实验报告二

专业____20 统计实验班______学号__2020111509__姓名___徐之皓____

一、实验目的

该实验目的是：

1、 利用二分法、简单迭代、埃特金加速和牛顿法迭代求解非线性方程

2、 求出所有的根，并比较各种方法的优劣（收敛速度/迭代速度，是否能求出所有解等）。

实验的注意事项

1、 尽量以最简练的语言完成计算和迭代

2、 实验应减少不必要的迭代次数（事先预估解的大致存在区间）

3、 实验应避免大数值溢出和数值错误

二、实验题目

实验二 非线性方程求解实验

实验题目：求方程 $f(x)=x^3-\cos x-5x-1=0$ 的全部根。

方案一 用二分法求解

方案二 用简单迭代求解

方案三 用埃特金迭代加速法求解

方案四 用牛顿法求解

通过四种方案分别求出方程的解并且比较各方法的收敛速度。

三、实验原理

二分法：

二分法是一种基于区间缩小的迭代方法，它的基本思想是将待求解的区间不断缩小，直到找到方程的根。具体来说，二分法的步骤如下：

1. 选择一个初始区间[a,b]，使得 f(a)和 f(b)异号，即 f(a)×f(b)<0 。

2. 将区间[a,b]平分为两个子区间，即 c=(a+b)/2。

3. 判断 f(c)与 0 的关系，如果 f(c)等于 0，则 c 就是方程的根；否则，根据 f(c)与 f(a)或 f(b)的符号关系，确定新的区间。

不动点迭代法：

将方程 f(x)=0 改成 x=φ(x)

要求 x* 满足 f(x*)=0，则 x*=φ(x*)

称 x* 为函数 φ(x) 的一个不动点，求f(x)的零点等价于求 φ(x) 的不动点，选择一个初始近似值 x_0，将它代入 x=φ(x) 式右端可以求得：

$x_1 = \varphi(x_0)$

如此反复迭代计算：

$x_{k+1} = \varphi(x_k)$ k = 0, 1, ... (2.2)

φ(x)称为迭代函数。

如果对任何 x_0 ∈[a, b]，由 $x_{k+1} = \varphi(x_k)$ 得到的序列{ x_k }有极限

$\lim_{k\to\infty} x_k = \varphi(x^*)$

则称迭代方程 $x_{k+1} = \varphi(x_k)$ 收敛，且x*=φ(x*)为φ(x)的不动点。称 $x_{k+1} = \varphi(x_k)$ 为不动点迭代法。

埃特金加速迭代：

假设 $\varphi(x)$ 在 α 处可导，有

$$\begin{aligned} x_{k+1}-\alpha=\varphi'(\xi_1)(x_k-\alpha) \\ x_{k+2}-\alpha=\varphi'(\xi_2)(x_{k+1}-\alpha) \end{aligned}$$

假设$\varphi'(\xi_1)\approx\varphi'(\xi_2)$,有

$$\frac{x_{k+1}-\alpha}{x_{k+2}-\alpha}=\frac{x_k-\alpha}{x_{k+1}-\alpha}$$

得到

$$\alpha=x_k-\frac{(x_{k+1}-x_k)^2}{x_{k+2}-2x_{k+1}+x_k}$$

记

$$\hat{x}_k=x_k-\frac{(x_{k+1}-x_k)^2}{x_{k+2}-2x_{k+1}+x_k}\quad k=1,2,3,\ldots$$

以上即为Aitken加速算法的迭代格式，序列 $\{\hat{x}_k\}$ 要比序列 $\{x_k\}$ 更快地收敛于 α 。

为方便计算可构造如下Aitken加速算法

$$\begin{cases} y_k=\varphi(x_k) \\ z_k=\varphi(x_k) \\ x_{k+1}=x_k-\frac{(y_k-x_k)^2}{z_k-2y_k+x_k}, \quad k=0,1,2,\cdots \end{cases}$$

