Deeraj Gurram
dgurram@ucsc.edu
10 October 2021

CSE 13S Fall 2021
Assignment 2 - A Little Slice of π
Design Document

Description of the Programs:

This assignment implements various mathematical functions to compute the fundamental constants e and π. The file e.c finds e using a Taylor series computation, madhava.c finds π by using the Madhava series, and euler.c finds π using Euler's solution to the Basel problem. Bbp.c finds π using the Bailey-Borwein-Plouffe formula, viete.c approximates the value of π using Viète's formula, and newton.c computes the square root of an argument passed to it using the Newton-Raphson method. All of these .c files also track and return the number of iterations they undergo/ amount of computed terms. This assignment also includes mathlib-test.c which is a self-created test harness that I use to test each program, along with the Makefile which formats and compiles each .c file in the directory.

Deliverables:

1. e.c
    - This file contains the functions e() and e_terms(). The former uses a Taylor series computation that finds e, while the latter returns the number of computed terms.
2. madhava.c
    - This file contains the functions pi_madhava() and pi_madhava_terms(). The former uses the Madhava series to approximate the value of π, while the latter returns the number of computed terms.
3. euler.c
    - This file contains the function pi_euler() and pi_euler_terms(). The former approximates the value of π using Euler's solution to the Basel problem, while the latter returns the number of computed terms.
4. bbp.c
    - This file contains the functions pi_bbp() and pi_bbp_terms(). The former approximates the value of π using the Bailey-Borwein-Plouffe formula, while the latter returns the number computed terms.
5. viete.c
    - This file contains the functions pi_viete() and pi_viete_terms(). The former approximates the value of π using the Viète's formula, while the latter returns the number of computed factors.
6. newton.c
    - This file contains the functions sqrt_newton() sqrt_newton_iters(). The former approximates the value of the square root of the argument passed to it using the Newton-Raphson method, while the latter returns the number of iterations taken.

7. mathlib.h
    - This file is given by the professor and hasn't been modified by me. It contains the definition of epsilon as well as the function names from all the programs.
8. mathlib-test.c
    - This file contains the main test harness for my implemented math library. It supports the following 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.
9. Makefile
    - This file formats, cleans, and compiles all .c files in the directory
10. README.md
    - This file contains a description of the assignment and instructions on how to build and run the code. It also includes minor errors found in the code.
11. WRITEUP.pdf
    - This file analyzes the differences in the output of my programs and compares it to the output from <math.h>. It also provides reasoning for differences in outputs, and includes graphs to support my arguments.
12. DESIGN.pdf
    - This pdf file gives a description of the programs found in the assignment, describes the deliverables, provides pseudocode for each program, and gives credit to sources I used parts of my code from.

Pseudocode:

e.c

Include the following files:
mathlib.h
stdio.h

Declare static variable i

Create function e()
    Declare variable factorial and set it equal to 0
    Declare variable n

Create for loop using the variable i and make sure that (1/ variable factorial) is greater than epsilon as defined in mathlib.h. When the loop iteration is done, increment i by 1

Set variable factorial equal to factorial multiplied by variable i

Set variable n equal to n + (1/factorial)

Return the variable n

Create function e_terms()

Return the static variable i

madhava.c

Include the following files:

mathlib.h

stdio.h

Initialize static variable g

Create function pi_madhava()

Initialize variable current and set it equal to 1.0

Initialize variable sum and set it equal to 0.0

Initialize variable exp and set it equal to -3.0

Create for loop using the variable g and make sure that the absolute value of current is greater than epsilon as defined in mathlib.h. When the loop iteration is done, increment g by 1

Create if statement and check if g is greater than 0

Reset value of variable current to 1.0

Set variable current equal to current divided by exp

Set variable exp equal to exp multiplied by -3.0

Set variable current equal to current divided by (2g+1)

Set variable sum equal to sum + current

Set variable sum equal to sum multiplied by sqrt_newton(12)

Return the sum

Create function pi_madhava_terms()

Return the static variable g

euler.c

Include the following files:
mathlib.h
stdio.h

Create a static variable f

Create the function pi_euler()

  Initialize variable current and set it equal to 1.0
  Initialize variable sum and set it equal to 0.0
  Create a variable called temp

  Create for loop using the variable f and make sure that current is greater than epsilon as defined in mathlib.h. When the loop iteration is done, increment f by 1

