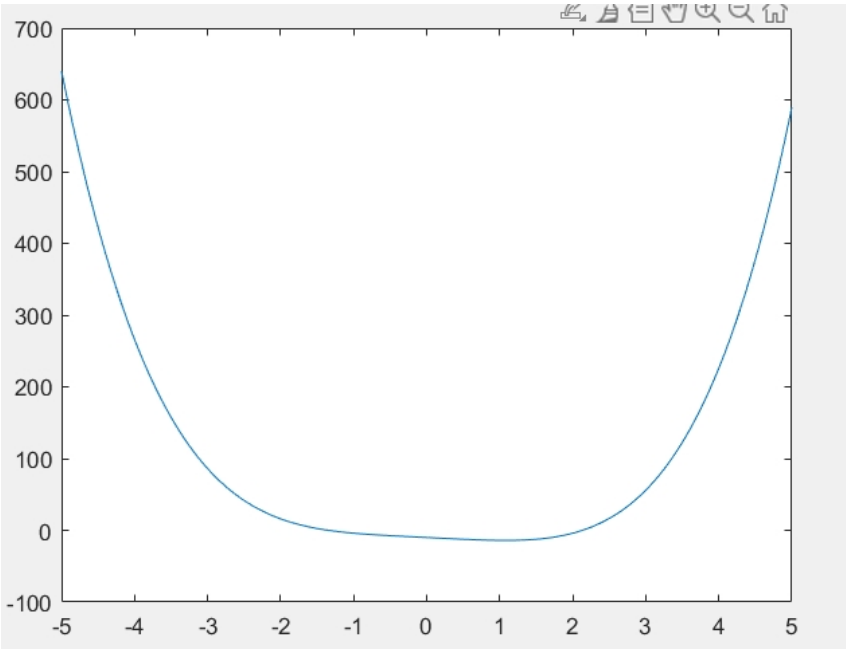
## 1 问题描述

编写非线性方程求根的不动点迭代算法程序
从在0<x<7不同初始点开始，分别采用以下四种迭代方式求解方程
$f(x)=x^4-5x-10=0$的正根，记录迭代过程并分析原因
(1)$g(x)=\frac{x^4-10}{5}$ (2)$g(x)=(10+5x)^{\frac{1}{4}}$ (3)$g(x)=(\frac{10+5x}{x})^{\frac{1}{3}}$ (4)
$g(x)=\sqrt{\frac{10+5x}{x^2}}$

## 2 问题分析

首先分析f(x)根的情况，用matlab画图有：

```
figure;
x=-5:0.01:5;
f=@(x)(power(x,4)-5*x-10);
y_value=f(x);
plot(x,y_value);
```




共有两个根，一个负根一个实根。

## 3 代码实现

不动点迭代法

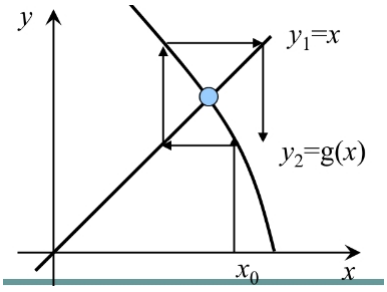
```
function [k,p,err,P]=fixpt(g,p0,tol,max1)
P(1)=p0;
for k=2:max1
    P(k)=feval(g,P(k-1));
    err=abs(P(k)-P(k-1));
    relerr=err/abs(P(k)+eps);
    p=P(k);
    if (err<tol)||(relerr<tol),break;end
end
end
```

输入-g是迭代函数，p0是不动点初始值，tol是误差容限，max1是最大迭代次数
输出-k是当前迭代的次数，err是最终误差上界，p是不动点最终结果，P是p迭代的序列

选择2，4，6作为初始值，分别用四种方法来进行迭代。
为了方便输出，定义一个输出函数

```
function show(k,p,err,P)
disp(['迭代次数: ', num2str(k)]);
fprintf("值%.16f\n",p);
fprintf("误差上界%.28f\n",err);
fprintf("\n");
for i=1:k
    fprintf("%.16f\n",P(i));
end
fprintf("\n");
end
```

我们已知不动点迭代法可能会出现四种情况，分别为单调收敛、振荡收敛、单调发散、振荡发散。由不动点迭代法的定义，同时方便我们观察其收敛过程，可以将收敛过程中各个点与函数y=x，y=g(x)画成图象。参考下图，我们可以定义一个绘图函数