牛顿法：

设函数 $f(x)=0$ 在有根区间 $[a,b]$ 上二阶连续可微，x_0 是 α 的近似值，将 $f(x)$ 在 x_0 处作Taylor展开，有

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+f''(\xi_0)\frac{(x-x_0)^2}{2}$$

用其线性主部近似 $f(x)$

$$f(x)\approx f(x_0)+f'(x_0)(x-x_0)$$

将非线性方程 $f(x)=0$ 近似化为线性方程

$$f(x_0)+f'(x_0)(x-x_0)=0$$

若 $f'(x_0)$ 不为0，则有

$$x_1=x_0-\frac{f(x_0)}{f'(x_0)}$$

一般地，有

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$$

四、实验内容

实验软件和语言：利用 Python 语言和集成 Jupyter Notebook 编辑器。
实验步骤与方案（自制）：

实验二：求非线性方程

(1) 二分法

give . a, b, h, eps, iter, f

$x_0 = \frac{a+b}{2}$

- $f(a)f(x_0) > 0 \longrightarrow x_1 = \frac{x_0+b}{2}$ ……
- $f(a)f(x_0) < 0 \longrightarrow x_1 = \frac{a+x_0}{2}$

if $f(x_i) <$ eps ; print x_i

(2) 不动点迭代

give a.b.h. eps. iter.

