第二次作业

李阳 5150272910019 黄弘毅 515072910038 王潇卫 515072910032

1

（a）

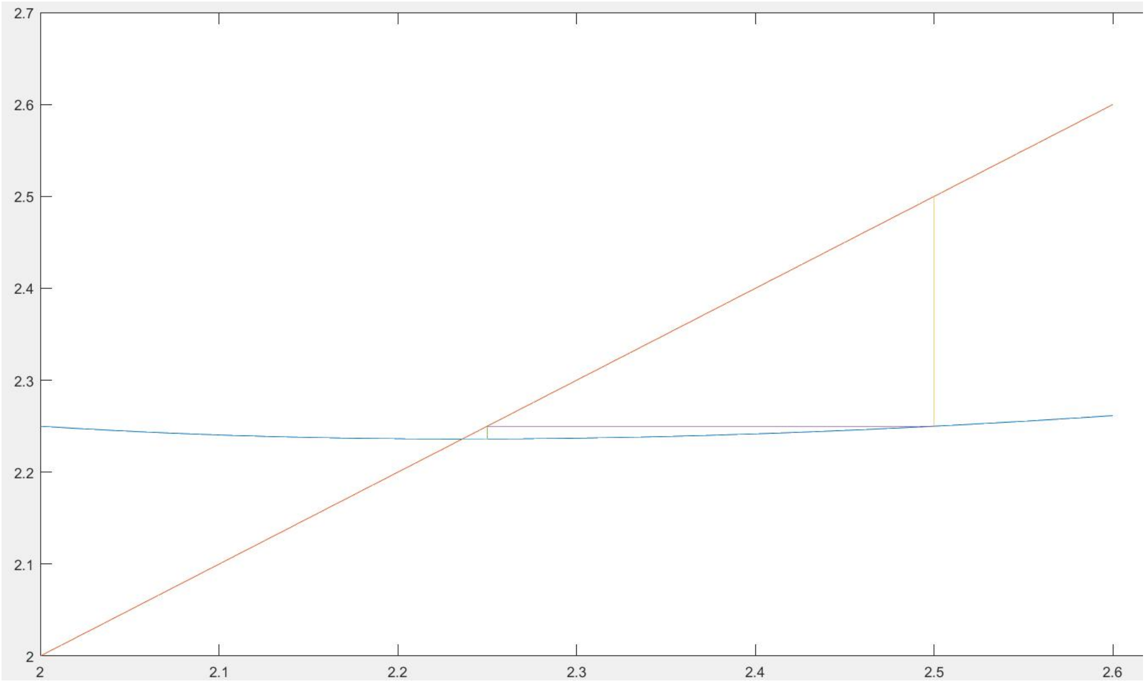
the solution of x=g(x)=$\frac{1}{2}$(x+$\frac{a}{x}$) is x*=$\sqrt{a}$ the first derivative:g'(x)=$\frac{1}{2}-\frac{a}{2{x}^2}$=0 the second derivative:g''(x)=$\frac{a}{{x}^3}=\frac{1}{\sqrt{a}}$ so the limit is $\lim_{n\to+\infty}\frac{x_{n+1}-x^*}{(x_n-x^*)^2}=\frac{g''(x^*)}{2!}\neq 0$ So the convergence is quadratic, and for any $x_1>0$, the iteration converges to $\sqrt{a}$

(b)

Code:Fixedpoint.m

```
%% to write a function of fixed point and output the convergence result
and graph.
%Author:Lyon(515072910019)
function Fixedpoint(x_initial,g)%函数形参：x_initial:初始点 g：函数表达式
N=100;eps=1e-8;%循环次数和结果精度
x_former=x_initial;
x=linspace(2,2.6,101);
% the interval of picture change with the function g when we use this
code.
y=g(x);
plot(x,y,x,x);% 首先画出两个主图：x和g(x)
hold on
for i=1:N
    x_latter=g(x_former);
    fprintf("N:%d \t xf:%.8f \t xl:%.8f \n",i,x_former,x_latter);
    plot([x_former,x_former],[x_former,x_latter]);
    hold on
    plot([x_former,x_latter],[x_latter,x_latter]);%画fixedpoint过程线
    hold on
    if abs(x_latter-x_former)<eps
        fprintf("the final result is %.8f \n",x_latter);
        break
    end
    x_former=x_latter;
end
end
```

```
>> Fixedpoint(2.5,g)
N:1       xf:2.50000000    x1:2.25000000
N:2       xf:2.25000000    x1:2.23611111
N:3       xf:2.23611111    x1:2.23606798
N:4       xf:2.23606798    x1:2.23606798
the final result is 2.23606798
```



(c)

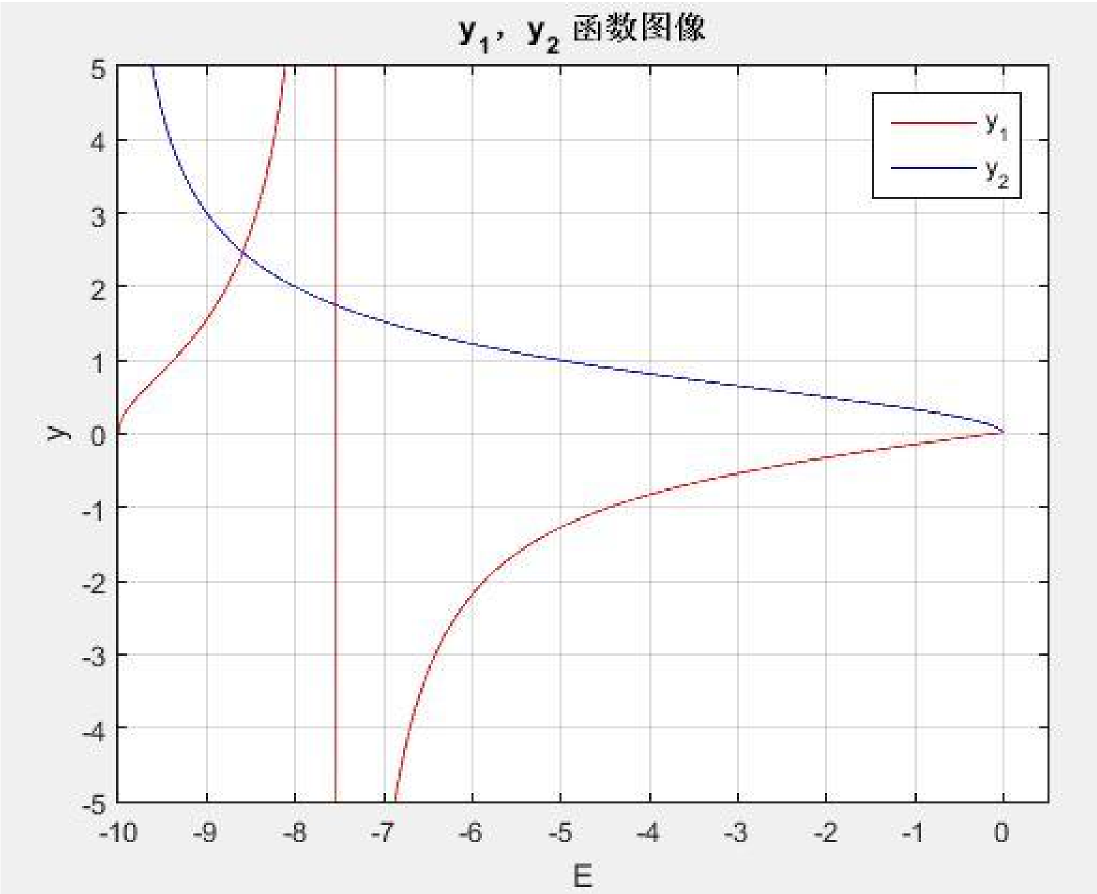
we use g(x)=$x^2 - 5$ to evaluate $\sqrt{5}$ by using Newton-Rafson method. code:New_Raf.m

```
clear all;close all;clc;
N=100;eps=1e-8;%循环次数和结果精度
x1=2.2;%接近2.5
g=@(x) x.^2-5;
g_prime=@(x) 2.*x;
for i=1:N
    x2=x1-g(x1)./g_prime(x1);
    fprintf("N:%d \t x1:%.8f \t x2:%.8f \n",i,x1,x2);
    if abs(x2-x1)<eps
        fprintf("the final result is %.8f \n",x2);
        break
    end
    x1=x2;
end
```