输入P为迭代得到的不动点序列，g是迭代函数，通过分析图象，我们可以发现迭代过程中，除去不动点初始值，其余的迭代值每个都充当了两次x值参与绘图，其对应的y值依次是y1=x与y2=g(x)，然后依次循环，又因为y=x函数的特殊性后一组的y1等于前一组的y2，有奇偶变换的规律，因此可以新建两个序列来绘图表示迭代过程。

```
function P_value=cycle(P,g)
Pnew=P(1);
for i=2:length(P)
    Pnew=[Pnew,P(i),P(i)];
```

```
end  %建立除了第一个值，其余值都重复的序列
P_value(1)=feval(g,Pnew(1));
for i=2:length(Pnew)
    if mod(i,2)==0
        P_value(i)=P_value(i-1);
    end
    if mod(i,2)==1
        P_value(i)=feval(g,P_value(i-1));
    end
end    %建立对应的y值序列
    plot(Pnew,P_value,"-o");
    hold on;
    x=[-2:0.01:-0.01,0.01:0.01:2.5];
    %注意应该挖去x=0，因为g3、g4都涉及分母等于零，同时为保证g2与g4被开方数为正，
x>-2
    g_value=g(x);
    tool=@(x)(x);
    tool_value=tool(x);
    plot(x,g_value);
    plot(x,tool_value);
    hold off
end
```

# 4 结果分析

## 4.1 确定初始值

```
f=@(x)(power(x,4)-5*x-10);
g1=@(x)((power(x,4)-10)/5);
g2=@(x)(power((10+5*x),0.25));
g3=@(x)(nthroot((10+5*x)./x,3));
g4=@(x)(sqrt((10+5*x)./(x.^2)));
p0=2;%4 or 6
p0=double(p0);
```

其中g3选择nthroot而不是power是为了防止对负根求三次根号时出现复数根增根，只取实数根。p0选用双精度

## 4.2 求解与画图

```
[k1,p1,err1,P1]=fixpt(g1,p0,eps,1000);
[k2,p2,err2,P2]=fixpt(g2,p0,eps,1000);
[k3,p3,err3,P3]=fixpt(g3,p0,eps,1000);
[k4,p4,err4,P4]=fixpt(g4,p0,eps,1000);

figure;%将迭代序列用图象表示
subplot(4,1,1);
x1=1:length(P1);
plot(x1,P1,"-o");
```

```
subplot(4,1,2);
x2=1:length(P2);
plot(x2,P2,"-o");
subplot(4,1,3);
x3=1:length(P3);
plot(x3,P3,"-o");
subplot(4,1,4);
x4=1:length(P4);
plot(x4,P4,"-o");

figure%将具体的收敛过程用图象表示
cycle(P1,g1);
figure
cycle(P2,g2);
figure
cycle(P3,g3);
figure
cycle(P4,g4);

show(k1,p1,err1,P1);
show(k2,p2,err2,P2);
show(k3,p3,err3,P3);
show(k4,p4,err4,P4);
```

**初始值为2时**

(自左而右g1,g2,g3,g4)

```
迭代次数: 1000                          迭代次数: 18
值-0.7368529308954301                   值2.1319739552620884
误差上界0.84842706980663207261272873344 误差上界0.00000000000000044440892098501

迭代次数: 21                            迭代次数: 113
值2.1319739552620884                    值2.1319739552620884
误差上界0.00000000000000044440892098501 误差上界0.00000000000000044440892098501
```

