.

Рассмотрим двойной маятник состоящий из двух невесомых жёстких стержней длин l_1 и l_2 точечных масс m_1и m_2. Движение системы полностью описывается двумя функциями определяющими изменение во времени углов α_1и α_2. Эти функции удовлетворяют следующей системе ОДУ второго порядка.

$$\begin{cases} \alpha_1'' = \dfrac{-g(2m_1 + m_2)\sin\alpha_1 - m_2 g\sin(\alpha_1 - 2\alpha_2) - 2m_2((l_1(\alpha_1')^2\cos\alpha + l_2(\alpha_2')^2)\sin\alpha}{l_1(2m_1 + m_2 - m_2\cos 2\alpha)} \\ \alpha_2'' = \dfrac{2\sin\alpha\,(l_1 m(\alpha_1')^2 + gm\cos\alpha_1 + l_2 m_2(\alpha_2')^2\cos\alpha)}{l_2(2m_1 + m_2 - m_2\cos 2\alpha)} \end{cases}$$

$$m = m_1 + m_2$$

$$\alpha = \alpha_1 + \alpha_2$$

Подготовка к решению

Так как у нас в системе ОДУ второго порядка, представим их в виде первого. Сделаем замену

$$y_1 = \alpha_1$$

$$y_2 = \alpha_1'$$

$$y_3 = \alpha_2$$

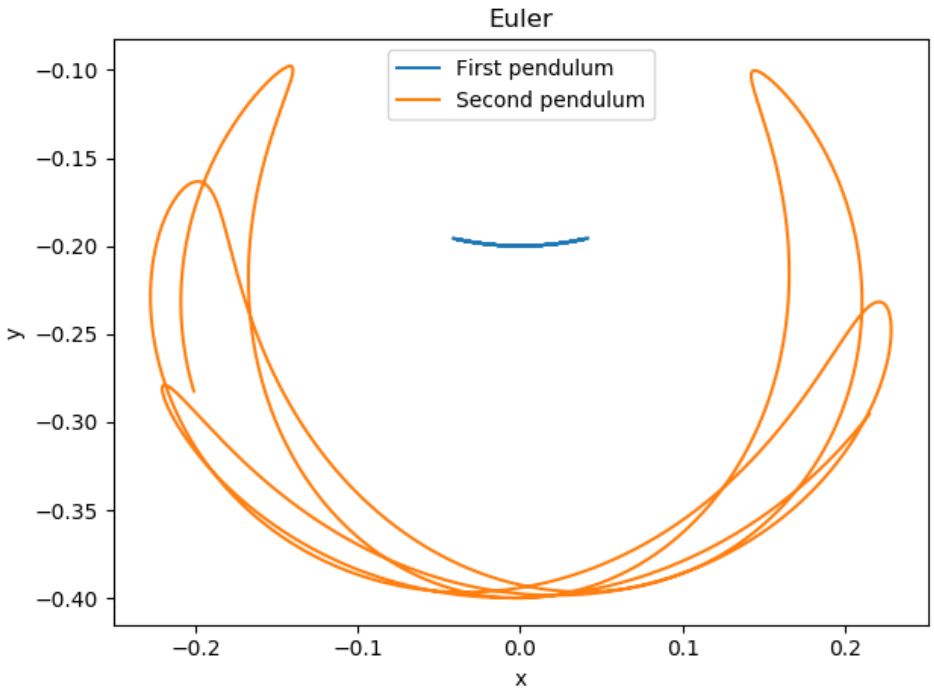
$$y_4 = \alpha_2'$$

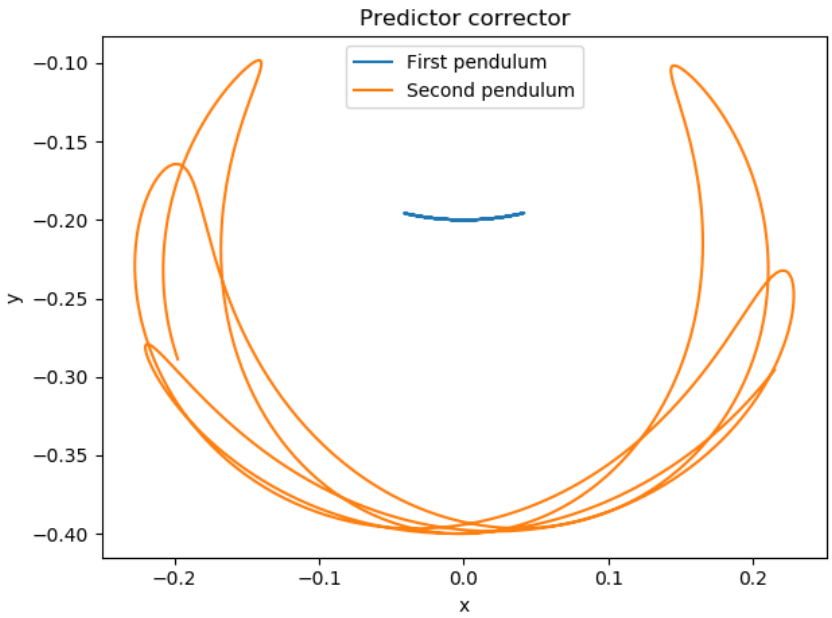
Теперь будем решать систему вида

$$\begin{cases} y_1 = \alpha_1' \\ y_2 = \alpha_1'' \\ y_3 = \alpha_2' \\ y_4 = \alpha_2'' \end{cases}$$

Результаты

tol	Метод	N_{tot}	N_a	N_r	t
1e-6	Предиктор корректор	1579	1575	4	0.613
1e-6	Метод Эйлера	21827	21826	1	3.119





Код программы

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

tol = 1e-6
m_1 = 3
m_2 = 0.01
l_1 = 0.2
l_2 = 0.2
g = 9.8


def f(t, alpha):
    m = m_1 + m_2
    a = alpha[0] - alpha[2]
    return np.array([alpha[1], (
            -g * (2 * m_1 + m_2) * np.sin(alpha[0]) - m_2 * g * np.sin(alpha[0] - 2 *
alpha[2]) - 2 * m_2 * (
            l_1 * (alpha[1] ** 2) * np.cos(a) + l_2 * (alpha[3] ** 2)) * np.sin(a)) /
(
                               l_1 * (2 * m_1 + m_2 - m_2 * np.cos(2 * a))), alpha[3],
(2 * np.sin(a) * (
            l_1 * m * (alpha[1] ** 2) + g * m * np.cos(alpha[0]) + l_2 * m_2 *
(alpha[3] ** 2) * np.cos(a))) / (
