Assignment 2: "A Little Slice of Pi"
WRITEUP

Vincent Liu
1/28/23

Introduction

Assignment 1, titled "A little slice of pi", instructs students to use various mathematical formulas to estimate the value of pi. The purpose of this assignment is to allow students to become further acquainted with C by creating a mathematical library similar to that of "math.h"; a mathematical library used in the C language (Note that equivalent versions of this library also exist in other programming languages). Students will estimate the value of e, pi, and square roots using specific mathematical methods as well as comparing the difference in accuracy between their outputs and math library's outputs. Students will also be expected to determine the efficiency of each of their programs through the use of an iteration counter in order to compare the efficiency of the various formulas. Overall, students will be expected to work with (at least) 6 formulas/tests.

Estimating the Value of '*e*'

Similar to the more widely known pi / 'π', '*e*' or "Euler's number", is also an irrational and infinite constant used in mathematical calculations. Euler's number plays an important role in mathematics, as it is the base of natural logarithms. Euler's number's first six digits are 2.71828, however it must be noted that is an infinite constant. Students are expected to estimate the value of Euler's number through the use of a provided formula in the assignment pdf. This formula ais as follows:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

The '.c' file that estimates the value of *e* is 'e.c'.

Estimating the Value of Pi

It is common knowledge that pi, also written as 'π', is an infinite number; one that cannot be accurately expressed. Therefore, students are expected to estimate the value of pi down to the

15th decimal, using the mathematical formulas provided in the assignment formula. These formulas include:

- The Bailey Borwein Plouffe Formula (bbp.c):

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

- Euler's solution (euler.c):

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

- Viete's Formula (viete.c):

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

- Madhava Series (madhava.c):

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

Upon passing the appropriate arguments when running the executable, the program should print out the requested formula as well as the difference in accuracy between student programmed and library provided estimates.

Estimating the Value of Square Roots

Students are expected to calculate the value of square roots using the Newton-Raphael method. This method creates, "successively better estimations" of any given square root over the course of a few iterations. What this means is that each guess/estimation is more accurate than the previous. For this formula, the pseudocode was provided in python within the assignment pdf. Students were allowed to use this pseudocode as long as they were able to convert it into C for their own use, which allows for students to better understand not only what the provided pseudocode does, but also what the C equivalent of the provided python code is.
Testing the Newton-Raphson method requires that the program computes the square roots of all values between 0.0 and 10.0. Upon running the tests for the Newton–Raphson method, I was able to notice that the square roots that would result in irrational numbers somehow always took (at most) 7 iterations. Of course for square root for 1 the calculation only took one iteration.
The Newton-Raphson method is as follows (newton.c):

$$x_{k+1} = x_k - \frac{f(x_k)}{f^l(x_k)}$$

UNIX commands used

- ./mathlib-test : Used to run the main file. Expects a passed argument. Arguments are as follows:
  -a: Runs all tests
  -e: Runs e approximation test
  -b: Runs Bailey Borwein Plouffe pi approximation test
  -m: Runs Madhava pi approximation test
  -r: Runs Euler sequence pi approximation test
  -v: runs Viete pi approximation test
  -n: Runs Newton-Raphson square root approximation test
  -s: Prints statistics to see total computed terms/iterations for tested function(s).
  -h : Display a help message detailing program usage.
- make / make all : Used to create executable files as well as compile the necessary '.c' files into objects files (".o" files)
- make clean : Used to remove the executable main file as well as remove the associated object files
- make format : Used to properly format all '.c' files.

Findings

In terms of accuracy when estimating the value of pi I found that pi_bbp() was the most accurate in comparison to the other formulas with a difference of 0.000000000000000. Second in this was pi_viete() at 0.000000000000004, and third was pi_madhava(). However when ranking in terms of speed, pi_bbp() is the fastest at 11 iterations. pi_euler() came in last at 100,000 iterations only to be. In short, pi_bbp() was the most efficient and the most accurate out of all pi estimating formulas. Additionally, I found that the least efficient and least accurate of the pi estimating formulas was Euler's solution. One thing that surprised me was how accurate my pi_viete() was compared to the example executable provided in the resources repository. The reason for this is still a mystery of course, as I am unable to view the code for the example program.

