CSE 13S: Assignment 2 Write-Up

Johnny Li

January 29, 2023

1 Introduction

Throughout all the coding classes I've taken so far (which isn't that many to be fair), I've encountered numerous frustrating challenges. But one assignment in particular stands out. This assignment, CSE 13S: Assignment 2, marked a significant departure from the introductory nature of all my previous coding assignments, including Assignment 1. While the latter was a relatively straightforward introduction to the course and its expectations, Assignment 2 represented an unexpected significant jump in difficulty.

Prior to starting this project, I received ample warning from both my professors and classmates who had previously taken the class. They all advised me to start working on the assignments as early as possible, but I disregarded their well-intentioned advice. Unfortunately, this decision led to me using up the maximum amount of late days and ultimately earning a 45% grade reduction.

This assignment required me to build everything from scratch, unlike previous assignments which were significantly smaller and/or simpler and non-holistic. We had to code each individual file before connecting them together with a test harness while creating our own Makefile.

Looking back, I now realize the importance of starting my assignments early and following the advice of my professors and classmates. I hope to learn from my mistakes and apply this newfound lesson for future assignments.

2 Tasks

Through the deliverables below and some additional files, several mathematical functions from math.h will be manually implemented. Said functions will vital in calculating the natural constants of π and e. The constants will be derived with with infinite (technically not infinite – since computation is stopped when the difference between the previous and current iteration is less than EPSILON, or 1e-14) series such as Euler's Solution, The Bailey-Borwein-Plouffe (BBP) Formula, Viète's Formula, and Madhava's Formula. Although the series' may be infinite, computation will be halted after a certain point with $\epsilon = 10^{-14}$ which will be defined in the provided source file – mathlib.h.

2.1 e.c

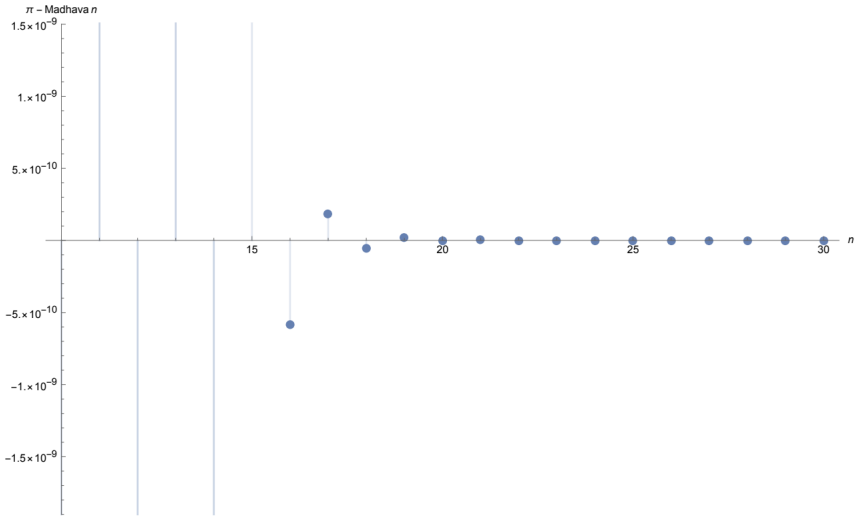
The file e.c should contain two separate functions, named e() and e_terms(). The first function, e(), approximates the value of Euler's number, a mathematical constant, through the use of the Taylor Series. This is achieved by using a loop that calculates each term in the Taylor series and adds them together. The number of terms computed is tracked through the use of a static, local variable.

The second function, e_terms(), serves a much simpler purpose. It is only responsible for returning the number of terms that have been computed by the e() function at that moment. This function does not perform any calculations or approximations, but instead retrieves the information that was recorded by the e() function. This function is useful in monitoring the progress of the approximation process and determining if more terms need to be computed in order to reach a desired level of accuracy. As the Taylor series is infinite, the computation process will be terminated once a predetermined number of terms have been calculated.

2.2 madhava.c

The file "madhava.c" comprises of two functions, "pi_madhava()" and "pi_madhava_terms()". The primary objective of the "pi_madhava()" function is to estimate the value of the mathematical constant π using the Madhava series, a method discovered by the Indian mathematician Madhava of Sangamagrama in the 14th century. This series is based on an infinite series expansion of the inverse tangent function and is represented by the formula: $\pi = 4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 - ...)$ where each term in the series is a fraction whose numerator is 1 and whose denominator is an odd number. The series begins with the term 1 and alternates between adding and subtracting the following terms as it progresses. As the number of terms increases, the sum of the series approaches π.

