Computing Method - Chapter10-11

Zhixin Zhang, 3210106357

Problems: Chapter10-10, Chapter11-1

10-10

用 Newton 法求方程组

$$\begin{cases} u = x^2 + y^2 - 1 = 0 \\ v = x^3 - y = 0 \end{cases}$$

在 $x_0 = 0.8, y_0 = 0.6$ 附近的根，作 3 次迭代.

解: 计算 Jacobi 矩阵：

$$J = \begin{bmatrix} 2x & 2y \\ 3x^2 & -1 \end{bmatrix}$$

求解代码为：

```
import numpy as np
def J(x, y):
    return [[2*x, 2*y], [3*x**2, -1]]
def f1(x, y):
    return x**2+y**2-1
def f2(x, y):
    return x**3-y
def Newton(x0, y0):
    for i in range(3):
        F1 = -f1(x0, y0)
        F2 = -f2(x0, y0)
        J0 = J(x0, y0)
        [dx, dy] = np.linalg.solve(np.array(J0), np.array([F1, F2]))
        x0 = x0 + dx
        y0 = y0 + dy
        print(i+1, x0, y0)
Newton(0.8, 0.6)
```

输出结果为：

```
1 0.8270491803278689 0.5639344262295083
2 0.8260323731676462 0.5636236767037873
3 0.8260313576552345 0.5636241621608473
```

所以三次迭代后 $x_3 = 0.8260, y_3 = 0.5636$.

□

11-1

在区间 $[0, 1]$ 上数值解

$$\begin{cases} y' = y - \frac{2x}{y} \\ y(0) = 1 \end{cases}$$

(1) 用 Euler 方法和改进 Euler 方法，取 $h = 0.1$；
(2) 用经典 R-K 方法，取 $h = 0.2$.

精确解 $y = \sqrt{1 + 2x}$ 可供比较.

解:

(1) 经典 Euler 方法：$y(x_0) = y_0, y_{n+1} = y_n + hf(x_n, y_n)$
改进 Euler 方法：$y_{n+1} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_{n+1}, y_n + hf(x_n, y_n)))$

```python
import numpy as np
import pandas as pd

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

def euler_method(f, x0, y0, h, n):
    x = [x0]
    y = [y0]
    for i in range(n):
        y_next = y[-1] + h * f(x[-1], y[-1])
        x_next = x[-1] + h
        x.append(x_next)
        y.append(y_next)
    return x, y

def improved_euler_method(f, x0, y0, h, n):
    x = [x0]
    y = [y0]
    for i in range(n):
        y_temp = y[-1] + h * f(x[-1], y[-1])
        x_next = x[-1] + h
        y_next = y[-1] + h / 2 * (f(x[-1], y[-1]) + f(x_next, y_temp))
        x.append(x_next)
        y.append(y_next)
    return x, y

def theoretical_solution(x):
    return np.sqrt(1 + 2 * x)

x0 = 0
y0 = 1
h = 0.1
n = 10

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)
y_theoretical = theoretical_solution(np.array(x_euler))

error_euler = np.abs(y_theoretical - np.array(y_euler))
error_improved_euler = np.abs(y_theoretical - np.array(y_improved_euler))
```

```
data = {
    '$x$': x_euler,
    'Euler $y_1$': y_euler,
    'Improved Euler $y_2$': y_improved_euler,
    'Theoretical $y^*$': y_theoretical,
    'Euler Error $abs(y_1-y^*)$': error_euler,
    'Improved Euler Error $abs(y_2-y^*)$': error_improved_euler
}

df = pd.DataFrame(data)
markdown_table = df.to_markdown(index=False)
print(markdown_table)
```

x	Euler y_1	Improved Euler y_2	Theoretical y^*	Euler Error $\|y_1 - y^*\|$	Improved Euler Error $\|y_2 - y^*\|$
0.1	1.1	1.09591	1.09545	0.00455488	0.000463976
0.2	1.19182	1.1841	1.18322	0.00860223	0.000880613
0.3	1.27744	1.2662	1.26491	0.0125268	0.0012903
0.4	1.35821	1.34336	1.34164	0.0165718	0.00171936
0.5	1.43513	1.4164	1.41421	0.0209194	0.00218837
0.6	1.50897	1.48596	1.48324	0.0257266	0.0027159
0.7	1.58034	1.55251	1.54919	0.0311449	0.00332075
0.8	1.64978	1.61647	1.61245	0.0373319	0.00402323
0.9	1.71778	1.67817	1.67332	0.0444593	0.00484631
1	1.78477	1.73787	1.73205	0.05272	0.00581659

(2) 经典 RK 方法：

$$
\begin{cases}
y_{n+1} = y_n + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\
K_1 = hf(x_n, y_n) \\
K_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{K_1}{2}\right) \\
K_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{K_2}{2}\right) \\
K_4 = hf(x_n + h, y_n + K_3)
\end{cases}
$$

```
def f(x, y):
    return y - (2 * x) / y
def rk4_method(f, x0, y0, h, n):
    x = [x0]
    y = [y0]
    for i in range(n):
        K1 = h * f(x[-1], y[-1])
        K2 = h * f(x[-1] + h / 2, y[-1] + K1 / 2)
        K3 = h * f(x[-1] + h / 2, y[-1] + K2 / 2)
        K4 = h * f(x[-1] + h, y[-1] + K3)
        y_next = y[-1] + 1 / 6 * (K1 + 2 * K2 + 2 * K3 + K4)
        x_next = x[-1] + h
        x.append(x_next)
```

```
        y.append(y_next)
    return x, y
def theoretical_solution(x):
    return np.sqrt(1 + 2 * x)
x0 = 0
y0 = 1
h = 0.2
n = 5
x_rk4, y_rk4 = rk4_method(f, x0, y0, h, n)
y_theoretical = theoretical_solution(np.array(x_rk4))
error_rk4 = np.abs(y_theoretical - np.array(y_rk4))
data = {
    'x': x_rk4,
    'R-K $y$': y_rk4,
    'Theoretical $y^*$': y_theoretical,
    'R-K Error $abs(y-y^*)$': error_rk4
}
df = pd.DataFrame(data)
markdown_table = df.to_markdown(index=False)
print(markdown_table)
```

x	R-K y	Theoretical y^*	R-K Error $\|y-y^*\|$
0	1	1	0
0.2	1.18323	1.18322	1.33308e-05
0.4	1.34167	1.34164	2.61434e-05
0.6	1.48328	1.48324	4.17609e-05
0.8	1.61251	1.61245	6.2492e-05
1	1.73214	1.73205	9.10751e-05

□