中国科学技术大学

实验报告

计算方法 B

Project 3

学生姓名: 朱云沁

学生学号: PB20061372

完成时间: 二〇二二年四月十六日

目录

一、 实验题目

程序 1 用 Newton 迭代法求解非线性方程组

$$\begin{cases} f(x,y) = x^2 + y^2 - 1 = 0 \\ g(x,y) = x^3 - y = 0 \end{cases} \tag{1}$$

取 $\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 0.8 \\ 0.6 \end{pmatrix}$, 误差控制 $\max\{|\Delta x|, |\Delta y|\} \leq 10^{-5}$.
输入: 初始点 (x_0, y_0), 精度控制值 ε, 定义函数 $f(x,y), g(x,y)$.
输出: 迭代次数 k, 第 k 步的迭代解 (x_k, y_k).

程序 2 用二阶 Runge-Kutta 公式求解常微分方程初值问题

$$\begin{cases} y'(x) = f(x,y), \quad a \leq x \leq y \\ y(a) = y_0 \end{cases} \tag{2}$$

求解以下问题

$$\begin{cases} y'(x) = ysin(\pi x) \\ y(0) = 1 \end{cases} \tag{3}$$

输入: 区间剖分点数 n，区间端点 a, b，定义函数 $y'(x) = f(x,y)$.
输出: $y_k, k = 1, 2, \cdots, n$.

程序 3 用改进的 Euler 公式求解常微分方程组初值问题. 已知计算公式为

$$\begin{gathered} \begin{pmatrix} \bar{y}_{n+1} \\ \bar{z}_{n+1} \end{pmatrix} = \begin{pmatrix} y_n \\ z_n \end{pmatrix} + h \begin{pmatrix} f(x_n, y_n, z_n) \\ g(x_n, y_n, z_n) \end{pmatrix} \\ \begin{pmatrix} y_{n+1} \\ z_{n+1} \end{pmatrix} = \begin{pmatrix} y_n \\ z_n \end{pmatrix} + \frac{h}{2} \left[\begin{pmatrix} f(x_n, y_n, z_n) \\ g(x_n, y_n, z_n) \end{pmatrix} + \begin{pmatrix} f(x_{n+1}, \bar{y}_{n+1}, \bar{z}_{n+1}) \\ g(x_{n+1}, \bar{y}_{n+1}, \bar{z}_{n+1}) \end{pmatrix} \right] \end{gathered} \tag{4}$$

求解以下问题

$$\begin{cases} \dfrac{\mathrm{d}u}{\mathrm{d}t} = 0.09u\left(1 - \dfrac{u}{20}\right) - 0.45uv \\ \dfrac{\mathrm{d}v}{\mathrm{d}t} = 0.06v\left(1 - \dfrac{v}{15}\right) - 0.001uv \\ u(0) = 1.6 \\ v(0) = 1.2 \end{cases} \tag{5}$$

输入: 区间剖分点数 N，区间端点 a, b，定义函数 $y'(x) = f(x,y,z), z'(x) = g(x,y,z)$.
输出: $(y_k, z_k), k = 1, 2, \cdots, N$.

二、 实验结果

1. 数值结果

表 1: Newton 迭代法求解 (1) 式非线性方程组

k	x_k	y_k	$f(x_k, y_k)$	$g(x_k, y_k)$	Δx_k	Δy_k
0	0.8000000	0.6000000	0.0000	-0.0880		
1	0.8270492	0.5639344	0.0020	0.0018	0.0270	-0.0361
2	0.8260324	0.5636237	1.1305×10^{-6}	2.5642×10^{-6}	-0.0010	-0.0003
3	0.8260314	0.5636242	1.2670×10^{-12}	2.5555×10^{-12}	-1.0155×10^{-6}	4.8546×10^{-7}

