Assignment 2: A Little Slice of Pi

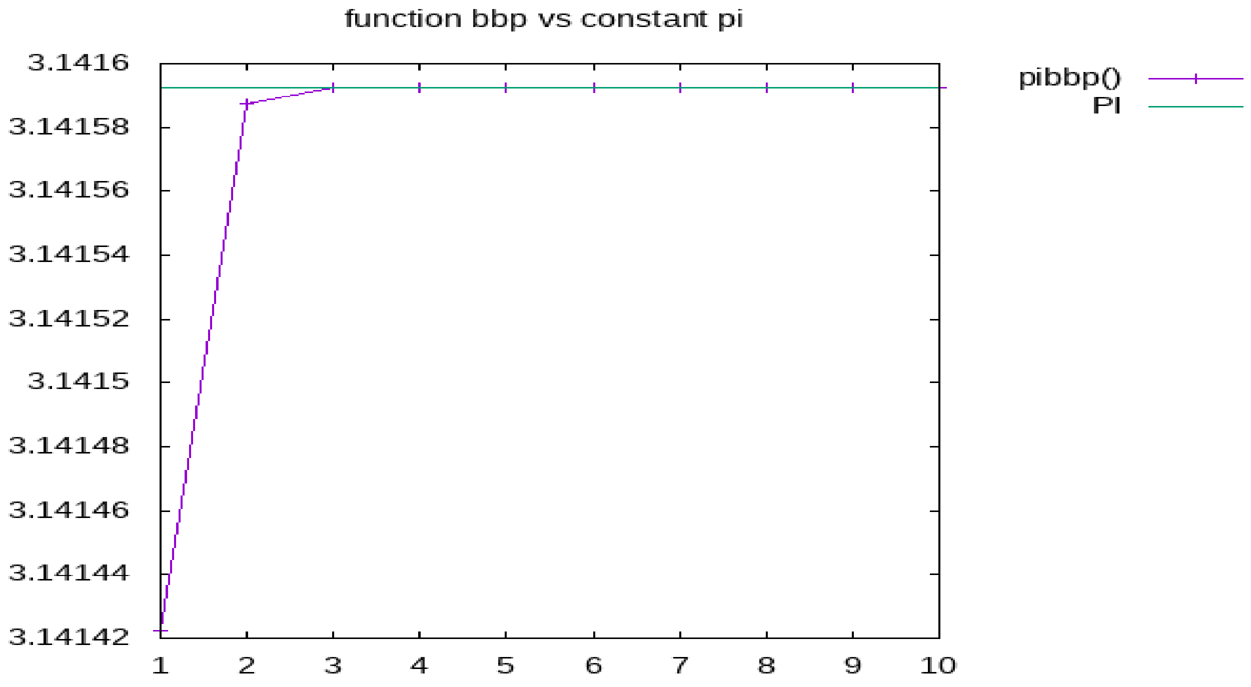
WRITEUP.pdf

Nathan Ong

This program is a math library that contains various functions that calculate **π**, Euler's number (*e*), and the square root of a given argument. The formulas used for the above operations are as follows:

