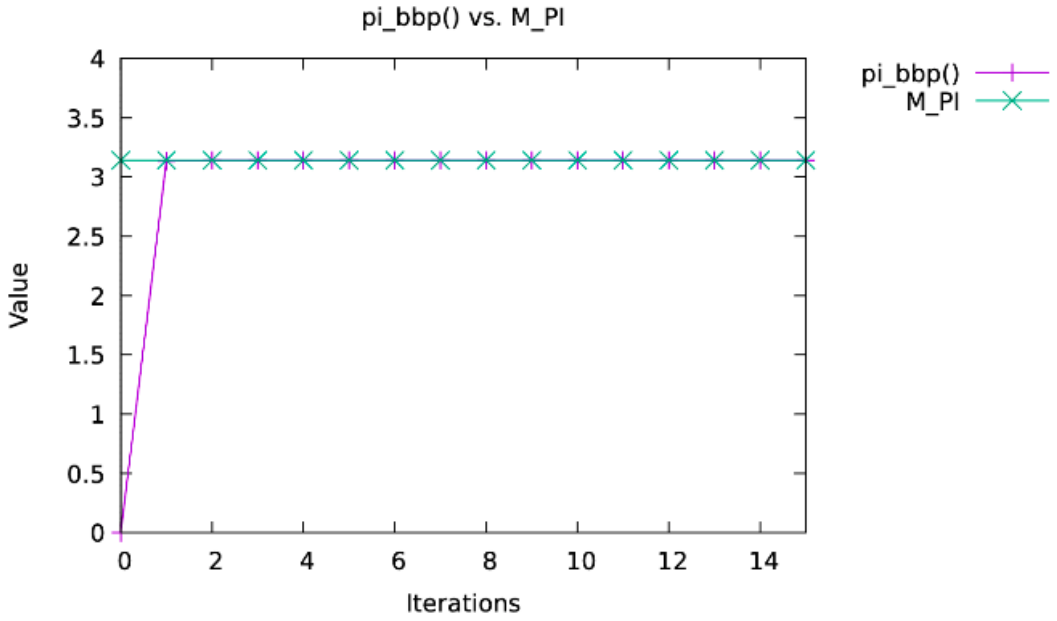
Diwa Ashwini Vittala
CSE 13S - Fall 2021

