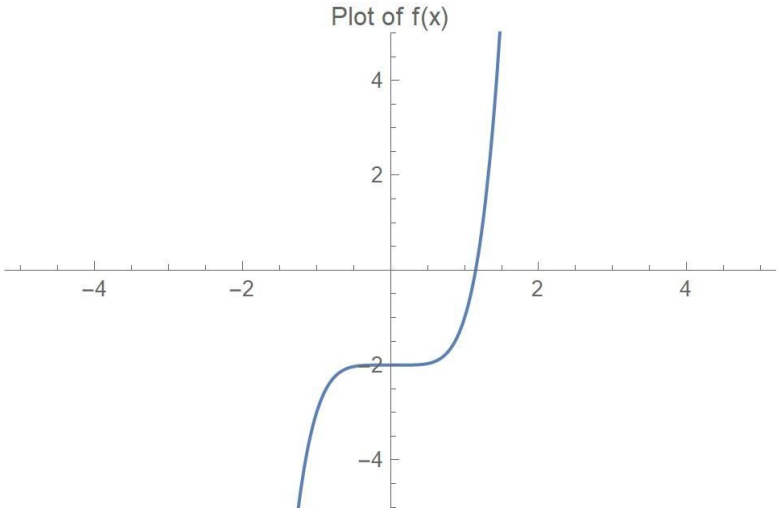
Sarah Mahl
Intro to Numerical Methods
1/22/21
HW2

2. (a) To find the real root of $f(x) = x^5 - 2$, I first plotted the function in Mathematica to be able to set an accurate range for the bisection function. The root can also be found by setting *f(x)* equal to zero, which is equal to $x = 2^{\frac{1}{5}}$, or 1.1487.

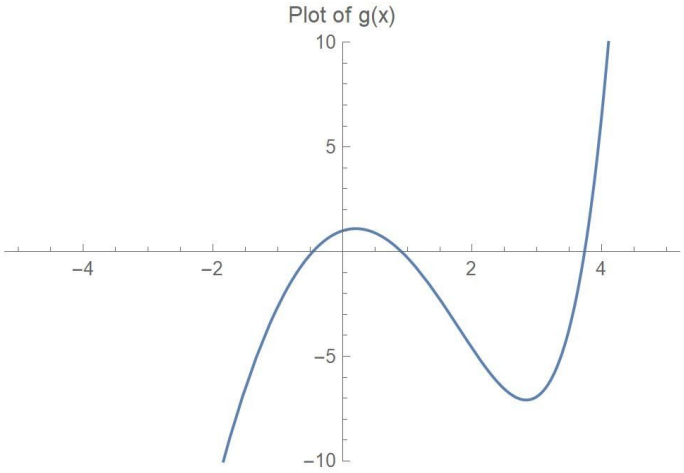


To find the root using the bisection method, I tested four different ranges (a,b) for two different values for N iterations, N=10 and N=100. I kept my first range (0,2) close to exact value to test the accuracy of the method, but then widened the range to test whether that accuracy would hold. A table of the values found as well as the error is displayed below.

(a,b)	N=10	Error	N=100	Error
(0,2)	1.15234375	0.0036437499999999456	1.1486983549970349	1.645002965178e-06
(0,5)	1.142578125	0.006121875000000054	1.1486983549970349	1.645002965178e-06
(-5,5)	1.15234375	0.0036437499999999456	1.1486983549970349	1.645002965178e-06
(-10,10)	1.1328125	0.015887500000000054	1.1486983549970349	1.645002965178e-06

As shown, the accuracy of the bisection method decreases as the range (a,b) widens, however with a larger amount of iterations, this becomes more obsolete and the values found converge to the exact value. The values for the root found for N=100 iterations is very close to the exact value, and although the values for each range at N=100 are the same in the table, I assume that, because of the double precision, the digits began to differ after the sixteenth digit (which is not shown).

Before using the bisection method to find the roots of $g(x) = e^x - 3x^2$, I also plotted the function to get a better idea of which ranges to set and how many zeros I needed to find. That plot is on the next page.



Using Mathematica, the exact roots were found to be $x_1 = -0.458962$, $x_2 = 0.910008$, and $x_3 = 3.73308$. For the first root, I chose the range to be (-1.0, 0.0). For the second root, I chose the range to be (0.5, 1.5), and for the third root, the range was (3.0, 4.0). I also decided to evaluate the root over these ranges at iterations of N = 5, 10, 50, and 100. The values found for the roots using the bisection method are displayed below, as are the errors for each trial.

