数值实验报告 I

实验名称	第二次上机作业			实验时间	2021 年 9 月 18 日		
姓名	孙百乐	班级	本研 AI2001	学号	2007010218	成绩	

一、实验目的

1．理解并掌握二分法、迭代法和牛顿法的使用。

2．学会通过编程实现上述三种方法的求值应用实例。

二、实验内容

二分法：

（1） 二分法的基本原理

数学方面牛顿二分法

一般地，对于函数 f(x),如果存在实数 c,当 x=c 时，若 f(c)=0,那么把 x=c 叫做函数 f(x)的零点。

解方程即要求 f(x)的所有零点。

假定 f(x)在区间（x，y）上连续

先找到 a、b 属于区间（x，y），使 f(a)，f(b)异号，说明在区间(a,b)内一定有零点，然后求 f[(a+b)/2],

现在假设 f(a)<0,f(b)>0,a<b

①如果 f[(a+b)/2]=0，该点就是零点，

②如果 f[(a+b)/2]<0,则在区间（(a+b)/2，b)内有零点，(a+b)/2 赋给 a，从①开始继续使用中点函数值判断。

③如果 f[(a+b)/2]>0，则在区间(a,(a+b)/2)内有零点，(a+b)/2 赋给 b，从①开始继续使用中点函数值判断。

这样就可以不断接近零点。当区间小于一定值时，结束迭代过程。

通过每次把 f(x)的零点所在小区间收缩一半的方法，使区间的两个端点逐步迫近函数的零点，以求得零点的近似值，这种方法叫做二分法。

从以上可以看出，每次运算后，区间长度减少一半，是线性收敛。另外，二分法不能计算复根和重根。

（2） 编程实现

T2.1 (3)

```python
def f(x):
    return x**3+4*x**2-10
a=1;b=2;c=0.00025
```

```python
import math
k = math.log((b-a)/c)/math.log(2)-1
if k!=int(k):
    k = int(k)+1
print(f"迭代{k}次")
```

```
迭代11次
```

```python
for i in range(k+1):
    mid = (b+a)/2
    if f(a)*f(mid) > 0:
        a = mid
    else:
        b = mid
    print(f"第{i+1}次mid={mid}")
```

```
第1次mid=1.5
第2次mid=1.25
第3次mid=1.375
第4次mid=1.3125
第5次mid=1.34375
第6次mid=1.359375
第7次mid=1.3671875
第8次mid=1.36328125
第9次mid=1.365234375
第10次mid=1.3642578125
第11次mid=1.36474609375
第12次mid=1.364990234375
```