表 2: 二阶 Runge-Kutta 公式 (见 (23) 式) 求解 (3) 式常微分方程初值问题

x	$y\|_{n=10}$	$y\|_{n=20}$	$y\|_{n=40}$	$y\|_{n=80}$	$y\|_{n=160}$
0.0	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
0.7	1.5221240	1.6273693	1.6507791	1.6560464	1.6572724
1.4	1.1872162	1.4595916	1.5068327	1.5149108	1.5164731
2.1	0.8486241	0.9876154	1.0120792	1.0152182	1.0156330
2.8	1.3931523	1.7034568	1.7662329	1.7763725	1.7781563
3.5	0.8438227	1.2880932	1.3614096	1.3725043	1.3743589
4.2	0.6714567	1.0033967	1.0546988	1.0615566	1.0625064
4.9	1.1295513	1.7384684	1.8426574	1.8579982	1.8603636
5.6	0.5707655	1.1410565	1.2308038	1.2436556	1.2456030
6.3	0.5104204	1.0448954	1.1269264	1.1382755	1.1398998
7.0	0.8296881	1.7196324	1.8655813	1.8864674	1.8894995

表 3: 改进的 Euler 公式求解 (5) 式常微分方程组初值问题

x	$u\|_{N=5}$	$u\|_{N=10}$	$u\|_{N=200}$	$v\|_{N=5}$	$v\|_{N=10}$	$v\|_{N=200}$
1.0	1.6000000	1.6000000	1.6000000	1.2000000	1.2000000	1.2000000
2.0	1.0245663	1.0047660	0.9994871	1.2663436	1.2663196	1.2663150
3.0	0.6409123	0.6138593	0.6068011	1.3366014	1.3365852	1.3365849
4.0	0.3912111	0.3640354	0.3571228	1.4107733	1.4107839	1.4107932
5.0	0.2328620	0.2091345	0.2032773	1.4888936	1.4889415	1.4889633
6.0	0.1351379	0.1161834	0.1116680	1.5710237	1.5711143	1.5711500

2. 图示结果

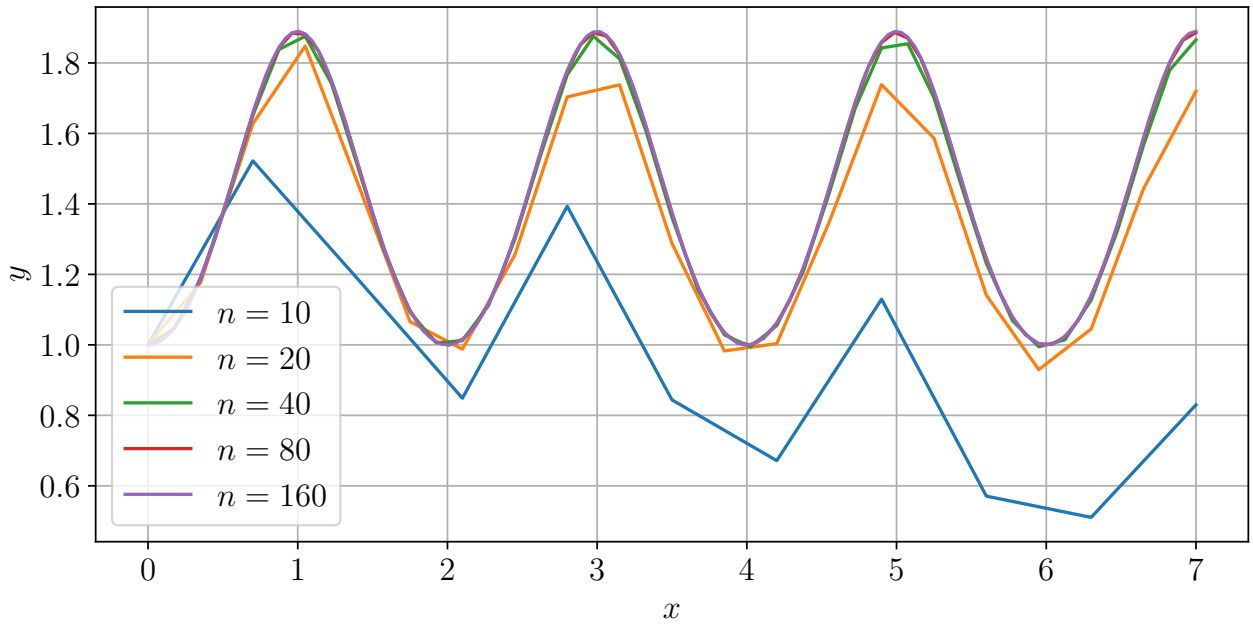

图 1: 二阶 Runge-Kutta 公式求解 (3) 式常微分方程初值问题 ($a = 0, b = 7$)