迭代序列依次为

g1 g2 g3 g4

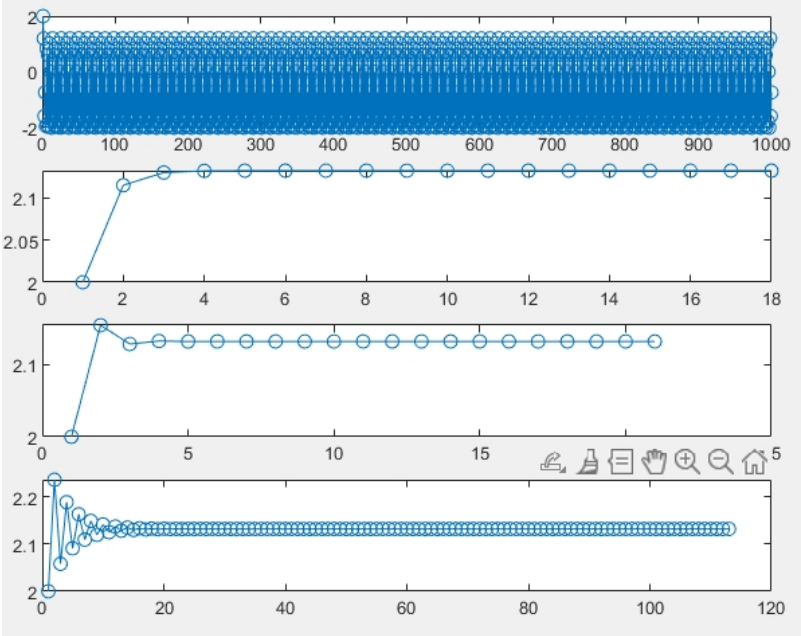
```
2.0000000000000000
1.2000000000000000
-1.5852800000000000
-0.7368529331330356
-1.9410405719153936
0.8390199760817598
-1.9008896066799097
0.6113048791892531
-1.9720706103946277
1.0249614139308716
-1.7792706623788157
0.0044631072546146
-1.9999999999206441
1.1999999994921224
-1.5852800007020900
-0.7368529308953417
-1.9410405726315916
0.8390199802718843
-1.9008896047000525
```

```
2.0000000000000000
2.1147425268811282
2.1297477464238703
2.1316867330684097
2.1319369047986942
2.1319691760155677
2.1319733387750515
2.1319738757399040
2.1319739450043262
2.1319739539389144
2.1319739550914090
2.1319739552400723
2.1319739552592485
2.1319739552617221
2.1319739552620414
2.1319739552620827
2.1319739552620880
2.1319739552620884