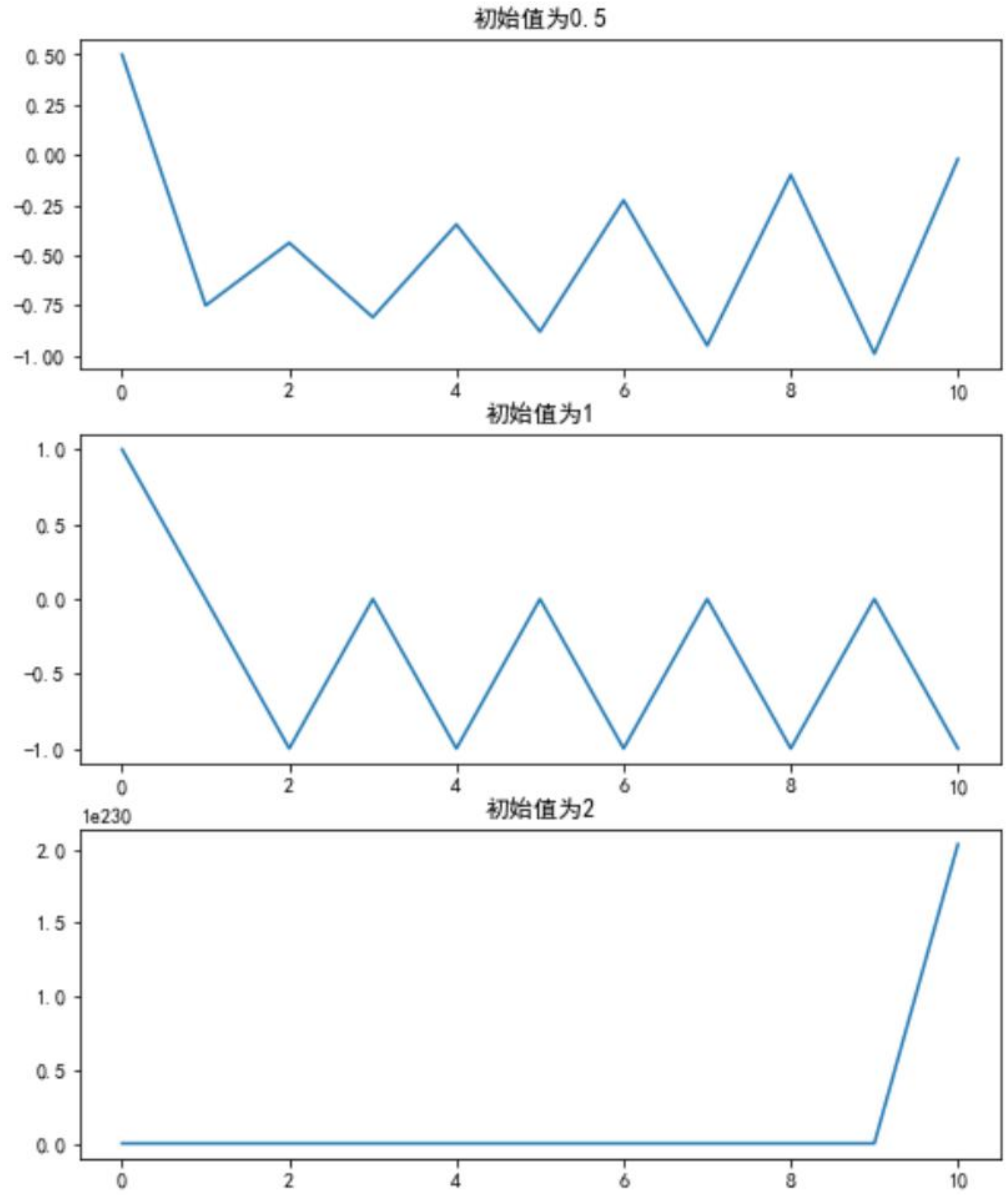
代码及结果：

迭代法：
代码及结果：

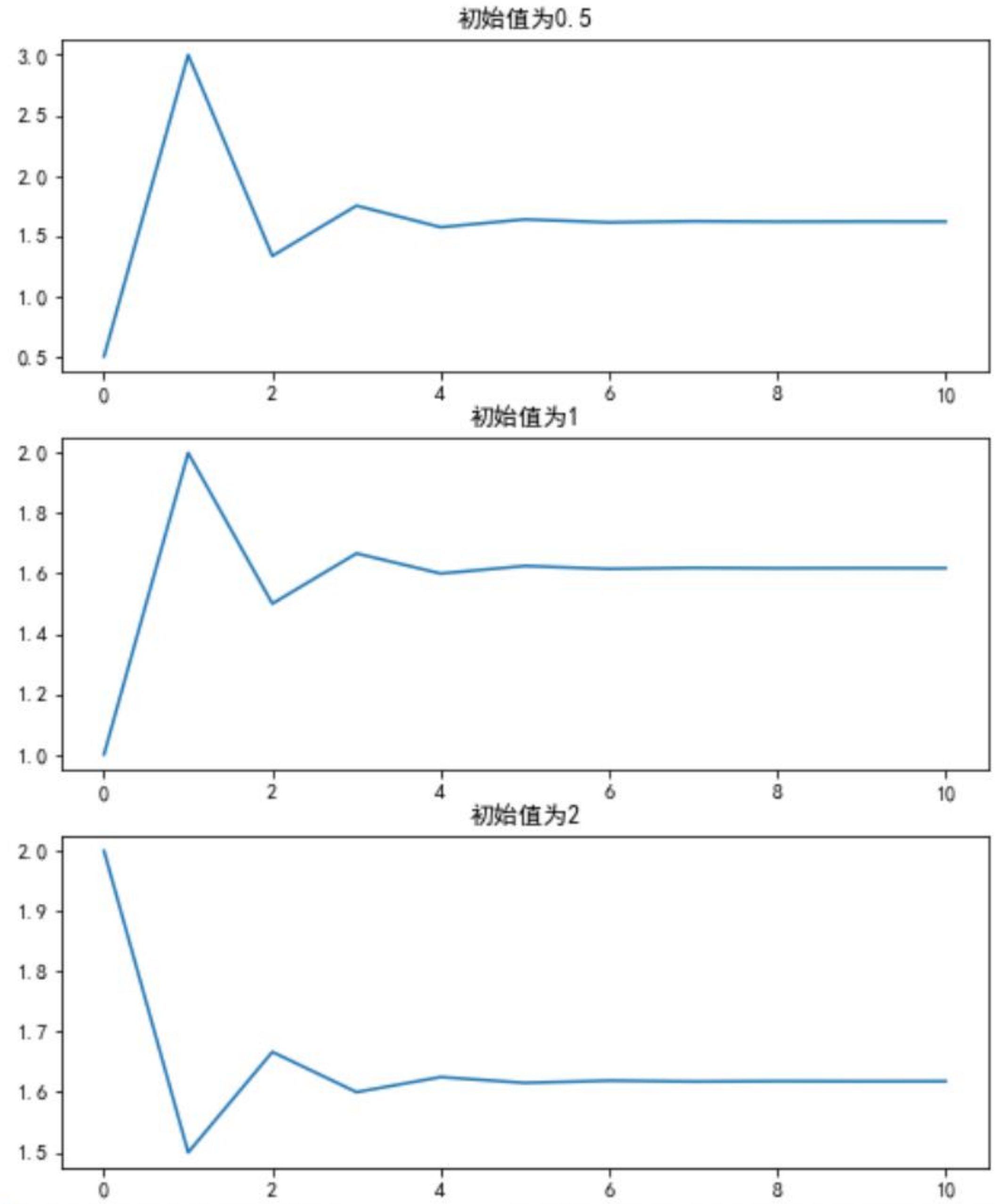
```python
import math
def f1(x):
    return x**2-1
def f2(x):
    return 1+1/x
def f3(x):
    return math.sqrt(x+1)
```

```python
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
# import numpy as np
def plot_img(init, f):


    plt.figure(figsize=(8,10))

    ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)
    ax2 = plt.subplot2grid((3, 3), (1, 0), colspan=3)
    ax3 = plt.subplot2grid((3, 3), (2, 0), colspan=3)

    diedai = [init[0]]
    for i in range(10):
        diedai.append(f(diedai[-1]))
    ax1.plot(list(range(11)),diedai)
    ax1.set_title(f"初始值为{init[0]}")

    diedai = [init[1]]
    for i in range(10):
        diedai.append(f(diedai[-1]))
    ax2.plot(list(range(11)),diedai)
    ax2.set_title(f"初始值为{init[1]}")


    diedai = [init[2]]
    for i in range(10):
        diedai.append(f(diedai[-1]))
    ax3.plot(list(range(11)),diedai)
    ax3.set_title(f"初始值为{init[2]}")


    plt.show()
```