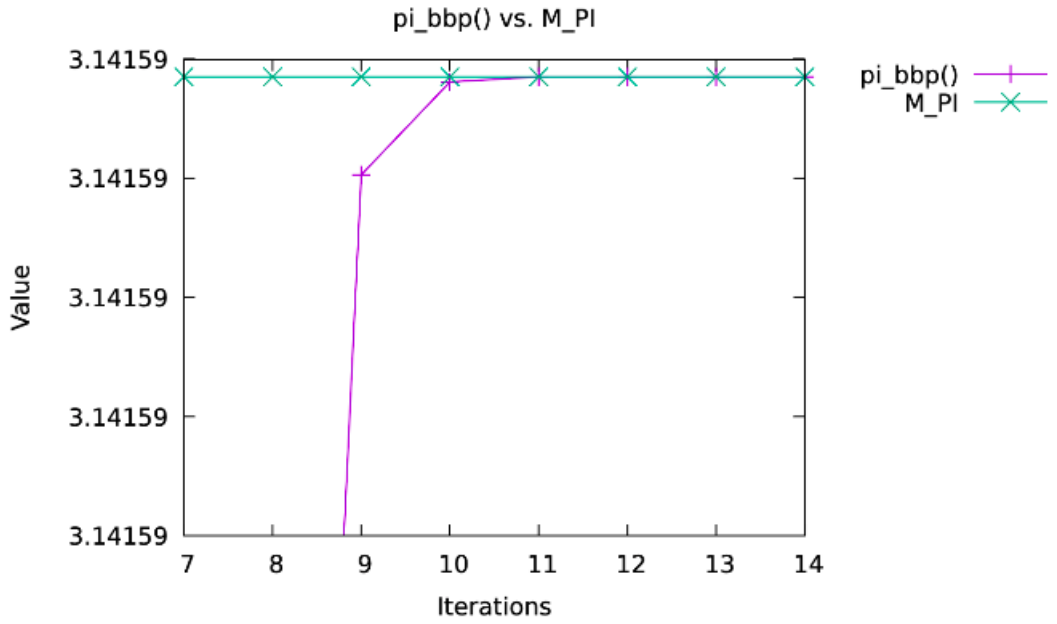
⁰❀ Assignment 2: Writeup ❀。

A LITTLE SLICE OF π

BBP



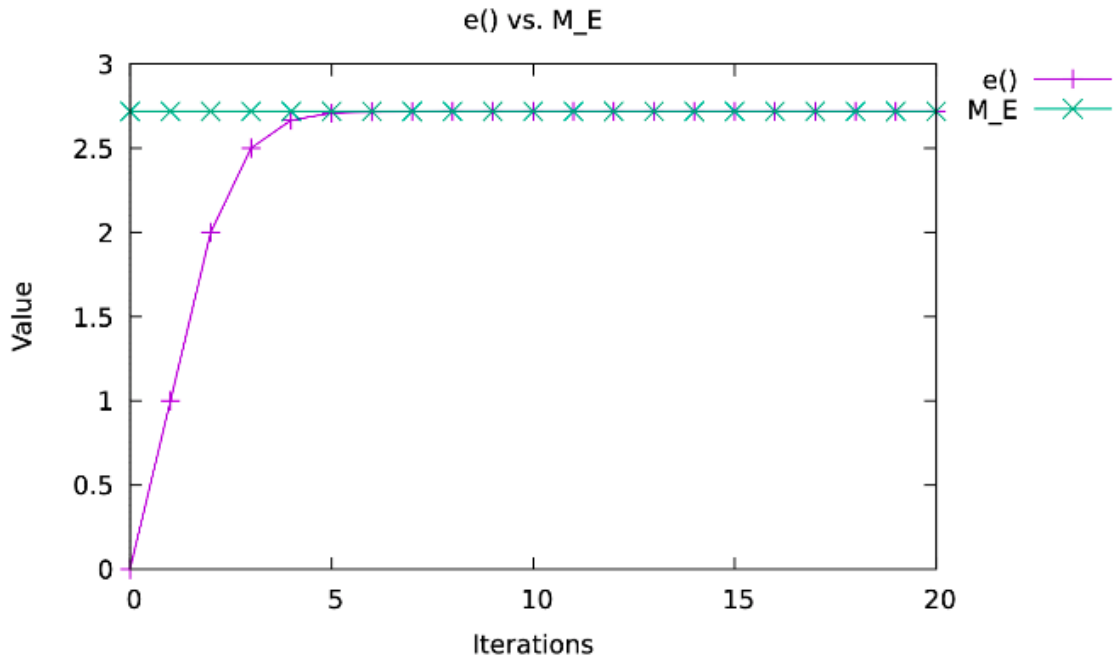
This graph plots my pi_bbp() function's approximation of pi along with the approximation of pi from the math library. We can see that the difference between their approximations drastically decreases after the first iteration through the pi_bbp() function. From our mathlib-test, we saw that the two approximations are supposed to merge at around 11 terms (making it the fastest formula to approach pi out of all the ones we are testing).



