数值分析第 10 章上机

By 211870125 陈睿硕

§1 问题

对于常微分方程初值问题：

$$\begin{cases} y' = -\frac{1}{x^2} - \frac{y}{x} - y^2, 1 \leqslant x \leqslant 2, \\ y(1) = -1. \end{cases}$$

Q1 分别用 Euler 方法、改进的 Euler 方法、Heun 公式、中点方法和四阶 Runge-Kutta 方法求解上述初值问题，列表、画图比较他们的计算结果。

Q2 用经典的 Runge-Kutta 方法提供初始出发值，与四阶 Adams 预测-校正方法的 PECE 模式结合起来求解上述初值问题，并与**Q1**中的四阶 Runge-Kutta 方法求解进行比较。

§2 算法思路

I 对Q1，Q2的解答

使用 python 实现，代码见代码 1。计算出初值问题的精确解为：$y = (x\tan(\ln(x) - \frac{\pi}{4}))^{-1}$。取 $h = 0.0001$，得到误差（绝对值）和 x 的关系如图 1。

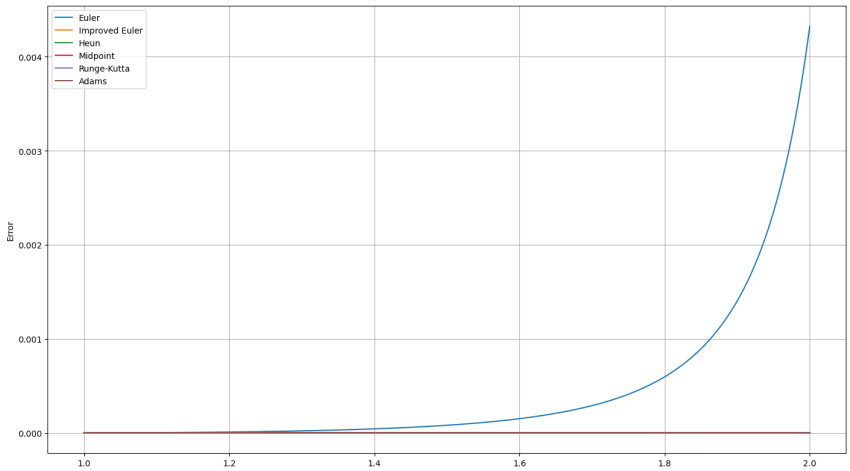

图 1: 误差（绝对值）和 x 的关系

得到 x=2 处的误差比较如表 1。

Method	Error
Euler 方法	0.004321987481701761
改进的 Euler 方法	4.807119715621866e-07
Heun 公式	7.116845477384004e-07
中点方法	8.271718101582337e-07
四阶 Runge-Kutta 方法	3.0047075938455237e-12
Adams 方法	3.375077994860476e-12

表 1: x=2 处的误差

§3 结果分析

I 对Q1，Q2的分析

如表 1，各种方法中 Euler 方法作为最朴实无华的方法，收敛速度远远慢于其他方法，而改进的 Euler 方法、Heun 公式、中点方法收敛速度较为接近，四阶 Runge-Kutta 方法、Adams 方法拥有阶数和步数上的优势，所以有更快的收敛速度。

§4 附录：程序代码

代码 1: Tenth Chapter.py

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

def y(x):
    return 1/(x*np.tan(np.log(x)-np.pi/4))


h = 0.0001
x = np.arange(1, 2+h, h)
y0 = -1


def f(x, y):
    return -1/x**2 - y/x - y**2


def euler(x, y0, f):
    y = np.zeros(x.shape)
    y[0] = y0
    for i in range(len(x) - 1):
```

```
        y[i+1] = y[i] + h * f(x[i], y[i])
    return y


def improved_euler(x, y0, f):
    y = np.zeros(x.shape)
    y[0] = y0
    for i in range(len(x) - 1):
        k1 = f(x[i], y[i])
        k2 = f(x[i+1], y[i] + h * k1)
        y[i+1] = y[i] + h * (k1 + k2) / 2
    return y


def heun(x, y0, f):
    y = np.zeros(x.shape)
    y[0] = y0
    for i in range(len(x) - 1):
        k1 = f(x[i], y[i])
        k2 = f(x[i] + 2*h/3, y[i] + 2*h/3 * k1)
        y[i+1] = y[i] + h * (k1 + 3*k2) / 4
    return y


def midpoint(x, y0, f):
    y = np.zeros(x.shape)
    y[0] = y0
    for i in range(len(x) - 1):
        k1 = f(x[i], y[i])
        k2 = f(x[i] + h/2, y[i] + h/2 * k1)
        y[i+1] = y[i] + h * k2
    return y


def runge_kutta(x, y0, f):
    y = np.zeros(x.shape)
    y[0] = y0
    for i in range(len(x) - 1):
        k1 = f(x[i], y[i])
        k2 = f(x[i] + h/2, y[i] + h/2 * k1)
        k3 = f(x[i] + h/2, y[i] + h/2 * k2)
```

```
        k4 = f(x[i] + h, y[i] + h * k3)
        y[i+1] = y[i] + h * (k1 + 2*k2 + 2*k3 + k4) / 6
    return y

def adams(x, y0, dydx, runge_kutta):
    y = np.zeros(x.shape)
    y[:4] = runge_kutta(x[:4], y0, dydx)
    for i in range(3, len(x) - 1):
        y_pred = y[i] + h/24 * (55*dydx(x[i], y[i]) - 59*dydx(x[i-1], y[i
            -1]) + 37*dydx(x[i-2], y[i-2]) - 9*dydx(x[i-3], y[i-3]))
        y[i+1] = y[i] + h/24 * (9*dydx(x[i+1], y_pred) + 19*dydx(x[i], y[
            i]) - 5*dydx(x[i-1], y[i-1]) + dydx(x[i-2], y[i-2]))
    return y

y_precise=y(x)
error_euler = np.abs(y_precise-euler(x, y0, f))
error_improved_euler = np.abs(y_precise-improved_euler(x, y0, f))
error_heun = np.abs(y_precise-heun(x,y0, f))
error_midpoint = np.abs(y_precise-midpoint(x, y0, f))
error_runge_kutta = np.abs(y_precise-runge_kutta(x, y0, f))
error_adams = np.abs(y_precise-adams(x, y0, f, runge_kutta))

plt.figure(figsize=(12, 8))
plt.plot(x, error_euler, label='Euler')
plt.plot(x, error_improved_euler, label='Improved Euler')
plt.plot(x, error_heun, label='Heun')
plt.plot(x, error_midpoint, label='Midpoint')
plt.plot(x, error_runge_kutta, label='Runge-Kutta')
plt.plot(x, error_adams, label='Adams')
print(error_euler[-1],error_improved_euler[-1],error_heun[-1],
        error_midpoint[-1],error_runge_kutta[-1],error_adams[-1])
plt.legend(loc='best')
plt.ylabel('Error')
plt.grid(True)
plt.show()
```