                               l_2 * (2 * m_1 + m_2 - m_2 * np.cos(2 * a)))])


def euler(x_0, y_0, h, func):
    return y_0 + h * func(x_0, y_0)


def predictor_corrector(x_0, y_0, h, func):
    y_2 = y_0 + h * func(x_0, y_0)
    return y_0 + (h / 2) * (func(x_0, y_0) + func(x_0 + h, y_2))


def choose_step(method, p, func, x_0, y_0, h_0, alpha=0.9):
    h = h_0
    N_r = 0
    while True:
        y_1 = method(x_0, y_0, h, func)
        y_2 = method(x_0 + h, y_1, h, func)
        y_2_2 = method(x_0, y_0, 2 * h, func)
        err = np.linalg.norm(y_2 - y_2_2) / (2 ** p - 1)
        delta = (tol / err) ** (1.0 / (p + 1))
        h_new = alpha * delta * h
        if delta < 1:
            h = h_new
            N_r += 1
            continue
        return x_0 + 2 * h, y_2, h_new, N_r


def run(method, p, h_0, T):
    x = 0
    y = np.array([np.pi / 15, 0, np.pi / 3, 0])
    h = h_0
    N_tot = 0
    N_a = 0
```

```
    N_r = 0
    y_arr = [y]
    while x < T:
        x, y, h, dN_r = choose_step(method, p, f, x, y, h)
        y_arr.append(y)
        N_r += dN_r
        N_a += 1
        h = min(h, (T - x) / 2)
    N_tot = N_a + N_r
    return y_arr, N_tot, N_a, N_r


def main():
    start_time = time.time()
    alpha, N_tot, N_a, N_r = run(predictor_corrector, 1, 1.0, 3)
    end_time = time.time()
    x_1, y_1, x_2, y_2 = [], [], [], []
    for value in alpha:
        x_1.append(l_1 * np.sin(value[0]))
        y_1.append(-l_1 * np.cos(value[0]))
        x_2.append(x_1[-1] + l_2 * np.sin(value[2]))
        y_2.append(y_1[-1] - l_2 * np.cos(value[2]))
    print("Time:", "%.3f" % (end_time - start_time))
    print("N_tot:", N_tot)
    print("N_a:", N_a)
    print("N_r:", N_r)
    plt.plot(x_1, y_1, label="First pendulum")
    plt.plot(x_2, y_2, label="Second pendulum")
    plt.title("Predictor corrector")
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()


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