Assignment 2 – Slice of Pi

Noah Meisel

CSE 13S – Spring 2023

Purpose

The purpose of this program is to provide various approximations of e, and pi, two very important mathematical constants for computation. This was done by testing out a variety of different methods for approximating PI and implementing them as independent functions in a larger program. Each of these methods approximations of their respective constants can be compared to the math library's approximations to see how each method stacks up in relation to each other.

How to Use the Program

The overarching program is split up across eight different C files. Mathlib-test.c is the central test harness for the other seven functions and is used to test out all of the different methods for approximating e and pi. The desired methods can be selected by using a variety of different command line arguments which is specified in Asgn2 Section 7.1[1].Aside from these command line arguments, there are no other inputs for the function.

The format for using a command line argument for this program is as follows(done after you have compiled the program using the provided makefile).

```
./mathlib-test -"charecter for command"
```

The output for each functions test is in the following format

```
Function_Name = Approximation, MathLibrary Value = Approximation, diff = difference.
```

Program Design

The design of the program is relatively simple. Each method for calculating a constant has it's own C file. All of the C files are linked together and used in the main mathlib-test.c.

Mathlib-test.c uses the getopt() function to take command line options. The options are used in a switch statement combined with an array of 0's with length = to the number of potential arguments(10 in this case) to track which options have been specifed. This array is then used in subsequent structure to print the output using an if-else tree, because this way it is possible to control for multiple of the same arguments as well as implement a desired order for each term in the output.

Data Structures

All of the functions for calculating the constants accept no arguments and return a double. The square root function also returns a double but accepts a double as the argument. The counter functions, which go hand-in-hand with the actual constant calculating functions, all return an int and accept no arguments. The counter is tracked with a static int in each functions respective C file. The counter is incremented by 1 each time the functions are called. In mathlib-test.c, a variable called statistics is used to track whether the -s command line option has been specified.

Algorithms

The Pseudo code for each approximation function is defined below

```
square root algorithim
while |next_value - current_value | > EPSILON
    set current_value to next_value
    set next_value to 0.5(current_value + number to be square rooted / current value)
return next_value to the caller
```

```
e algorithim
k starts at 0
do the following
    set previous factorial to current_factorial
    set current_factorial to k, the sum counter.
    set current_factorial to factorial x previous factorial
    add 1/current_factorial to the sum
    incremement k
while 1/current_factorial is greater than EPSILON, repeat this loop
return sum to caller
```

```
Bailey-Borwein-Plouffe Formula
k starts at 0
do the following
    while i is less than k
        power = power x 16
    compute the Bailey-Borwein-Plouffe Formula expression using power for 16^k and k for k.
    add this to the sum
    increment k
while the absolute value of the expression is greater than EPSILON, repeat this loop
return sum to caller
```

```
Madhava Series
k starts at 0
do the following
    while i is less than k
        power = power x (1/-3)
    compute the Madhava Series using k for k and power for the top part of the expression
    add this to the sum
    increment k
while the absolute value of the expression is greater than EPSILON, repeat this loop
return sum to caller
```

```
Euler's Solution
k starts at 1
do the following
    add 1/k^2 to the sum
    while the absolute value of the expression is greater than EPSILON, repeat this loop
    increment k
while the absolute value of the expression is greater than EPSILON, repeat this loop
return sum to caller
```

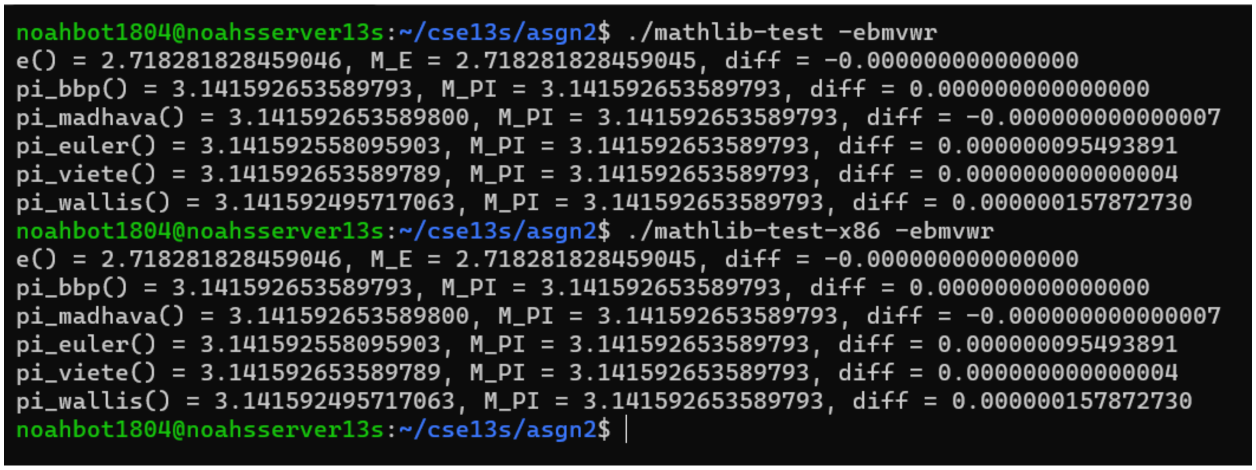
```
Viete's Formula
k starts at 1
do the following
    term = Recursive Function with input k/2.0
    product = poduct x term
    increment k
while the absolute value of term is greater than EPSILON, repeat this loop
product = 2/product
return product to caller

Recursive Function
if k == 1
    return square_root(2)
else
    return square_root(2 + Recursive Function(k-1)
```

```
The Wallis Series
k starts at 1
do the following
    calculate the current Wallis Series value
    product = poduct x current_value
    increment k
while the absolute value of term is greater than EPSILON, repeat this loop
product = product x 2.0
return product to caller
```