    Set variable temp equal to f
    Set variable temp equal to temp multiplied by f
    Set variable current equal to 1.0 divided by temp
    Set variable sum equal to sum + current

  Set variable sum equal to sum multiplied by 6
  Set variable sum equal to sqrt_newton(sum)
  Return the variable sum

Create the function pi_euler_terms()

  # Because I started at f = 1, I need to subtract 1 from the total terms
  Return the static variable (f-1)

bbp.c

Include the following files:
mathlib.h
stdio.h

Define static variable k

Create function pi_bbp()

  Initialize variable current and set equal to 1.0
  Initialize variable sum and set equal to 0.0
  Initialize variable exp and set equal to 16.0

  Create for loop using the variable k and make sure that current is greater than epsilon as defined in mathlib.h. When the loop iteration is done, increment k by 1

    Set variable current equal to numerator of the Horner normal form (k(120k+151)+47)

Set variable current equal to current divided by denominator of Horner normal form (k(k(k(512k+1024)+712)+194)+15)

Create if statement where k must be greater than 0
Set variable current equal to current divided by variable exp
Set variable exp equal to exp multiplied by 16

Set variable sum equal to sum + current

Return the variable sum

Create function pi_bbp_terms()
Return the static variable k

viete.c

Include the following files:
mathlib.h
stdio.h

Define the static variable d

Create the function pi_viete()
Initialize variable current and set equal to 0.0
Initialize variable product and set equal to 1.0
Initialize variable repeat and set equal to sqrt_newton(2)

Create for loop using the variable d and make sure that current is less than 1 - epsilon.
When the loop iteration is done, increment d by 1
Set variable current equal to variable repeat divided by 2
Set variable product equal to product multiplied by current
Set variable repeat equal to sqrt_newton(2 + repeat)

Set variable product equal to 2 divided by product
Return the variable product

Create function pi_viete_terms()
Return the static variable d

newton.c

Include the following files:

mathlib.h
stdio.h

Define static variable newt_count

Create function sqrt_newton(x) with an argument of variable x
    Define variable z and set it equal to 0.0
    Define variable y and set it equal to 1.0
    Set newt_count equal to 0

    Create while loop with stipulation that the absolute value of (y - z) must be greater than epsilon
        Set variable z equal to y
        Set variable y equal to 0.5 multiplied by (z + x divided by z)
        Increment variable newt_count by 1

    Return variable y

Create function sqrt_newton_iters()
    Return the static variable newt_count

mathlib-test.c

Include the following files:
Mathlib.h
Math.h
Stdio.h
Unistd.h
Stdbool.h
Stdlib.h

Define OPTIONS with the value aebmrvnsh

Create a main function with the arguments argc and **argv
    Create a variable opt and set it equal to 0
    Create a boolean no_input and set it equal to true
    Create a boolean h_flag and set it equal to false
    Create a boolean a_flag and set it equal to false
    Create a boolean e_flag and set it equal to false
    Create a boolean m_flag and set it equal to false
    Create a boolean r_flag and set it equal to false
    Create a boolean b_flag and set it equal to false
    Create a boolean v_flag and set it equal to false
    Create a boolean n_flag and set it equal to false

Create a boolean s_flag and set it equal to false
Create a boolean s_true and set it equal to false

Create a while loop with the stipulation that opt equals getopt of argc, argv, and OPTIONS and that it doesn’t equal -1

Set boolean no_input to false

Create a switch construct with argument opt

Create a case for “h”
    Set h_flag to true
    Exit this case

Create a case for “a”
    Set a_flag to true
    Exit this case

Create a case for “e”
    Set e_flag to true
    Exit this case

Create a case for “b”
    Set b_flag to true
    Exit this case

Create a case for “m”
    Set m_flag to true
    Exit this case

Create a case for “r”
    Set r_flag to true
    Exit this case

Create a case for “v”
    Set v_flag to true
    Exit this case

Create a case for “n”
    Set n_flag to true
    Exit this case

Create a case for “s”
    Set s_flag to true

Check if (a_flag is false, e_flag is false, b_flag is false, m_flag is false, r_flag is false, v_flag is false, n_flag is false, h_flag is false)

Set s_true to true

Exit this case

Check if h_flag is true or no_input is true

Print the following:

SYNOPSIS

A test harness for the small numerical library.

USAGE

./mathlib-test [-aebmrvnsh]

OPTIONS

-a Runs all tests.