: $x = (5x + \cos x + 1)^{\frac{1}{3}}$, x_0

即 $\varphi(x) = 5x + \cos x + 1$

for x_0 in range $(-5, 5, 1)$

$x_1 = \varphi(x_0)$

NO

if $\varphi'(x_0) < 1$

Yes

$i = 1, 2$ …… 迭代次数 $i < 50$

$x_k = \varphi(k-1)$

if $x_k - x_{k-1} <$ eps

print x_k 迭代值

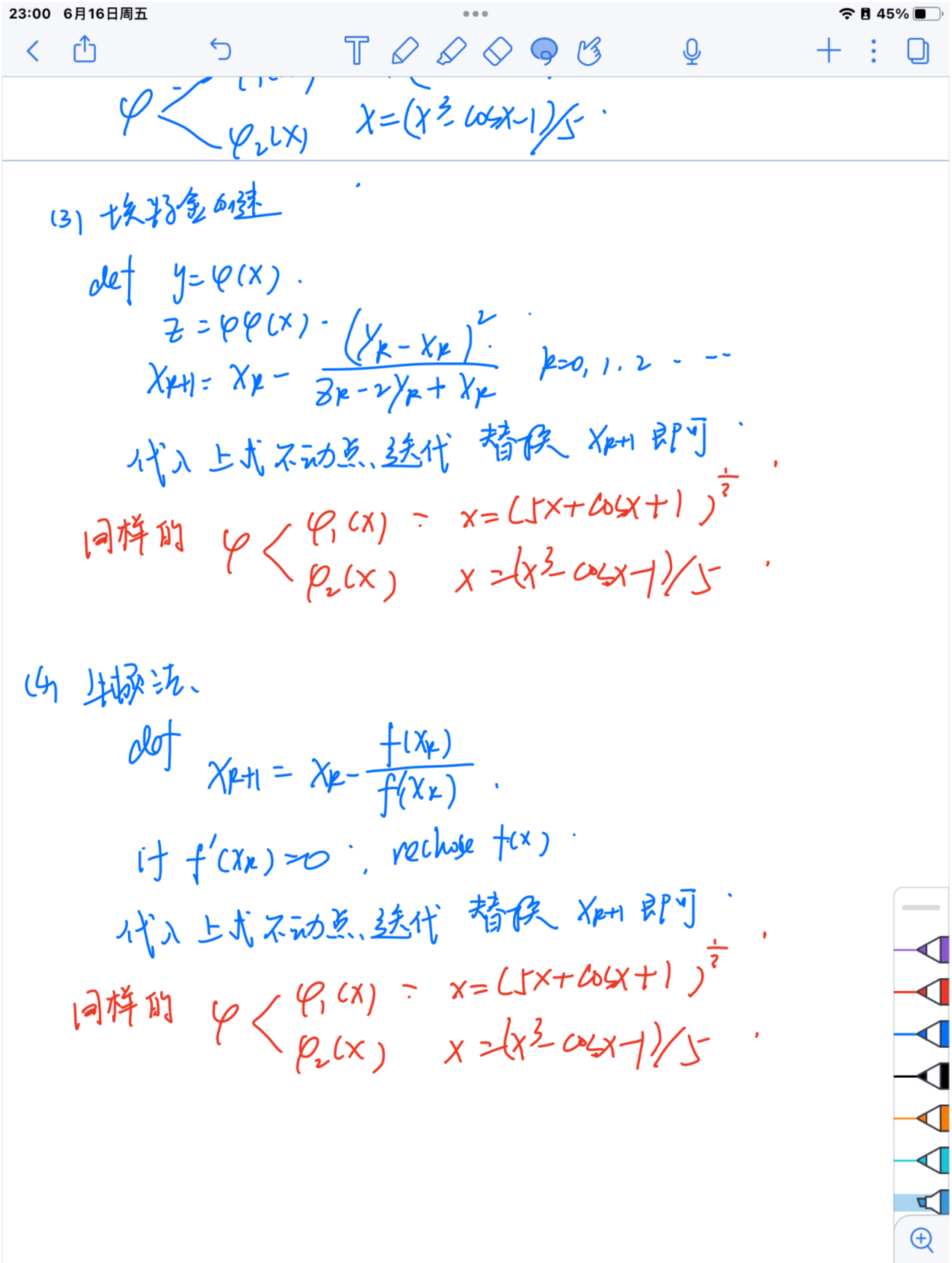
else

$x_{k-1} = x_k$

$i \geq 50$

else 方法失败

$\varphi_1(x)$: $x = (5x + \cos x + 1)^{\frac{1}{3}}$



$\varphi \begin{cases} \varphi_1(x) \\ \varphi_2(x) \quad x=(x^3-\cos x-1)/5 . \end{cases}$

(3) 埃特金加速

def $y=\varphi(x)$.

$z=\varphi\varphi(x)$. $x_{k+1}=x_k-\dfrac{(y_k-x_k)^2}{z_k-2y_k+x_k}$ $k=0,1,2 \cdots$

代入上式不动点迭代 替换 x_{k+1} 即可.

同样的 $\varphi \begin{cases} \varphi_1(x) : x=(5x+\cos x+1)^{\frac{1}{3}} \\ \varphi_2(x) \quad x=(x^3-\cos x-1)/5 . \end{cases}$

(4) 牛顿法.

def $x_{k+1}=x_k-\dfrac{f(x_k)}{f'(x_k)}$.

if $f'(x_k)=0$: rechose $f(x)$.

代入上式不动点迭代 替换 x_{k+1} 即可.

同样的 $\varphi \begin{cases} \varphi_1(x) : x=(5x+\cos x+1)^{\frac{1}{3}} \\ \varphi_2(x) \quad x=(x^3-\cos x-1)/5 . \end{cases}$

具体实验内容详见附录

五、实验结果

（1）二分法

```
-5和-4之间没有根
-4和-3之间没有根
-2.1931327991187572 是用二分法求解方程的根
迭代次数为 28
-2和-1之间没有根
-0.39695845916867256 是用二分法求解方程的根
迭代次数为 28
0和1之间没有根
1和2之间没有根
2.2708289455622435 是用二分法求解方程的根
迭代次数为 29
3和4之间没有根
4和5之间没有根
```

图 1： 二分法求解

（2）不动点迭代

1、当迭代函数为 x = (5*x + np.cos(x) +1)**(1/3) 时，只能迭代出 2.270828 这个根。
2、当迭代函数为 x = (x**3-math.cos(x) - 1)/5 时，只能迭代出-0.396958 这个根。