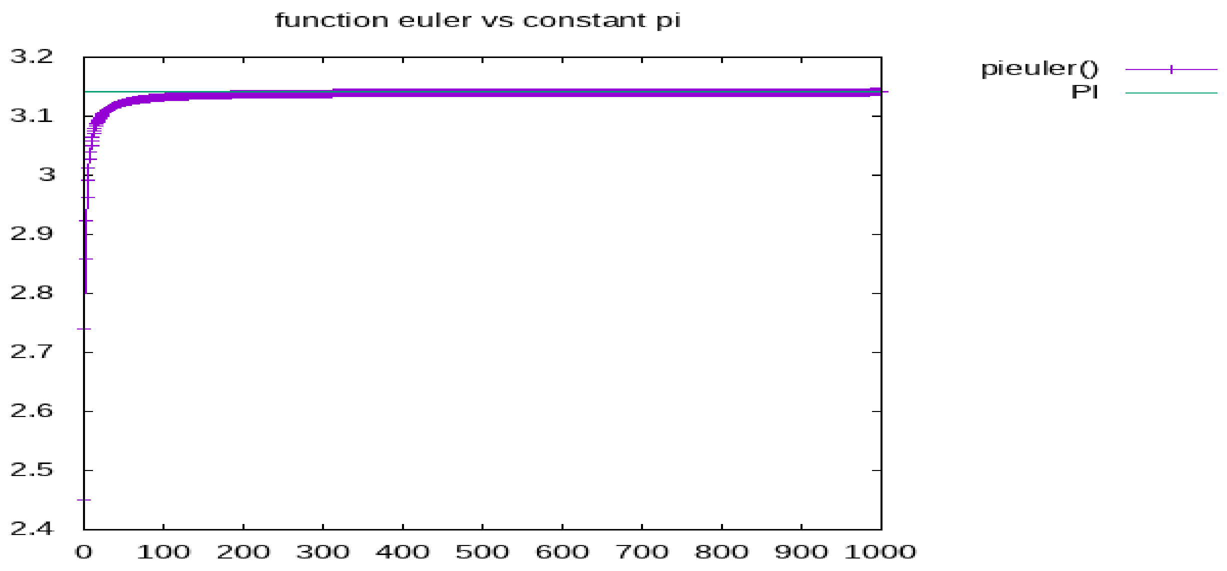
1. Bailey-Borwein-Plouffe Formula: $p(n) \sum_{k=0}^{n} 16^{-k}(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k-6})$
2. Euler's Number (e): $\sum_{k=0}^{\infty} \frac{1}{k!}$
3. The Madhava Series: $\sum_{k=0}^{\infty} \frac{-3^{-k}}{2k+1} = \frac{\pi}{\sqrt{12}}$
4. Newton-Raphson Method $(\sqrt{x})$: $\mathrm{x}_1 = 1.0, x_{k+1} = \mathrm{x}_k - \frac{f(x_k)}{f'(x_k)}$
5. Viete's Formula: $\frac{2}{\pi} = \prod_{k=1}^{\infty} \frac{a_k}{2}, a_1 = \sqrt{2}, a_k = \sqrt{2 + a_{k-1}}$
6. Euler's Solution to the Basel Problem: $p(n) = \sqrt{6 \sum_{k=1}^{n} \frac{1}{k^2}}$

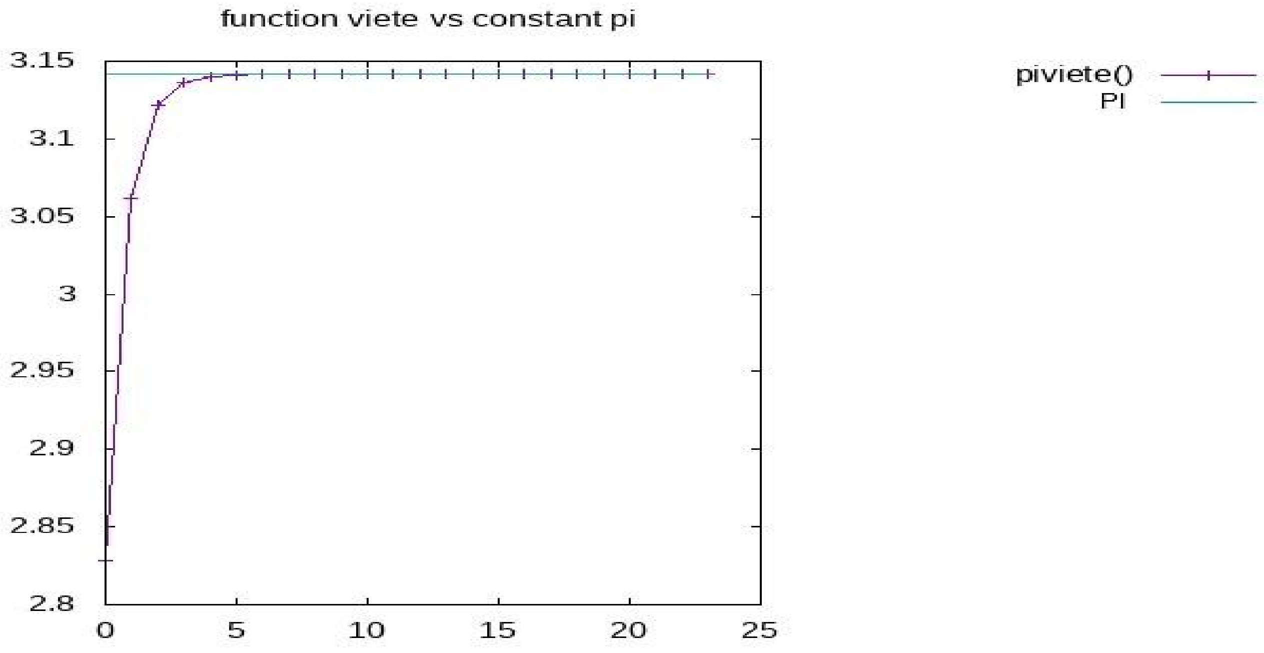
First off, the Bailey-Borwein-Plouffe formula, or BBP (which is what it will be referred to as for the rest of this document), is a summation formula that was discovered in the mid-'90s and was named for its discoverer and the authors of the article in which it was shown.



