本科生实验报告

实验课程: 常微分方程

任课教师: 汪涛

实验题目: 数值解微分方程

专业名称: 信息与计算科学

学生姓名: 郑鸿鑫

学生学号: 22336313

实验时间: 2024/5/11

Section 1 实验概述

在实用上有重大意义的很多微分方程，即使他们能满足解的存唯一性条件，但他们的解常常不能表达为初等函数的形式，对于这类微分方程解的讨论，最常用的方法就是数值积分，也就是对常微分方程求数值解。本次实验将涉及欧拉法和龙格-库塔法，并通过编程来实现他们。

Section 2 预备知识与实验环境

- 预备知识：数值积分相关知识，欧拉法和龙格-库塔法的相关知识、Python 程序设计基础
- 实验环境：Windows11，Python 3.12.0
  - ■ 代码编辑环境：PyCharm Professional 2023.3.5
  - ■ 代码编译工具：pip（用于安装第三方库）
  - ■ 重要三方库信息：matplotlib 3.3.2,Python 标准库,math 库

Section 3 实验任务

- 实验任务 1：推导出三阶龙格-库塔公式。
- 实验任务 2：编程实现欧拉法和龙格-库塔法，用三种不同的方法求解两个微分方程。
- 实验任务 3：求出任务 2 中方程 a 的精确解比较分析与数值解之间的误差。

Section 4 实验步骤与实验结果

------------------------- 实验任务 1 -------------------------

- 实验步骤：

推导三阶龙格-库塔公式，推导过程如下所示：

$$\begin{cases} y_{n+1} = y_n + h(\lambda_1 k_1 + \lambda_2 k_2 + \lambda_3 k_3) & ① \\ k_1 = f_n = f(x_n, y_n) & ② \\ k_2 = f(x_n + \alpha_2 h,\ y_n + h\beta_{21} k_1) & ③ \\ k_3 = f(x_n + \alpha_3 h,\ y_n + h[\beta_{31} k_1 + \beta_{32} k_2]) & ④ \end{cases}$$

将 k_2 按泰勒展开得： 以下 $f(x_n, y_n)$ 用 f_n 简写

$$k_2 = f_n + \alpha_2 h f_x + h\beta_{21} f_n f_y + \frac{1}{2}(\alpha_2^2 h^2 f_{xx} + 2\alpha_2 h^2 \beta_{21} f_n f_{xy} + h^2 \beta_{21}^2 f_n^2 f_{yy}) + O'' \quad \blacktriangle$$

(O'' 指位于该点展开式的余项)

对 k_3 作相似的处理：

$$k_3 = f_n + \alpha_3 h f_x + h[\beta_{31} f_n + \beta_{32} k_2] f_y$$

$$+ \frac{1}{2}(\alpha_3^2 h^2 f_{xx} + 2\alpha_3 h^2 (\beta_{31} f_n + \beta_{32} k_2) f_{xy} + h^2 (\beta_{31} f_n + \beta_{32} k_2)^2 f_{yy})$$

将 k_2 用式 $\blacktriangle$ 代入得：

$$k_3 = f_n + \alpha_3 h f_x + \beta_{31} h f_n f_y + \beta_{32} h (f_n + \alpha_2 h f_x) f_y$$
$$+ \beta_{32}\beta_{21} h^2 f_n f_y f_y + \frac{1}{2}(\alpha_3^2 h^2 f_{xx} + 2\alpha_3 h^2 \beta_{31} f_n f_{xy}$$
$$+ 2\alpha_3 h^2 \beta_{32} f_n) + (\beta_{31}^2 + \beta_{32}^2 + 2\beta_{31}\beta_{32}) h^2 f_n^2 f_{yy} + O'' \quad \blacksquare$$

再对 $y(x_{n+1})$ 在点 x_n 处泰勒展开式

$$y(x_{n+1}) = y(x_n + h)$$
$$= y(x_n) + hy' + \frac{h^2}{2} y'' + \frac{h^3}{6} y''' + O(h^4)$$
$$= y_n + h f_n + \frac{h^2}{2}(f_x + f_y f) + \frac{h^3}{6}(f_{xx} + 2 f_{xy} f$$
$$+ f_y f_x + f_{yy} f^2 + f_y f_y f) + O(h^4)$$

