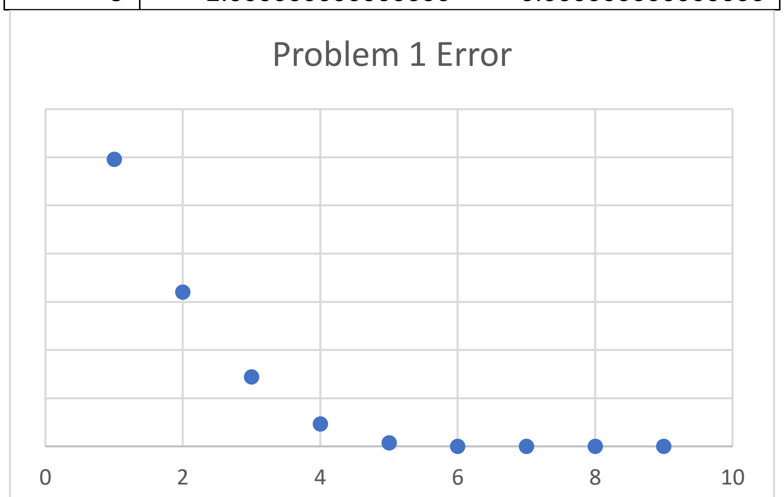
Lab 4

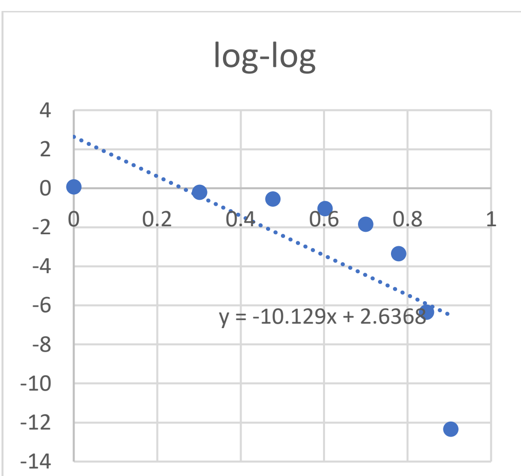
Drake Song

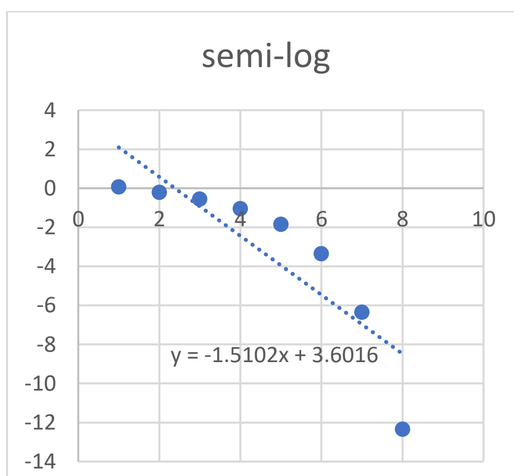
<u>Results</u>

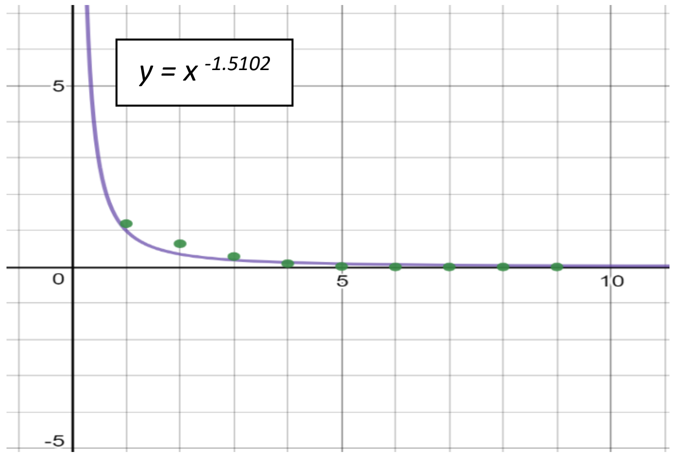
Problem 1

Iteration	Result	Error
1	3.192307692307690	1.192307692307690
2	2.640061241913510	0.640061241913510
3	2.287983298553010	0.287983298553010
4	2.092678794961760	0.092678794961760
5	2.014513253181620	0.014513253181620
6	2.000450360155210	0.000450360155210
7	2.000000455613670	0.000000455613670
8	2.000000000000460	0.000000000000460
9	2.000000000000000	0.000000000000000









