Assignment 2: Write-up, discussion and results

James Contini
10.2.22

Test Harness Discussion-

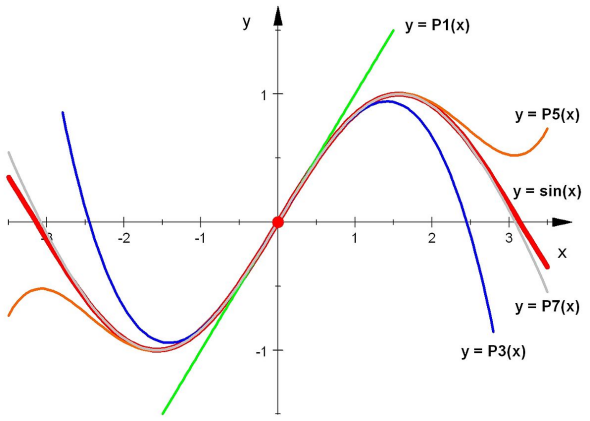
My original idea for the test harness was to see if -a was an option **first** and then run all tests but I was unable to use *getopt* twice in one function which is why I used the switch case to change global values in an array instead of running the tests. That was my final work around, but after I realized I couldn't use *getopt* I thought maybe I could parse my own options, using a for loop, to look for -a but since -a is a string and the comparison operator couldn't be used with it. I looked into string comparisons but I was too scared that I would get docked points for using another library. I also tried to use its ASCII number but it kept only printing the ASCII symbol for a hyphen (45). The pointers and global array was the only way I could think of at that point. I also received inspiration from my *one()* function from *swap()* function two lectures ago.

```
void one (int *rtc) {
        int something = 1; // This was in
        int *uno;       // pointer type v
        uno = &something;
        *rtc = *uno; // the value at memo
inters point to the value at the adress
}
```

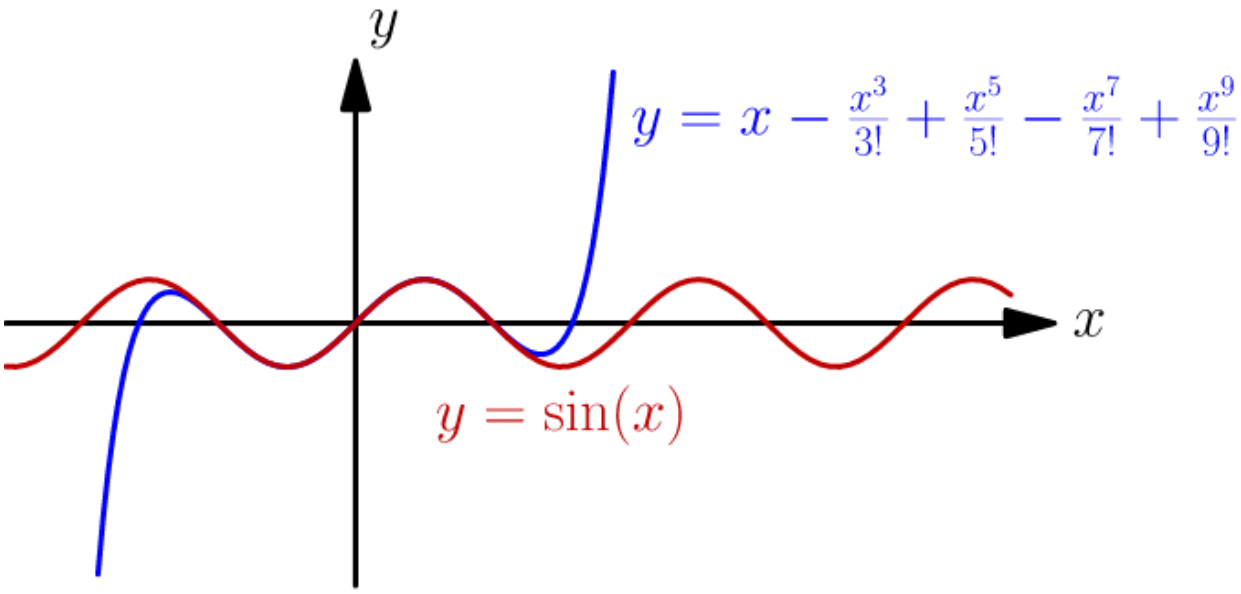
Library Implementation vs. math.h

Overview: All of my results were up to nine decimal places accurate except for the ones I am going to talk about.

