《数值分析》课程实验报告

实验名称 实验二 非线性方程求根

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 班级 |  | 姓名 |  | 学号 |  | 序号 |  |
| 教师 |  | 地点 | 数学实验中心 | | | 评分 |  |
|  | | | | | | | |

1. 实验目的

① 掌握二分法、牛顿迭代法等常用的非线性方程迭代算法；

② 了解迭代算法的设计原理及初值对收敛性的影响。

1. 实验过程和结果

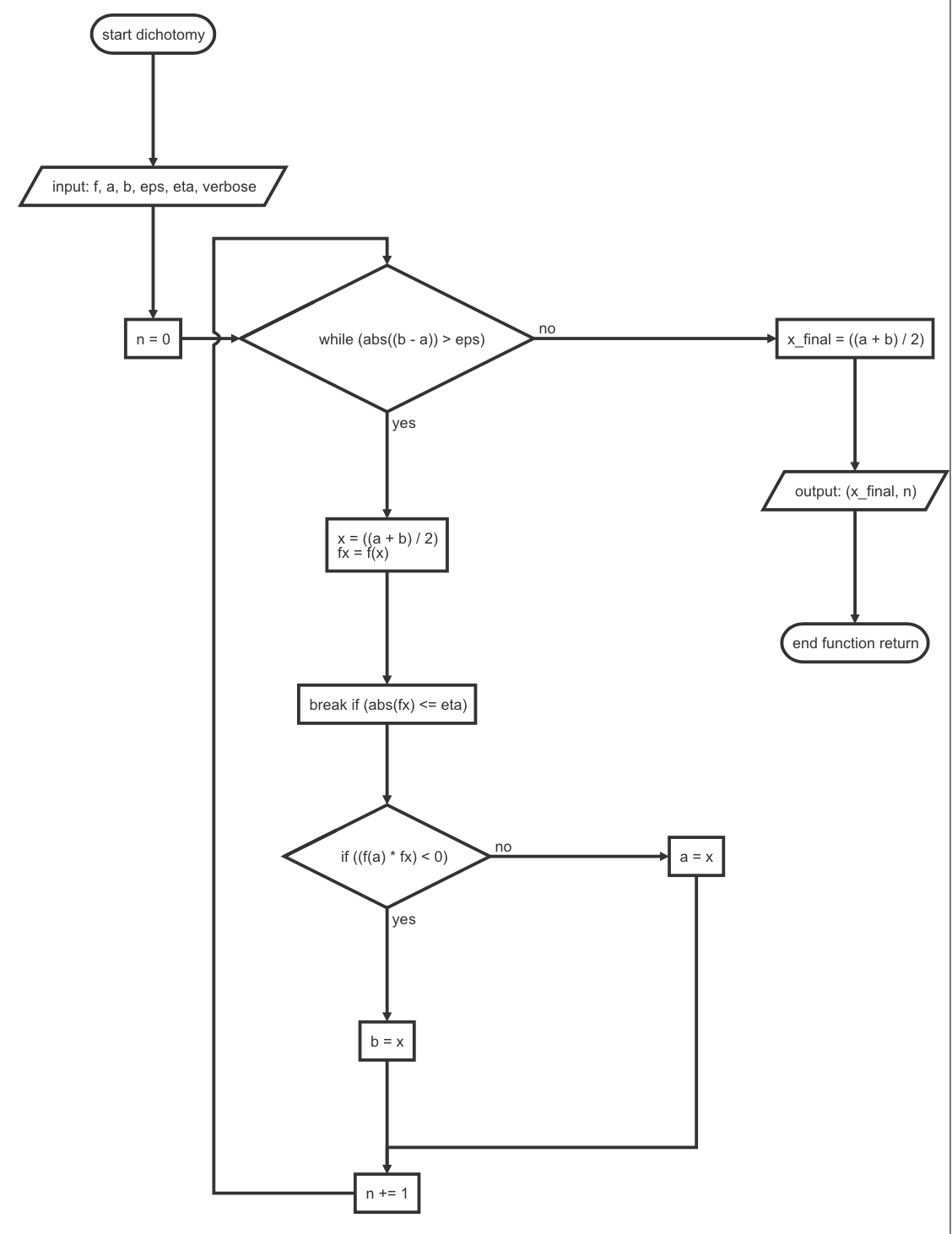
**1.** 二分法实现**[[1]](#footnote-1)**：

二分法的思想是对区间进行二分，在左右两个区间中确定下一次求根搜索的区间，如此反复来得到方程的根。