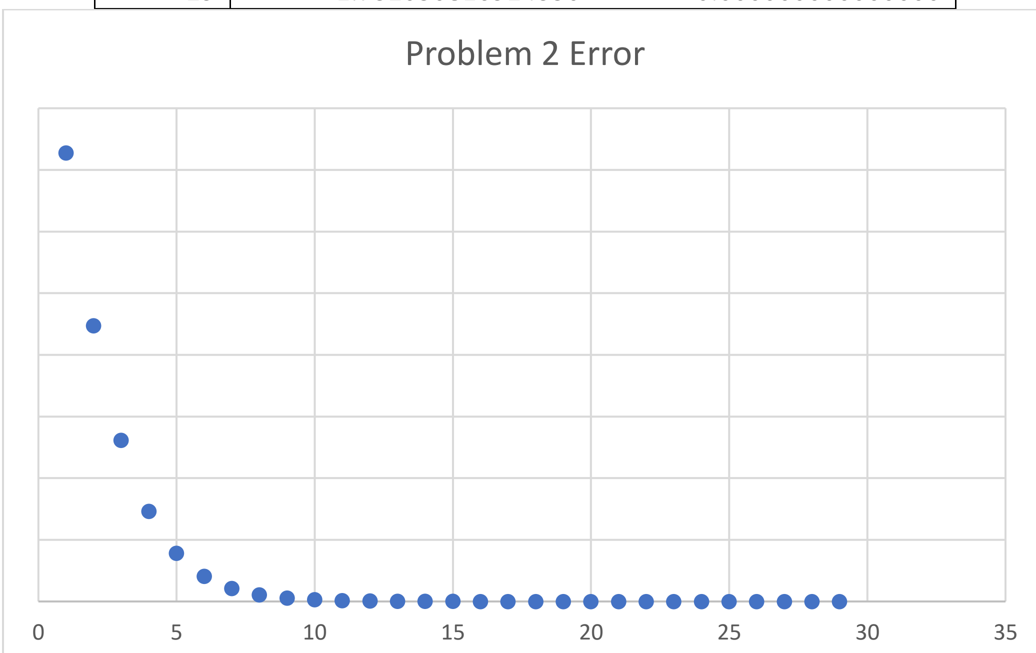
A noted in advance: although the program uses 15 decimal places for accuracy, Excel does not allow any numbers greater than 15 digits because it follows the specifications of IEEE 754; meaning, the numbers used in Excel are only accurate to 14 decimal places.

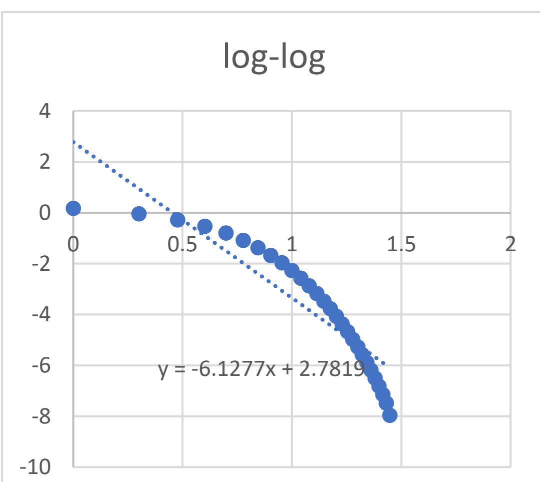
For Problem 1, the program found the root to be at *2* after nine iterations from starting at $x = 4$. From just plotting the error, it looks like the points are not in a linear but a quadratic relationship. In order to check, log-log plot was used but it turned out horribly with no linearity that would help to determine the explicit formula. With the help of semi-log plot, an approximate power for x was determined to be -1.5102. After graphing $y = x^{-1.5102}$, it can be concluded that the explicit formula generally fits the plots very well.

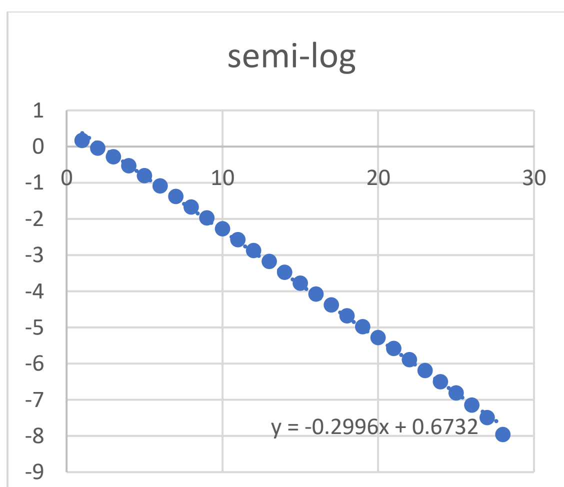
Problem 2

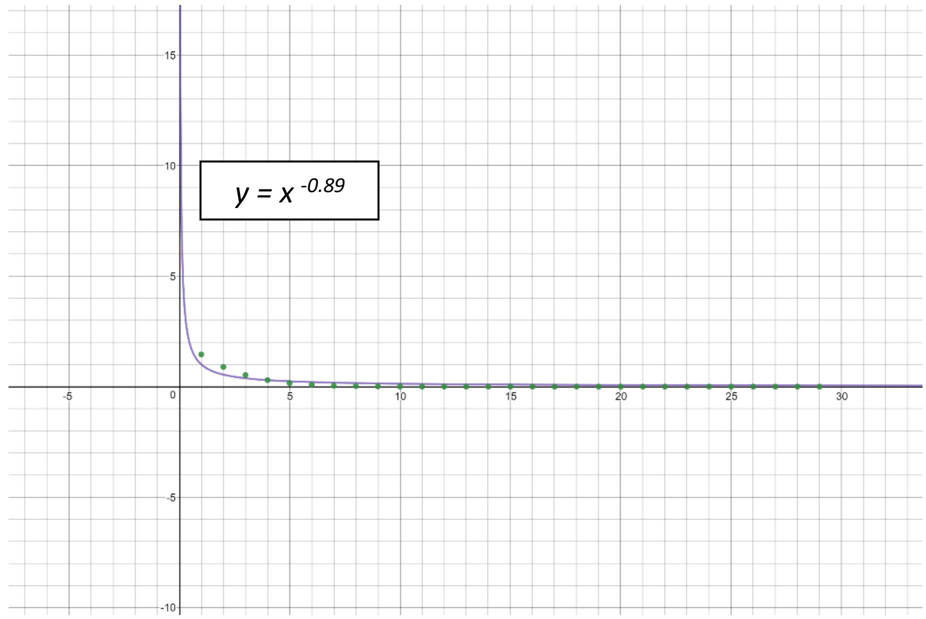
Iteration	Result	Error
1	3.187500000000000	1.455449183075650
2	2.625919117647050	0.893868300722700
3	2.255053618949330	0.523002802024980
4	2.023876541072570	0.291825724148220
5	1.888483369693270	0.156432552768920
6	1.813506612325790	0.081455795401440
7	1.773693381016560	0.041642564092210
8	1.753116514205130	0.021065697280780
9	1.742646943046580	0.010596126122230
10	1.737364982713700	0.005314165789350
11	1.734711958832230	0.002661141907880
12	1.733382403791610	0.001331586867260
13	1.732716861415640	0.000666044491290
14	1.732383898499860	0.000333081575510
15	1.732217369045650	0.000166552121300
16	1.732134092311430	0.000083275387080
17	1.732092450942060	0.000041634017710