What I learned

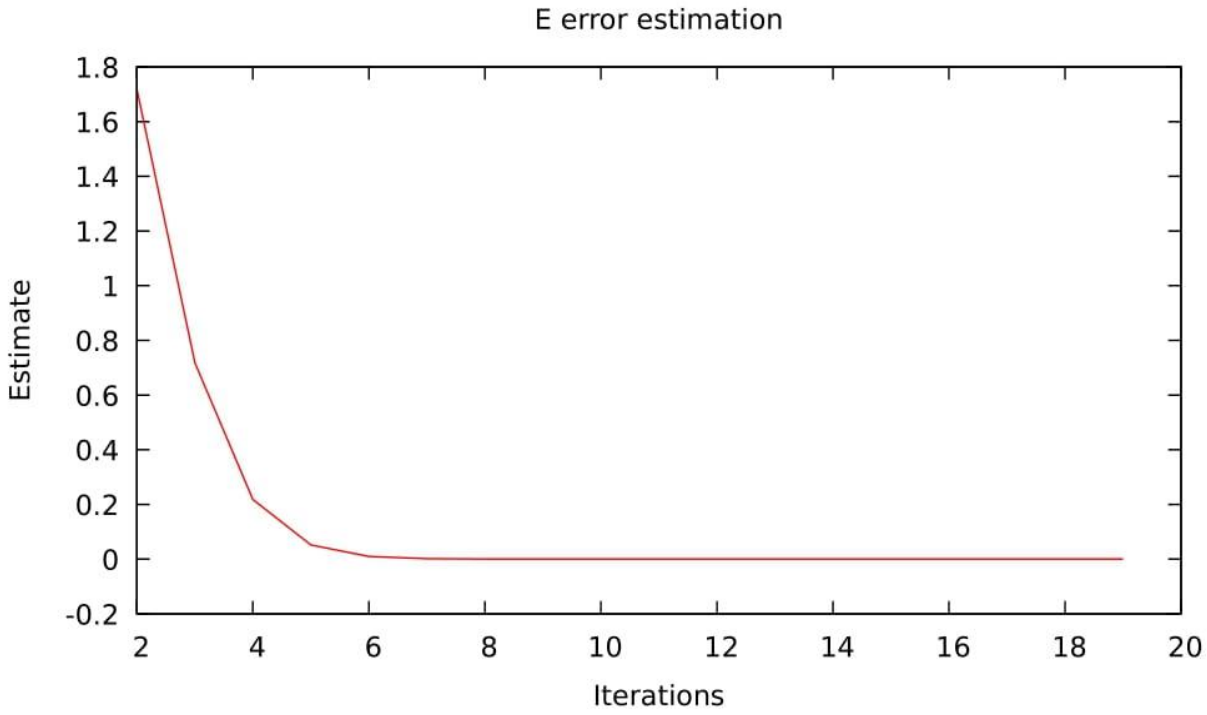
I was able to learn how to properly use conditions and loops in C, as well as how to program specific mathematical formulas. One problem I encountered towards the end of the project was how to create a graph using the printed outputs. My idea was to use a .sh file similar to assignment 1 which I got to make a graph, however the input data for the graph from the c files was only plotted to be the y axis points. I tried another possible solution where I created 2

separate c files for every math formula for an x and y input. The idea was to use 2 temporary .dat (data) files to plot with. I was unable to find a solution for this as well. (Note: I finished the main assignment on friday, thus had no chance to attend any more sections. I plan to attend the following week's sections to figure out how to do this.)