```

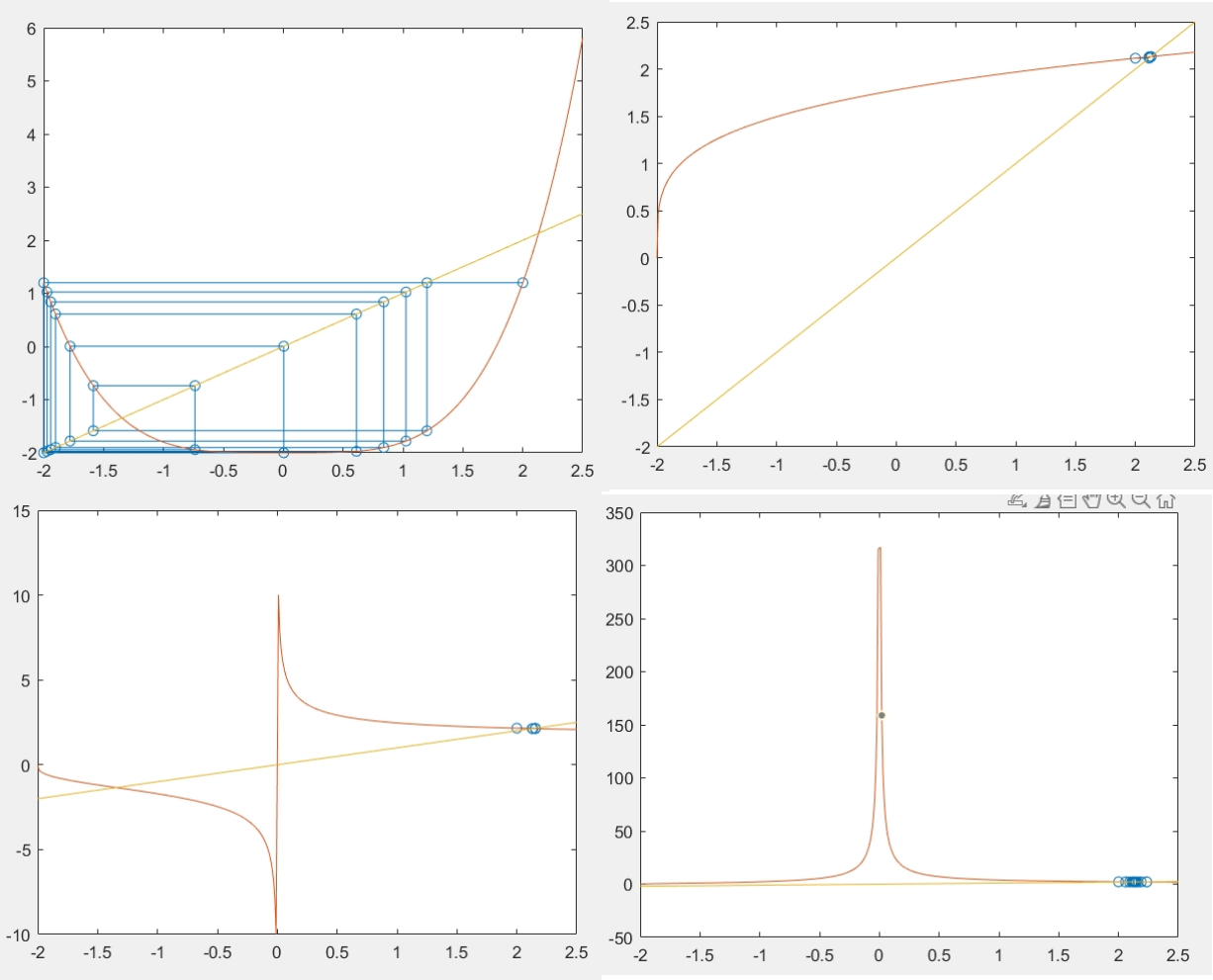
```
2.0000000000000000
2.1544346900318838
2.1283817955791058
2.1325543466179000
2.1318803343302273
2.1319890609364958
2.1319715180759218
2.1319743484863327
2.1319738918179678
2.1319739654983780
2.1319739536105309
2.1319739555285566
2.1319739552190957
2.1319739552690251
2.1319739552609693
2.1319739552622692
2.1319739552620596
2.1319739552620933
2.1319739552620880
2.1319739552620889
```

```
2.0000000000000000
2.2360679774997898
2.0581710272714919
2.1886116909541369
2.0909870237333532
2.1629561523417298
2.1092992827800869
2.1489716361286346
2.1194567159633046
2.1413143253543576
2.1250721370684640
2.1371110946788363
2.1281708724778321
2.1348007166337197
2.1298791085537552
2.1335298233637854
2.1308202818184858
2.1328304410441139
```

