cse13s asgn2 DESIGN.pdf

Lucas Lee; CruzID: luclee

1/14/2022

1 Program Details

This program is designed to create a library of math functions inside of a file called mathlib.c. We must create an exponent function that displays:

1. function that approximates e^x
2. function that approximates sin(x)
3. function that approximates cos(x)
4. function that approximates $\sqrt{x}$
5. function that approximates log(x)
6. function that uses Simpson's 1/3 rule for integration using the previous functions

We will need to use $\epsilon = 10^-14$ to limit the computations.

2 Pseudocode/Program Ideas

Making e^x:

1. initialize x and k
2. for loop that uses ϵ to check differences(floating point absolute value) and break loop when approximation is close to the value of ϵ
    (a) multiply previous x/k! by x/k
        i. use local variables to keep track of previous values in order to computer factorial without using a factorial command

Making Sin(x) and Cos(x):

1. initialize local variables(most likely need about 3-5 to placehold numbers)
   (a) while loop checks if floating point absolute value is greater than ϵ
      i. implement $\frac{x^n}{n!} + (-1)^n \frac{x^n+2}{n+2!}$
      ii. for Cos(x) start at 0 instead of x and implement a similar function to Sin(x)
      iii. variables hold previous values - use for factorials
      iv. variable that holds n value, and also one that controls the (-1) coefficient.

Making $\sqrt{x}$ and log(x):

1. set variables x and y, y being equal to x^2, as well as a n or k variable to keep track of iterations.
2. create while loop that compares floating point abs to ϵ.
   (a) take half of the given x + $\frac{y}{x_{korn}}$
   (b) return that number after loop breaks

Making log(x)

1. log(x) function will need to incorporate our exp() function to get e^x
2. log(0) does not exist, so we start at x + 1.
3. make vars and while loop checking for ϵ again with floating point abs.
   (a) since e^x is the inverse of log(x), we use it in our function to invert back to that form when it's used.
   (b) implement a similar function to $\sqrt{x}$, except use y - e^x and its derivative instead of $x^2 - y$.

Lastly: the integrate(function pointer, lower bound a, upper bound b, unsigned integer n) function:

1. This integrate function uses Simpson's 1/3 rule.
2. The function pointer takes a single double value as an input, so we can do arithmetic inside function-pointer(...) in order to get a value we want.

3. We can follow a similar structure to the Simpson's 3/8 rule in the pdf for the assignment.

4. Set sum equal to f(lower bound) + f(upper bound)

5. Set h = (b-a)/n (as stated in asgn2 document)

6. for loop (lets call current loop iteration i)from 1 to n

   (a) if conditional = odd/even

      i. if even add to sum: 2 * f(a + i(2((b-a)/n)))

      ii. if odd add to sum: 4 * f(a + i(2((b-a)/n)-1))

   (b) after loop finishes, add to sum: h/3.

Above are the programs for the mathlib.c file that should contain the math library functions. The Makefile for this assignment should use clang commands, clang flag commands, build the integrate executable file with linkers, clean files that the compiler makes, and format source code. For integrate.c I will need to be able to take arguments from command line using argc and **argv.

1. main function:

   (a) create opt, boolean values for each option

   (b) create a variable as function pointer

   (c) while loop: conditional = getopt != -1

      i. switch opt

      ii. for each case, set corresponding boolean to true and set function pointer equal to corresponding function from functions.c.

      iii. store number variables using strtod and optarg

      iv. allow the input of "pi" in command line by comparing strings

      v. make a default case and H case that prints options

   (d) conditionals that check each case's boolean operator

   (e) run integrate in each conditional for increasing number of partitions.

2. return int

3 Files

1. functions.c - file is provided: contains function implementations that
2. functions.h - file is provided: header file for functions.c shows functions that should be in main program
3. integrate.c - contains integrate() and main() functions that integrate functions based on command line inputs
4. mathlib.c - contains above programs for math library
5. mathlib.h - contains implementation of mathlib.c functions
6. Makefile - stated above
7. README.md - markdown formatted document that describes how to use the program and other bugs/errors in program
8. DESIGN.pdf - this document: describes program details and pseudocode for the programs
9. WRITEUP.pdf - pdf that includes graphs of each function when plotted using gnuplot, as well as other tools used to finish the project. Graphs should use relative difference and other ideas relating to floating point numbers.