18	1.732071629506290	0.000020812581940
19	1.732061218601680	0.000010401677330
20	1.732056013104280	0.000005196179930
21	1.732053410349800	0.000002593425450
22	1.732052108912250	0.000001291987900
23	1.732051458254510	0.000000641330160
24	1.732051132932430	0.000000316008080
25	1.732050970054230	0.000000153129880
26	1.732050888516660	0.000000071592310
27	1.732050849199480	0.000000032275130
28	1.732050827864740	0.000000010940390
29	1.732050816924350	0.000000000000000









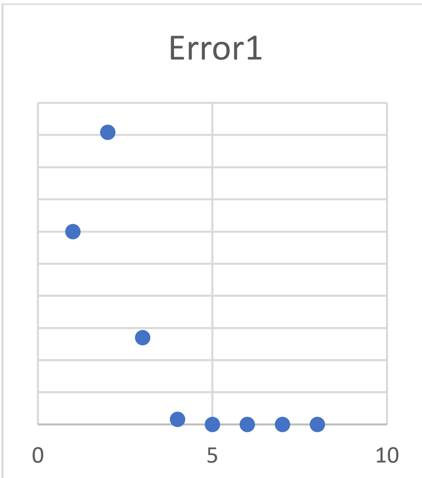
For Problem 2, due to the increase in the number of iterations, it is clearer to see the quadratic convergence that Newton's Method has. After 29 iterations, the program has determined that the root is at $x = 1.73205081692435$. Here the log-log plot also performs terribly. Using semi-log plot as an estimation, $y = x^{-0.89}$ turned out to be fairly close to the relationship the points were demonstrating.

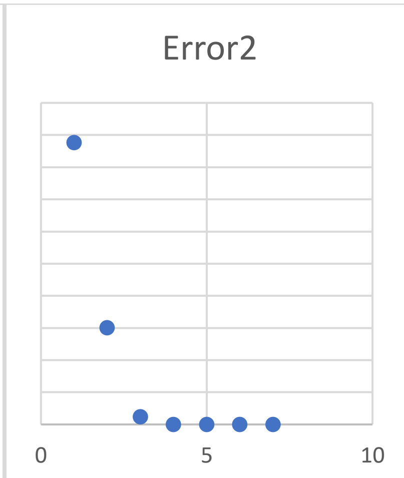
Problem 3

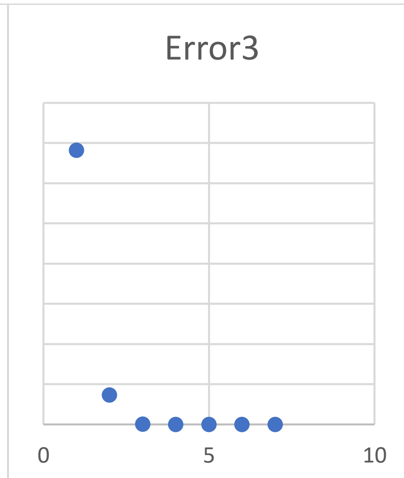
Iter	Root1	Root2	Root3
1	0.215686274509804 - 0.196078431372549j	-0.743589743589744 - 0.217948717948718j	6.066666666666666 + 0.000000000000000j
2	2.209183051160079 + 1.626222329320136j	-1.161088332533062 + 0.145468385345191j	5.762156839576194 + 0.000000000000000j
3	1.773065105491892 + 0.389408194436490j	-1.114429331165062 + 0.006842546966749j	5.725946376553419 + 0.000000000000000j
4	1.417103368381330 + 0.024945284725822j	-1.124007064862002 - 0.000071658883772j	5.725448828664244 + 0.000000000000000j
5	1.398495201492018 + 0.000150646103662j	-1.123983177030080 - 0.000000001856949j	5.725448735301111 + 0.000000000000000j
6	1.398534440706909 - 0.000000001953991j	-1.123983179505812 + 0.000000000000000j	5.725448735301107 + 0.000000000000000j
7	1.398534444204705 + 0.000000000000000j		

