非线性方程求根-上机作业

张晨，2017011307

2.2

【实验内容】
对比牛顿法，阻尼牛顿法，matlab fzero函数求根效果。
【实验结果】
三种算法均可求解题目中的两个方程。
【结果分析】
本实验中设$\lambda = 0.9$。
第一个方程，牛顿迭代法使用了13次迭代，阻尼牛顿法仅使用6次迭代，阻尼牛顿法取得了较好的效果。
第二个方程，牛顿法与阻尼牛顿法都使用了10次迭代，且迭代过程完全一致，阻尼牛顿法未起到优化作用。
因此，阻尼牛顿法在某些情况下能取得较好结果，但不保证优于牛顿法。
【关键代码】

```
function ret = damping_newton_method(f, f_, x, use_damping)
    % f-the function; f_-derivative of f; x0-start point
    e1 = 1e-8; % hyper-parameter
    e2 = 1e-8; % hyper-parameter
    lambda_initial = 0.9; % hyper-parameter
    k = 0;
    pre = x+1;
    while (abs(f(x))>e1 || abs(x-pre)>e2)
        pre = x;
        s = f(x)/f_(x);
        x = pre - s;
        i = 0;
        lambda = lambda_initial;
        if (use_damping)
            fprintf("    iter 0 : lambda = %f x = %.10f f(x) = ...
                %.10f\n", lambda, x, f(x))
            while (abs(f(x) ≥ abs(f(pre))))
                x = pre - lambda * s;
                i = i + 1;
                lambda = lambda * 0.5;
                fprintf("    iter %d : lambda = %f x = %.10f ...
                    f(x) = %.10f\n", i, lambda, x, f(x))
            end
        end
        k = k+1;
        fprintf("  Step %d : x = %.10f f(x) = %.10f\n", k, x, f(x))
    end
```

```
26      ret = x;
27  end
```

【程序输出】

```
1   Solving x^3-x-1=0
2   not use damping
3       Step 1 : x = 17.9000000000 f(x) = 5716.4390000000
4       Step 2 : x = 11.9468023286 f(x) = 1692.1735328021
5       Step 3 : x = 7.9855203519 f(x) = 500.2394160290
6       Step 4 : x = 5.3569093148 f(x) = 147.3675178083
7       Step 5 : x = 3.6249960329 f(x) = 43.0096132035
8       Step 6 : x = 2.5055891901 f(x) = 12.2244425918
9       Step 7 : x = 1.8201294223 f(x) = 3.2097247646
10      Step 8 : x = 1.4610441099 f(x) = 0.6577735401
11      Step 9 : x = 1.3393232243 f(x) = 0.0631369611
12      Step 10 : x = 1.3249128677 f(x) = 0.0008313726
13      Step 11 : x = 1.3247179926 f(x) = 0.0000001509
14      Step 12 : x = 1.3247179572 f(x) = 0.0000000000
15      Step 13 : x = 1.3247179572 f(x) = 0.0000000000
16  root using ordinary newton method 1.324718
17  use damping
18      --iter 0 : lambda = 0.900000 x = 17.9000000000 f(x) = ...
            5716.4390000000
19      --iter 1 : lambda = 0.450000 x = 16.1700000000 f(x) = ...
            4210.7821130000
20      --iter 2 : lambda = 0.225000 x = 8.3850000000 f(x) = ...
            580.1494666250
21      --iter 3 : lambda = 0.112500 x = 4.4925000000 f(x) = ...
            85.1776339531
22      --iter 4 : lambda = 0.056250 x = 2.5462500000 f(x) = ...
            12.9620794004
23      --iter 5 : lambda = 0.028125 x = 1.5731250000 f(x) = ...
            1.3199224641
24      Step 1 : x = 1.5731250000 f(x) = 1.3199224641
25      --iter 0 : lambda = 0.900000 x = 1.3676629524 f(x) = ...
            0.1905532689
26      Step 2 : x = 1.3676629524 f(x) = 0.1905532689
27      --iter 0 : lambda = 0.900000 x = 1.3263416845 f(x) = ...
            0.0069350829
28      Step 3 : x = 1.3263416845 f(x) = 0.0069350829
29      --iter 0 : lambda = 0.900000 x = 1.3247204087 f(x) = ...
            0.0000104547
30      Step 4 : x = 1.3247204087 f(x) = 0.0000104547
31      --iter 0 : lambda = 0.900000 x = 1.3247179573 f(x) = ...
            0.0000000000
32      Step 5 : x = 1.3247179573 f(x) = 0.0000000000
33      --iter 0 : lambda = 0.900000 x = 1.3247179572 f(x) = ...
            0.0000000000
34      Step 6 : x = 1.3247179572 f(x) = 0.0000000000
35  root using damping newton method 1.324718
36  Answer using fzero: 1.324718
37  ----------------------------------------------
38  Solving -x^3+5*x=0
39  not use damping
40      Step 1 : x = 10.5256684492 f(x) = -1113.5072686208
41      Step 2 : x = 7.1242866256 f(x) = -325.9750111810
```