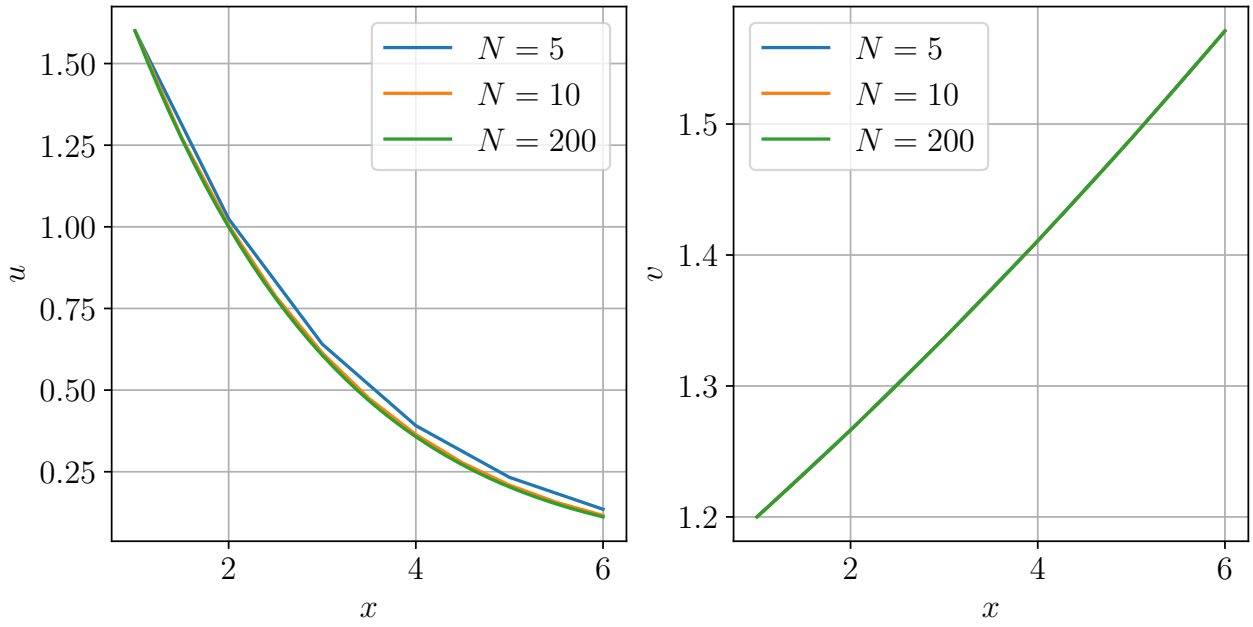

图 2: 改进的 Euler 公式求解 (5) 式常微分方程组初值问题 ($a = 1, b = 6$)

三、 算法分析

1. Newton 迭代法

对于一般的非线性方程组

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(x_1, x_2, \cdots, x_n) \\ f_2(x_1, x_2, \cdots, x_n) \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) \end{pmatrix} = \mathbf{0} \tag{6}$$

将 $f_i(\mathbf{x}), i = 1, 2, \cdots, n$ 在 $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \cdots, x_n^{(0)})^{\mathrm{T}}$ 处作 n 元 Taylor 展开, 得

$$f_i(\mathbf{x}) = f_i(\mathbf{x}^{(0)}) + \nabla f_i(\mathbf{x}^{(0)})^{\mathrm{T}}(\mathbf{x} - \mathbf{x}^{(0)}) + \frac{1}{2}\mathbf{x}^{\mathrm{T}}\nabla^2 f_i(\mathbf{x}^{(0)})\mathbf{x} + \cdots \tag{7}$$