-e Runs e test.

-b Runs BBP pi test.

-m Runs Madhava pi test.

-r Runs Euler pi test.

-v Runs Viete pi test.

-n Runs Newton square root tests.

-s Print verbose statistics.

-h Display program synopsis and usage.

Set a_flag to false

Set e_flag to false

Set b_flag to false

Set m_flag to false

Set r_flag to false

Set v_flag to false

Set n_flag to false

Set s_flag to false

Set s_true to false

Check if s_true is true

Set s_flag to false

Print the following:

SYNOPSIS

A test harness for the small numerical library.

USAGE

./mathlib-test [-aebmrvnsh]

OPTIONS

-a Runs all tests.
-e Runs e test.
-b Runs BBP pi test.
-m Runs Madhava pi test.
-r Runs Euler pi test.
-v Runs Viete pi test.
-n Runs Newton square root tests.
-s Print verbose statistics.
-h Display program synopsis and usage.

Check if a_flag is true
    Print the following:
        e() = e(), M_E = M_E, diff = absolute value of (e() - M_E)
        Check if s_flag is true
            Print e() terms = e_terms()

        pi_euler() = pi_euler(), M_PI = M_PI, diff = absolute value of (pi_euler() - M_PI)
        Check if s_flag is true
            Print pi_euler() terms = pi_euler_terms()

        pi_bbp() = pi_bbp, M_PI = M_PI, diff = absolute value of (pi_bbp() - M_PI)
        Check if s_flag is true
            Print pi_bbp() terms = pi_bbp_terms()

        pi_madhava() = pi_madhava, M_PI = M_PI, diff = absolute value of (pi_madhava() - M_PI)
        Check if s_flag is true
            Print pi_madhava() terms = pi_madhava_terms()

        pi_viete() = pi_viete(), M_PI = M_PI, diff = absolute value of (pi_viete() - M_PI)
        Check if s_flag is true
            Print pi_viete() terms = pi_viete_factors()

        Create a for loop using variable t, t must be less than or equal to 10.0, increment t by 0.1
            Print the statement sqrt_newton(t) = sqrt_newton(t), sqrt(t) = sqrt(t), diff = Absolute value of (sqrt_newton(t) - sqrt(t))
            Check if s_flag is true
                Print sqrt_newton terms() = sqrt_newton_iters()

    Check if e_flag is true and a_flag is false
        Print e() = e(), M_E = M_E, diff = absolute value of (e() - M_E)
        Check if s_flag is true

Print e() terms = e_terms()

Check if b_flag is true and a_flag is false
Print pi_bbp() = pi_bbp, M_PI = M_PI, diff = absolute value of (pi_bbp() - M_PI)
Check if s_flag is true
Print pi_bbp() terms = pi_bbp_terms()

Check if m_flag is true and a_flag is false
Print pi_madhava() = pi_madhava, M_PI = M_PI, diff = absolute value of (pi_madhava() - M_PI)
Check if s_flag is true
Print pi_madhava() terms = pi_madhava_terms()

Check is r_flag is true and s_flag is false
Print pi_euler() = pi_euler(), M_PI = M_PI, diff = absolute value of (pi_euler() - M_PI)
Check if s_flag is true
Print pi_euler() terms = pi_euler_terms()

Check if v_flag is true and a_flag is false
Print pi_viete() = pi_viete(), M_PI = M_PI, diff = absolute value of (pi_viete() - M_PI)
Check if s_flag is true
Print pi_viete() terms = pi_viete_factors()

Check if n_flag is true and a_flag is false
Create a for loop using variable t, where t must be less than or equal to 10.0. When the loop iteration has been complete, iterate t by 0.1
Print the statement sqrt_newton(t) = sqrt_newton(t), sqrt(t) = sqrt(t), diff = Absolute value of (sqrt_newton(t) - sqrt(t))
Check if s_flag is true
Print sqrt_newton terms() = sqrt_newton_iters()

Credits:

Used the python pseudocode for sqrt_newton() from Professor Long on the asgn2.pdf

Created all of my programs from the formulas provided in the asgn2.pdf

Copied the text for the “h” case of mathlib-test.c from the output in resources/asgn2

Copied the mathlib.h file from resources/asgn2

Got the idea of using booleans in mathlib-test from TA Christian Ocon’s section video

Based my Makefile off of TA Christian Ocon’s Makefile template from section