非线性方程求根

计11 周韧平 2021010699

编程实现牛顿法与牛顿下山法求解下面两个方程. 要求: (1) 设定合适的迭代判停准则; (2) 设置合适的下山因子序列; (3) 打印每个迭代步的近似解及下山因子; (4) 请用其他较准确的方法 (如MATLAB软件中的 fzero 函数) 验证牛顿法与牛顿下山法结果的正确性。最后，总结哪个问题需要用牛顿下山法求解，及采用它之后的效果

实现牛顿法时，依照课本中的算法，迭代公式为 $x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$，判停准则为 $|f(x)| \le \epsilon$ 且 $\Delta x \le \epsilon$ 时，停止迭代，实现时我选取 $\epsilon = 10^{-6}$，其它算法实现细节与课本中的伪代码一致。为了避免不收敛，陷入死循环，我还设置了最大循环次数。

实现牛顿下山法时，我沿用了牛顿法中的判停准则，迭代公式为 $x_{t+1} = x_t - \lambda_i \frac{f(x_t)}{f'(x_t)}$，其中 λ_i 为预定义的因子序列，实现时我选取为首项为$1 - 10^{-10}$，公差0.5 的等比数列，课本伪代码要求当函数值的绝对值小于前一步函数值绝对值时，将此时的 λ_i 作为当前迭代步的下山因子，这部分我完全参考课本算法的实现。

对于第一个方程，牛顿下山法迭代过程如下，经过21次迭代最终停在 -1.76929235，迭代初期因子较小，后期逐渐变大，最后几步迭代都是第一个因子

```
x:1.0,lambda:0.9999999999
x:0.750000000025,lambda:0.1249999999875
x:0.8421875000481875,lambda:0.015624999984375
x:0.8142908075825246,lambda:0.001953124998046875
x:0.8168680524529868,lambda:1.5258789060974121e-05
x:0.8163905784348932,lambda:4.768371581554413e-07
x:0.8165997949836216,lambda:5.960464476943016e-08
x:0.8164923743696567,lambda:2.980232238471508e-08
x:0.8164975226024949,lambda:5.820766090764664e-11
x:0.8164960852516686,lambda:3.637978806727915e-12
x:0.8164967679143563,lambda:9.094947016819788e-13
x:0.8164965417087396,lambda:1.1368683771024735e-13
x:0.8164966091145992,lambda:7.105427356890459e-15
x:0.8164965622206183,lambda:3.5527136784452295e-15
x:0.8164965975492952,lambda:1.7763568392226148e-15
x:-1.8518477772026234,lambda:1.1920928953886032e-07
x:-1.7737928968917849,lambda:0.9999999999
x:-1.7693068309703985,lambda:0.9999999999
x:-1.769292354389133,lambda:0.9999999999
x:-1.7692923542386314,lambda:0.9999999999
final x:-1.7692923542386314
```

而如果采用牛顿法求解会导致陷入 -1，1 的死循环，算法最终没有收敛

```
0.0
x:1.0
x:0.0
x:1.0
...
x:1.0
x:0.0
final x:0.0
not converge
```

而对于第二个方程，两种算法都可以收敛到 2.23606

```
x:2.496958556035034,lambda:0.06249999999375
x:2.2719762054987482,lambda:0.9999999999
x:2.2369017057490677,lambda:0.9999999999
x:2.2360684433836067,lambda:0.9999999999
x:2.2360679774999355,lambda:0.9999999999
final x:2.2360679774999355
-1.4583889651476056e-12
x:10.525668449197836
x:7.124286625588786
x:4.910780653019383
x:3.516911305892172
x:2.7097430061997922
x:2.3369400314687754
x:2.2422442539928538
x:2.2360934030219455
x:2.236067977933435
x:2.23606797749979
final x:2.23606797749979
```

而通过调用 `scipy.optimize.root_scalar` 库可以得到两个方程的解为 -1.7692923542386314 和 2.23606797749979，我发现求解第一个方程时若采用 $x_0 = 0$ 也会导致无法收敛，我调整了合适的初值。该方法求解的值和牛顿（下山）法求解的接近

```python
from scipy.optimize import root_scalar
x = root_scalar(f,x0=-1)
print(x)
x = root_scalar(g,x0=1.35)
print(x)
```

