第二章

第二题

将书上的伪代码翻译成Python代码即可

- 设定初始阻尼因子和误差阈值：
  程序运行后由用户输入，如果用户不输入，那么用默认值0.2和0.001

  ```
  lamd0 = input('lambda0:')
  lamd0 = np.float64(lamd0) if lamd0 != '' else 0.2
  eps = input('eps:')
  eps = np.float64(eps) if eps != '' else 0.001
  ```

- 逐次折半法更新阻尼因子：

  ```
  i = 0
  lamdk = lamd
  while abs(f(xk)) > abs(f(xk_pre)):
      xk = xk_pre - lamdk * s
      lamdk /= 2.0
      i += 1
  ```

- 打印每一个迭代步骤:
  damp为是否使用阻尼因子的开关

  ```
  if damp:
      print ("\tlamd%d: %.10f\tx%d: %.10f" % (k, lamdk, k, xk))
  else:
      print ("\tx%d: %.10f" % (k, xk))
  ```

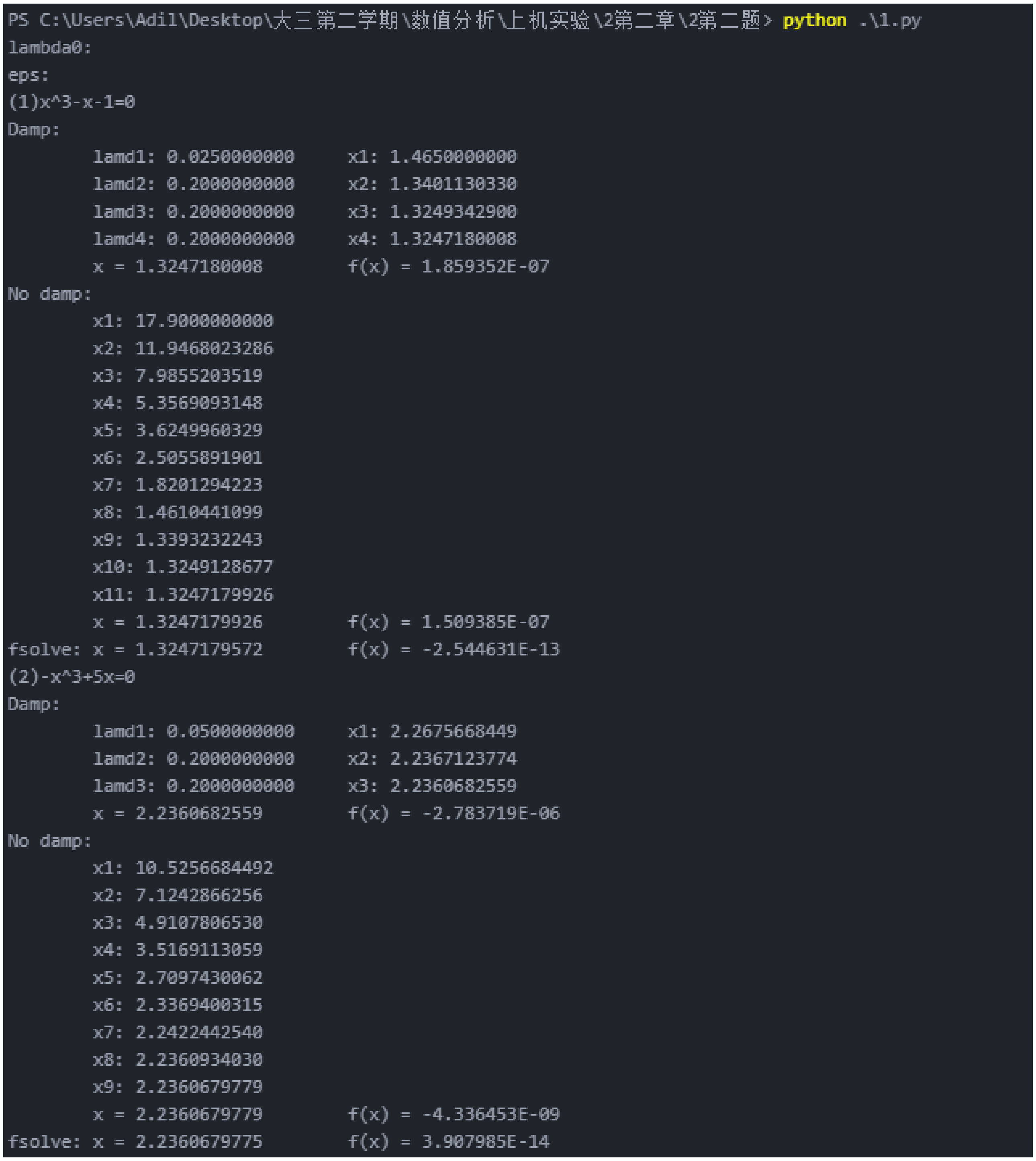
- 其他方法验证结果：
  采用Python scipy.optimize库中的fsolve()函数求解方程，得到结果

  ```
  result = fsolve(f1, x0)
  print ("fsolve:\tx = %.10f\tf(x) = %E" % (result, f1(result)))
  ```

