Analysis - A Little Slice of π

Deeraj Gurram

Fall 2021

1 Introduction

As there exists a difference between an approximation and the exact number, we must compare our results with the correct value to determine accuracy. This writeup consists of an examination of my computed values from the different mathematical formulas with the exact values taken from the math.h library. In the next parts of this writeup, you will find the formulas that were used to create my code, an analysis on the differences between my computed values and the exact terms along with commentary on why this appears, and a conclusion that sums up the differences that were found.

2 Formulas

Calculating e

$$\frac{x^k}{k!} = \frac{x^{k-1}}{(k-1)!} \times \frac{x}{k}.$$

Calculating π using the Madhava Series

$$\sum_{k=0}^{\infty} \frac{(-3)^{-k}}{2k+1} = \sqrt{3}\tan^{-1}\frac{1}{\sqrt{3}} = \frac{\pi}{\sqrt{12}}$$

Calculating π using Euler's Solution

$$p(n) = \sqrt{6 \sum_{k=1}^{n} \frac{1}{k^2}}$$

Calculating π using the Bailey-Borwein-Plouffe Formula

$$p(n) = \sum_{k=0}^{n} 16^{-k} \times \frac{(k(120k+151)+47)}{k(k(k(512k+1024)+712)+194)+15}.$$

Calculating π using Viète's Formula

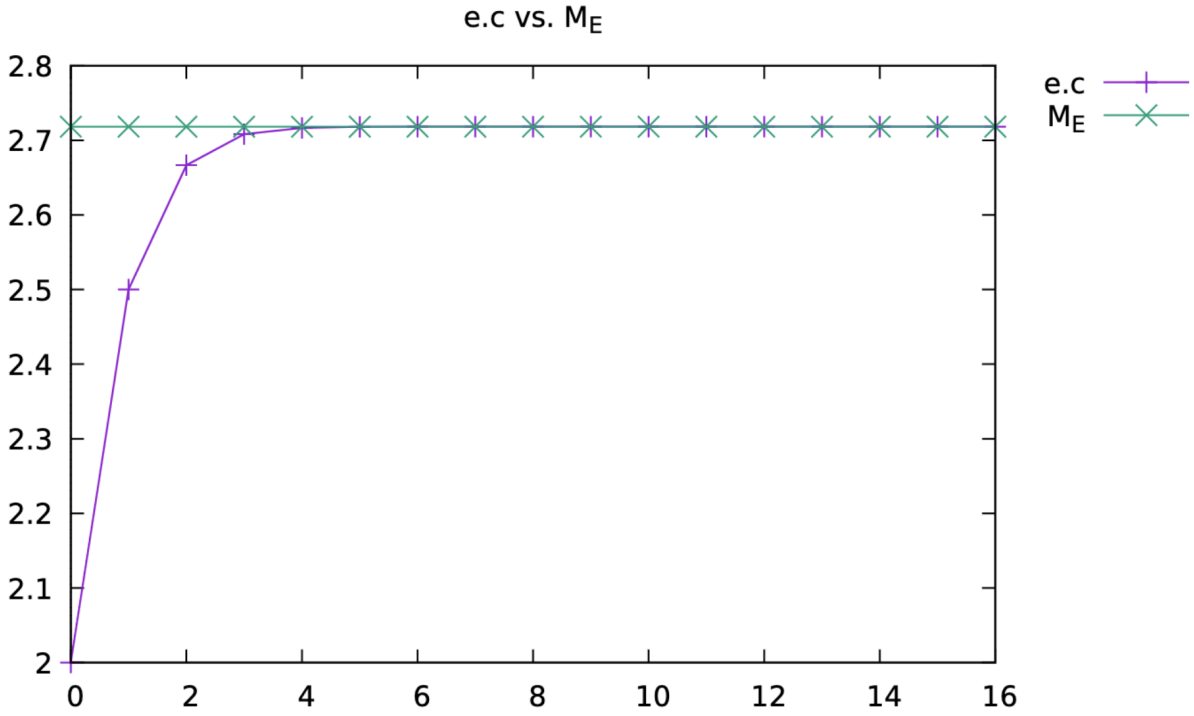
$$\frac{2}{\pi} = \prod_{k=1}^{\infty} \frac{a_i}{2}$$

