Jacob McClure
jatmcclu@ucsc.edu



---

*In this document, I will evaluate and compare the results from my math.c program with respect to the functions built into the math.h library.*

**Section 1 – Functions and Their Respective Outputs:**

`double Sin(double x)`

- My function approximates more and more accurately the closer you get to the center of the boundaries for the function. That is, the farther away from the center you are, the less accurate the output is, and the greater the difference is between my Sin() function and the math.h library's sin() function. Observe the output below:

| x | Sin | Library | Difference |
|---|---|---|---|
| -6.283185 | 0.105039 | 0.000000 | -0.105039 |
| -6.086836 | 0.267153 | 0.195090 | -0.072063 |
| -5.890486 | 0.431226 | 0.382683 | -0.048543 |
| -5.694137 | 0.587640 | 0.555570 | -0.032070 |
| -5.497787 | 0.727861 | 0.707107 | -0.020754 |
| -5.301438 | 0.844609 | 0.831470 | -0.013139 |
| -5.105088 | 0.932004 | 0.923880 | -0.008125 |
| -4.908739 | 0.985685 | 0.980785 | -0.004900 |
| -4.712389 | 1.002876 | 1.000000 | -0.002876 |
| -4.516039 | 0.982425 | 0.980785 | -0.001640 |
| -4.319690 | 0.924785 | 0.923880 | -0.000906 |
| -4.123340 | 0.831953 | 0.831470 | -0.000484 |
| -3.926991 | 0.707356 | 0.707107 | -0.000249 |
| -3.730641 | 0.555693 | 0.555570 | -0.000123 |
| -3.534292 | 0.382741 | 0.382683 | -0.000058 |
| -3.337942 | 0.195116 | 0.195090 | -0.000026 |
| -3.141593 | 0.000011 | -0.000000 | -0.000011 |
| -2.945243 | -0.195086 | -0.195090 | -0.000004 |
| -2.748894 | -0.382682 | -0.382683 | -0.000002 |
| -2.552544 | -0.555570 | -0.555570 | -0.000001 |
| -2.356194 | -0.707107 | -0.707107 | -0.000000 |
| -2.159845 | -0.831470 | -0.831470 | -0.000000 |
| -1.963495 | -0.923880 | -0.923880 | -0.000000 |
| -1.767146 | -0.980785 | -0.980785 | -0.000000 |

| | | | |
|---|---|---|---|
| -1.570796 | -1.000000 | -1.000000 | -0.000000 |
| -1.374447 | -0.980785 | -0.980785 | -0.000000 |
| -1.178097 | -0.923880 | -0.923880 | -0.000000 |
| -0.981748 | -0.831470 | -0.831470 | -0.000000 |
| -0.785398 | -0.707107 | -0.707107 | -0.000000 |
| -0.589049 | -0.555570 | -0.555570 | -0.000000 |
| -0.392699 | -0.382683 | -0.382683 | 0.000000 |
| -0.196350 | -0.195090 | -0.195090 | -0.000000 |
| 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 0.196350 | 0.195090 | 0.195090 | -0.000000 |
| 0.392699 | 0.382683 | 0.382683 | 0.000000 |
| 0.589049 | 0.555570 | 0.555570 | 0.000000 |
| 0.785398 | 0.707107 | 0.707107 | 0.000000 |
| 0.981748 | 0.831470 | 0.831470 | 0.000000 |
| 1.178097 | 0.923880 | 0.923880 | 0.000000 |
| 1.374447 | 0.980785 | 0.980785 | 0.000000 |
| 1.570796 | 1.000000 | 1.000000 | 0.000000 |
| 1.767146 | 0.980785 | 0.980785 | 0.000000 |
| 1.963495 | 0.923880 | 0.923880 | 0.000000 |
| 2.159845 | 0.831470 | 0.831470 | 0.000000 |
| 2.356194 | 0.707107 | 0.707107 | 0.000000 |
| 2.552544 | 0.555570 | 0.555570 | 0.000001 |
| 2.748894 | 0.382682 | 0.382683 | 0.000002 |
| 2.945243 | 0.195086 | 0.195090 | 0.000004 |
| 3.141593 | -0.000011 | -0.000000 | 0.000011 |
| 3.337942 | -0.195116 | -0.195090 | 0.000026 |
| 3.534292 | -0.382741 | -0.382683 | 0.000058 |
| 3.730641 | -0.555693 | -0.555570 | 0.000123 |
| 3.926991 | -0.707356 | -0.707107 | 0.000249 |
| 4.123340 | -0.831953 | -0.831470 | 0.000484 |
| 4.319690 | -0.924785 | -0.923880 | 0.000906 |
| 4.516039 | -0.982425 | -0.980785 | 0.001640 |
| 4.712389 | -1.002876 | -1.000000 | 0.002876 |
| 4.908739 | -0.985685 | -0.980785 | 0.004900 |
| 5.105088 | -0.932004 | -0.923880 | 0.008125 |
| 5.301438 | -0.844609 | -0.831470 | 0.013139 |
| 5.497787 | -0.727861 | -0.707107 | 0.020754 |
| 5.694137 | -0.587640 | -0.555570 | 0.032070 |
| 5.890486 | -0.431226 | -0.382683 | 0.048543 |

