Лабораторная работа 3 по курсу «Нелинейная динамика и её приложения». Отчёт.

Владислав Соврасов
381503м4

1 Численное интегрирование уравнений методом Эйлера с постоянным шагом

Пусть необходимо решить численно уравнение вида $\dot{x}(t) = f(x,t)$ с начальным условием $x(0) = x_0$. $x(t)$ и $f(x,t)$ в рассматриваемом уравнении могут быть, вообще говоря, вектор-функциями. Метод Эйлера имеет простую вычислительную формулу:

$$x_{n+1} = x_n + hf(x_n, h \cdot n), n = \overline{0, s},$$

где $h > 0$ — шаг метода.

Рассмотрим работу метода на примере интегрирования уравнений $\dot{x}(t) = \pm x, x(0) = 1$. В случае положительной правой части решением уравенения является функция $x(t) = e^x$. Метод Эйлера при решении этого уравнения строит последовательность, которая растёт как геометрическая прогрессия со знаменателем $h+1$, в то время, как последовательность значений функции-решения в узлах сетки является геометрической прогрессией со знаменателем e^h. $e^h - 1 - h = h^2/2 + O(h^3) > 0$ при малом h, поэтому разность между численным и истинным решениями на сетке будет расти также как геометрическая прогрессия, т.е. экспоненциально. Этот эффект виден на рис. 1. В случае решения уравнения $\dot{x}(t) = -x, x(0) = 1$ верны те же самые рассуждения, но прогрессия здесь будет убывающей (рис. 2).

Из рис. 1, 2 также видно, что метод Эйлера имеет первый порядок сходимости: при уменьшении шага в 10 раз ошибка интегрирования на отрезке падает примерно в 10 раз.

Далее рассмотрим интегрирование уравнения второго порядка: $\ddot{x} + x = 0$ при начальных условиях $x(0) = 1, \dot{x}(0) = 0$. Это уравнение после введения фазовой скорости $y(t) = \dot{x}(t)$ эквивалентно системе:

$$\begin{cases} \dot{x} = y \\ \dot{y} = -x \end{cases}$$

Эту систему можно решать методом Эйлера. В качестве ошибки интегрирования рассматривается абсолютная разность численного и настоящего решений по координате x. Решением исходного уравнения при заданных начальных условиях является функция $x(t) = cos(t)$. Как видно из рис. 3, ошибка похожа на периодическую функцию, однако со временем нарастает. Минимумы этой функции соответствуют нулям решения $x(t) = cos(t)$. Как и на предыдущих графиках, здесь также подтверждается первый порядок сходимости метода Эйлера.

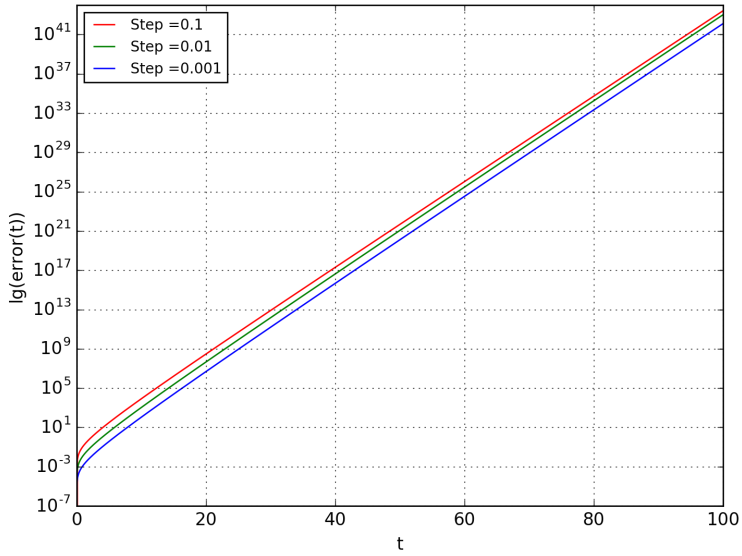

Рис. 1: Ошибка интегрирования уравнения $\dot{x}(t) = x$ при различных значениях шага