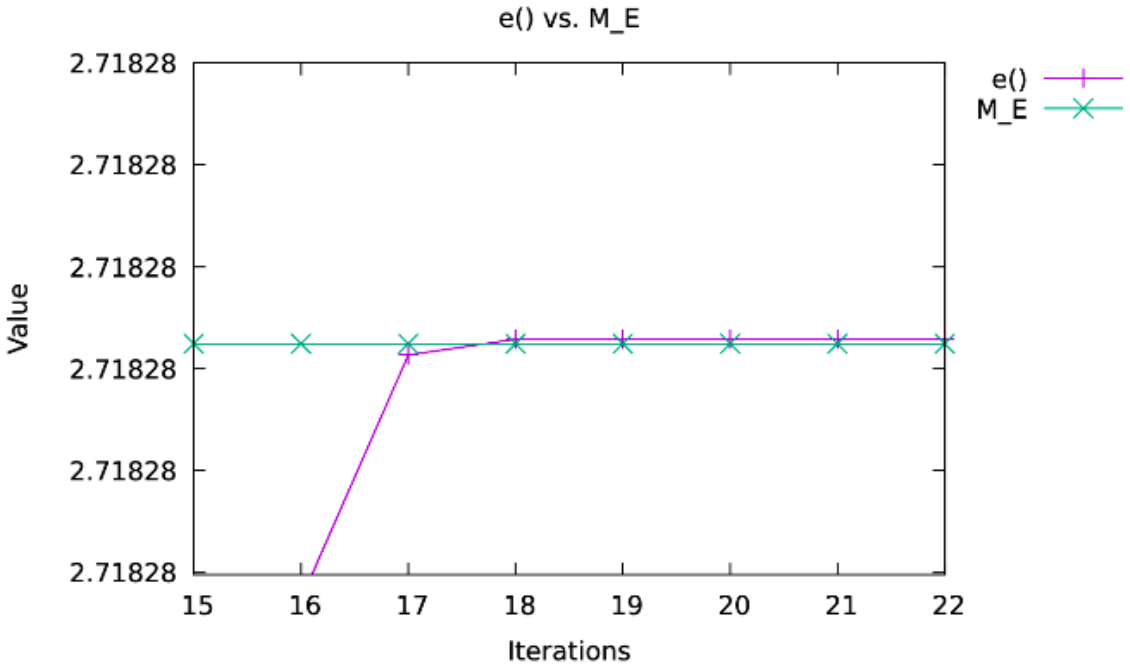
When we zoom into the pi_bbp() function around 7 to 14 iterations, we can better understand how the function is getting closer to the approximate value of pi from the math library. Around 11 terms into the BBP formula,

we see that the lines meet, which means the approximations are either the same from this term and on, or that they are so close to the same value that it is difficult for us to see the difference.

e()



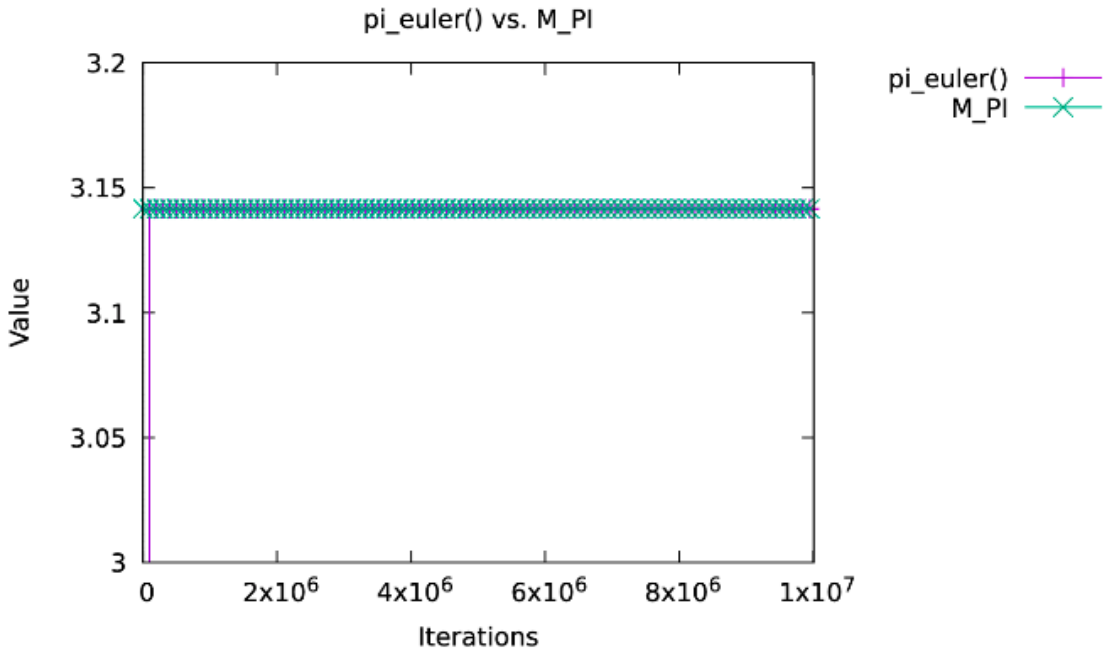
When we plot the approximations of e made by my e() function against the approximation by the math library over 20 iterations, we can see that they are close to the same value early around 6 terms. This may be due to the nature of factorial functions, where they rapidly increase as the numbers get larger.



