Assignment 2 A Little Slice of π

Design Document

Derrick Ko - Winter 2023

1 Overview

Creating our own math functions that will be almost identical precision to the original C math library.

2 bbp.c

This contains the implementation of the Bailey-Borwein-Plouffe formula to approximate π and the function to return the number of computed terms.

2.1 double pi_bbp(void)

approximate the value of π using the Bailey-Borwein-Plouffe formula and track the number of computed terms.

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

- create a static global variable to keep track of computed terms
- loop until 16^{-k} is less than EPSILON
- each loop divide by 16 for the amount of loops. For example, Loop 1 divide by 16, Loop 2 divide by 256 etc.

2.2 int pi_bbp_terms(void)

return the number of computed terms

3 e.c

This contains the implementation of the Taylor series to approximate Euler's number e and the function to return the number of computed terms.

3.1 double e(void)

approximate the value of e using the Taylor series and track the number of computed terms by means of a static variable local to the file.

$$\frac{x^k}{k!} = \frac{x^k - 1}{(k-1)!} \times \frac{x}{k}$$

- create a static global variable to keep track of computed terms

- Loop until $\frac{x}{k}$ is less than EPSILON which means the value will be small enough to be negligible
- new = previous * current;
- previous = new;
- This effectively calculates the factorial using a shortcut since we know the previous output.

3.2 int e_terms(void)

return the number of computed terms.

4 euler.c

This contains the implementation of Euler's solution used to approximate π and the function to return the number of computed terms.

4.1 double pi_euler(void)

approximate the value of π using the formula derived from Euler's solution to the Basel problem. It should also track the number of computed terms.

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

- create a static global variable to keep track of computed terms
- starting at 1 loop until $\frac{1}{k^2}$ is less than EPSILON which means the value will be small enough to be negligible
- in the loop calculate $\frac{1}{k^2}$
- after loop done calculate the sum of $\frac{1}{k^2}$ iterations and multiply it by 6 and sqrt it.

4.2 int pi_euler_terms(void)

return the number of computed terms

5 madhava.c

This contains the implementation of the Madhava series to approximate π and the function to return the number of computed terms.

5.1 double pi_madhava(void)

approximate the value of π using the Madhava series and track the number of computed terms with a static variable, exactly like in e.c.

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

$$\frac{\pi}{2} 3^{-n-1} 3^{n+\frac{1}{2}} = \frac{\pi}{\sqrt{12}}$$

- create a static global variable to keep track of computed terms
- loop until 16^{-k} is less than EPSILON
- (-1 or 1) negative or positive according to the iteration number
- multiply by $\sqrt{12}$ to cancel out $\frac{\pi}{\sqrt{12}}$ which is π

5.2 int pi_madhava_terms(void)

return the number of computed terms

6 viete.c

This contains the implementation of Viète's formula to approximate π and the function to return the number of computed factors.

6.1 double pi_viete(void)

approximate the value of π using Viète's formula and track the number of computed factors.

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

- create a static global variable to keep track of computed terms
- loop until the iteration changes by smaller than EPSILON
- use the formula to calculate pi

6.2 int pi_viete_factors(void)

return the number of computed factors

7 newton.c

This contains the implementation of the square root approximation using Newton's method and the function to return the number of computed iterations.

7.1 double sqrt_newton(double)

approximate the the square root of the argument passed to it using the Newton-Raphson method. This function should also track the number of iterations taken

- create a static global variable to keep track of computed terms
- using long's code of newton

7.2 int sqrt_newton_iters(void)

returns the number of iterations taken.

8 mathlib-test.c

This contains the main() function which tests each of your math library functions.

- -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.