A Slice of Pi

Examining the written numerical library

Written by: Ralph Miller

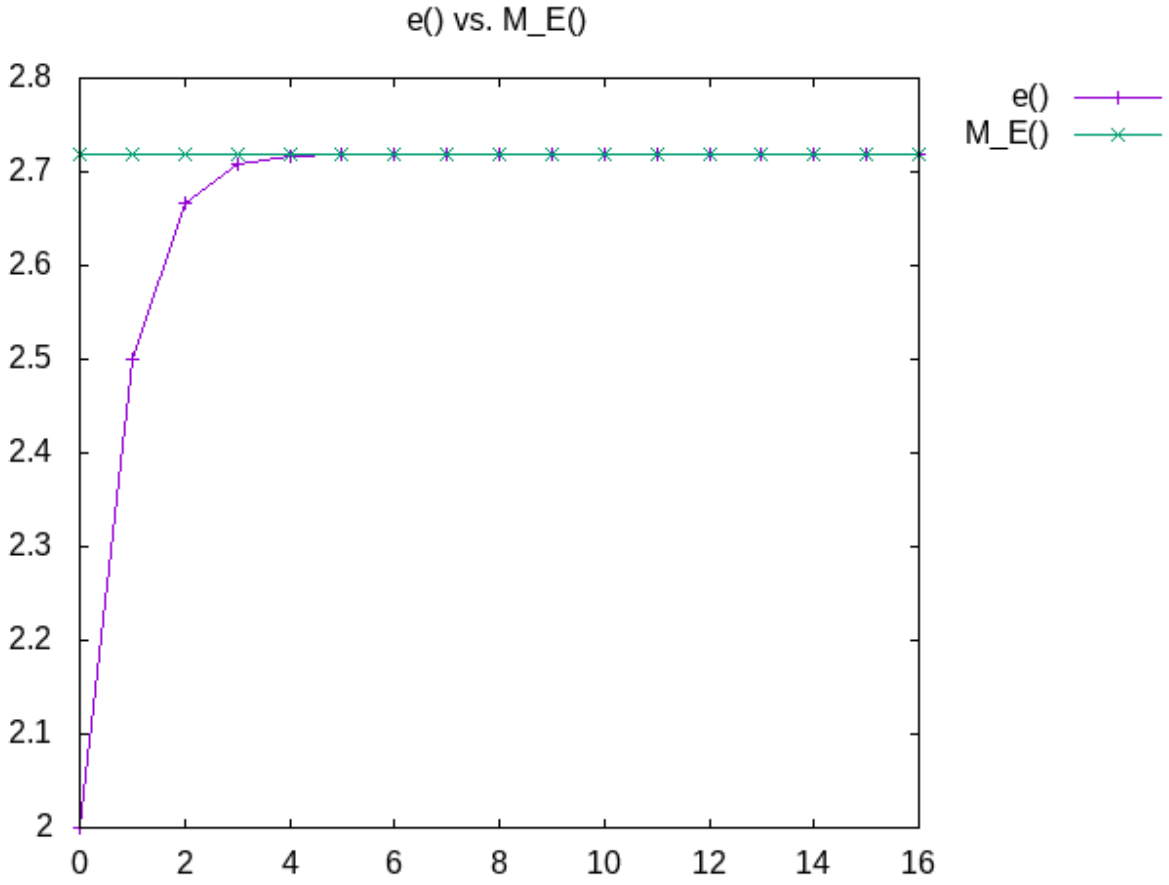
In this assignment I wrote a numerical library that mimicked the <math.h> library in c using different strategies to approximate e^x, $\sqrt{x}$, and π. The test harness compares my function approximations to the math libraries output. Printing each and the differences of the two. In this write up I will examine these differences and try to explain them to the best of my abilities.

---

e^x

To approximate e^x I used the Taylor Series, completing the summation when the individual term is no longer greater than Epsilon, $1e^{-14}$.