画成图象



迭代过程依次为(g1,g2,g3,g4)

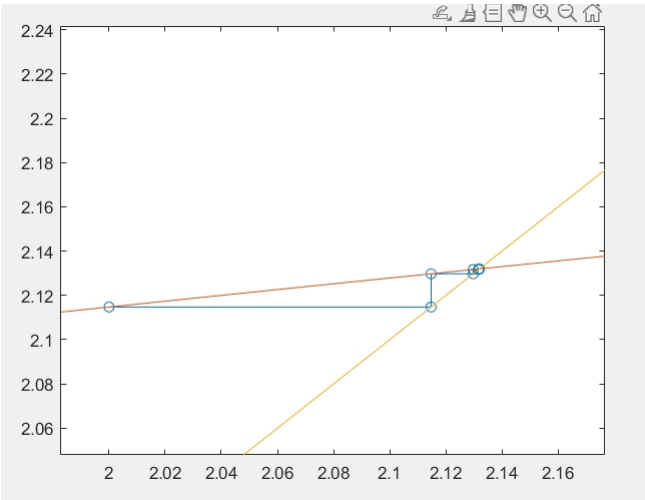


```
disp(f(p1));
disp(f(p2));
```

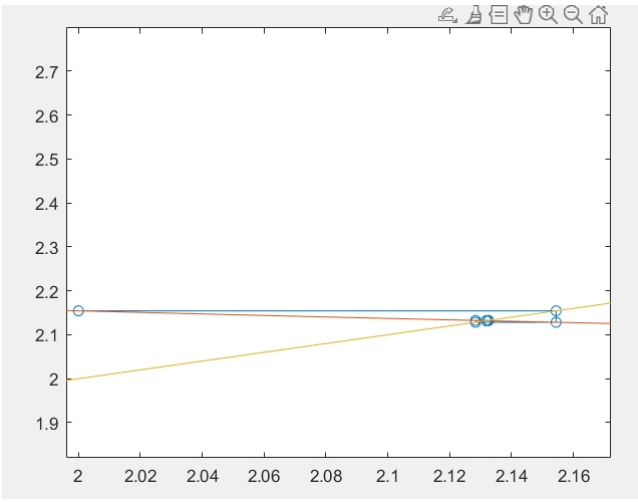
```
-6.0209

-5.3291e-15
```

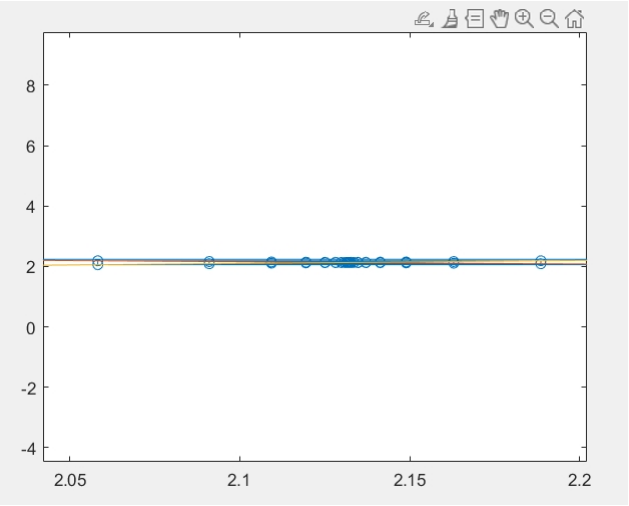
从这个结果中可以看出g1迭代次数最多，且无法收敛，最终值与准确值差别巨大，误差较大。其余三种迭代函数均收敛到了相同的且准确度较高的值。但是迭代次数有不同，次数g4>g3>g2。

通过迭代过程我们可以发现，g1出现了振荡发散，g2,g3,g4均收敛。
放大图象继续观察
g2是单调收敛




g3,g4是振荡收敛




其原因是迭代函数g1不收敛，$x->\infty$时g1(x)并不收敛，而g2,g3,g4均收敛，其收敛速度又影响了迭代次数，三个函数的指数分别为$\frac{1}{4}$，$\frac{1}{3}$，$\frac{1}{2}$，指数越小，收敛速度越快，迭代次数越小，因此会有g4>g3>g2。

### 初始值为4时

四种方法的结果依次为

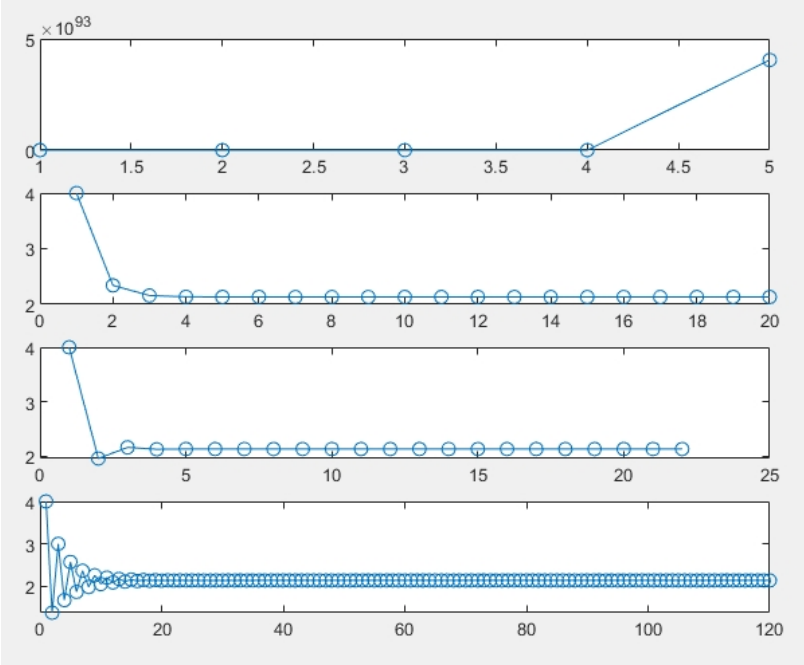
```
>> work1
迭代次数: 1000
值Inf
误差上界NaN
```

```
迭代次数: 20
值2.1319739552620884
误差上界0.000000000000000044408920985O1
```

