Assignment 2 - A Slice of Pi

Teresa Joseph

CSE 13S - Professor Long

Fall 2021 - RD October 7/ FD October 10

Purpose

Assignment 2 is the implementation of several mathematical functions, namely ones that serve to approximate values such as e and pi. These functions will be implemented in files called e.c, madhava.c, euler.c, bbp.c, viete.c, and newton.c. Another file called mathlib-test.c will also be made to test these files.

Breakdown of Functions

e.c

- Contains e() and e_terms()
    - e() approximates the value of e using taylor series and track the number of computed terms through a static variable local to the file
    - e_term() returns number of computed terms

madhava.c

- Contains pi_madhava() and pi_madhava_terms()
    - pi_madhava() approximate value of pi using Madhava series and track the number of computed terms with a static variable (like e.c)
    - pi_madhava_terms() returns number of computed terms

euler.c

- Contains pi_euler() and pi_euler_terms()
    - pi_euler() approximate the value of pi using formula from Euler's solution to Basel problem
    - pi_euler_terms() returns number of computed terms

bbp.c

- Contains pi_bbp() and pi_bbp_terms()
    - pi_bbp() approximates value of pi using Bailey-Borwein-Plouffe formula and track the number of computed terms
    - pi_bbp_terms() returns number of computed terms

viete.c

- Contains pi_viete() and pi_viete_factors()
    - pi_viete() approximates the value of pi using Viete's formula and track the number of computed terms
    - pi_viete_factors() returns number of computed terms

newton.c

- Contains sqrt_newton() and sqrt_newton_iters()
    - sqrt_newton() approximates the square root of the argument passed to it using the Newton-Raphson method and and track the number of iterations taken
    - sqrt_newton_iters() returns the number of iterations taken

mathlib-test.c

- Contain the main test harness for implemented math library
- Should support -a (run all tests), -e (run e approximation test), -b (run bbp pi approximation test), -m (run Madhava pi approximation test), -r (run Euler pi approximation test), -v (run Viete pi approximation test), -n (run Newton square root approximation tests), -s (enable printing statistics to see computed terms and factors for each tested function), and -h (display a help message detailing program usage)

Pseudocode

e.c

```
define e()
        k = 0 (for summation formula)
        y = 0 (for tracking summation)
        counter for terms = 0
        while 1/k! > epsilon, do summation
                x = 1/k!
                y += x
                increase k by 1
                increase counter by 1
        return y (the approximation of e)
define e_terms()
        call e()
        return counter
```

madhava.c

define pi_madhava()

k = 0 (for summation formula)

y = 0 (for tracking summation)

counter for terms = 0

while (-1/3)^k/(2k+1) > epsilon, do summation

x = (-1/3)^k/(2k+1)

y += x

increase k by 1

increase counter by 1

calculate the sqrt of 12 with newton and multiply it with y

return final y value (the approximation of pi)

define pi_madhava_terms()

call pi_madhava()

return counter

euler.c

define pi_euler()

k = 1 (for summation formula)

y = 0 for tracking summation

counter for terms = 0

while 1/k^2 > epsilon, do summation

x = 1/k^2

y += x

increase k by 1

increase counter by 1

take sqrt of 6y and return this value (the approximation of pi)

define pi_euler_terms()

call pi_euler()

return counter

bbp.c

define pi_bbp()

```
	k = 0 (for summation formula)
	y = 0 (for tracking summation)
	counter for terms = 0
	while __ > epsilon, do summation
		x = 16^(-k)*(4/(8k+1) - 2/(8k+4) - 1/(8k+5) - 1/(8k+6))
		y += x
		increase k by 1
		increase counter by 1
	return y (approximation of pi)
define pi_bbp_terms()
	call pi_bbp()
	return counter
```

viete.c

```
define pi_viete()
	k = sqrt(2) (for summation formula, will use newton here)
	y = sqrt(2)/2 (for tracking summation, use newton here)
	counter for iters = 0
	while (last iterator) > epsilon, do summation
		x = sqrt(2 + k)/2
		y *= x
		increase (add) k to k
		increase counter by 1
	return 2/y (approximation of pi)
define pi_viete_factors()
	call pi_viete()
	return counter
```

newton.c

```
define sqrt_newton(x)
	y = 0.0
	z = 1.0
	counter for iterations = 0
```

while absolute value of (z-y) > epsilon, apply formula

y = z

z = 0.5 * (y + x / y)

increase counter by 1

return z (the approximation of sqrt)

define sqrt_newton_iters(x)

call sqrt_newton(x)

return counter

mathlib-test.c

check command line inputs

if -a, run all tests (return all)

if -e, run e approximation test

if -b, run bbp pi approximation test

if -m, run Madhava pi approximation test

if -r, run Euler pi approximation test,

if -v, run Viete pi approximation test,

if -n, run Newton_raphson square root approximation tests,

if -s, enable printing statistics to see computed terms and factors for each tested function

if -h, display a help message detailing program usage

Notes

- mathlib.h library will be included
- While loops may be replaced with for loops if my implementation ideas change later
- The iterator in viete.c may be altered (unsure of its implementation at the moment)

Overall Description

All files except mathlib-test.c will have two functions, one that approximates either e or pi and another that returns the number of terms or iterations needed to reach that approximation. These functions have similar setups: variables will be initialized, while loops will iterate over certain conditions, and returns at the end will provide approximations/the number of iterations. viete.c will require more contemplation, as its iterator is tricky to implement, but it is expected to follow a similar pattern. The file mathlib-test.c will test my implementations by running them with command line inputs as described in the assignment document.

<u>Goals/Intended Process</u>

- Replicate the described pseudocode for each file in C
- Address possible errors
- Make code more readable and efficient if possible
- Add sufficient comments and clean up format