由式▲和■代入①再与上述展开式比较（注：$O(h^4)$ 余项略去）

得：
$$
\begin{cases}
\lambda_1+\lambda_2+\lambda_3 = 1 \\
\alpha_2 = \beta_{21} \\
\alpha_3 = \beta_{31}+\beta_{32} \\
\lambda_2\alpha_2+\lambda_3\alpha_3 = \frac{1}{2} \\
\lambda_2\alpha_2^2+\lambda_3\alpha_3^2 = \frac{1}{3} \\
\lambda_2\beta_{32}\alpha_2 = \frac{1}{6}
\end{cases}
$$

方程组有 8 个未知数，6 个方程，故解不唯一

故取 $\lambda_1 = \lambda_3 = \frac{1}{6}$

则
$$
\begin{cases}
\lambda_2 = \frac{4}{6} \\
\alpha_2 = \frac{1}{2} \\
\alpha_3 = 1 \\
\beta_{32} = 2 \\
\beta_{31} = -1 \\
\beta_{21} = \frac{1}{2}
\end{cases}
$$

故三阶龙格-库塔公式为：
$$
\begin{cases}
y_{n+1} = y_n + h(\frac{1}{6}k_1+\frac{4}{6}k_2+\frac{1}{6}k_3) \\
k_1 = f(x_n, y_n) \\
k_2 = f(x_n+\frac{h}{2}, y_n+\frac{h}{2}k_1) \\
k_3 = f(x_n+h, y_n-hk_1+2hk_2)
\end{cases}
$$

------------------------- **实验任务 2** -------------------------

- 任务要求：

  编程实现欧拉法和 3 阶，4 阶龙格-库塔法，三种方法分别求解下列微分方程

  (a) $\frac{dy}{dx} = 5y - 6e^{-x}, y(0) = 1 + 1.0E13$，$x \in [0,6], h = 0.01,0.001,0.0001;$

