Assignment 2 – Slice of Pi

Mateo Garcia

CSE 13S – Spring 2023

Purpose

The purpose of this program is to implement a number of mathematical functions in order to compute e and π. The program will also compare them to the C language's math library to ensure they are completely functional.

How to Use the Program

By running mathlib-test.c, you can add 10 different options to the command to tell it which mathematical function you want to run. These are the several different flags you can run:

- -a : Runs all tests.
- -e : Runs e approximation test.
- -b : Runs Bailey-Borwein-Plouffe approximation test.
- -m : Runs Madhava approximation test.
- -r : Runs Euler sequence approximation test.
- -v : Runs Vi'ete approximation test.
- -w : Runs Wallis approximation test.
- -n : Runs Newton-Raphson square root approximation tests, calling sqrt_newton() with various inputs for testing. This option does not require any parameters, and will only test within the range of [0, 10) (it will not test the value 10).
- -s : Enable printing of statistics to see computed terms and factors for all tested functions.
- -h : Display a help message detailing program usage. This can be anything you want, but must be specific and readable.

Program Design

There are 8 different C files: **e.c**, **madhava.c**, **euler.c**, **bbp.c**, **viete.c**, **wallis.c**, **newton.c**, and **mathlib-test.c**. The first 6 calculate an approximation of either π or e, and **newton.c** approximates square roots, and the **mathlib-test.c** will test all of these files.

Data Structures

A double will be used to store Epsilon. It is used in while loops to know when to end the loop as the approximation gets closer and closer to convergence. Other data structures that will be used are either ints or doubles to store variables in the functions, as well as iterations in the while loops.

Algorithms

Most of the algorithms will be using a while loop to converge on the answer and will break when the difference between the last value and current value is less than a pre-determined value (epsilon). Inside the while loop, a mathematical operation will be performed according to which one is chosen in the command line.

```
int approximate_function(x){
    next_val = 1.0
    val = 0.0
    while abs(next_value - val) > epsilon{
        val = next_val
        next_val = // math operation to get next value in function
    }
    return val
}
```

Function Descriptions

There are several files part of this program and each will have more than 1.

- The **e.c** file will have 2 functions, **e()** and **e_terms()**. The former function will approximate the value of e using Taylor series and track the number of computed terms with a static variable, and the other will return the number of computed terms.
- The **madhava.c** file will contain **pi_madhava()** and **pi_madhava_terms**, the former function will approximate the value of π using the Madhava series, and track the number of computed terms with a static variable. The latter will return the number of computed terms.
- The **euler.c** file will contain **pi_euler()** and **pi_euler_terms()**. The former function will approximate the value of π using the formula derived from Euler's solution to the Basel problem, and track the number of computed terms using a static variable. The latter will return the number of computed terms.
- The **bbp.c** file will contain **pi_bbp()** and **pi_bbp_terms()**. The former function will approximate the value of π using the Bailey-Borwein-Plouffe formula, and track the number of computed terms using a static variable. The latter will return the number of computed terms.
- The **viete.c** file will contain **pi_viete()** and **pi_viete_factors()**. The former function will approximate the value of π using Viete's formula, and track the number of computed terms using a static variable. The latter will return the number of computed terms.
- The **wallis.c** file will contain **pi_wallis()** and **pi_wallis_factors()**. The former will approximate the value of π using Wallis's formula, and track the number of computed terms using a static variable. The latter will return the number of computed terms.
- The **newton.c** file will contain **sqrt_newton()** and **sqrt_newton_iters()**. The former function will take a parameter that will be greater than zero, and will be expected to produce an approximation of the square root using the Newton-Raphson method. This parameter can be outside the range of values that are tested by **mathlib-test.c**. It will also track the number of iterations using a static variable. The latter will return the number of computed terms.
- **mathlib-test.c** will contain the main test for this implemented math library. The expected output for each of the e or π tests should resemble the following.

```
$ ./mathlib-test -e -b -v
e() = 2.718281828459046, M_E = 2.718281828459045, diff = 0.000000000000000
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
pi_viete() = 3.141592653589775, M_PI = 3.141592653589793, diff = 0.000000000000018
```

Results

Have not started yet, this a draft.

Audience: Write this section for the graders. If you completed only part of the assignment, explain that here.

To write this section, use your code according to its intended purpose. Does it successfully achieve everything it should? Is anything lacking? Could anything be improved? Talk about all of that here, and use your code's output to prove it.

Numeric results

You can include screenshots of program output, as I have in Fig. 1.

Figure 1: Screenshot of the program running.

Error Handling

When it comes to handling errors, the Assignment specifications are relatively lax. In this section, describe the errors While the assignment only requires that you be close to epsilon in a certain range,