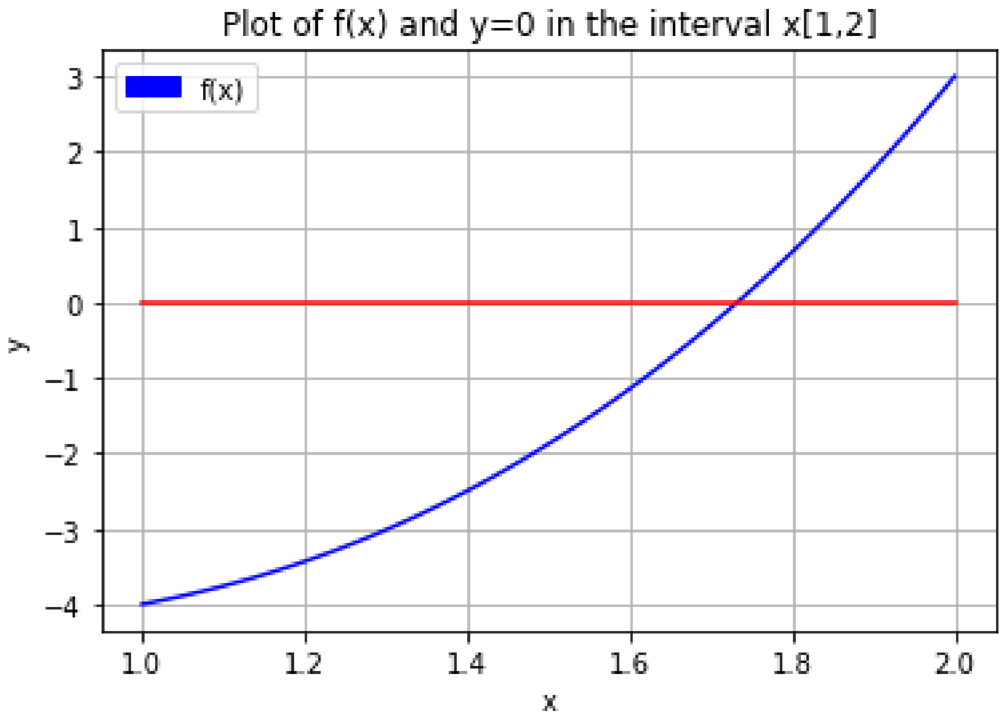
HW1_Deshmukh_Rahul

September 12, 2017

1 Homework 1 by *Rahul Deshmukh*

Problem 1
Given: Function $f(x) = x^3 + x^2 - 3x - 3$
Part 1: Verify that $f(x)$ has a solution in $x \in [1,2]$ by plotting the function

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import math
a=1;b=2
x = np.linspace(a,b,101)
def f(x):
    return x**3 + x**2 -3*x - 3
#defining z so as to plot the line y=0
z = 0*x;

plt.plot(x,f(x),'b')
plt.plot(x,z,'r')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(1)
legend_f = mpatches.Patch(color='blue', label='f(x)')
plt.legend(handles=[legend_f])
plt.title('Plot of f(x) and y=0 in the interval x[1,2]')
plt.show()

print('f({})= '.format(a)+str(f(a))+', f({})= '.format(b)+str(f(b))+
      ' and f({})*f({}) = '.format(a,b)+str(f(a)*f(b))+'\n')
if f(a)*f(b)<0:
    print('Then there exists a root in the interval [{},{}]'.format(a,b))
```




```
f(1)= -4, f(2)= 3 and f(1)*f(2) = -12

Then there exists a root in the interval [1,2]
```

As we can notice from the above plot the function $f(x)$ intersects with $y = 0$ at exactly one point in the interval $x \in [1, 2]$. Therefore there is a zero in this interval

Part 2: To Perform 5 iterations of the bisection method

```
In [2]: N=5;tol=10^-9
        p=np.zeros(N)
        print('n\tpn       \tf(pn)\n')
        for i in range (0,N):
            if f(p[i])==0 or (b-a)/2<tol:
                print('*** The root is {} with accuracy {}***'.format(p[i-1],tol))
                break
            p[i] = (a + b)/2
            if (f(p[i])*f(b))<0:
                a = p[i]
            else:
                b = p[i]
            print(str(i+1)+'\t{:0.9f}\t{:0.9f}\n'.format(p[i],f(p[i])))
        if i == N-1:
            print('***We have reached the maximum number of {} iterations***\n'.format(N))
```

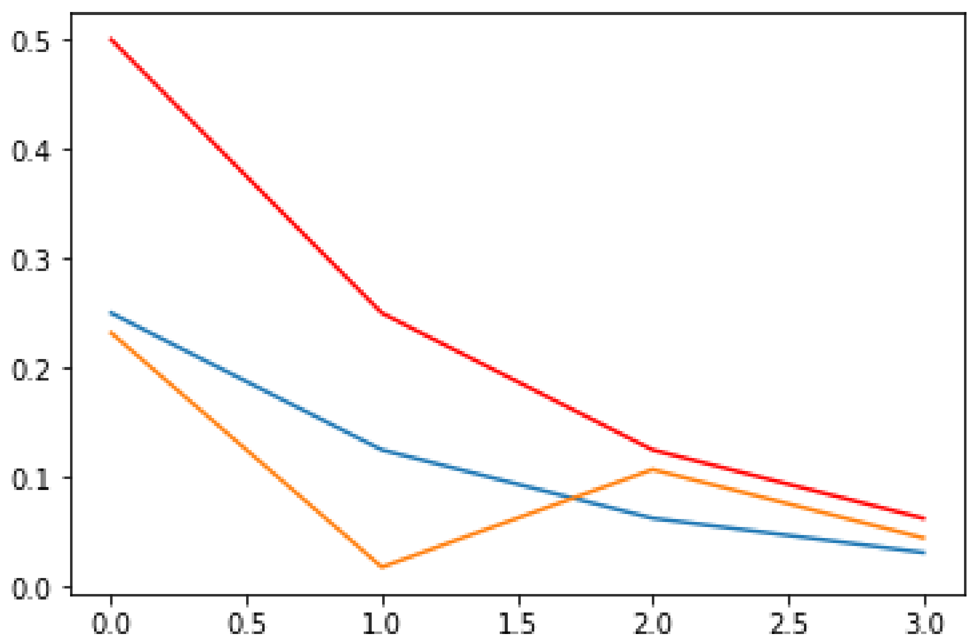
```
n        pn                f(pn)

1        1.500000000       -1.875000000

2        1.750000000       0.171875000

3        1.625000000       -0.943359375

4        1.687500000       -0.409423828

5        1.718750000       -0.124786377

***We have reached the maximum number of 5 iterations***
```

Verifying the error bound

In [3]:
```python
er = np.zeros(N)
actual =np.zeros(N)
th = np.zeros(N)
root = 3**0.5
a=1;b=2;
for i in range(0,N-1):
    er[i] = np.abs(p[i+1]-p[i])
    actual[i] = np.abs(p[i]-root)
    th[i] = (b-a)/(2**(i+1))
th[-1]=(b-a)/(2**N)
#print(er)
erp = np.zeros(N)
actualp = np.zeros(N)
for i in range(0,N-2):
    erp[i] = er[i+1]
#print(erp)
print('Table for iterative error vs theoretical error vs actual error\n\n')
print('n\titerative error       \ttheoretical error\tactual error\n')
for i in range(0,N):
    print(str(i+1)+'\t{:0.9f}        '.format(er[i])+'\t{:0.9f}        '.format(th[i])
          +'\t{:0.9f}\n'.format(actual[i]))
plt.plot(np.arange(N-1),er[:N-1])
plt.plot(np.arange(N-1),th[:N-1],'r')
plt.plot(np.arange(N-1),actual[:N-1])
plt.show()
```

```
Table for iterative error vs theoretical error vs actual error


n        iterative error               theoretical error        actual error
```

1	0.250000000	0.500000000	0.232050808
2	0.125000000	0.250000000	0.017949192
3	0.062500000	0.125000000	0.107050808
4	0.031250000	0.062500000	0.044550808
5	0.000000000	0.031250000	0.000000000

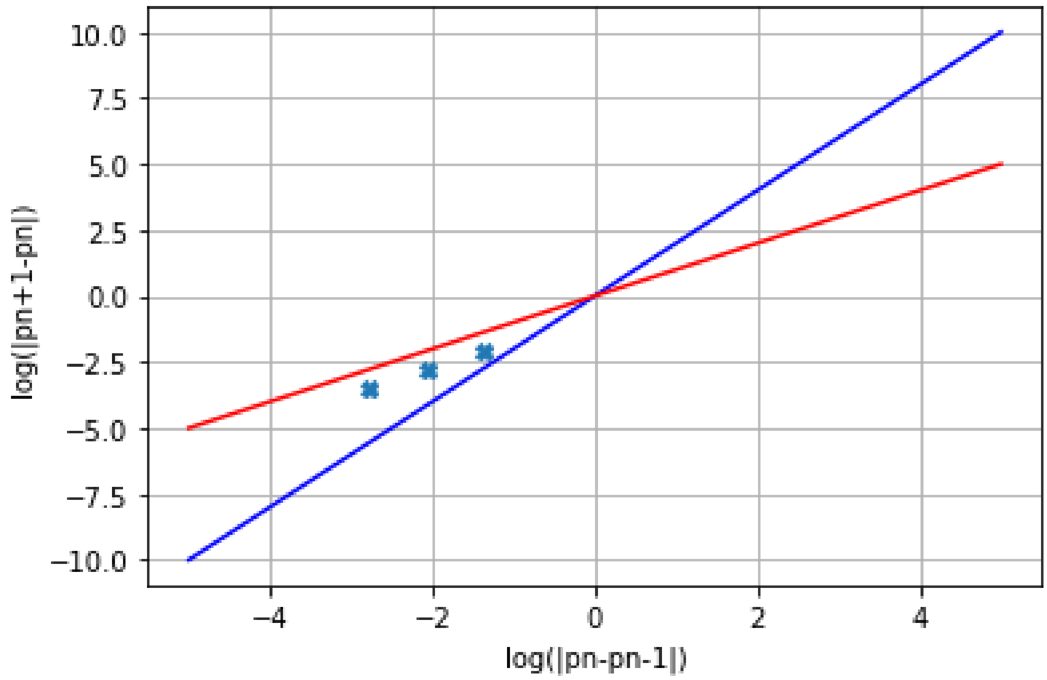


We can observe from the above table and the plot, that values of the *iterative error* and *actual error* are always **less than** the *theoretical error(red)* **but** the actual errors do not steadily decrease

```
In [4]: # Now plotting the log - log plot
        t=np.linspace(-5,5,101)
        plt.plot(t,2*t,'b') #for plotting y=2x
        plt.plot(t,t,'r')#for plotting y=x
        plt.plot(np.log(er),np.log(erp),'X')
        plt.xlabel('log(|pn-pn-1|)')
        plt.ylabel('log(|pn+1-pn|)')
        plt.grid(1)
        plt.show()
```

```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:5
```

"""

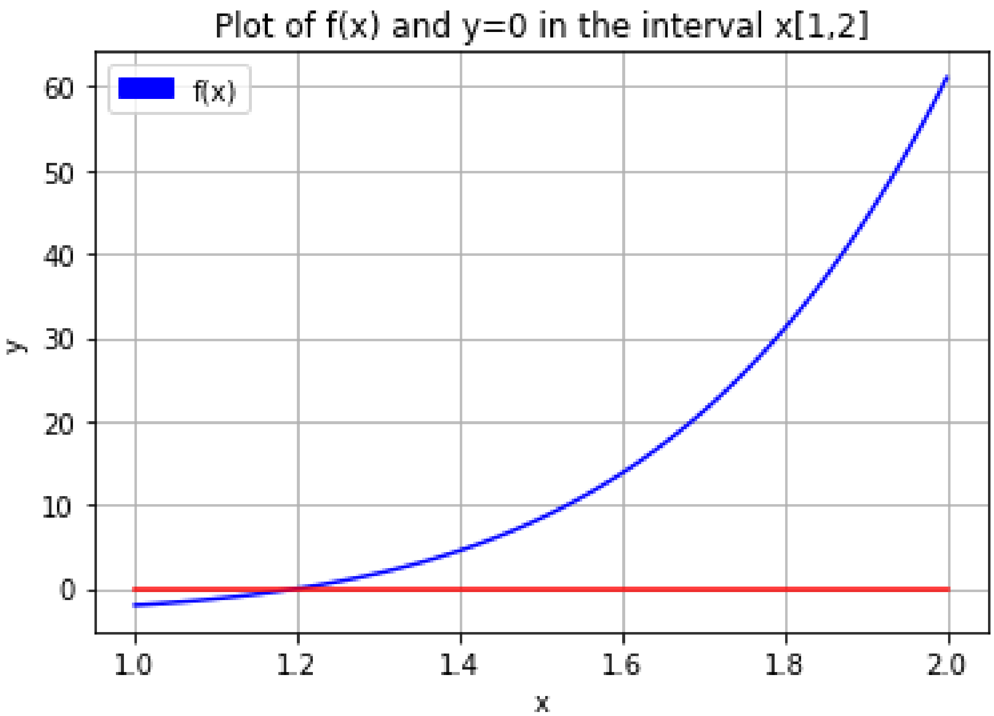


Therefore from the log-log plot we can say that the bisection method has a **linear** error (as the points are along y=x(red line))

Problem 2
Given: Function $f(x) = x^6 - 3$
Part 1: Verify that $f(x)$ has a solution in $x \in [1, 2]$

In [5]:
```
a=1;b=2
x = np.linspace(a,b,101)
def f(x):
    return x**6- 3
#defining z so as to plot the line y=0
z = 0*x;

plt.plot(x,f(x),'b')
plt.plot(x,z,'r')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(1)
legend_f = mpatches.Patch(color='blue', label='f(x)')
plt.legend(handles=[legend_f])
```