- 比较采用阻尼和不采用阻尼算法的效果差别:
  利用damp开关控制是否采用阻尼

```python
...
def newton(f, f_, x0, eps1, eps2, lamd, damp=True):
    print ('Damp:') if damp else print ('No damp:')
    k = 0
    xk = x0
    xk_pre = x0
    while abs(f(xk)) > eps1 or abs(xk - xk_pre) > eps2:
        s = f(xk) / f_(xk)
        xk_pre = xk
        xk = xk_pre - s
        if damp:
            i = 0
            lamdk = lamd
            while abs(f(xk)) > abs(f(xk_pre)):
                xk = xk_pre - lamdk * s
                lamdk /= 2.0
                i += 1
        k += 1
...

k, xk = newton(f1, f1_, np.array(x0, dtype=np.float64), eps, eps, lamd0)
k, xk = newton(f1, f1_, np.array(x0, dtype=np.float64), eps, eps, lamd0, damp=False)
```

效果:

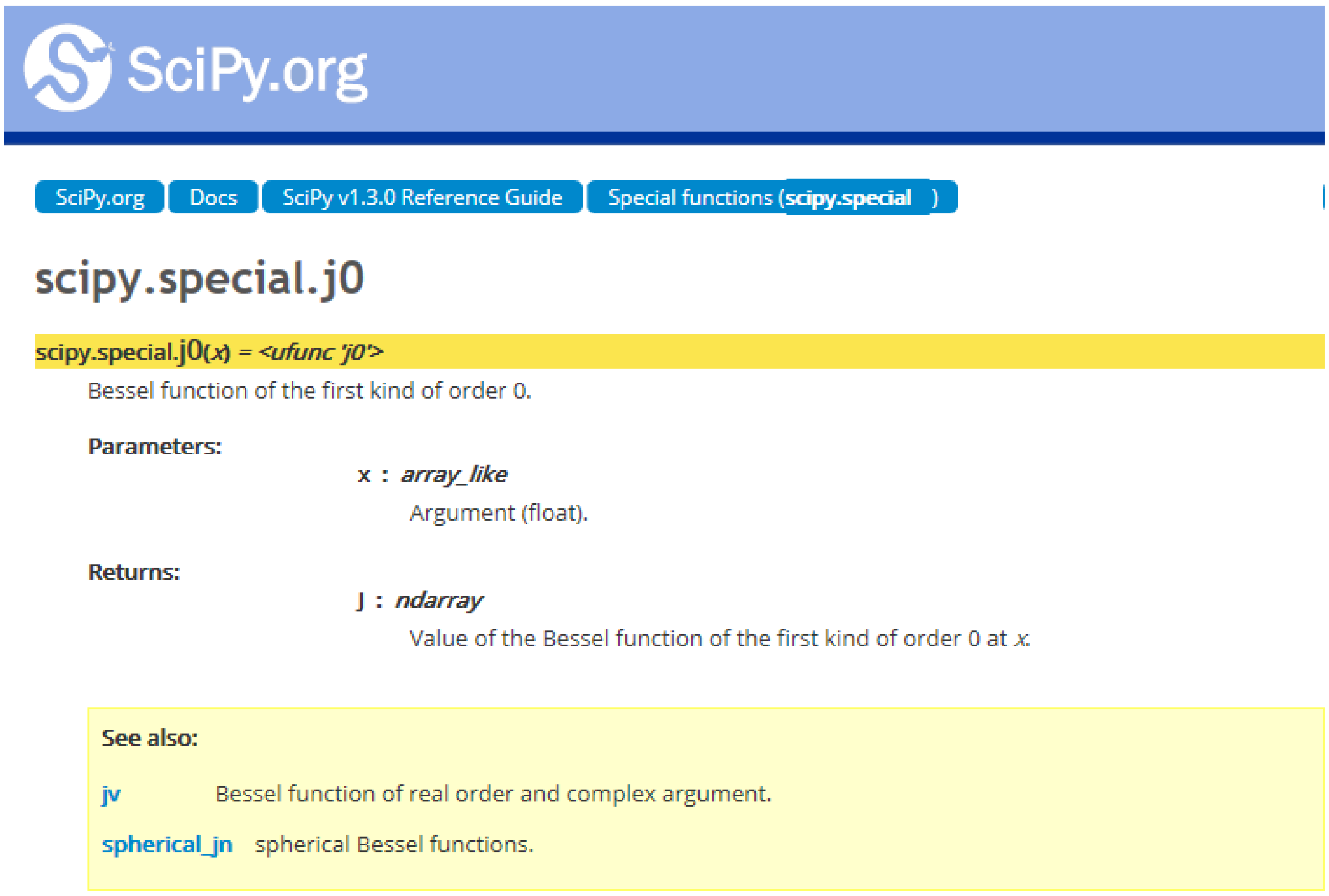


```
PS C:\Users\Adil\Desktop\大三第二学期\数值分析\上机实验\2第二章\2第二题> python .\1.py
lambda0:
eps:
(1)x^3-x-1=0
Damp:
        lamd1: 0.0250000000     x1: 1.4650000000
        lamd2: 0.2000000000     x2: 1.3401130330
        lamd3: 0.2000000000     x3: 1.3249342900
        lamd4: 0.2000000000     x4: 1.3247180008
        x = 1.3247180008        f(x) = 1.859352E-07
No damp:
        x1: 17.9000000000
        x2: 11.9468023286
        x3: 7.9855203519
        x4: 5.3569093148
        x5: 3.6249960329
        x6: 2.5055891901
        x7: 1.8201294223
        x8: 1.4610441099
        x9: 1.3393232243
        x10: 1.3249128677
        x11: 1.3247179926
        x = 1.3247179926        f(x) = 1.509385E-07
fsolve: x = 1.3247179572        f(x) = -2.544631E-13
(2)-x^3+5x=0
Damp:
        lamd1: 0.0500000000     x1: 2.2675668449
        lamd2: 0.2000000000     x2: 2.2367123774
        lamd3: 0.2000000000     x3: 2.2360682559
        x = 2.2360682559        f(x) = -2.783719E-06
No damp:
        x1: 10.5256684492
        x2: 7.1242866256
        x3: 4.9107806530
        x4: 3.5169113059
        x5: 2.7097430062
        x6: 2.3369400315
        x7: 2.2422442540
        x8: 2.2360934030
        x9: 2.2360679779
        x = 2.2360679779        f(x) = -4.336453E-09
fsolve: x = 2.2360679775        f(x) = 3.907985E-14
```

