Four Slices of π

Arnav Nepal

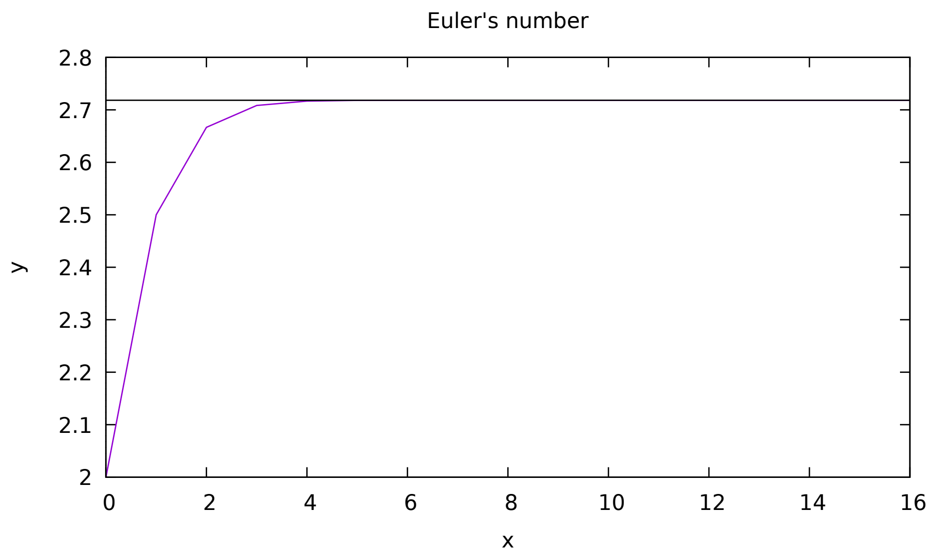
January 30, 2023

1 Introduction

In this assignment, we were tasked with designing and implementing a small number of mathematical function for the purpose of approximating π in different ways. These including a square root functions and a function for Euler's number "e". We approximated π by implementing functions based on 4 formulas or series. This report will analyze the various different formulas we were tasked with implementing and their effectiveness.

2 Math Library: e and $\sqrt{}$

In order to implement some of the π approximation series, such as the Madhava series and the Viete series, we had to first make a square root function. Additionally, we were also tasked with approximating e. We were provided example programs (Written by Professor Long) for these mathematical function in the "portable.py" python file. Thus, I utilized this python code and "translated" it into C code for my square root and e functions. Since the square root and e functions were relatively simple, it was not too difficult for me to translate the python code to C. The biggest difference was, of course, the need to explicitly define types in C. One of the things I learned while implementing these functions was the need to explicitly place decimals in numbers. Otherwise, they would be treated as integers and cause the equation to fail since integer division and multiplication cannot result in doubles.



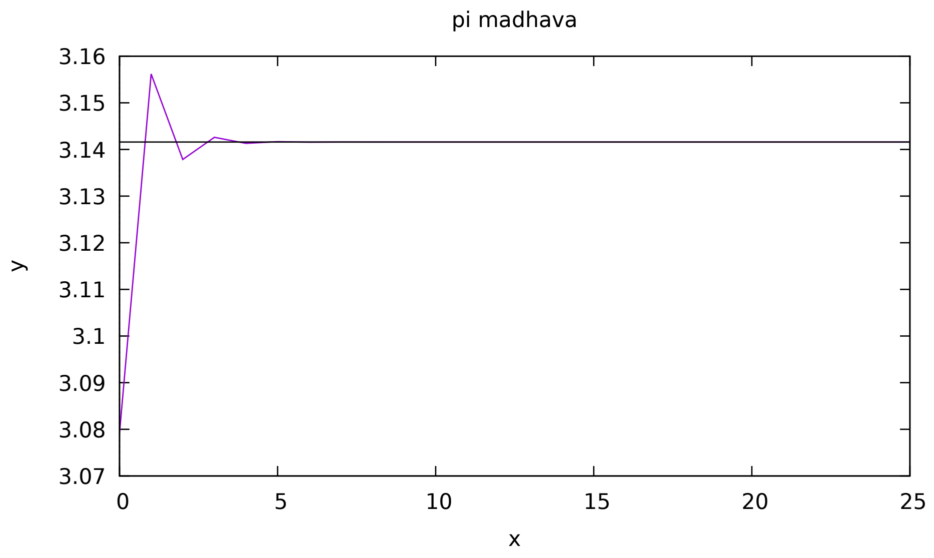
3 Approximating π

With a square root function completed, we can move on to implementing functions and formulas to approximate π. For each of the graphs below, x represents the number of terms.