作一阶近似

$$0 \approx f_i(\mathbf{x}^{(0)}) + \nabla f_i(\mathbf{x}^{(0)})^{\mathrm{T}}(\mathbf{x} - \mathbf{x}^{(0)}) \tag{8}$$

进而

$$\mathbf{0} \approx \mathbf{f}(\mathbf{x}^{(0)}) + \mathbf{J}_{\mathbf{f}}(\mathbf{x}^{(0)})(\mathbf{x} - \mathbf{x}^{(0)}) \tag{9}$$

其中 $\mathbf{J}_{\mathbf{f}}(\mathbf{x}^{(0)}) = (\nabla f_1(\mathbf{x}^{(0)}), \nabla f_2(\mathbf{x}^{(0)}), \cdots, \nabla f_n(\mathbf{x}^{(0)}))^{\mathrm{T}}$ 为向量值函数 $\mathbf{f}(\mathbf{x})$ 的 Jacobi 矩阵. 用 $\mathbf{x}^k, \mathbf{x}^{k+1}$ 替换 $\mathbf{x}^0, \mathbf{x}$, 得 Newton 迭代公式

$$\mathbf{x}^{k+1} = \mathbf{f}(\mathbf{x}^k) - \mathbf{J}_{\mathbf{f}}^{-1}(\mathbf{x}^k)\mathbf{f}(\mathbf{x}^k) \tag{10}$$

可以证明, 当 $||\mathbf{J}_{\mathbf{f}}(\mathbf{x})||_\infty < 1$ 且初始值充分接近 $\mathbf{x}$ 时, Newton 迭代收敛. 通常以如下形式计算

$$\begin{aligned} \mathbf{J}_{\mathbf{f}}(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k+1)} &= -\mathbf{f}(\mathbf{x}^{(k)}) \\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k+1)} \end{aligned} \tag{11}$$

当 $||\Delta\mathbf{x}^{(k)}||_\infty < \varepsilon$ 时, 满足精度, 结束迭代.

对于 (1) 式问题, 求得 $\mathbf{J}_{\mathbf{f}} = \begin{pmatrix} 2x & 2y \\ 3x^2 & -1 \end{pmatrix}$, 从而由 (11) 式有

$$\begin{pmatrix} \Delta x_{k+1} \\ \Delta y_{k+1} \end{pmatrix} = \frac{1}{-6x_k^2 y_k - 2x_k} \begin{pmatrix} 2x_k^3 y_k + x_k^2 - y_k^2 - 1 \\ x_k^4 + 3x_k^2 y_k^2 - 3x_k^2 + 2x_k y_k \end{pmatrix} \tag{12}$$

对形如 (1) 式的一般二阶非线性方程组, 给出算法如下:

算法 1 Newton 迭代法求解非线性方程组

输入: 初始点 (x_0, y_0), 精度控制值 ε, 定义函数 $f(x,y), g(x,y)$.

输出: 迭代次数 k, 第 k 步的迭代解 (x_k, y_k).

```
1 for k ← 1 to M do                                   // M 为最大迭代次数.
2     计算 J_f(x, y);
3     求解 J_f(x, y)(Δx, Δy)^T = −(f(x, y), g(x, y))^T;
4     if max{|Δx|, |Δy|} < ε then
5         return k, (x, y);
6     end
7     (x, y) ← (x + Δx, y + Δy)
8 end
9 return NULL;                                        // x_0 附近无解
```

对于上述算法第 2 到 3 行, 若无法求得形如 (12) 式的公式解, 应当采用数值微分方法 (弦截法).