该算法的实现为以下函数：

dichotomy(f, a, b, eps, eta=1e-16, verbose=**False**)

输入：一元函数，表示要求根的方程: f(x) = 0， 有根区间 [a, b] 的端点，以及给定精度eps。



【图1】二分法程序流程图

输出：二分法求得的近似根 x\_final 和迭代次数 N。

其核心代码如下：

while abs(b - a) > eps:

x = (a + b) / 2

if abs(f(x))<=eta:

break

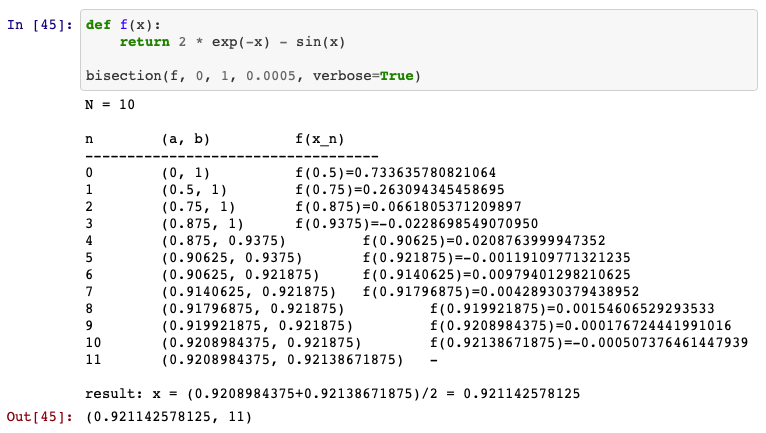
if f(a) \* f(x) < 0:

b = x

else:

a = x

x\_final = (a + b) / 2

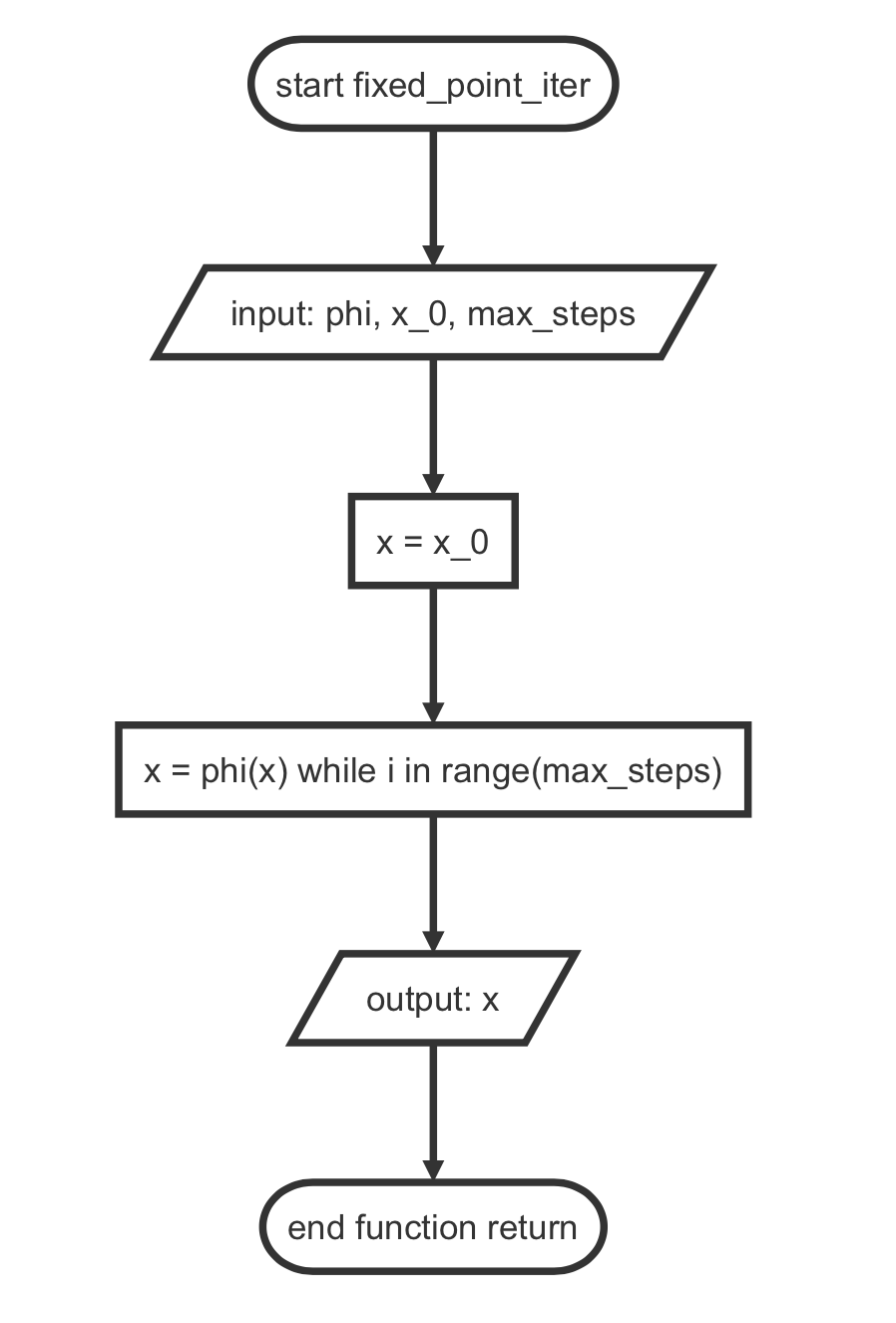
二分法实现测试：

**2.** 不动点迭代**[[2]](#footnote-2)**

不动点迭代的思想很简单，就是用一个函数反复作用于 x 来等到新的 x。

用如下函数来实现这个方法：

fixed\_point\_iter(phi, x\_0, max\_steps=25, verbose=**False**)



【图2】不动点迭代程序流程图

输入：

* + phi: function, 迭代函数
  + x\_0: float, 初值
  + max\_steps: 最大迭代次数

输出：

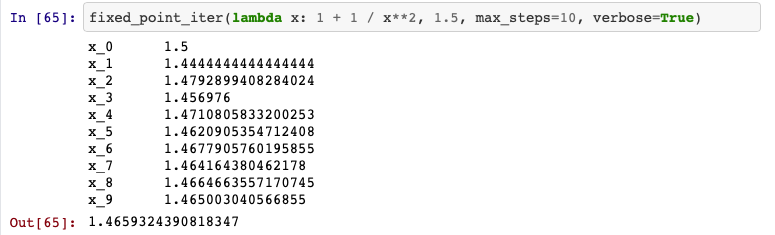
* + x\_final: float, 最终的近似根 x

不动点迭代的核心代码为：

x = x\_0

**for** i **in** range(max\_steps):

x = phi(x)