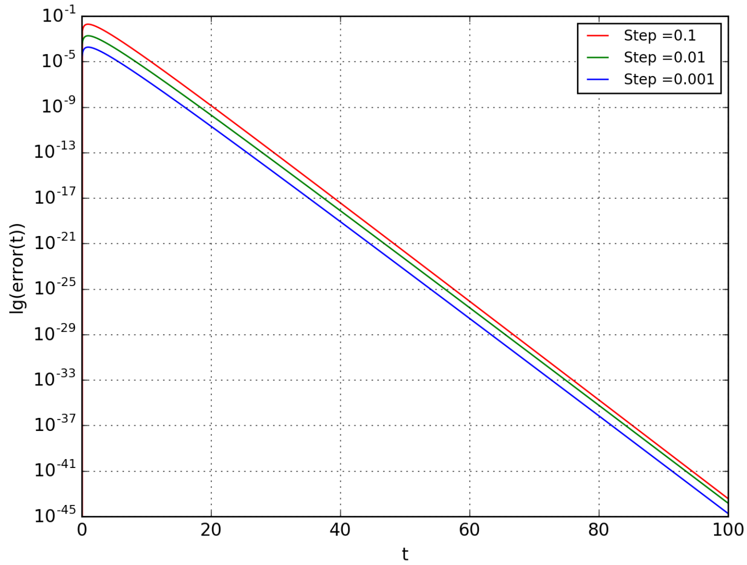

Рис. 2: Ошибка интегрирования уравнения $\dot{x}(t) = -x$ при различных значениях шага

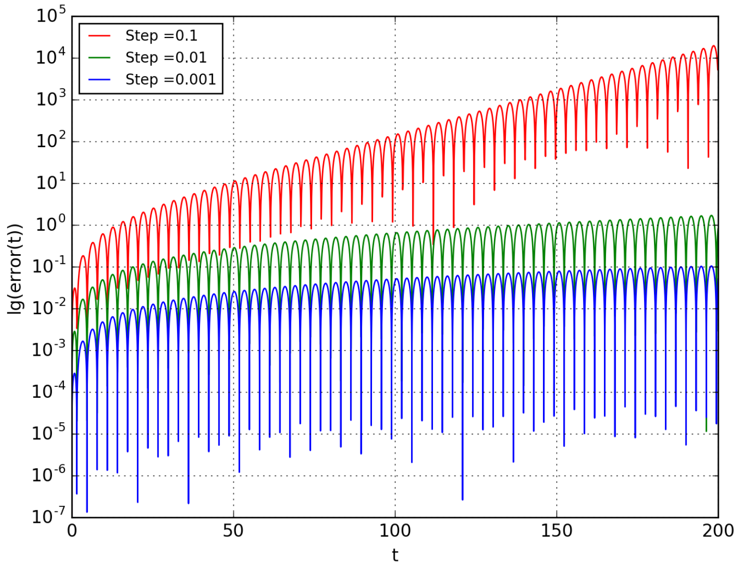

Рис. 3: Ошибка интегрирования уравнения $\ddot{x}(t) + x = 0$ при различных значениях шага

2 Численное интегрирование системы Рёсслера методом Рунге-Кутты с постоянным шагом

Рассмотрим систему из трёх уравнений:

$$\begin{cases} \dot{x} = -y - z \\ \dot{y} = x + 0.3y \\ \dot{z} = 0.3 + (x - 5.7)z \end{cases}$$

при начальных условиях $x(0) = 1, y(0) = 0, z(0) = 100$.

Для численного решения этой системы применялся метод Рунге-Кутты 4го порядка с постоянным шагом. Точное решение неизвестно, поэтому для оценки ошибки интегрирования использовалось численное решение, полученное тем же методом, но с вдвое меньшим шагом. На рис. 4 показана разность численных решений по норме $L1$. Из графика виден четвёртый порядок сходимости метода, а также то, что оценка ошибки интегрирования сначала нарастает, а потом становится похожей на периодическую функцию. При шаге 10^{-1} видно, что на большом промежутке времени оценка ошибки нарастает, однако в этом случае нет уверенности в том, что поведение оценки полностью совпадает с поведением реальной ошибки, т.к. шаг дополнительной сетки достаточно большой. При меньших значениях шага интегрирования оценка ошибки имеет более сложное поведение, которое меняется с изменением величины шага. Тем не менее, схема с двойной сеткой позволяет оценить порядок ошибки в случае использования шага, равного 10^{-2} или 10^{-3}. Этот порядок является вполне приемлемым для интегрирования системы.