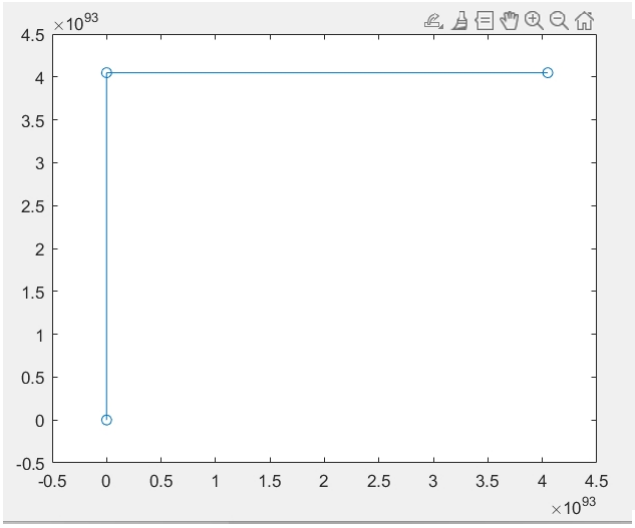
```
迭代次数: 22
值2.1319739552620884
误差上界0.000000000000000044408920985O1
```

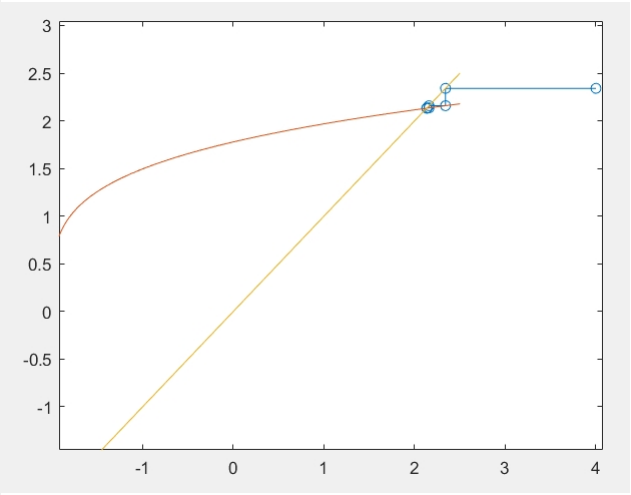
```
迭代次数: 120
值2.1319739552620884
误差上界0.000000000000000044408920985O1
```

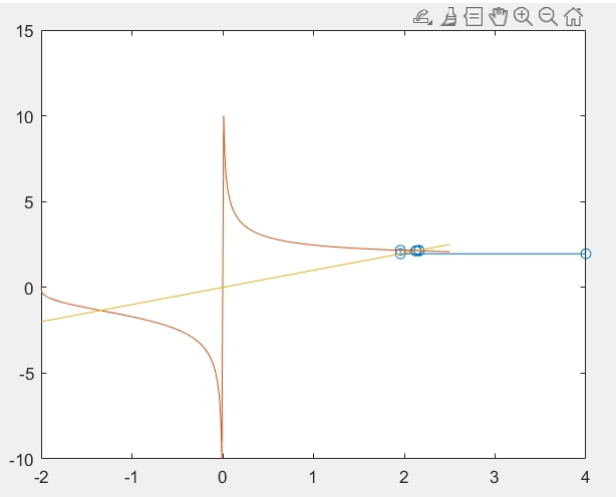
序列图象为

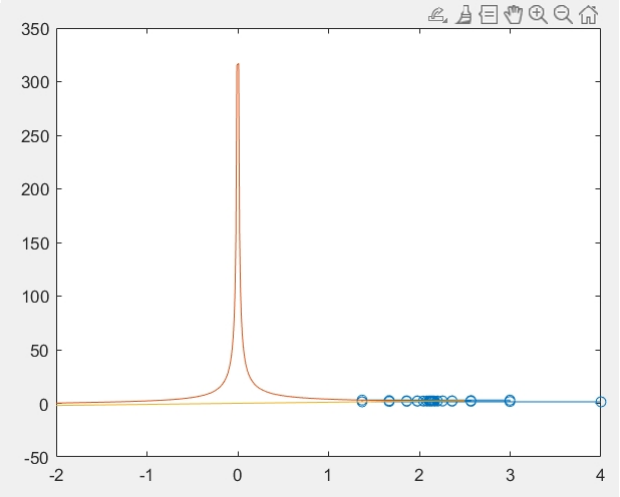



迭代过程为




初始值改为4后，发现g1输出为lnf，说明数极大，已经超越范围了，对g1的迭代序列分析，其单调发散，数越来越大，不会收敛。其余三种算法得到了与初始值为2相同的值，但是迭代次数均变多了。

```
4.0000000000000000
49.2000000000000028
1171897.6019200002774596
3772147426364469343682560.0000000000000000
40493435769218922178409221427166771699699837194159231313200556348322748156439123975721587834880.00000000(
Inf
Inf
Inf
Inf
```

