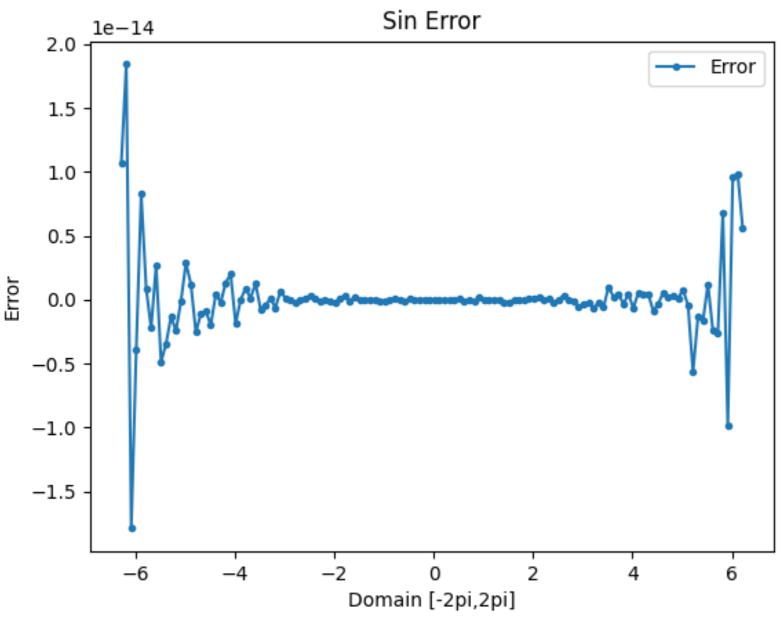
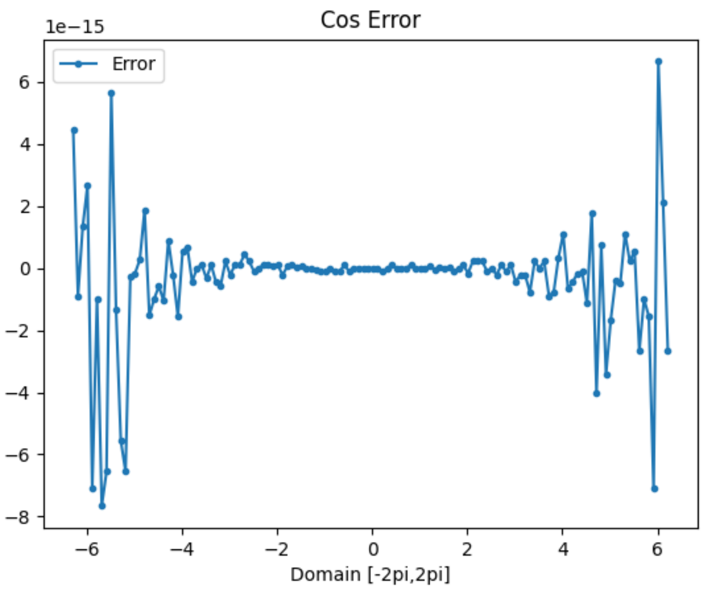
**Error Analysis:**

There can be no perfect implementation of these trigonometric functions. C uses the IEE 754 standard, and because floating points themselves are simply approximations of reals and rationals, there isn't any way to achieve pitch-perfect precision when computing floating points. However, we can increase the precision and accuracy by allocating more memory to store more bits of data. For this assignment, we store our numbers as doubles, 64 bits of memory allocated for storage, so if we add, subtract, divide etc... floats together we can maintain less error, but some error will be maintained and carried over from the original initialization when manipulating them in our Taylor series.