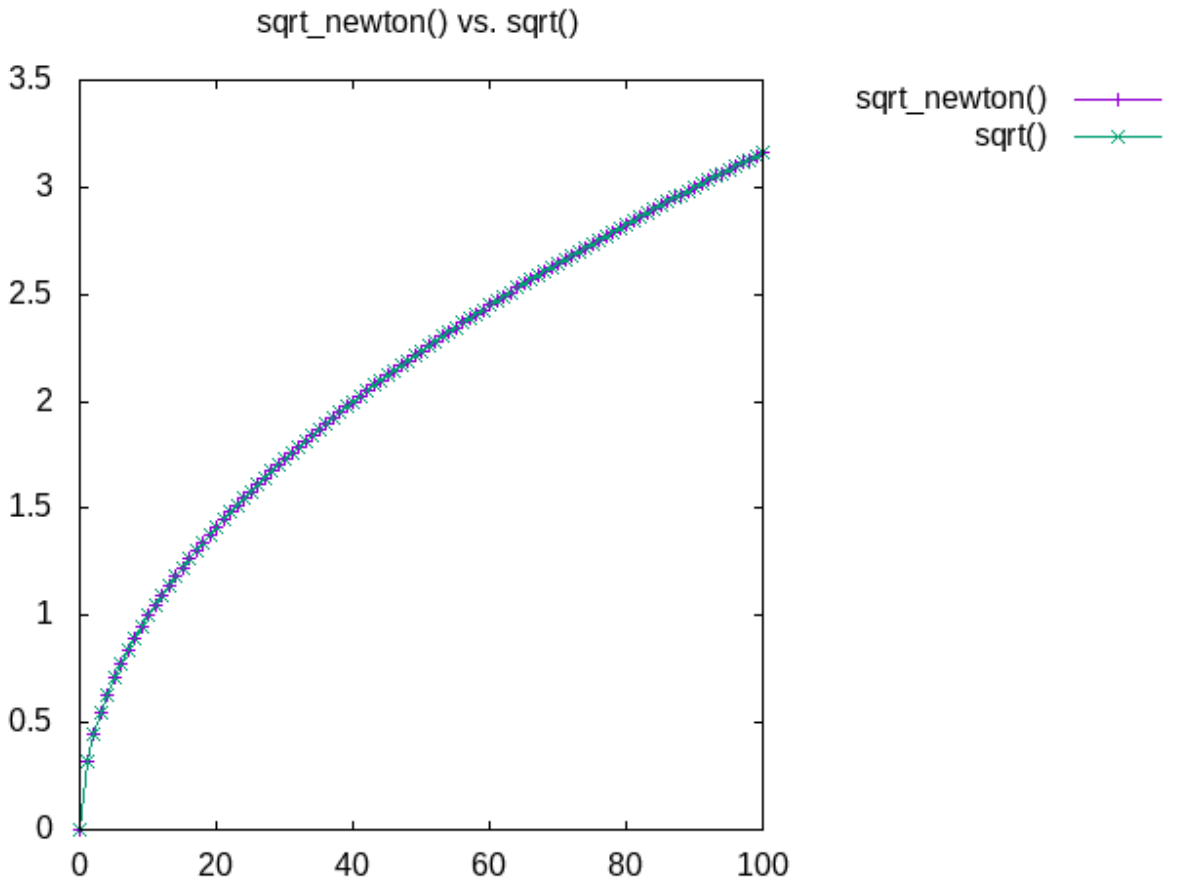
The result produced is completely accurate to the 15th decimal place with no difference between my implementation and the math libraries implementation. It took 17 iterations to produce my approximation. This is a pretty quick approach as the individual term approaches $1e^{-14}$ very fast as shown by the graph below. Within 4 increments the approximation is very close.



$\sqrt{x}$

To approximate $\sqrt{x}$ I used the Newton-Rapson method of computing the inverse of x^2. Completing the approximation when the difference between consecutive approximations are within $1e^{-14}$.

In my test harness I calculated $\sqrt{x}$ from [0.0, 10.0] with 0.1 step increments. Graphed below is my approximated $\sqrt{x}$ vs the math libraries approach. The results were accurate for every result except $\sqrt{0}$ in where the difference with the math library was $7 * 10^{-15}$. I think this error could be avoided by hard coding a case for $\sqrt{0}$ that returns 0.