## 初始值为6时

四种方法的结果依次为

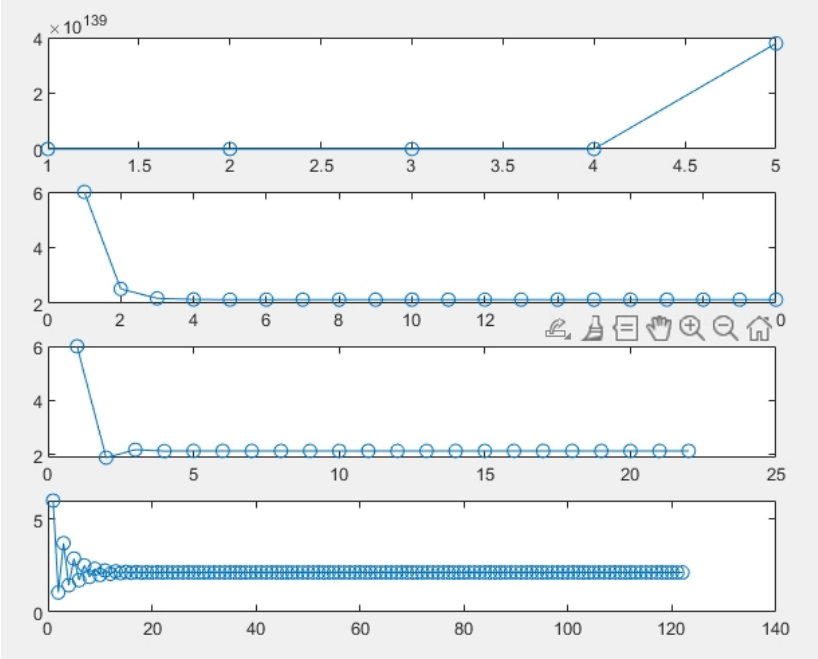
迭代次数：1000
值Inf
误差上界NaN

迭代次数：20
值2.1319739552620884
误差上界0.000000000000000044408920985O1

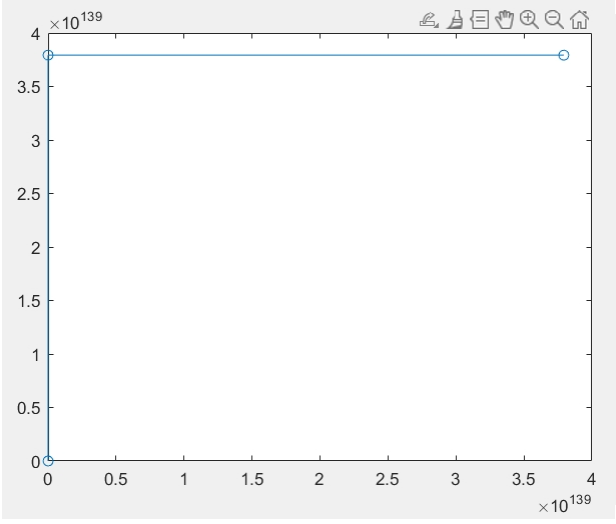
迭代次数：22
值2.1319739552620884
误差上界0.000000000000000044408920985O1