  $(b)\frac{dy}{dx} = (x+y)\sin(xy), y(0) = 5, x \in [0,6], h = 0.2,0.1,0.01$

- 思路分析：按照三种不同的方法用 Python 实现其求解过程，并绘

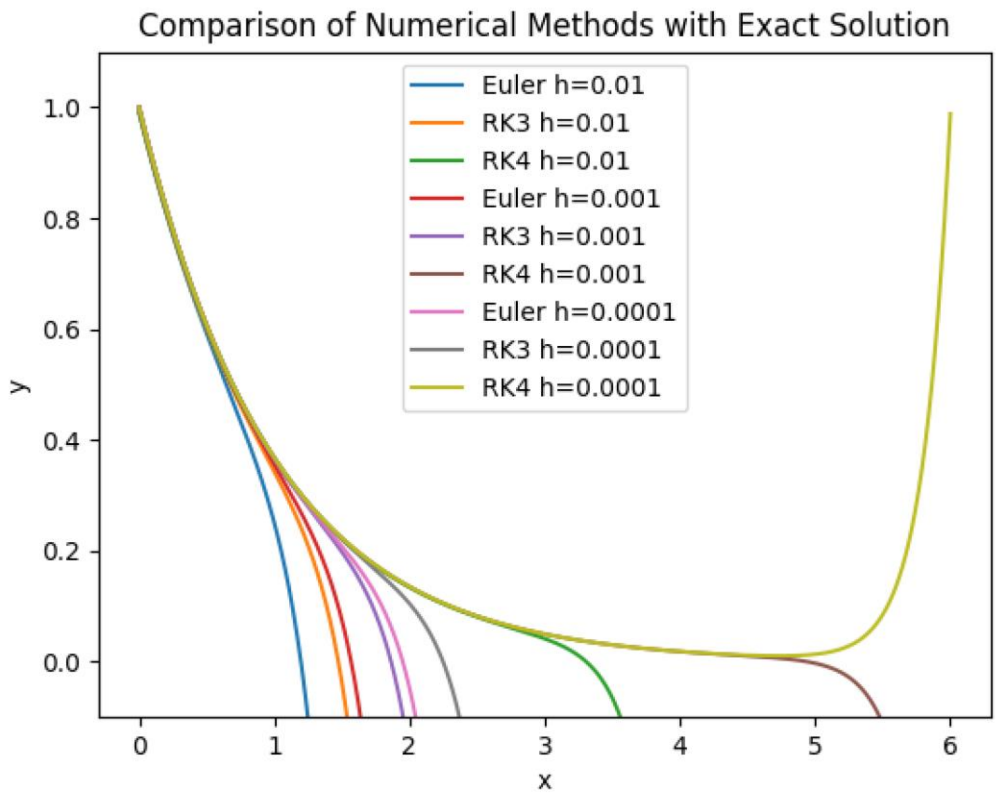
制图像观察

- 实验步骤：
  1. 首先编写三种方法的实现函数，并且以数组的形式返回得到的数值解。
  2. 设置好初值条件和步长等参数进行求解。
  3. 求解后绘制得到的数值解的图像

  （详细代码见 Section7 代码清单）

- 实验结果展示：

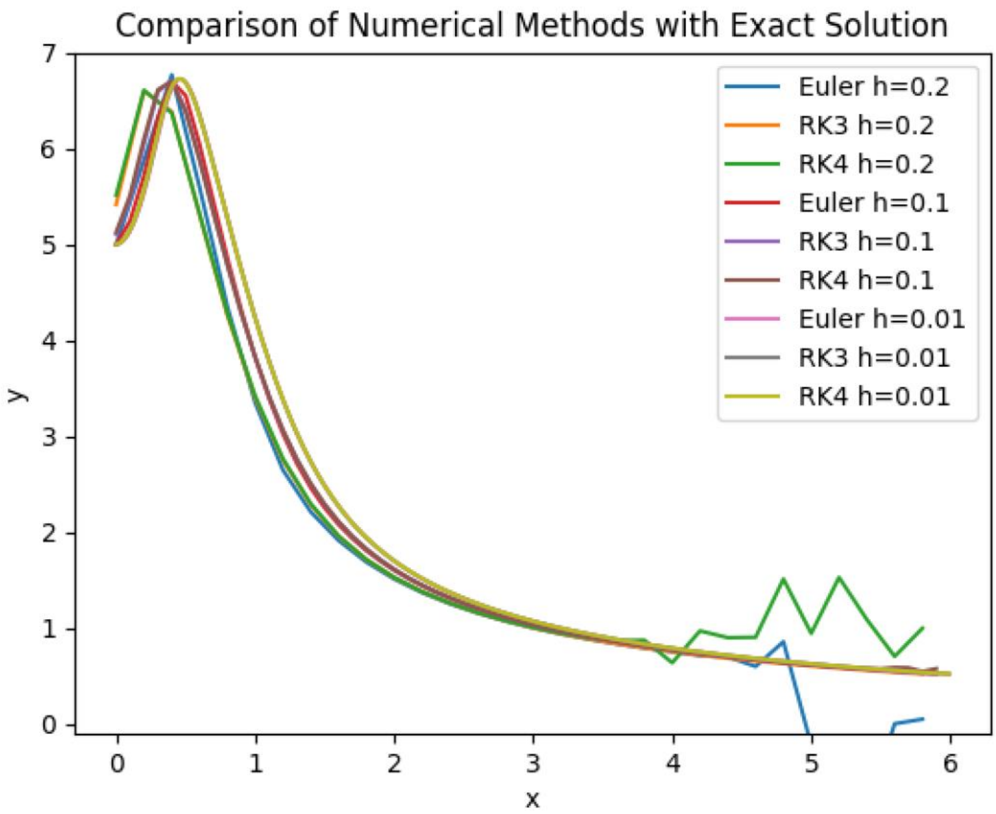
  a. 方程 a 求解：



可以看到对于三种方法，在 x 较小时都维持非常接近的值，但是由于只有一阶精度，欧拉法很快产生了较大误差，RK3 有三阶精度，在后半段产生偏差，RK4 有四阶精度，但是也只有在 h = 0.0001 的时候完美拟合（在任务三中会详细说明）。同时可以看出，随着步长的减小，