可以看出，采用阻尼的收敛速度明显高于不采用阻尼的收敛速度。

第三题

- 将书上的fzerotx()函数代码翻译成Python代码即可
  要注意的是设置eps的值：
  ```
  eps = 1.0E-10
  ```
- 注意到Python scipy.special库中提供了第一类零阶贝塞尔函数：


SciPy.org
SciPy.org
Docs
SciPy v1.3.0 Reference Guide
Special functions (scipy.special)

scipy.special.j0

scipy.special.j0(*x*) = <ufunc 'j0'>

Bessel function of the first kind of order 0.

Parameters:

x : *array_like*

Argument (float).

Returns:

J : *ndarray*

Value of the Bessel function of the first kind of order 0 at *x*.

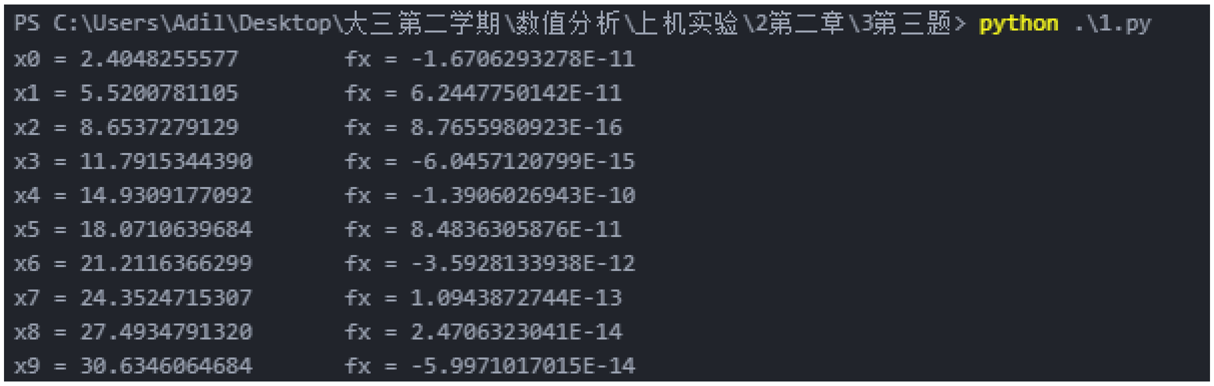
See also:

jv	Bessel function of real order and complex argument.
spherical_jn	spherical Bessel functions.

