In this assignment I was able to find all the zeros for a function using both newtons and bisection. The zero function has two required arguments, a function, and the derivative of that function. If you would like you can set the window using the lo= and hi= optional parameters, by default they are set using Cauchy’s bound. The user can also get the bounds if they call the function that they want the bound for with the optional parameter b=, where b needs to be an empty list k. The bound will be returned with the lo window value in the zero position of k and the high window parameter in the first position. There is also an e= optional parameter for the zero function that is set to Rational (1, 1e11) by default. Zero returns a list of Rational numbers that are the zeros of the function.

The way the function works is that it traverses the window and finds any zeros using the newton method, but because the newton method can sometimes not return the correct error in a reasonable amount of time or can cause overflow errors for any given error the program will use the bisection method to find an exact root around the area that the newton method returns. Once a root is found then that root becomes the next starting point for the loop to continue from. If the zero is a zero that has already been found, then the program skips it and does not append it to the list of zeros. Additionally, if the program keeps finding the same root, then it jumps farther into the window to get away from that root, this is a performance increase.

At first, I was just using bisection by finding where in the function there was a change of sign and then using bisection between two points where there was a change. This method led to a major mistake, anywhere that the function just touched the x-axis would not show up using this method because there was no change of sign. This method can find the zero because the newton method is able to find zeros in this case. Additionally, the method I implemented was faster at traversing the function because it does not have to check every step and can make jumps as it does on line 90.

One downside is that this implementation can be a little slow and tedious, especially if I decrease the error for the newton function. As it is right now the newton function does not have an incredibly low error threshold for speed, but I make up for this by using a better error for when I call bisection.

Also, this function uses tqdm, I did not have to install this I think it comes default in python, but if you do not have tqdm installed you can either do a pip install or comment out lines 53, 91 & 92, and 98 & 99.