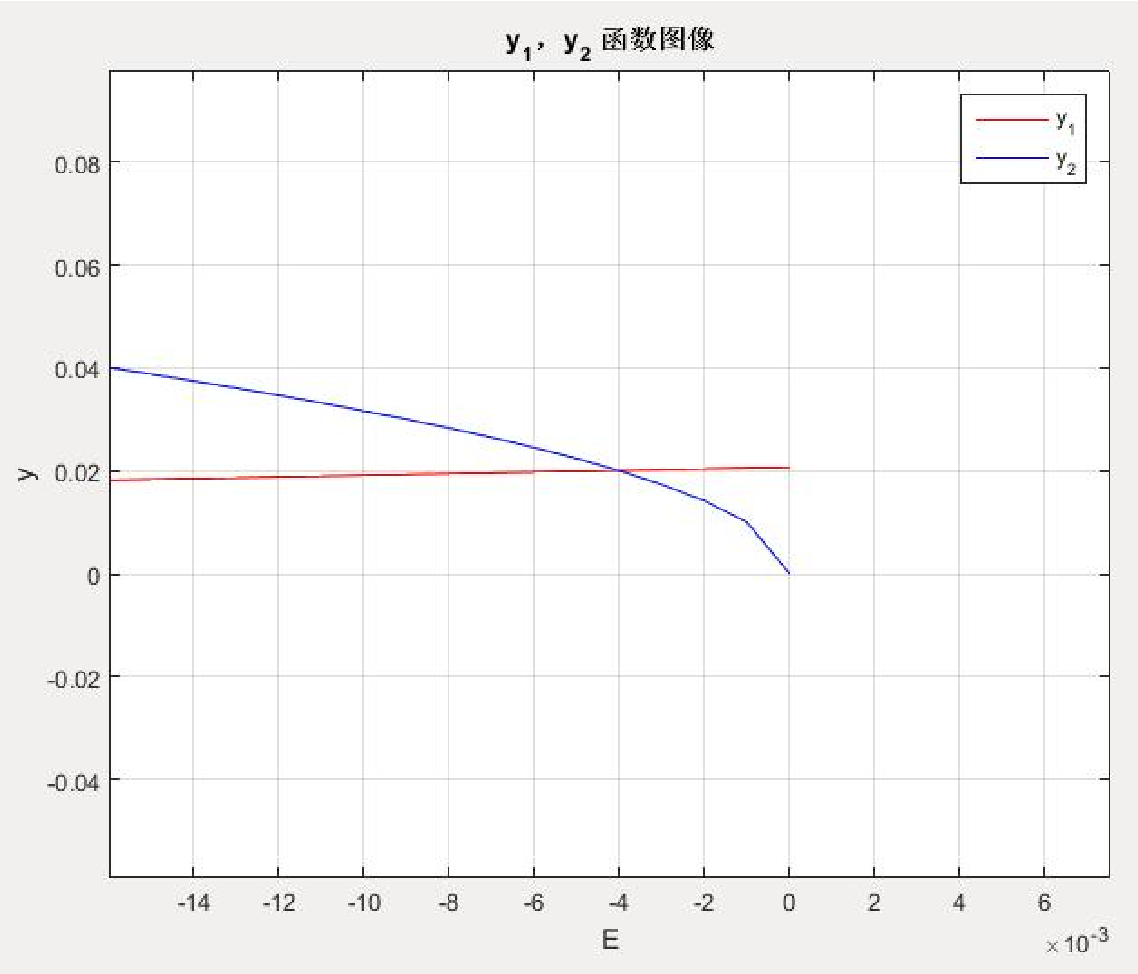
After running the code, we get the result:

```
N:1        x1:2.20000000    x2:2.23636364
N:2        x1:2.23636364    x2:2.23606800
N:3        x1:2.23606800    x2:2.23606798
N:4        x1:2.23606798    x2:2.23606798
the final result is 2.23606798
```

2 (a) For even wave function, we has the equation $\sqrt{10+E}\tan(\sqrt{10+E}) = \sqrt{-E}$,where $-10 < E < 0$ First, transform the equation into $\tan(\sqrt{10+E}) = \sqrt{\frac{-E}{10+E}}$,and plot $y_1 = \tan(\sqrt{10+E}), y_2 = \sqrt{\frac{-E}{10+E}}$ at $-10 < E < 0$ Then we have



Note that there is a singular point near $E = -7.5$, so at $-10 < E < -7$ there is only one point of intersection at $E \approx -8.6$. What's more, $y_1(0) < 0, y_2(0) = 0$ so there is another solution around 0. The figure shows that the soultion is -0.004.



Using fixed-point method to solve the equation. Transform the equation into $E = \arctan(\sqrt{\frac{-E}{10+E}})^2 - 10$

Setting initial guess $x_1 = -8.5$, then we have $x = -8.592791$

Fixed-point is: -8.592786 ,found in iteration 12 ,intial guess -8.500000

Here is the code for question (a).

```
%HW_2_2_a
%计算物理 第二次作业 第2题(a)
%王潇卫 515072910032

clc
clear all
%% First part: plot the  figure.
f1 = @(E) tan(sqrt(10+E));
f2 = @(E) sqrt(-E./(10+E));
E = -10:0.01:0;
y1 = f1(E);
y2 = f2(E);
```

```
axis([-10,0.5,-5,5])
grid on
xlabel('E')
ylabel('y')
title('y_1, y_2 函数图像')
legend ('y_1','y_2')

%% Second part: use fixed-point method to find the solution.
espon = 1e-5;
Iteration = 20;
E_intial = -8.5; %猜测值
E0 = E_intial;
E1 = 0;
f3 = @(E) (atan(sqrt(-E./(10+E)))).^2 -10;
for i=1:Iteration
    E1 = f3(E0);
    if abs(E0-E1) <= espon
        fprintf (1,'Fixed-point is: %f ,found in iteration %d ,intial
guess %f \n',E0,i,E_intial)
        break
    else
        E0 = E1;
    end
end
```

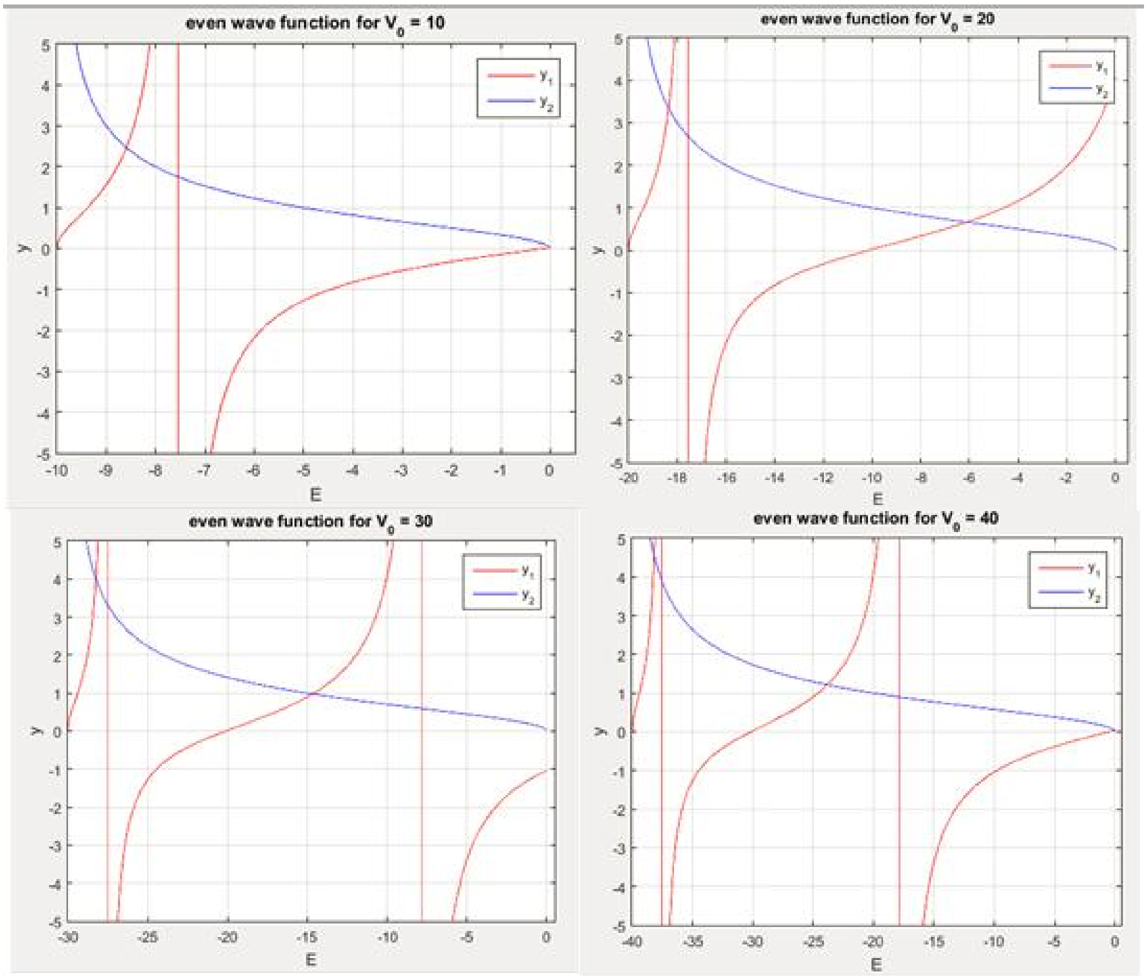
(b)The change of the depth of the potential causes equations turning into