The graph above shows the graph of the BBP formula's computed terms as run by the math library. The graph shows that coming up from 0 computed terms (assumed value is 0), the formula immediately goes towards pi with a relatively accurate floating-point number, being within a ten-thousandth of the irrational constant. As the number of computing terms goes up, the term becomes more and more accurate, as shown by the second and third computed terms, with the third being so close to the constant that we cannot see it more accurately. After the function stops running at the epsilon limit, the computed term is accurate up to 10^{-14}.

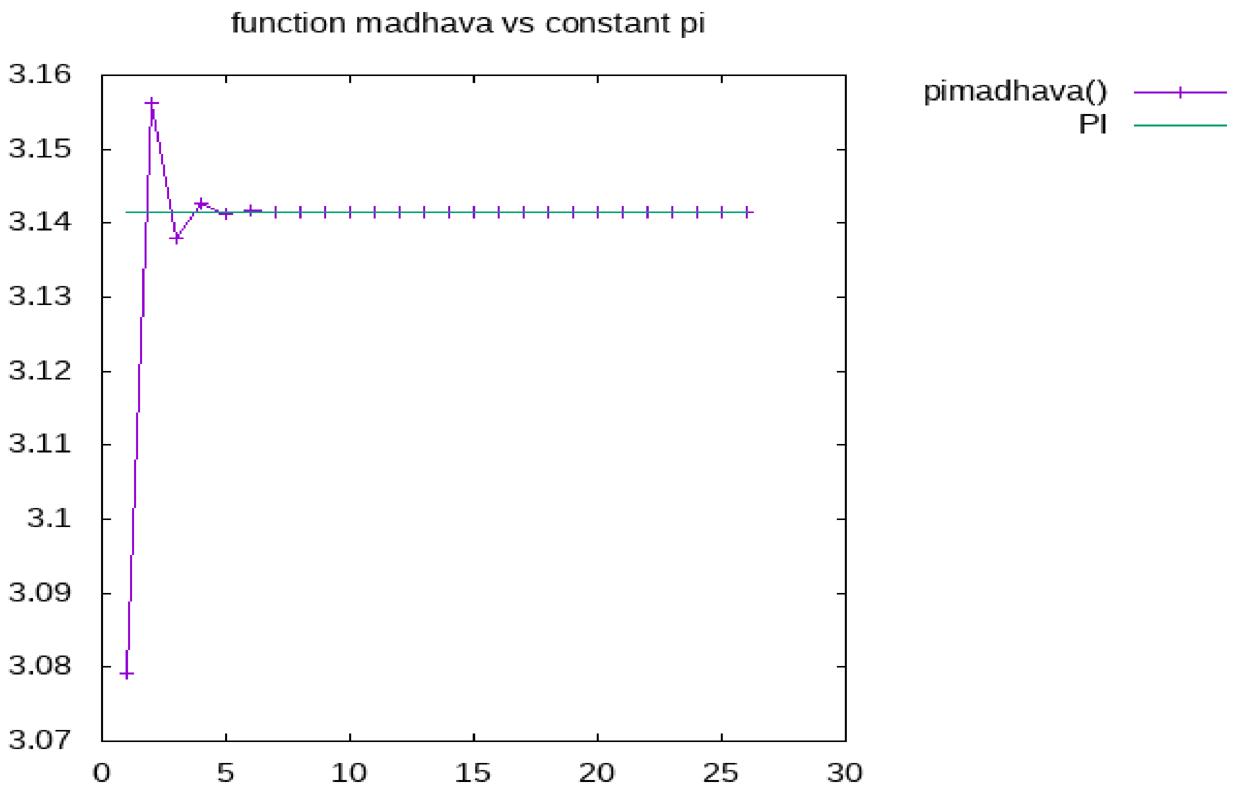
Most of the other **π** functions follow a similar pattern with their computed terms, including Viete's formula and Euler's solution to the Basel problem, as shown below.