（3）埃特金加速迭代

1、当迭代函数为 x = (5*x + np.cos(x) +1)**(1/3) 时 Aitken 只能迭代出 2.270828 •• 这个根。

2、而当迭代函数为 x =(x**3-math.cos(x) - 1)/5 时 就能迭代出所有根。

3、迭代速度比普通不动点迭代快。

（4）牛顿法

1、牛顿法可以解出所有的根，且收敛速度较快（若收敛）。

##具体迭代结果见附录

六、实验结果分析

1、只要规定误差 eps（统一设为 10**(-8)），二分法 100%可以求出所有解，但是其缺点是迭代速度慢，计算量大。
2、不动点迭代法有收敛条件，所以并不是每一个迭代函数 phi(x)都能收敛。由于不动点迭代每一个初值最多收敛到一个 x，故须多次尝试初值。若导数为零则至少二次收敛，若导数非零则以导数绝对值为收敛常数线性收敛。对于一个非线性方程，必须先找到它对应的一个收敛的不动点迭代，有可能需要多次

构造。
3、埃特金加速迭代法收敛速度较快（相比普通不动点），但同时寻找一个合适的初值和迭代函数是关键，这会导致迭代结果的完全不同。
4、牛顿法的收敛速度快，在范围内尝试可以解出所有根，而且通常情况下是二阶收敛。但由于需要求导数，时间代价比较大。
5、综上所述，若抛开其余客观因素，选择牛顿法或埃特金加速法解决非线性方程是值得推荐的。

