Assignment 2
A Little Slice of Pi

By Jiancheng (Jason) Xiong
CSE 13S Spring 2023

Due May 3rd 2023, 11:59 PM

1 Purpose

The purpose of this assignment is to create a program that calculates the value of pi and e using a variety of arithmetic sequences run by a main function.

2 Using the Program

In order to use the program, compile and run the file mathlib-test.c. In order to run the program, run ./mathlib-test and specify the commands in which you want to run. The available options are;

-a; running all tests
-e; runs e approximation test
-b; runs the Bailey Borwein Plouffe pi approximation test
-m; runs the Madhava pi approximation test
-r; runs the Euler pi approximation test
-v; runs the Viete pi approximation test
-w; runs the Wallis pi approximation test
-n; runs the Newton-Raphson square root approximation test
-s; enables printing of statistics to see computed terms and factors for all tested functions
-h; displays a help message detailing program usage.

3 Program Design

The main program will be in the file matlib-test.c, while each of the separate individual sequence tests will be in their respective files. Each separate sequence test will simply be a function which the main mathlib file runs.

4 Data Structures

The main data structures will be the usage of for loops to run the sequences an undefined number of times until the margin of error is lower than an epsilon we will define to be 10^-14. The sequence functions will take no input and output an integer that is within the epsilon margin of error from the actual value.

5 Algorithms

Main Routine:

Using getopt, iterate through the console command and retrieve all the requested commands

Instead of directly running the sequential function when the command is detected, instead use a temporary variable so that we can call them in the expected order.

Run each requested sequence until the margin of error is below 10^-14

Pseudocode for the sequences:

Initialize a variable k to keep track of iterations and a sum variable for total output

Do:

Plug in k into the algorithm and add it to the running total

Increment k by 1

while:

K plugged into the equation is greater than epsilon, otherwise break

If asked (-s), print the output number and the number of iterations they took.

6 Function Descriptions

Main function: takes its arguments from the command line, using getopt to know what sequential functions to run. Takes one (command line) input and outputs the value of the sequences requested.

For all sequence functions: takes no input and returns an output that is within the epsilon margin of error from the expected value. Prints nothing unless the command -s is called, then the value and number of iterations taken is printed.

7 Results

```
sqrt_newton(8.90) = 2.983286394428406, sqrt(8.90) = 2.983286394428406, diff = 0.
000000000000000
sqrt_newton(9.00) = 2.999999682108545, sqrt(9.00) = 2.999999682108544, diff = 0.
000000000000000
sqrt_newton(9.10) = 3.016620372887695, sqrt(9.10) = 3.016620372887695, diff = 0.
000000000000000
sqrt_newton(9.20) = 3.033149988970348, sqrt(9.20) = 3.033149988970348, diff = 0.
000000000000000
sqrt_newton(9.30) = 3.049590011306528, sqrt(9.30) = 3.049590011306528, diff = 0.
000000000000000
sqrt_newton(9.40) = 3.065941881140325, sqrt(9.40) = 3.065941881140325, diff = 0.
000000000000000
sqrt_newton(9.50) = 3.082207001484488, sqrt(9.50) = 3.082207001484488, diff = 0.
000000000000000
sqrt_newton(9.60) = 3.098386738525345, sqrt(9.60) = 3.098386738525345, diff = 0.
000000000000000
sqrt_newton(9.70) = 3.114482422962032, sqrt(9.70) = 3.114482422962033, diff = 0.
000000000000000
sqrt_newton(9.80) = 3.130495351283752, sqrt(9.80) = 3.130495351283752, diff = 0.
000000000000000
sqrt_newton(9.90) = 3.146426786988521, sqrt(9.90) = 3.146426786988521, diff = 0.
000000000000000
```

8 References

"C - Switch Statement." *Tutorials Point*, https://www.tutorialspoint.com/cprogramming/switch_statement_in_c.htm.

https://pubs.opengroup.org/onlinepubs/009695399/basedefs/math.h.html.