Data of Each File

e.c

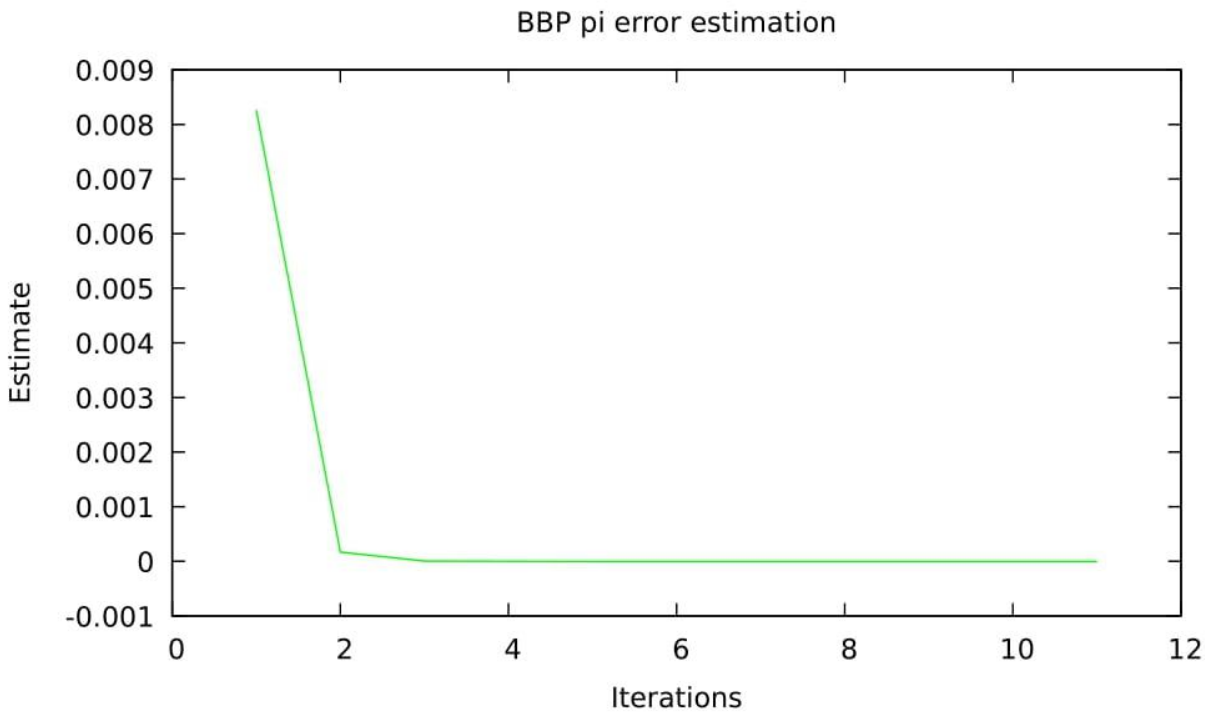
My program's output for the estimation of e with use of the formula was 2.718281828459046 over the course of 18 iterations.
The math library's estimation of e was 2.718281828459045
The difference between the two was .000000000000001



In this graph we are able to visualize the error estimation of e.c. We are able to notice that there is a significant jump in accuracy between the first and fourth iterations.

bbp.c

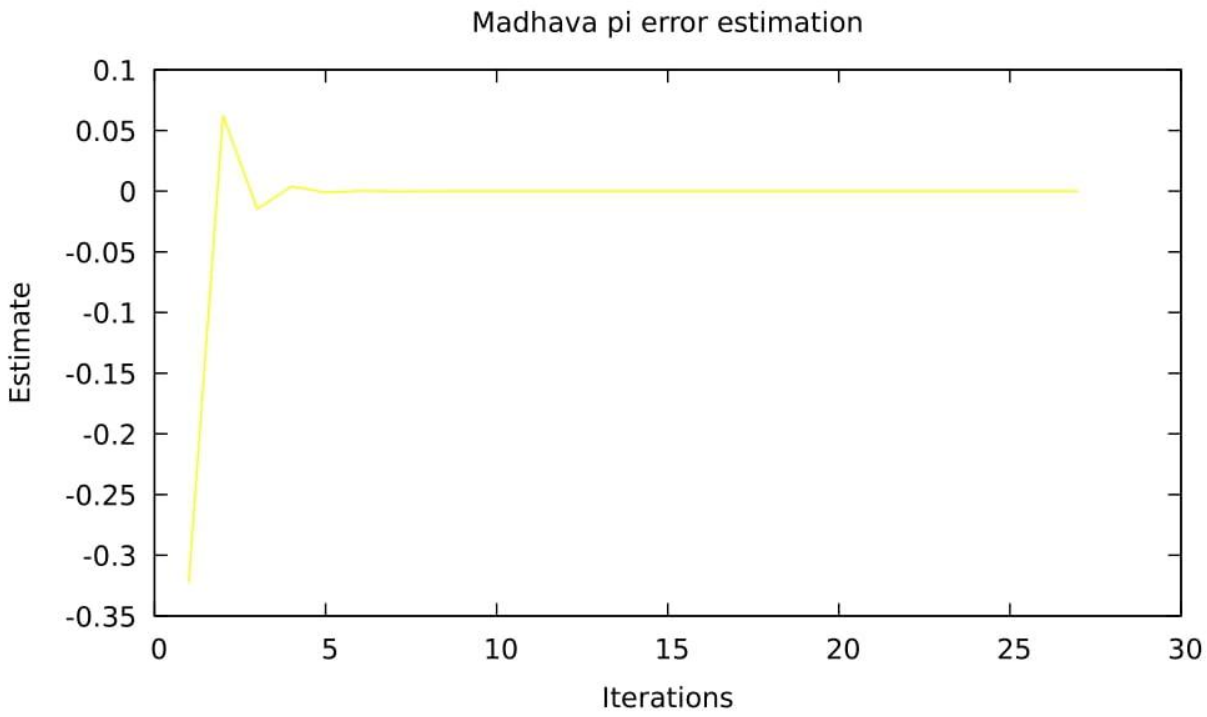
My program's output for the estimation of pi using the Bailey Borwein Plouffe formula was 3.141592653589793 over the course of 11 iterations
The math library's estimation of pi is 3.141592653589793
The difference between the two was 0.00000000000000



