非线性方程求根 实验报告

> 计73 郑林楷 2017011474

我使用 python3 + numpy + scipy + matplotlib 完成以下若干实验。

2-2

代码可见于 `lab2/2-2.py`。

运行结果

第一个函数：$x^3 - x - 1 = 0$，取 $x_0 = 0.6$

```
damping Newton Raphson:
Step 1: x = 1.140625, f(x) = -0.656642913818358
Step 2: x = 1.3668136615928, f(x) = 0.186639718200229
Step 3: x = 1.32627980400832, f(x) = 0.00667040146992925
Step 4: x = 1.32472022563606, f(x) = 9.67387688377563e-6
Step 5: x = 1.32471795724954, f(x) = 2.04494199351757e-11
Step 6: x = 1.32471795724475, f(x) = 2.22044604925031e-16
Step 7: x = 1.32471795724475, f(x) = 2.22044604925031e-16

basic Newton Raphson:
Step 1: x = 17.9, f(x) = 5716.43900000002
Step 2: x = 11.9468023286088, f(x) = 1692.17353280208
Step 3: x = 7.98552035193622, f(x) = 500.239416029005
Step 4: x = 5.35690931479547, f(x) = 147.367517808272
Step 5: x = 3.6249960329461, f(x) = 43.0096132035221
Step 6: x = 2.50558919010663, f(x) = 12.2244425918391
Step 7: x = 1.82012942231947, f(x) = 3.20972476461149
Step 8: x = 1.46104410988768, f(x) = 0.657773540085028
Step 9: x = 1.33932322426253, f(x) = 0.0631369611469705
Step 10: x = 1.32491286771866, f(x) = 0.000831372624715332
Step 11: x = 1.32471799263781, f(x) = 1.50938453735705e-7
Step 12: x = 1.32471795724475, f(x) = 4.88498130835069e-15
Step 13: x = 1.32471795724475, f(x) = 2.22044604925031e-16

Basic Newton Raphson: 1.32471795724475
Newton Raphson with damp: 1.32471795724475
SciPy: 1.3247179572446863
Newton error: 0.00000000%
Newton with damp error: 0.00000000%
```

第二个函数：$-x^3 + 5x = 0$，取 $x_0 = 1.35$

```
damping Newton Raphson:
Step 1: x = 2.49695855614973, f(x) = -3.08324949677949
Step 2: x = 2.27197620552696, f(x) = -0.367778144156867
Step 3: x = 2.23690170575035, f(x) = -0.00834194597647553
Step 4: x = 2.23606844338361, f(x) = -4.65883963940428e-6
Step 5: x = 2.23606797749994, f(x) = -1.45838896514761e-12
Step 6: x = 2.23606797749979, f(x) = -1.77635683940025e-15
```

```
Step 7: x = 2.23606797749979, f(x) = -1.77635683940025e-15

basic Newton Raphson:
Step 1: x = 10.5256684491978, f(x) = -1113.5072686208
Step 2: x = 7.12428662558879, f(x) = -325.975011180964
Step 3: x = 4.91078065301939, f(x) = -93.8733368952927
Step 4: x = 3.51691130589217, f(x) = -25.9149417173825
Step 5: x = 2.70974300619979, f(x) = -6.34813434143222
Step 6: x = 2.33694003146878, f(x) = -1.07800405405571
Step 7: x = 2.24224425399285, f(x) = -0.0620188943055542
Step 8: x = 2.23609340302195, f(x) = -0.000254259558142067
Step 9: x = 2.23606797793343, f(x) = -4.33645297448493e-9
Step 10: x = 2.23606797749979, f(x) = -1.77635683940025e-15

Basic Newton Raphson: 2.23606797749979
Newton Raphson with damp: 2.23606797749979
SciPy: 2.236067977499786
Newton error: 0.00000000%
Newton with damp error: 0.00000000%
```

实现过程

$\lambda_0 = 1$，$\epsilon = 10^{-8}$。

阻尼牛顿法

迭代判停准则：$\lambda \leq 0$，即找不到合适的步长。

```python
def damp_Newton_Raphson(f, x):
    step = 0
    while True:
        tmp = f(x) / diff(f, x)
        lambd = 1
        while lambd > 0 and np.abs(f(x - lambd * tmp)) >= np.abs(f(x)):
            lambd /= 2
        x -= lambd * tmp
        step += 1
        print('Step {}: x = {}, f(x) = {}'.format(step, x, f(x)))
        if lambd <= 0:
            break
    return x
```