三种方法的误差都在变小。

b. 方程 b 求解：



分析与方程 a 类似，但是可以看到方程 b 的近似效果要比 a 好很多，并且还是在步长较大的情况下，可以解释为方程 b 对数值解的精度要求不是很高。欧拉法和 RK4 只是在 h = 0.2 时，在很靠后的位置出现偏差，而 RK3 在给定的三个步长中都表现的异常好，这有些难以解释，我们猜想可能的原因：

1. 阶数的影响：RK3 是一个三阶的 Runge-Kutta 方法，而 RK4 是一个四阶的方法。通常情况下，更高阶的方法会提供更高的数值精度。然而，在特定情况下，低阶方法可能会更适合，特别是在步长较大时。可能在这种情况下，RK3 的误差受到了更少的累积影响，导致比 RK4 更准确的结果。

2. 数值稳定性：RK4 是一个更复杂的方法，可能在某些情况下会导致数值不稳定性，尤其是在步长较大时。相比之下，RK3 可能更稳定，使得在较大步长下产生更准确的结果。

------------------------- 实验任务 3 -------------------------

- 任务要求：手动计算任务 2 中方程 a 的精确解，并比较精确解和数值解的相似度
- 实验步骤：

1. 采用常数变易法求解，过程如下：

$$\frac{dy}{dx} = 5y - 6e^{-x}$$

先求齐次解： $\frac{dy}{dx} = 5y$

变量分离法知： $y = e^{5x} \cdot c$

由常数变易法得： $y = e^{5x} \cdot c(x)$

$$\frac{dy}{dx} = 5e^{5x} \cdot c(x) + e^{5x} \cdot c'(x)$$

$$= 5y + e^{5x} \cdot c'(x)$$

$$\therefore c'(x) = -6e^{-6x}$$

$$\therefore c(x) = e^{-6x} + c_1$$

$$\therefore y = e^{5x} \cdot (e^{-6x} + c_1)$$

$$= c_1 e^{5x} + e^{-x}$$

其中 c_1 为任意常数

由于 $y(0) = 1.0 \times 10^{-13}$

$$\therefore y(0) = c_1 \times 1 + 1 = 1 + 10^{-13}$$

$$c_1 = 10^{-13}$$

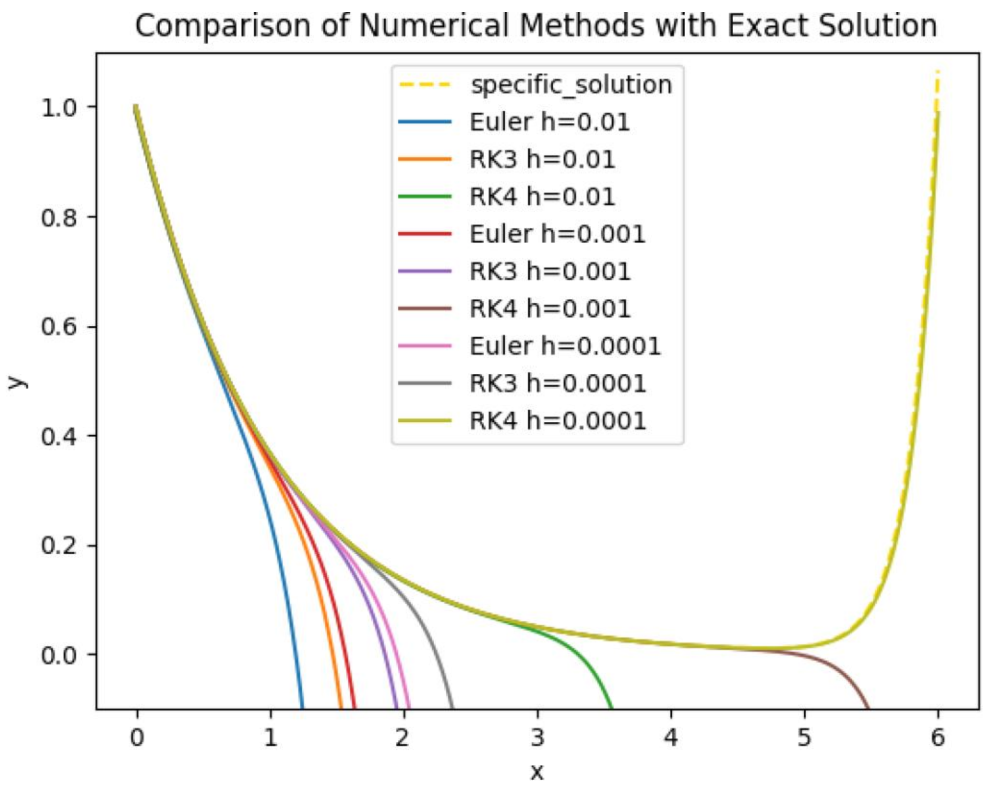
$$\therefore y = 10^{-13} \cdot e^{5x} + e^{-x}$$