In this graph we are able to visualize the error estimation of the Bailey Borwein Plouffe pi approximation test. You will notice that the iterations begin at 1. This is because the formula's *'k'* variable also began at 1 so that the formula could be utilized. We see how the error between the actual value of pi and the estimated value reaches 0 immediately within the first iteration. I believe the reasoning for this is because of how the formula uses 16^{-k}, an exponential that rapidly helps in the accuracy of the estimation.

madhava.c

My program's output for the estimation of pi using madhava is 3.141592558095903 over the course of 27 iterations
The math library's estimation of e is 3.14159253589793
The difference between the two was 0.00000000000007



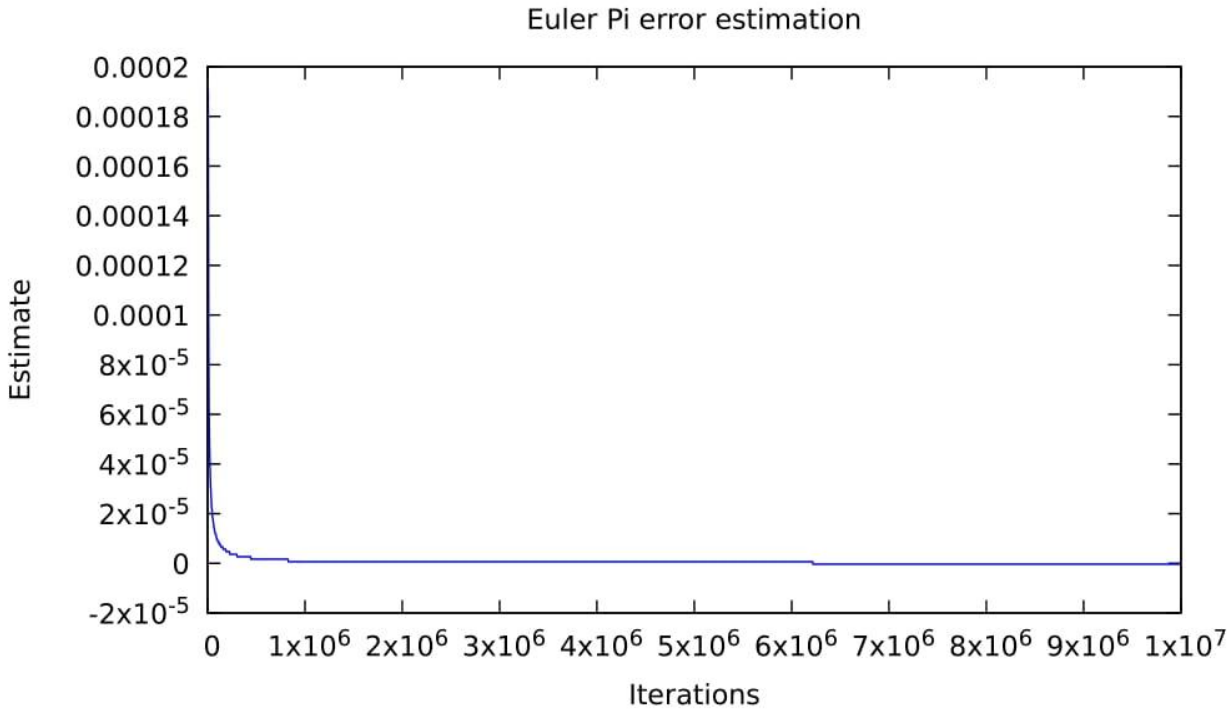
In this graph we are able to see the error estimate between the madhava series and the actual value of pi. Similar to bbp.c, we start our iterator at 1 for the function to properly work. Similarly, the error estimation becomes exponentially accurate before the fifth iteration even comes around.

euler.c

My program's output for the estimation of pi using euler's method is 3.141592558095903 over the course of 10,000,000 iterations