$\sqrt{V_0+E}\tan(\sqrt{V_0+E})=\sqrt{-E}$, where $-V_0<E<0$ (even)
$\sqrt{V_0+E}\cot(\sqrt{V_0+E})=-\sqrt{-E}$, where $-V_0<E<0$ (odd)

Change the potential deeper :

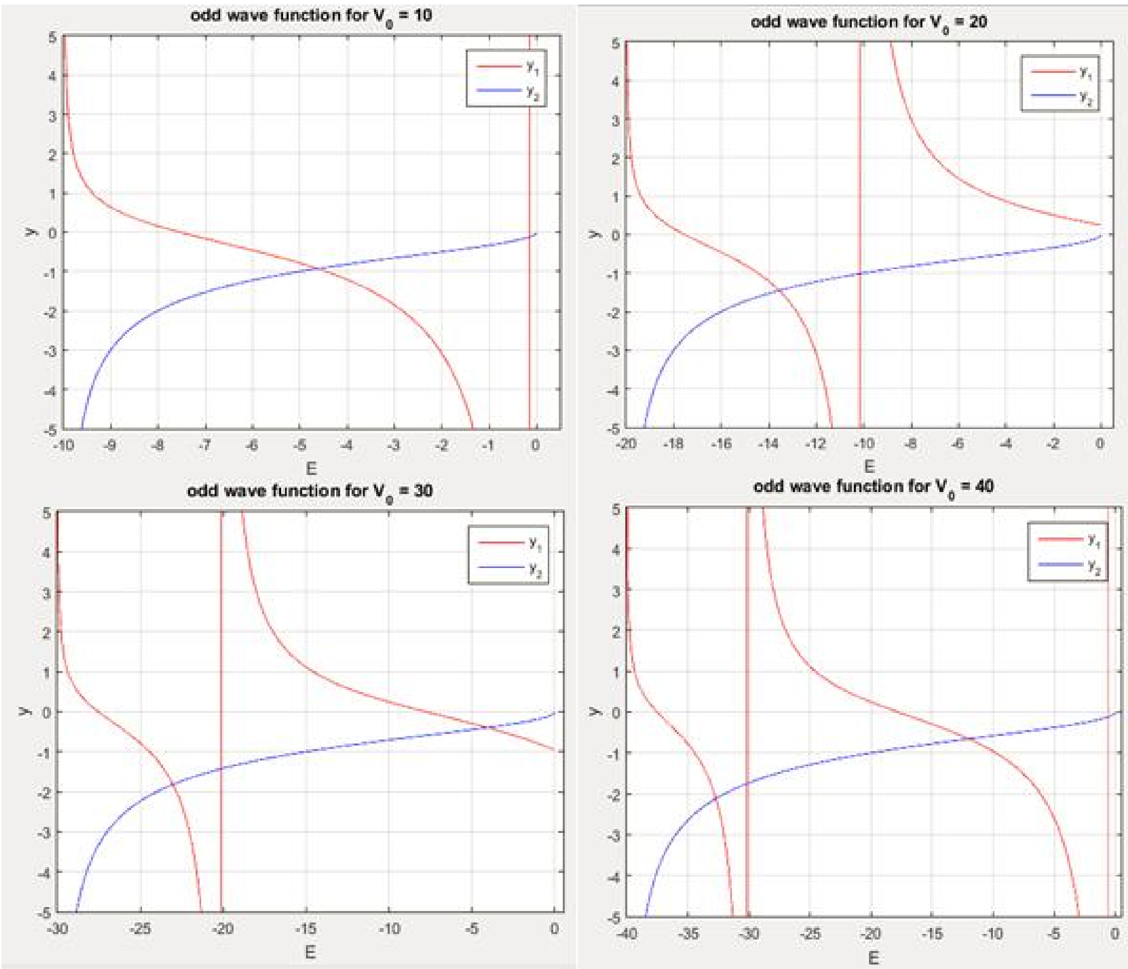
Using the graphical method to find the approximation of bound states when $V_0=10,20,30$, 40 then we have

For even wave function



$V_0 = 10$: Two bound state: $E = -8.592786$ (exact solution) and $E \approx -0.004$ $V_0 = 20$: Two bound states: $E \approx -6.1, -18.4$ $V_0 = 30$ Two bound states:$E \approx -14.9, -28.3$ $V_0 = 40$ Three bound states:$E \approx -0.055, -23.7, -38.2$

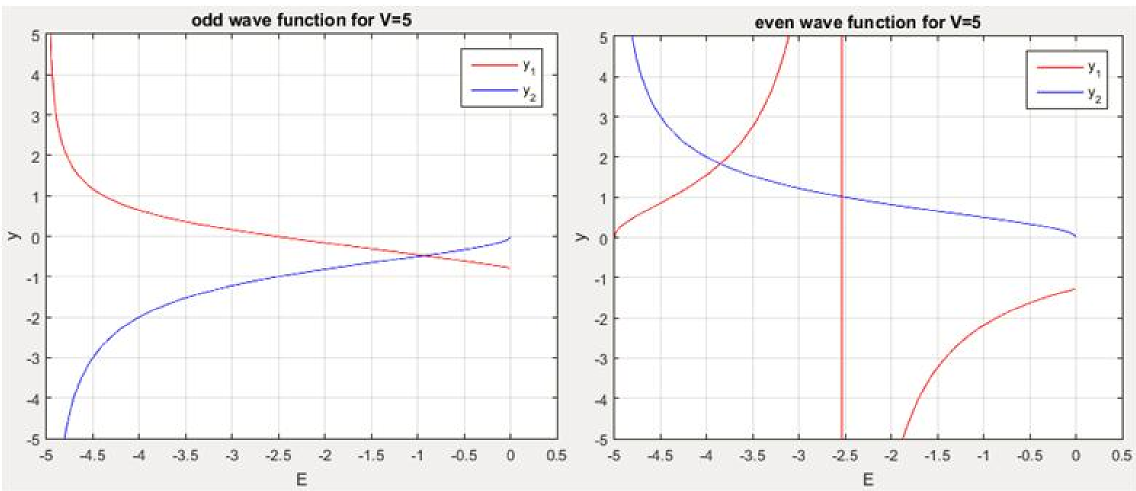
For odd wave function



$V_0 = 10$: One bound state: $E \approx -4.6$ $V_0 = 20$: One bound state: $E \approx -13.6$ $V_0 = 30$ Two bound states:$E \approx -23.1, -4.1$ $V_0 = 40$ Two bound states:$E \approx -32.7, -12.0$

Change the potential shallower

$V_0 = 5$ Two bound states:$E \approx -0.9$ for odd, and $E \approx -3.8$ for even



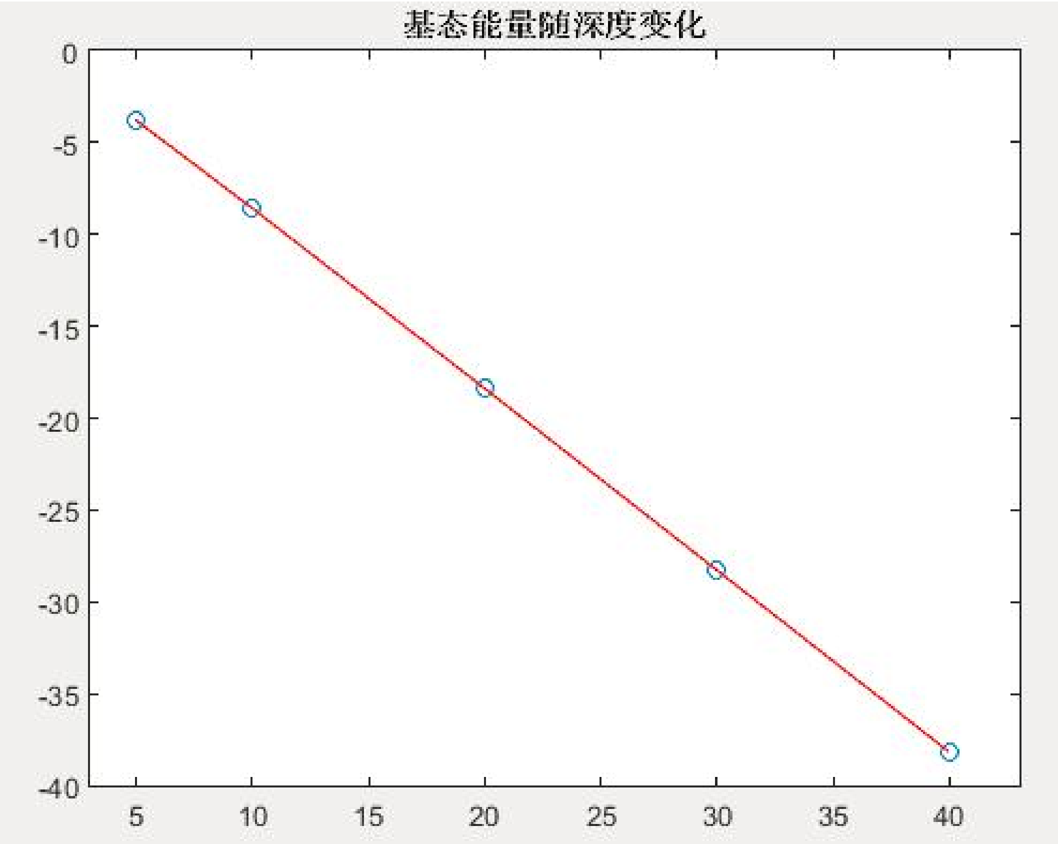
To sum up:

potential	number of bound state	ground state

20	1(o) + 2(e)	-18.4
30	2(o) + 2(e)	-28.3
40	2(o) + 3(e)	-38.2

(o) means number of bound state for odd wave functions; (e) means number of bound state for even wave functions;