2. 将精确解加入到代码中，然后绘制图像，比较和数值解间的区别（详细代码见 Section7 代码清单）

● 实验结果展示：

加入方程 a 的精确解后，得到图像如下：

（为了明显易看，我们使用黄色的虚线绘制出了精确解）



由这幅图可以解释任务 2 中我们仍留下的问题，由于黄色的虚线也就是精确解，与 RK4 在 h = 0.0001 的情况下的曲线完全拟合，所以在所有进行实验的测试中只有这种情况下能与精确解完美拟合，而其他情形都会出现或多或少的误差。

Section 5 实验总结与心得体会

◆ 实验总结：

本次实验通过实现欧拉法、三阶龙格-库塔法（RK3）和四阶龙格-库塔法（RK4）来求解给定的一阶常微分方程。实验中，我们使用了不同的步长来观察数值解的精度，并与已知的精确解进行了比较。通过绘图，我们直观地展示了不同数值方法的解以及它们

与精确解之间的差异。

实验结果表明，随着步长的减小，所有数值方法的解都趋向于精确解，但四阶龙格-库塔法（RK4）在相同步长下提供了最高的精度。此外，实验也展示了数值方法的误差分析，通过比较不同方法和步长的误差，我们可以更好地理解各种方法的优缺点。

◆ 心得体会：

通过本次实验，我深刻体会到了数值方法在解决实际问题中的应用价值。我学习到了如何使用 Python 实现不同的数值求解方法，并通过实际操作加深了对这些方法理论基础的理解。实验过程中，我认识到步长选择对数值解精度的重要性，以及不同方法之间精度和稳定性的权衡。

此外，我也意识到了误差分析在数值计算中的重要性。通过比较不同数值解与精确解之间的误差，我学会了如何评估和选择合适的数值方法。整体而言，这次实验不仅提升了我的编程技能，也加深了我对数值分析知识的认识，对我的学习和研究具有重要意义。

Section 6 附录：参考资料清单

1. 常微分方程算法之龙格-库塔法（Runge-Kutta 法）_龙格库塔法-CSDN 博客
2. 1-1 (open.com.cn)
3. 三阶_龙格-库塔公式 详细推导过程 - 百度文库 (baidu.com)