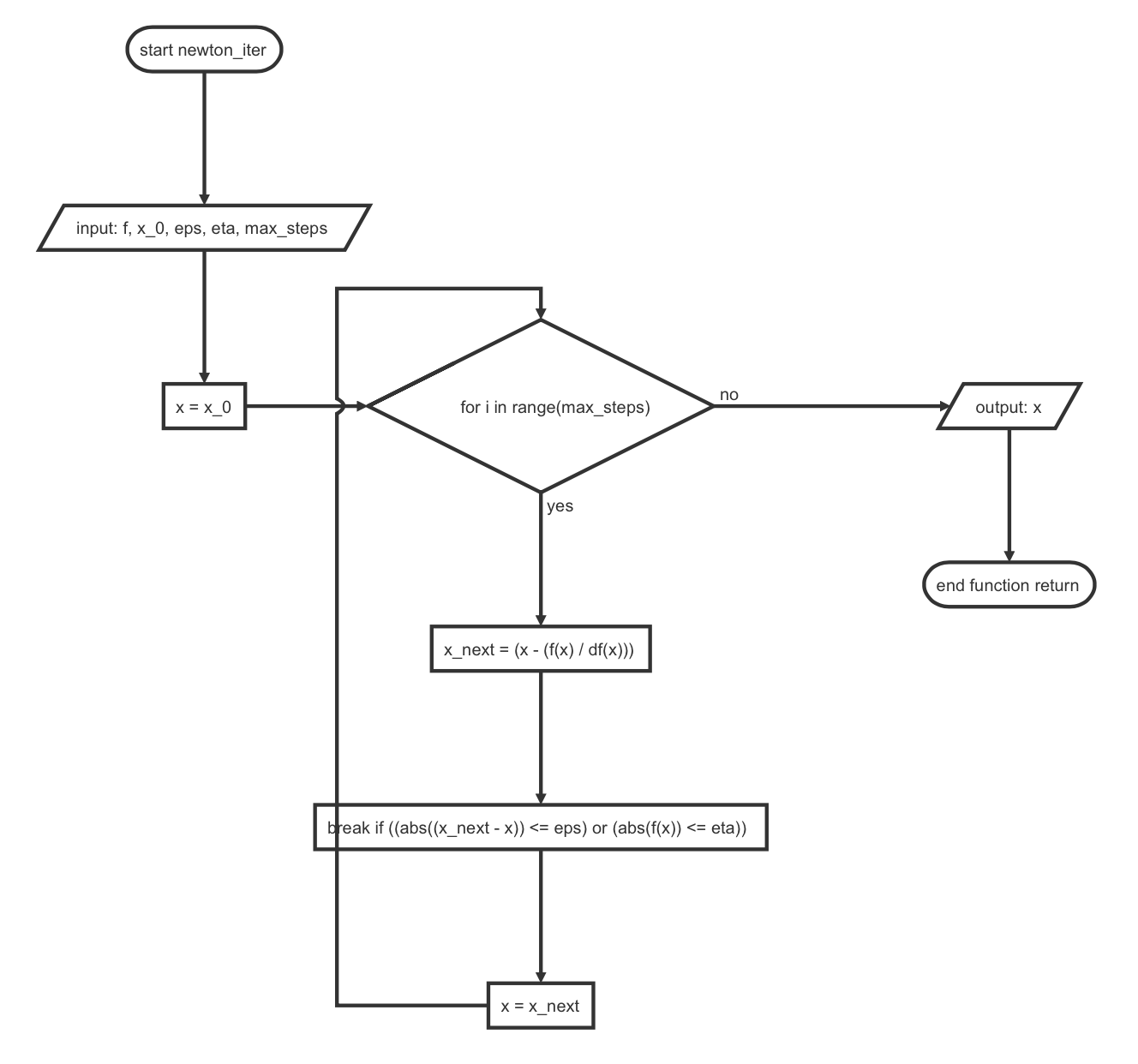
不动点迭代测试：

**3.** 牛顿迭代法**[[3]](#footnote-3)**

牛顿迭代法用函数的导数（切线）去迭代逼近曲线的根。这个方法的收敛速度明显优于二分法。

实现函数：

newton\_iter(f, x\_0, eps=0, eta=0, df=**None**, max\_steps=20, frac=**False**, verbose=**False**)