Calculating the square root

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

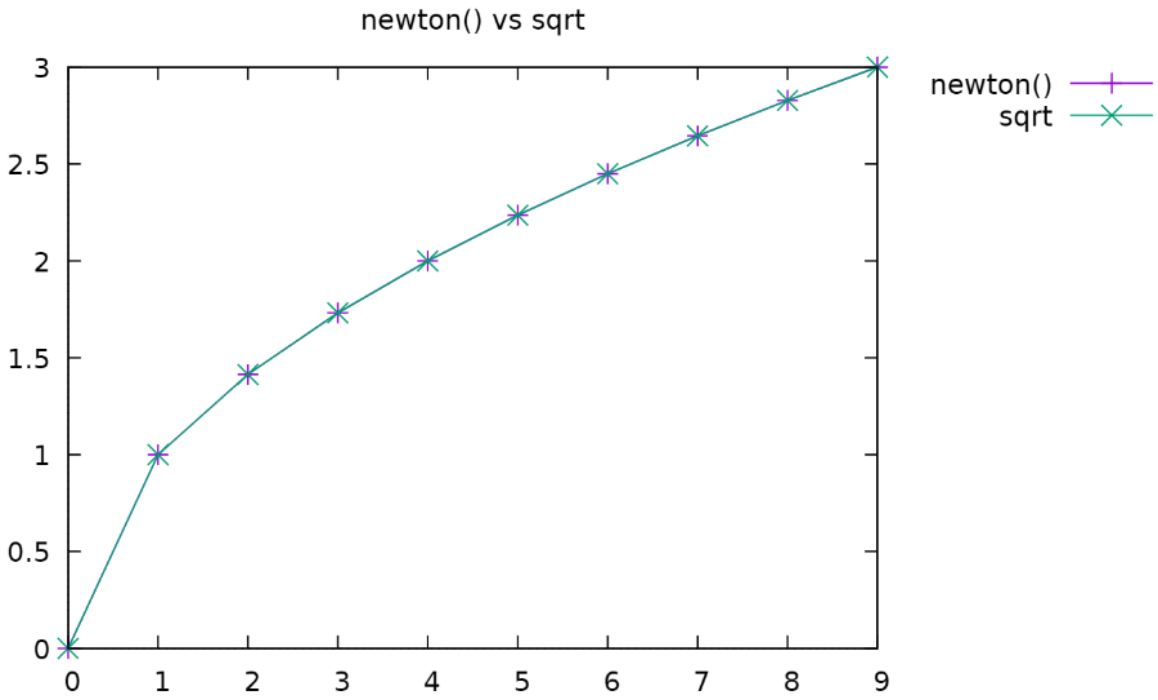
3 Analysis

E.c Graph



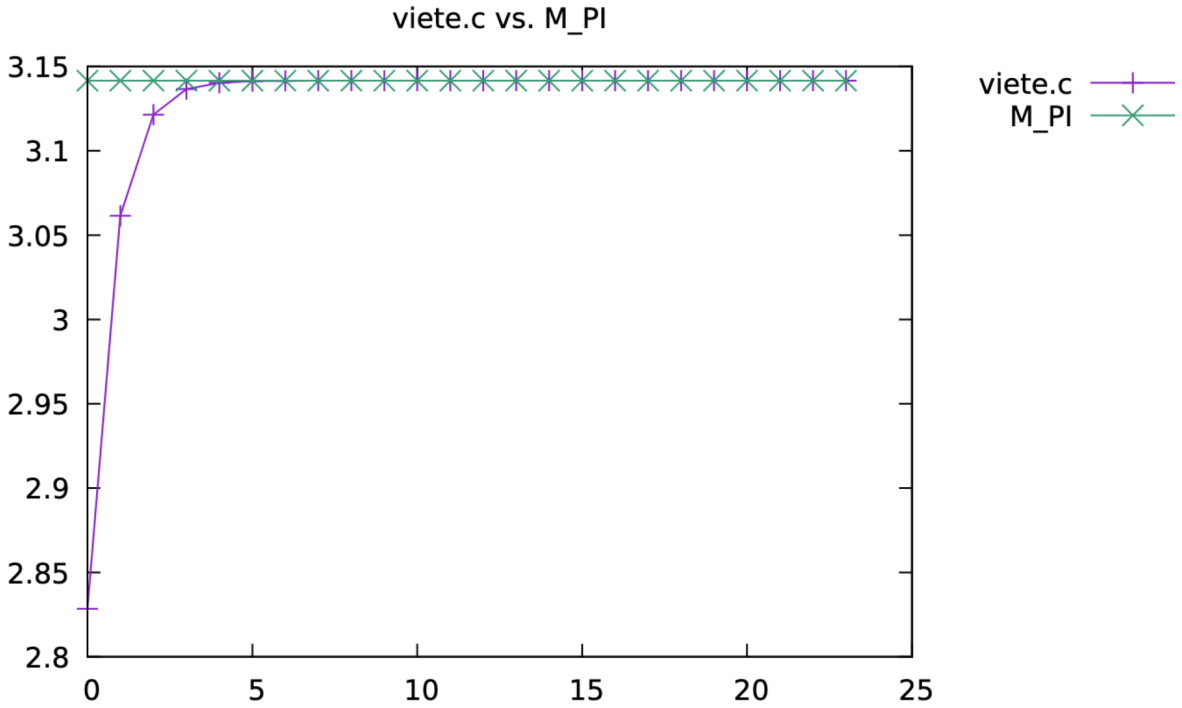
From this graph, we can see that my e() function quickly approaches the constant e as shown by the math.h constant M_E. After approaching M_E, e() seems to approximate the line extremely closely until the two lines become indistinguishable. As