**Sin and Cos Error:**

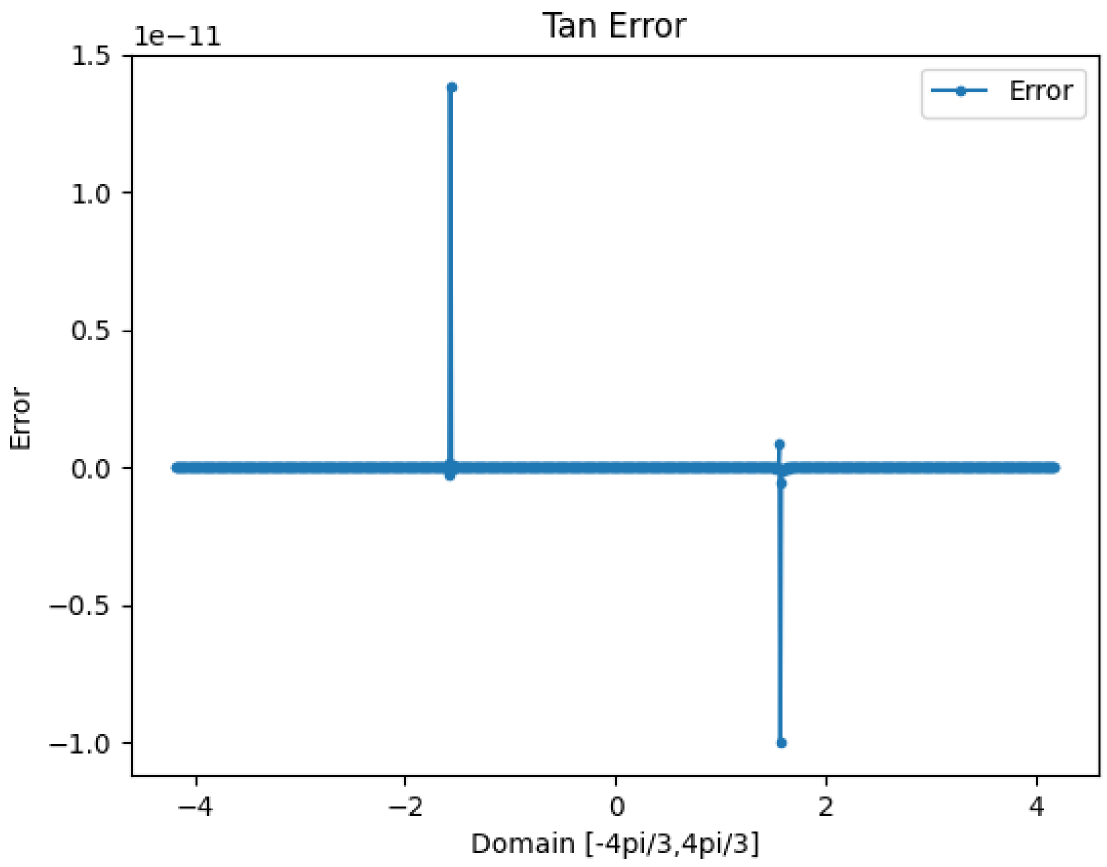
My Taylor series for Sin, like all my series, is based around 0, so the closer to 0 the inputs are the higher degree of accuracy there should be. We are essentially taking the derivative of sin(x) many times over to get better approximations of the sin curve around a certain point, and as the approximation curve becomes more accurate around sin(0), unless specified with a change in x values, the curve will be less accurate farther away from the center, not to mention the values of P(x) will sky rocket. Now for the error graph, the slope represents the average distance or value that my custom outputs differ from the expected value. Our function stops taking in values when around 14 decimal points of precision have been achieved and truncates there. The sin and cos library functions become extremely complex the farther the x value is to 0, and since our taylor series really only works for values close to 0, the error will become even greater the farther our x values are compared. We use a domain that's fairly close to 0, so its reasonable that the difference is less in outputs closer to 0.

