Analysis on Approximation of Irrational Numbers

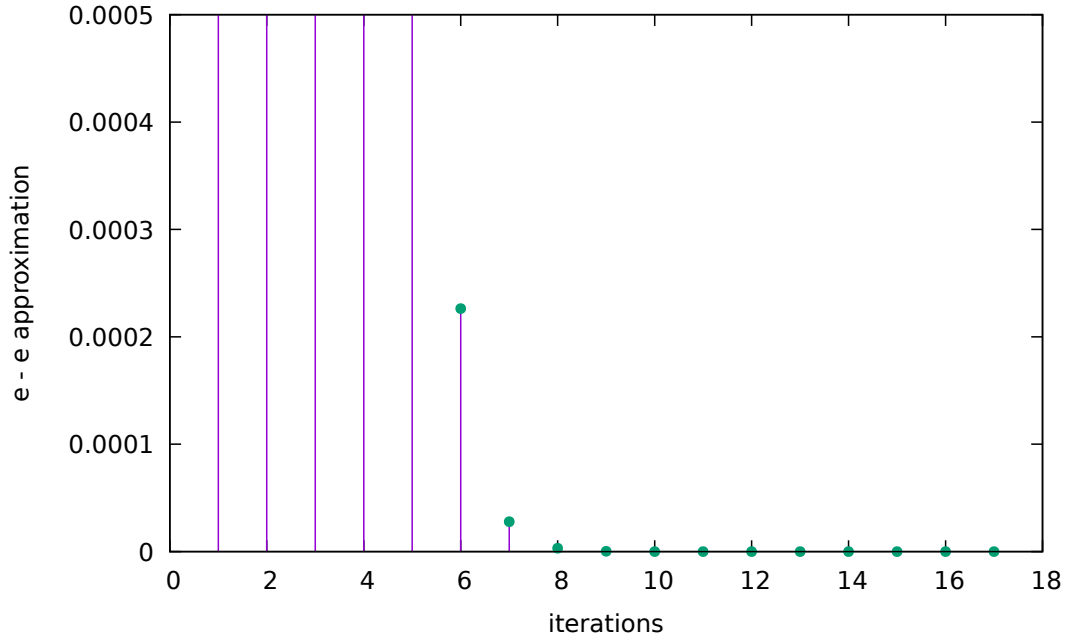
Hsiang Yun Lu

January 30, 2023

1 Calculating *Euler's number*

mathlib-test Output:

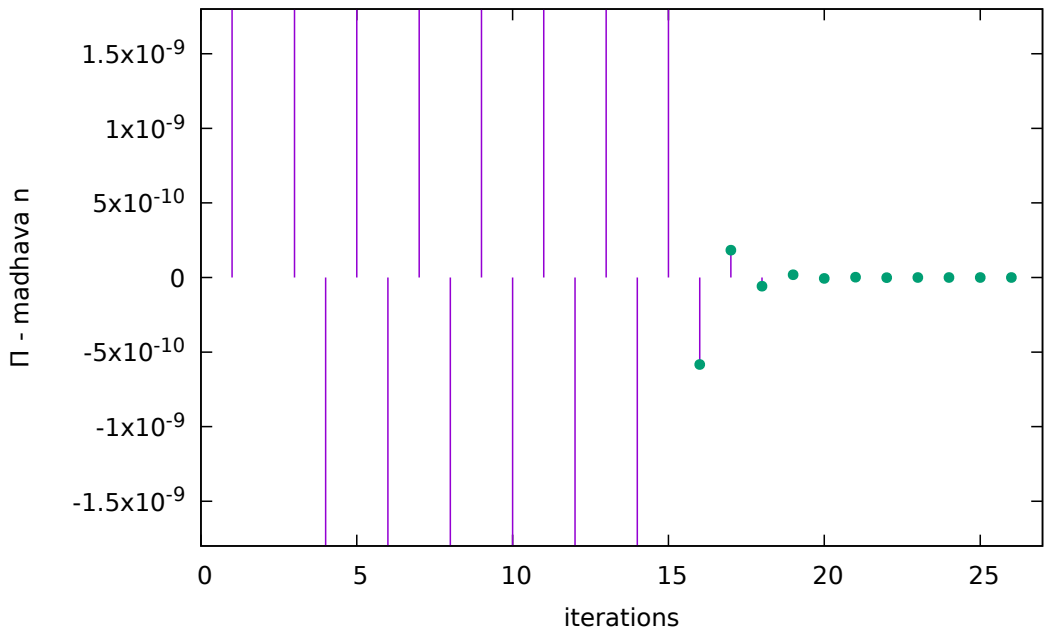
```
e() = 2.718281828459046, M_E = 2.718281828459045, diff = 0.000000000000000
e() terms = 34
```