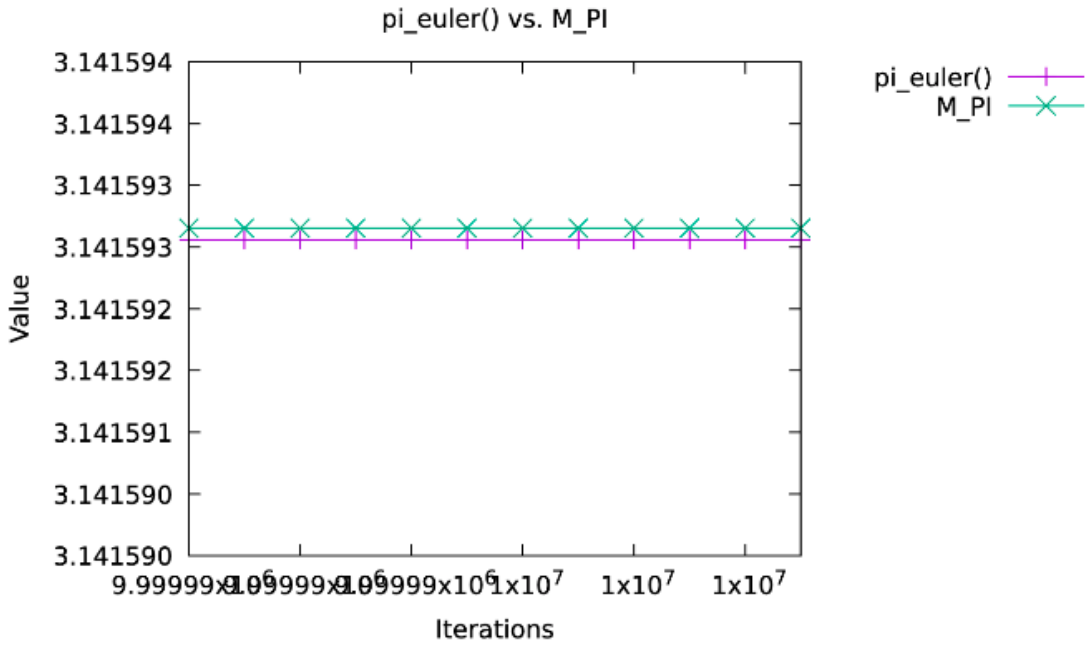
Upon closer inspection around 15 to 22 terms, we see that these approximations are just really close to each other until they meet between 17 to 18 terms (which is consistent with the mathlib-test statistics). It is also evident that the value of pi calculated by e() actually crosses the math library approximation and stays above

that value after the intersection. The reason why this may be happening is due to the requirement for the number to have a difference less than EPSILON, and the fact that we are incrementing by a whole number. After crossing the value, the difference between e() and the math library value still exists, and that may be why it is slightly above the intersection point.