```
plt.title('Plot of f(x) and y=0 in the interval x[1,2]')
plt.show()

print('f({})= '.format(a)+str(f(a))+', f({})= '.format(b)+str(f(b))+
      ' and f({})*f({}) = '.format(a,b)+str(f(a)*f(b))+' < 0\n')
if f(a)*f(b)<0:
    print('Then there exists a root in the interval [{},{}]'.format(a,b))
```




```
f(1)= -2, f(2)= 61 and f(1)*f(2) = -122 < 0

Then there exists a root in the interval [1,2]
```

```
In [6]: N=5;tol=10^-9
        p=np.zeros(N)
        print('n\tpn       \tf(pn)\n')
        for i in range (0,N):
            if f(p[i])==0 or (b-a)/2<tol:
                print('*** The root is {} with accuracy {}***'.format(p[i-1],tol))
                break
            p[i] = (a + b)/2
            if (f(p[i])*f(b))<0:
                a = p[i]
            else:
```

```
            b = p[i]
        print(str(i+1)+'\t{:0.9f}\t{:0.9f}\n'.format(p[i],f(p[i])))
    if i == N-1:
        print('***We have reached the maximum number of {} iterations***\n'.format(N))
```

```
n        pn              f(pn)

1        1.500000000       8.390625000

2        1.250000000       0.814697266

3        1.125000000       -0.972713470

4        1.187500000       -0.195846975

5        1.218750000       0.277085499

***We have reached the maximum number of 5 iterations***
```

Verifying the error bound

```
In [7]: er = np.zeros(N)
        th=np.zeros(N)
        a = 1;b=2;
        for i in range(0,N-1):
            er[i] = np.abs(p[i+1]-p[i])
            actual[i] = np.abs(p[i]-root)
            th[i]=(b-a)/(2**i)
        #print(er)
        th[-1]=(b-a)/(2**N)
        erp = np.zeros(N)
        for i in range(0,N-2):
            erp[i] = er[i+1]
        #print(erp)
        print('Table for iterative error vs theoretical error\n\n')
        print('n\titerative error      \ttheoretical error\n')
        for i in range(0,N):
            print(str(i+1)+'\t{:0.9f}        '.format(er[i])+'\t{:0.9f}\n'.format(th[i]))

        #plt.plot(np.arange(N-1),er[:N-1])
        #plt.plot(np.arange(N-1),th[:N-1],'r')
        plt.show()
```

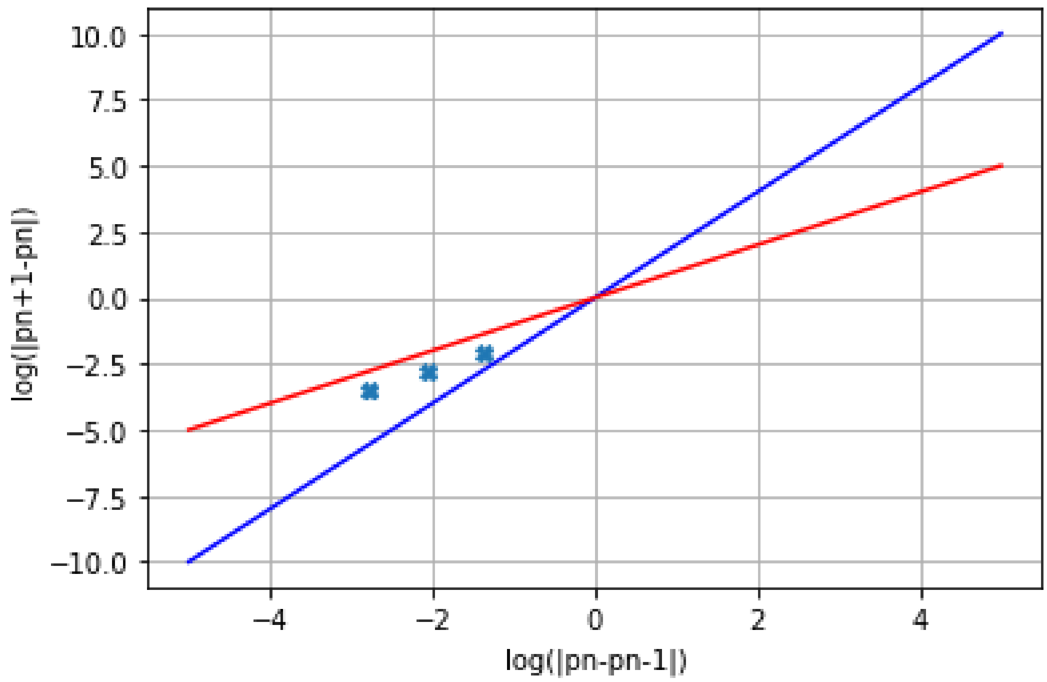
```
Table for iterative error vs theoretical error
```

```
n        iterative error             theoretical error

1        0.250000000               1.000000000

2        0.125000000               0.500000000

3        0.062500000               0.250000000

4        0.031250000               0.125000000

5        0.000000000               0.031250000
```

We can observe from the above table that values of the *Iterative error* are always **less than** the *Theoretical error*

```
In [8]: # Now plotting the log - log plot
        t=np.linspace(-5,5,101)
        plt.plot(t,2*t,'b') #for plotting y=2x
        plt.plot(t,t,'r')#for plotting y=x
        plt.plot(np.log(er),np.log(erp),'X')
        plt.plot(np.log(actual),np.log(actualp),'*')
        plt.xlabel('log(|pn-pn-1|)')
        plt.ylabel('log(|pn+1-pn|)')
        plt.grid(1)
        plt.show()
```

```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:5
  """
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:6
```



Therefore from the log-log plot we can say that the bisection method has a **linear** (as it is along y=x (red line)) error theoretically

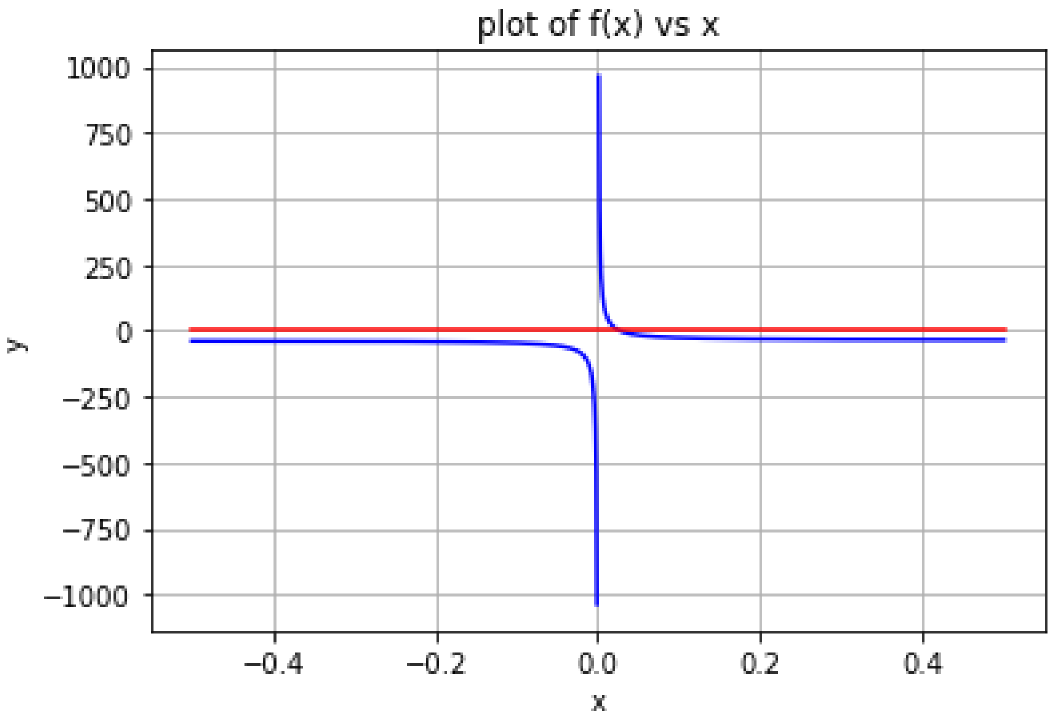
---

Problem 3

To find value of $\frac{1}{37}$ to the order of five decimal places using bisection method for $f(x) = \frac{1}{x} - 37 = 0$

```
In [9]: x= np.linspace(-0.5,0.5,1001)
        def f(x):
            return x**-1-37
        z = 0*x
        plt.plot(x,f(x),'b')
        plt.plot(x,z,'r')
        plt.title('plot of f(x) vs x')
        plt.ylabel('y')
        plt.xlabel('x')
        plt.grid(1)
        plt.show()
```

```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:3
  This is separate from the ipykernel package so we can avoid doing imports until
```



We know that the function $f(x)$ has a root at $x = \frac{1}{37}$. Also, From the above plot we can say that the function $f(x)$ will have a **positive value** for $0 < x < \frac{1}{37}$ and a **negative value** for $x > \frac{1}{37}$

In [10]:
```
#estmating value for a & b
a0 = 1/50;b0=1/2;tol=10**-5
print('f({})= '.format(a0)+str(f(a0))+', f({})= '.format(b0)+str(f(b0))+
      ' and f({})*f({}) = '
      .format(a0,b0)+str(f(a0)*f(b0))+' < 0\n')
if f(a0)*f(b0)<0:
    print('Then there exists a root in the interval [{},{}]'.format(a0,b0))
```

```
f(0.02)= 13.0, f(0.5)= -35.0 and f(0.02)*f(0.5) = -455.0 < 0

Then there exists a root in the interval [0.02,0.5]
```

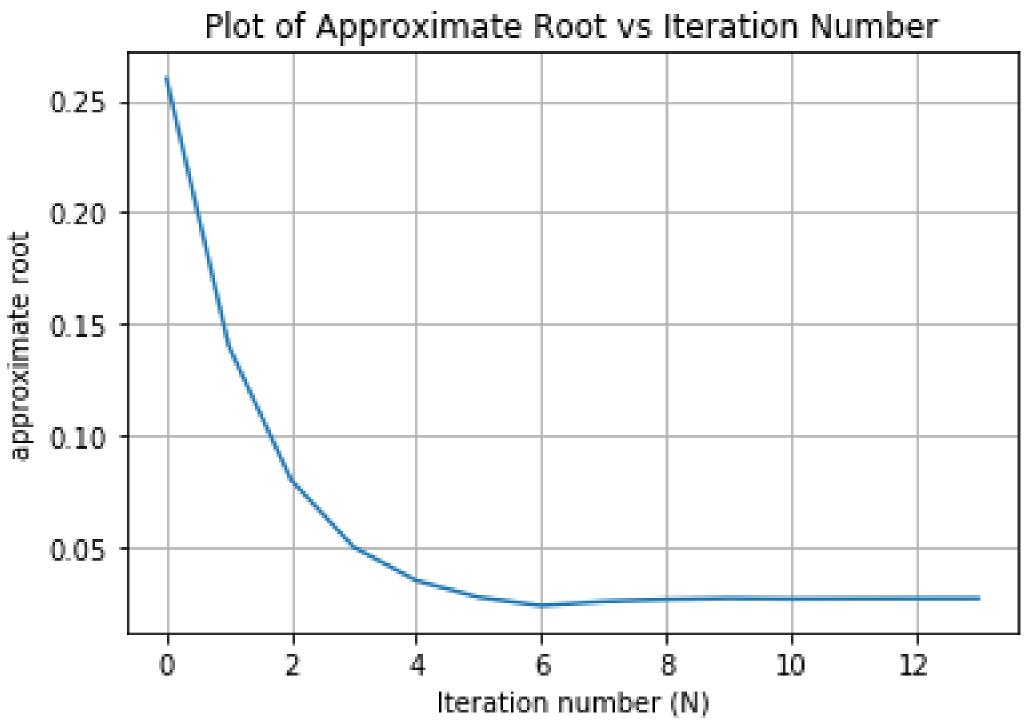
We need to find an estimate value of the root upto 5 decimal points which means $tolerance(\delta) = 10^{-5}$.

In [11]:
```
N=40
a=a0;b=b0;
p=np.zeros(N)
print('n\tpn        \tf(pn)\n')
for i in range (0,N):
    if f(p[i])==0 or ((b-a)/2)<tol:
```

