数值分析第 9 次上机作业

学号：221840189，姓名：王晨光

§1 问题

常微分方程初值问题：

$$\begin{cases} y' = -\dfrac{1}{x^2} - \dfrac{y}{x} - y^2, \quad 1 \le x \le 2, \\ y(1) = -1. \end{cases}$$

1.1 问题一

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.

1.1.1 算法思路

根据插值型数值微分的思想，我们可以利用精确解在当前步和前几步的值做插值多项式, 再用其导数近似初值问题的导数得到数值方法，这样构造的方法称为 BDF (Backward differential formula) 方法 (也称为 Gear 方法)。

下面我们考虑如何利用前面介绍的函数插值法或者数值积分方法设计微分方程的离散化方法. 首先对微分方程 $y' = f(t, y)$ 在区间 $[t_n, t_{n+1}]$ 上求积分得：

$$y\left(t_{n+1}\right) - y\left(t_n\right) = \int_{t_n}^{t_{n+1}} f(t, y(t)) dt$$

把 $f(t, y(t))$ 换成它的插值多项式，或对右端积分使用数值积分公式, 我们将得到解初值问题的数值方法. 例如应用矩形公式：

$$\int_{t_n}^{t_{n+1}} f(t, y(t)) dt \simeq h f\left(t_n, y\left(t_n\right)\right)$$

再用 y_n 代替 $y(t_n)$，便得到 Euler 方法：

$$y_{n+1} = y_n + h f\left(t_n, y_n\right), n = 0, 1, 2, \cdots, N-1,$$
$$y_0 = \eta.$$

仿照这样的思想，我们可以得到更多更精确的微分方程 $\begin{cases} y'(x) = f(x, y), \\ y(x_0) = y_0. \end{cases}$ 数值解的公式：

1. 改进的 Euler 公式：

$$\begin{cases} c_1 = c_2 = 1/2 \\ a_2 = 1 \end{cases} \Rightarrow \begin{cases} y_{n+1} = y_n + h/2\left[K_1 + K_2\right] \\ K_1 = f\left(x_n, y_n\right) \\ K_2 = f\left(x_n + h, y_n + h K_1\right) \end{cases}$$

2. Heun 公式：

$$\begin{cases} c_1 = 1/4, c_2 = 3/4 \\ a_2 = 2/3 \end{cases}$$
$$y_{n+1} = y_n + \frac{1}{4} h\left(K_1 + 3 K_2\right), n = 0, 1, \cdots, N-1$$

其中：

$$\begin{aligned}
&K_1 = f\left(x_n, y_n\right),\\
&K_2 = f\left(x_n + \frac{2}{3}h, y_n + \frac{2}{3}hK_1\right),\\
&y_0 = \eta,
\end{aligned}$$

3. 中点方法：

$$\begin{cases} c_1 = 0, c_2 = 1 \\ a_2 = 1/2 \end{cases}$$
$$y_{n+1} = y_n + hK_2, n = 0, 1, \cdots, N-1$$

其中：

$$\begin{aligned}
&K_1 = f\left(x_n, y_n\right)\\
&K_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{1}{2}hK_1\right)\\
&y_0 = \eta
\end{aligned}$$

4. 四阶 Runge-Kutta 方法：

$$y_{n+1} = y_n + \frac{h}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right), n = 0, 1, \cdots, N-1$$

其中：

$$\begin{aligned}
&K_1 = f\left(x_n, y_n\right),\\
&K_2 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hK_1\right),\\
&K_3 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hK_2\right),\\
&K_4 = f\left(x_n + h, y_n + hK_3\right),\\
&y_0 = \eta.
\end{aligned}$$

1.1.2 结果分析

为了更显著地比较不同方法的效果，统一规定所有方法的步长 $h = 0.1$，即将区间 $[1, 2]$ 划分为 10 个.

将使用 python 中 scipy 库中的 integrate.solve_ivp 函数求解得到的结果作为该初值问题的精确解，并与五种方法得到的结果进行比较.

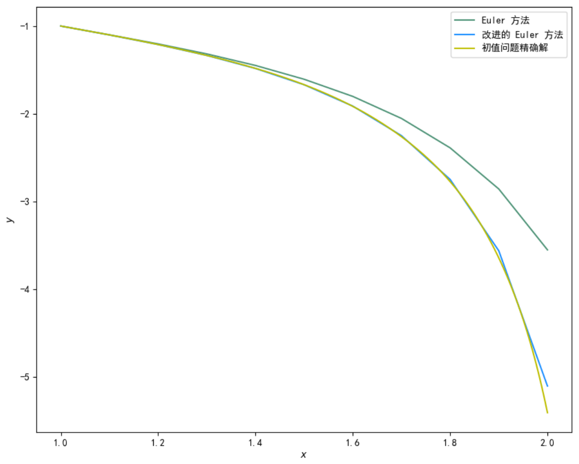

图 1: Euler 方法的改进效果

可见在 Euler 方法基础上的改进给了算法很大的提升.

