CSE 13S: Assignment 2 Design Doc

Johnny Li

January 29, 2023

1 Introduction

For this assignment, we are tasked with the task of manually implementing a select number of mathematical functions that are typically found within the C library, math.h. These functions will be utilized to calculate the fundamental constants of e and π. In addition to this, we are also required to create a separate and dedicated test program, known as mathlib-test, which will compare our implemented functions to those found within the math library. This will allow us to analyze and evaluate the performance of our functions. Afterwards, our finding and report will be presented in a formal write-up.

It is important to note that the interface for our math library, mathlib.h, will be provided to us and must not be edited. Since the purpose of this assignment is to develop our understanding and proficiency in implementing mathematical functions, as such, we are not allowed to use any functions from the math.h library during this process. Additionally, we are also prohibited from writing a factorial() function. The challenge of this assignment lies in our ability to thoroughly understand how these functions are made before manually creating and implementing said mathematical functions.

2 Tasks

Through the deliverables below and some additional files, several mathematical functions from math.h will be manually created from scratch. Said functions will vital in calculating some natural constants. The constants will be derived with with infinite (or not so infinite) series in addition to Euler's Solution, The Bailey-Borwein-Plouffe Formula, Viète's Formula, and Fastest Series. Although the series' may be infinite, computation will be halted after a certain point with $\epsilon = 10^{-14}$ which will be defined in mathlib.h.

2.1 e.c

The deliverable file e.c should include two distince and separate functions, e() and e_terms(). The former approximates the value of Euler's number, a mathematical constant, by using the Taylor Series. A loop will be used to compute each term of the Taylor series and add them together while a static, local variable stores the amount of terms computed.

Meanwhile, the second function, "e_terms()", has a much simpler role. Its purpose is solely to return the number of terms that have been computed by the "e()" function up until that point. It does not perform any calculations or approximations, but simply retrieves the information that was tracked by the "e()" function. This function is useful for checking the progress of the approximation process and determining whether more terms need to be computed to achieve a sufficient level of accuracy. Since the Taylor series is infinite, computation will stop after a hitting a certain mark as described above.

2.2 madhava.c

The "madhava.c" file, similar to the "e.c" file, is made up of two functions: "pi_madhava()" and "pi_madhava_terms()". The primary purpose of the "pi_madhava()" function is to approximate the value of the mathematical constant π using the Madhava series, which is a method discovered by the Indian mathematician Madhava of Sangamagrama in the 14th century. It is based on an infinite series

expansion of the inverse tangent function, 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 starts with the term 1, and then alternates between adding and subtracting the following terms. The sum of the series approaches pi as the number of terms increases.

To keep track of the number of terms computed, the function makes use of a static variable which is local to the file. The second function, "pi_madhava_terms()", serves the purpose of simply returning the number of computed terms, providing an easy way to keep track of the progress of the approximation. This method is similar to the one used in the file "e.c" for approximating euler's number using the taylor series.

2.3 euler.c

The "pi_euler.c" file comprises of two functions: "pi_euler()" and "pi_euler_terms()". The primary function, "pi_euler()", is responsible for approximating the value of the mathematical constant π by utilizing the formula derived from Euler's solution to the Basel Problem. This formula, which was derived by the famous mathematician Leonhard Euler, provides a solution to the Basel problem, which is a problem in mathematics that asks for the value of the sum of the reciprocals of the squares of the natural numbers.

In addition to approximating the value of pi, the "pi_euler()" function also keeps track of the number of computed terms through the use of a variable. The second function, "pi_euler_terms()", serves the simple purpose of returning the number of computed terms, thus providing an easy way to keep track of the progress of the approximation. This way, the program can be easily modified to increase the number of terms computed for greater accuracy.

2.4 bbp.c

The pi_bbp.c file comprises two functions: pi_bbp() and pi_bbp_terms(). The primary function, pi_bbp(), is intended to approximate the value of the mathematical constant π using the Bailey-Borwein-Plouffe (BBP) formula. Additionally, it tracks the number of computed terms using a static variable. The secondary function, pi_bbp_terms(), again, just returns the number of computed terms.

2.5 viete.c

The viete.c file comprises two functions: pi_viete() and pi_viete_factors(). The primary function, pi_viete(), is intended to approximate the value of the mathematical constant π using Viete's Formula. Additionally, it tracks the number of computed terms using a static variable. The secondary function, pi_viete_factors(), returns the number of computed terms.

2.6 newton.c

In this file, we have provided two distinct functions - sqrt_newton() and sqrt_newton_iters() - which work in tandem. The sqrt_newton() function is designed to approximate the square root of a given input by utilizing the Newton-Raphson method while keeping track of the number of iterations required for this approximation. The second function, sqrt_newton_iters(), simply returns the number of iterations.

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.