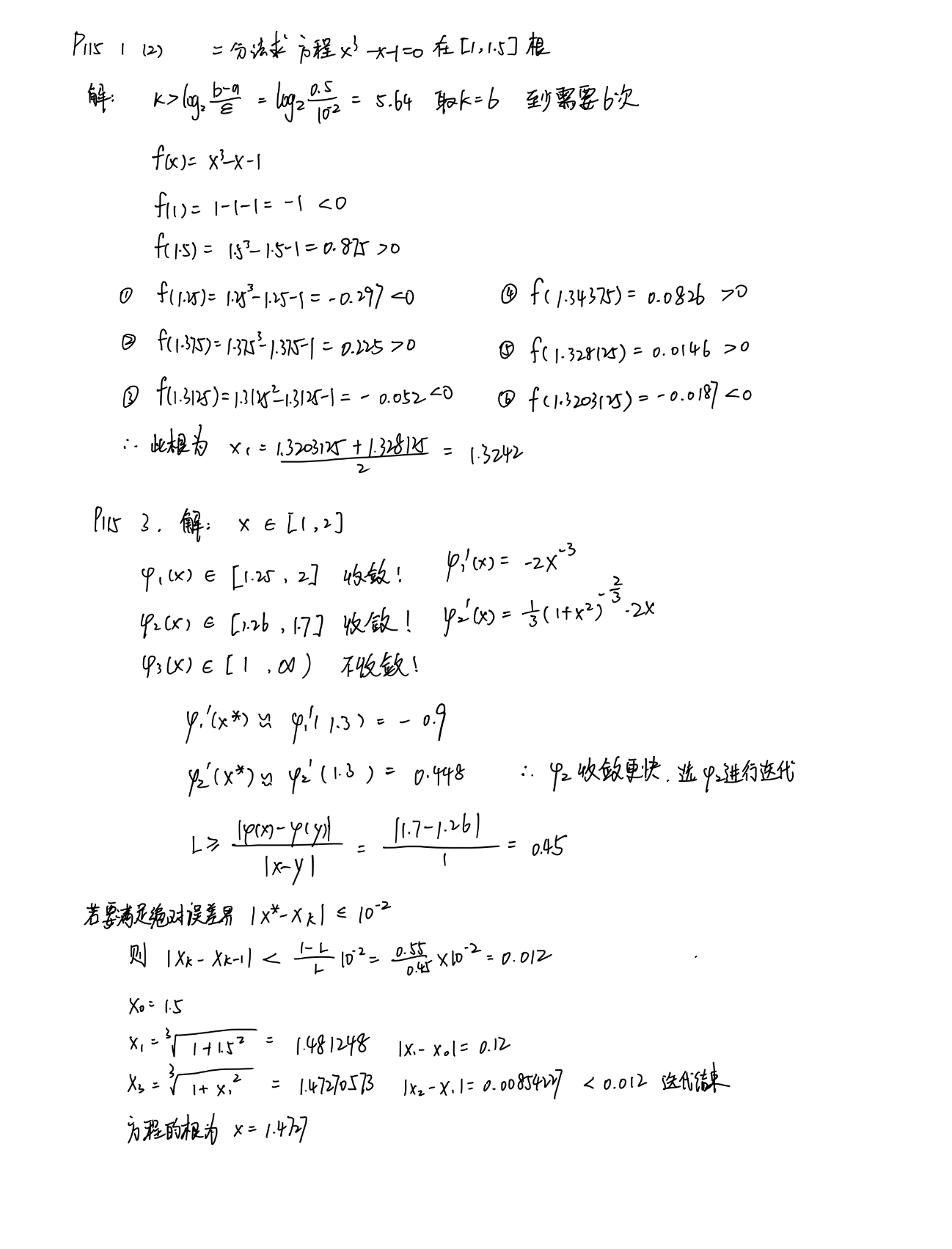
