DESIGN.pdf ASGN2

Nathan Ong

October 11, 2021

1 Description

This collection of files contains a functional math library that has various methods that are used to approximate π and one method to approximate e. This library is also usable with the following command line options:

- -a: Runs all functions
- -e: Runs the e approximation function
- -b: Runs the Bailey-Borwein-Plouffe function
- -m Runs the Madhava series function
- -r: Runs the Euler solution function
- -v: Runs the Viete function
- -n: Runs the Newton-Raphson square root function
- -s: Enables printing to statistics to see the computed terms and factors for each function
- -h: Displays a help message detailing the usage of the program

1.1 Formulas

1. Bailey-Borwein-Plouffe Formula: $p(n)$ $\sum_{k=0}^{n} 16^{-k}(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k-6})$
2. Euler's Number (e): $\sum_{k=0}^{\infty} \frac{1}{k!}$
3. The Madhava Series: $\sum_{k=0}^{\infty} \frac{-3^{-k}}{2k+1} = \frac{\pi}{\sqrt{12}}$
4. Newton-Raphson Method ($\sqrt{x}$): $x_1 = 1.0, x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
5. Viete's Formula: $\frac{2}{\pi} = \prod_{k=1}^{\infty} \frac{a_k}{2}, a_1 = \sqrt{2}, a_k = \sqrt{2 + a_{k-1}}$
6. Euler's Solution to the Basel Problem: $p(n) = \sqrt{6 \sum_{k=1}^{n} \frac{1}{k^2}}$

2 Files Included in the Directory

1. mathlib-test.c
   (a) This file contains the main() function that tests the math library functions.
2. bbp.c
   (a) This file contains the code for the Bailey-Borwein-Plouffe formula to calculate an approximation of π along with an additional function that returns the number of computed terms.
3. e.c

(a) This file contains the code for the Euler's formula to calculate an approximation of e, or Euler's number, along with an additional function that returns the number of computed terms.

4. euler.c
   (a) This file contains the code for Euler's formula used to calculate an approximation of π along with an additional function that returns the number of computed terms.
5. madhava.c
   (a) This file contains the code for the Madhava series used to calculate an approximation of π along with an additional function that returns the number of computed terms.
6. newton.c
   (a) This file contains the code for Newton's method used to calculate an approximation of $\sqrt{x}$ along with an additional function that returns the number of computed terms.
7. viete.c
   (a) This file contains the code for Viete's formula used to calculate an approximation of π along with an additional function that returns the number of computed terms.
8. mathlib.h
   (a) This file contains the interface for the math library contained in mathlib-test.c.

3 Pseudocode and Structure

3.1 mathlib-test.c

while opt isn't -1
    switch statement for all library command line functions
perform functions with Boolean to separate called on functions from those that weren't requested

3.2 bbp.c

set static counter variable
  loop until computed term is over $x * 10^{-14}$
    calculate 16^{-k}
    calculate $\frac{(k(120k+151)+47)}{k(k(k(512k+1024)+712)+194)+15}$
    multiply 16^{-k} and above and add to summation
    increment term counter by 1
  return total term

3.3 e.c

set static counter variable
  loop until absolute value of the current term minus previous term is over $1 * 10^{-14}$
    calculate 1/n where n = (k-1)! * k
    add above to summation
    increment term counter by 1
  return total term

3.4 euler.c

set static counter variable
    loop until absolute value of the current term minus previous term is over $1 * 10^{-14}$
        calculate $1/k^2$
        add to summation
        increment total terms by 1
    multiply 6 and final summation
    get square root of above
    return final term

3.5 madhava.c

set static counter variable
    loop until absolute value of the current term minus previous term is over $1 * 10^{-14}$
        calculate $\frac{(-3)^{-k}}{2k+1}$
        add to summation
        increment term counter by 1
    calculate $\sqrt{12}$
    multiply $\sqrt{12}$ and final summation together
    return final term

3.6 newton.c

set static counter variable
    define and set two variables to 1.0 and 0.0 respectively (y and z)
        loop until the absolute value of y - z is greater than $1 * 10^{-14}$
            equate y to z
            y = 0.5 * (z + $\frac{x}{z}$)
            increment total terms by 1
    return y

3.7 viete.c

set static counter variable
    calculate $\sqrt{2}$
    loop until absolute value of the current term minus previous term is over $1 * 10^{-14}$
        calculate $\frac{a_k}{2}$ where $a_1 = \sqrt{2}, a_k = \sqrt{2 + a_{k-1}}$
        multiply above and total term together
    divide 2 by current term to get π
    return final term

4 Additional Credits

- I attended Sloan's in-person 1:15 PM section on October 6^{th}, and he provided the general structure of the Makefile including the shortening of implementations with various Linux commands.
- I derived the newton.c formula from the Python function given in the asgn2.pdf document.
- I derived the basis of mathlib-test.c from the supplied code given in the asgn2.pdf document.
- I watched the recording of Christian's remote section and used the comparison between the absolute value of the difference of the current computed term and the previous computed term and EPSILON.
- I watched the recording of Eugene's remote section and used Boolean statements in a similar fashion to regulate what functions were performed.

5 Error Handling

1. Numerous times during programming I encountered computational problems relating to either order of operations or just simple carelessness with calculating.
   (a) Solution: After some trial and error I understood what the correct solution to the problem was and corrected and in some cases, restructured the program to compute terms separately as to provide a clearer computation.
2. While coding the library test file, getting the command prompt to perform similarly, if not the same, as the binary required a lot of experimentation in order to get it to perform properly.
   (a) First, attempting to get the help message to appear when no input was detected was stumping at first, but upon further investigation the while loop isn't used, so a Boolean value was used to automatically make the message appear with no detected input.
   (b) Secondly, the number of terms was incorrect while working on this assignment. In fact it was doubled from its actual value, so I had to append a variable reset at the beginning of each function to prevent values from stacking on top of one another.