6.086836 -0.267153 -0.195090 0.072063

- As you can see, the closer you get to the edges of the boundaries, the less accurate my function is compared to the actual math.h library sine function. This is largely because I used Padé Approximants (as per the instructions in the lab manual) as opposed to the Taylor Series Approximations. I will discuss more on that later in this document.

`double Cos(double x)`

- Like my Sine function, my Cosine function approximates more and more accurately the closer you get to the center of the boundaries for the function. The farther away from the center you are, the less accurate the output is, and the greater the difference is between my Cos() function and the math.h library's cos() function. Observe the output below:

| x | Cos | Library | Difference |
|---|---|---|---|
| -6.283185 | 0.727641 | 1.000000 | 0.272359 |
| -6.086836 | 0.784162 | 0.980785 | 0.196623 |
| -5.890486 | 0.784422 | 0.923880 | 0.139457 |
| -5.694137 | 0.734397 | 0.831470 | 0.097073 |
| -5.497787 | 0.640870 | 0.707107 | 0.066237 |
| -5.301438 | 0.511323 | 0.555570 | 0.044247 |
| -5.105088 | 0.353787 | 0.382683 | 0.028896 |
| -4.908739 | 0.176670 | 0.195090 | 0.018420 |
| -4.712389 | -0.011441 | -0.000000 | 0.011441 |
| -4.516039 | -0.202001 | -0.195090 | 0.006911 |
| -4.319690 | -0.386734 | -0.382683 | 0.004050 |
| -4.123340 | -0.557868 | -0.555570 | 0.002298 |
| -3.926991 | -0.708365 | -0.707107 | 0.001258 |
| -3.730641 | -0.832132 | -0.831470 | 0.000662 |
| -3.534292 | -0.924214 | -0.923880 | 0.000334 |
| -3.337942 | -0.980946 | -0.980785 | 0.000161 |
| -3.141593 | -1.000074 | -1.000000 | 0.000074 |
| -2.945243 | -0.980817 | -0.980785 | 0.000032 |
| -2.748894 | -0.923892 | -0.923880 | 0.000013 |
| -2.552544 | -0.831474 | -0.831470 | 0.000005 |
| -2.356194 | -0.707108 | -0.707107 | 0.000002 |
| -2.159845 | -0.555571 | -0.555570 | 0.000001 |
| -1.963495 | -0.382684 | -0.382683 | 0.000000 |
| -1.767146 | -0.195090 | -0.195090 | 0.000000 |