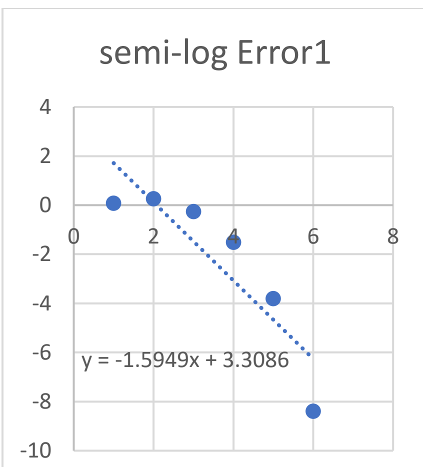
Iteration	Error1	Error2	Error3
1	1.19898980137451000	0.4384071278429670	0.341217931365560
2	1.81707188309601000	0.1501260920563530	0.036708104275090

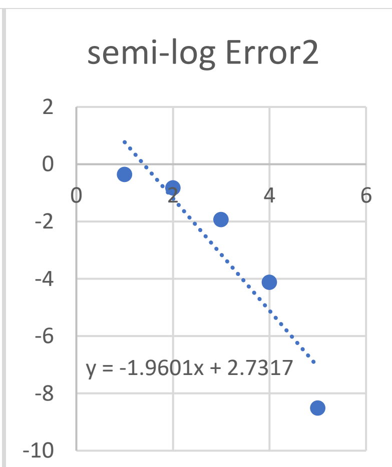
3	0.54028877291547300	0.0117514453200539	0.000497641252310
4	0.03109778408713010	0.0000755347990252	0.000000093363140
5	0.00015567350143110	0.0000000030947533	0.000000000000010
6	0.00000000400658034	0.0000000000000000	0.000000000000000
7	0.00000000000000000		