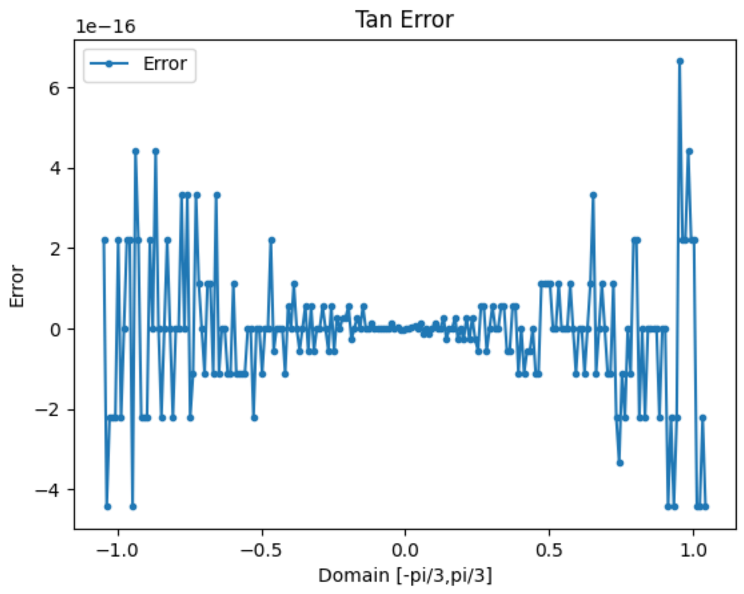




**Tan Error:**

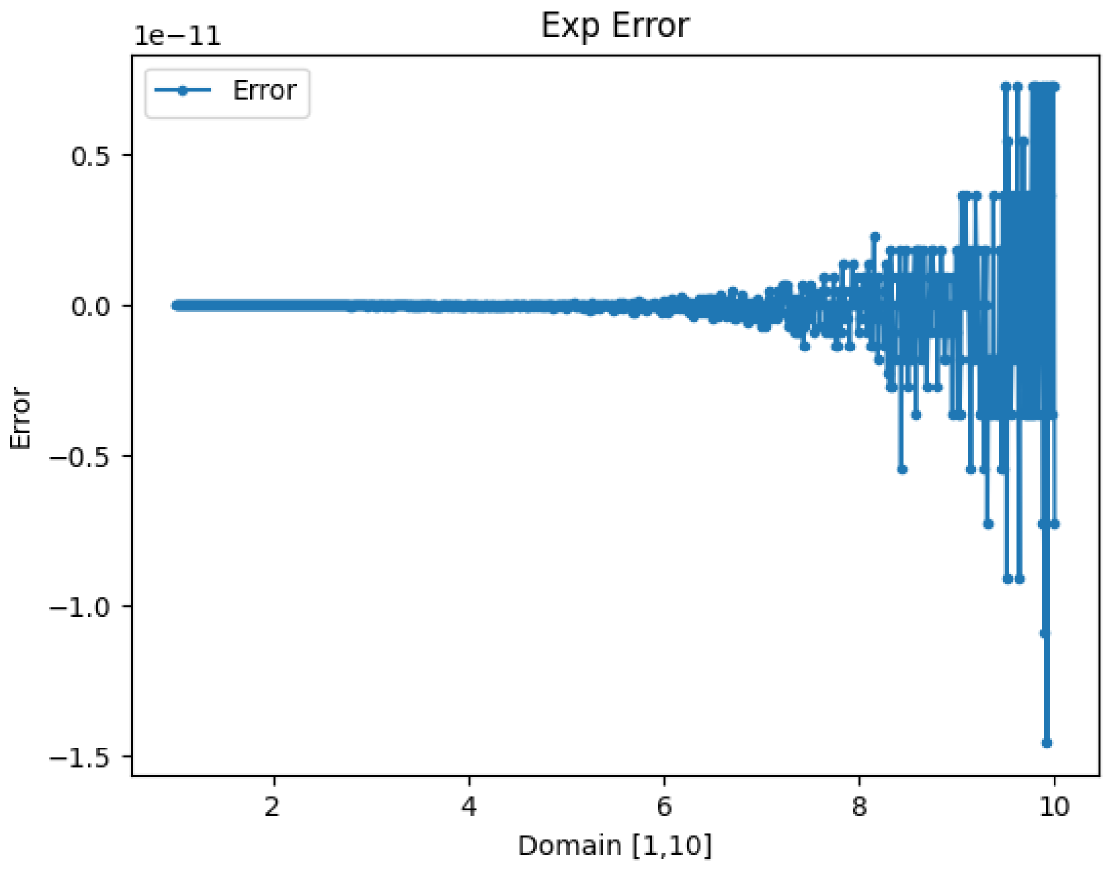
Now as for the Tangent function, the only notable error is when we take a variant of pi/2, where sin(pi/2) = 1 and cos(pi/2) = 0, and dividing 0 into anything results in an error. I changed the domain just for this particular graphing instance to  better visualize the errors, as we can see around where tan(pi/2) is being taken the errors spike, meaning the compiler is trying to calculate a divide by 0.