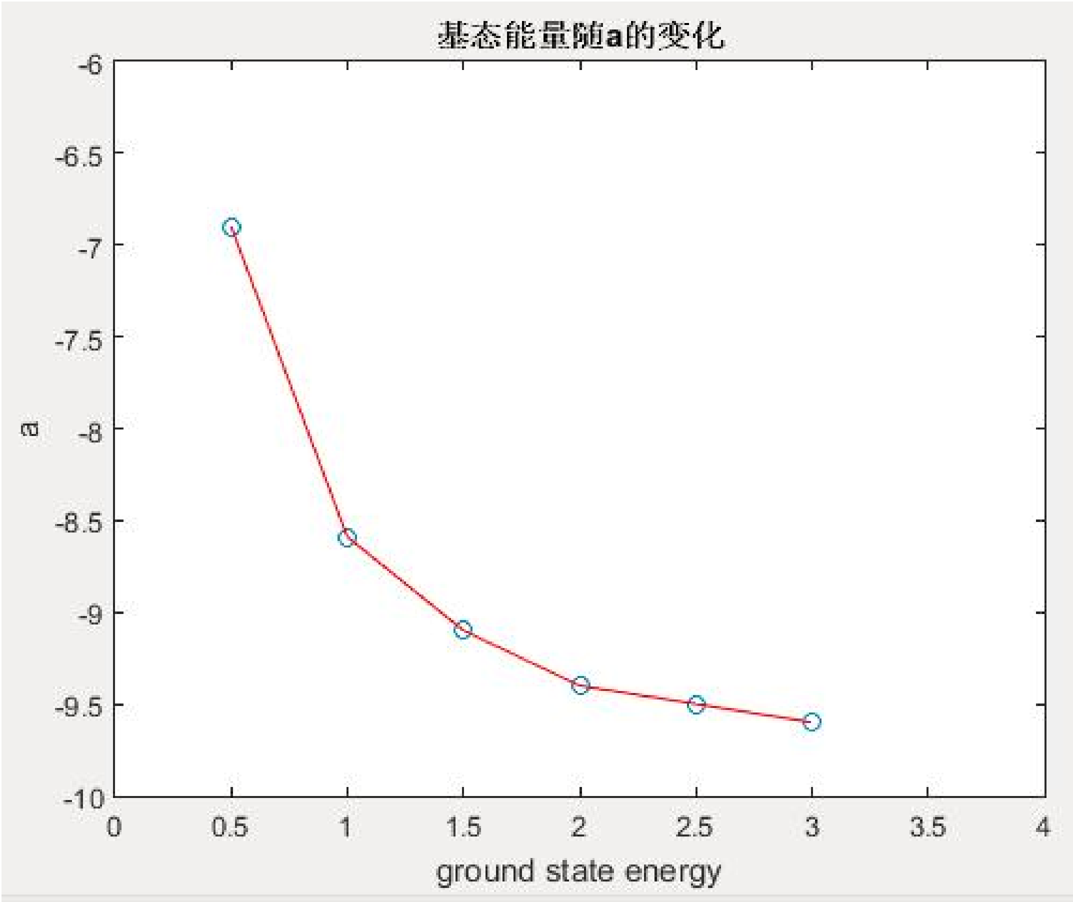
So making the potential deeper will produce a lager number of and deeper bound states. And the relationship between ground state energy and depth is showen as follow.



(c) Changing the width of the barrier causes equations turning into
$a\sqrt{10+E}\tan(a\sqrt{10+E}) = \sqrt{-E}$, where $-10 < E < 0$ (even)
$a\sqrt{10+E}\cot(a\sqrt{10+E}) = -\sqrt{-E}$, where $-10 < E < 0$ (odd)

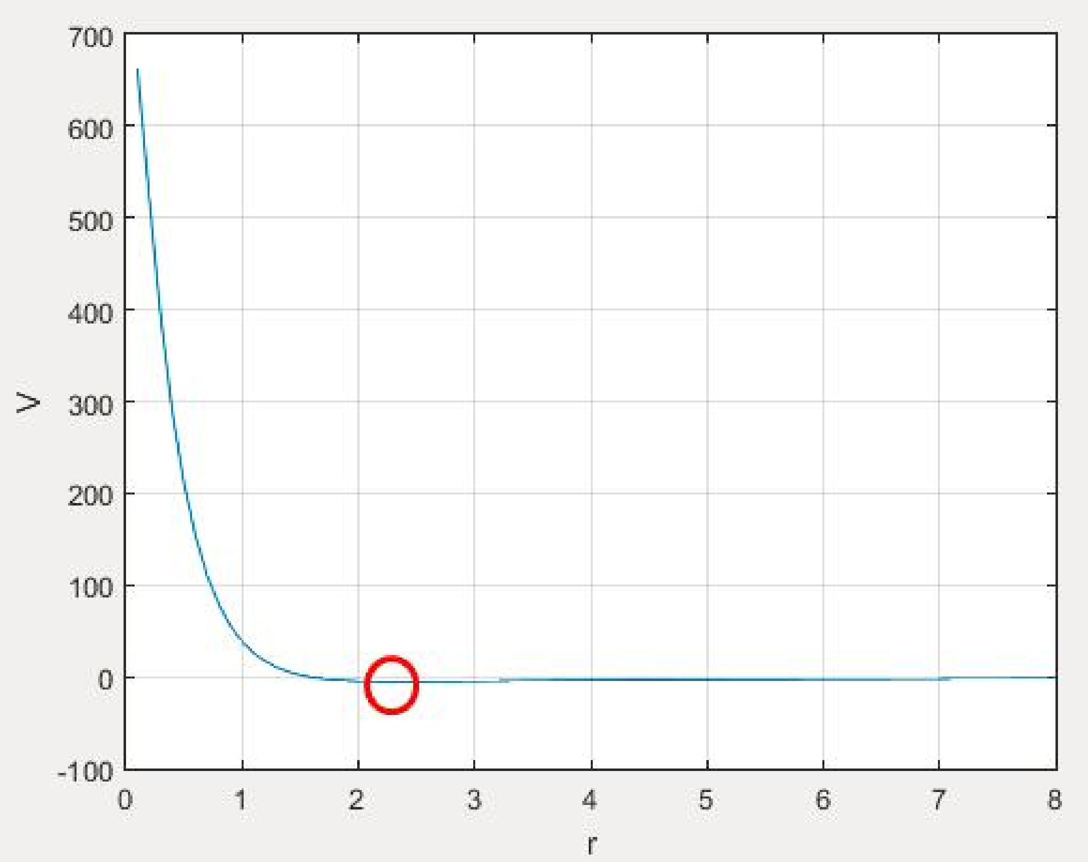
barrier	even state	odd state	ground state
0.5	-6.9	-0.9	-6.9
1.0	-8.592786	-4.6	-8.59
1.5	-9.1 ; -2.0	-6.5	-9.1

2.5	-9.5 ; -5.0	-8.0 ; -0.7	-9.5
3.0	-9.6 ; -5.9	-8.4 ; -2.1	-9.6



As the width of the barrier increase,ground state energy approach to -10.