Euler



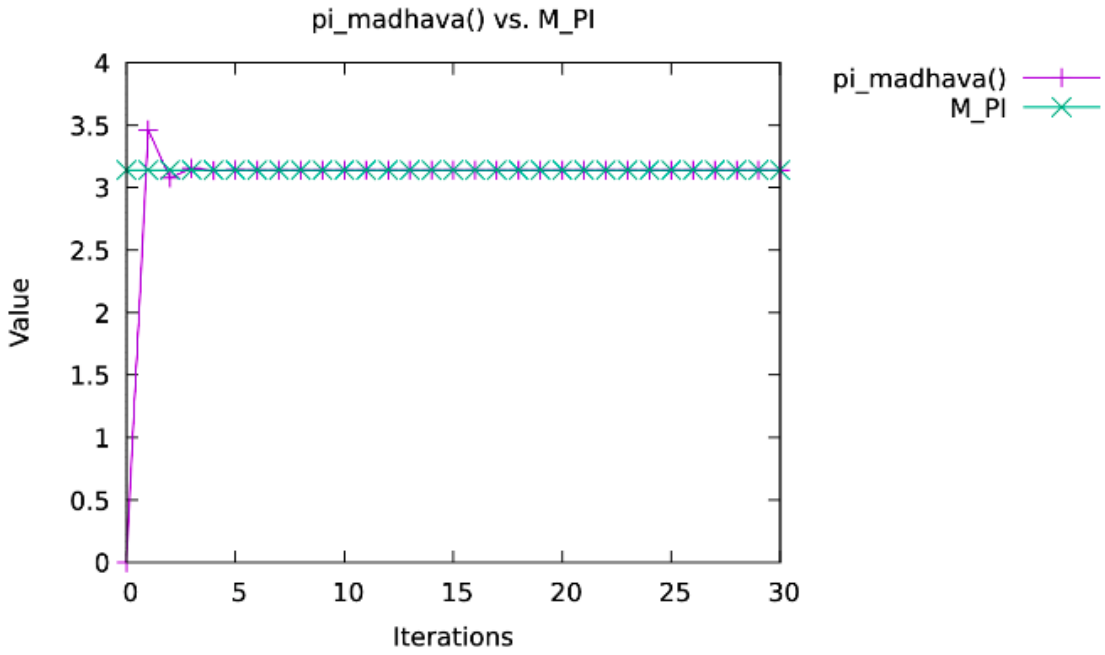
When we plot our pi_euler() function's approximation of pi in comparison to the math library's approximation of pi, we see that they are very close to each other from the beginning.



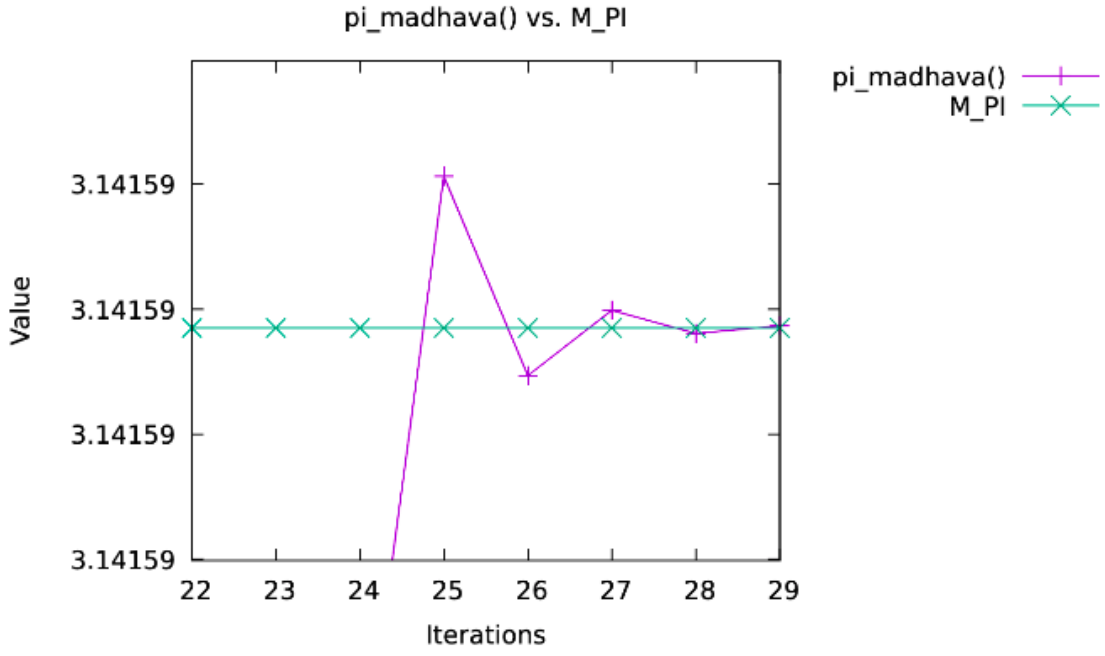
When we zoom in around 10,000,000 terms, we see that they do not really touch, even after the mathlib-test's approximation of how many terms it takes for the difference to be insignificant. The approximations are extremely close to each other but even after this many terms, the difference cannot be 0 due to the nature of the

formula used to create the function. The formula using Euler's method uses 1/k^2, and we know that 1/x for any number x cannot be 0, but can get really close to that number as x gets larger.