```
    #if f(p[i])==0:
        print('*** The root is {:0.5f} with accuracy {}***'.format(p[i-1],tol))
        count = i-1
        break
    p[i] = (a + b)/2
    if (f(p[i])*f(b))<0:
        a = p[i]
    else:
        b = p[i]
    print(str(i+1)+'\t{:0.5f} \t{:0.5f}\n'.format(p[i],f(p[i])))
    count = i-1

if i == N-1:
    print('***We have reached the maximum number of {} iterations***\n'.format(N))
er = np.zeros(N)
for i in range(0,count):
    er[i] = abs(p[i+1]-p[i])
plt.plot(np.arange(count),p[:count])
plt.grid(1)
plt.xlabel('Iteration number (N)')
plt.ylabel('approximate root')
plt.title('Plot of Approximate Root vs Iteration Number')
plt.show()
```

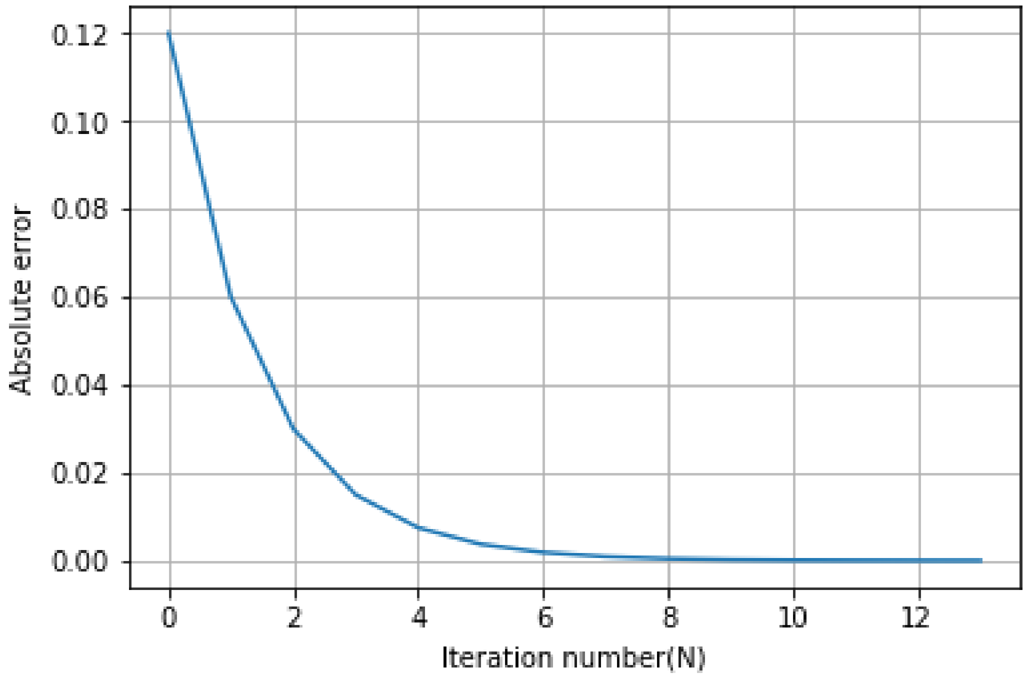
```
n         pn                f(pn)

1         0.26000          -33.15385

2         0.14000          -29.85714

3         0.08000          -24.50000

4         0.05000          -17.00000

5         0.03500          -8.42857

6         0.02750          -0.63636

7         0.02375          5.10526

8         0.02563          2.02439

9         0.02656          0.64706

10         0.02703          -0.00578

11         0.02680          0.31778
```

```
12        0.02691        0.15530

13        0.02697        0.07458

14        0.02700        0.03436

15        0.02702        0.01428

*** The root is 0.02702 with accuracy 1e-05***


/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:3
  This is separate from the ipykernel package so we can avoid doing imports until
```



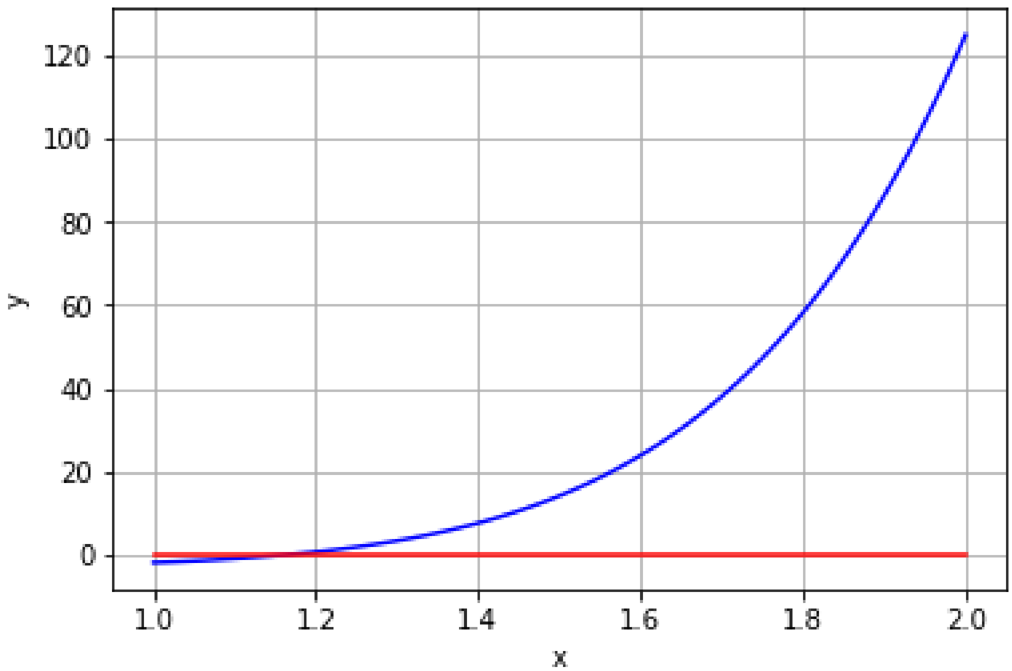
Plot of Absolute error vs Iteration number

```
In [12]: plt.plot(np.arange(count),er[:count])
         plt.xlabel('Iteration number(N)')
         plt.ylabel('Absolute error')
         plt.grid(1)
         plt.show()
```



Problem 4
To find the root of $f(x) = x^7 - 3$ \$ $\forall$\$\$$x \in [1,2]$\$

In [13]:
```
x = np.linspace(1,2,101)
def f(x):
    return x**7-3
def df(x):
    return 7*(x**6)
z = 0*x
plt.plot(x,f(x),'b')
plt.plot(x,z,'r')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(1)
plt.show()
```



Performing **5** iterations of newtons method with initial guess as $p_0 = 1.6$

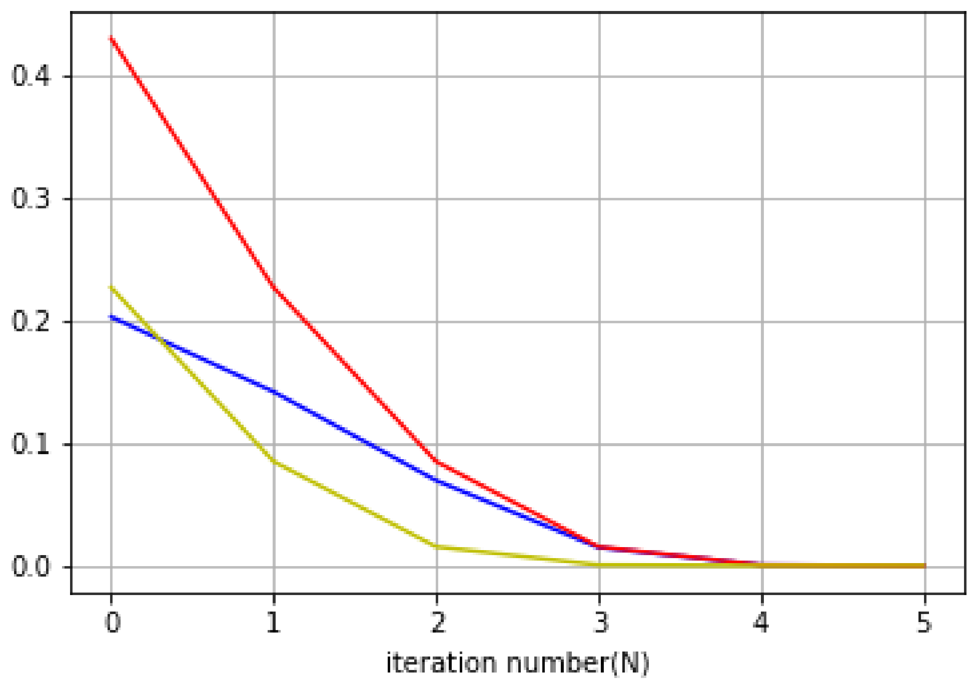
```
In [14]: N = 6
         p = np.zeros(N)
         root = 3**(1/7)
         #taking initial guess of the root as 1.6
         p[0]=1.6
         for i in range(0,N-1):
             p[i+1]= p[i] - f(p[i])/df(p[i])
         print(p)
```

```
[ 1.6         1.39697342  1.25506841  1.18542549  1.17052534  1.16993172]
```

```
In [15]: er = np.zeros(N)
         er2 = np.zeros(N)
         er3 = np.zeros(N)
         print('n\tpn\t|pn-pn-1|\t|pn-1-p|\t|pn-p|\n')
         for i in range(0,N-1):
             er[i]=np.abs(p[i+1]-p[i])
             er2[i]=np.abs(p[i]-root)
             er3[i]=np.abs(p[i+1]-root)
             print(str(i+1)+'\t'+str(p[i])+'\t'+str(er[i])+'\t'+str(er2[i])+
                   '\t'+str(er3[i])+'\n')
         plt.plot(np.arange(N),er,'b')
         plt.plot(np.arange(N),er2,'r')
```

```
plt.plot(np.arange(N),er3,'y')
plt.xlabel('iteration number(N)')
plt.grid(1)
plt.show()
```

```
n        pn          |pn-pn-1|          |pn-1-p|          |pn-p|

1        1.6         0.203026580811          0.430069187241          0.227042606431

2        1.39697341919          0.141905013016          0.227042606431          0.0851375934151

3        1.25506840617          0.0696429124585          0.0851375934151          0.0154946809566

4        1.18542549372          0.0149001533702          0.0154946809566          0.000594527586381

5        1.17052534035          0.000593622444113          0.000594527586381          9.05142268914e-0
```



The above plot indicates that the iterative error $|p_n - p_{n-1}|$(blue) lies between the actual errors $|p_n - p|$(yellow) and $|p_{n-1} - p|$(red) and all converging together to 0

```
In [16]: ratio = np.zeros(len(er))

         for i in range(0,len(er)):
             #ratio[i] = np.abs(p[i+1]-root)/(np.abs(p[i]-root))**2
```

```
        ratio[i]= er3[i]/(er2[i])**2
    print('ratio is {}'.format(ratio))
    def ddf(x):
        return 42*x**5
    a=np.abs(ddf(root)/(2*df(root)))
    print(a)
```

```
ratio is [ 1.22752602  1.65160672  2.13766624  2.47631982  2.56078327         nan]
2.56425419972