3



From the lecture, we know that the bond length req is the equilibrium distance when $V(r)$ is at its minimum. So we should find the root of $V\prime(r) = 0$ $V(r) = -\frac{e^2}{r} + V_0 e^{-\frac{r}{r_0}}$ $V\prime(r) = \frac{e^2}{r^2} - \frac{V_0}{r_0} e^{-\frac{r}{r_0}}$ so using the Newton-Rafson method: code: pro3.m

```
clear all;close all;clc;
x1=2.5;e_square=14.4;v0=1090;r0=0.330;
g=@(x) e_square./(x.^2) - (v0/r0)*exp(-x./r0);
g_prime=@(x) -2*e_square./(x.^3)+(v0/(r0^2))*exp(-x./r0);
N=100;eps=1e-8
for i=1:N
    x2=x1-g(x1)./g_prime(x1);
    fprintf("N:%d \t x1:%.8f \t x2:%.8f \n",i,x1,x2);
    if abs(x2-x1)<eps
        fprintf("the result of Newton-Rafson is %.8f \n",x2);
        break
    end
    x1=x2;
end

v=@(x) -e_square/x + v0*exp(-x/r0);
req=fminsearch(v,x1);
fprintf("the result of matlab built-in fminsearch is %.8f \n",req)
```

the result is: the result is:

```
N:1       x1:2.50000000    x2:2.31439241
N:2       x1:2.31439241    x2:2.35685357
N:3       x1:2.35685357    x2:2.36051342
N:4       x1:2.36051342    x2:2.36053848
N:5       x1:2.36053848    x2:2.36053848
the result of Newton-Rafson is 2.36053848
the result of matlab built-in fminsearch is 2.36053848
```

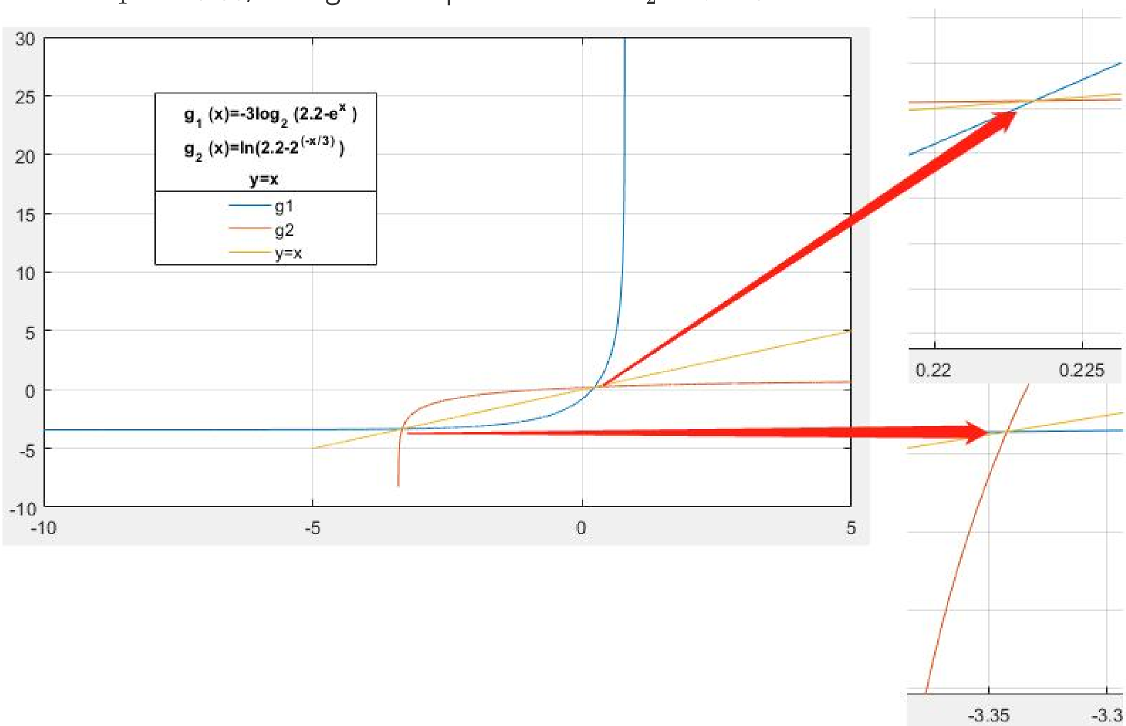
we find that the result is equal.

4

$2^{-\frac{x}{s}}+e^{x}=2.2\, g_1(x)=-3\log_2(2.2-e^{x}), g_2(x)=\ln(2.2-2^{-\frac{x}{3}})$

(a)

We plot g_1 and g_2 and $y=x$ in a single graph. Reading from the graph, the left fixed point is about $x_1=-3.35$, the right fixed point is about $x_2=0.225$



(b)

For $g_1(x)$, We start from $x_1=-4$ and $x_1=-3$ and apply 4 iterations. Numerical result shows in the following. And we find in both cases x_1 converge to the same result, about -3.34. We also using plot to demonstrate process of the convergence graphically. $\mathbf{Code:}$

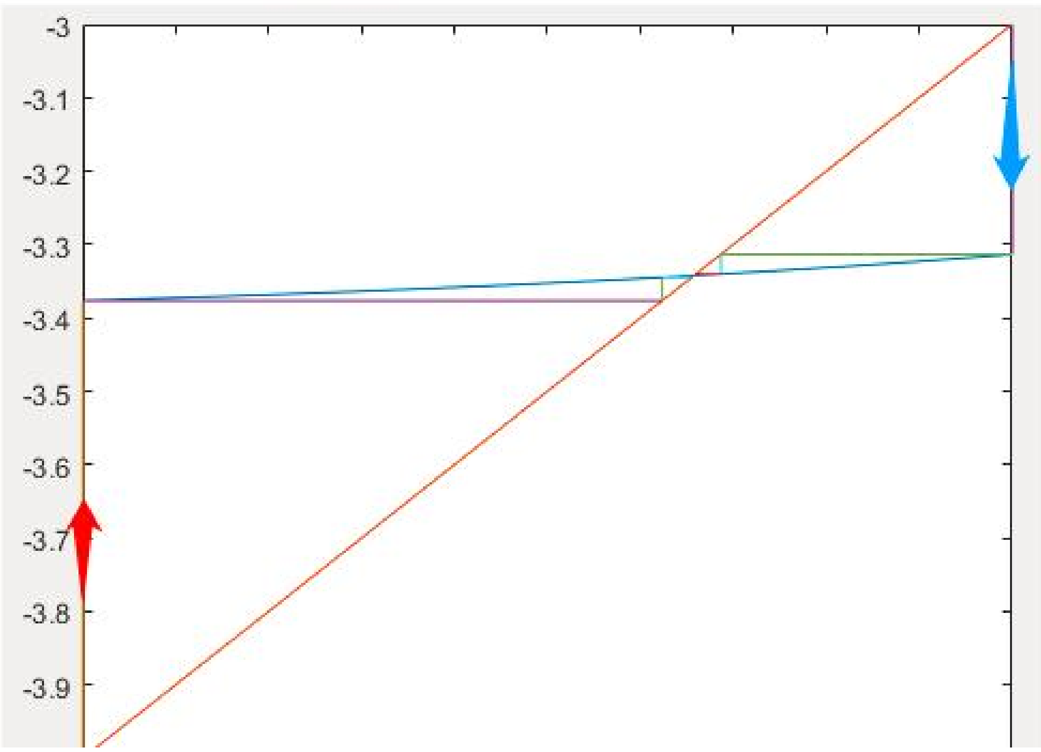
```
%g1
g1=@(t) -3*log2(2.2-exp(t));
x=-4:0.001:-3;
y=g1(x);
```

```
x1=-4;
result1=zeros(4,2);
for i=1:4
    x2=g1(x1);
    plot([x1,x1],[x1,x2]);
    plot([x1,x2],[x2,x2]);
    x1=x2;
    result1(i,:)=[i,x2];
end

%smaller fixed point, right side integer
x1=-3;
result2=zeros(4,2);
for i=1:4
    x2=g1(x1);
    plot([x1,x1],[x1,x2]);
    plot([x1,x2],[x2,x2]);
    x1=x2;
    result2(i,:)=[i,x2];
end
```

Result :

$g_1(x) = -3\log_2(2.2 - e^x)$		
Iteration	x1=-4	x1=-3
1	-3.376327177213716	-3.313438602977319
2	-3.344754557425512	-3.340319761994467
3	-3.342563538129475	-3.342250111379362
4	-3.342408865593288	-3.342386711413074