Madhava

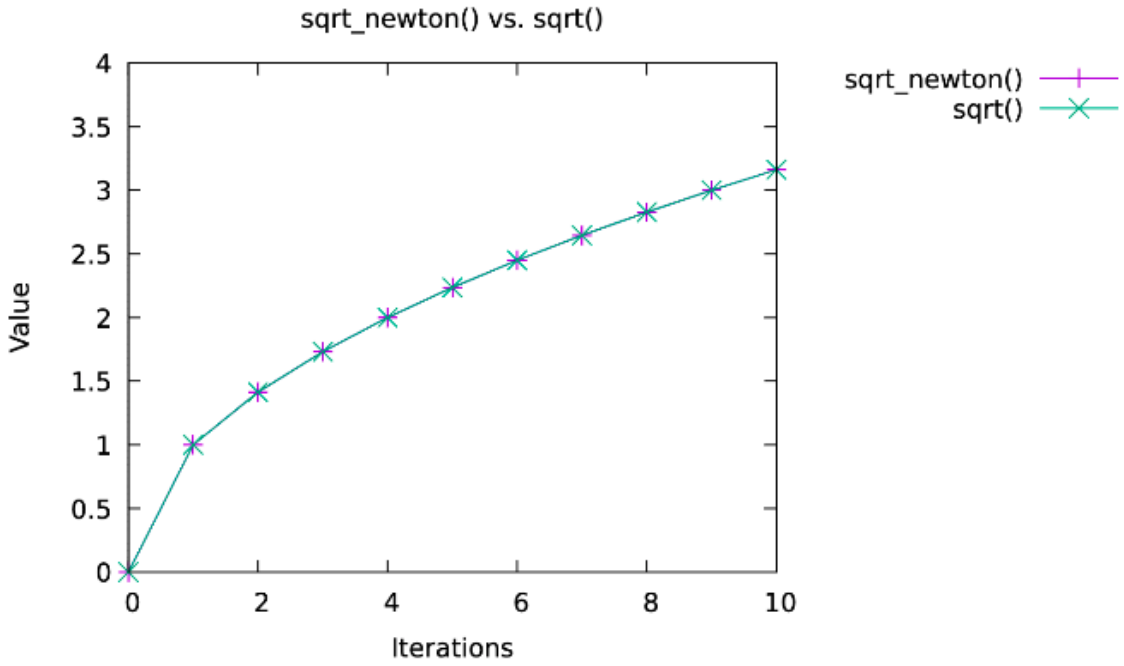


The graph for the Madhava is the most interesting out of all of them. The value keeps oscillating from above the value of pi in the math library to below it. This happens with smaller differences until it is smaller than EPSILON (which we defined as 1e-14).