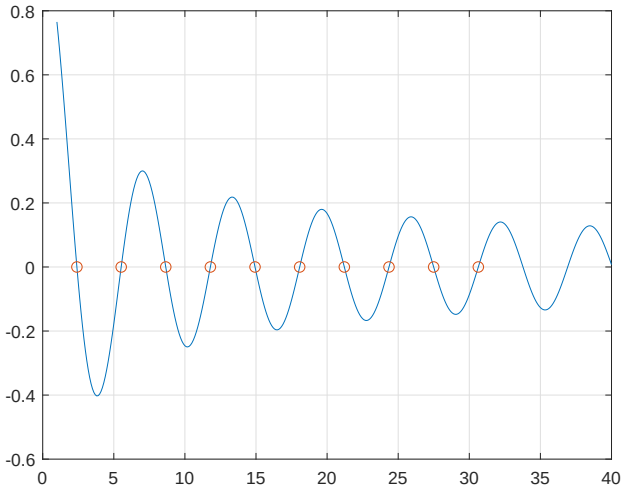
```
    Step 3 : x = 4.9107806530 f(x) = -93.8733368953
    Step 4 : x = 3.5169113059 f(x) = -25.9149417174
    Step 5 : x = 2.7097430062 f(x) = -6.3481343414
    Step 6 : x = 2.3369400315 f(x) = -1.0780040541
    Step 7 : x = 2.2422442540 f(x) = -0.0620188943
    Step 8 : x = 2.2360934030 f(x) = -0.0002542596
    Step 9 : x = 2.2360679779 f(x) = -0.0000000043
    Step 10 : x = 2.2360679775 f(x) = 0.0000000000
root using ordinary newton method 2.236068
use damping
    --iter 0 : lambda = 0.900000 x = 10.5256684492 f(x) = ...
        -1113.5072686208
    Step 1 : x = 10.5256684492 f(x) = -1113.5072686208
    --iter 0 : lambda = 0.900000 x = 7.1242866256 f(x) = ...
        -325.9750111810
    Step 2 : x = 7.1242866256 f(x) = -325.9750111810
    --iter 0 : lambda = 0.900000 x = 4.9107806530 f(x) = ...
        -93.8733368953
    Step 3 : x = 4.9107806530 f(x) = -93.8733368953
    --iter 0 : lambda = 0.900000 x = 3.5169113059 f(x) = ...
        -25.9149417174
    Step 4 : x = 3.5169113059 f(x) = -25.9149417174
    --iter 0 : lambda = 0.900000 x = 2.7097430062 f(x) = ...
        -6.3481343414
    Step 5 : x = 2.7097430062 f(x) = -6.3481343414
    --iter 0 : lambda = 0.900000 x = 2.3369400315 f(x) = ...
        -1.0780040541
    Step 6 : x = 2.3369400315 f(x) = -1.0780040541
    --iter 0 : lambda = 0.900000 x = 2.2422442540 f(x) = ...
        -0.0620188943
    Step 7 : x = 2.2422442540 f(x) = -0.0620188943
    --iter 0 : lambda = 0.900000 x = 2.2360934030 f(x) = ...
        -0.0002542596
    Step 8 : x = 2.2360934030 f(x) = -0.0002542596
    --iter 0 : lambda = 0.900000 x = 2.2360679779 f(x) = ...
        -0.0000000043
    Step 9 : x = 2.2360679779 f(x) = -0.0000000043
    --iter 0 : lambda = 0.900000 x = 2.2360679775 f(x) = ...
        0.0000000000
    Step 10 : x = 2.2360679775 f(x) = 0.0000000000
root using damping newton method 2.236068
Answer using fzero: 2.236068
```

2.3

【实现方法】
采用教材第49页的fzerox函数，10个初始区间为$[2+3k, 5+3k], k = 0, 1, ..., 9$
【程序输出】
计算得到的十个根为

```
2.404826 5.520078 8.653728 11.791534 14.930918
18.071064 21.211637 24.352472 27.493479 30.634606
```

【函数图像】



【关键代码】

```
x = [1:0.1:40];
ab = [[2 : 3 : 31]; [5 : 3 : 34]]';
f = @(x) besselj(0, x);
rt = zeros(1, 10);
for i = 1:10
    rt(i) = fzerotx(f, ab(i, :));
end
plot(x, f(x), rt, f(rt), 'o')
grid on
fprintf("%.6f %.6f % .6f %.6f %.6f\n", rt)
```