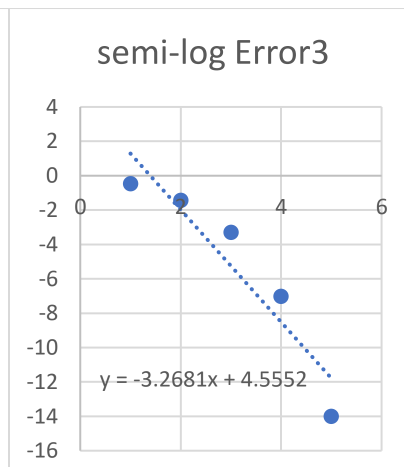


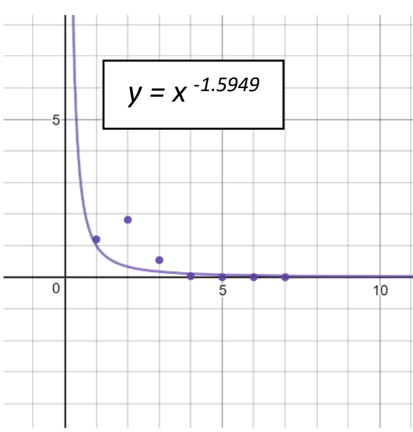


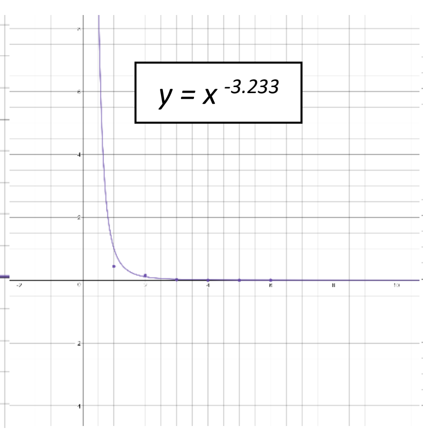


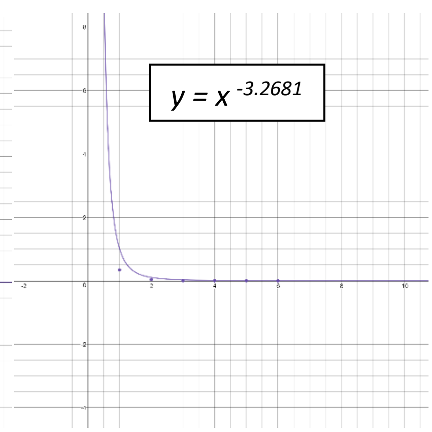










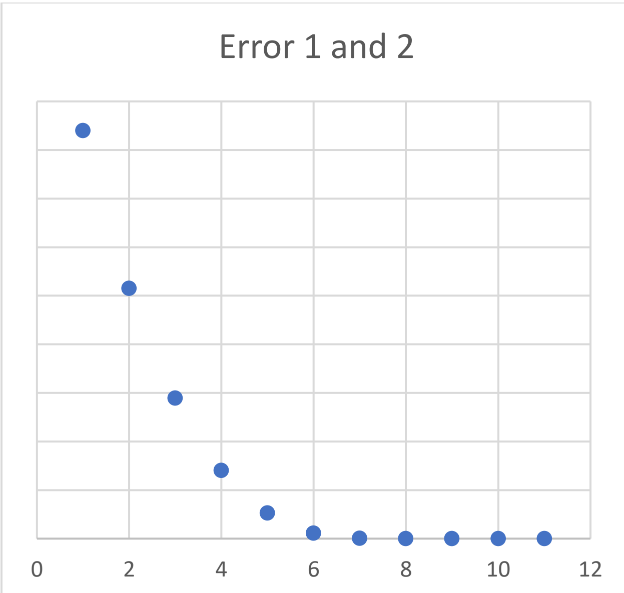


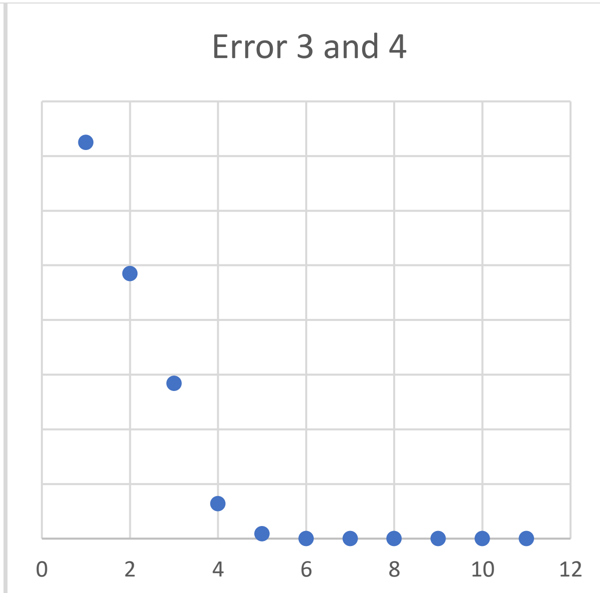
Note: Python uses *j* for imaginary number. All three of the roots show non-linear convergence. For Root 1, the algorithm does take a wild step during Iteration 2; however, the program quickly converges back to the root. Distance Formula was used to determine the Errors.

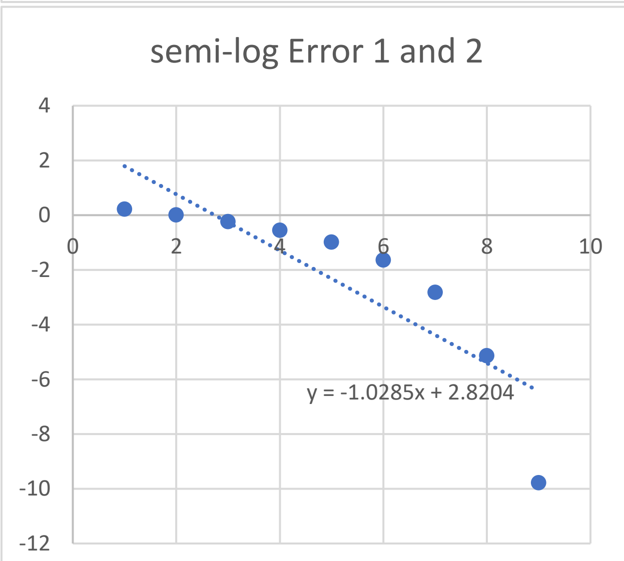
Problem 4

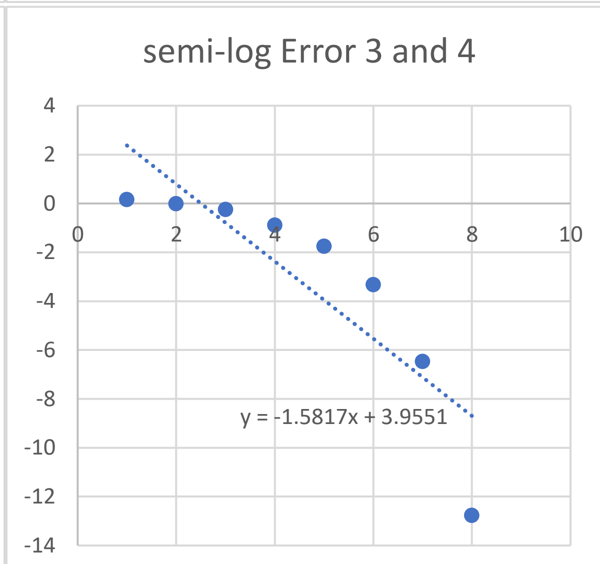
Iter	Root1	Root2	Root3	Root4
1	3.094827586206897j	-3.094827586206897j	1.327955736750146 + 0.416030867792662j	1.327955736750146 - 0.416030867792662j
2	2.444790617429405j	-2.444790617429405j	0.749735789439617 + 0.384336082503261j	0.749735789439617 - 0.384336082503261j
3	1.992690529478473j	-1.992690529478473j	0.192951916008524 + 0.465571947348322j	0.192951916008524 - 0.465571947348322j
4	1.695399712818046j	-1.695399712818046j	-0.116456415973182 + 0.946584963124381j	-0.116456415973182 - 0.946584963124381j
5	1.519559172150619j	-1.519559172150619j	-0.017471389750358 + 1.002671686092732j	-0.017471389750358 - 1.002671686092732j
6	1.437288677533860j	-1.437288677533860j	0.000096097991093 + 1.000464738341655j	0.000096097991093 - 1.000464738341655j
7	1.415729766314224j	-1.415729766314224j	-0.000000134504316 + 0.999999689134646j	-0.000000134504316 - 0.999999689134646j
8	1.414220821153880j	-1.414220821153880j	-0.000000000000125 + 0.999999999999882j	-0.000000000000125 - 0.999999999999882j
9	1.414213562540747j	-1.414213562540747j	-0.000000000000000 + 1.000000000000000j	-0.000000000000000 - 1.000000000000000j
10	1.414213562373095j	-1.414213562373095j	1.000000000000000j	-1.000000000000000j