3.1 The Madhava Series

The Madhava series was the first approximation formula that I implemented in C:

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

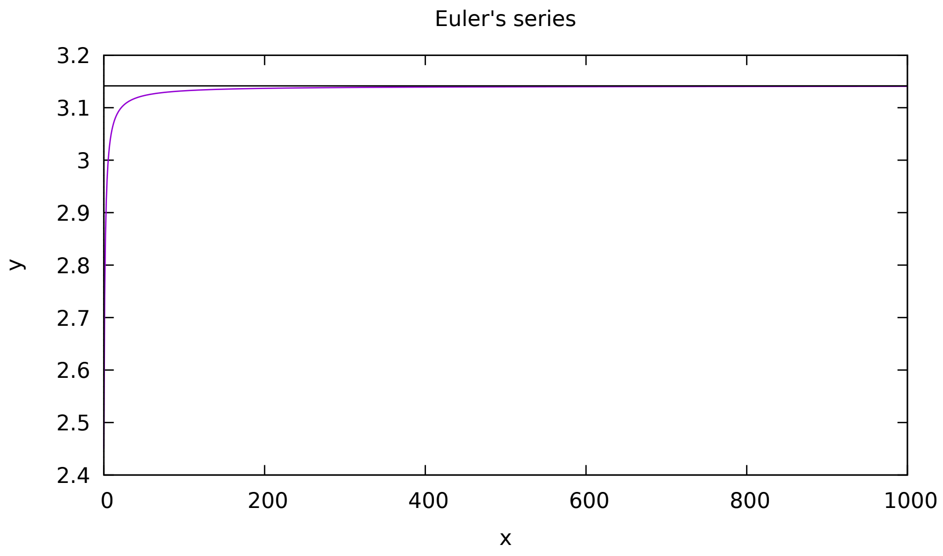


The Madhava series was a middle of the pack series. It took 27 terms to converge to 14 digits of precision as specified by ϵ. It executed instantly (in < .1 seconds). The graph shows that it becomes indistinguishable with the line $y = \pi$ within ~ 5 terms. The difference between the final value output by my Madhava series and the math library π was 7.0^{-15}, which makes sense since EPSILON only offer 14 digits of precision, and so the function exited before the 15 digit could be approximated correctly.

3.2 The Euler Series

The Euler Series was the second formula I implemented in C:

$$\sqrt{6\sum_{k=1}^{\infty}\frac{1}{k^2}}$$

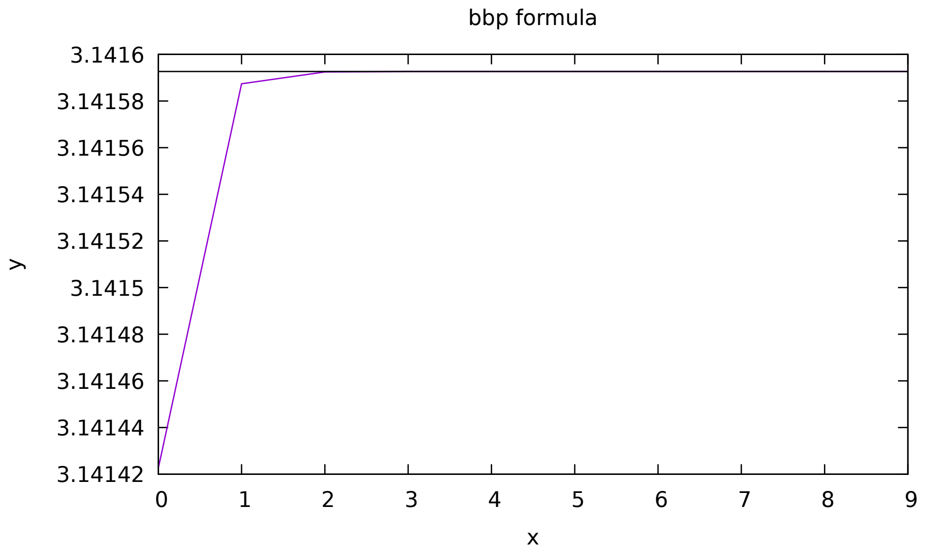


The Euler series was the slowest series. It took 10 million terms to converge to 14 digits of precision. It took relatively longer to execute than the other series, but was still instant (< .1 seconds) in human terms. As the graph shows, Euler requires a much greater number of computations to converge on π. Euler also had the greatest difference between the Library function and my function, with a difference of about $\sim 9.5^{-8}$. One reason for this might be the need for extra calculations after the main loop of the program, which might result in rounding errors.