graphical process of convergence

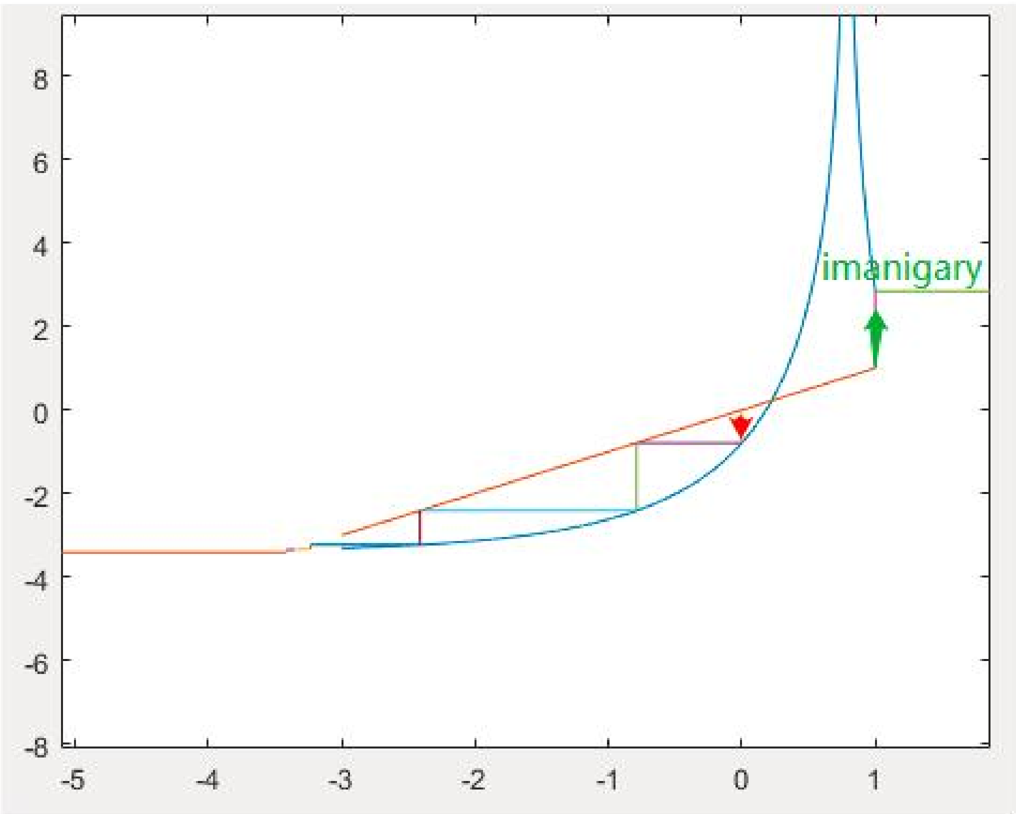


And we apply the same algorithm to the bigger fixed point of $g_1(x)$ Set $x_1 = 0$and $x_1 = 1$

$g_1(x) = -3\log_2(2.2 - e^x)$		
Iteration	x1=0	x1=1
1	-0.789103217501382	Imaginary
2	-2.411536014525200	Imaginary
3	-3.232390583529030	Imaginary
4	-3.334169720882290	Imaginary

In this case $g_1(1)$ is already imaginary. And start from $x_1 = 0$, it converges to the smaller fixed point ≈ -3.3 after 4 iterations.

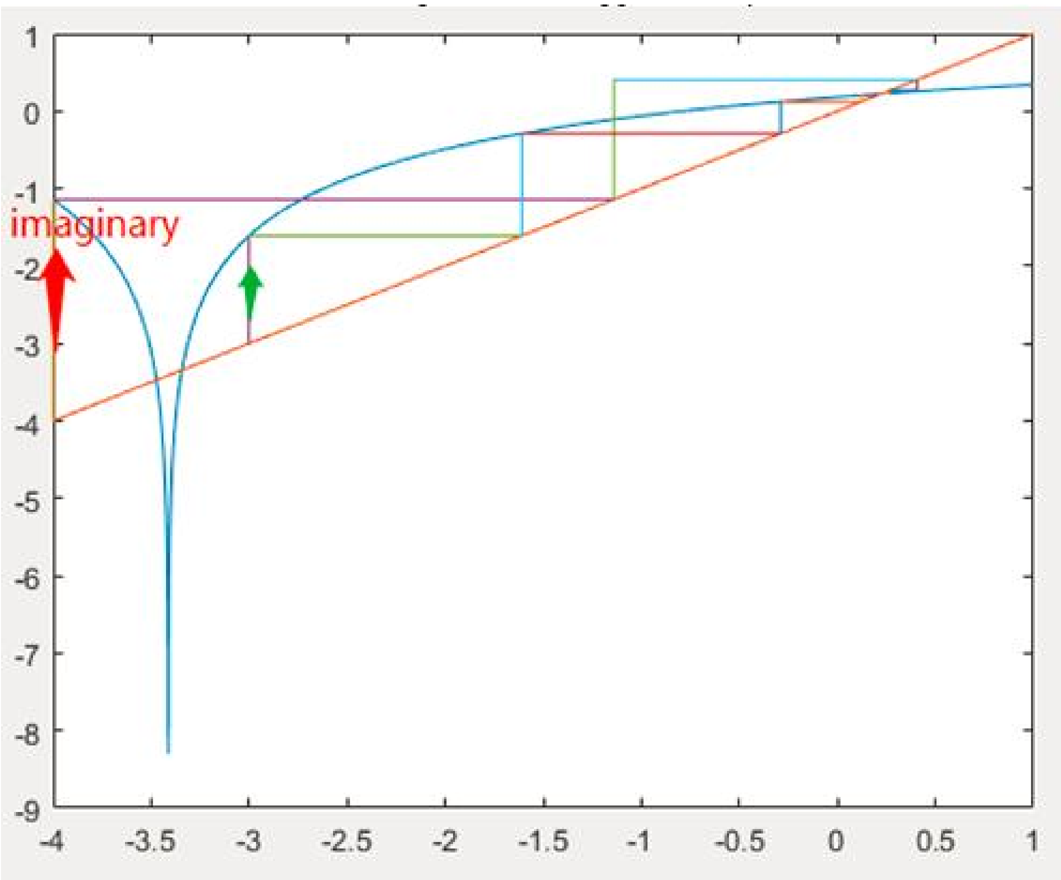
g1, start from x1= 0 and 1



Then we do the same process on $g_2(x)$

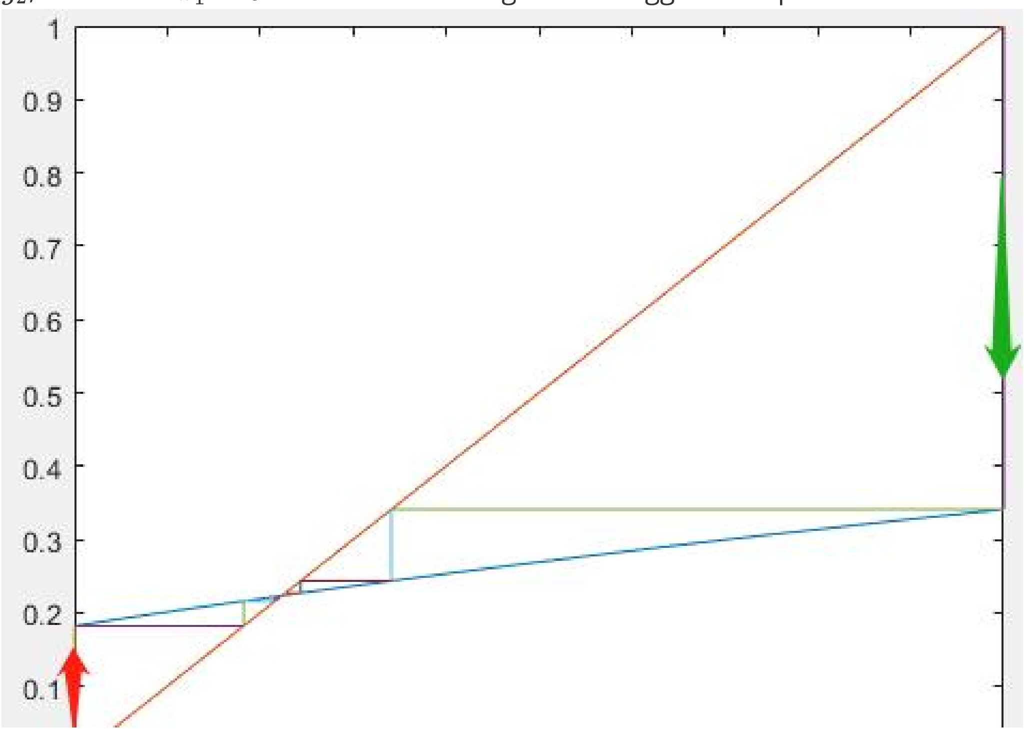
$g_2(x) = \ln\left(2.2 - 2^{-\frac{x}{3}}\right)$		
Iteration	x1=-4	x1=-3
1	Imaginary	-1.609437912434100
2	Imaginary	-0.288253737557777
3	Imaginary	0.123218324184092
4	Imaginary	0.205442236170297

g_2, start from $x_1 = -4$ and -3 Here $x_1 = -3$ converges to the bigger fixed point ≈ 0.21



$g_2(x) = \ln\left(2.2 - 2^{-\frac{x}{3}}\right)$		
Iteration	x1=0	x1=0
1	0.182321556793955	0.340961767874156
2	0.216119166591763	0.243538743454569
3	0.222109349725443	0.226908898256741
4	0.223162446115954	0.224004375308323

g_2, start from $x_1 = 0$ and 1 both converge to the bigger fixed point ≈ 0.22



Comments:

For g_1, near the smaller fixed point $|g\prime(x)| < 1$, and near the bigger fixed point $|g\prime(x)| > 1$. In our experiment, start from $x = -4, -3, 0, 1$ all converge to the smaller fixed point(≈ -3.34) where $|g'(x)| < 1$. similar for g_2, near the smaller fixed point $|g\prime(x)| > 1$, and near the bigger fixed point $|g\prime(x)| < 1$. Start from $x = -4, -3, 0, 1$ it all converge to the bigger fixed point (≈ 0.22) where $|g\prime(x)| < 1$. Theory of fixed point says if $|g\prime(x)| < 1$ at fixed point, convergence is guaranteed. And g_1, g_2 are two good examples demonstrating the theory.

