Noel Ball
CSE13S

mathlib-test.c Design Document

Description:

An executable test harness mathlib-test takes command line arguments to run different estimation functions, their terms, and a help menu. The functions include approximating e, and approximating pi with the Euler sequence, Bailey-Borwein-Plouffe formula, Madhava series, Viete's formula, and estimating square roots with the Newton-Raphson method. The test harness can run all the functions, specific functions, enable stats, and include the help menu. The functions are displayed with their approximation, compared to math.h library's value, and the difference to demonstrate accuracy.

Goal:

Run ./math-lib-test with chosen arguments, and getopt() parses the input argument. A switch-case statement is used to enable a boolean value for which functions need to be run, as well as if stats or the help menu are enabled. A set of if-statements run the functions and format the data to print for which boolean values are true. Stats are printed on the next line after each function is run.

Sudo:

All the functions use a static int to track the number of computed terms, which is reset to 0 at the beginning of each function call to avoid the term count stacking from multiple calls.

Noel Ball
CSE13S

All the functions track the iteration step as k in a do-while loop for the summations and the product. The previous term is stored in a variable every iteration, starting at 0 for the summations and 1 for the product. The loop starts by adding the last term to the total, not affecting the total until the second iteration. The term is computed, depending on the method, then stored in the previous term at the end of the loop. The counter variables are updated, and then the term is checked against epsilon before the next iteration, to be added to the sum until the term is smaller than epsilon.

File e.c's approximation function is e() and starts with a k of 0. To calculate the 1/k! term throughout each iteration, the function runs a for loop and multiplies the denominator by the iterating variable to act as a factorial, then 1 is divided by the calculated k! to add that term to the sum until the loop range is met.

File euler.c's approximation function is pi_euler() and starts with a k of 1. To calculate the 1/k^2 term throughout each iteration, the function calculates the denominator by multiplying k*k and divides 1 by the value to add that term to the sum until the loop range is met. The final sum is then multiplied by 6 and square rooted.

File bbp.c's approximation function is pi_bbp() and starts with a k of 0. To calculate the 16^-k term throughout each iteration, the function runs a for loop to multiply 1 * 16 k times, so when k is 0 the term is 1, and when k > 0 the term is 16 multiplied by itself k times to act as an exponent. 1 is divided by that value because k is negative, and it is multiplied by the rest of the formula.

Noel Ball
CSE13S

File madhava.c's approximation function is pi_madhava() and starts with a k of 0. The same loop method is used as pi_bbp() to calculate the (-3)^k in the denominator of the term, and that value is multiplied by the rest of the denominator from the series. The final sum is multiplied by the square root of 12.

File viete.c's approximation function is pi_viete() and starts with a k of 1. The first a_k of the term is calculated by adding 2 to the previous a_k and square rooting it. That a_k is divided by 2 to create the term to be multiplied to the total. 2 is then divided by the final sum.

File newton.c's approximation function is sqrt_newton(). It uses a while loop to multiply .5 with the previous term plus x divided by the previous term. The absolute value of the current term minus the previous term is checked against epsilon, and the final term is the estimation.