/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:5
  """
```

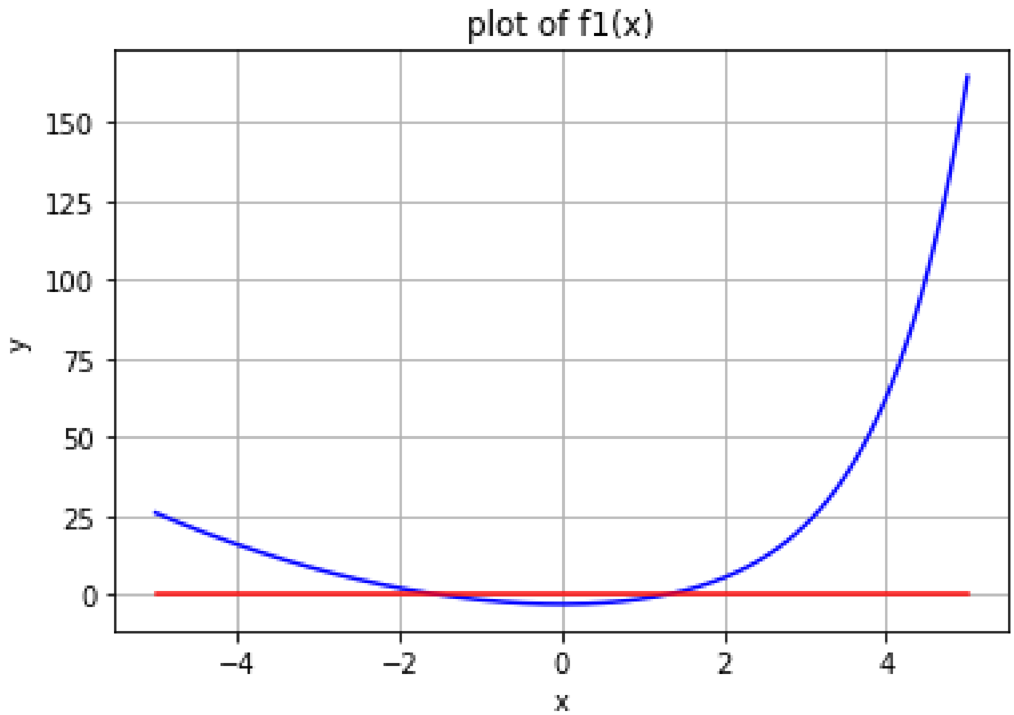
From the above calculation it is clear that the ratio $\frac{|p_n-p|}{|p_{n-1}-p|^2} \rightarrow |f''(p)/2f'(p)|$

Problem 5

We need to find **all** the roots of the functions (using newtons method): $f_1(x) = e^x + x^2 - x - 4$, $f_2(x) = x^3 - x^2 - 10x + 7$, $f_3(x) = 1.05 - 1.04x + ln(x)$ with tolerance(δ) = 10^{-6}

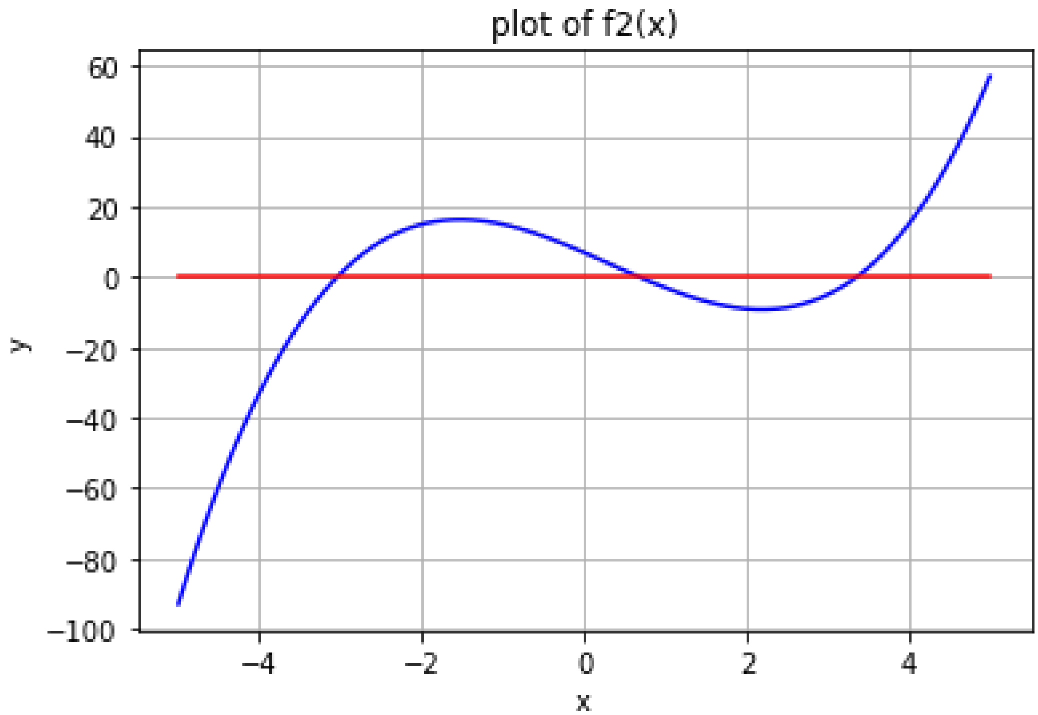
First lets plot the functions to find out the number of roots

```
In [17]: x = np.linspace(-5,5,1001)
         tol = 10**-6
         def f1(x):
             return (np.e)**x+x**2-x-4
         def f2(x):
             return x**3-x**2-10*x+7
         def f3(x):
             return 1.05-1.04*x+np.log(x)
         z=0*x
         plt.plot(x,f1(x),'b')
         plt.plot(x,z,'r')
         plt.xlabel('x')
         plt.ylabel('y')
         plt.grid(1)
         plt.title('plot of f1(x)')
         plt.show()
```



We can observe that $f_1(x)$ has **02** roots and we can take the initial guess as **-4** and **+4** for the two roots

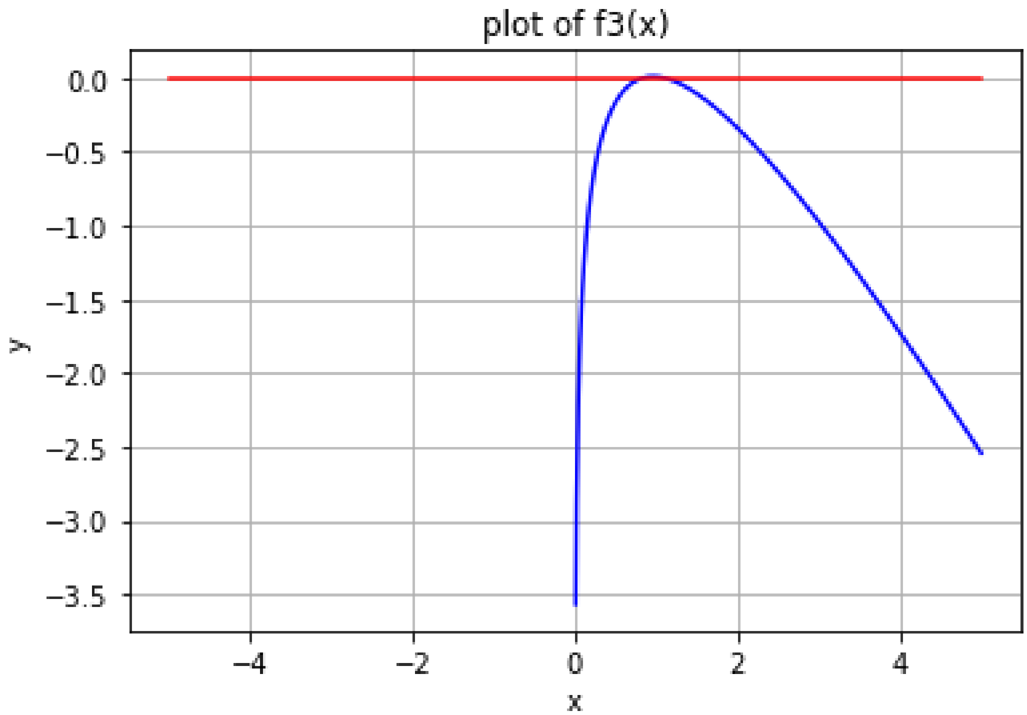
```
In [18]: plt.plot(x,f2(x),'b')
         plt.plot(x,z,'r')
         plt.xlabel('x')
         plt.ylabel('y')
         plt.grid(1)
         plt.title('plot of f2(x)')
         plt.show()
```



We can observe that $f_2(x)$ has **03** roots and we can take the initial guesses as **-4, 0 & +4** for the roots

```
In [19]: plt.plot(x,f3(x),'b')
         plt.plot(x,z,'r')
         plt.xlabel('x')
         plt.ylabel('y')
         plt.title('plot of f3(x)')
         plt.grid(1)
         plt.show()
```

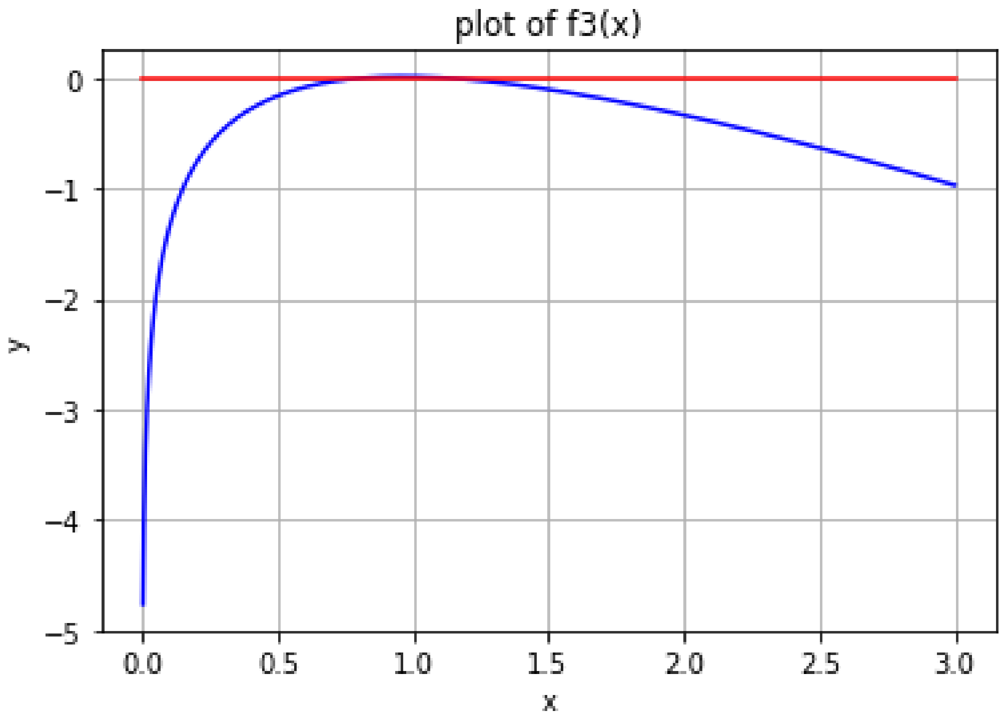
```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:8

/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:8
```



from the above plot its not clear that the function $f_3(x)$ has **02** or **01** roots we can plot the funtion again with a new bound for **x**

```
In [20]: t = np.linspace(0,3,1001)
         z = 0*t
         plt.plot(t,f3(t),'b')
         plt.plot(t,z,'r')
         plt.xlabel('x')
         plt.ylabel('y')
         plt.title('plot of f3(x)')
         plt.grid(1)
         plt.show()
```

```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:8
```



Again, there are either **02** or **01** roots, we will have to evaluate the value of $f_3(1)$ and check its sign