| | | | |
|---|---|---|---|
| -1.570796 | -0.000000 | 0.000000 | 0.000000 |
| -1.374447 | 0.195090 | 0.195090 | 0.000000 |
| -1.178097 | 0.382683 | 0.382683 | 0.000000 |
| -0.981748 | 0.555570 | 0.555570 | 0.000000 |
| -0.785398 | 0.707107 | 0.707107 | 0.000000 |
| -0.589049 | 0.831470 | 0.831470 | 0.000000 |
| -0.392699 | 0.923880 | 0.923880 | 0.000000 |
| -0.196350 | 0.980785 | 0.980785 | 0.000000 |
| 0.000000 | 1.000000 | 1.000000 | 0.000000 |
| 0.196350 | 0.980785 | 0.980785 | -0.000000 |
| 0.392699 | 0.923880 | 0.923880 | 0.000000 |
| 0.589049 | 0.831470 | 0.831470 | 0.000000 |
| 0.785398 | 0.707107 | 0.707107 | 0.000000 |
| 0.981748 | 0.555570 | 0.555570 | 0.000000 |
| 1.178097 | 0.382683 | 0.382683 | 0.000000 |
| 1.374447 | 0.195090 | 0.195090 | 0.000000 |
| 1.570796 | -0.000000 | -0.000000 | 0.000000 |
| 1.767146 | -0.195090 | -0.195090 | 0.000000 |
| 1.963495 | -0.382684 | -0.382683 | 0.000000 |
| 2.159845 | -0.555571 | -0.555570 | 0.000001 |
| 2.356194 | -0.707108 | -0.707107 | 0.000002 |
| 2.552544 | -0.831474 | -0.831470 | 0.000005 |
| 2.748894 | -0.923892 | -0.923880 | 0.000013 |
| 2.945243 | -0.980817 | -0.980785 | 0.000032 |
| 3.141593 | -1.000074 | -1.000000 | 0.000074 |
| 3.337942 | -0.980946 | -0.980785 | 0.000161 |
| 3.534292 | -0.924214 | -0.923880 | 0.000334 |
| 3.730641 | -0.832132 | -0.831470 | 0.000662 |
| 3.926991 | -0.708365 | -0.707107 | 0.001258 |
| 4.123340 | -0.557868 | -0.555570 | 0.002298 |
| 4.319690 | -0.386734 | -0.382683 | 0.004050 |
| 4.516039 | -0.202001 | -0.195090 | 0.006911 |
| 4.712389 | -0.011441 | 0.000000 | 0.011441 |
| 4.908739 | 0.176670 | 0.195090 | 0.018420 |
| 5.105088 | 0.353787 | 0.382683 | 0.028896 |
| 5.301438 | 0.511323 | 0.555570 | 0.044247 |
| 5.497787 | 0.640870 | 0.707107 | 0.066237 |
| 5.694137 | 0.734397 | 0.831470 | 0.097073 |
| 5.890486 | 0.784422 | 0.923880 | 0.139457 |

6.086836 0.784162 0.980785 0.196623

- Again, the closer we are to the edges of the boundaries, the greater the difference between my Sin() and math.h's sin() function outputs. This is because of the limited nature of Padé Approximants. I will go into further detail on that later.

`double Tan(double x)`

- My tangent function is extremely accurate. With the number of decimal places we are observing, it is showing a negligible difference when compared to the math.h library's tan() function. Observe the output below:

| x | Tan | Library | Difference |
|---|---|---|---|
| -1.569796 | -999.999666 | -999.999667 | -0.000000 |
| -1.373447 | -5.001197 | -5.001197 | -0.000000 |
| -1.177097 | -2.407402 | -2.407402 | -0.000000 |
| -0.980748 | -1.493371 | -1.493371 | 0.000000 |
| -0.784398 | -0.998002 | -0.998002 | -0.000000 |
| -0.588049 | -0.666733 | -0.666733 | 0.000000 |
| -0.391699 | -0.413042 | -0.413042 | 0.000000 |
| -0.195350 | -0.197873 | -0.197873 | 0.000000 |
| 0.001000 | 0.001000 | 0.001000 | 0.000000 |
| 0.197350 | 0.199952 | 0.199952 | 0.000000 |
| 0.393699 | 0.415386 | 0.415386 | -0.000000 |
| 0.590049 | 0.669626 | 0.669626 | 0.000000 |
| 0.786398 | 1.002002 | 1.002002 | 0.000000 |
| 0.982748 | 1.499850 | 1.499850 | 0.000000 |
| 1.179097 | 2.421059 | 2.421059 | 0.000000 |
| 1.375447 | 5.053746 | 5.053746 | 0.000000 |

- As you can see, the outputs are nearly identical. This is because I spent a lot of time working through the calculation for the approximant equation for this particular function. Another reason that it is so much closer to the math.h library's output (compared to the differences observed in the latter two functions) is because the bounds of the Tan() function are further away from the edges; that is, the boundaries of this function are far away from where the approximant diverges from the actual tangent function.