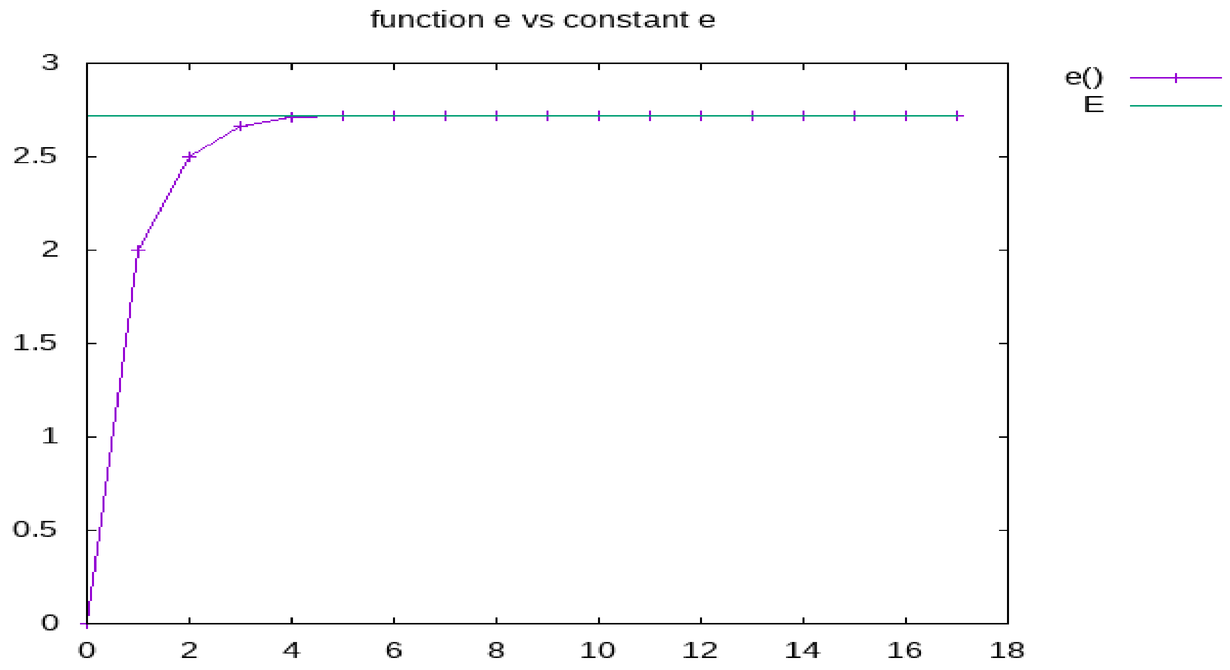
Although initially less accurate than the BBP formula, both functions calculate more iterations to get to **π**, and consequently, can be assumed to be more accurate later on since we can assume that the computed terms begin to increase by less and less, requiring more computations to get to the target digit.

The only exception to this pattern is the Madhava series, an older formula compared to the rest, being dated back to the 14th century.

