

实验报告（一）

1004151126 沈嘉浩

1.实验目的

分别使用二分法、牛顿法、牛顿下山法和弦截法，计算下列两个方程的实根。

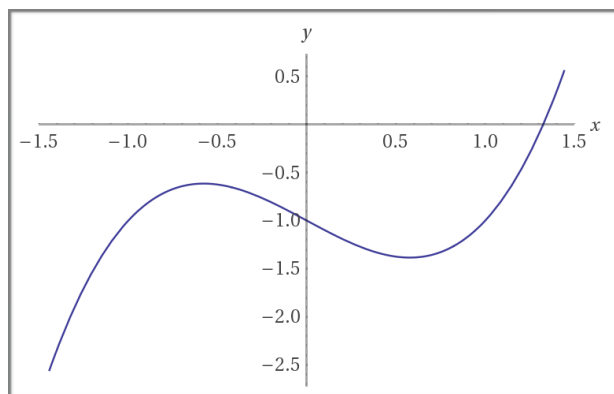
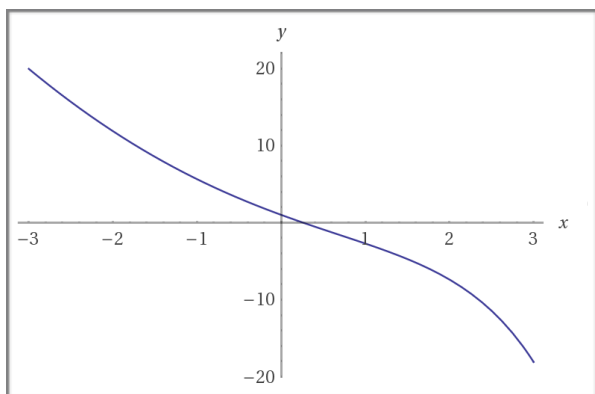
1. $x^2 - 3x + 2 - e^x = 0$

2. $x^3 - x - 1 = 0$

2.实验要求

- ① 计算结果精确到 $|x_k - x_{k-1}| < 10^{-8}$
- ② 输出迭代初始值以及迭代次数，比较各方法的优劣

3.实验原理



左侧图片是 $f(x) = x^2 - 3x + 2 - e^x$ 的函数图像，右侧是函数 $g(x) = x^3 - x - 1$ 的图像，我们首先估计一下两个方程的零点大致在哪个区间内，这样便于下一步的计算。

(1) 二分法

- ① 首先我们要确定一个区间 $[left, right]$ ，其中 $f(x)$ 在该区间上是连续且单调的，且满足条件 $f(left) \cdot f(right) < 0$
- ② 然后计算 $mid = \frac{left + right}{2}$
- ③ 如果 $f(left) \cdot f(mid) < 0$ ，则令 $right = mid$

- ④ 否则令 $left = mid$
- ⑤ 跳转至步骤2直到满足精度条件

(2) 牛顿法

- ① 对于 $f(x)$ ，我们首先求解其导数 $f'(x)$
- ② 然后选取一个合适的初始值 x_0
- ③ 然后根据递推公式 $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ 计算出下一个值
- ④ 反复求解直到满足精度条件

对于牛顿法，如果一开始选取的初始值 x_0 不合理，可能会导致在递推的过程中， x_k 的值不收敛，因此选取合适的初始值显得特别重要。

(3) 牛顿下山法

相比于牛顿法，牛顿下山法带入了一个下山因子 λ ，具体过程如下：

- ① 对于 $f(x)$ ，求解其导数 $f'(x)$
- ② 选取一个任意的初始值 x_0
- ③ 对于每一个 x_{k+1} 和 x_k ，其递推公式为 $x_{k+1} = x_k - \lambda \frac{f(x_k)}{f'(x_k)}$ ，首先