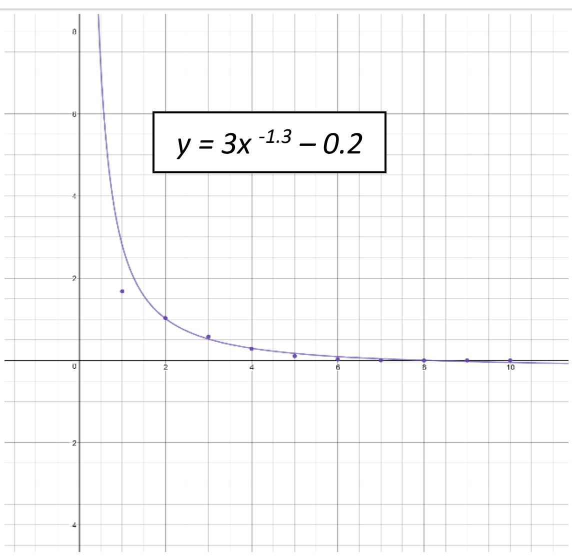
Iteration	Error 1 and 2	Error 3 and 4
1	1.6806140238338000	1.4506847990306500000
2	1.0305770550563100	0.9701266995986030000
3	0.5784769671053800	0.5681934400821830000
4	0.2811861504449500	0.1281220628375240000
5	0.1053456097775200	0.0176744834828914000
6	0.0230751151607700	0.0004745698579679920
7	0.0015162039411301	0.0000003387162224472
8	0.0000072587807900	0.0000000000001719097
9	0.0000000001676499	0.0000000000000000000
10	0.0000000000000000	

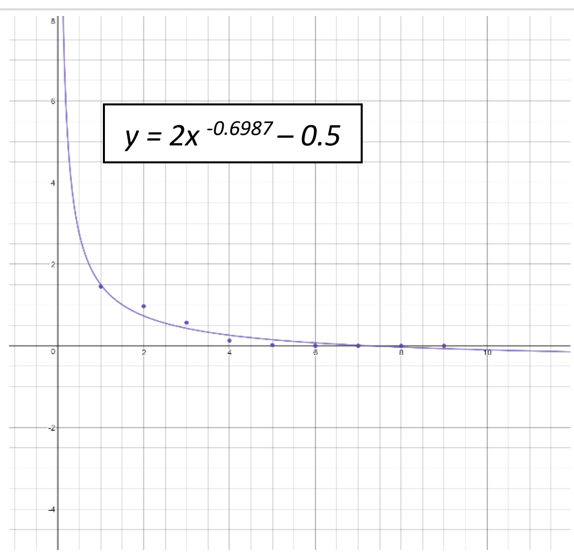

Error 1 and 2
0
2
4
6
8
10
12


Error 3 and 4
0
2
4
6
8
10
12


semi-log Error 1 and 2
4
2
0
-2
-4
-6
-8
-10
-12
0
2
4
6
8
10
y = -1.0285x + 2.8204


semi-log Error 3 and 4
4
2
0
-2
-4
-6
-8
-10
-12
-14
0
2
4
6
8
10
y = -1.5817x + 3.9551