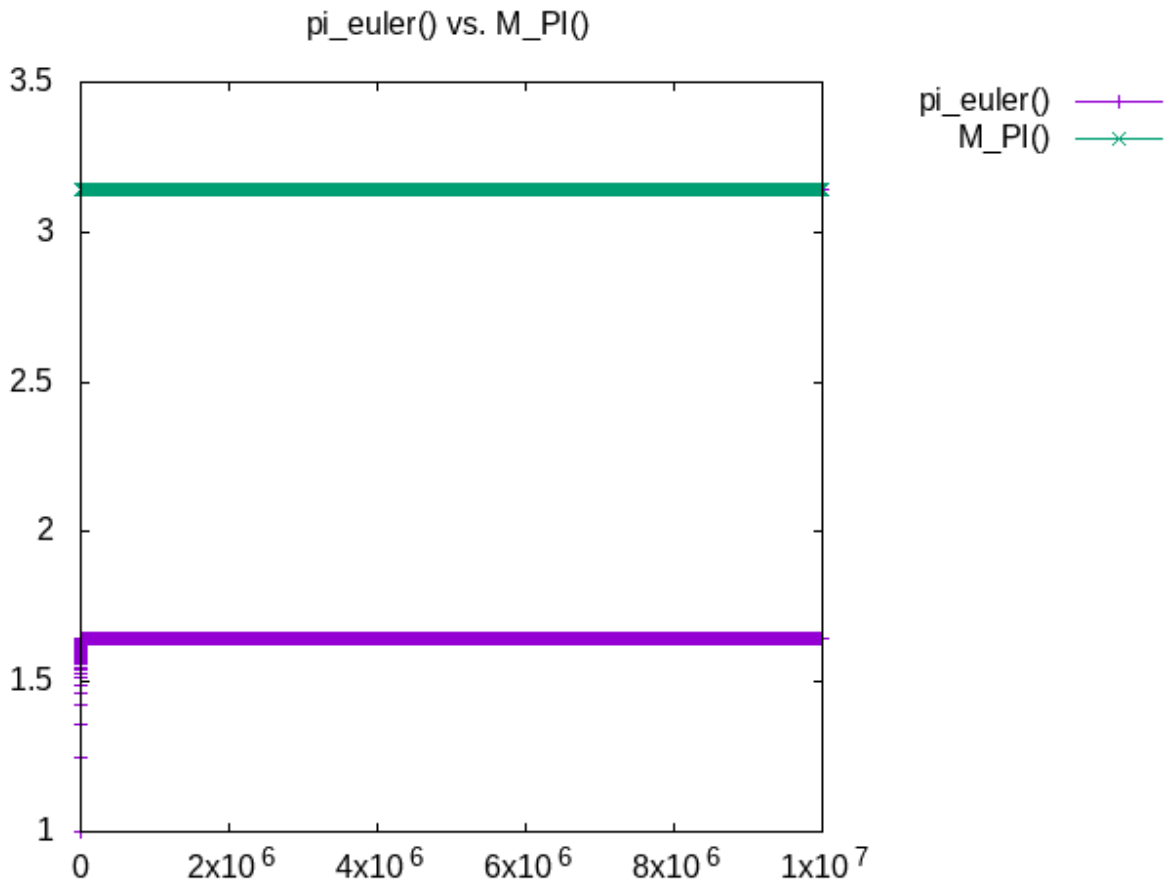


π

1. Euler's Solution

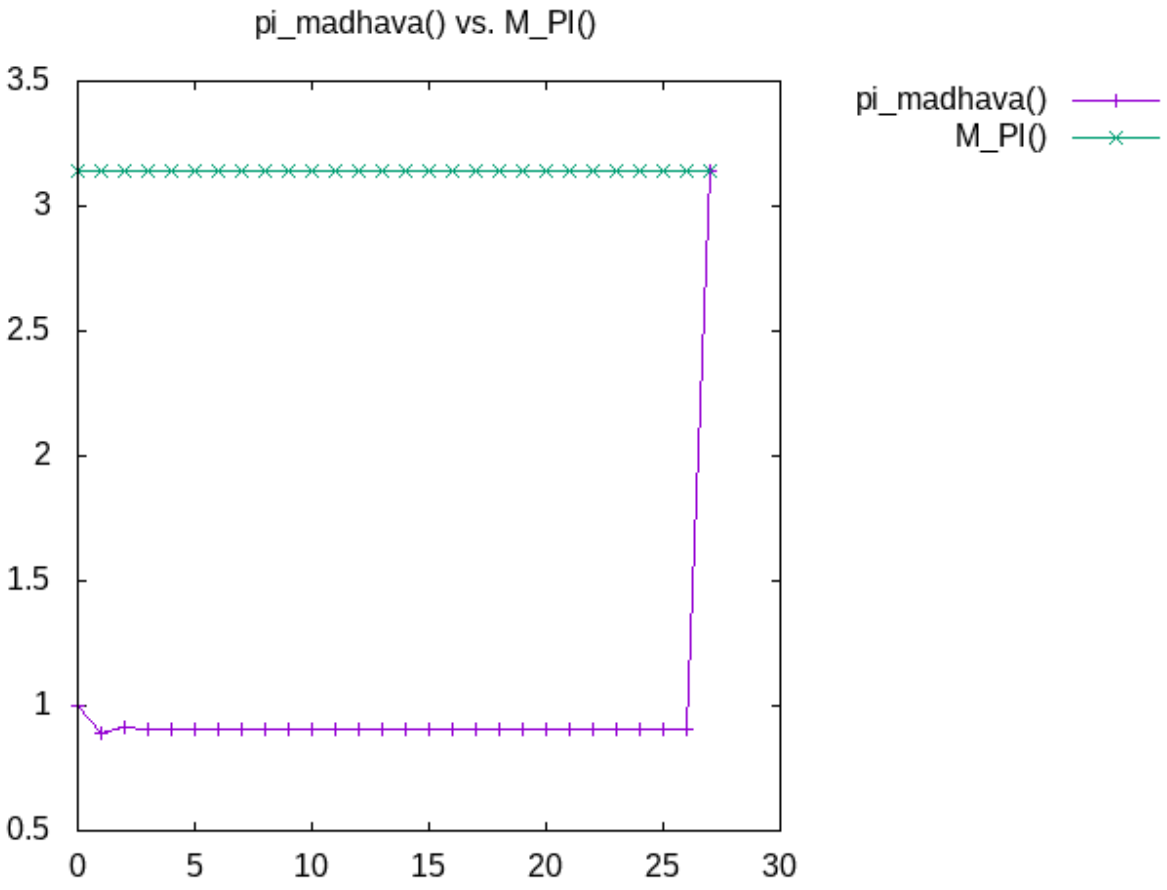
Using Euler's Solution I was able to approximate π correctly within 7 decimal places, with a difference between my approximation and the math libraries of: 0.000000095493891. I concluded that the difference is due to two factors.

First, within the calculations I used my approximation for $\sqrt{x}$ to get the result for π. Which is not completely accurate compared to the math libraries implementation. Secondly, I believe that the math libraries approach to calculating π uses a faster and more accurate formula. As it took 10,000,000 iterations to complete the series. By analyzing the graph, I believe that it is possible to get a faster but more inaccurate solution by stopping the approximation when it approaches 1.6, where the approximation stays around until the last iteration where the sum is multiplied by 6 and the $\sqrt{sum}$ is taken.