2. 显式 Runge-Kutta 方法

对于形如 (2) 式的初值问题, s 阶显式 Runge-Kutta 方法可写为

$$\begin{cases} y_{n+1} = y_n + h\sum_{i=1}^{s} c_i k_i \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + a_2 h, y_n + b_{21} h k_1) \\ k_3 = f(x_n + a_3 h, y_n + b_{31} h k_1 + b_{32} h k_2) \\ \quad\vdots \\ k_s = f(x_n + a_s h, y_n + b_{s1} h k_1 + b_{s2} h k_2 + \cdots + b_{s,s-1} h k_{s-1}) \end{cases} \tag{13}$$

为了构造二阶公式, 首先将 $y(x)$ 在 $x+h$ 处作 Taylor 展开

$$y(x+h) = y(x) + hy'(z) + \frac{h^2}{2} y''(x) + O(h^3) \tag{14}$$

由微分方程, 有

$$\begin{aligned} y'(x_n) &= f(x_n, y(x_n)) \\ y''(x_n) &= f_x(x_n, y(x_n)) + f_y(x_n, y(x_n)) f(x_n, y(x_n)) \end{aligned} \tag{15}$$

带入 (14) 式并截断余项, 得

$$y(x_{n+1}) \approx y(x_n) + hf(x_n, y(x_n)) + \frac{h^2}{2}\left(f_x(x_n, y(x_n)) + f_y(x_n, y(x_n)) f(x_n, y(x_n))\right) \tag{16}$$

于是有二阶近似公式

$$\tilde{y}_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2}\left(f_x(x_n, y_n) + f_y(x_n, y_n) f(x_n, y_n)\right) \tag{17}$$

另一方面, 由 (13) 式定义的近似公式为

$$\begin{aligned} y_{n+1} &= y_n + hc_1 f(x_n, y_n) + hc_2 f(x_n + a_2 h, y_n + b_{21} h f(x_n, y_n)) \\ &= y_n + h(c_1 + c_2) f(x_n, y_n) + h^2 c_2 a_2 f_x(x_n, y_n) + h^2 c_2 b_{21} f_y(x_n, y_n) f(x_n, y_n) + O(h^3) \end{aligned} \tag{18}$$

为了使 (18) 式有局部截断误差 $O(h^3)$, 应有 $y(x_{n+1}) - \tilde{y}(x_{n+1}) = O(h^3)$, 于是

$$\begin{cases} c_1 + c_2 = 1 \\ c_2 a_2 = \dfrac{1}{2} \\ c_2 b_{12} = \dfrac{1}{2} \end{cases} \tag{19}$$

令 $c_1 = c_2 = \frac{1}{2}$, 得到常见二阶 Runge-Kutta 公式

$$\begin{cases} y_{n+1} = y_n + h\left(\dfrac{1}{2}k_1 + \dfrac{1}{2}k_2\right) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + h, y_n + hk_1) \end{cases} \tag{20}$$

或写为预估-矫正的形式

$$\begin{cases} \bar{y}_{n+1} = y_n + hf(x_n, y_n) \\ y_{n+1} = y_n + h\left(\dfrac{1}{2}f(x_n, y_n) + \dfrac{1}{2}f(x_{n+1}, \bar{y}_{n+1})\right) \end{cases} \tag{21}$$

称作改进的 Euler 公式.

另外两种常见的二阶 Runge-Kutta 公式是

$$\begin{cases} y_{n+1} = y_n + hk_2 \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \dfrac{1}{2}h, y_n + \dfrac{1}{2}k_1) \end{cases} \tag{22}$$

称作 (显式) 中点公式.

$$\begin{cases} y_{n+1} = y_n + h\left(\dfrac{1}{4}k_1 + \dfrac{3}{4}k_2\right) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \dfrac{2}{3}h, y_n + \dfrac{2}{3}k_1) \end{cases} \tag{23}$$

具有最小的局部截断误差限.

对于一般的 s 阶显式 Runge-Kutta 公式, 给出如下算法:

算法 2 s 阶显式 Runge-Kutta 公式求解常微分方程初值问题

输入: 区间剖分点数 N，区间端点 a, b，定义函数 $y'(x) = f(x, y)$.

输出: $y_k, k = 1, 2, \cdots, n$.