such, it can be seen that my code is very accurate to the fundamental constant e.

Sqrt_newton graph



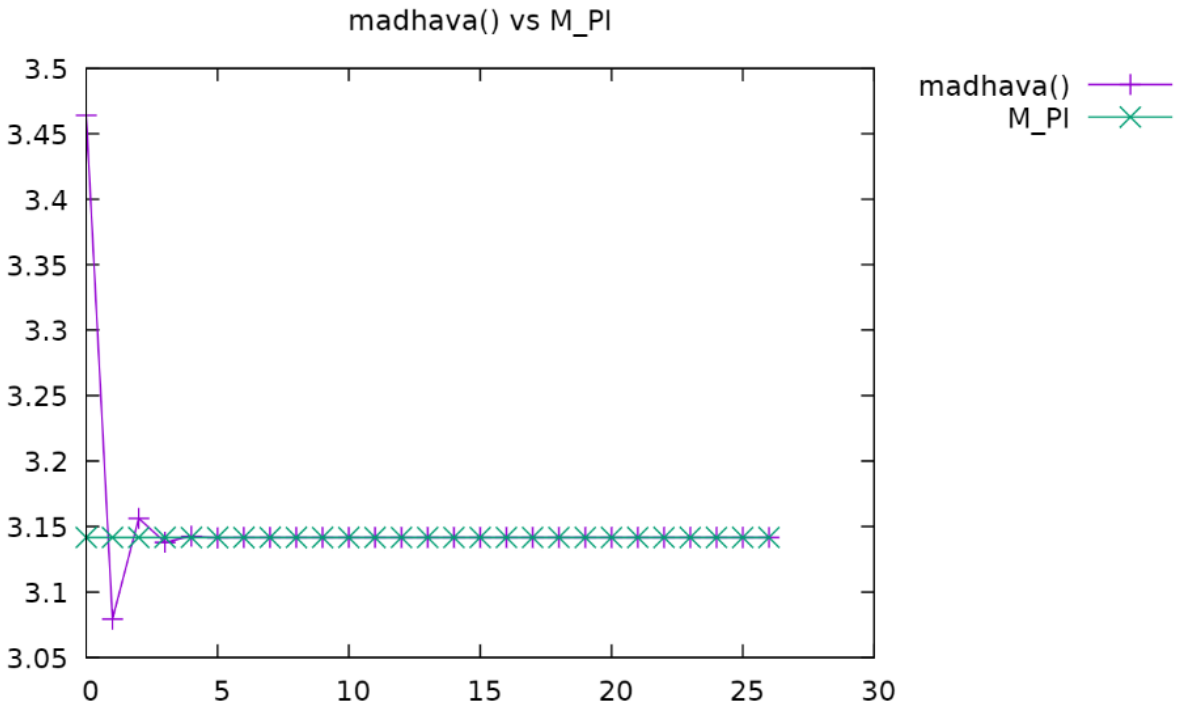
From this graph. We can see that my newton() function and the actual sqrt() function from the math.h library are identical from x=0 to x=9. Both functions follow the exact same path when plotted. As such, we can see that sqrt_newton() is extremely accurate to the math.h function.