3.3 Bailey-Borwein-Plouffe formula (BBP)

The Bailey-Borwein-Plouffe (BBP) formula was the thrid formula I implemented in C:

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

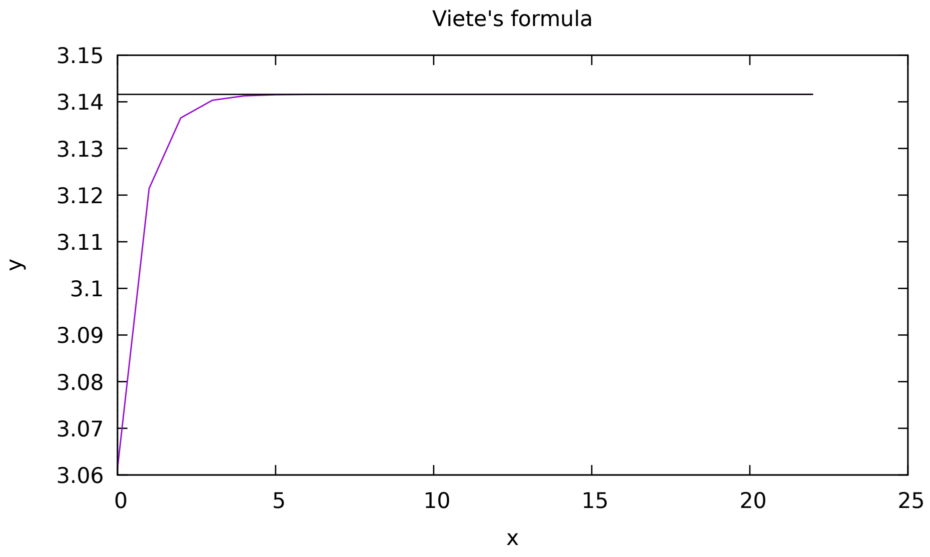


BBP was the fastest and most efficient formula. It took a mere 11 terms to converge within 14 digits of precision of π. Like the others, it executed instantly (in < .1 seconds). As the graph shows, BBP also has the advantage of starting very close to π, and merges with the π line in just 2 terms. There was 0 error for BBP, which might be because there was no need for extraneous calculations outside the loop that would have incurred rounding errors.

3.4 Viete's formula

Viete's series was the last formula I implemented in C:

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



Viete's series was similar to Madhava's in terms of efficieny. It took 24 terms to converge within 14 digits of precision of π. Like Madhava, it also executed instantly (in $< .1$ seconds) and also becomes indistinguishable with the $y = \pi$ line within ~ 5 terms. The final value output by my Viete function differed from π in the math library by 4.0^{-15}. This might be because Viete also required extraneous calculations after the main loop (with the sum being divided into 2), which might have caused rounding errors.

4 Conclusion

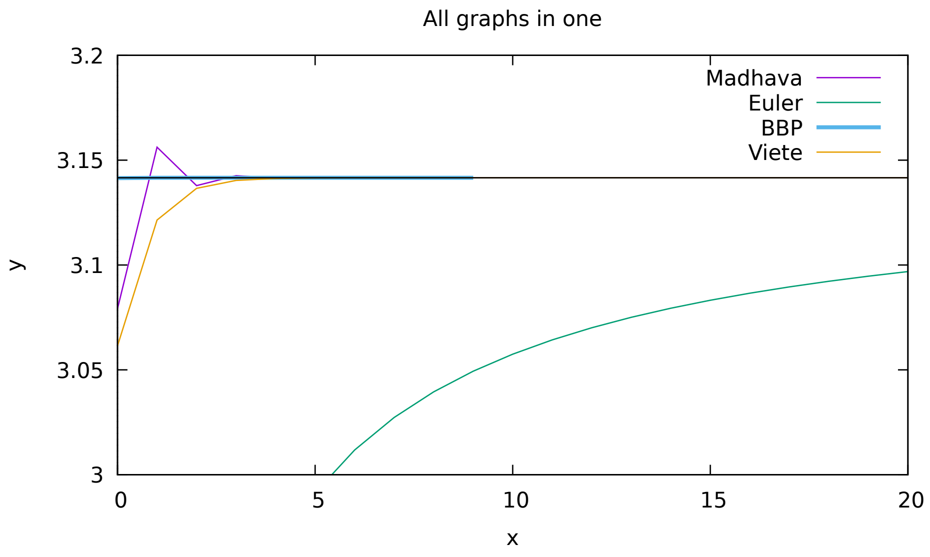


Figure 1: Plot of all formulas in one graph. BBP is thickened to faintly display the line since it converges so quickly.

This was an interesting and fun assignment. I've always wanted to know how supercomputers calculated billions of digits of pi, and while I'm sure they use more sophisticated algorithms, it was interesting to get a glimpse of that. In this assignment, I learned how to work with floating point numbers and how they interacted with integers. It also taught me the importance of pacing myself. While I finished all functionality yesterday, it still took me a while to finalize everything in teh assignment, including updating the design, writing this report, writing the README, etc. In future assignments, I will try to get done early rather than later, especially since assignments will more likely get harder in the future.