```
In [21]: print(f3(1))
```

```
0.01
```

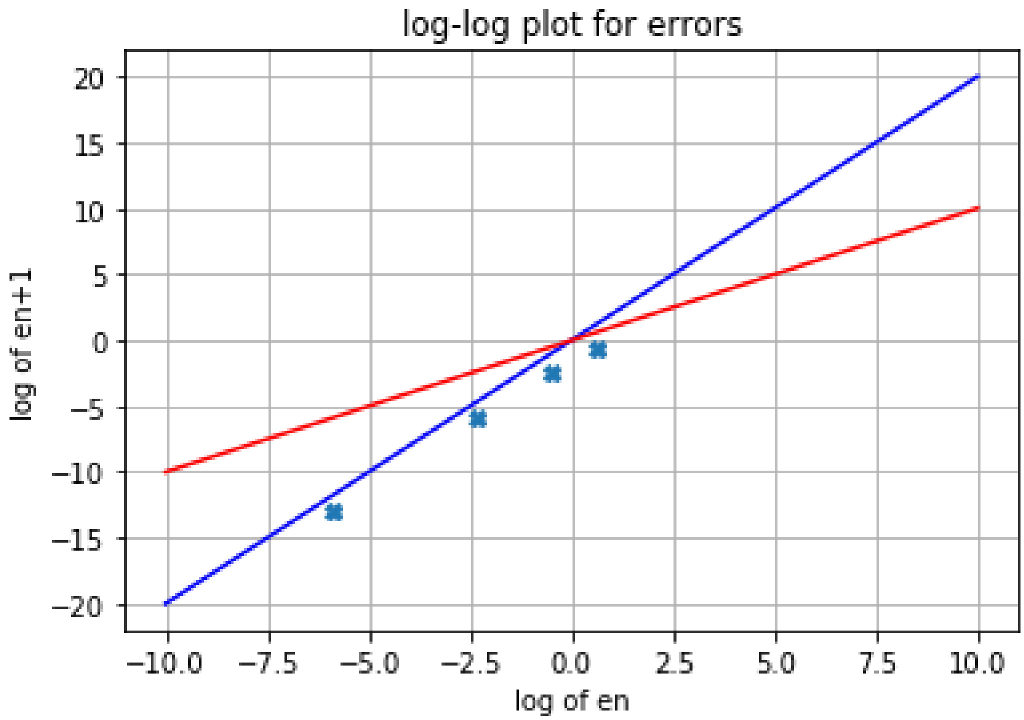
as we can see that the value of $f_3(1)$ is positive therefore there are two roots and we can take the initial guesses for the newtons method as **+0.25** and **+3**

solving for $f_1(x)$:

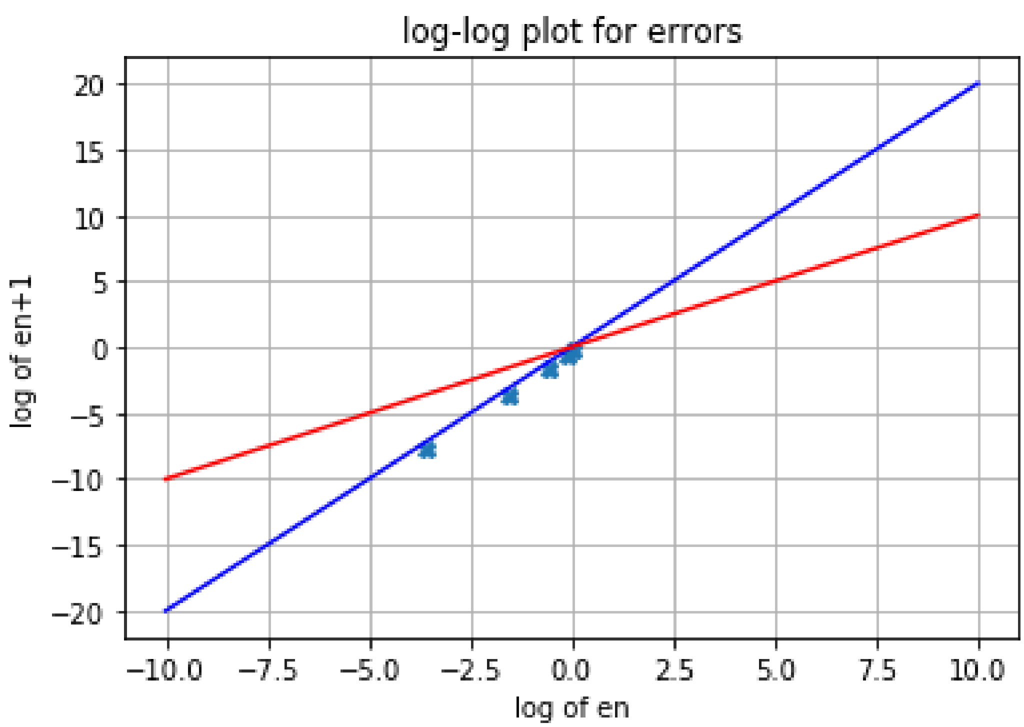
```
In [22]: def df1(x):
             return (np.e)**x+2*x-1

         def newton(ini,N,f,df,tol):
             r=np.zeros(len(ini))
             for k in range(0,len(ini)):
                 p = [0]
                 p[0]=ini[k]
                 i=0
                 while 1 or i<=N:
                     p.append(p[i] - f(p[i])/df(p[i]))
                     if np.abs((p[i+1]-p[i]))<=tol:
                         count = i
```

```
                    print('***The approximate root is '+str(p[i])+' with tolerance '
                          +str(tol)+'***\n')
                    break
                i = i+1
                if i==N:
                    count = i
                    print('!! We have reached the maximum number of iterations'+
                          ' and cannot meet the tolerance, the last approximate root was '+
                          str(p[-1])+'\n')
            er = np.zeros(count)
            for j in range(0,count):
                er[j]=np.abs(p[j+1]-p[j])
            erp =np.zeros(count)
            for j in range(0,count-1):
                erp[j] = er[j+1]
            plt.plot(np.log(er),np.log(erp),'X')
            t=np.linspace(-10,10,101) #for plotting y=2x
            plt.plot(t,2*t,'b')
            plt.plot(t,t,'r')
            plt.plot()
            plt.xlabel('log of en')
            plt.ylabel('log of en+1')
            plt.title('log-log plot for errors')
            plt.grid(1)
            plt.show()
            r[k]=p[-1]
        return(r)

    # initial guesses for f1(x) were -4 and +4
    f1_ini =[-4,4]
    root1 = newton(f1_ini,100,f1,df1,tol)
    print('All the roots of f1(x) are'+str(root1))
    #print('x='+str(root1)+'is one root of f1 with f1(x)='+str(f1(root1))+'\n')
```

```
***The approximate root is -1.5070994840769143 with tolerance 1e-06***


/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:3
```




```
***The approximate root is 1.2886780624119591 with tolerance 1e-06***
```



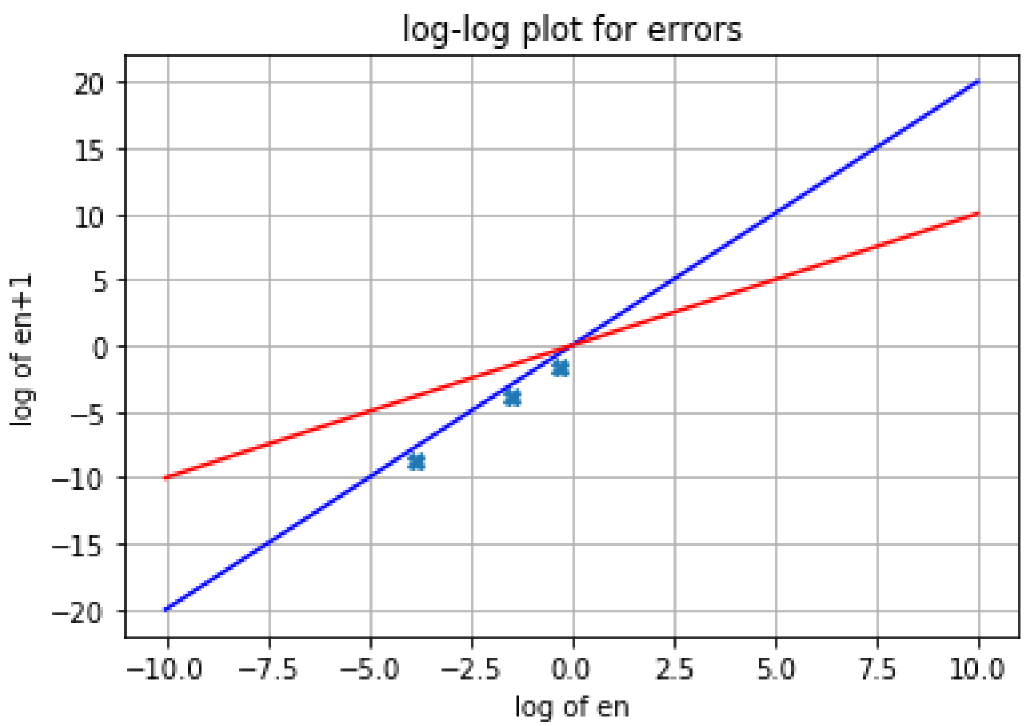
```
All the roots of f1(x) are[-1.50709948  1.28867797]
```

Solving for $f_2(x)$ with initial guess as **-4,0,+4**

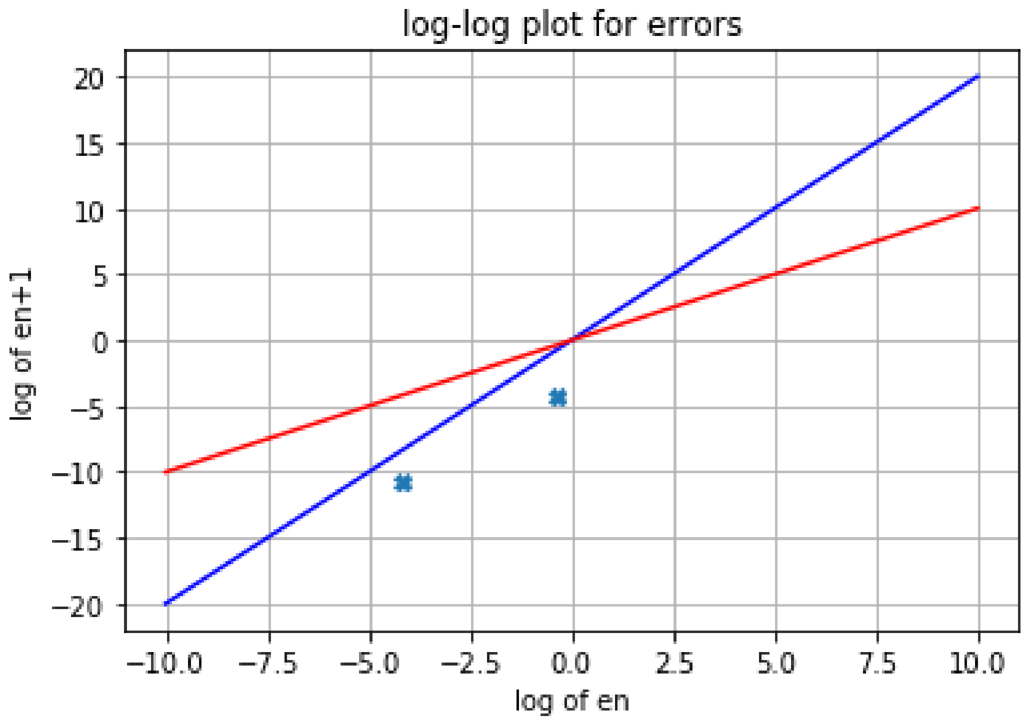
```
In [23]: def df2(x):
             return 3*x**2-2*x-10
         f2_ini =[-4,0,4]
         root1 = newton(f2_ini,100,f2,df2,tol)
         print('All the roots of f2(x) are'+str(root1))
```