`double Exp(double x)`

- My exponential function is also extremely accurate. With the number of decimal places we are observing, it is showing a negligible difference when compared to the math.h library's exp() function. Observe the output below:

| x | Exp | Library | Difference |
|---|---|---|---|
| 0.000000 | 1.000000 | 1.000000 | 0.000000 |
| 0.100000 | 1.105171 | 1.105171 | 0.000000 |
| 0.200000 | 1.221403 | 1.221403 | 0.000000 |
| 0.300000 | 1.349859 | 1.349859 | 0.000000 |
| 0.400000 | 1.491825 | 1.491825 | 0.000000 |
| 0.500000 | 1.648721 | 1.648721 | 0.000000 |
| 0.600000 | 1.822119 | 1.822119 | 0.000000 |
| 0.700000 | 2.013753 | 2.013753 | 0.000000 |
| 0.800000 | 2.225541 | 2.225541 | 0.000000 |
| 0.900000 | 2.459603 | 2.459603 | 0.000000 |
| 1.000000 | 2.718282 | 2.718282 | 0.000000 |
| 1.100000 | 3.004166 | 3.004166 | 0.000000 |
| 1.200000 | 3.320117 | 3.320117 | 0.000000 |
| 1.300000 | 3.669297 | 3.669297 | 0.000000 |
| 1.400000 | 4.055200 | 4.055200 | 0.000000 |
| 1.500000 | 4.481689 | 4.481689 | 0.000000 |
| 1.600000 | 4.953032 | 4.953032 | 0.000000 |
| 1.700000 | 5.473947 | 5.473947 | 0.000000 |
| 1.800000 | 6.049647 | 6.049647 | 0.000000 |
| 1.900000 | 6.685894 | 6.685894 | 0.000000 |
| 2.000000 | 7.389056 | 7.389056 | 0.000000 |
| 2.100000 | 8.166170 | 8.166170 | 0.000000 |
| 2.200000 | 9.025013 | 9.025013 | 0.000000 |
| 2.300000 | 9.974182 | 9.974182 | 0.000000 |

- I did not include the entire output because it was too spacious to cram into this document, but as you can see, the Exp() function I wrote is nearly identical to the exp() function built into the math.h library. This is because this function did not rely on any form of approximation, but rather an algorithm for calculating exponentiation; there is no approximation taking place here. I did not use a Padé Approximant for this function.

**Section 2 – Why There Are Differences:**

The primary reason my Sine and Cosine functions differ in output when compared to the math.h library's built-in sin() and cos() functions is because I used Padé Approximants to calculate those functions (as per the suggestion of lab instructors). If I had used a Taylor Series instead, it's possible that I could've had a smaller margin of difference. I will explain why below.

<u>Padé Approximants versus Taylor Series Approximation:</u>
When using a Padé Approximant for functions like Cosine and Sine, you can only achieve a certain degree of accuracy. There is a reason for this limitation, and that reason is relatively simple to understand if you look at the functions graphically. Using Sin(x) as an example, we can see (below) that as the function functions approach -2π and 2π, they diverge. More specifically, the Padé Approximant veers off into oblivion. Padé Approximants are only accurate up to a certain range, and then they are no longer useful. A Taylor Series Approximation on the other hand, is as accurate as you allow it to become; the more terms you add, the better the approximation will be. I did not use the Taylor Series Approximation because it was not what we were instructed to do. But I know that if I had implemented the Taylor Series instead of a Horner Normal Form for the Padé Approximants of sine and cosine, the observable difference between my functions' outputs and the outputs from math.h's outputs would've been much lesser.
*Padé Approximant = yellow*
*sin(x) = blue*