【图3】牛顿迭代程序流程图

输入参数：

* f: function, 迭代函数
* x\_0: float, 初值
* eps: float, 根的容许误差, default 0.
* eta: float, abs(f(x)) 的容许误差, default 0.
* df: function, f 的导函数, 默认 None 表示自动调用 sympy.diff 求导(会导致后续迭代中使用分数运算)。
* max\_steps: int, 最大迭代次数, default 20.
* frac: bool, True 则输出分数(仅对 df=None 时生效)，否则使用 float, default False.
* verbose: bool, 打印出每一步的值, default False.

输出：

* x\_final: float, 最终的近似根 x

该算法的核心代码是：

x = x\_0

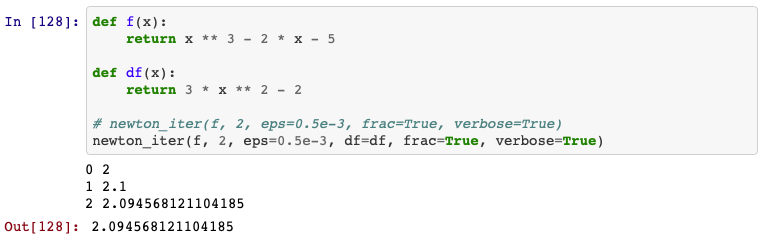
**for** i **in** range(max\_steps):

x\_next = x - f(x) / df(x)

**if** abs(x\_next - x) <= eps **or** abs(f(x)) <= eta:

**break**

x = x\_next

牛顿迭代法测试：

**4.** 单点弦截法**[[4]](#footnote-4)**

虽然牛顿法的收敛速度快，但每一步都需要计算导数值 。为了避开导数的计算，使用 来代替导数，就得到了单点弦截法：

编程实现：

single\_point\_truncation(f, x\_0, x\_1, eps=0, eta=0, max\_steps=20, verbose=**False**)

这个函数和牛顿迭代的实现 newton\_iter 很类似，只需把迭代更新修改了一下：

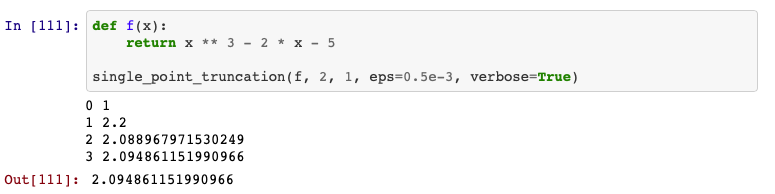
x\_next = x - f(x) / (f(x) - f\_x0) \* (x - x\_0)

输入：

* + f: function, 迭代函数
  + x\_0, x\_1: float, 初值

输出：

* + x\_final: float, 最终的近似根 x

测试：

**5.** 两点弦截法（割线法）**[[5]](#footnote-5)**

使用差商 来代替牛顿迭代法中的导数，就得到了两点弦截法（割线法）:

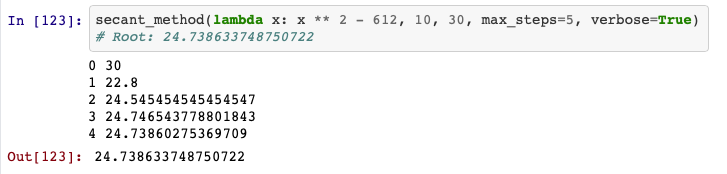
编程实现：

secant\_method(f, x0, x1, eps=0, eta=0, max\_steps=20, verbose=**False**)