Viete.c graph



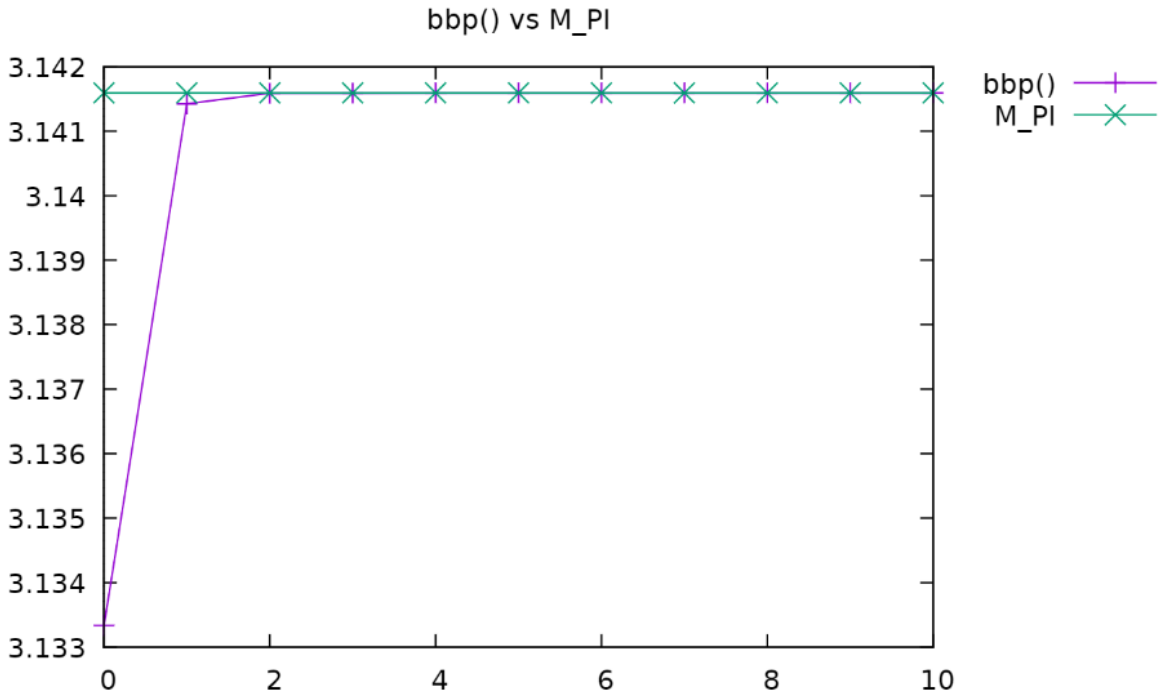
This graph shows how viete quickly approaches pi after starting from the value 0. Once it reaches what appears to be 3.14, the line quickly flattens until it is extremely close to the math.h library definition of pi (M_PI).