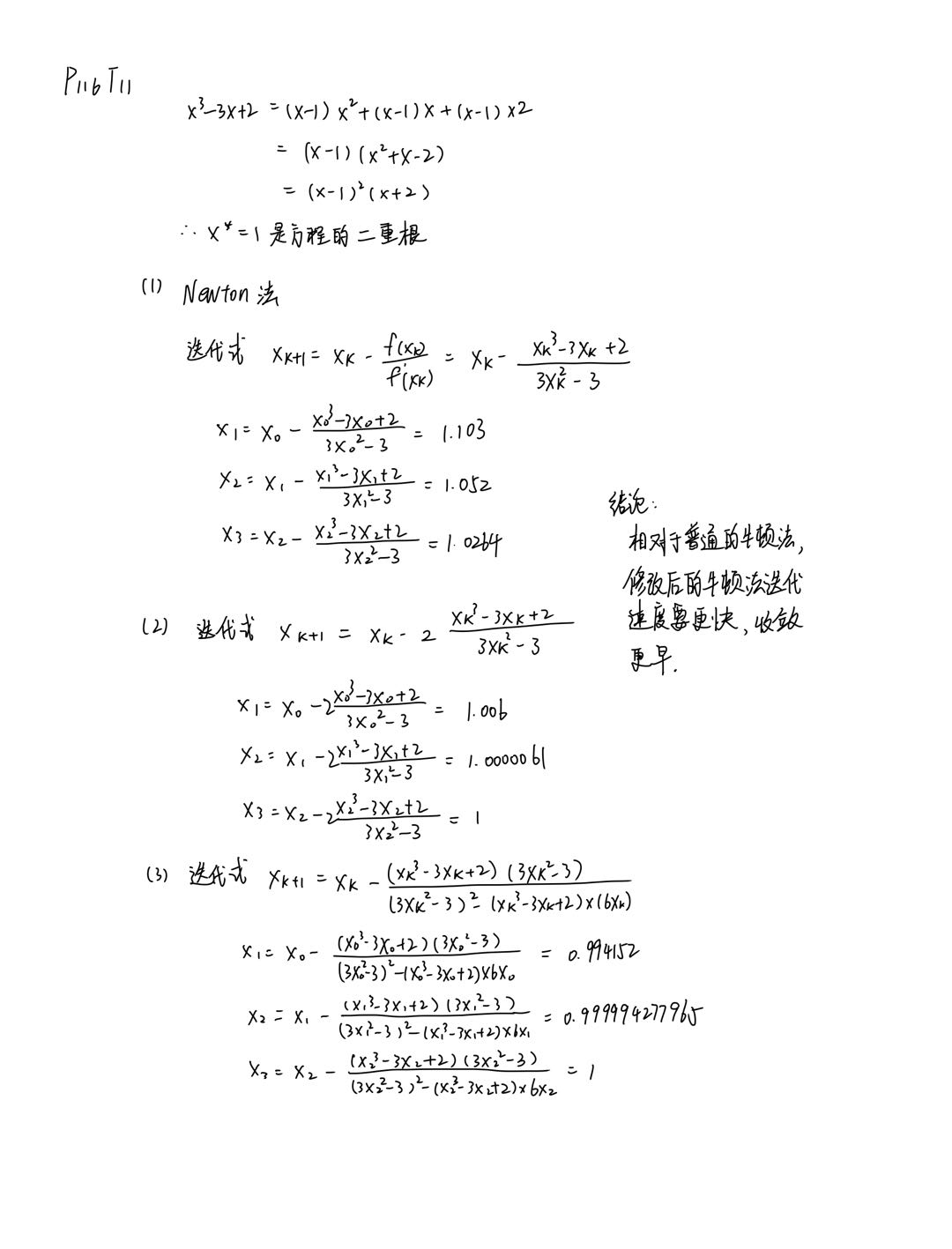
第五次作业