```
***The approximate root is -3.042682799149429 with tolerance 1e-06***
```

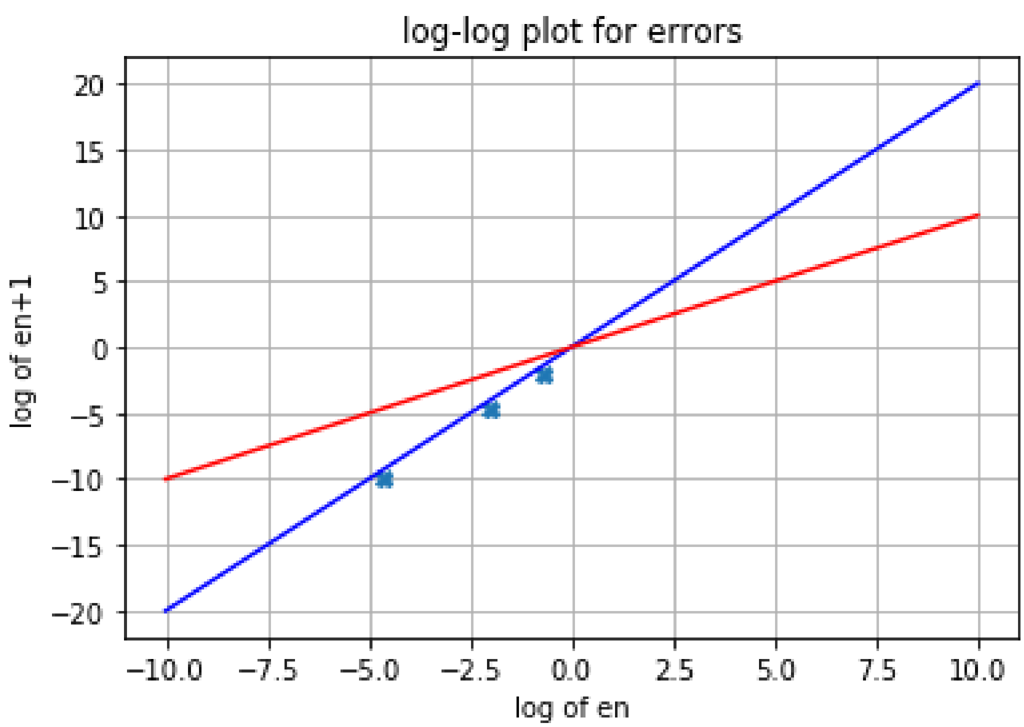
```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:3
```




```
***The approximate root is 0.6852202473404514 with tolerance 1e-06***
```




```
***The approximate root is 3.3574625381598624 with tolerance 1e-06***
```



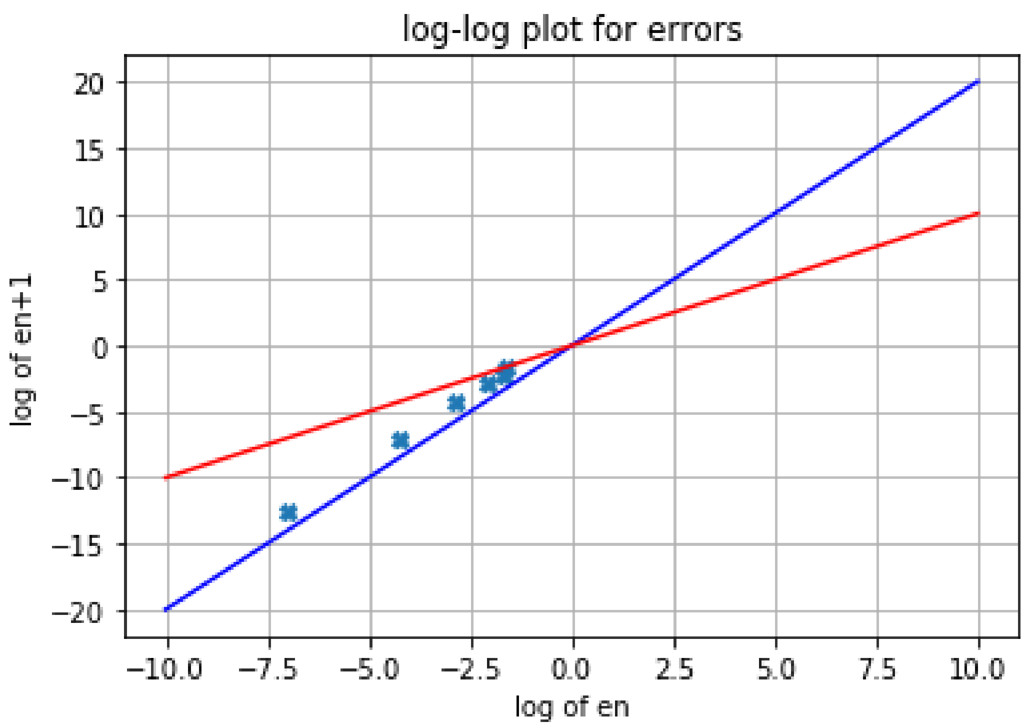
```
All the roots of f2(x) are[-3.04268278  0.68522025  3.35746254]
```

Solving for $f_3(x)$ with initial guesses **+0.25** and **+3**

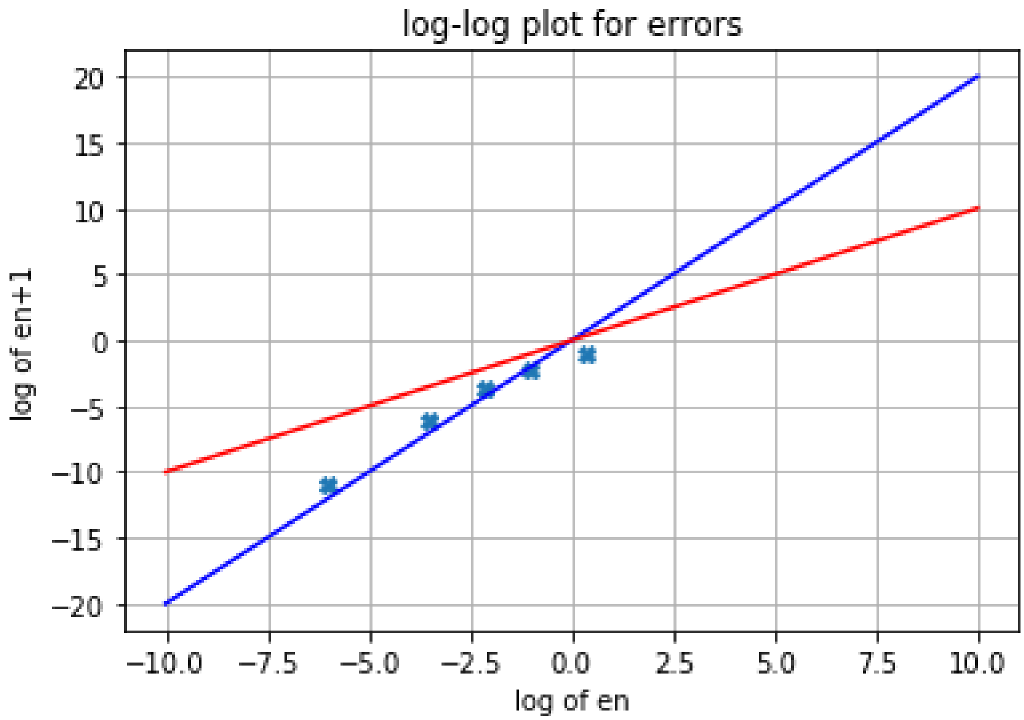
```
In [24]: def df3(x):
             return -1.04+(1/x)
         f3_ini =[+0.25,3]
         root1 = newton(f3_ini,100,f3,df3,tol)
         print('All the roots of f2(x) are'+str(root1))
```

```
***The approximate root is 0.827180908455 with tolerance 1e-06***
```

```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:3
```




```
***The approximate root is 1.1097123047 with tolerance 1e-06***
```




```
All the roots of f2(x) are[ 0.82718091  1.1097123 ]
```

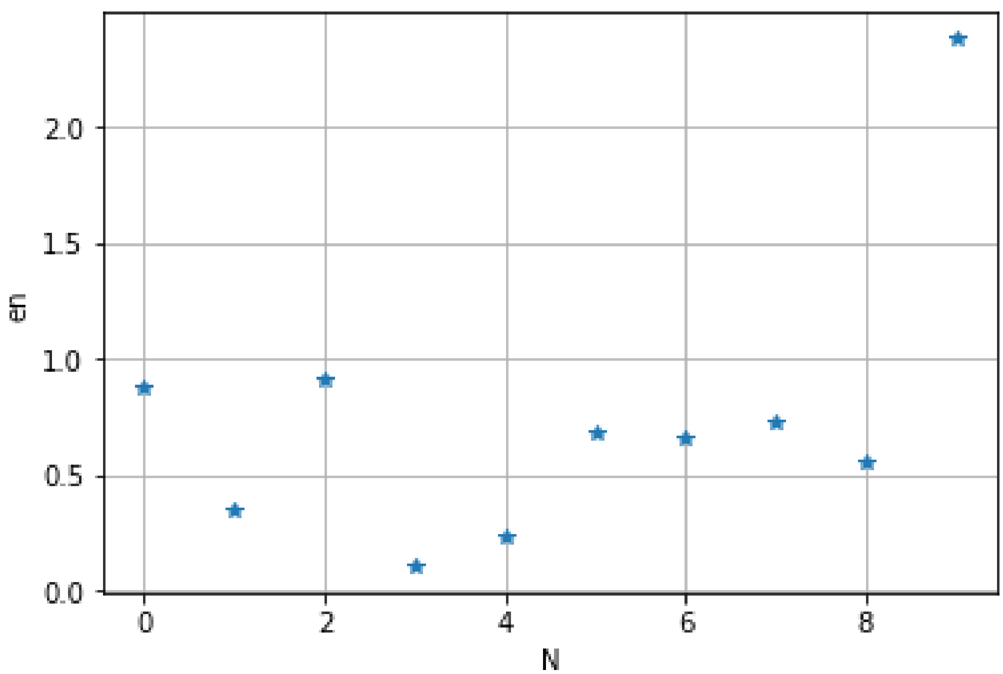
As the log-log plots of the errors are along the blue line which is $y = 2x$. **Therefore, We can say that the Newtons method converges Quadratically**

Problem 6:

We need to find the root for $f(x) = 3x(1 - cos(\pi x))$ with initial guess $p_0 = 0.5$ and need to comment on the order of convergence

```
In [25]: x= np.linspace(0,1,101)
         def f(x):
             return 3*x*(1-np.cos(np.pi*x))
         def df(x):
             return 3*(1-np.cos(np.pi*x))-3*x*(np.pi*(np.sin(np.pi*x)))
         def g(p,f,df,N):
             q=np.zeros(N+1)
             er = np.zeros(N+1)
             erp = np.zeros(N+1)
             q[0]=p
             for i in range(0,N):
                 q[i+1]=q[i]-f(q[i])/(df(q[i]))
```

```
    for i in range(0,N):
        er[i]=np.abs(q[i+1]-q[i])
    for i in range(0,N):
        erp[i]=er[i+1]
    plt.plot(range(0,N),er[:N],'*')
    #plt.plot(np.log(er),np.log(erp),'X')
    plt.xlabel('N')
    plt.ylabel('en')
    plt.grid(1)
    plt.show()
    print('the iterative error is')
    print(er[:-1])
    return(q)
p0=0.5
root = g(p0,f,df,10)
```

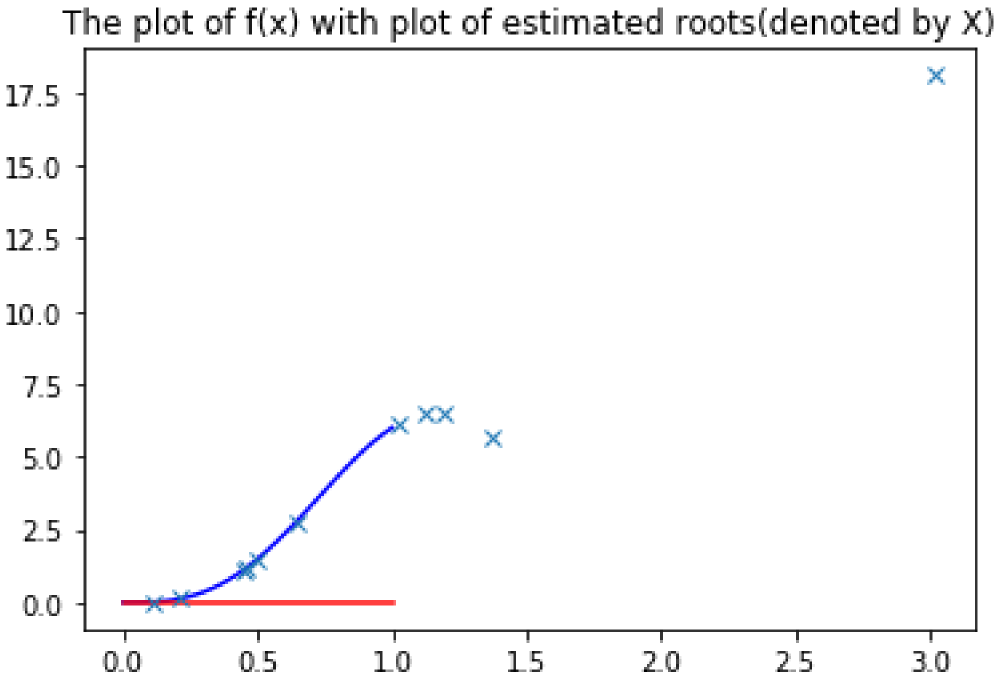