$y = 3x^{-1.3} - 0.2$

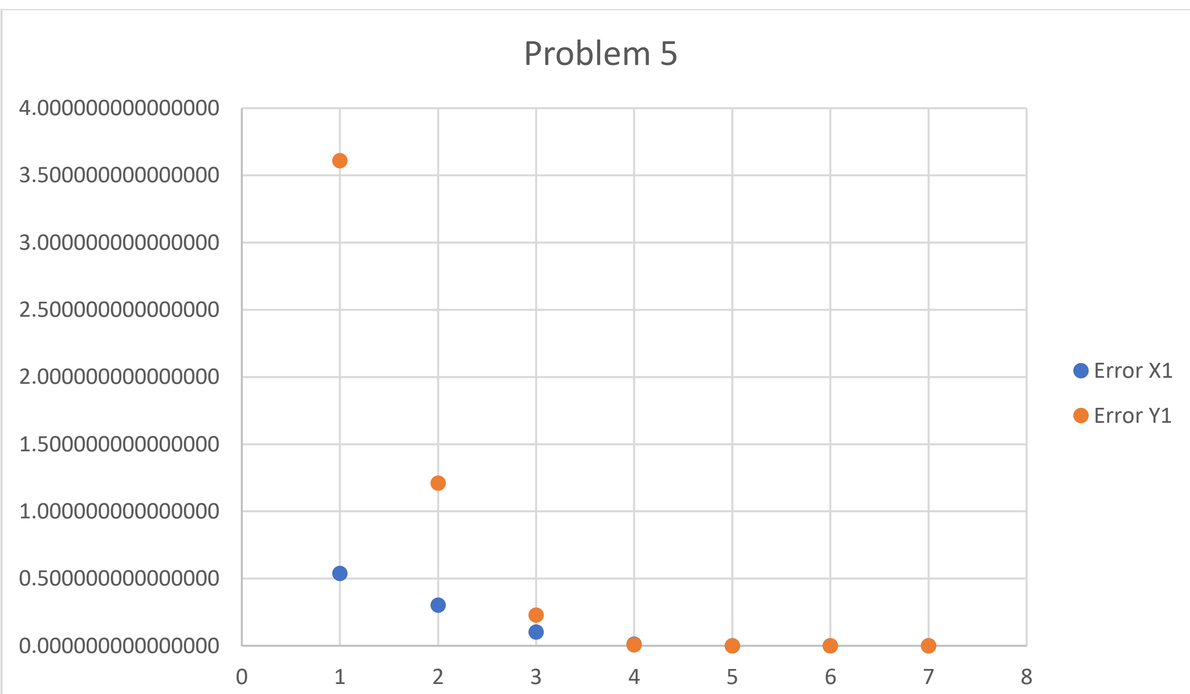

$y = 2x^{-0.6987} - 0.5$

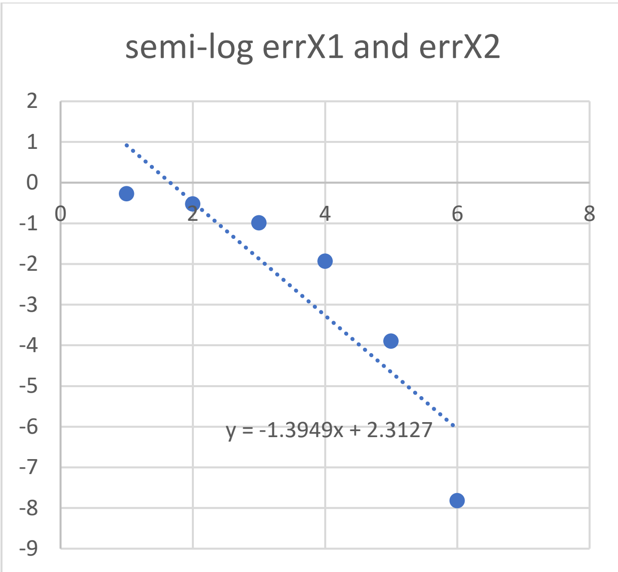
Root 1 and 2 had the same convergence; the roots were the same except for the fact that the sign of the imaginary part was flipped. The same occurred for Root 3 and 4. Convergence is non-linear again. Explicit equations are roughly estimated using trial and error.

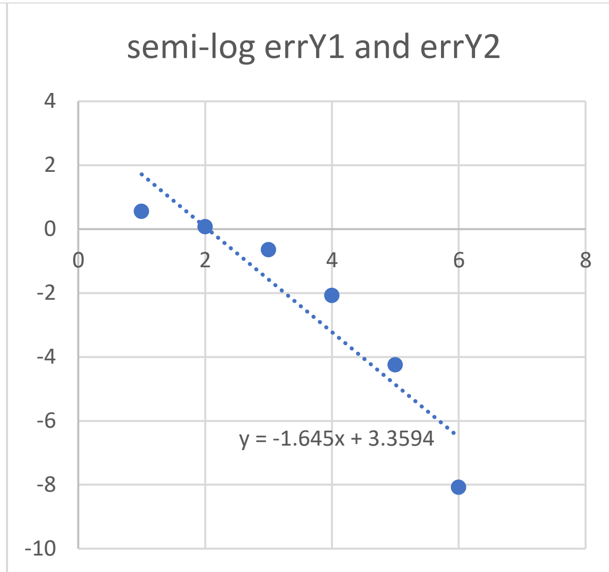
Problem 5

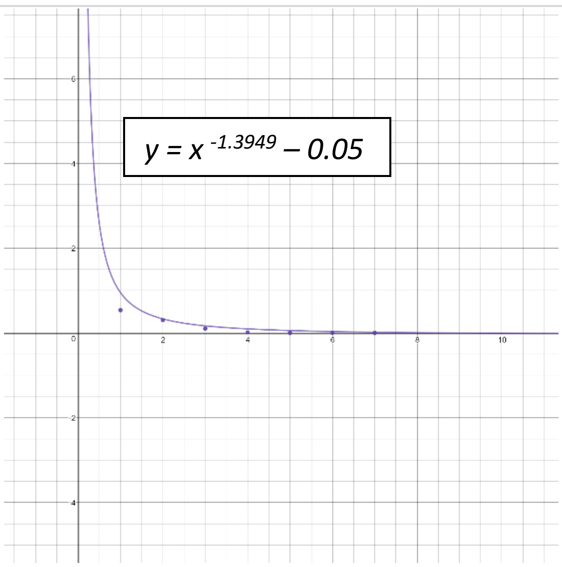
Iter	X 1	Y 1	X 2	Y 2
1	1.969325153374230	5.006134969325150	-1.969325153374230	5.006134969325150
2	1.733112735762740	2.608150203947770	-1.733112735762740	2.608150203947770
3	1.534270194238800	1.627207052894270	-1.534270194238800	1.627207052894270
4	1.442630592381030	1.405788395338620	-1.442630592381030	1.405788395338620
5	1.431016830015220	1.397282385167180	-1.431016830015220	1.397282385167180
6	1.430888557138360	1.397339599659980	-1.430888557138360	1.397339599659980
7	1.430888542038860	1.397339608063080	-1.430888542038860	1.397339608063080