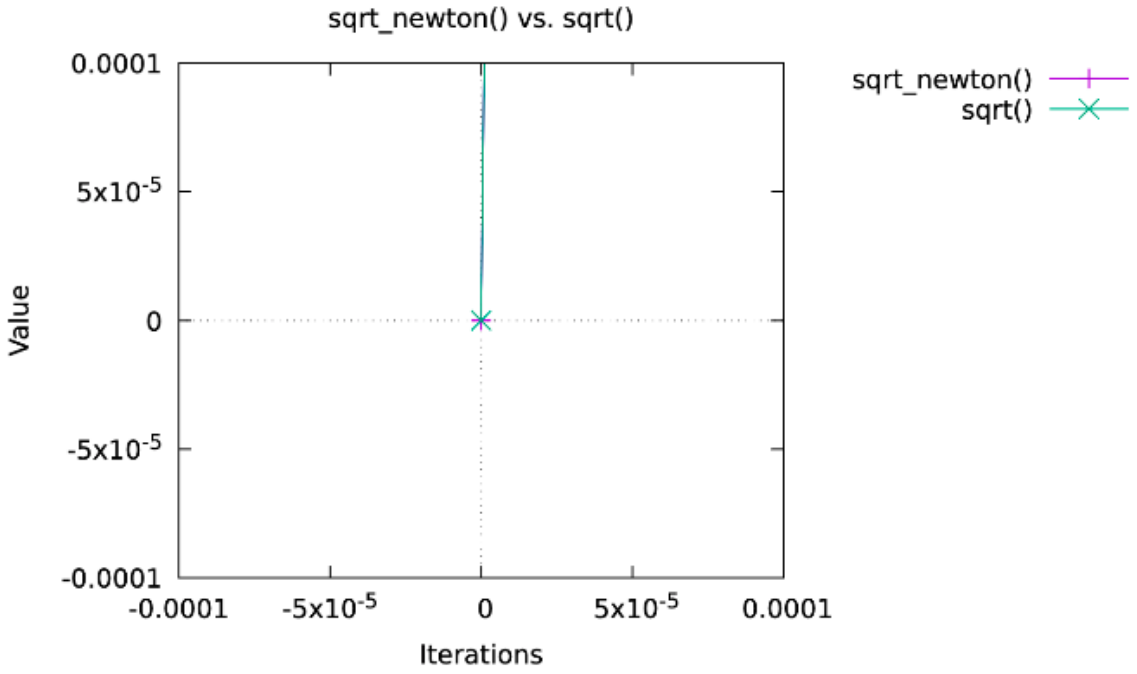


This is easier to see in the zoomed in version of the graph over the range of values 22 to 29. Our mathlib-test indicated that the difference between our function becomes smaller than epsilon after around 27 terms, but we can see from the graph that it continues to oscillate as we move to a larger number of terms. The oscillation likely occurs due to the (-3)^k portion of the Madhava Series formula.