$h \leftarrow \frac{b-a}{n}, x_0 \leftarrow a$;
for $k \leftarrow 0$ **to** $n-1$ **do**
  $x_{k+1} \leftarrow x_k + h$;
  for $i \leftarrow 1$ **to** s **do**
    $k_i \leftarrow f(x_k + a_i h, y_k + h\sum_{j=1}^{i-1} b_{ij} k_j)$;
  end
  $y_{n+1} \leftarrow y_n + h\sum_{i=1}^{s} c_i k_i$;
end
return $y_k, k = 1, 2, \cdots, n$;

将 (21) 式应用到向量值函数 $\mathbf{y}(x) = (y, z)^{\mathrm{T}}$, 立即得到求解二阶常微分方程组初值问题的 (4) 式. 算法如下:

算法 3 改进的 Euler 公式求解常微分方程组初值问题

输入: 区间剖分点数 N，区间端点 a, b，定义函数 $y'(x) = f(x, y)$.

输出: $(y_k, z_k), k = 1, 2, \cdots, N$.

$h \leftarrow \frac{b-a}{N}, x_0 \leftarrow a$;
for $k \leftarrow 0$ **to** $N-1$ **do**
  $x_{k+1} \leftarrow x_k + h$;
  $y_{k+1} \leftarrow y_k + hf(x_k, y_k, z_k)$;
  $z_{k+1} \leftarrow z_k + hg(x_k, y_k, z_k)$;
  $\begin{pmatrix} y_{k+1} \\ z_{k+1} \end{pmatrix} \leftarrow \begin{pmatrix} y_k + \frac{h}{2}(f(x_k, y_k, z_k) + f(x_{k+1}, y_{k+1}, z_{k+1})) \\ z_k + \frac{h}{2}(g(x_k, y_k, z_k) + g(x_{k+1}, y_{k+1}, z_{k+1})) \end{pmatrix}$;
end
return $(y_k, z_k), k = 1, 2, \cdots, N$;

四、 结果分析

- 由表 1 知, 对 (1) 式作 3 次 Newton 迭代得到的解 (x_3, y_3) 满足 $\max\{|\Delta x_3|, |\Delta y_3|\} \leq 10^{-5}$, 符合精度要求. 根据原始数据, 程序 1 求得问题的解为

$$x = 0.8260313576552345$$
$$y = 0.5636241621608473$$

而 $(0.8, 0.6)$ 附近实际的解为

$$x = \sqrt{\sqrt[3]{\frac{9+\sqrt{93}}{18}} - \sqrt[3]{\frac{2}{3(9+\sqrt{93})}}} \approx 0.826031357654187$$

$$y = x^3 \approx 0.563624162161259$$

按 ∞-范数定义的相对误差约为 1.27×10^{-12}, 可见使用 Newton 迭代法求解该问题的精度极高, 与此同时需要的迭代次数较少.

- 由表 2 和图 1 知, 用 (23) 式定义的二阶显式 Runge-Kutta 公式计算 (3) 式的常微分方程初值问题, 所得到解的截断误差受区间剖分点数影响较大. 随步长减小, 截断误差收敛, 得到的解接近真实解 $y = e^{\frac{1-\cos \pi x}{\pi}}$. 根据原始数据, 当 $n = 160, h = 0.04375$ 时, Runge-Kutta 公式计算得区间右端点的解为

  $$y = 1.8894994705275012$$

  $x = 7$ 时真实解为

  $$y = e^{\frac{2}{\pi}} \approx 1.89008116457222198$$

  相对误差约为 -0.03%; 然而, 当 $n = 10, h = 0.7$ 时, 每次单步计算均造成较大误差, 在区间右端点绝对误差达到 -1.06039310047, 严重偏离真实解. 上述一系列现象映证了二阶 Runge-Kutta 方法具有 $O(h^3)$ 局部截断误差和 $O(h^2)$ 总体截断误差.

