Assignment 2 - A Little Slice of π Design

Sean Carlyle

October 4 2021

1 Introduction

This is my design for a little slice of pi. It is a program that finds π, the square root of a number, and e in various ways. It expects inputs -a -e -r -b -m -v -n -s -h. As an output it prints the approximate value I achieved of either e, π, or square root. Then it prints the math.h version of the value and the difference between mine and the math libraries. -a stands for all which will call all functions. -e stands for e.c which will approximate e using the Taylor series method. -r stands for euler.c which will approximate π using the euler method. -b stands for bbp.c which approximates π using the Bailey-Bornwin-Plouff formula. -m stands for madhava.c which approximates π using the madhava formula. -v stands for viete.c which approximates π using the viete formula. -n stands for newton.c which approximates the square root of a number. -s stands for statistics and shows how many terms/factors or iterations it took to find our approximation for each function. -h stands for help and prints out an explanation of the program.

2 Pseudocode

mathlib-test.c

Take number of arguments in command line and characters in command line
While all parsing command line arguments

Depending on what option was chosen in the command line, mark them to be called

If taylor version of e was marked, call and print e. Also print the difference from the math.h e. If statistics was also marked, print the number of terms it took to get the e.

If euler was marked, call and print the output of euler. Also print the difference from the math.h π. If statistics was also marked, print the number of terms it took to get the π.

If bbp was marked, call and print the output of bbp. Also print the difference from the math.h π. If statistics was also marked, print the number of terms it took to get the π.

If viete was marked, call and print the output of viete. Also print the difference from the math.h π. If statistics was also marked, print the number of factors it took to get the π.

If newton was marked, call and print the output of newton. Also print the difference from the math.h π. If statistics was also marked, print the number of iterators it took to get the π. Do this for numbers 0-10 at every 0.1 interval.

Create a difference function

Return the absolute value of the first term minus the second

e.c

Create counter i

While one over i factorial is greater than the epsilon value

Add 1 divided by i to to the total sum

Track number of terms

Then multiply i by itself plus one;

Return the total sum

viete.c

Take the square root of 2

While the numerator of the equation is less than 2 minus the epsilon value

Multiply the numerator (which is the square root of 2) into a product.

add 2 to the numerators

take the square root of the numerators

count the number of factors

return 2 divided by the product

madhava.c

While the each term of the madhava equation is not between +/-epsilon

Add 1/(2k+1)*(-3*i) to the total sum

Count the number of terms

Add one to k

Multiply i by -3

Return the total sum multiplied by the square root of 12

euler.c

Add i as the variable

While the each term of the euler equation is greater than epsilon

Add 1/(i*i)

Increase i by 1

Count the number of terms

Return the square root of the total sum multiplied by 6

bbp.c

Add i as the variable

While the each term of the bbp equation is greater than epsilon

Equate the current term and add it to the sum

Let the 16 to the power instead the number 1 that is multiplied by 16 every term

Increase i by 1

Count the number of terms

Return the summation

newton.c

While the absolute value of the new value - old value is greater than epsilon

Count iterations

Make old value new value

Estimate new value

Return new value

3 Explanation

mathlib-test.c

Mathlib-test is designed to first check the for which arguments are going to be called. Then flip bool values for each argument whether to call them or not. Then, call the functions which had their bool values set to true. When these functions are called print them out along with the difference from the math.h version. It is easier to put the print statements outside of the case switch because it allows to call all the printfs at once with -a. If they were in the switch case you could only call all of them by writing all arguments.

e.c

e approximates the value e. It does this by summing 1/k! until the value of each term is less than epsilon. Instead of calculating factorial for every term I instead just multiply the denominator each term by k+1. This way I get the same number as the factorial but do not have to re-multiply every term.

viete.c

Viete finds $2/\pi$ by finding the product of many factors. These factors are each sqrt(2 + the previous factor's numerator) / 2. The first factor is sqrt(2) / 2. I decide to keep multiplying factors until the numerator of a single factor is not at least an epsilon away from 2. I do this because the numerators of the factors eventually approach 2 and do not increase further since sqrt(2 + 2) / 4 is equal to 1. I use our sqrt newton() function to approximate the sqrt value. I return the value of 2 / the product since this approximates π.

madhava.c

Madhava is a summation that approximates π. I loop until the terms are less than or greater than -/+ epsilon respectively. This is because the way Madhava approximates is by alternating between positive and negative terms each bringing the sums value closer to π. Instead of doing multiplying out -3 to the exponent every time I instead just multiply -3's value every term since this is the same number as if I actually raised it to the power of k (the iterator). But this saves computing time because I do not need to re-compute the power terms value each time. I return the final value multiplied by the sqrt(12) as the equation calls for it.

euler.c

Euler approximates π by summing 1/(k*k). Then multiplying that value by 6 and square rooting the final number. I choose to loop until a single term is less than epsilon. I just increase i by one every iteration of the loop and keep adding to the total sum. At the end I multiply the final sum by 6 and square root it with our sqrt newton() function.

bbp.c

bbp uses a summation formula to approximate π. I loop until a single term is less than epsilon. For all values i I just increase i by one every term of the summation. Since I have a 16 to the power i, instead of calculating it every term I multiply the sixteen power by 16 each term so that I do not have to calculate sixteen to the power i every term. The total sum is returned at the end.

newton.c

Newton approximates the square root of a number based on the Newton-Rhapson method. It subtracts the estimate value by the function over its derivative and then assigns this new value to the estimate. This slowly makes the value approach the root of the original number. Once the difference between the estimate and the new value is below epsilon newton returns the value.

4 Diagram of Program