Madhava.c



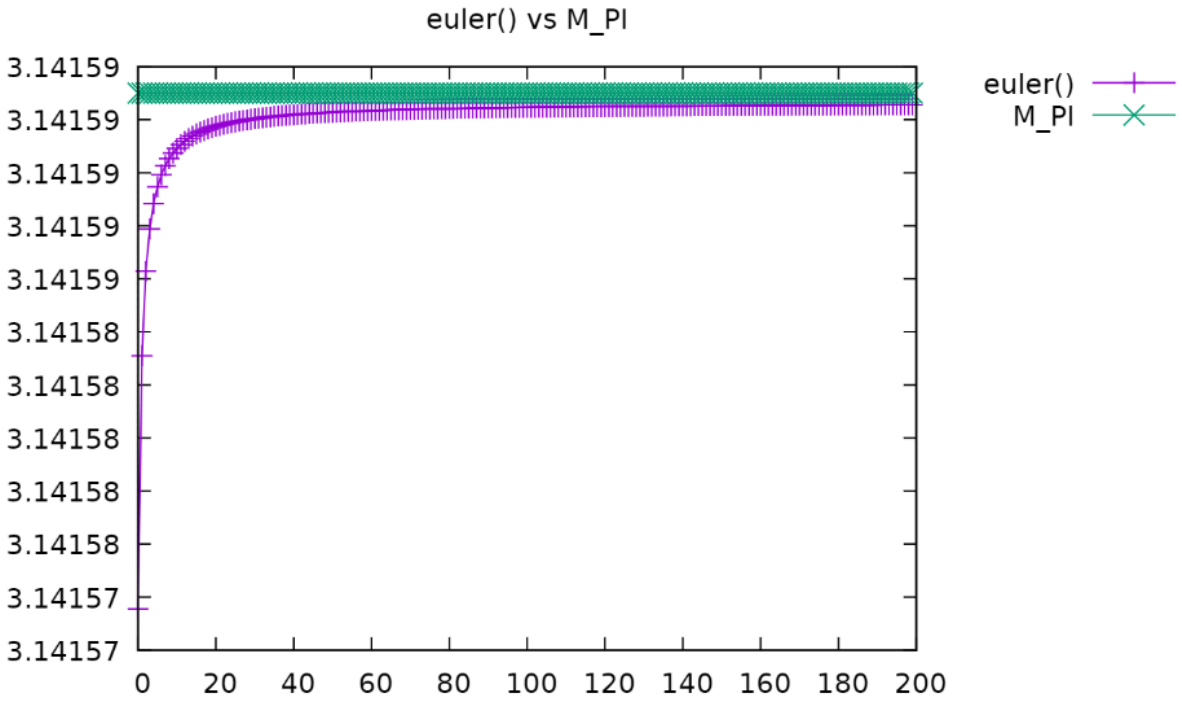
From this graph, we see that the madhava series starts at a larger number before correcting itself into one that approximates pi. After passing x=3, the line quickly straightens until it mirrors the line created by the constant M_PI from math.h. Though it starts off wildly inaccurate, it gets closer and closer to pi as the number of iterations increase.

Bbp.c



From this graph, we see that pi_bbp() starts at zero before making a large jump closer to pi at x=1. Then it quickly straightens out to better approximate M_PI. We can see from the number of iterations it takes that the Bailey-Borwein-Plouffe formula is very fast in getting an accurate representation of pi.

Euler.c



This graph is incremented in steps of 50,000 which serve to show the contrast between pi_bbp()'s efficiency and this program's efficiency. While bbp() only took 10 iterations to accurately represent pi, euler's solution instead took 10,000,000. However, though it took many more iterations, Euler's solution still accurately represents pi as can be seen by the way the curve approaches and mirrors M_PI.

4 Conclusion

These graphs show a variety of methods in approximating the square root along with the fundamental constants e and π. After analyzing the graphs of all 6 functions, we can draw some conclusions based on their results. The graph of the approximation of e shows how my e() function starts at 0, but quickly approaches and approximates the constant M_E as defined in math.h. In fact it only takes 16 iterations to reach e. My square root graph is even more accurate, as the sqrt_newton() function is identical to the sqrt() function from math.h at each point on the graph. When approximating π, my viete graph started at 0 before quickly approaching M_PI. It took less than 25 iterations to mirror M_PI. Similarly, my pi_madhava() function took 27 iterations before it approximated M_PI, though it did so in a different manner. Instead of starting at y=0, it instead started at y=3.4 before dropping to y=3.07 and then getting closer to π. Due to the nature of the madhava series (which bounces between negative and positive exponents), this is not an unexpected outcome. In contrast to the more than 20 iterations needed by the viete and madhava functions, the bbp formula only required 10 iterations before approximating M_PI. In fact, pi_bbp() started at y=0 before jumping in value to y=3.1415 at the next iteration. In stark contrast to the timely results of pi_bbp(), Euler's solution took 10,000,000 iterations to approximate M_PI. Though it got there in the end, it was an extremely large number of computed terms. In conclusion, each function did their job in approximating their target values, though some functions did it faster and more efficiently than others.