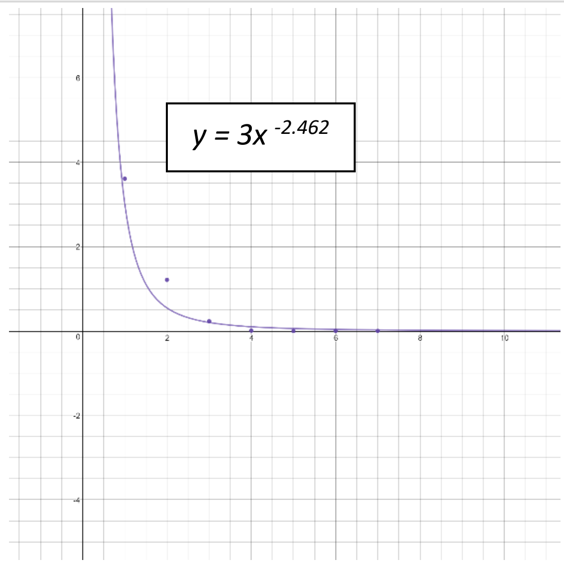
Iter	Error X1	Error Y1	Error X2	Error Y2
1	0.538436611335370	3.608795361262070	0.538436611335370	3.608795361262070
2	0.302224193723880	1.210810595884690	0.302224193723880	1.210810595884690
3	0.103381652199940	0.229867444831190	0.103381652199940	0.229867444831190
4	0.011742050342170	0.008448787275540	0.011742050342170	0.008448787275540
5	0.000128287976360	0.000057222895900	0.000128287976360	0.000057222895900
6	0.000000015099500	0.000000008403100	0.000000015099500	0.000000008403100
7	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000











The two roots are have same values reflected across the y-axis. As observed previously, the convergence rate here is non-linear.

```
# CS 317 Algorithm Anaylsis Lab 4
# Drake Song
# Python 3.6

import numpy as np


def f(p, x, y):
    if p == 1:
        return np.power(x,4) - 6 * np.power(x,2) + 8
    elif p == 2:
        return np.power(x,4) - 6 * np.power(x,2) + 9
    elif p == 3:
        return np.power(x,3) - 6 * np.power(x,2) + 9
    elif p == 4:
        return np.power(x,4) + 3 * np.power(x,2) + 2
    elif p == 5.1:
        return np.power(x,2) + np.power(y,2) - 4
    elif p == 5.2:
        return 3 * y - np.power(x,4)

def fprime(p, x):
    if p == 1:
        return 4 * np.power(x,3) - 12 * x
    elif p == 2:
        return 4 * np.power(x,3) - 12 * x
    elif p == 3:
        return 3 * np.power(x,2) - 12 * x
    elif p == 4:
        return 4 * np.power(x,3) + 6 * x

def newton(p, x):
    return x - f(p,x)/fprime(p,x)


def problem1():
    print("Problem 1")
    different = True
    n = 4
    while different:
        a = newton(1,n)
        print('{0:.15f}'.format(a))
        if n == a:
            different = False
        else:
            n = a
    print("")
```

```
def problem2():
    print("Problem 2")
    different = True
    n = 4
    while different:
        a = newton(2,n)
        print('{0:.15f}'.format(a))
        if n == a:
            different = False
        else:
            n = a
    print("")

def problem3():
    print("Problem 3")
    print("Root 1")
    different = True
    n = 0 + 1j
    while different:
        a = newton(3,n)
        print('{0:.15f}'.format(a))
        if n == a:
            different = False
        else:
            n = a
    print("")

    print("Root 2")
    different = True
    n = -1 - 1j
    while different:
        a = newton(3,n)
        print('{0:.15f}'.format(a))
        if n == a:
            different = False
        else:
            n = a
    print("")

    print("Root 3")
    different = True
    k = 0
    n = 5 + 0j
    while different:
        a = newton(3,n)
        print('{0:.15f}'.format(a))
        k += 1
        if n == a or k == 20:
```

```
            different = False
        else:
            n = a
    print("")

def problem4():
    print("Problem 4")
    print("Root 1")
    different = True
    n = 0 + 4j
    while different:
        a = newton(4,n)
        print('{0:.15f}'.format(a))
        if n == a:
            different = False
        else:
            n = a
    print("")

    print("Root 2")
    different = True
    n = 0 - 4j
    while different:
        a = newton(4,n)
        print('{0:.15f}'.format(a))
        if n == a:
            different = False
        else:
            n = a
    print("")

    print("Root 3")
    different = True
    n = 2 + 0.5j
    while different:
        a = newton(4,n)
        print('{0:.15f}'.format(a))
        if n == a:
            different = False
        else:
            n = a
    print("")

    print("Root 4")
    different = True
    n = 2 - 0.5j
    while different:
        a = newton(4,n)
```

```
        print('{0:.15f}'.format(a))
        if n == a:
            different = False
        else:
            n = a
    print("")

def jacobianInverse(x, y):
    mat = np.matrix([[3, -2*y], [4 * np.power(x,3), 2*x]])
    det = 6*x + 8 * np.power(x,3) * y
    return (1 / det) * mat

def problem5(x,y):
    different = True
    while different:
        a = np.matrix([[x], [y]])
        b = jacobianInverse(x,y)
        c = np.matrix([[f(5.1,x,y)], [f(5.2,x,y)]])
        d = a - b * c
        x_1 = d.item(0)
        y_1 = d.item(1)
        print('[[ {0:.15f}]'.format(x_1))
        print(' [ {0:.15f}]]'.format(y_1))
        print()
        if x == x_1 and y == y_1:
            different = False
        else:
            x = x_1
            y = y_1


problem1()
problem2()
problem3()
problem4()

problem5(2,10)
print()
print()
problem5(-2,10)
```