Assignment 2 Design

Caitlin Smith

January 29, 2023

Program Description

In this assignment, I created multiple files containing mathematical functions that mimic the <math.h> functions to calculate e and pi. I created a program to then test my written functions against the <math.h> calculations. I utilized command-line arguments in the main function in order to call specific math functions.

1 e.c

Purpose: This contains the function using the Taylor series to approximate the value of e and track the number of computed terms. It also contains a function to return that tracked number. Below is the series I referenced.

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \frac{1}{720} + \frac{1}{5040} + \frac{1}{40320} + \frac{1}{362880} + \frac{1}{3628800} + \cdots$$

Pseudocode:

Include the mathlib.h file.
Create a static count to track the number of terms.

Create a function double e(void) that will return the approximated value of e.
Initialize double variables for the result, previous term, and current term all to 1.
Create a while loop that will run until the current term is less than epsilon.
    Increase the count.
    Set the current to the previous variable times 1 divided by the count.
    Set the previous to the current.
    Add the current to the result variable.
    Increase the count by 1 to account for the very first term of 1.
Return the result

Create a function int e_terms(void) that will return the number of terms, which is stored in the count variable.

2 madhava.c

Purpose: This calculates the approximation of pi using the Madhava series while tracking the number of computed terms. There is a function to return that tracked number.

$$\sum_{k=0}^{\infty} \frac{(-3)^{-k}}{2k+1} = \sqrt{3}\tan^{-1}\frac{1}{\sqrt{3}} = \frac{\pi}{\sqrt{12}}$$

Pseudocode:

Include the mathlib.h file.
Create a static count to track the number of terms.

Create a function double pi_madhava(void) that will return the approximated value of pi.
Initialize double variables for the result, term, the value of the previous -3 to the power of -k, and that current value.
Create a while loop that will run until the term is less than epsilon.
    Set the term to the current times 1 divided by the counter doubled plus 1.
    Set the previous to the current.
    Add the term to the result.
    Set the current to the previous times 1 divided by -3.
    Increase the count.
Return the result times the square root of 12, using sqrt_newton().

Create a function int pi_madhava_terms(void) that will return the number of terms, which is stored in the count variable.

3 euler.c

Purpose: This file contains a function approximating the value of pi based on Euler's solution to the Basel problem. It tracks the number of computed terms, and there is a function to return that value.

$$p(n) = \sqrt{6\sum_{k=1}^{n}\frac{1}{k^2}}$$

Pseudocode:

Include the mathlib.h file.
Create a static count to track the number of terms.

Create a function double pi_euler(void) that will return the approximated value of pi.
Initialize variables for the result and term.
Create a while loop that will run until the term is less than epsilon.
    Increase the count.
    Set the term to 1 divided by the square of the count (using count multiplied by itself).
    Add the term to the result.
Return the square root of the result multiplied by 6, using sqrt$_n ewton()$.

Create a function int pi_euler_terms(void) that will return the number of terms, which is stored in the count variable.

4 bbp.c

Purpose: This contains the function using the Bailey-Borwein-Plouffe formula for pi. It tracks the number of computed terms, and the file also contains a function to return that value.

$$p(n) = \sum_{k=0}^{n} 16^{-k} \times \frac{(k(120k+151)+47)}{k(k(k(512k+1024)+712)+194)+15}.$$

Pseudocode:

Include the mathlib.h file.
Create a static count to track the number of terms.

Create a function double pi_bbp(void) that will return the approximated value of pi.
Initialize variables for the result, previous value of 16 to the power of -k, and current term.
Create a while loop that will run until the current term is less than epsilon.
    Create variables for the numerator and denominator with the corresponding formulas.
    Set the term to the previous variable times the numerator divided by the denominator.
    Multiply the previous variable by 1/16.
    Add the term to the result variable.
    Increase the count.
Return the result.

Create a function int pi_bbp_terms(void) that will return the number of terms, which is stored in the count variable.

5 viete.c

Purpose: This contains the function using Viete's formula to approximate pi while also tracking the number of computed factors. There is a function to return this number.

$$\frac{2}{\pi} = \prod_{k=1}^{\infty} \frac{a_k}{2}$$

Pseudocode:

Include the mathlib.h file.
Create a static count to track the number of factors.

Create a function double pi_viete(void) that will return the approximated value of pi.
Initialize variables for the term, result, previous, and current.
Create a while loop that will run until the absolute value of the previous divided by 2 minus the term is less than epsilon.
    Set the previous to the current.
    Set the current to the square root of 2 plus the previous, using sqrt_newton().
    Set the term to current divided by 2.
    Multiply the result by the term.
    Increase the count.
Return 2 divided by the result.

Create a function int pi_viete_factors(void) that will return the number of factors, which is stored in the count variable.

6 newton.c

Purpose: This file contains the function to approximate the square root using the Newton-Raphson method. It tracks the iterations and there is a function to return that number.

Pseudocode:

Include the mathlib.h file.
Create a static count to track the number of iterations.

Create a function double sqrt_newton(double x) that will return the approximated square root of the passed x. Base it on the Python code below. Additionally, make sure to reset the count to 0 and increase the counter within the while loop.

```python
def sqrt(x):
    z = 0.0
    y = 1.0
    while abs(y - z) > epsilon:
        z = y
        y = 0.5 * (z + x / z)
    return y
```

Create a function int sqrt_newton_iters(void) that will return the number of iterations, which is stored in the count variable.

7 mathlib-test.c

Purpose: This file contains the main function. It utilizes the command-line to call the mathematical functions from the created .c files. It is used to compare the values from my functions to those in the <math.h> library.
Pseudocode:

Include <stdio.h>, <unistd.h>, <math.h>, and "mathlib.h".
Define OPTIONS to be "aebmrvnsh".

The following will all be within the main().

Initialize int opt.
Initialize flags for all of the command-line options except the all option. They are all set to 0.
Create a while loop with getopt.
Case a will 'turn on' all of the test flags.
Case e will set the e approximation test flag.
Case b will set the BBP pi approximation test flag.
Case m will set the Madhava pi approximation test flag.
Case r will set the Euler pi approximation test flag.
Case v will set the Viete pi approximation test flag.
Case n will set the Newton-Raphson square root approximation test flag.
Case s will set the statistic printing flag.
Case h will print the help message.
The default prints the help message.

If the e flag is set, run the e approximation and compare it to the math library result. If the s flag is set, print the stats.

If the b flag is set, run the BBP approximation and compare it to the math library result. If the s flag is set, print the stats.

If the m flag is set, run the Madhava approximation and compare it to the math library result. If the s flag is set, print the stats.

If the r flag is set, run the Euler approximation and compare it to the math library result. If the s flag is set, print the stats.

If the v flag is set, run the Viete approximation and compare it to the math library result. If the s flag is set, print the stats.

If the n flag is set, run the Newton approximation in the range [0,10] with steps of 0.1 and compare it to the math library results. If the s flag is set, print the stats.

Return 0.

8 matlib.h

Purpose: This is a given file containing the interface of the mathematical function library I created.

9 Makefile

Purpose: This is a file that is used to clean, format, and compile the entire program.