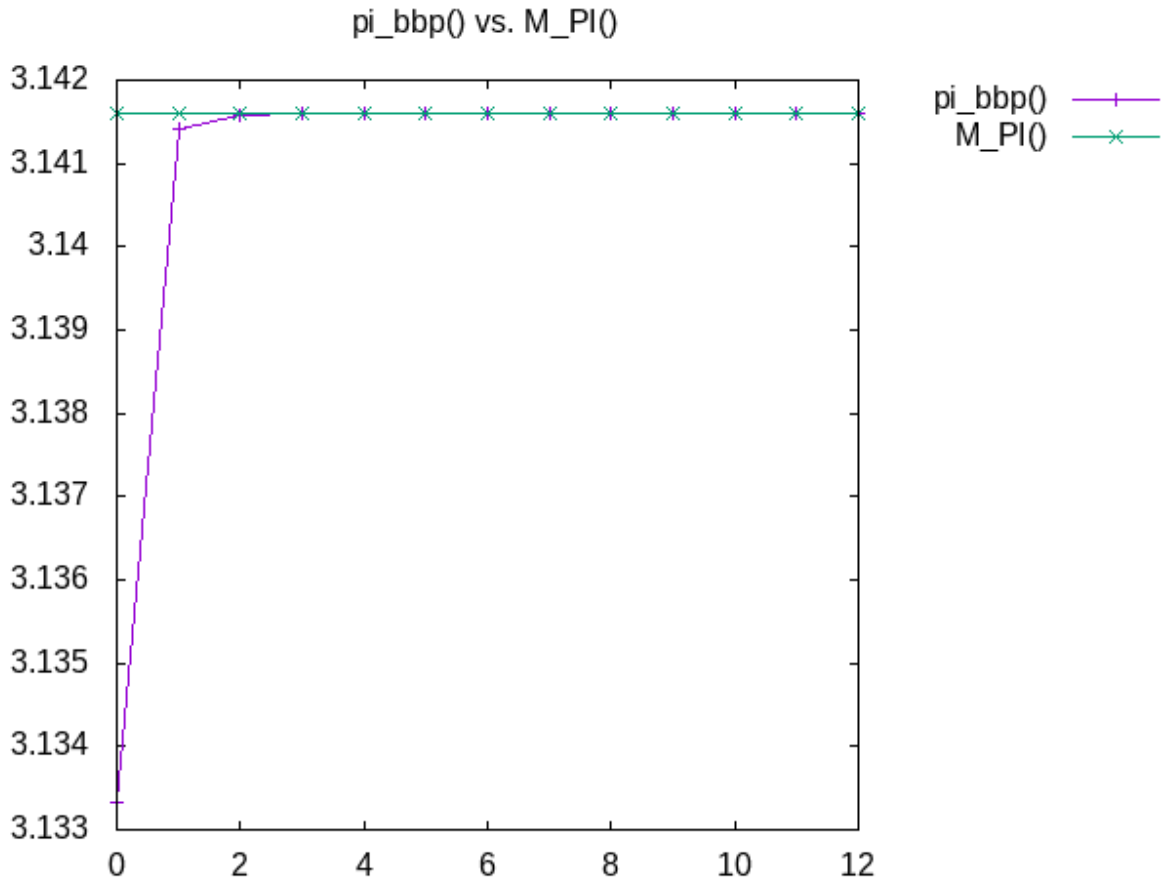
2. The Madhava Series

Using the Madhava series I was able to approximate π correctly within 14 decimal places in 27 iterations, with a $7 * 10^{-15}$ difference from the math libraries implementation. I concluded that the discrepancy between the two implementations also has to do with the inaccuracies of my approximation of $\sqrt{x}$ used in the calculations to compute π. After analyzing the graph I believe that a faster but more inaccurate approximation can be made if the summation is stopped around the 5th iteration and then is multiplied by $\sqrt{12}$ to get π.



3. The Bailey-Borwein-Plouffe Formula

Using the Bailey-Borwein-Plouffe formula I was able to approximate π correctly within 15 decimal places in 13 iterations, with no difference between my implementation and the math libraries implementation. I concluded that this is a very quick and accurate approach as shown by the graph below. In addition, the outside term of the equation approaches $1e^{-14}$ extremely fast and does not use an approximation of $\sqrt{x}$ to get the result.



4. Viete's Formula

Using Viete's formula I was able to approximate π accurately within 14 decimal places in 24 iterations, with a $4 * 10^{-15}$ difference between my implementation and the math libraries approach. I concluded that this difference is due to the inaccuracy of floating point division as the last step for solving for π is to divide 2 by the sum to get the result.