	Root 1, (a,b)=(-1.0,0.0)	Root 2, (a,b)=(0.5,1.5)	Root 3, (a,b)=(3.0,4.0)
N=5	-0.4375	0.9375	3.6875
N=10	-0.458984375	0.908203125	3.732421875
N=50	-0.4589622675369487	0.9100075724887073	3.7330790286328135
N=100	-0.45896226753694847	0.9100075724887089	3.7330790286328144

	Root 1 Error	Root 2 Error	Root 3 Error
N=5	0.02146199999999998	0.02749199999999996	0.045580000000000176
N=10	2.2375000000018908e-05	0.001804875000000039	0.0006581250000001759
N=50	2.675369487059598e-07	4.275112926999114e-07	9.713671866862228e-07
N=100	2.675369484839152e-07	4.2751129114559916e-07	9.713671857980444e-07

For each root, as N increases, the approximate root converges to the exact value found using Mathematica.

(b) Using Newton's method to find the real root of *f(x)*, I set the initial estimates for the root at x0 = 0.5, 1.0, and 2.0. These were evaluated over N = 10 and N = 100 iterations. The results are displayed in the table below.

	N=10	Error	N=100	Error
x0=0.5	1.1694157867717034	0.02071578677170338	1.148698354997035	1.645002964956177e-06
x0=1.0	1.148698354997035	1.645002964956177e-06	1.148698354997035	1.645002964956177e-06
x0=2.0	1.148698354997035	1.645002964956177e-06	1.148698354997035	1.645002964956177e-06

Once again, the approximate values converge to the exact value for the root. If we were to continue increasing the values for N, it can be assumed that the value for the root would continue to converge. Changing the initial estimate so that it was further away from the exact value for the

root would slow the convergence, but I'd also estimate that the function would eventually converge.

To find the roots of *g(x)* using Newton's Method, I chose the initial estimates for each root to be -0.5, 1.0, and 4.0 over N = 10 and N= 100 iterations. The results for the roots are in the table below alongside the error for each trial.

	Root 1, x0=-0.5	Root 2, x0=1.0	Root 3, x0=4.0
N=10	-0.4589622675369485	0.910007572488709	3.733079028632814
N=100	-0.4589622675369485	0.910007572488709	3.7330790286328144

	Root 1 Error	Root 2 Error	Root 3 Error
N=10	2.6753694853942633e-07	4.2751129103457686e-07	9.713671857980444e-07
N=100	2.6753694853942633e-07	4.2751129103457686e-07	9.713671857980444e-07

The value for the root at each trial is immediately very close to the exact value. I'm not sure if there is a problem with my code, or if the initial estimates I chose are too close to the exact value to really see convergence as N increases. Regardless, the values do equate to the exact value.

(c) Before using Newton's Method to find the complex roots of *f(x)*, I once again used Mathematica to solve for the exact values, which are given in the screenshot below. Those values also include the real root, found in part (b). A complex plot would have also worked to estimate these values, but I used the values below to compare the values I found with my code.

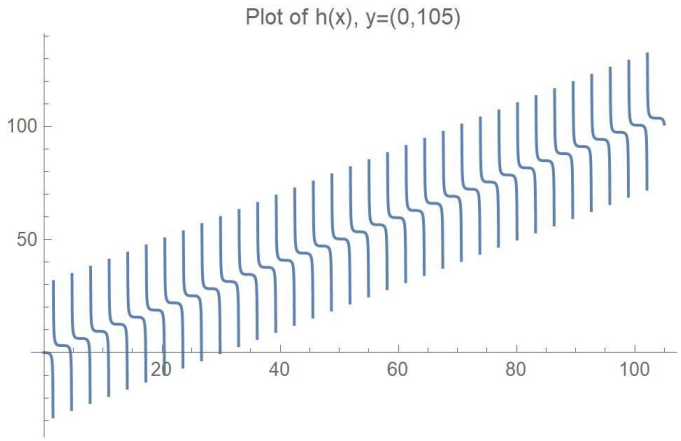
```
Out[3]= {{x → -0.929316 - 0.675188 i},
         {x → 1.1487}, {x → 0.354967 + 1.09248 i},
         {x → 0.354967 - 1.09248 i}, {x → -0.929316 + 0.675188 i}}
```

At N = 100 iterations, I tried to keep my guesses for the roots close to the exact values. Those are given in the table below, as well as the approximate values found.