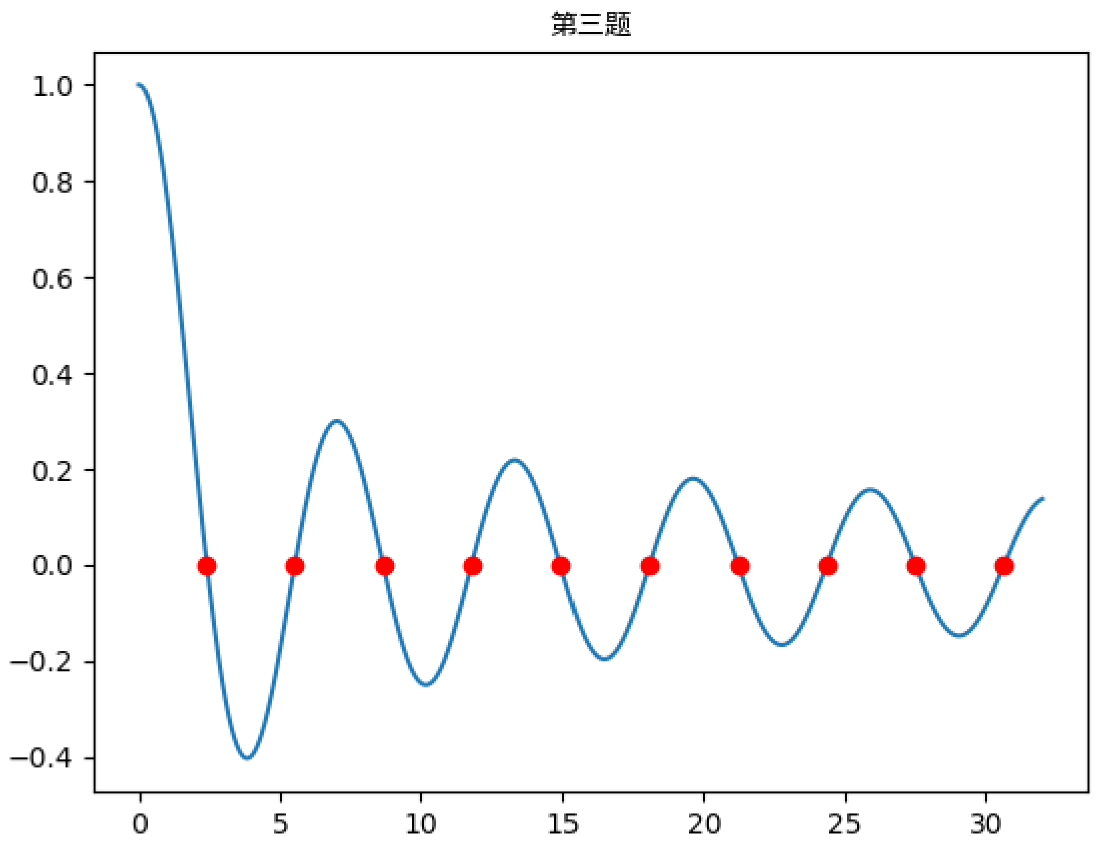
- 求出J0(x)的前10个正的零点：

```
list = [0]
ansX = [0] * 10
ansY = [0] * 10
f = j0(0)
num = 0
for i in range(1, 50):
    if np.sign(j0(i)) != np.sign(f):            #符号相反说明有零点
        list.append(i)
        f = j0(i)
        ansX[num], ansY[num] = fzerotx(j0, (list[num], i))
        print ("x%d = %.10f\t fx = %.10E" % (num, ansX[num], ansY[num]))
        num += 1
        if num >= 10:
            break

plt.figure(num="第三题")

x = np.arange(0, 32, 0.01)                      #画出J0(x)的图像
y = j0(x)
plt.plot(x, y)
plt.plot(ansX, ansY, 'ro')                      #标出零点的位置
plt.show()
```

- 效果：



```
PS C:\Users\Adil\Desktop\大三第二学期\数值分析\上机实验\2第二章\3第三题> python .\1.py
x0 = 2.4048255577         fx = -1.6706293278E-11
x1 = 5.5200781105         fx = 6.2447750142E-11
x2 = 8.6537279129         fx = 8.7655980923E-16
x3 = 11.7915344390        fx = -6.0457120799E-15
x4 = 14.9309177092        fx = -1.3906026943E-10
x5 = 18.0710639684        fx = 8.4836305876E-11
x6 = 21.2116366299        fx = -3.5928133938E-12
x7 = 24.3524715307        fx = 1.0943872744E-13
x8 = 27.4934791320        fx = 2.4706323041E-14
x9 = 30.6346064684        fx = -5.9971017015E-14
```


第三题