```
the iterative error is
[ 0.8759692   0.35300379  0.91664988  0.10833349  0.23239711  0.67742613
  0.66159408  0.72953811  0.55000021  2.37864661]
```

We can observe that the error does not converge for this problem

```
In [26]: plt.plot(x,f(x),'b')
         plt.plot(x,0*x,'r')
```

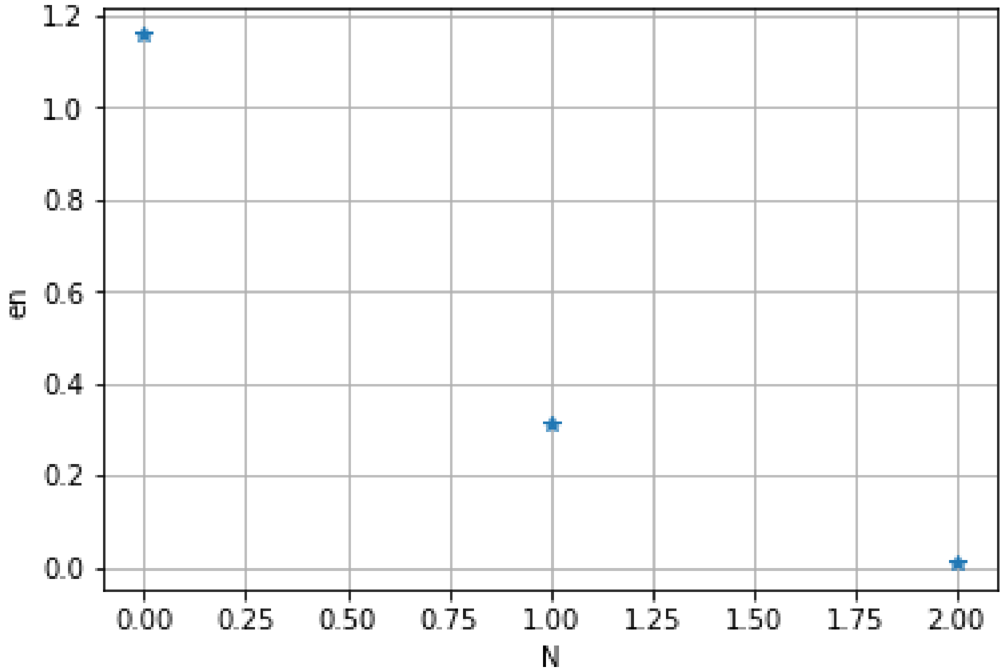
```
plt.plot(root,f(root),'x')
plt.title('The plot of f(x) with plot of estimated roots(denoted by X)')
plt.show()
```



Problem 7:

Given: For $f(x) = sin(x)$ there is a root $x \in [3,4]$ at $x = \pi$. We have to find the root using **03** iterations of newtons method with initial guess as $p_0 = 4$

```
In [27]: x = np.linspace(3,4,101)
         def f(x):
             return np.sin(x)
         def df(x):
             return np.cos(x)
         p0=4
         root = g(p0,f,df,3)
         print('the estimated roots are\n'+str(root))
```

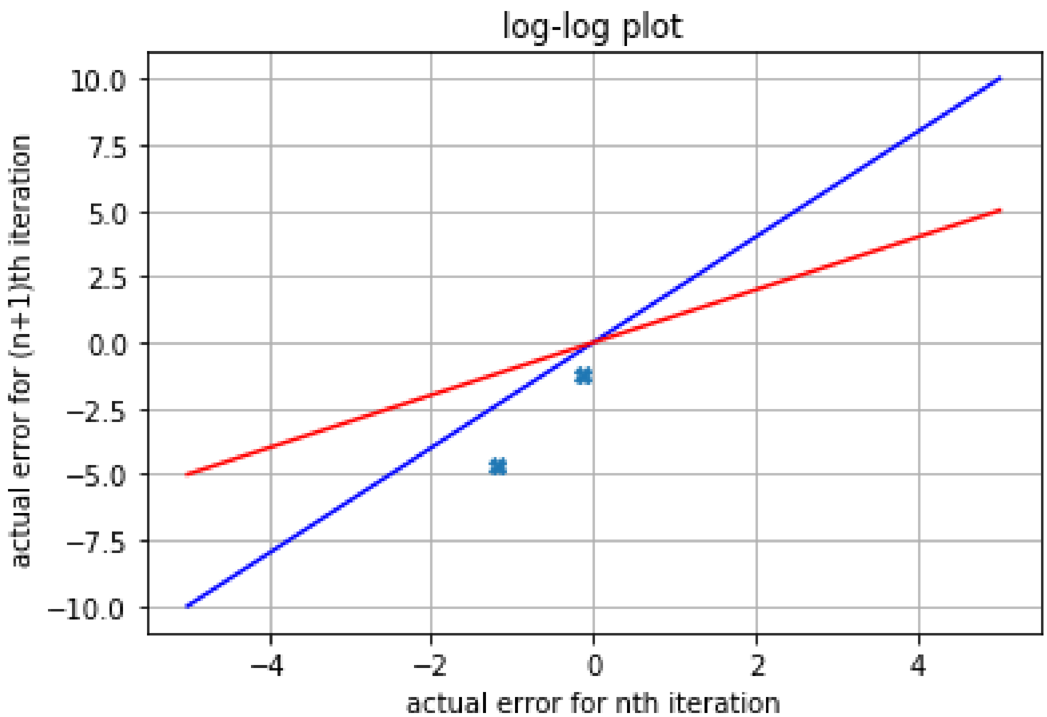



```
the iterative error is
[ 1.15782128  0.30869422  0.00928055]
the estimated roots are
[ 4.          2.84217872  3.15087294  3.14159239]
```

```
In [28]: actual = np.zeros(len(root))
         actualp =np.zeros(len(root))
         er = np.zeros(len(root))
         erp = np.zeros(len(root))
         for i in range(0,len(root)-1):
             er[i]=np.abs(root[i+1]-root[i])
             actual[i]=np.abs(root[i]-np.pi)

         for i in range(0,len(root)-1):
             actualp[i]=actual[i+1]
             erp[i]=er[i+1]

         t=np.linspace(-5,5,101)
         plt.plot(t,2*t,'b') #for plotting y=2x
         plt.plot(t,t,'r')#for plotting y=x
         plt.plot(np.log(actual),np.log(actualp),'X')

         plt.xlabel('actual error for nth iteration')
         plt.ylabel('actual error for (n+1)th iteration ')
```

```python
plt.title('log-log plot')
plt.grid(1)
plt.show()

print('log(actual)');print(np.log(actual))
print('log(actualp)');print(np.log(actualp))

slope = (np.log(actualp[0])-np.log(actualp[1]))/(np.log(actual[0])-np.log(actual[1]))
print('\nApproximate slope(m) is '+str(slope))
print('actualp/(actual^m)');print(actualp/actual**slope)
```

```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:1
  app.launch_new_instance()
```




```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:2


log(actual)
[-0.15267653 -1.20592826 -4.6798629         -inf]
log(actualp)
[-1.20592826 -4.6798629         -inf         -inf]

Approximate slope(m) is 3.298294733
```

```
actualp/(actual^m)
[ 0.49541669  0.49541669  0.                 nan]


/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:2
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:2
```

The apparent order of convergence of actual errors is the approximate slope **3.29**

```
In [29]: def ddf(x):
             return -np.sin(x)
         def dddf(x):
             return -np.cos(x)
         print('The last estimated root is '+str(root[-1])+
               ' and the corresponding value of f(x) is '+str(f(root[-1]))+'\n'+
               ' and df(x) is '+str(df(root[-1]))+'\n'+
               ' and ddf(x) is '+str(ddf(root[-1]))+'\n'+
               'and dddf(x) is '+str(dddf(root[-1])))
```

```
The last estimated root is 3.14159238716 and the corresponding value of f(x) is 2.66426734575e-0
 and df(x) is -1.0
 and ddf(x) is -2.66426734575e-07
and dddf(x) is 1.0
```

As Calculated above $f''(x) = 0$ therefore our convergence is higher than 02 and now \$|e_{n+1}|$\approx |\frac{f'''(p)}{3f'(p)}||e\_\{n\}|^3$ $whereistheapproximateroot$.

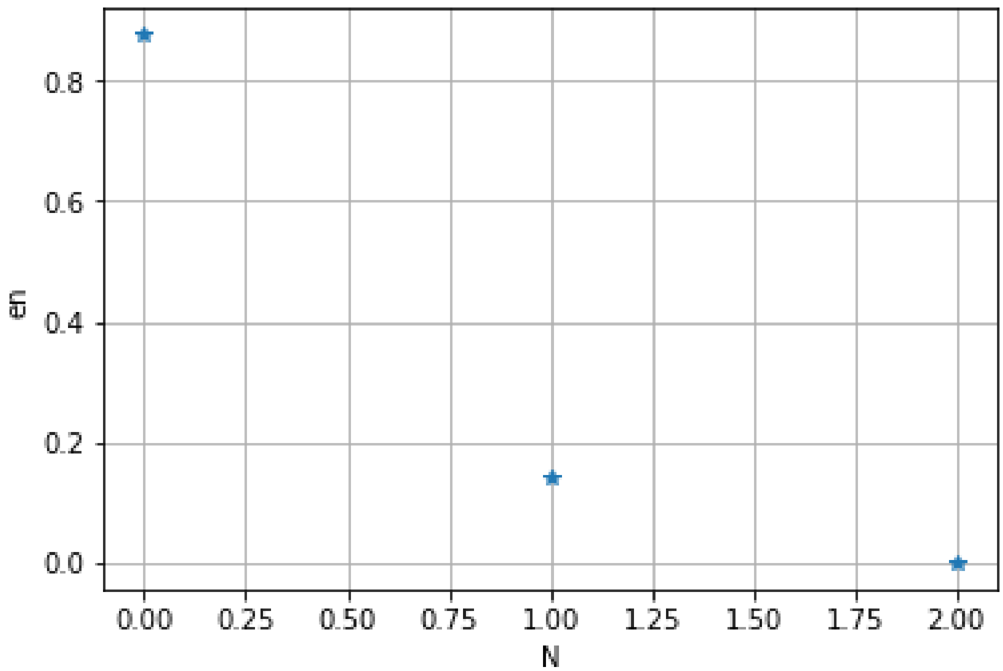
Problem 8:
$a)$ The function is $f(x) = x^4 - 18x^2 + 45 = 0$ has a root in the interval $x \in [1,2]$

```
In [30]: a= 1; b=2;
         x  = np.linspace(a,b,101)
         def f(x):
             return x**4-18*x**2+45
         def df(x):
             return 4*x**3-36*x
         print('f({})= '.format(a)+str(f(a))+
               ', f({})= '.format(b)+str(f(b))+
               ' and f({})*f({}) = '.format(a,b)+str(f(a)*f(b))+'\n')
         if f(a)*f(b)<0:
             print('Then there exists a root in the interval [{},{}]'.format(a,b))
```

```
f(1)= 28, f(2)= -11 and f(1)*f(2) = -308

Then there exists a root in the interval [1,2]
```

Now, estimating root using newtons method for **3** iterations with $p_0 = 1$

```
In [31]: p0=1;r=3**0.5
         root = g(p0,f,df,3)
         print('the estimated roots are\n'+str(root))
```




```
the iterative error is
[ 0.875       0.14396368  0.00101448]
the estimated roots are
[ 1.          1.875       1.73103632  1.73205081]
```

```
In [32]: actual = np.zeros(len(root))
         actualp =np.zeros(len(root))
         for i in range(0,len(root)-1):
             actual[i]=np.abs(root[i]-r)
         for i in range(0,len(root)-1):
             actualp[i]=actual[i+1]

         t=np.linspace(-5,5,101)
         plt.plot(t,2*t,'b') #for plotting y=2x
         plt.plot(t,t,'r')#for plotting y=x
         plt.plot(np.log(actual),np.log(actualp),'X')
         plt.xlabel('actual error for nth iteration')
         plt.ylabel('actual error for (n+1)th iteration ')
```