Function Descriptions

- The square root function and the recursive function take a double as an input.Mathlib-test.c utilizes the function data_printer, which takes a double, an int, another int, and a string as arguments. The function is intended to be used with one of the math function, the math term functions, the statistics variable, and the name of the function as a string. The rest of the functions in this program have void for input.
- The square root function outputs the square root of the number it was given as an input. The recursive function outputs sqrt(2 + recursive call(k-1). The data_printer function has no return value but prints a formatted string. The main expression functions output an approximation for either e or pi depending on the function. The number of term functions essentially output the final value of k in each series.
- The functions in each C file all have a similar purpose. The pi/e name() functions all replicate a particular mathematical sequence. The pi/e name terms/factors() count the number of terms/factors needed to calculate the approximation for each method. The only excpetions to this are the square root function, though this also has a terms function to go with it, and the recursive function implemented in Viete.c. The recursive function essentially acts as a fixed value for the top part of the series in the main functions loop. data_printer is designed to reduce code overlap and cut down the size of mathlib-test.c by creating a standard method for printing out each functions values and difference compared to the math libaraies values for pi.
- The pseudocode for each mathematical function is provided above, in the algorithims section.
- The only series worth mentioning/explaining here is the Viete series. This function was implemented by making a secondary recursive function because the formula itself is naturally recursive, so implementing it this way was the most painless route that I could see.



```
noahbot1804@noahsserver13s:~/cse13s/asgn2$ ./mathlib-test -ebmvwr
e() = 2.718281828459046, M_E = 2.718281828459045, diff = -0.000000000000000
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
pi_madhava() = 3.141592653589800, M_PI = 3.141592653589793, diff = -0.000000000000007
pi_euler() = 3.141592558095903, M_PI = 3.141592653589793, diff = 0.000000095493891
pi_viete() = 3.141592653589789, M_PI = 3.141592653589793, diff = 0.000000000000004
pi_wallis() = 3.141592495717063, M_PI = 3.141592653589793, diff = 0.000000157872730
noahbot1804@noahsserver13s:~/cse13s/asgn2$ ./mathlib-test-x86 -ebmvwr
e() = 2.718281828459046, M_E = 2.718281828459045, diff = -0.000000000000000
pi_bbp() = 3.141592653589793, M_PI = 3.141592653589793, diff = 0.000000000000000
pi_madhava() = 3.141592653589800, M_PI = 3.141592653589793, diff = -0.000000000000007
pi_euler() = 3.141592558095903, M_PI = 3.141592653589793, diff = 0.000000095493891
pi_viete() = 3.141592653589789, M_PI = 3.141592653589793, diff = 0.000000000000004
pi_wallis() = 3.141592495717063, M_PI = 3.141592653589793, diff = 0.000000157872730
noahbot1804@noahsserver13s:~/cse13s/asgn2$ |
```

Figure 1: Screenshot of output for all functions except sqrt_newton, as it takes up too much space to effectively show in an image.

Results

Overall everything specified by the assignment is accomplished here. I was able to get functions using each method that seem to find values exactly matching the ones from the binary file(as seen in Figure 1), which in my mind means it is a success.

The biggest challenge with this assignment was the mathlib-test.c file. It was difficult to find a simple method to ensure each command-line option is only read once. The solution to this issue for me was to create an array to track whether each option had been specified, and use the switch statement to merely change values in the array which I use below the switch statement in a series of if statements. By doing it this way, I was able to essentially read through the entire input string and determine exactly what commands were entered by the user before printing any output.

Numeric results

Error Handling

The values obtained by the function's implemented in my program match the ones values from the binary file. This leads me to believe that the reason for some of the inaccuracy in the approximations, specifically in the case of pi_wallis() and pi_euler(), is due to the formula behind the function itself. Euler's solution specifically is quite inaccurate when compared to the rest of the functions.

This error could also be due to the the value of EPSILON. With a smaller EPSILON, the accuracy of the functions would increase no doubt, but it is unlikely to impact the approximated value by that much because the numbers we would be adding/multiplying into the total value would be infinitesimally small.

References

[1] Darrell Long Kerry Veenstra. *A Little Slice of PI.* UCSC, Santa Cruz, CA, Spring, 2023.