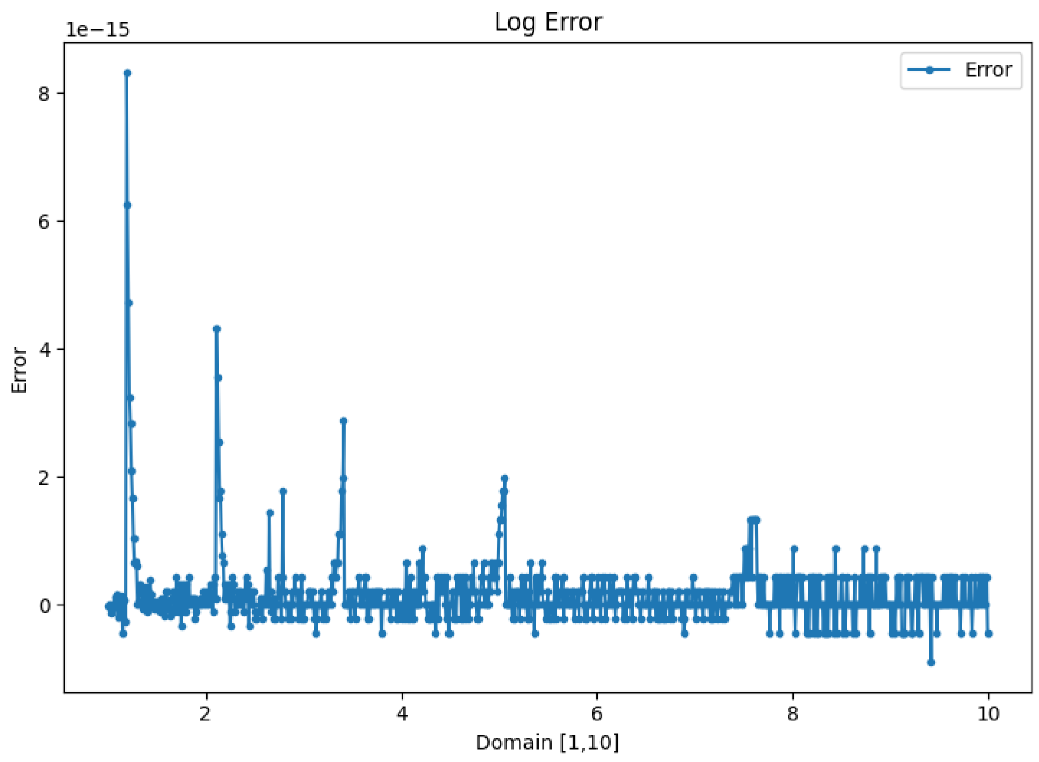
**Exp Error:**

As x values increase for e^x, the taylor series produces massive numbers for later values, like the rubric said, after around the value of 7. It becomes harder to approximate at values farther from 0 so the taylor series becomes less effective as function values skyrocket. The math library function has a way to handle this and better approximate around higher values, resulting in a greater error between functions as the taylor series becomes less accurate with higher values.



**Log Error:**

The errors for x in some cases seem to spike in that the error is different by an additional decimal place or two,  for example the error at x = 1.2 is different by 1e-15, while the values around it stay around 1e-17. This range of difference does decrease as x increases, but i still can't figure out why. Other than that the differences are not 0, however they don't produce the same range of difference like Sin, Cos, and Exp as values move away from 0. This is probably the case because the Newton Raphson method makes a tangent line to approximate the value, and becomes less effective around points of slope change, like when x is closer to 0. That is my guess for why it becomes less accurate when approximating closer to 0.


Log Error
1e−15
Error
Domain [1,10]
Error