Newton

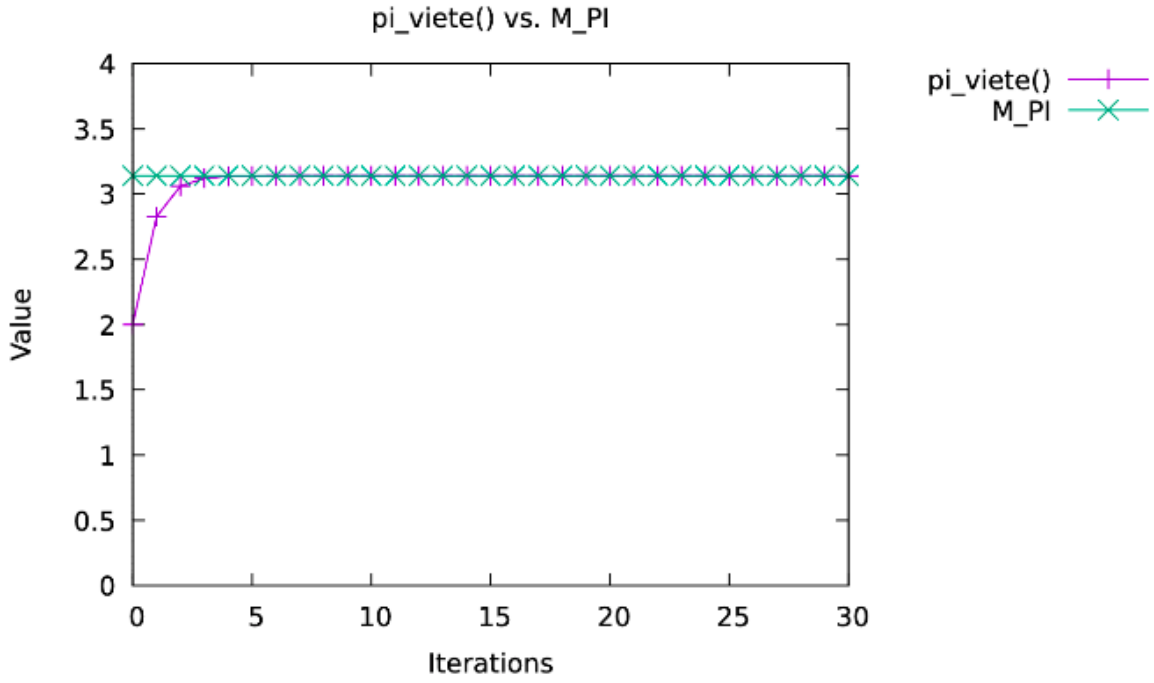


The graphs comparing our sqrt_newton() function with the math library's square root function follows a pattern where it increases less and less as we input larger numbers. Since root functions are the same as exponential functions but with a fractional exponent, the rate of incrementation decreases over the number of terms.

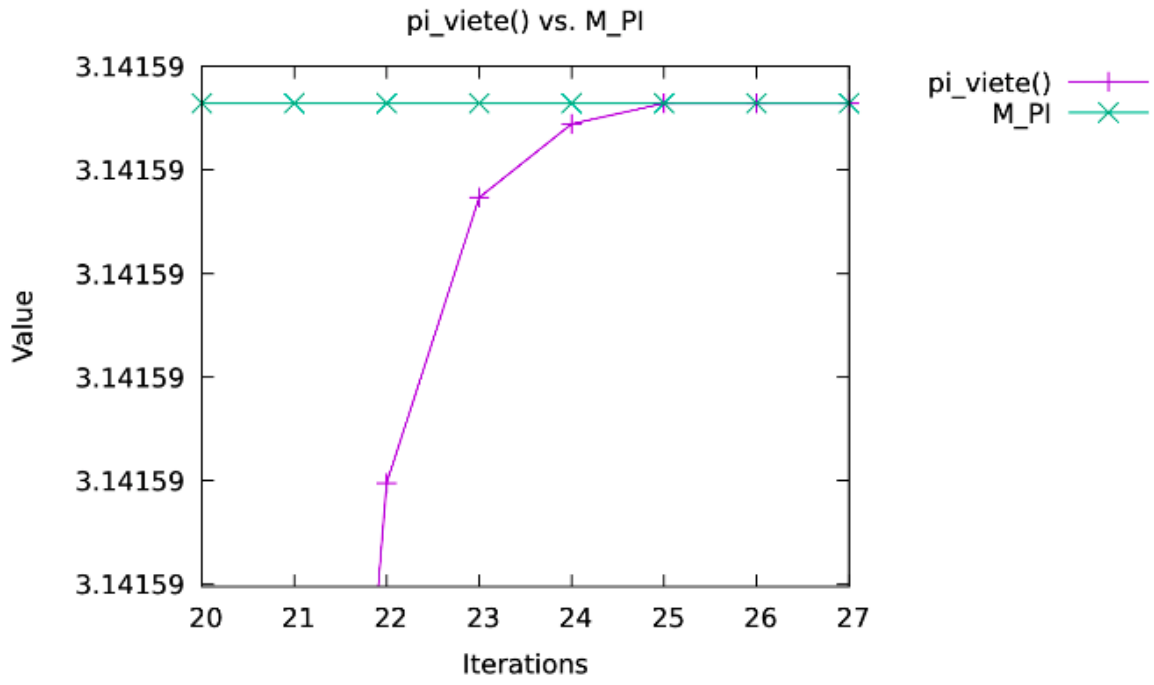


What is interesting about this graph is that as we get closer to 0, due to the way we approximate the square root by dividing into the middle of the two numbers 0 and the input, the square root of the number 0 is really close to 0 but never actually equals 0.

Viete



The function pi_viete() approximates the value of pi close to the math library value around 5 iterations when we look at the overall graph, but when we zoom in we can see that there is a larger difference. The graph looks similar to that of sqrt_newton() and e() because there are components of the viete formula that are similar, such as continuously square rooting 2+the previous value and the sum of fractions like (value)/2.



In the zoomed-in version over the range of 20 to 27, we can see the functions intersect at around 25 terms, or perhaps they are just so close that they seem like they intersect but pi_viete() approaches the same value.