七、附录：

##由于使用 notebook 无法，笔者提供一个 github 地址，里面存放有项目相关的 notebook 供查阅，另附件里也有，烦请查看，谢谢。

##github 地址：rqkgHH/Numerical_analysis (github.com)

```
#!/usr/bin/env python
# coding: utf-8

# ![%E5%9B%BE%E7%89%87.png](attachment:%E5%9B%BE%E7%89%87.png)

# In[184]:


import math
import numpy as np

def y(x):
    return (x**3-np.cos(x)-5*x - 1)
math.cos(math.pi/3) ##cos 是弧度制
np.cos(math.pi/3)


# In[278]:


y(4)


# ## 二分法

# In[235]:


## 粗略判断 根只有可能在（-5,5）之间
```

```
def iabs(x):
        if x>0:
            return x
        else:
            return -x
for a in range(-5,5):

    b = a + 1
    fa=y(a)
    fb=y(b)
    eps=10**(-8)
    iter=0

    while a<=b:

        iter +=1
        x0=(a+b)/2
        f = y(x0)
        if iabs(f) < eps:
            print(x0,'是用二分法求解方程的根')
            print('迭代次数为',iter)
            break
        if fa*f<0:
            b=x0
            fb=f
        elif fb*f<0:
            a=x0
            fa=f
        if iter>100:
            print(f"{a}和{b}之间没有根")
            break


# ## 简单迭代求解

# In[265]:


import numpy as np
import math

def phi(x):
    return (5*x + np.cos(x) +1)**(1/3)
def dphi(x):
```

```
    return (1/3)*(5*x + np.cos(x) +1)**(-2/3)*(5 - np.sin(x))

def diedai(x0,epsilon,iternum,phi,dphi):# 初值，精度要求，最大迭代次
数,迭代函数,迭代函数导数收敛
    xk_1 = x0
    for i in range(iternum):
        y = phi1(xk_1)
        if (dphi(y))<1:
            xk = phi(xk_1)
            print("第",i+1,"次迭代: ","xk=",xk,"  xk-1=",xk_1,"  差值
为",abs(xk-xk_1));
            if abs(xk-xk_1)<epsilon:
                return xk
            else:
                xk_1 = xk
        else:
            print("方法失败")
            break
    return 0
for x1 in range(-10,10):
    diedai(x1,10**(-8),20,phi,dphi)
    print(f'当初值为{x1}时，迭代结果如上')
    print()

# 当迭代函数为 x = (5*x + np.cos(x) +1)**(1/3) 时 普通迭代能迭代出
2.270828 •• 这个根


# In[264]:



import numpy as np
import math
########新迭代函数#########
phi1= lambda x: (x**3-math.cos(x) - 1)/5
dphi1= lambda x: (3*x**2+math.sin(x))/5

def diedai(x0,epsilon,iternum,phi,dphi):# 初值，精度要求，最大迭代次
数,迭代函数,迭代函数导数收敛
    xk_1 = x0
    for i in range(iternum):
        y = phi1(xk_1)
        if (dphi1(y))<1:
            xk = phi1(xk_1)
```

```
            print("第",i+1,"次迭代 ","xk=",xk,"  xk-1=",xk_1,"  差值为",abs(xk-xk_1));
            if abs(xk-xk_1)<epsilon:
                return xk
            else:
                xk_1 = xk
        else:
            print("方法失败")
            break
    return 0
for x1 in range(-10,10):
    diedai(x1,10**(-8),20,phi1,dphi1)
    print(f'当初值为{x1}时，迭代结果如上')
    print()

## 当迭代函数为 x =(x**3-math.cos(x) - 1)/5  时 只能迭代出-0.396958··这个根


# ## 埃特金加速算法

# In[260]:


def Aitken(x0,epsilon,iternum,phi):#初值，精度要求，最大迭代次数,迭代函数
    xk_1 = x0
    for i in range(iternum):
        y = phi(xk_1)
        z = phi(y)
        if (z - 2*y +xk_1)!= 0:
            xk = xk_1 - (y - xk_1)**2 / (z - 2*y +xk_1)
            print("第",i+1,"次迭代 ","xk=",xk,"  xk-1=",xk_1," 差值为",abs(xk-xk_1))
            if abs(xk-xk_1)<epsilon:
                return xk
            else:
                xk_1 = xk
        else:
            return x
    print("方法失败")
    return 0
for x1 in range(-10,10):
    Aitken(x1,10**(-8),20,phi)
```

```
    print(f'当初值为{x1}时，迭代结果如上')
    print()

##当迭代函数为 x = (5*x + np.cos(x) +1)**(1/3) 时 Aitken只能迭代出
2.270828••这个根


# In[267]:


def Aitken(x0,epsilon,iternum,phi):#初值，精度要求，最大迭代次数,迭代
函数
    xk_1 = x0
    for i in range(iternum):
        y = phi(xk_1)
        z = phi(y)
        if (z - 2*y + xk_1)!= 0:
            xk = xk_1 - (y - xk_1)**2 / (z - 2*y +xk_1)
            print("第",i+1,"次迭代 ","xk=",xk,"  xk-1=",xk_1," 差值为
",abs(xk-xk_1))
            if abs(xk-xk_1)<epsilon:
                return xk
            else:
                xk_1 = xk
        else:
            return x
    print("方法失败")
    return 0
for x1 in range(-10,10):
    Aitken(x1,10**(-8),50,phi1)
    print(f'当初值为{x1}时，迭代结果如上')
    print()

##  而当迭代函数为 x =(x**3-math.cos(x) - 1)/5 ## 时 就能迭代出所有根


# In[ ]:





# ## 牛顿法
```

```
# In[258]:


def f(x):
    return x**3-np.cos(x)-5*x - 1
def df(x):
    return 3*x**2+np.sin(x)-5

def Newton(x0,epsilon,iternum,phi):#初值，精度要求，最大迭代次数,迭代
函数
    xk_1 = x0
    for i in range(iternum):
        y = f(xk_1)
        z = df(xk_1)
        if z != 0:
            xk = xk_1 - y / z
            print("第",i+1,"次迭代: ","xk=",xk,"  xk-1=",xk_1," 差值
为",abs(xk-xk_1))
            if abs(xk-xk_1)<epsilon:
                return xk
            else:
                xk_1 = xk
        else:
            return x
            print("方法失败")
    return 0
for x1 in range(-10,10):
    Newton(x1,10**(-8),50,phi)
    print(f'当初值为{x1}时，迭代结果如上')
    print()

##从结果上看，用牛顿法可以解出所有的根
```