赖显松 2021214726

1 书面作业题目





2 编程题目

2.1 P116 T1

**不动点迭代法函数**

编写不动点迭代法函数（输入：*x*初值，误差要求，迭代函数Fx；返回：迭代求解结果）。代码如下：

1. # fixed-point Iteration
2. **def** fixedPtIter(x\_old, eps, Fx, \*args, \*\*kwargs):
3. max\_iter = 100
4. err = 10
5. it\_num = 1
6. **while** err > eps **and** it\_num <= max\_iter:
7. x\_new = Fx(x\_old)
8. err = abs(x\_new-x\_old)
9. x\_old = x\_new
10. it\_num += 1
11. **print** ('iteration times: ', it\_num)
12. **return** x\_new

编写(1)、(2)、(5)方法的迭代公式：

(1):

1. **def** iterFx1(x):
2. **return** 20 / (x\*\*2 + 2\*x +10)

(2):

1. **def** iterFx2(x):
2. **return** (20 - 2\*x\*\*2 - x\*\*3)/10

(5):

1. **def** Fx(x):
2. **return** x\*\*3 + 2\*x\*\*2 + 10\*x - 20

5. **def** dFx(x):
6. **return** 3\*x\*\*2 + 4\*x + 10

9. **def** newtonIter(x):
10. **return** x - Fx(x) / dFx(x)

**结果**

待验证的迭代结果约为1.368808107，所以迭代精度设置为10-9。初值取1，求得不同迭代方法的迭代次数以及结果如图2. 1所示：

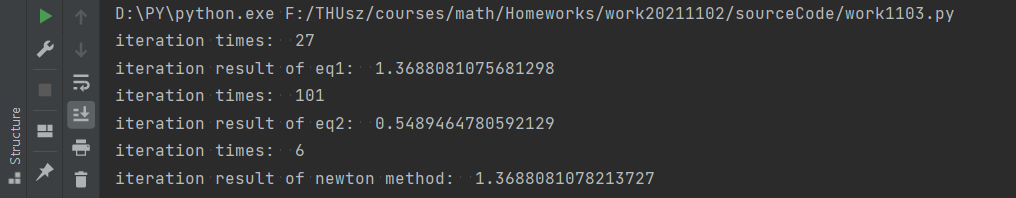


图2. 1 3种方法的迭代结果

证明Leonardo所得到结果是准确的。

从结果我们能够看出，迭代法(1)、(5)牛顿法都是可以收敛的，且牛顿法的收敛速度要快得多。而(2)迭代式最终并没有成功收敛。

2.2 附加题

已知根多项式方程可以表示为(2- 1)：



(2- )

根据乘法求导法则，其一阶导数为(2- 2)：



(2- )

以此类推，这种多项式的导数相当于每次求导，将连乘项减少一项进行组合，并将组合相加，例如(2- 3)：



(2- )

**编程细节**

通过编程实现这种多项式的生成及求导，采用查表的方式：

一个二维数组表示用来生成已知根多项式及其导数，数组中的元素代表这些根，表格中的每一行元素相乘，最后将所有行相加。

例如：可以表示为：[[2, 3],[1, 3],[1, 2]]。这种表示方法我称为“乘加表”。

**函数1：**除外组合

这个函数输入一个这样的乘加表，输出对应多项式导数的乘加表。原理在于，对于源数组的每一行，求其减少一个元素的组合（暗顺序依次去掉其中的一个元素，剩下的元素就是一个组合），最后将所有组合作为一行按照从上至下的顺序排列。若输入是维的数组，则输出为维的数组。代码如下：

1. **def** exceptTable(na):
2. m, n= na.shape[0], na.shape[1]
3. nb = np.zeros((n\*m, n-1), dtype=int)
4. **for** i1 **in** range(m):
5. **for** j1 **in** range(n):
6. ej = 0
7. j2 = 0
8. **while** ej < n:
9. **if** ej != j1:
10. nb[i1\*n+j1, j2] = na[i1, ej]
11. j2 += 1
12. ej += 1
13. **return** nb

**函数2：**乘加表变为多项式

行元素相乘，最后将每一行相加，代码如下：

1. **def** mulPlus(x, na):
2. y = 0
3. **for** i **in** range(na.shape[0]):
4. yr = 1
5. **for** j **in** range(na.shape[1]):
6. yr = yr \* (x-na[i, j])
7. y += yr
8. **return** y

**函数3：**生成已知根多项式及其导数

输入自变量、最初的已知根多项式系数序列、求导次数，输出代入自变量的多项式计算结果，没求一次导，相当于多调用一次除外组合函数。代码如下：

1. **def** omigaPolynomial(x, xi, d\_times):
2. **for** k **in** range(d\_times):
3. xi = exceptTable(xi)
4. **return** mulPlus(x, xi)

多根多项式求解需要用到修正的牛顿迭代函数。进入实现环节，先定义原多项式，再定义其一阶导、二阶导函数。代码如下：

1. **def** MrFx(x, xi):
2. **return** nleqi.omigaPolynomial(x, xi, 0)