***my_sin() and my_cos()*:** I found my_sin to be a really easy function to implement. I just had to copy the Taylor Series into code. I also used dynamic programming like the instructions said in order to save processing time by avoiding recalculating steps every time.



note: **this is not my image** but I think it very clearly illustrates how depending on how many iterations of the polynomial you have the closer you will get to your function. I knew the accuracy of my sin and cos approximations depended on my for loop's number of iterations because that number represents the index of the sum and the degree of the polynomial, so I chose seventeen iterations in order to make my function accurate up to nine decimal places from 0-2pi (this is why my difference is 0 for all values). Sin and cosine in particular are very good at illustrating how Taylor Polynomials approximate values because for every added x term in a polynomial another extrema can be added.

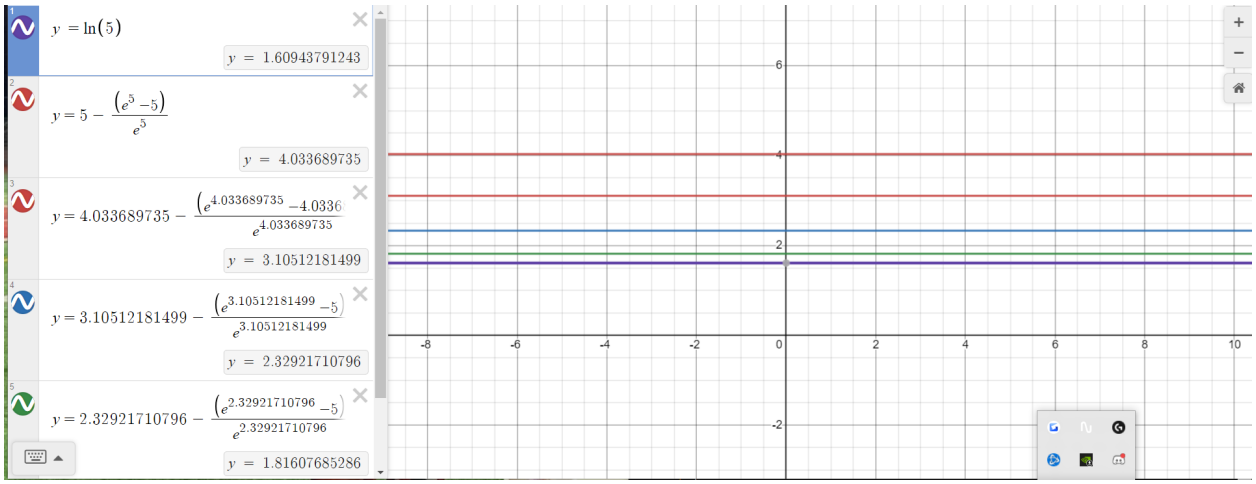


(Not my image) With a polynomial up to **x^9** we can see two local extrema which encompass just about a full period of this sin function. And since sin oscillates forever, increasing amounts of extrema in the polynomials suggest a direct connection for every added term.

Newton Raphson Method Functions-

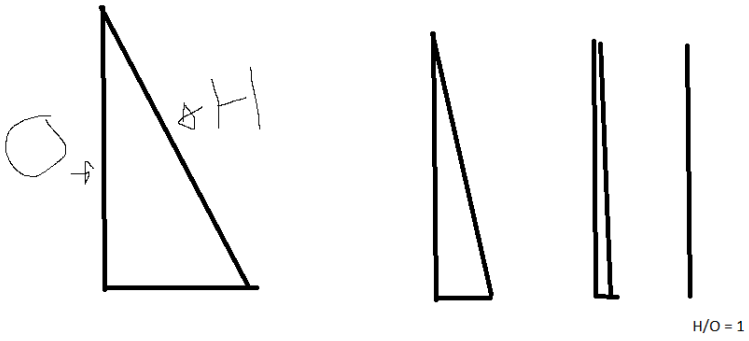
The Newton Raphson was difficult to grasp at first. But once I understood it became my go-to method to calculate arcsin, arccos, arctan, and my_sqrt. The way I was able to prove to myself that it worked was by

using desmos to graph each iteration. The natural log lends itself nicely to illustrate the approximations because it doesn't converge as fast as arcsin(.5) for example.



As you can see here for every iteration the lines appear to come closer and closer, converging to the purple line which is the true value of ln(5).

I think the reason that the arcsin messes up so badly when it approaches zero is because in terms of geometry, arcsin represents the (opposite side/hypotenuse), so if that ratio were ever to reach zero the triangle would become a line and no longer be a triangle. So as the value of arcsin converges to one the triangle is collapsing in on itself, and at one its geometrically not even a triangle because it look like:



I'm not exactly sure of the number at arcsin(1) but this is my guess as to why it acts so strangely.