```
plt.title('log-log plot')
plt.grid(1)
plt.show()
print('log(actual)');print(np.log(actual))
print('log(actualp)');print(np.log(actualp))
slope = (np.log(actualp[0])-np.log(actualp[1]))/(np.log(actual[0])-np.log(actual[1]))
print('\nApproximate slope(m) is '+str(slope));print('\n')
print('actualp/(actual^m)');print(actualp/actual**slope);print('\n')
print('The apparent order of convergence is '+str(slope));print('\n')
```

```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:1
  # This is added back by InteractiveShellApp.init_path()
```


```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:1


log(actual)
[-0.31190536 -1.94526601 -6.89337637        -inf]
log(actualp)
[-1.94526601 -6.89337637        -inf        -inf]

Approximate slope(m) is 3.02940465509
```

```
actualp/(actual^m)
[ 0.36774024  0.36774024  0.                 nan]


The apparent order of convergence is 3.02940465509




/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:1
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:2
```

```
In [33]: def ddf(x):
             return 12*x**2-36
         def dddf(x):
             return 24*x
         print('The last estimated root is '+str(root[-1])+
               ' and the corresponding value of f(x) is '+str(f(root[-1]))+'\n'+
               ' and df(x) is '+str(df(root[-1]))+' and ddf(x) is '+str(ddf(root[-1]))+'\n'+
              ' and dddf(x) is '+str(dddf(root[-1])))
```

```
The last estimated root is 1.73205080792 and the corresponding value of f(x) is -1.44640353028e-
 and df(x) is -41.5692193817 and ddf(x) is 1.44640281974e-08
 and dddf(x) is 41.56921939
```

As Calculated above $f''(x) = 0$ therefore our convergence is higher than 02 and now $|e_{n+1}| \approx |\frac{f'''(p)}{3f'(p)}||e_{n}|^3$ $whereistheapproximateroot$

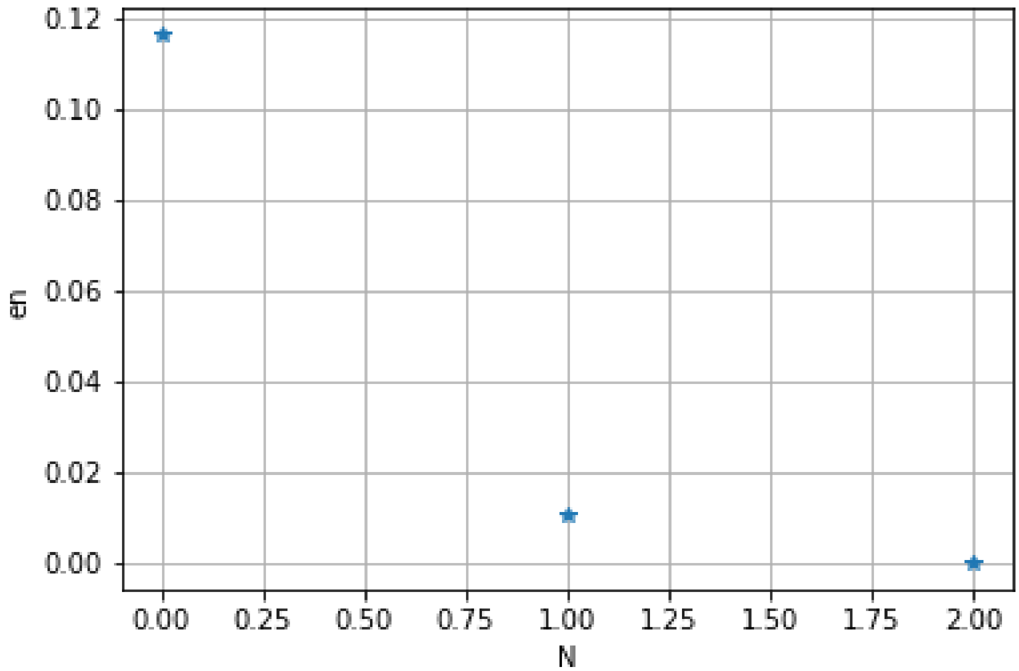
$b)$ The function is $f(x) = x^4 - 18x^2 + 45 = 0$ has a root in the interval $x \in [3,4]$

```
In [34]: a= 3; b=4;
         x  = np.linspace(a,b,101)
         def f(x):
             return x**4-18*x**2+45
         def df(x):
             return 4*x**3-36*x
         print('f({})= '.format(a)+str(f(a))+', f({})= '.format(b)+str(f(b))+
               ' and f({})*f({}) = '.format(a,b)+str(f(a)*f(b))+'\n')
         if f(a)*f(b)<0:
             print('Then there exists a root in the interval [{},{}]'.format(a,b))
```

```
f(3)= -36, f(4)= 13 and f(3)*f(4) = -468

Then there exists a root in the interval [3,4]
```

Now, estimating root using newtons method for **5** iterations with $p_0 = 4$

```
In [35]: p0=4;r=15**0.5
         root = g(p0,f,df,3)
         print('the estimated roots are\n'+str(root))
```




```
the iterative error is
[  1.16071429e-01   1.08535525e-02   9.16661997e-05]
the estimated roots are
[ 4.          3.88392857  3.87307502  3.87298335]
```

```
In [36]: actual = np.zeros(len(root))
         actualp =np.zeros(len(root))
         for i in range(0,len(root)-1):
             #er[i]=np.abs(root[i+1]-root[i])
             actual[i]=np.abs(root[i]-r)
         for i in range(0,len(root)-1):
             actualp[i]=actual[i+1]
             #erp[i]=er[i+1]
         t=np.linspace(-5,5,101)
         plt.plot(t,2*t,'b') #for plotting y=2x
         plt.plot(t,t,'r')#for plotting y=x
         plt.plot(np.log(actual),np.log(actualp),'X')
         plt.xlabel('actual error for nth iteration')
         plt.ylabel('actual error for (n+1)th iteration ')
         plt.title('log-log plot')
```

```
plt.grid(1)
plt.show()
print('log(actual)');print(np.log(actual))
print('log(actualp)');print(np.log(actualp))
slope = (np.log(actualp[0])-np.log(actualp[1]))/(np.log(actual[0])-np.log(actual[1]))
print('\nApproximate slope(m) is '+str(slope));print('\n')
print('actualp/(actual^m)');print(actualp/actual**slope);print('\n')
print('The apparent order of convergence is '+str(slope));print('\n')
```

```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:1
  if sys.path[0] == '':
```


```
log(actual)
[-2.06343707 -4.51485197 -9.29728584        -inf]
log(actualp)
[-4.51485197 -9.29728584        -inf        -inf]

Approximate slope(m) is 1.9508871651


actualp/(actual^m)
[ 0.61304369  0.61304369  0.                 nan]
```

```
The apparent order of convergence is 1.9508871651
```

```
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:1
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:1
/apps/share64/debian7/anaconda/anaconda3-4.4/lib/python3.6/site-packages/ipykernel_launcher.py:2
```

As the log-log plot of errors is along the y=2x line therefore the order of convergence is **2**

```
In [37]: def ddf(x):
             return 12*x**2-36
         print('The last estimated root is '+str(root[-1])+
               ' and the corresponding value of f(x) is '+str(f(root[-1]))+'\n'+
               ' and df(x) is '+str(df(root[-1]))+'\n'+' and ddf(x) is '+str(ddf(root[-1])))
```

```
The last estimated root is 3.87298335272 and the corresponding value of f(x) is 6.05017703492e-0
 and df(x) is 92.9516012463
 and ddf(x) is 144.000000605
```

c) From the above calculations, we can conclude that the apparent order of convergence for the newtons method depends on the value of the $f''(x)$, $f'''(x)$......

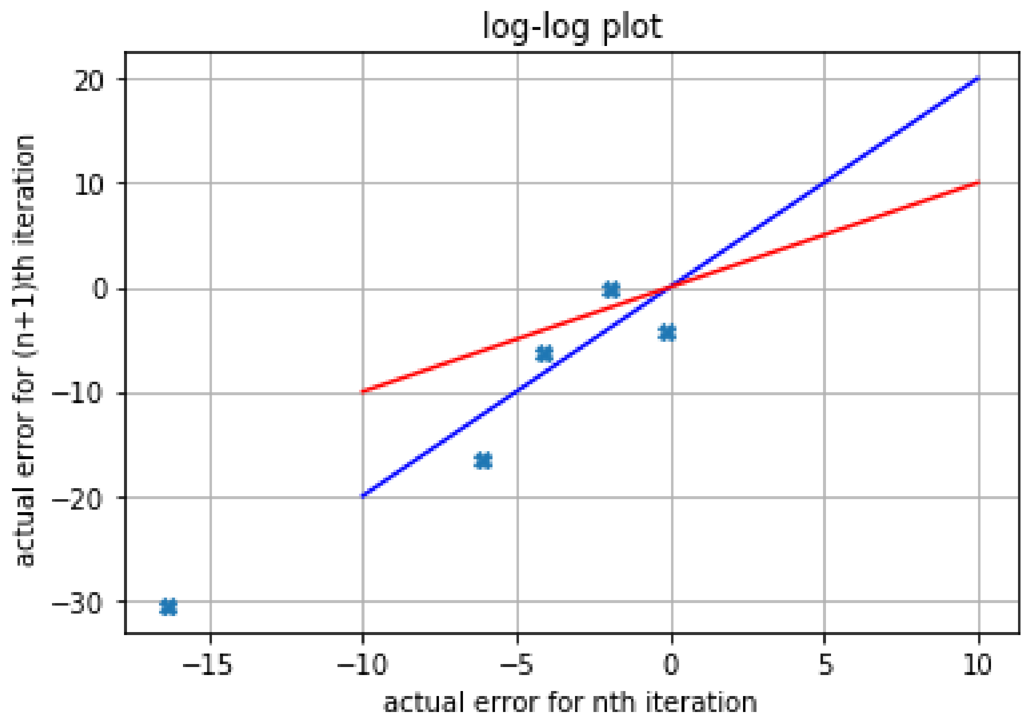
Problem 9:

Given: For $f(x) = sin(x)$ there is a root $x \in [3,4]$ at $x = \pi$. We have to find the root using **05** iterations of secants method with initial guess as $ p_0=3$ and $p_1 = 4 $

```
In [38]: N=5
         p = np.zeros(2+N)
         p[0]=3
         p[1]=4
         def f(x):
             return np.sin(x)
         for i in range(0,N):
             p[i+2]=p[i+1]-(f(p[i+1]))/((f(p[i+1])-f(p[i]))/(p[i+1]-p[i]))
         print(p)
         r= np.pi
         root = p
```

```
[ 3.          4.          3.15716279  3.1394591   3.14159273  3.14159265
  3.14159265]
```

In [39]:
```
actual = np.zeros(len(root))
actualp =np.zeros(len(root))

for i in range(0,len(root)-1):
    actual[i]=np.abs(root[i]-r)
for i in range(0,len(root)-1):
    actualp[i]=actual[i+1]

t=np.linspace(-10,10,101)
plt.plot(t,2*t,'b') #for plotting y=2x
plt.plot(t,t,'r')#for plotting y=x
plt.plot(np.log(actual[:N]),np.log(actualp[:N]),'X')
plt.xlabel('actual error for nth iteration')
plt.ylabel('actual error for (n+1)th iteration ')
plt.title('log-log plot')
plt.grid(1)
plt.show()
print('log(actual)');print(np.log(actual[:N]))
print('log(actualp)');print(np.log(actualp[:N]))
```




```
log(actual)
[ -1.95480098  -0.15267653  -4.16240037  -6.1499655   -16.41387624]
log(actualp)
```

```
[ -0.15267653  -4.16240037  -6.1499655  -16.41387624 -30.50631912]
```

```
In [40]: x_av= sum(np.log(actual[:N]))/N;
         y_av= sum(np.log(actualp[:N]))/N;
         A = 0;B=0;
         for i in range(0,N):
             A = A+((actual[i]-x_av)*(actualp[i]-y_av))
             B = B+(actual[i]-x_av)**2
         m = A/B;
         print('We can observe that the log-log plot of the errors do not follow'+'\n'+
               ' the line y=x or y=2x but rather will be a line with slope {}'.format(m))
```

```
We can observe that the log-log plot of the errors do not follow
 the line y=x or y=2x but rather will be a line with slope 1.945463552046095
```

Therefore, convergence of Secant method $\rightarrow$ Newtons Method

As the Secant Method uses *finite derivative* to **approximate** the value of the *exact derivative* hence it's convergence order also approximates to the order of Newtons method. i.e. ***convergence of secant method*** $\rightarrow 2$