- 由表 3 和图 2 知, 用改进的 Euler 公式, 即 (21) 式的二阶 Runge-Kutta 方法计算 (5) 式的常微分方程组初值问题, 随步长减小, 截断误差收敛于 0, 求得的解有较好的预测效果. 映证了二阶 Runge-Kutta 公式的截断误差理论、Euler 方法与 Runge-Kutta 方法的统一性, 以及将以上方法应用于向量值函数的可行性.

附录 A　Python 程序代码

1.　用 Newton 迭代法求解非线性方程组

```python
import numpy as np
import numpy.linalg as la
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd

mpl.rc('font', family='serif', size=15)
mpl.rc('text', usetex=True)

out_dir = '../assets/output/'


def newton(f, Jf, x0, eps=1e-5, max_iter=100, print_flag=False):
    x, fx = x0.copy(), f(x0)
    if (print_flag):
        df = pd.DataFrame(columns=[f'x{i}' for i in range(x.size)] +
                          [f'f{i}(x)' for i in range(x.size)] + [f'Δx{i}' for i in
                          ↪  range(x.size)])
        df.loc[0] = x0[:, 0].tolist() + fx[:, 0].tolist() + [np.nan] * x.size
    for k in range(max_iter):
        Jx = Jf(x)
        dx = la.solve(Jx, -fx)
        x += dx
        fx = f(x)
        if (print_flag):
            df.loc[k + 1] = x[:, 0].tolist() + fx[:, 0].tolist() + dx[:, 0].tolist()
        if la.norm(dx, np.inf) < eps:
            return x, df
    raise RuntimeError('Failed to converge.')


def f(X):
    x, y = X[0, 0], X[1, 0]
    return np.array([[x**2 + y**2 - 1], [x**3 - y]])


def Jf(X):
    x, y = X[0, 0], X[1, 0]
    return np.array([[2 * x, 2 * y], [3 * x**2, -1]])


x0 = np.array([[0.8], [0.6]])

x, df = newton(f, Jf, x0, print_flag=True)
```

```
44  df.to_csv(out_dir + 'newton.csv')
```

2. 用二阶 Runge-Kutta 公式求解常微分方程初值问题

```
1   import numpy as np
2   import matplotlib as mpl
3   import matplotlib.pyplot as plt
4   import pandas as pd
5
6   mpl.rc('font', family='serif', size=15)
7   mpl.rc('text', usetex=True)
8
9   out_dir = '../assets/output/'
10
11  # Butcher tableau for modified Euler method, midpoint method, and Ralston method
12  MODIFIED_EULER = 2, [0.0, 1.0], [[0.0, 0.0], [1.0, 0.0]], [0.5, 0.5]
13  MIDPOINT = 2, [0.0, 0.5], [[0.0, 0.0], [0.5, 0.0]], [0.0, 1.0]
14  RALSTON = 2, [0.0, 2.0 / 3.0], [[0.0, 0.0], [2.0 / 3.0, 0.0]], [0.25, 0.75]
15
16
17  def runge_kutta(f, y0, begin, end, n, butcher=RALSTON, print_flag=False):
18      s, a, b, c = butcher
19      x = begin
20      y = y0.copy()
21      h = (end - begin) / n
22      if (print_flag):
23          df = pd.DataFrame(columns=['x'] + [f'y{i}' for i in range(y.size)])
24          df.loc[0] = [x] + y0.tolist()
25      k = np.zeros((s, y.size))
26      for step in range(1, n + 1):
27          for i in range(s):
28              tmp = np.zeros(y.shape)
29              for j in range(i):
30                  tmp += b[i][j] * k[i - 1]
31              k[i] = f(x + a[i] * h, y + tmp * h)
32          x += h
33          for i in range(s):
34              y += h * c[i] * k[i]
35          if (print_flag): df.loc[step] = [x] + y.tolist()
36      return y, df
37
38
39  def f(x, Y):
40      y = Y[0]
41      return np.array([y * np.sin(np.pi * x)])
42
```

```

y0 = np.array([1.0])
begin, end, n_list = 0.0, 7, [10, 20, 40, 80, 160]

df = pd.DataFrame({'x': np.linspace(begin, end, n_list[0] + 1)})

fig, ax = plt.subplots(figsize=(8, 4), constrained_layout=True)
for n in n_list:
    df_ = runge_kutta(f, y0, begin, end, n, print_flag=True)[1].rename(columns={'y0':
    ↪  f'n={n}'})
    df = pd.merge_asof(df, df_, on='x', direction='nearest')
    df_.plot(ax=ax, x='x', y=f'n={n}', label=f'$n={n}$')
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')
ax.grid()
ax.legend()
fig.savefig('../assets/output/runge-kutta.pdf')
df.to_csv(out_dir + 'runge-kutta.csv')
```