2 Calculating π using Madhava series

mathlib-test Output:

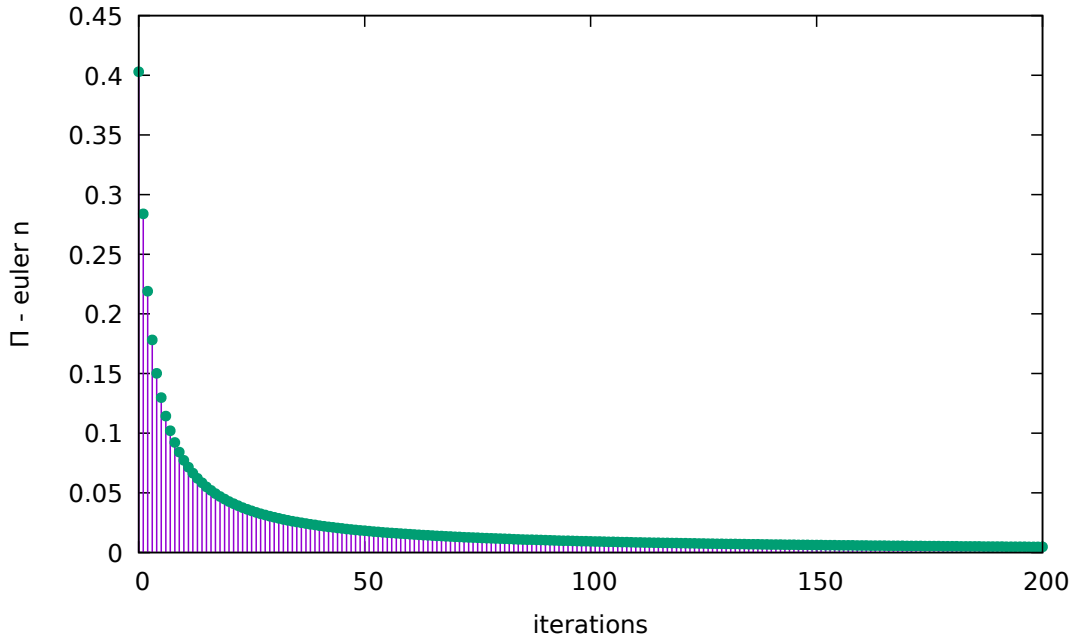
```
pi_madhava() = 3.141592653589800, M_PI = 3.141592653589793, diff = 0.000000000000007
pi_madhava() terms = 52
```



3 Calculating π using Euler's Solution

mathlib-test Output:

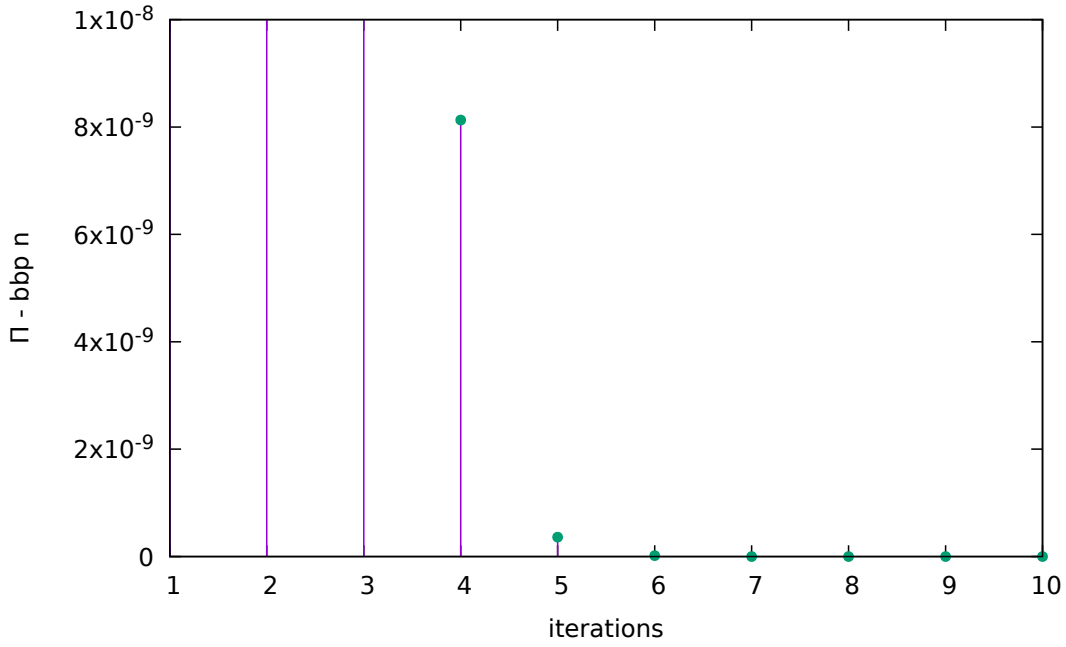
```
pi_euler() = 3.141592557608133, M_PI = 3.141592653589793, diff = 0.000000095981660
pi_euler() terms = 19897758
```