基本牛顿法：

```python
def basc_Newton_Raphson(f, x):
    step = 0
    last_x = x
    while np.abs(f(x)) > eps or np.abs(x - last_x) > eps:
        last_x = x
        x -= f(x) / diff(f, x)
        step += 1
        print('Step {}: x = {}, f(x) = {}'.format(step, x, f(x)))
    return x
```

结果分析和实验总结

可以看到阻尼牛顿法所需要的迭代次数明显少于基本牛顿法，而且两者的误差都非常小。

通过实验我们可以很直观的感受到阻尼牛顿法的好处，使用阻尼牛顿法能够很好地解决步长过长的问题，收敛速度更加快。

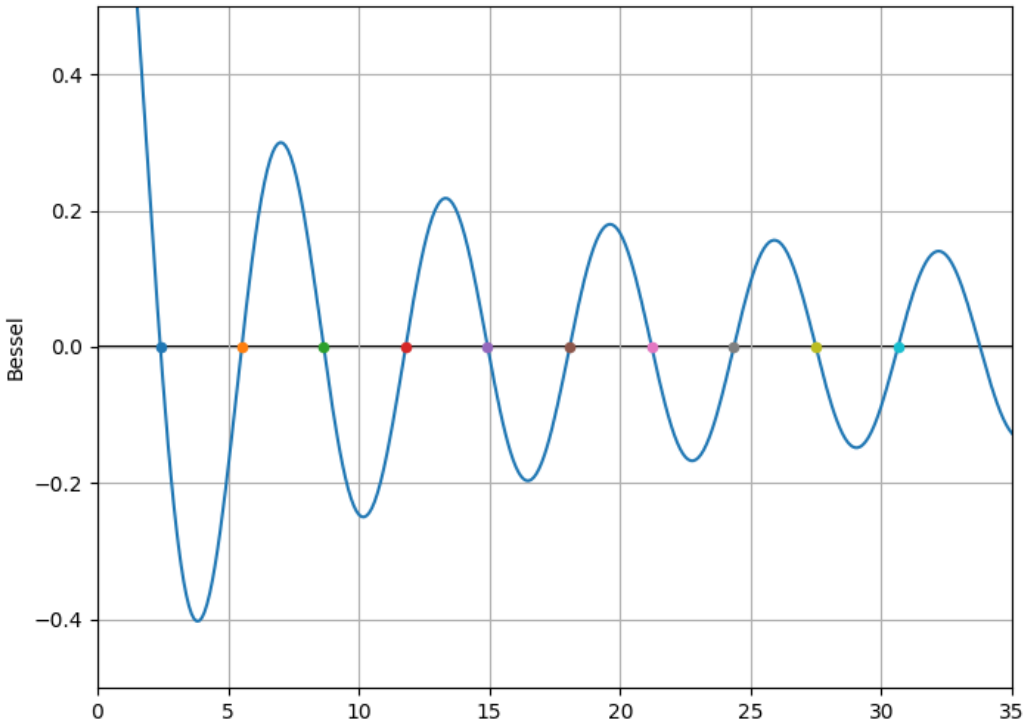
2-3

代码可见于 `lab2/2-3.py`。

运行结果

$J_0(x)$ 前 10 个正的零点：

```
zeroin method took 8 steps to solve the equation: 2.40482556
zeroin method took 6 steps to solve the equation: 5.52007806
zeroin method took 5 steps to solve the equation: 8.65372792
zeroin method took 5 steps to solve the equation: 11.79153444
zeroin method took 6 steps to solve the equation: 14.93091771
zeroin method took 6 steps to solve the equation: 18.07106397
zeroin method took 5 steps to solve the equation: 21.21163663
zeroin method took 5 steps to solve the equation: 24.35247153
zeroin method took 7 steps to solve the equation: 27.49347916
zeroin method took 5 steps to solve the equation: 30.63460649
```



实现过程

首先按照 2.6.3 节实现 zeroin 算法，接着用其求解第一类零阶贝塞尔曲线函数 $J_0(x)$ 前 10 个正的零点，并将零点绘制在函数曲线图上。

求零点可以先通过绘制函数图象，目测出零点所在的区间，然后硬编码到代码中。

结果分析和实验总结

通过这次实验我学会了 Zeroin 零点迭代法。这种算法不需要导数，也可以快速地求得函数上的零点，因而是一种通用且高效的求零点算法。