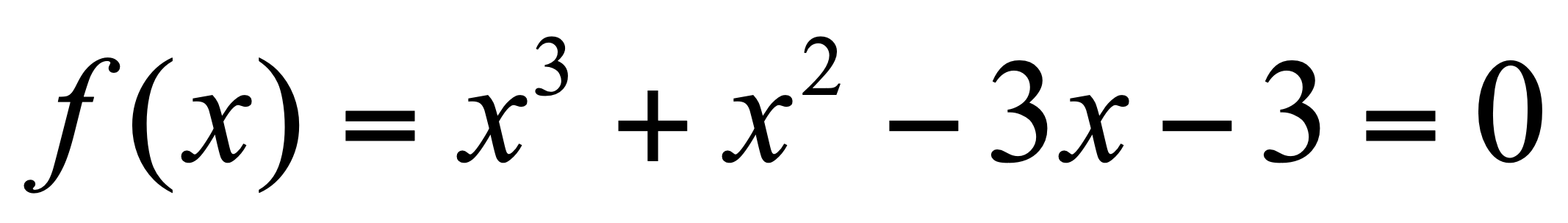
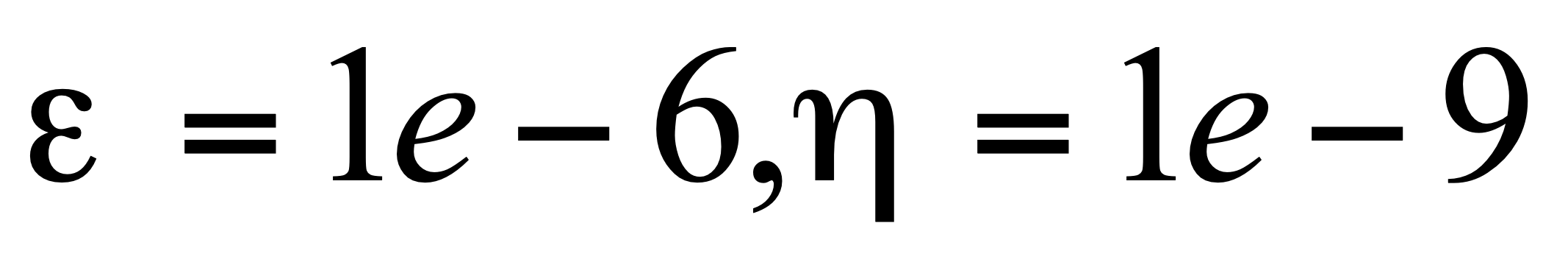
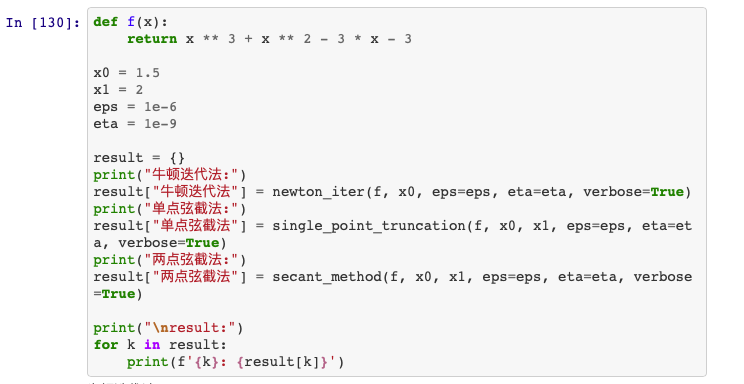
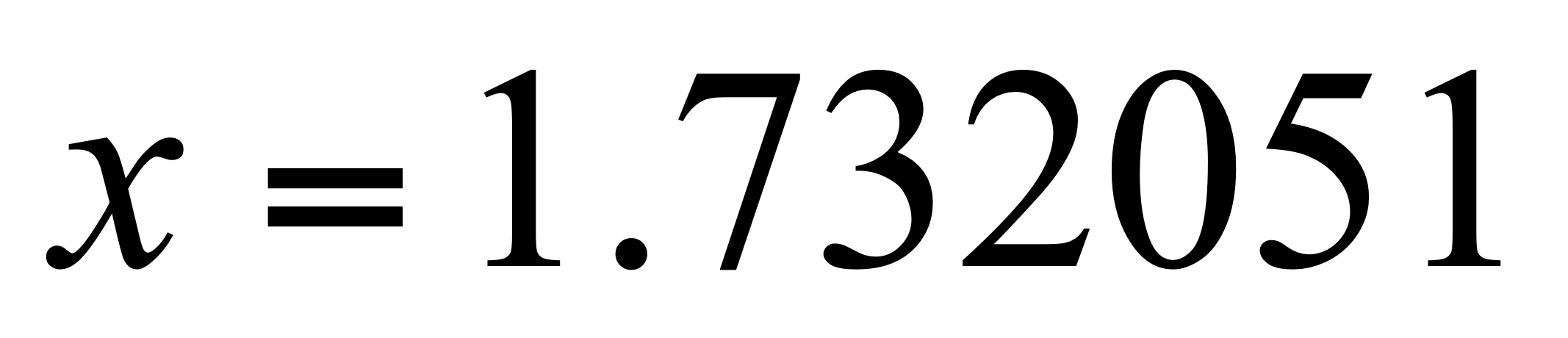
代码中只需要修改迭代更新：