5 Halley's model is : $x_{n+1} = x_n - \frac{2f(x_n)f\prime(x_n)}{2f\prime(x_n)^2 - f(x_n)f\prime\prime(x_n)}$

(a)Show that Halley's method has order of convergence 3 for simple zero.

First order

$$\lim_{n\to\infty}\frac{|x_{n+1}-x^*|}{|x_n-x^*|} = 1 - 2f\left[\frac{f'}{2(f')^2 - ff''}\right]' - 2f'\frac{f'}{2(f')^2 - ff''}\Bigg|_{x=x^*} = 0$$

Second order

$$\lim_{n\to\infty}\frac{|x_{n+1}-x^*|}{|x_n-x^*|^2} = \frac{1}{2}\left(1 - 2f\left[\frac{f'}{2(f')^2 - ff''}\right]' - 2f'\frac{f'}{2(f')^2 - ff''}\right)'\Bigg|_{x=x^*}$$

$$= \frac{1}{2}\left(-2f'\left[\frac{f'}{2(f')^2 - ff''}\right]' - 2f\left[\frac{f'}{2(f')^2 - ff''}\right]'' - \left[\frac{2(f')^2}{2(f')^2 - ff''}\right]'\right)\Bigg|_{x=x^*}$$

$$= \frac{3ff'f''^2 - 2ff'^2f'''}{(2f'^2 - ff'')^2}\Bigg|_{x=x^*} = 0$$

Third order

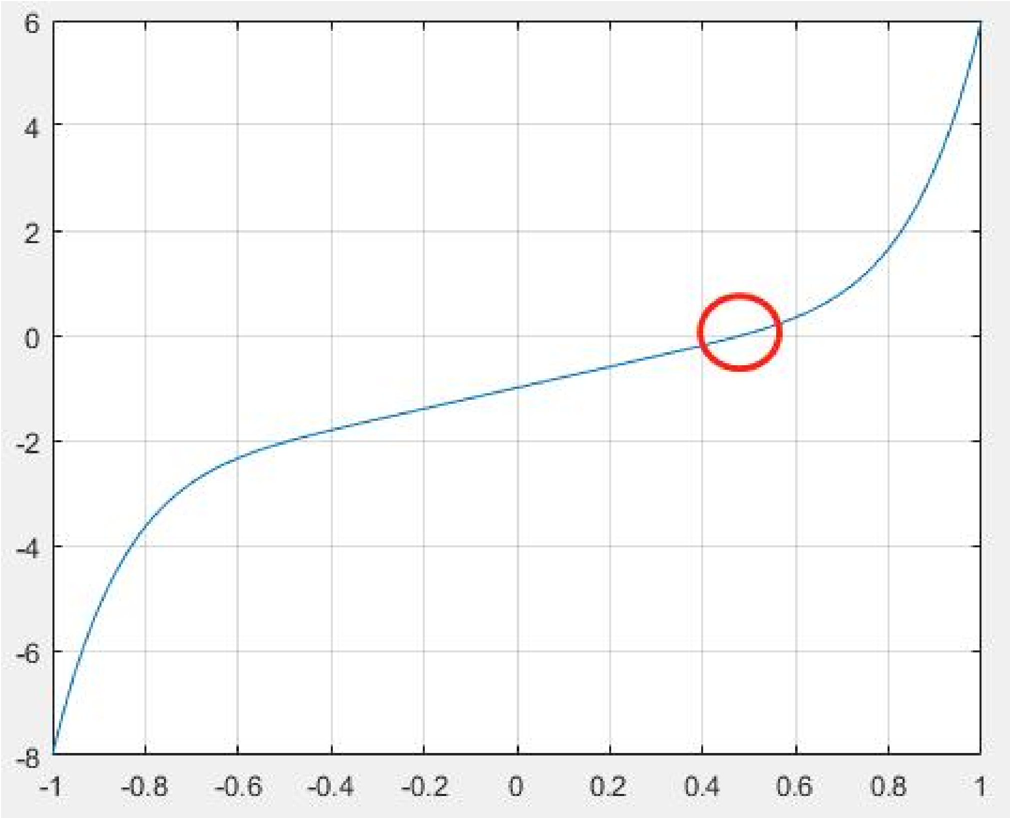
$$\lim_{n\to\infty}\frac{|x_{n+1}-x^*|}{|x_n-x^*|^3} = \frac{1}{3}[f\frac{3f'f''^2 - 2f'^2f'''}{(2f'^2 - ff'')^2}]'\Bigg|_{x=x^*}$$

$$= \frac{1}{3}f'\frac{3f'f''^2 - 2f'^2f'''}{(2f'^2 - ff'')^2}\Bigg|_{x=x^*} = \frac{3f''^2 - 2f'f'''}{12f'^2}\Bigg|_{x=x^*}$$

So for simple zero(which means $f' \neq 0$), the convergence of Halley method has order of 3.

(b)

Apply Halley's method to $f(x) = 5x^7 + 2x - 1$ and $g(x) = \frac{1}{x^3} - 10$ For $f(x) = 5x^7 + 2x - 1$



a quick plot shows the zero point is near 0.5 Use Halley's method to find the zero point

Code :

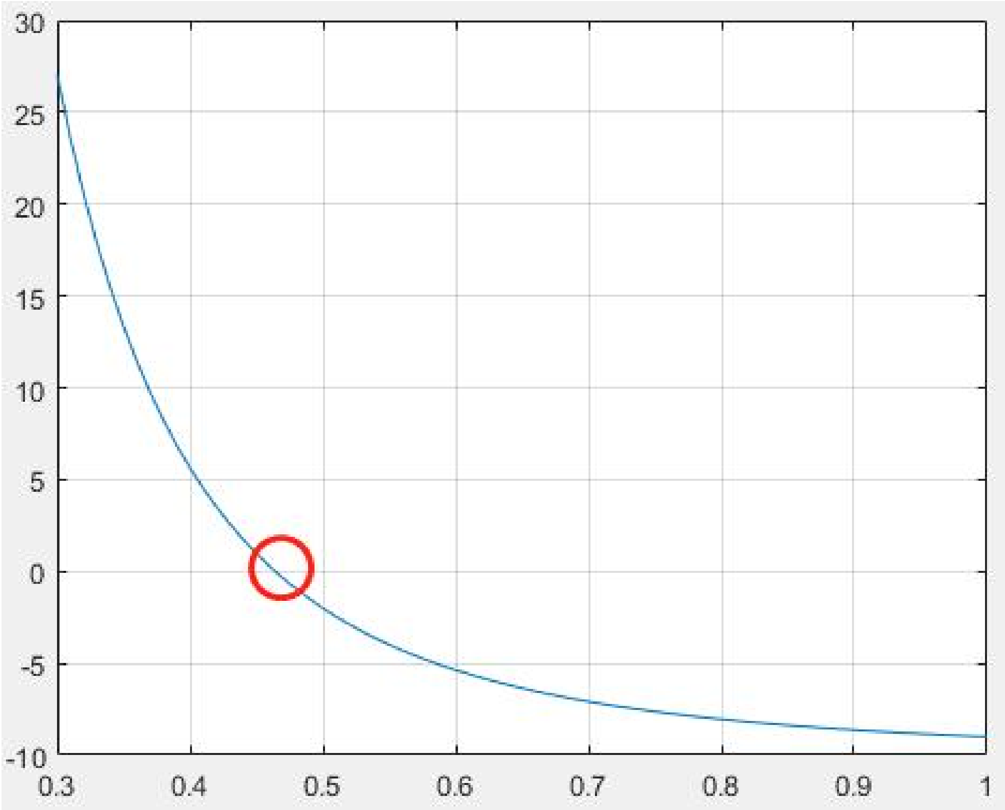
```
%Halley's method
y=@(t) 5*power(t,7)+2*t-1;
y1=@(t) 35*power(t,6)+2;
y2=@(t) 210*power(t,5);
x0=0.5;  %initial guess
format long
for i=1:20
    [i-1,x0,y(x0)]
    x0=x0-2*y(x0)*y1(x0)/(2*y1(x0)*y1(x0)-y(x0)*y2(x0));
end
```

Result :

Iteration	x_n	$f(x_n)$
0	0.500000000000000	0.039062500000000
1	0.484353401935187	-0.000024736639697
2	0.484363490583419	0.000000000000007
3	0.484363490583416	0.000000000000000
4	0.484363490583416	0.000000000000000

We can see only after 3 iterations, the x_n reaches its best value – it can't be improved more due to machine precision.