3. 用改进的 Euler 公式求解常微分方程组初值问题

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd

mpl.rc('font', family='serif', size=15)
mpl.rc('text', usetex=True)

out_dir = '../assets/output/'

# Butcher tableau for modified Euler method, midpoint method, and Ralston method
MODIFIED_EULER = 2, [0.0, 1.0], [[0.0, 0.0], [1.0, 0.0]], [0.5, 0.5]
MIDPOINT = 2, [0.0, 0.5], [[0.0, 0.0], [0.5, 0.0]], [0.0, 1.0]
RALSTON = 2, [0.0, 2.0 / 3.0], [[0.0, 0.0], [2.0 / 3.0, 0.0]], [0.25, 0.75]


def runge_kutta(f, y0, begin, end, n, butcher=RALSTON, print_flag=False):
    s, a, b, c = butcher
    x = begin
    y = y0.copy()
    h = (end - begin) / n
    if (print_flag):
        df = pd.DataFrame(columns=['x'] + [f'y{i}' for i in range(y.size)])
        df.loc[0] = [x] + y0.tolist()
    k = np.zeros((s, y.size))
```

```
    for step in range(1, n + 1):
        for i in range(s):
            tmp = np.zeros(y.shape)
            for j in range(i):
                tmp += b[i][j] * k[i - 1]
            k[i] = f(x + a[i] * h, y + tmp * h)
        x += h
        for i in range(s):
            y += h * c[i] * k[i]
        if (print_flag): df.loc[step] = [x] + y.tolist()
    return y, df


def f(x, Y):
    u, v = Y[0], Y[1]
    return np.array(
        [0.09 * u * (1 - u / 20) - 0.45 * u * v, 0.06 * v * (1 - v / 15) - 0.001 * u * v])


u0, v0 = 1.6, 1.2
y0 = np.array([u0, v0])
begin, end, n_list = 1.0, 6.0, [5, 10, 200]

udf = pd.DataFrame({'x': np.linspace(begin, end, n_list[0] + 1)})
vdf = pd.DataFrame({'x': np.linspace(begin, end, n_list[0] + 1)})

fig, ax = plt.subplots(1, 2, figsize=(8, 4), constrained_layout=True)
for n in n_list:
    df_ = runge_kutta(f, y0, begin, end, n, butcher=MODIFIED_EULER, print_flag=True)[1]
    udf = pd.merge_asof(udf,
                        df_.get(['x', 'y0']).rename(columns={'y0': f'n={n}'}),
                        on='x',
                        direction='nearest')
    vdf = pd.merge_asof(vdf,
                        df_.get(['x', 'y1']).rename(columns={'y1': f'n={n}'}),
                        on='x',
                        direction='nearest')
    df_.plot(ax=ax[0], x='x', y=f'y0', label=f'$n={n}$')
    df_.plot(ax=ax[1], x='x', y=f'y1', label=f'$n={n}$')
ax[0].set_xlabel(r'$x$')
ax[0].set_ylabel(r'$u$')
ax[0].grid()
ax[0].legend()
ax[1].set_xlabel(r'$x$')
ax[1].set_ylabel(r'$v$')
ax[1].grid()
ax[1].legend()
fig.savefig('../assets/output/euler.pdf')
```

```
74  udf.to_csv(out_dir + 'euler1.csv')
75  vdf.to_csv(out_dir + 'euler2.csv')
```