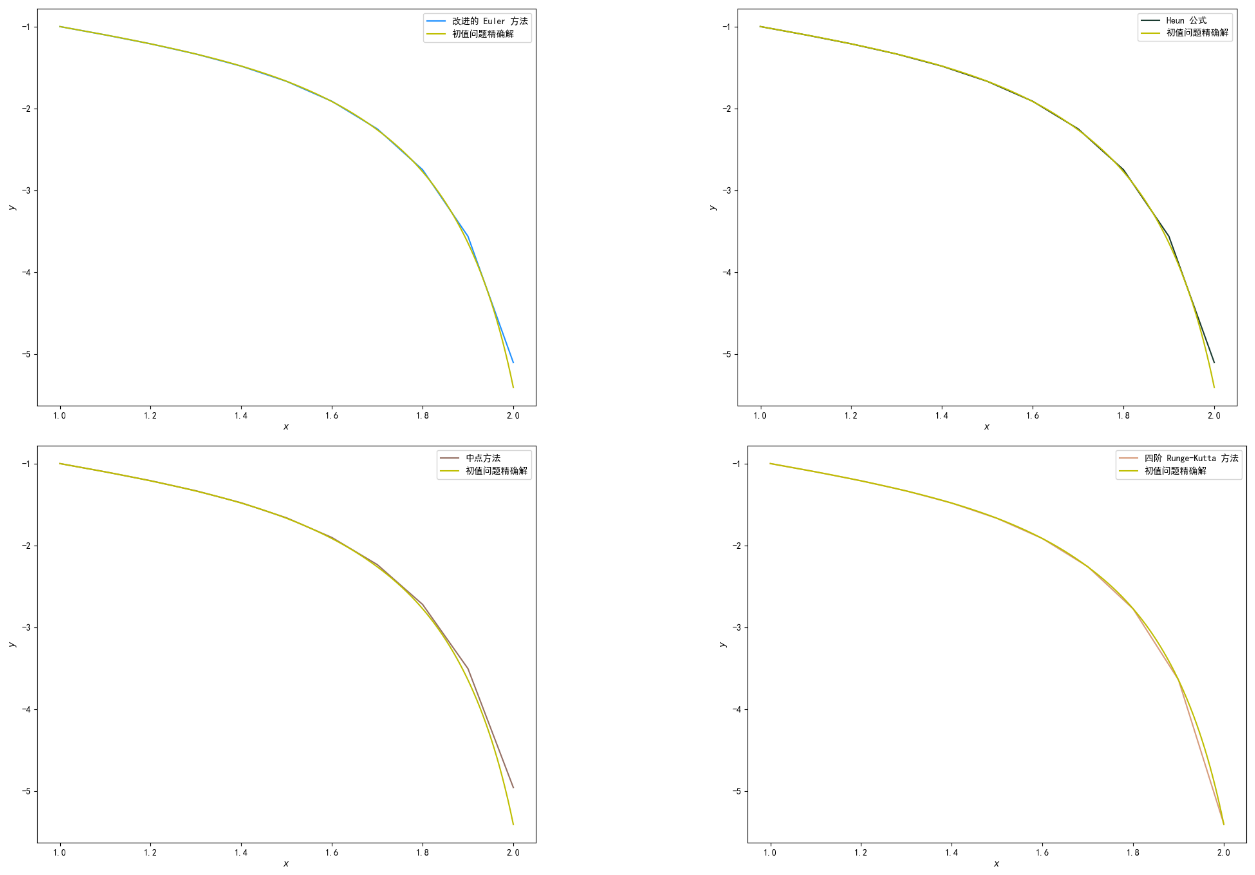

图 2: 结果比较

可见这四种方法在这个问题上面都有不错的使用效果.

x	y	Euler	Improved Euler	Heun	Midpoint	4-order Runge-Kutta
1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
1.1	-1.101282248845198	-1.1	-1.1018223140495869	-1.1018223140495869	-1.1009529478458049	-1.1012826678659193
1.2	-1.2110433325930738	-1.2036446280991737	-1.2109329616626514	-1.2109329616626514	-1.2092239073203017	-1.2100751453889105
1.3	-1.335157166350982	-1.3176613926105536	-1.3350267950205723	-1.3350267950205723	-1.3323361028616876	-1.3340877915750706
1.4	-1.482415949857855	-1.4490975761389777	-1.4834058063949158	-1.4834058063949158	-1.479385598657663	-1.482749368299053
1.5	-1.6684358216956912	-1.6065993930952163	-1.6691140645680864	-1.6691140645680864	-1.6630840854650728	-1.6694425780891586
1.6	-1.915656859375965	-1.8020533723227048	-1.9126352193216483	-1.9126352193216483	-1.9032722875944785	-1.9154952225080968
1.7	-2.26116559591342	-2.053227172222499	-2.249514782136032	-2.249514782136032	-2.234074247576292	-2.2586983818103605
1.8	-2.7733458410437857	-2.388625361468082	-2.7484745656328373	-2.7484745656328373	-2.720602215799331	-2.774579055172638
1.9	-3.642936111981181	-2.8573412617733496	-3.5620301560265935	-3.5620301560265935	-3.5044281499982297	-3.640916010383189
2.0	-5.40707572282345	-3.551095619222935	-5.103210160551191	-5.103210160551191	-4.955788578459564	-5.400991565128523