令 $\lambda = 1$ ，然后对 λ 逐次取半直到满足条件 $|f(x_{k+1})| < |f(x_k)|$

牛顿下山法将牛顿法和下山法结合，保证了函数值稳定下降的前提，又用牛顿法加快了收敛速度。

(4) 弦截法

弦截法的递推公式为 $x_{k+1} = x_k - \frac{f(x_k)}{f(x_k) - f(x_{k-1})}(x_k - x_{k-1})$

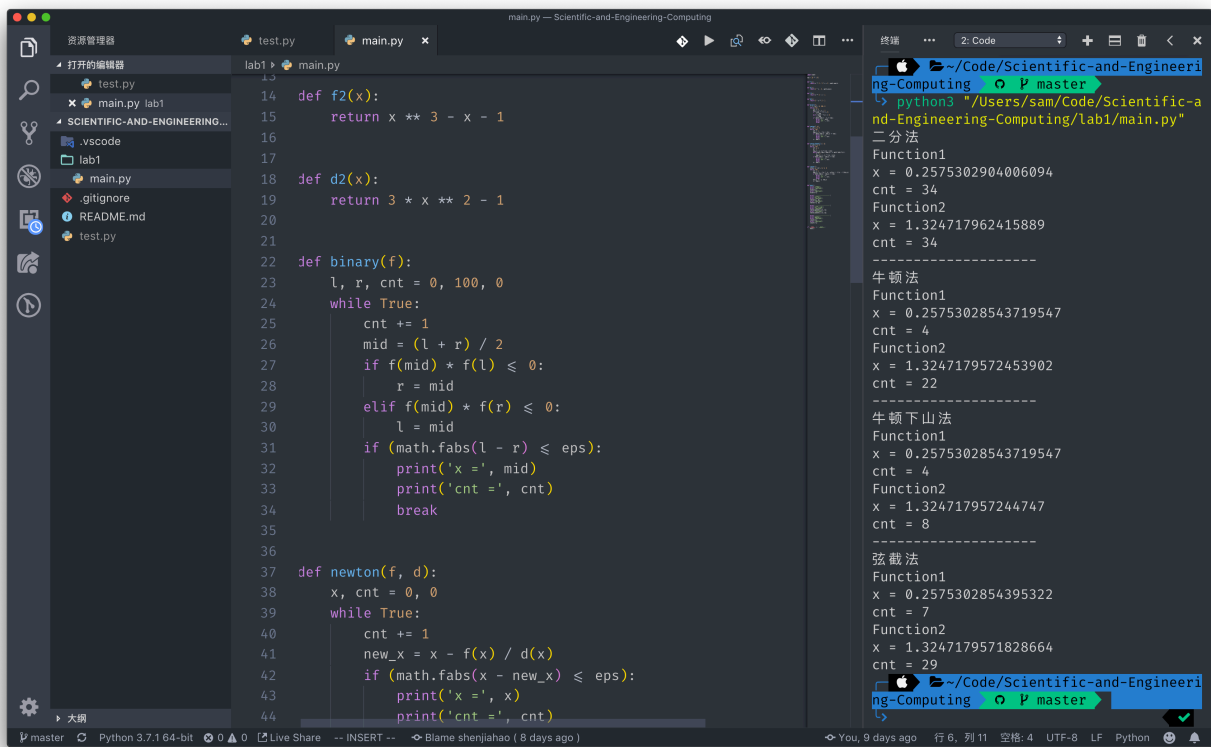
其中弦截法需要确定2个初始值 x_0 和 x_1 ，然后分别带入递推公式进行求解，直到满足精度条件。

4.实验结果

实验在macOS Mojave 10.14上进行编写，使用的Python3.7.1运行，代码已在Github上开源，地址为：

<https://github.com/jiahao-shen/Scientific-and-Engineering-Computing/blob/master/lab1/main.py>

运行截图如下：



The screenshot displays the Visual Studio Code (VS Code) interface. The main editor window shows a Python file named `main.py` with the following code:

```
14 def f2(x):
15     return x ** 3 - x - 1
16
17
18 def d2(x):
19     return 3 * x ** 2 - 1
20
21
22 def binary(f):
23     l, r, cnt = 0, 100, 0
24     while True:
25         cnt += 1
26         mid = (l + r) / 2
27         if f(mid) * f(l) <= 0:
28             r = mid
29         elif f(mid) * f(r) <= 0:
30             l = mid
31         if (math.fabs(l - r) <= eps):
32             print('x =', mid)
33             print('cnt =', cnt)
34             break
35
36
37 def newton(f, d):
38     x, cnt = 0, 0
39     while True:
40         cnt += 1
41         new_x = x - f(x) / d(x)
42         if (math.fabs(x - new_x) <= eps):
43             print('x =', x)
44             print('cnt =', cnt)
```

The right-hand side of the image shows the output of the program execution in the terminal. The output is as follows:

```
~/Code/Scientific-and-Engineering-Computing
python3 "/Users/sam/Code/Scientific-and-Engineering-Computing/lab1/main.py"
二分法
Function1
x = 0.2575302904006094
cnt = 34
Function2
x = 1.324717962415889
cnt = 34
-----
牛顿法
Function1
x = 0.25753028543719547
cnt = 4
Function2
x = 1.3247179572453902
cnt = 22
-----
牛顿下山法
Function1
x = 0.25753028543719547
cnt = 4
Function2
x = 1.324717957244747
cnt = 8
-----
弦截法
Function1
x = 0.2575302854395322
cnt = 7
Function2
x = 1.3247179571828664
cnt = 29
```

The status bar at the bottom indicates the current file is `main.py` in the `Scientific-and-Engineering-Computing` workspace, using Python 3.7.1 64-bit. The terminal output shows the results of the binary, Newton, Newton下山, and弦截 methods, including the number of iterations (cnt) for each function.