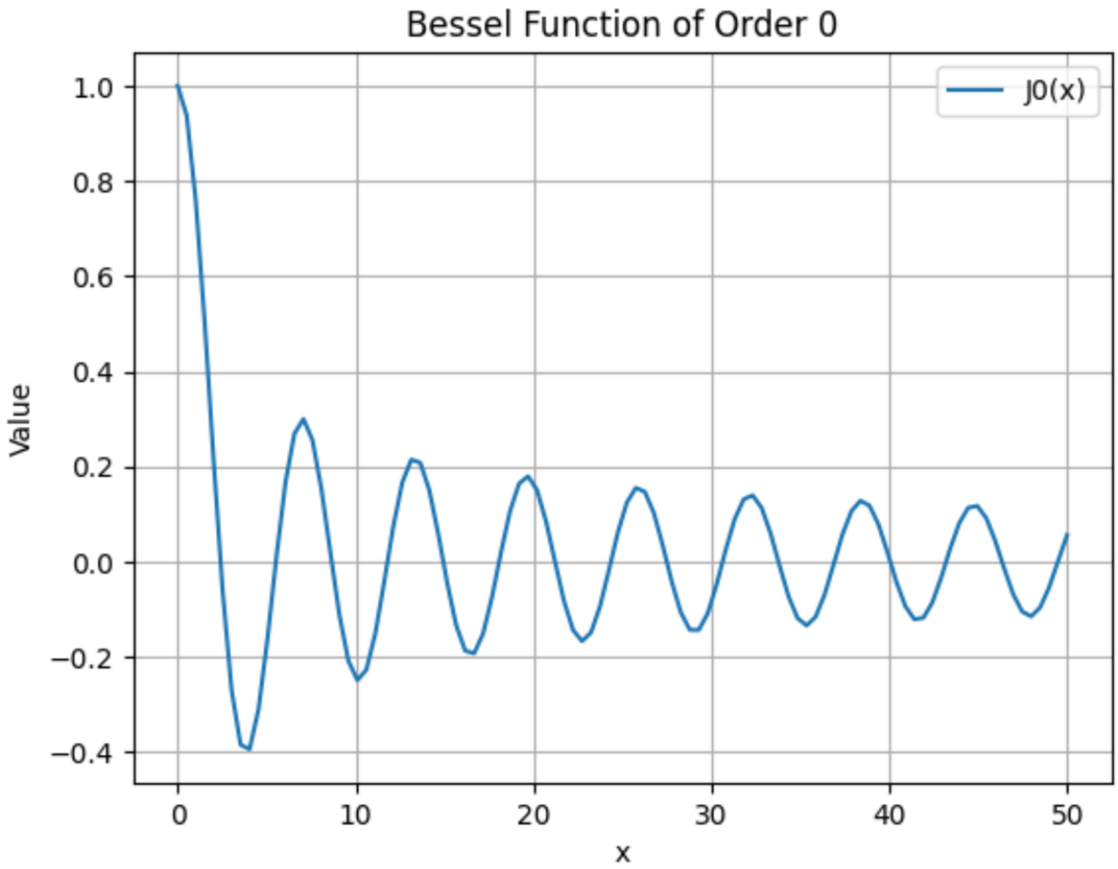
✓ 0.0s

```
      converged: True
           flag: converged
 function_calls: 16
     iterations: 8
           root: -1.769292354238631
         method: newton
      converged: True
           flag: converged
 function_calls: 20
     iterations: 10
           root: 2.23606797749979
         method: newton
```

利用 2. 6. 3 节给出的 fzerotx 程序，编程求第一类的零阶贝塞尔函数 的零点（在MATLAB 中可通过 besselj(0,x) 得到）. 试求 的前10个正的零点, 并绘出函数曲线和零点的位置

fzerotx 程序的实现完全参考课本，特别的，如果求解区间两端同号，函数返回 None，这是为了便于后面寻找区间内的解。

对于贝塞尔函数，我通过调用 `scipy.special.jn` 首先观察其形状，可以看到，前10个解大致在(0，50)的范围内，并且解之间的间隔不小于0.5



为了找到最小的10个正解，我以 $h=0.1$ 为步长，每次在区间 $(low, high)$ 中使用 zeroin 算法寻找解，一开始 $low=0$， $high=h$，之后逐步增大high，每次增加 h，直到找到一个解x，这时，将low设为 $low=x+h$， $high=low+h$，继续迭代，直到找到前10个解，最终找到的解为 2.4035503109479257, 5.52355031094786, 8.646223210325896, 11.796223210325829, 14.936223210325762, 18.076223210326063, 21.216223210326554, 24.356223210327045, 27.496223210327535, 30.636223210328026。绘图如下，可以看到求出的解时较为准确的，基本和函数与x轴交点重合。