表 1: 五种方法在插值基点上的取值与精确值的比较

可以看出除了四阶 Runge-Kutta 方法，其他方法的误差都有增大的趋势，这个现象在 $x = 2$ 处可以直观地看出.

1.2 问题二

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 (1) 中的四阶 Runge-Kutta 方法求解进行比较.

1.2.1 算法思路

预测公式：

$$y_{n+1}^{(0)} = y_n + \frac{n}{24}\left[55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}\right]$$

校正公式：

$$y_{n+1} = y_n + \frac{h}{24}\left[9f_{n+1}^{(0)} + 19f_n - 5f_{n-1} + f_{n-2}\right]$$

其 PECE 模式为：

$$\begin{aligned}
P:&\quad y_{n+1}^{(0)} = y_n + \frac{h}{24}\left[55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}\right]\\
E:&\quad f_{n+1}^{(0)} = f\left(t_{n+1}, y_{n+1}^{(0)}\right)\\
C:&\quad y_{n+1} = y_n + \frac{h}{24}\left(9f^{(0)}{}_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}\right)\\
E:&\quad f_{n+1} = f\left(t_{n+1}, y_{n+1}\right)
\end{aligned}$$

其中用 P 表示预测过程. C 表示校正过程, E 表示计算 f 的过程.

1.2.2 结果分析

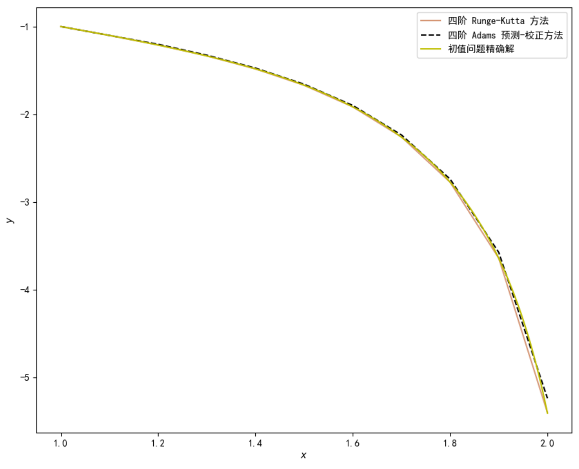

图 3: 结果比较

可见如果仅仅使用极少数的 Runge-Kutta 方法提供的初始点, 四阶 Adams 预测-校正方法的 PECE 模式就会有较不错的表现, 但最终的效果不如完全使用四阶 Runge-Kutta 方法. 尤其是在 $x = 2$ 处, 这在下表中也可以明显地看出:

x	y	4-order Runge-Kutta	4-order Adams
1.0	-1.0	-1.0	-1.0
1.1	-1.101282248845198	-1.1012826678659193	-1.1012826678659193
1.2	-1.2110433325930738	-1.2100751453889105	-1.2018351932142624
1.3	-1.335157166350982	-1.3340877915750706	-1.3261530760597708
1.4	-1.482415949857855	-1.482749368299053	-1.4732204768780879
1.5	-1.6684358216956912	-1.6694425780891586	-1.6572302727892014
1.6	-1.915656859375965	-1.9154952225080968	-1.8991814924893602
1.7	-2.26116559591342	-2.2586983818103605	-2.2355054775965906
1.8	-2.7733458410437857	-2.774579055172638	-2.7384415918600347
1.9	-3.642936111981181	-3.640916010383189	-3.5752795942449733
2.0	-5.40707572282345	-5.400991565128523	-5.2377466864277435

表 2: 两种方法在插值基点上的取值与精确值的比较

§2　结论

如果只从计算结果精确度来看, 四种解微分方程数值解的方法中, 无改进的 Euler 方法效果最差, 四阶 Runge-Kutta 方法和与之结合的四阶 Adams 预测-校正方法的 PECE 模式效果最好, 改进的 Euler 方法、Heun 公式、中点方法也都有着不错的效果.

