Computing Assignment 4

Daniel Todd
301428609
D100

What I did:

To get accurate M for my bisection method, I calculated the first 3 roots manually. I then used the difference between root 3 and root 2 to calculate my interval [a, b] for the 4th root, where $a_4 = a_3 + \text{diff}$, and $b_4 = b_3 + \text{diff}$, and diff is the difference, in terms of x, between p_3 and p_2. I then automated this process, to generate the generalized statement: $a_n = a_{n-1} + \text{diff}_n$, $b_n = b_{n-1} + \text{diff}_n$, where $\text{diff}_n = |p_{n-1} - p_{n-2}|$. This worked for relatively small M, but as I increased the size of M to over 200, this became an issue as the interval would result in same signs for $J_0(a_n)$ and $J_0(b_n)$. To resolve this issue, I added a small section of code that increments b_n by one until the signs for $J_0(a_n)$ and $J_0(b_n)$ are opposites. I then added some code to check that the error produced by the bisection method did not go over $1\text{x}10^{-5}$. I also set my tolerance(TOL) for the bisection method to $1\text{x}10^{-12}$ as this is the lowest tolerance that allows the script to run in reasonable time on my computer. Once this was all in place, I increased the value of M gradually to settle at M = 5000, as this is the maximum M that allows the script to produce results in reasonable time. The error produced by the bisection method also stays below $1\text{x}10^{-5}$ for this value of M. The segment of code being referred to in this paragraph is below.

```
%Iterate over the roots M times, using the average distance between the first 3
%roots to increment a
diff = roots(3)-roots(2);
a = a3;
b = b3;
for i=4:M
    a = a+diff;
    b = b+diff;
    while sameSign(f(a), f(b))
        b = b+1;
    end
    roots(i) = bisection(getP(a, b), a, b, 1e-12);
    diff = abs(roots(i) - roots(i-1));
    if((f(roots(i)))> 1e-5)
        disp("Broke at " + i + " with x = " + roots(i)+" and f(x) = "+f(roots(i)));
    end
end
```

I wanted the program to execute efficiently (less than 5 seconds), while also aiming for more accuracy by picking a large enough M and small enough TOL to get accurate results. In this scenario, the more accurate I can make M and TOL, the more robust my script is, as these have a bounding effect on the error for my bisection method.

For the determination of my a and b values, I wanted to be as efficient as possible, which I did by automatically adding the difference between the previous two roots. However, this method of determining the a and b values did not prove to be robust, and as such I added the incrementation algorithm for b previously detailed. This method for finding a and b is also accurate, as I noticed the difference between roots stays relatively the same, only slightly decreasing overtime, and as such I calculated diff dynamically.

The Results:

I then treated the roots produced by the script as output values for some input m where, $m \in \{1, 2, \dots, M\}$, and found the linear least squares interpolating polynomial for all M points. I then used the concept of extrapolation to find α and β from this interpolating polynomial, where α is the slope and β/α is the y-intercept, asking MATLAB to print the calculated values for α and β, I get the results α= 3.14159258220125, and β= -0.249919185192266. With these results I could immediately recognize that the exact value for α is likely to be π, and the exact value for β is likely to be -1/4.