CSE 13S Fall 2021
James Gu
jjgu@ucsc.edu
7 October 2021

Assignment 2: A Little Slice of Pi
DESIGN.pdf

Description of Program:

Computers can only do the simple operations of multiplication, division, addition, and subtraction so functions like integrals need to be computed using its Taylor series expansion. Not only integrals can be represented through these summations but most functions can be written as an infinite sum. In this program the goal is to approximate the value of e using the Taylor series, the value of pi using the Madhava series, Euler's solution to the Basel problem, Bailey-Borwein-Plouffe formula, and Viete's formula, and the square root of an argument using the Newton-Raphson method.

Files:

bbp.c
This file should contain two functions: 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, as described in §4.4. It should also track the number of computed terms. The latter function will simply return the number of computed terms.

e.c
This file should contain two functions: e() and e_terms(). The former function will approximate the value of e using the Taylor series presented in §3 and track the number of computed terms by means of a static variable local to the file. The latter function will simply return the number of computed terms.

euler.c
This file should contain two functions: 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, as described in §4.4. It should also track the number of computed terms. The latter function will simply return the number of computed terms.

madhava.c
This file should contain two functions: pi_madhava() and pi_madhava_terms(). The former function will approximate the value of π using the Madhava series presented in §4.2 and track the number of computed terms with a static variable, exactly like in e.c. The latter function will simply return the number of computed terms.

mathlib-test.c
This file will contain the main test harness for your implemented math library. It should support the following command-line options:
-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.

mathlib.h
This contains the interface for your math library.

newton.c
This contains the implementation of the square root approximation using Newton's method and the function to return the number of computed iterations.

viete.c
This file should contain two functions: pi_viete() and pi_viete_factors(). The former function will approximate the value of π using Viète's formula as presented in §4.6 and track the number of computed factors. The latter function will simply return the number of computed factors.

MakeFile
CC = clang must be specified.
CFLAGS = -Wall -Wextra -Werror -Wpedantic must be specified.
make must build the mathlib-test executable, as should make all and make mathlib-test.
make clean must remove all files that are compiler generated.
make format should format all your source code, including the header files.

README.md
This must use proper Markdown syntax. It must describe how to use your program and Makefile. It should also list and explain any command-line options that your program accepts. Any false positives reported by scan-build should be documented and explained here as well. Note down any known bugs or errors in this file as well for the graders.

DESIGN.pdf
This document must be a proper PDF. This design document must describe your design and design process for your program with enough detail such that a sufficiently knowledgeable programmer would be able to replicate your implementation. This does not mean copying your entire program in verbatim. You should instead describe how your program works with supporting pseudocode.

WRITEUP.pdf
This document must be a proper PDF. This writeup must include, at least, the following:

-Graphs displaying the difference between the values reported by your implemented functions and that of the math library's. Use a UNIX tool — not some website — to produce these graphs. gnuplot is recommended. Attend section for examples of using gnuplot and other UNIX tools. An example script for using gnuplot to help plot your graphs will be supplied in the resources repository.
-Analysis and explanations for any discrepancies and findings that you glean from your testing.

Pseudocode:

```
Set EPSILON to 10^-14
e.c:
Set count to 1.0
e():
	Set e to 0
	Set current_Factorial to 1.0
	While current_Term is greater than the EPSILON:
		Set the current_Factorial to count*current_Factorial
		Set current_Term to 1.0/current_Factorial
		Add current_Term to e
		Add 1.0 to count
	Return e
e_terms():
	Return count

madhava.c:
Set count to 0
pi_madhava():
	Set pi to 0
	While current_Term is greater than EPSILON:
		Set numerator to 1.0
		For count times:
			Multiply numerator by -3.0
		Set denominator to 2 * count + 1
		Set current_Term to numerator/denominator
		Add current_Term to pi
		Add 1.0 to count
	Return pi
pi_madhava_terms():
	Return count + 1

euler.c:
Set count = 1
```

```
pi_euler():
        Set pi = 0
        While current_Term is greater than EPSILON:
                Set current_Term = 1.0/(count*count)
                Add current_Term to pi
                Add 1 to count
        Multiply pi by 6
        Square root pi using sqrt_newton()  //see below
        Return pi
pi_euler_terms():
        Return count

bbp.c:
Set count to 0
pi_bbp():
        Set pi to 0
        While current_Term is greater than EPSILON:
                Set the multiplier to 1.0
                For count times:
                        Divide multiplier by 16
                Set fraction to (count*(120*count + 151) + 47)/(count*(count*(count(512*k
                + 1024) + 712) + 194) + 15)
                current_Term = multiplier * fraction
                Add current_Term to pi
                Add 1 to count
        Return pi
pi_bbp_terms():
        Return count + 1

viete.c:
Set count to 1
pi_viete():
        Set pi to 1
        Set numerator to 2
        Set current_Iter to 0
        While current_Term is greater than EPSILON:
                Set current_Iter to sqrt_newton(2 + current_Iter)/2
                Set current_Term to numerator/current_Iter
                Multiply pi by current_Term
                Add 1 to count
```

```
        Return pi
pi_viete_terms():
        Return count

newton.c:
Set count to 1
sqrt_newton(y):
        Set answer to 0
        Set guess to 1
        While abs(guess - answer) > EPSILON:
                Set answer to guess - (guess*guess - y)/(2*guess)
                Add 1 to count
        Return answer
sqrt_newton_iters():
        Return count

mathlib-test.c:
Set test_e, test_bbp, test_madhava, test_euler, test_viete, and test_newton to false
While the user wants to test functions using the command line:
        If case is ‘a’:
                Set test_e, test_bbp, test_madhava, test_euler, test_viete, and
                test_newton to true
        If case is ‘e’:
                test_e = true
        If case is ‘b’:
                test_bbp = true
        If case is ‘m’:
                test_madhava = true
        If case is ‘r’:
                test_euler = true
        If case is ‘v’:
                test_viete = true
        If case is ‘n’:
                test_newton = true
If test_e is true:
        Run e.c
If test_bbp is true:
        Run bbp.c
If test_madhava is true:
        Run madhava.c
```

```
If test_euler is true:
        Run euler.c
If test_viete is true:
        Run viete.c
If test_newton is true:
        Run newton.c
```