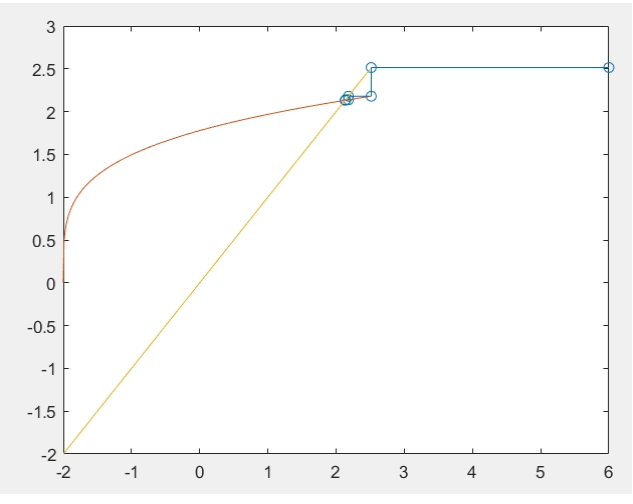
迭代次数：122
值2.1319739552620884
误差上界0.000000000000000044408920985O1

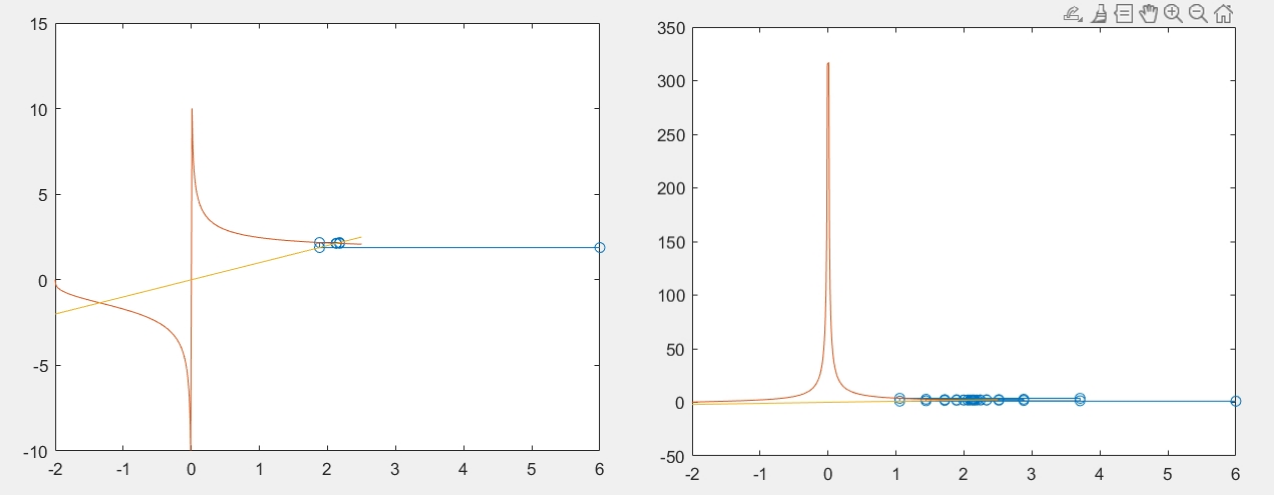
序列图象为




迭代过程为






情况与初始值为4时类似，g1单调发散至无穷大，g2,g3,g4均收敛至较准确的结果，其中g4的迭代次数略有增加。

### 4.3 分析

迭代函数g1不收敛，$x->\infty$时g1(x)并不收敛，而g2,g3,g4均收敛，因此选用g1时会出现振荡发散，单调发散的情况。

收敛速度影响了迭代次数，三个收敛函数的指数分别为$\frac{1}{4}$，$\frac{1}{3}$，$\frac{1}{2}$，指数越小，收敛速度越快，迭代次数越小，因此会有g4>g3>g2。对于收敛的迭代函数，初始值不影响最终结果，但是会影响迭代过程，初始值离真值越接近，迭代次数越少，因此可以用数据预处理估计出真值的大概区间，可以减少迭代次数，提高算法速度。

三个收敛函数均能得出相同的较为准确的值，说明在不考虑算法速度的情况下，只需保证迭代函数收敛便可得到理想精度的值。

## 5 个人心得

不动点迭代法的核心在于迭代函数的选择，迭代函数首先要收敛；然后要在不产生增根或者其他条件的前提下尽可能选择收敛速度快的函数以减少迭代次数；至于误差容限尽可能提高精度，不动点迭代法比较依赖误差容限，且不会出现由于始终达不到跳出函数的条件而无限循环迭代的情况；为提高算法速度，初始值也应设置的尽量离真值近，可以通过对原函数画图、或者按整数试根的方法进行预处理，找到真值位于哪两个相邻整数的区间内，其中任何一个整数都可作为较优合理的初始值。