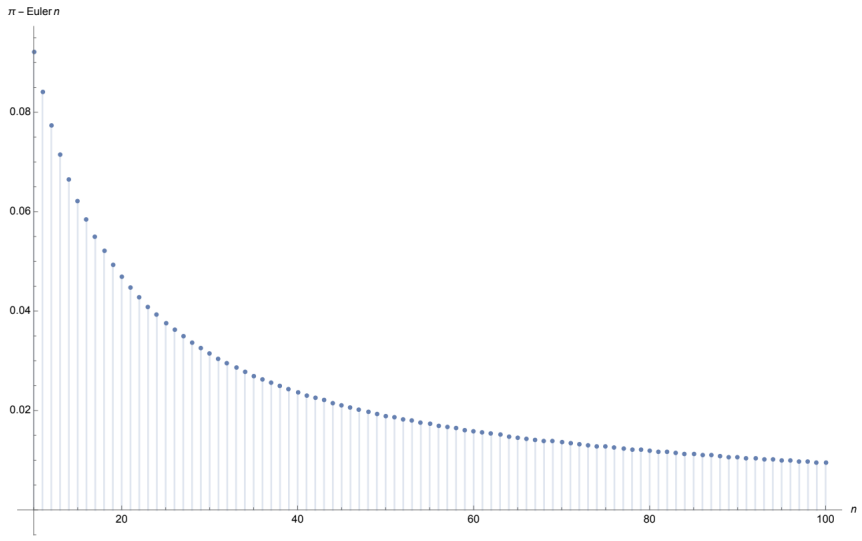
To keep track of the number of terms computed, the function makes use of a static, local variable. The "pi_madhava_terms()" function serves the simple purpose of returning the number of terms computed so far. This function provides an easy way to monitor the progress of the approximation process. This method is similar to the one used in the "e.c" file for approximating Euler's number using the Taylor series.



2.3 euler.c

The "pi_euler.c" file includes two functions, "pi_euler()" and "pi_euler_terms()". The primary function, "pi_euler()", is tasked with estimating the value of the mathematical constant π using the formula derived from Euler's solution to the Basel Problem. This formula was discovered by the renowned mathematician Leonhard Euler and provides a solution to the Basel Problem, which asks for the value of the sum of the reciprocals of the squares of the natural numbers.

To keep track of the number of terms computed, the "pi_euler()" function utilizes a variable. The second function, "pi_euler_terms()", is designed to retrieve the information about the number of computed terms. It serves the simple purpose of returning the number of terms computed thus far, providing a convenient way to monitor the progress of the approximation process. This way, the program can be easily adjusted to increase the number of terms computed, resulting in greater accuracy.



2.4 bbp.c

The "pi_bbp.c" file includes two distinct functions, "pi_bbp()" and "pi_bbp_terms()". The primary function, "pi_bbp()", is tasked with approximating the value of the mathematical constant π through the use of the Bailey-Borwein-Plouffe (BBP) formula. This formula, which was developed by Simon Plouffe, provides a fast and efficient way to approximate π using hexadecimal arithmetic. The function tracks the number of terms computed through the use of a static, local variable.

On the other hand, the secondary function, "pi_bbp_terms()", has a straightforward role. Its sole purpose is to return the number of terms that have been computed by the "pi_bbp()" function up to that point. This allows the program to easily keep track of the progress of the approximation and make any necessary modifications for greater accuracy.

2.5 viete.c

The function pi_viete() in the viete.c file aims to approximate the mathematical constant π using Viete's Formula. This formula, developed by the French mathematician François Viete, is a historical method for calculating the value of π. As the approximation process takes place, the function uses a static variable to keep track of the number of computed terms. The function pi_viete_factors() provides a means to monitor the progress of the approximation by simply returning the number of terms that have been computed.

2.6 newton.c

This method of approximation is based on the Newton-Raphson formula which is used to find the roots of a given equation. It is a fast and efficient method for finding the square root of a number and has been widely used for this purpose. The Newton-Raphson method works by making an initial guess at the root, and then iteratively refining this estimate until a satisfactory approximation is achieved. The sqrt_newton() function utilizes this method to provide an accurate approximation of the square root of the input number. The function makes use of a static variable to keep track of the number of iterations required for the approximation. The sqrt_newton_iters() function serves the purpose of providing the user with the number of iterations required for the approximation, allowing them to monitor the progress of the approximation and make necessary modifications.

2.7 mathlib-test.c

This file will contain the required main test harness for our implemented math library. It gets all the individual files by importing mathlib.c where everything is connected and prints their estimations and the number of iterations reached until computation was stopped by the constant EPSILON, which is

also in the mathlib.c file. As in accordance with the sample mathlib-test-arm64 (or amd64) file, the test harness prints the help message when nothing, or an invalid command is entered.

The following command-line options are be supported:

- -a : Runs all tests.
- -e : Runs e approximation test.
- -b : Runs Bailey-Borwein-Plouffe π approximation test.
- -m : Runs Madhava π approximation test.
- -r : Runs Euler sequence π approximation test.
- -v : Runs Viète π approximation test.
- -n : Runs Newton-Raphson square root approximation tests.
- -s : Enable printing of statistics to see computed terms and factors for each tested function
- -h : Display a help message detailing program usage.

3 Sources

- Stack Overflow
- Office Hours
- The C Programming Language
- Learn Enough Command Line to Be Dangerous