x2 = x1 - f(x1) \* (x1 - x0) / (f(x1) - f(x0))

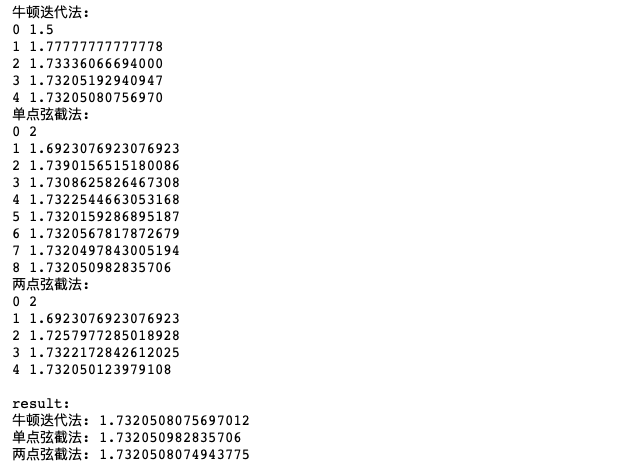
x0, x1 = x1, x2

测试：

**6.** 题目 **[[6]](#footnote-6)**

求方程在1.5 附近的根.（误差限为）。参考答案：原方程的根为

输出结果：



三、思考题分析解答

二分法简单易行，但固定每次缩短一半的区间，只能求单实根，不能求复根或偶数重根。而牛顿迭代法的迭代效率往往更高，一般情况下使用牛顿迭代法可以获得更快的收敛速度。

虽然牛顿法的收敛速度快，但每一步都需要计算导数值 。为了避开导数的计算，使用 来代替导数，即单点弦截法；或者，也可以使用差商 来代替导数，即两点弦截法（割线法）。

四、重点难点分析

重点：

1. 掌握二分法、牛顿迭代法等常用的非线性方程迭代算法；
2. 了解迭代算法的设计原理及初值对收敛性的影响。

难点：

1. 牛顿迭代法的改进。

1. 二分法完整代码实现见：<https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex2/src/dichotomy.py> [↑](#footnote-ref-1)
2. 不动点迭代的完整代码见：<https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex2/src/fixed_point_iter.py> [↑](#footnote-ref-2)
3. 牛顿迭代法的完整代码见：<https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex2/src/newton_iter.py> [↑](#footnote-ref-3)
4. 单点弦截法的完整代码见：<https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex2/src/single_point_truncation.py> [↑](#footnote-ref-4)
5. 两点弦截法（割线法）的完整代码见：<https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex2/src/secant_method.py> [↑](#footnote-ref-5)
6. 具体实现见：<https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex2/src/ex2.ipynb> [↑](#footnote-ref-6)