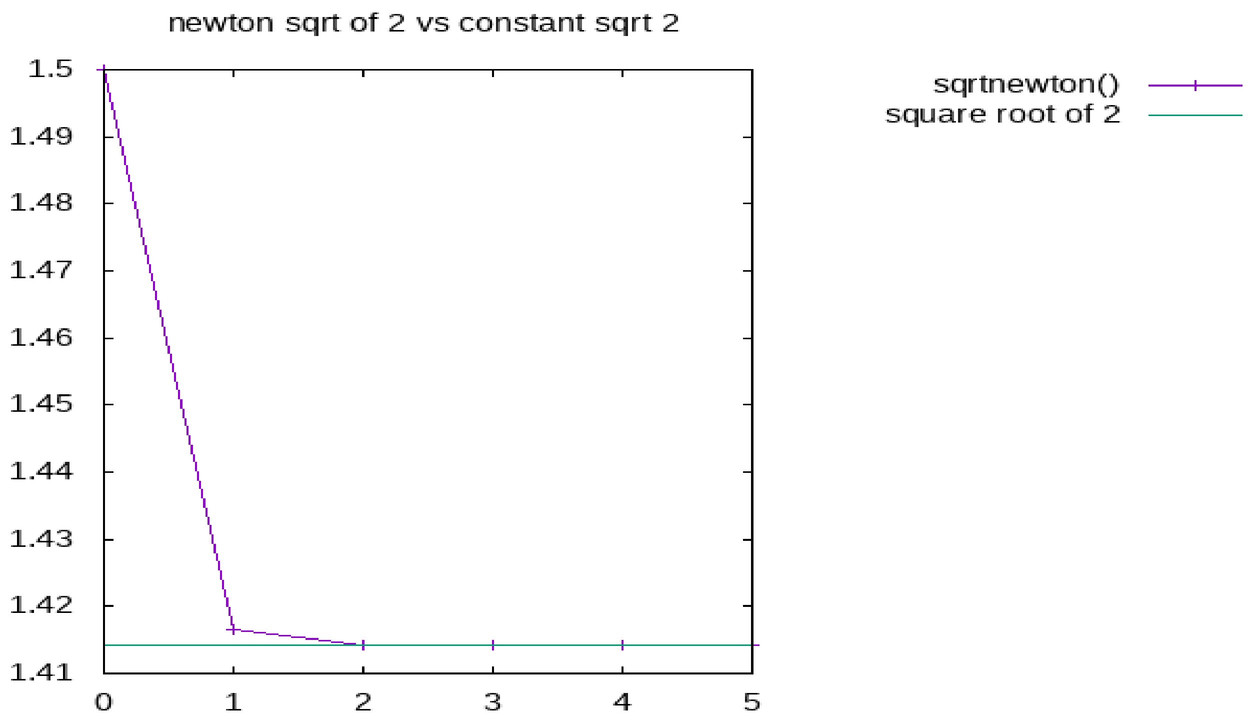


In contrast to the other three **π** functions, the Madhava series seems to be going back and forth between being under **π** and being over **π**, kind of like a binary search, which keeps splitting the range of search in half as it zeroes in on the target value. The computed terms get more and more accurate until the epsilon limit is reached at approximately 25 terms.

Moving on to Euler's number, the formula used to calculate it in the library is the basis of the natural logarithm and was discovered by Jason Bernoulli. The resulting Taylor series is a summation of fractions that increasingly and infinitely get smaller and smaller, as the denominator is a factorial of the index. As shown in the graph below, the summation follows the pattern shared across most of the **π** functions, where the computed term is increasing in smaller and smaller increments as the number of computed terms increases.



And finally, the last function, the square root. The square root function uses the Newton-Raphson method to compute the square root of the given argument. This method, similarly used in other coding languages as the basis of their math library function, multiplies adds the current term and the quotient of the current term and the given argument. The resulting number is then multiplied by 0.5. This method is surprisingly quick and accurate, to the point where most of the values tested in the math library were computed in either 6 or 7 terms until the term reached the epsilon limit. For the purposes of this document, we will use the square root of 2 as an example in the graph to show the process of the function. As shown in the graph below, the first computed term immediately descends close to the target value, which is the square root value as calculated by the C math library's square root function. Since we cannot see the terms in the graph, we can assume that the behavior of the graph remains the same, but the term is getting more and more accurate to the point where the next computed term only goes up by a very small fraction.


newton sqrt of 2 vs constant sqrt 2
sqrtnewton()
square root of 2
1.5
1.49
1.48
1.47
1.46
1.45
1.44
1.43
1.42
1.41
0
1
2
3
4
5