Section 7 附录：代码清单

本实验完整的代码如下：

```python
import numpy as np
import math
import matplotlib.pyplot as plt
# 精确解函数
def specific_solution(x):
    ans = []
    for i in x:
        y = 1e-13*math.exp(5*i)+math.exp(-1*i)
        ans.append(y)
    return np.array(ans)
# 欧拉法
def euler_method(f,x0, y0, h, x1):
    ans = []
    x = x0
    y = y0
    while x < x1:
        y += h * f(x, y)
        x += h
        ans.append(y)
    return ans
# 三阶龙格-库塔法
def rk3_method(f,x0, y0, h, x1):
    ans = []
    x = x0
    y = y0
    while x < x1:
        k1 = h * f(x, y)
        k2 = h * f(x + h / 2, y + k1 / 2)
        k3 = h * f(x + h / 2, y + k2 / 2)
        y += (k1 + 4 * k2 + k3) / 6
        x += h
        ans.append(y)
    return ans
# 四阶龙格-库塔法
def rk4_method(f,x0, y0, h, x1):
    ans = []
    x = x0
    y = y0
    while x < x1:
        k1 = h * f(x, y)
        k2 = h * f(x + h / 2, y + k1 / 2)
        k3 = h * f(x + h / 2, y + k2 / 2)
        k4 = h * f(x + h, y + k3)
        y += (k1 + 2 * k2 + 2 * k3 + k4) / 6
        x += h
        ans.append(y)
    return ans
def solution_and_draw(f,x0,y0,h_values,x1,y_min,y_max):
    # 为每种方法和每种步长计算数值解
    for h in h_values:
        y_euler = np.array(euler_method(f,x0, y0, h, x1))
        y_rk3 = np.array(rk3_method(f,x0, y0, h, x1))
        y_rk4 = np.array(rk4_method(f,x0, y0, h, x1))
        # 生成 x 值的数组用于绘图
        x_values = np.arange(x0, x1, h)
        # 裁剪 y 值数组
        y_euler = y_euler[:len(x_values)]
        y_rk3 = y_rk3[:len(x_values)]
        y_rk4 = y_rk4[:len(x_values)]
        # 绘图
        plt.plot(x_values, y_euler, label='Euler h=' + str(h))
        plt.plot(x_values, y_rk3, label='RK3 h=' + str(h))
        plt.plot(x_values, y_rk4, label='RK4 h=' + str(h))
```

```python
    # 添加图例
    plt.legend()
    # 添加标题和轴标签
    plt.title('Comparison of Numerical Methods with Exact Solution')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.ylim(y_min,y_max)
    # 显示图形
    plt.show()
# 定义微分方程
def f1(x, y):
    return 5 * y - 6 * math.exp(-x)
def f2(x, y):
    return math.sin(x * y) * (x + y)
# 初始条件和步长
x0, y0 = 0, 1+1.0e-13
x1 = 6
h_values = [0.01, 0.001, 0.0001]
x_values = np.arange(x0, x1, 0.001)
y_exact = specific_solution(x_values)
plt.plot(x_values, y_exact, label='specific_solution', linestyle='--',
color='gold')
solution_and_draw(f1,x0, y0, h_values, x1,-0.1,1.1)
# # 初始条件和步长
# x0, y0 = 0, 5
# x1 = 6
# h_values = [0.2,0.1,0.01]
# solution_and_draw(f2,x0,y0,h_values,x1,-0.1,7)
```

将三种方法对方程a求得的数值解与精确解的误差以把表格形式打印如下：

```
D:\python\python.exe D:\Code\Python\R-K\pythonProject1\R-K.py
当前步长h = 0.01
  Method     Max Error    Mean Error          RMSE
0  Euler  4.298457e+09  1.504460e+08  5.755252e+08
1    RK3  1.323502e+09  4.541782e+07  1.755087e+08
2    RK4  2.435899e+04  8.324364e+02  3.223670e+03
当前步长h = 0.001
  Method     Max Error    Mean Error          RMSE
0  Euler  8.261915e+08  2.767742e+07  1.070604e+08
1    RK3  1.467724e+08  4.906699e+06  1.899958e+07
2    RK4  2.473259e+00  8.281440e-02  3.200955e-01
当前步长h = 0.0001
  Method     Max Error    Mean Error          RMSE
0  Euler  8.838656e+07  2.947692e+06  1.141493e+07
1    RK3  1.482578e+07  4.943367e+05  1.914518e+06
2    RK4  8.237399e-02  2.763103e-03  1.063718e-02
```