§ 3 附录: 程序代码

```python
import numpy as np
import matplotlib.pyplot as plt
# 定义常微分方程
def f(x, y):
    return -1/x**2 - y/x - y**2

# 定义各数值方法
def euler_method(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)
    x[0] = x0
    y[0] = y0
    for i in range(n):
        y[i+1] = y[i] + h * f(x[i], y[i])
        x[i+1] = x[i] + h
    return x, y

def improved_euler_method(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)
    x[0] = x0
    y[0] = y0
    for i in range(n):
        x[i+1] = x[i] + h
    for i in range(n):
        y_star = y[i] + h * f(x[i], y[i])
        y[i+1] = y[i] + h/2 * (f(x[i], y[i]) + f(x[i+1], y_star))
    return x, y

def heun_method(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)
    x[0] = x0
    y[0] = y0
    for i in range(n):
        k1 = f(x[i], y[i])
        k2 = f(x[i] + h, y[i] + h * k1)
        y[i+1] = y[i] + h/2 * (k1 + k2)
        x[i+1] = x[i] + h
    return x, y

def midpoint_method(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)
    x[0] = x0
    y[0] = y0
    for i in range(n):
        k1 = f(x[i], y[i])
        k2 = f(x[i] + h/2, y[i] + h/2 * k1)
        y[i+1] = y[i] + h * k2
        x[i+1] = x[i] + h
    return x, y

def runge_kutta_4(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)
    x[0] = x0
    y[0] = y0
    for i in range(n):
        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/6 * (k1 + 2*k2 + 2*k3 + k4)
        x[i+1] = x[i] + h
    return x, y

# 四阶 Adams 预测-校正方法
def adams_predictor_corrector(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)
    x_copy, y_copy = runge_kutta_4(f, x0, y0, h, 1)
    for i in range(2):
        x[i] = x_copy[i]
        y[i] = y_copy[i]
    for i in range(1, n):
        x[i+1] = x[i] + h
    for i in range(1, n):
        y_predict = y[i] + h/24 * (55*f(x[i], y[i]) - 59*f(x[i-1], y[i-1]) + 37*f(
        y_correct = y[i] + h/24 * (9*f(x[i+1], y_predict) + 19*f(x[i], y[i]) - 5*f
        y[i+1] = y_correct
    return x, y
# 初始条件和步长
x0, y0 = 1, -1
h = 0.1
n = int((2 - x0) / h)
# 求解
x_euler, y_euler = euler_method(f, x0, y0, h, n)
```

```
x_improved_euler, y_improved_euler = improved_euler_method(f, x0, y0, h, n)
x_heun, y_heun = heun_method(f, x0, y0, h, n)
x_midpoint, y_midpoint = midpoint_method(f, x0, y0, h, n)
x_rk4, y_rk4 = runge_kutta_4(f, x0, y0, h, n)
x_adams, y_adams = adams_predictor_corrector(f, x0, y0, h, n)
# 列表输出计算结果
# print("x values:", x_rk4)
# print("Euler Method:", y_euler)
# print("Improved Euler Method:", y_improved_euler)
# print("Heun Method:", y_heun)
# print("Midpoint Method:", y_midpoint)
print("Runge-Kutta␣4␣Method:", y_rk4)
print("Adams␣Predictor-Corrector␣Method:", y_adams)
from scipy.integrate import solve_ivp
x_span=(1, 2)
y_0=np.array([-1])
sol=solve_ivp(f, x_span, y_0, method='RK45', dense_output=True)
x_values = np.linspace(x_span[0], x_span[1], 1000)
y_values = sol.sol(x_values)
print("Solution:␣", sol.sol(x_rk4)[0])
print("Runge-Kutta␣4␣Method:", y_rk4)
print("Adams␣Predictor-Corrector␣Method:", y_adams)
for i in range(11):
    # print(x_rk4[i])
    # print(y_euler[i])
    # print(y_improved_euler[i])
    # print(y_heun[i])
    # print(y_midpoint[i])

    # print(sol.sol(x_rk4)[0][i])
    # print(y_rk4[i])
    print(y_adams[i])
# 画图
plt.rcParams['font.sans-serif'] = ['SimHei']  # 指定默认字体
plt.rcParams['axes.unicode_minus'] = False
plt.figure(figsize=(10, 8))
# plt.plot(x_euler, y_euler, label='Euler 方法', color='#509579')
# plt.plot(x_improved_euler, y_improved_euler, label='改进的 Euler 方法', color='d
# plt.plot(x_heun, y_heun, label='Heun 公式', color='#1a3b30')
# plt.plot(x_midpoint, y_midpoint, label='中点方法', color='#957064')
plt.plot(x_rk4, y_rk4, label='四阶␣Runge-Kutta␣方法', color='#d89c7c')
plt.plot(x_adams, y_adams, label='四阶␣Adams␣预测-校正方法', color='black', linesty
plt.plot(x_values, y_values[0], label='初值问题精确解', color='y')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend()
plt.show()
```

§ 4　附录: 题目二程序代码