

Deeraj Gurram
dgurram@ucsc.edu
7 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()` and `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 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, and provides pseudocode for each program.

Pseudocode:

e.c

Include the following files:

`mathlib.h`

`stdio.h`

Declare static variable `count`

Create function `e()`

 Declare variable `factorial` and set it equal to 0

 Declare variable `n`

 Set static variable `count` equal to 1

Create for loop using a variable i, make sure current is greater than epsilon, increment i
Set variable factorial equal to factorial multiplied by (i + 1)
Set variable n equal to n + (1/factorial)
Increment static variable count by 1

Return the variable n

Create function e_terms()

Return the static variable count

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 0.0

Initialize variable sum and set it equal to 0.0

Initialize variable exp and set it equal to -3

Create a for loop using static variable g, current must be greater than epsilon, increment g by 1

Create if statement and check if g is greater than 0

Set variable current equal to current multiplied by exp

Set variable exp equal to exp multiplied by -3

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

Set variable sum equal to sum + current

Set variable current equal to current 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 0.0

Initialize variable sum and set it equal to 0.0

Create a for loop using static variable f, current must be greater than epsilon, increment f by 1

Set variable current equal to 1 divided by (f multiplied by f)

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()

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 a variable k, current must be greater than epsilon, 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 $(k(k(512k+1024)+712)+194)+15)$

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

vieta.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 1.0

Initialize variable product and set equal to 0.0

Initialize variable repeat and set equal to sqrt_newton(2)

Create for loop using variable d, current must be greater than epsilon, 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 $(z + x \text{ divided by } z)$

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

E.c

Newton.c

Madhava.c

Bbp.c

Vieta.c

Euler.c

Define the name OPTIONS with the value aebmrnvsh

Create a main function with the arguments argc and **argv

Create a variable opt and set it equal to 0

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

Create a switch construct with argument opt

Create a case for "a"

Run all tests

Exit this case

Create a case for "e"

Initialize variable e_test

Set e_test equal to e()

Initialize variable e_diff

Set e_diff equal to the absolute value of $(e_test - M_E)$
Print the statement $e() = e_test, M_E = M_E, diff = e_diff$
Exit this case

Create a case for "b"

Initialize a variable bbp_test
Set bbp_test equal to $pi_bbp()$
Initialize a variable b_diff
Set b_diff equal to the absolute value of $(bbp_test - M_PI)$
Print the statement $pi_bbp() = bbp_test, M_PI = M_PI, diff = b_diff$
Exit this case

Create a case for "m"

Initialize a variable m_test
Set bbp_test equal to $pi_madhava()$
Initialize a variable m_diff
Set m_diff equal to the absolute value of $(m_test - M_PI)$
Print the statement $pi_madhava() = m_test, M_PI = M_PI, diff = m_diff$
Exit this case

Create a case for "r"

Initialize a variable eul_test
Set eul_test equal to $pi_euler()$
Initialize a variable eul_diff
Set eul_diff equal to the absolute value of $(eul_test - M_PI)$
Print the statement $pi_euler() = eul_test, M_PI = M_PI, diff = eul_diff$
Exit this case

Create a case for "v"

Initialize a variable v_test
Set v_test equal to $pi_viete()$
Initialize a variable v_diff
Set v_diff equal to the absolute value of $(v_test - M_PI)$
Print the statement $pi_viete() = v_test, M_PI = M_PI, diff = v_diff$
Exit this case

Create a case for "n"

Initialize a variable n_diff and set it equal to 0.0
Initialize a variable $newt_test$

Create a for loop using variable t , t must be less than or equal to 10.0, increment t by 0.1

Set variable $newt_test$ equal to $sqrt_newton(t)$
Set variable n_diff equal to the absolute value of $(newt_test - sqrt(t))$

Print the statement `sqrt_newton(t) = newt_test, sqrt(t) = sqrt(t), diff = n_diff`

Exit this case

Create a case for "s"

Create an if statement that getopt must have -s in it

Go to whichever other cases were specified

Print the number of computed terms for those cases

Create a case for "h"

Print the following:

SYNOPSIS

A test harness for the small numerical library.

USAGE

`./mathlib-test [-aebmrnvsh]`

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.

Exit this case

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`