The math library's estimation of e is 3.14159253589793

The difference between the two is 0.00000095493891



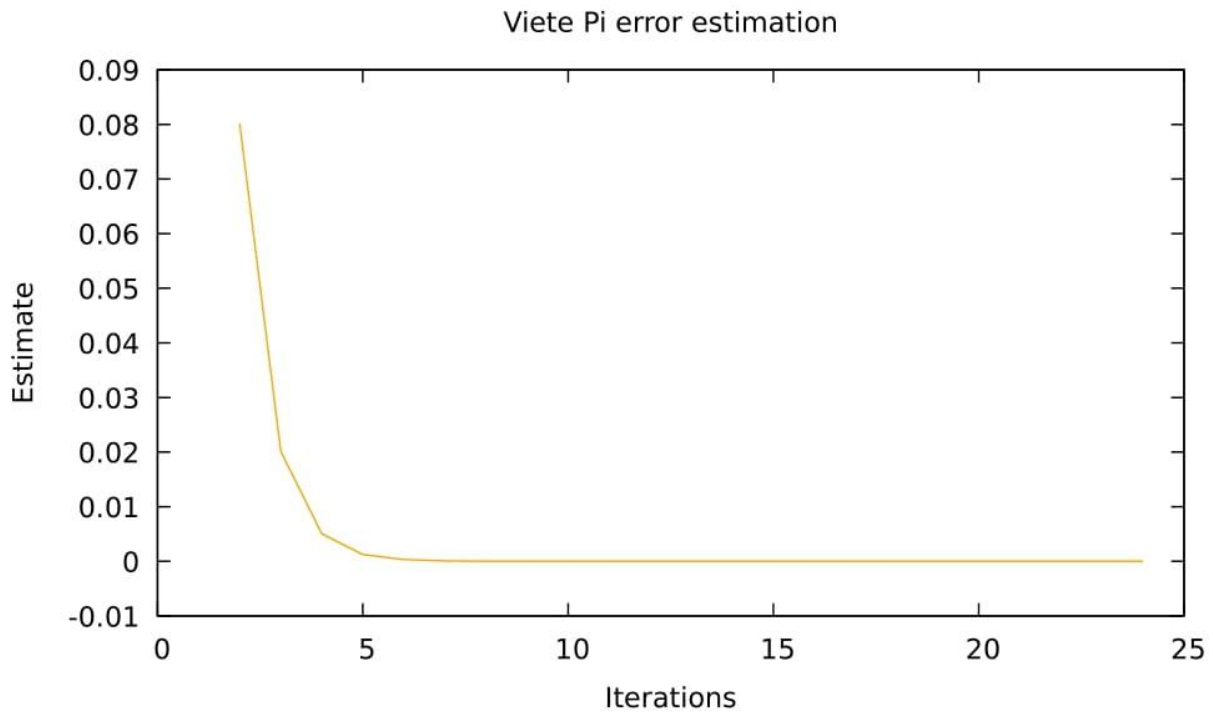
This graphs exhibits the error estimation of Euler's solution when estimating the value of pi. Because euler.c took well over 10000 iterations to complete (and still wasn't accurate), I shortened the function in such a way that there would be only one data output for every 5000 iterations. We are able to see that within the first million iterations the accuracy has significantly improved, yet there are still more improvements to be made. This is seen after $6 * 10^6$ iterations, where there is a slight increase in accuracy.

viete.c

My program's output for the estimation of pi using euler's method is 3.141592558095903 over the course of 23 iterations

The math library's estimation of e is 3.14159253589793

The difference between the two is 0.000000000000004



This graph shows the error estimation of Viete's formula in estimating the value of pi. Similar to bbp.c and madhava.c, we are able to see that the error in the estimation is significantly reduced within the first five iterations. As the iterations continue, the program is still able to make minor estimation corrections down to the 15th decimal place.

newton.c

My square root function was able to run and produce results that were exactly the same as the outputs of math.h when attempting to find the square roots of numbers between 0.0 and 10.0. Additionally, every square root estimated/calculated was able to do so (on average) within 7 iterations (each), which was generally the same as that of the example code provided to students as reference in the resources repo.

mathlib-test.c

Is the main file and is compiled into an executable. Its role as a main file is to 'weave' all .c files into a single program that can use all of the following arguments when properly executed from the terminal:

-a: Runs all tests
-e: Runs e approximation test
-b: Runs Bailey Borwein Plouffe pi approximation test
-m: Runs Madhava pi approximation test
-r: Runs Euler sequence pi approximation test
-v: runs Viete pi approximation test

-n: Runs Newton-Raphson square root approximation test
-s: Enable printing of statistics to see computed terms and iterations for each tested function.
-h : Display a help message detailing program usage.

Makefile
Used to compile all ‘.c’ files into ‘.o’ object files, or create an executable file, or remove all object files and executables.