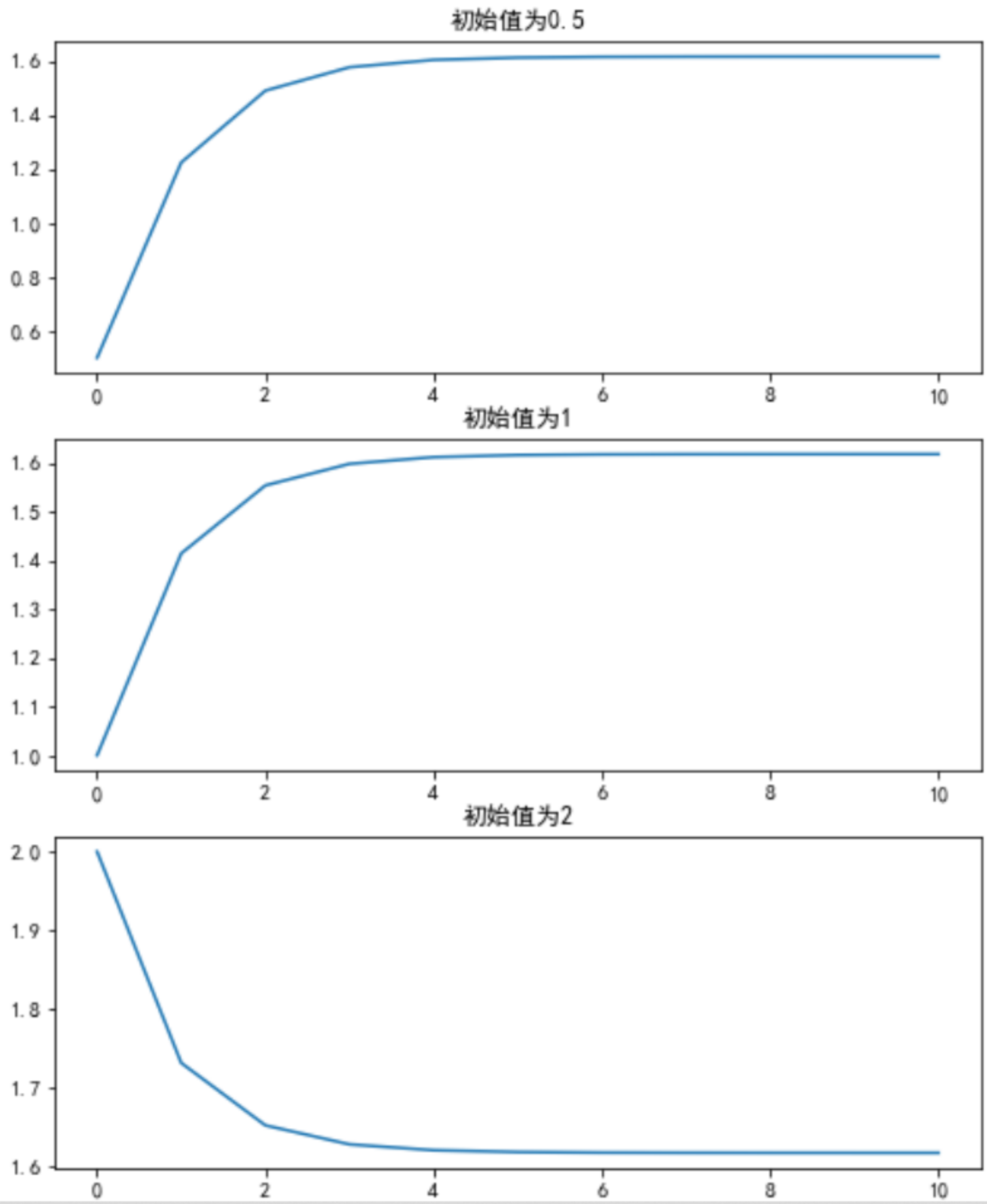
```
plot_img([0.5,1,2],f1)
```



```
plot_img([0.5,1,2],f2)
```



```
plot_img([0.5,1,2],f3)
```



通过上述三组图像可以分析得出：
1.随迭代次数的增加，迭代函数 2、3 的迭代结果收敛性很好，且二者趋于收敛为同一数值；而迭代函数 1 的迭代结果并不稳定，且在数值上与迭代函数 2、3 的收敛结果差距较大。
2.对于迭代函数 2、3，选取初值 1.0 所得的迭代结果，比初值 0.5 和初值 2.0 的收敛性更好。
因此，综合考虑而言，迭代函数 2、3 在初值选取为 1.0 时的迭代结果最佳。

牛顿法：
代码及结果：

T2.3

```
import math
def sqrt_newton(num):
    x=math.sqrt(num)
    y=num/2.0
    count=1
    while abs(y-x)>0.00000001:
        print(f"迭代{count}次，值：{y}")
        count+=1
        y=((y*1.0)+(1.0*num)/y)/2.0000
    return y

print("牛顿法结果：",sqrt_newton(5))
print("实际结果：",math.sqrt(5))
```

```
迭代1次，值：2.5
迭代2次，值：2.25
迭代3次，值：2.236111111111111
牛顿法结果： 2.2360679779158037
实际结果： 2.23606797749979
```

当迭代次数大于 4 次时，迭代结果趋于稳定，收敛性良好

教师评语	指导教师：　　　　年　月　日