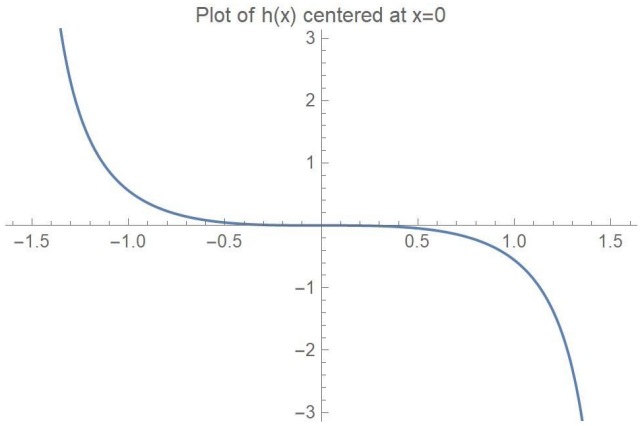
	N=100
R1: x0=0.5+i	0.35496731310463014+1.0924770557774537i
R2: x0=0.5-i	0.35496731310463014-1.092477055777453i
R3: x0=-1.0+0.5i	-0.9293164906031477+0.675187952399881i
R4: x0=-1.0-0.5i	-0.9293164906031477-0.675187952399881i

The values in the table equal the rounded values given by Mathematica.

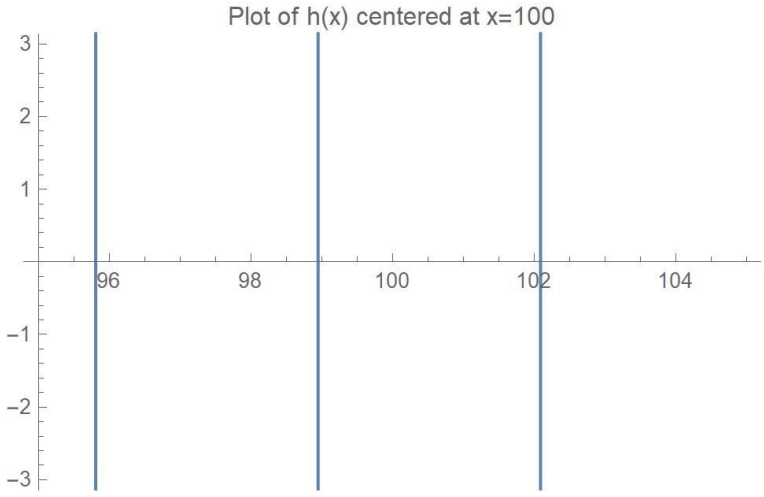
(d) Based on the overall plot for $h(x) = x - tan(x)$, I guessed that the smallest positive root of the function would be at or near x = 0. This plot is on the next page. Because of this, I decided to make four initial guesses that get closer and closer to 0 with N = 100 iterations. These are x = 0.4, 0.35, 0.3, and 0.25. At x = 0.4 and x = 0.35, I received the values 1.2774079439469027e-08 and 1.6407939216581627e-08 respectively. At x = 0.3 and x = 0.25, however, I received a "divide by zero" error on my terminal. This led me to believe that *h(x)* either crosses the x-axis between 0.35 and 0.3, giving us the smallest zero, or does not exist at some point in that interval.



Below is the same plot for *h(x)* but centered at x = 0 to show where the function could be crossing the x-axis.



To find the zero closest to x = 100, I set my initial guess at x = 100 and kept N at 100 iterations. I've also provided a plot of *h(x)* centered at x = 100 to show what the function looks like at the x-axis at this spot, however the lines pictured could just be asymptotes and not zeros.



At x = 100, Newton's method gave me the value 1.9424217388321484e+58. Because this number is extremely large, I imagine there is something wrong with my methodology.

4. To solve the system given in this problem,

$$x^2 + y^2 - 2x - 2y + 1 = 0$$
$$x + y - 2xy = 0$$

using my code for Newton's Method for systems of equations, I ran into some problems with the output I received on my terminal. I received a list of matrices as the output instead of one 2x1 matrix. I used the initial estimate $X_o = (-1, 1)$, and I've included my handwritten work for X_1 that I used to get started with the coding process.