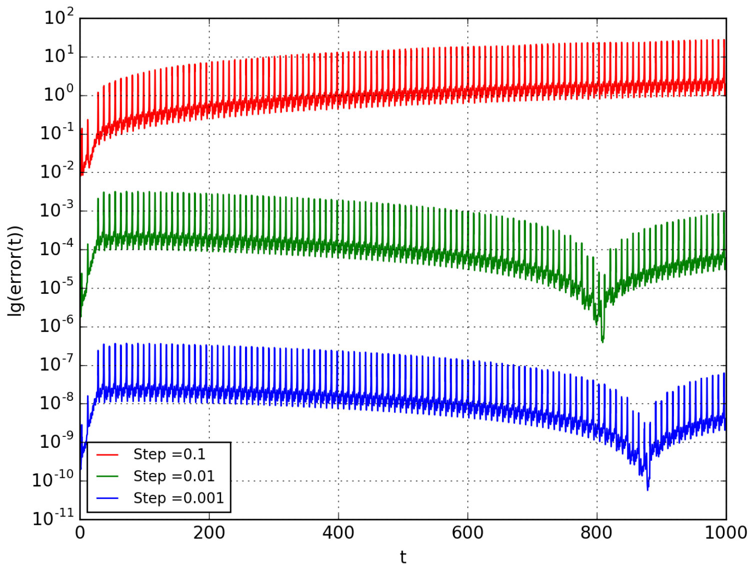

Рис. 4: Оценка ошибки интегрирования системы Рёсслера при различных значениях шага

3 Исходный код

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt

def eulerMethod(f, a, b, f_0, step):
    numSteps = int((b - a) / step - 0.5)

    xValues = [0.]*(numSteps + 1)
    tValues = [0.]*(numSteps + 1)
    xValues[0] = f_0
    tValues[0] = a

    for i in range(0, numSteps):
        xValues[i + 1] = xValues[i] + step*f(xValues[i], tValues[i])
        tValues[i + 1] = tValues[i] + step

    return tValues, xValues

def rungeKuttaMethod(f, a, b, f_0, step):
    numSteps = int((b - a) / step - 0.5)

    xValues = [0.]*(numSteps + 1)
```

```
    tValues = [0.]*(numSteps + 1)
    xValues[0] = f_0
    tValues[0] = a
    step2 = step / 2

    for i in range(0, numSteps):
        k1 = f(xValues[i], tValues[i])
        k2 = f(xValues[i] + step2*k1, tValues[i] + step2)
        k3 = f(xValues[i] + step2*k2, tValues[i] + step2)
        k4 = f(xValues[i] + step*k3, tValues[i] + step)
        xValues[i + 1] = xValues[i] + step * (k1 + 2*k2 + 2*k3 +k4) / 6.
        tValues[i + 1] = tValues[i] + step

    return tValues, xValues

def plotError(steps, tValues, errValues, outputName):
    colors = ['r', 'g', 'b', 'c']

    plt.xlabel('t')
    plt.ylabel('lg(error(t))')
    plt.yscale('log')

    for i, step in enumerate(steps):
        plt.plot(tValues[i], errValues[i], \
            colors[i] + '-', label='Step =' + str(step))

    plt.grid()
    plt.legend(loc = 'best', fontsize = 10)
    plt.savefig(outputName, format = 'png', dpi = 200)
    plt.clf()

def main():

    steps = [0.1, 0.01, 0.001]
    f = lambda x, t: x

    tValues = []
    errValues = []
    for i, step in enumerate(steps):
        timeGrid, xValues = eulerMethod(f, 0., 100., 1., step)
        errValues.append(np.abs(xValues - np.exp(timeGrid)))
        tValues.append(np.array(timeGrid))

    plotError(steps, tValues, errValues, '../pictures/lab3_exp_plus.png')

    f = lambda x, t: -x

    tValues = []
    errValues = []
```

```python
    for i, step in enumerate(steps):
        timeGrid, xValues = eulerMethod(f, 0., 100., 1., step)
        errValues.append(np.abs(xValues - np.exp(-np.array(timeGrid))))
        tValues.append(np.array(timeGrid))

    plotError(steps, tValues, errValues, '../pictures/lab3_exp_minus.png')

    f = lambda x, t: np.array([x[1], -x[0]])

    tValues = []
    errValues = []
    for i, step in enumerate(steps):
        timeGrid, xValues = eulerMethod(f, 0., 200., np.array([1.,0.]), step)
        errValues.append(np.abs(np.array(xValues)[:,0] \
            - np.cos(np.array(timeGrid))))
        tValues.append(np.array(timeGrid))

    plotError(steps, tValues, errValues, '../pictures/lab3_cons_system.png')

    f = lambda x, t: np.array([-x[1] - x[2], x[0] + 0.3*x[1], \
        0.3 + (x[0] - 5.7)*x[2]])

    tValues = []
    errValues = []
    for i, step in enumerate(steps):
        timeGrid, xValues = \
            rungeKuttaMethod(f, 0., 1000., np.array([1.,0., 100.]), step)
        _, xValues2 = \
            rungeKuttaMethod(f, 0., 1000., np.array([1.,0., 100.]), step / 2.)
        errValues.append(np.sum(np.abs(np.array(xValues) - \
            np.array(xValues2)[::2]), axis=1))
        tValues.append(np.array(timeGrid))

    plotError(steps, tValues, errValues, '../pictures/lab3_rossler_system.png')

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