5. **def** dMrFx(x, xi):
6. **return** nleqi.omigaPolynomial(x, xi, 1)

9. **def** d2MrFx(x, xi):
10. **return** nleqi.omigaPolynomial(x, xi, 2)

再定义不动点迭代法迭代函数，代码如下：

1. **def** mrNewtonIterF(x, xi: np.ndarray):
2. **return** x - (MrFx(x, xi)\*dMrFx(x, xi)) / ((dMrFx(x, xi)\*\*2)-(MrFx(x, xi)\*d2MrFx(x, xi)))

定义多项式根序列及初值，每次求解得到一个根之后，更新根序列表，最后一个根为一次函数，直接输出根的值。代码如下：

1. xi = np.array([[1, 2, 3, 4]])
2. root\_nums = xi.shape[1]
3. x\_0 = 28
4. x\_solve\_mn = []
5. **for** i **in** range(root\_nums):
6. # only one root left
7. **if** xi.shape[1] == 1:
8. x\_solve\_mn.append(xi[0, 0])
9. **break**
10. **else**:
11. x\_star = nleqi.mrNewtonFP(x\_0, 10\*\*(-3), mrNewtonIterF, xi)
12. x\_solve\_mn.append(x\_star)
13. temp = np. zeros((1, xi.shape[1]-1))
14. k = 0
15. **for** j **in** range(xi.shape[1]):
16. **if** round(xi[0, j]) != round(x\_star):
17. temp[0, k] = xi[0, j]
18. k += 1
19. **if** k == temp.shape[1]:
20. **break**
21. xi = temp.copy()
22. **print**('iteration result of fixed newton method: ', x\_solve\_mn)

**结果**

代入初值为0时的求解结果为如图2. 2：

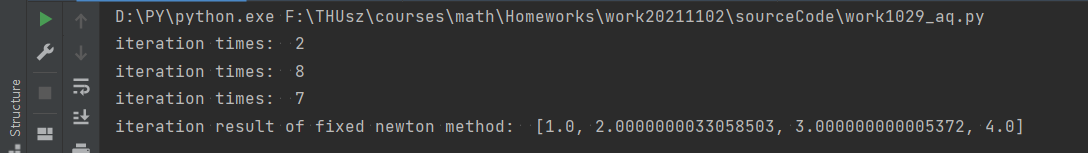


图2. 2 初值为零时的多根多项式修正牛顿法迭代结果

求出了四个根[1, 2, 3, 4]。

**初值敏感性分析**

以0.1为间隔，多次取不同的初值，得到四个解随初值求解先后顺序，画出变化图线。如图2. 3：

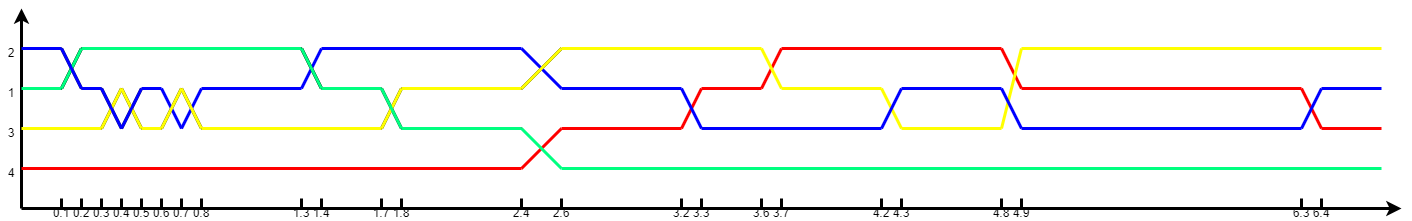


图2. 3 求根先后顺序随初值变化曲线

X轴为初值，先求出的根在Y轴的高度更高。在不同初值情况下，根被计算出来的先后次序不同。在初值很小的时候，先被计算出来的根为2，最后直接得出的是4；在初值很大的时候，先被计算出来的根为3，最后直接得出的是1。所以1和4的初值敏感度较大，初值不好时不易得到，粗略比较计算得出顺序，可以认为4的初值敏感性大于1，而2的初值敏感性小于3。最后的初值敏感性顺序为：2<3<1<4。

参考文献

1. 关冶．数值方法[M]．北京：清华大学出版社，2006：66-92．

附录

见附件python源代码。