For



a quick plot shows the zero point is near 0.5

Firstly we apply the same algorithm as before using $g(x) = \frac{1}{x^3} - 10$, but $g(x_n)$ still remains a small deviation from zero, 0.000000000000002, after several iterations when x_n can't be improved any more. We think this small deviation may comes from the division $\frac{1}{x^3}$, so instead, we turn to use another function which has the same zero point, that is, $g(x) = 10x^3 - 1$. And it successfully eliminate the small deviation.

Code :

```
%Halley's method
y=@(t) 10*t*t*t-1;
y1=@(t) 30*t*t;
y2=@(t) 60*t;
x0=0.5;  %initial guess
format long
for i=1:20
    [i-1,x0,y(x0)]
    x0=x0-2*y(x0)*y1(x0)/(2*y1(x0)*y1(x0)-y(x0)*y2(x0));
end
```

Result :

Iteration	x_n	$f(x_n)$
0	0.500000000000000	0.250000000000000
1	0.464285714285714	0.000819970845481
2	0.464158883367588	0.000000000040787

After 3 iterations, Halley’s method has accomplished its own task.

(c) Compare the results above with the results from the bisection method and the Newton's method for these functions.

We apply all Halley's method, bisection, Newton's method in every iteration, to figure out how fast every method converges.

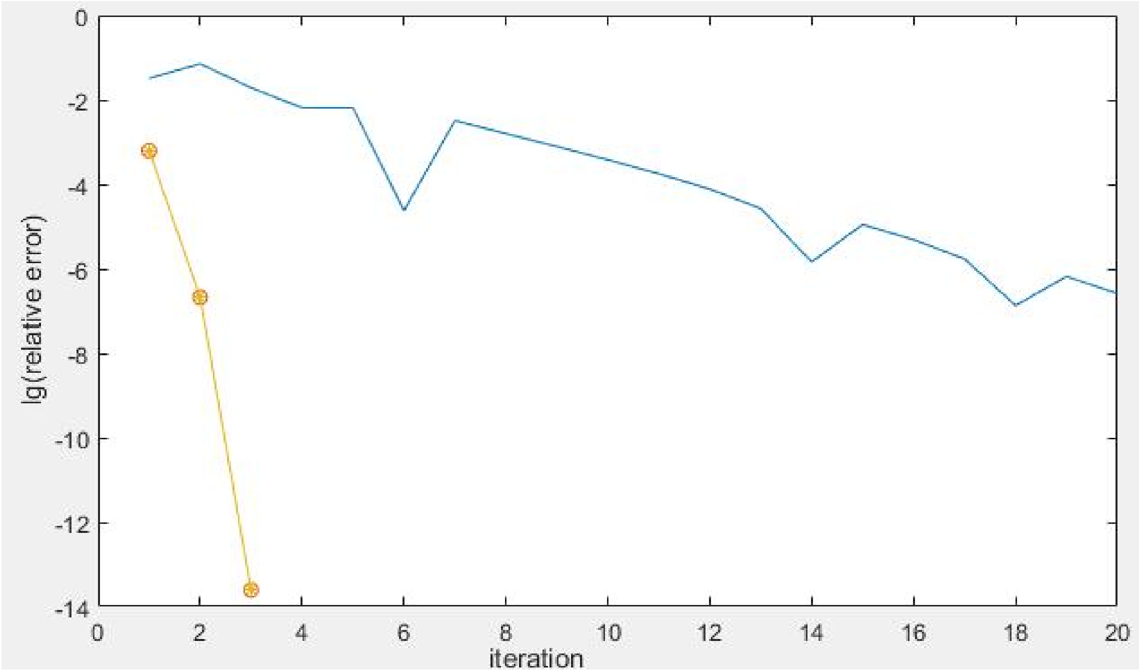
Code :

```
%Halley & bisection & Newton
y=@(t) 5*power(t,7)+2*t-1;
y1=@(t) 35*power(t,6)+2;
y2=@(t) 210*power(t,5);
h0=0.5;  %initial guess of Halley'method
b0=0.4; b1=0.6; %initial range of bisection
n0=0.5; %initial guess of Newton's method
result=zeros(20,7);
for i=1:20

    h0=h0-2*y(h0)*y1(h0)/(2*y1(h0)*y1(h0)-y(h0)*y2(h0));    %halley
method

    b2=(b0+b1)/2;   %bisection method
    if (y(b0)*y(b2)) < 0  %since the zero point is an irrational
number, it can't locate at b2
        b1=b2;
    else
        b0=b2;
    end

    n0=n0-y(n0)/y1(n0); %newton's method

    result(i,:)=[i,b2,y(b2),n0,y(n0),h0,y(h0)];
end
```

Since both $f(x) = 5x^7 + 2x - 1$ and $g(x) = 10x^3 - 1$ use similar codes: with only functions and numbers changing. We only attach the code of $f(x) = 5x^7 + 2x - 1$here. While we will show the results of both.

Result :

$f(x)= 5x^7+2x-1$						
	bisection		Newton's method		Halley's method	
	x_n	$f(x_n)$	x_n	$f(x_n)$	x_n	$f(x_n)$
1	0.500000000000000	0.039062500000000	0.484662576687117	0.000733596717110	0.484353401935187	-0.000024736639697
2	0.450000000000000	-0.081316527343750	0.484363592847448	0.000000250746967	0.484363490583419	0.000000000000007
3	0.475000000000000	-0.022721199371338	0.484363490583428	0.000000000000029	0.484363490583416	0.000000000000000
4	0.487500000000000	0.007718421621084	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
5	0.481250000000000	-0.007607295984676	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
6	0.484375000000000	0.000028220957347	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
7	0.482812500000000	-0.003796263089269	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
8	0.483593750000000	-0.001885716085254	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
9	0.483984375000000	-0.000929173031604	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
10	0.484179687500000	-0.000450582618750	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
11	0.484277343750000	-0.000211207502984	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
11	0.484277343750000	-0.000211207502984	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
12	0.484326171875000	-0.000091499944251	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
13	0.484350585937500	-0.000031641161730	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
14	0.484362792968750	-0.000001710519314	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
15	0.484368896484375	0.000013255114729	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
16	0.484365844726563	0.000005772271637	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
17	0.484364318847656	0.000002030869644	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
18	0.484363555908203	0.000000160173536	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
19	0.484363174438477	-0.000000775173297	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000
20	0.484363365173340	-0.000000307499982	0.484363490583416	0.000000000000000	0.484363490583416	0.000000000000000



Halley's method reaches its best x_n in 3 iterations, and Newton's method 4 iterations, while bisection only obtains 6 effective digits after all 20 iterations end. Clearly Halley's method and Newton's method ride over bisection algorithm in this case. And we plot $\log$ (relative error) of all three methods against iteration times, here we use $x^* = 0.484363490583416$ to calculate the relative error.

Theoretically, bisection algorithm converges as

$$\frac{|x_{n+1} - x^*|}{|x_n - x^*|} \cong \frac{1}{2}$$

Newton's method converges as

$$\sim \lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^2} = \frac{f''(x^*)}{2f'(x^*)} \approx 1.142(\textit{in this case})$$

Halley's method converges as

$$\sim \lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^3} = \frac{3f''^2 - 2f'f'''}{12f'^2} \approx 2.625(\textit{in this case})$$

Result of $g(x) = 10x^3 - 1$

$g(x)=10x^3-1$						
	bisection		Newton's method		Halley's method	
	x_n	$f(x_n)$	x_n	$f(x_n)$	x_n	$f(x_n)$
1	0.450000000000000	-0.088750000000000	0.466666666666667	0.016296296296297	0.464285714285714	0.000819970845481
2	0.475000000000000	0.071718750000000	0.464172335600907	0.000086948435035	0.464158883367589	0.000000000040787
3	0.462500000000000	-0.010683593750000	0.464158883751135	0.000000002519766	0.464158883361278	0.000000000000000
4	0.468750000000000	0.029968261718750	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
5	0.465625000000000	0.009505920410156	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
6	0.464062500000000	-0.000622825622558	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
7	0.464843750000000	0.004433035850525	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
8	0.464453125000000	0.001902979016304	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
9	0.464257812500000	0.000639545395970	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
10	0.464160156250000	0.000008227089420	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
11	0.464111328125000	-0.000307332462398	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
12	0.464135742187500	-0.000149560985882	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
13	0.464147949218750	-0.000070669023134	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
14	0.464154052734375	-0.000031221485590	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
15	0.464157104492188	-0.000011497327768	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
16	0.464158630371094	-0.000001635151595	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
17	0.464159393310547	0.000003295960808	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
18	0.464159011840820	0.000000830402580	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
19	0.464158821105957	-0.000000402375014	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000
20	0.464158916473389	0.000000214013657	0.464158883361278	0.000000000000000	0.464158883361278	0.000000000000000

In this case Halley's method reaches its best x_n in 3 iterations, and Newton's method 4 iterations, while bisection only obtains6 effective digits after all20 iterations end.