4 Calculating π using the Bailey-Borwein-Plouffe Formula

mathlib-test Output:

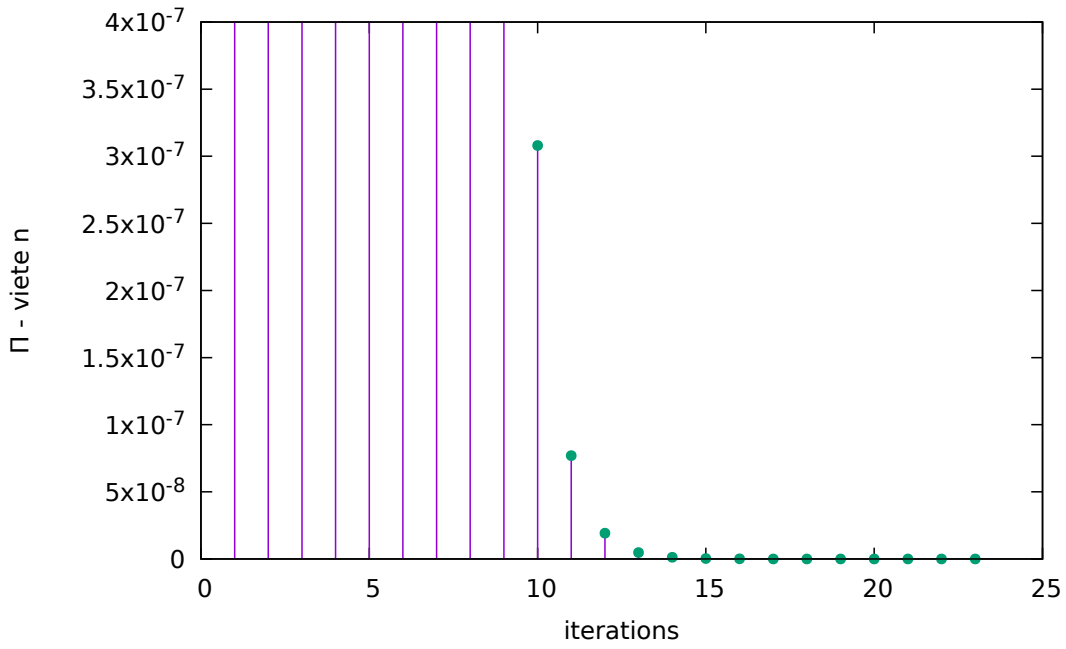
```
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
pi_bbp() terms = 20
```



5 Calculating π using Viète's Formula

mathlib-test Output:

```
pi_viete() = 3.141592653589789, M_PI = 3.141592653589793, diff = 0.000000000000004
pi_viete() terms = 46
```



6 Discussion

In all the math implementations of calculating π, Euler's Solution converges very slowly using 19897758 iterations, while other methods converge in around 30 to 50 iterations. This could result from the summation of the reciprocals of the squares of the natural numbers. Since the values of the reciprocals of the squares of the natural numbers decrease rapidly, the changes between each iteration are so little. Compared to the π constant from the `math.h` library, there is a difference of 0.000000095981660. This is the biggest difference among all the π approximation methods. The difference may have something to do with the square root calculation in the end.

The approximation results of Madhava series and Viète's Formula both show a small difference compared to the π constant from the `math.h` library, while Bailey-Borwein-Plouffe Formula shows 0.000000000000000 difference. Similarly, since newton square root function was used in Madhava series and Viète's Formula, the slightly higher difference could come from the apprximated newton square root method.

In Newton's method to approximate the square root, I received difference of 0.000000000000000 for all the number tested except for 0.000000. The result of 0.000000 :

```
sqrt_newton(0.000000) = 0.000000000000007, sqrt(0.000000) = 0.000000000000000,
diff = 0.000000000000007
sqrt_newton() terms = 694
```

It's confusing that the square root of 0.000000 is 0.000000000000007. The resolution of floating point numbers might be able to explain this. 0.000000 might not contain all 0 in the further digits but it won't show as a readable message to us.

7 Conclusion

After Assignment 1, this assignment again focuses on the floating point number approximation. After this implementation, I understood how approximation and floating point numbers work. The technique of building and using our own math library is also handy. I became more familiar with the logic and structure and the related commands.