Assignment 2

Numerical Intergration

DESIGN.pdf

Reuben T. Chavez

January 21, 2022

Description of Program

The following file is my implementation of the ¡math.h¿ library functions: *e*,sin, cos, log, *square root, and integrate*. They can find the closest approximation of a number to the 1e-14. The implementation of each of these mathematical functions uses the Taylor series. The Taylor Series is a sequence of numbers that add the last term to the following term; each function uses an iteration to replicate that sequence and approximate the value. The program can also be called onto another file to use its mathematical functions. At the same time, when the program itself is called on the terminal, the main() function is used to implement the integrated function on the terminal line. Depending on which option and function are chosen, the integrated function will return the integration of a specified mathematical equation.

Files in asgn2

Brief Description of the following files are as followed:

- **functions.c**: Provides and contains the implementations that the program mathlib.c should integrate
- **functions.h**: Header file that contains the functions prototypes that main program should integrate
- **integrate.c**: Contains the integrate() and the main() function to perform the integration of specified lines by the command-line over the specified interval
- **mathlib.c**: Contains the implemtaions of each of my math library functions
- **mathlib.h**: Header file that contains the interface of my math library
- **Makefile**: File that stores commands to complies the .c programs to executable programs , format the program in clang format, and removes any unnecessary binary files before the program, through the command make clean
- **README.md**: Marks Down File that briefly describes the program list how the program can be run and built on the terminal.
- **DESIGN.pdf**: A .pdf file that covers the purpose the program, has a lay out and structure of the program, with clear description/explanation on how each part works
- **WRITEUP.pdf**: A .pdf file that contains illustrations of the graphs produced through the program to analyze the differences between the math library and mathlib. Along with supporting arguments and reasoning for the outputs

Pseudeocode

```
#python

# Psuedocode for the Intergrate Function

def simpson(f,a,b,n):
    h = (b - a) / n  #interval width
    sum = f(a) + f(b)      #first term and last term

    even = 0.0
    for j in range(1, n / 2 -1): #even terms
        even += f(a + 2 * j * h)
    sum += 2.0 * even

    odd = 0.0
    for j in range(1, n/2): #odd terms
        odd += f(a + (2 * j - 1) * h)
    sum += 4 * odd
    return sum * (h / 3)
```

Notes on Pseudeocode and Structure
It first initializes the interval width $h = (b - a)/n$, the step length according to assignment 2. The sum variable is then initialized to equal the summation of the first and last term of the equation, $f(a) + f(b)$. The rest of the code then finds the summations for the even terms and odd terms. First, the even variable is set to 0 to store the total after passing the first arithmetic series represented as a for a loop. The first for loop represents the first arithmetic series within the formula, where from 1 to n/2 -1 it will add onto the variable **even**, where it will add specifically the *first term plus + 2 *current iteration * interval width* and then runs that in the specified mathematical function as input. After being multiplied by 2, it is added to the total sum. Then, the odd variable is initialed to store the total after passing the next iteration. The next for loop follows the next arithmetic series within the formula for all the odd terms within the sequence; where from 1 to n/2, it will add the *first term plus + (2 *current iteration -1) * interval width* and save the total onto odd. After the variable odd is multiplied by 4, it is added to the sum. Finally, the sum is multiplied by interval width / 3 and returned to the output.

```
Psuedo code

get mathlib library
get functions libary
```

```
# Psuedocode for the Intergrate
def usage(executable)
    print(
    "Synopsis of Integrate"

    "Usage of Integrate "

    "Options for Integrate"
    )

def main(sys.argv):
    #intialize varibles
    initialize getopt to 0
    initalize double pointer double function
    intialize function as empty strings
    intialize float varibles low, high, mid , min, max
    initialize interger n as 100

    #while loop that uses getop() for command line
    while(argmunts in terminal line are being pulled)
        #switch case used to parse exact commands

        In the case of 'a':

           set main and max to the bounds of the sqrt(1 - x*x*x*x)


    return 0
```

Credit

- I watched Eugen's Lecture to implement the getop() 1/14/22
- I used the professor's code comments for reference for my integrate function
- Along with using the following info to create my own variation of the integrate function
  - `https://www.bragitoff.com/2017/08/simpsons-13-rule-c-program/#:~:text=Simpson's%20Rule%20is%20a%20Numerical,function%`

20within%20a%20given%20interval.&text=And%20the%20area%
20is%20then,the%20better%20is%20the%20approximation.

- https://www.codesansar.com/numerical-methods/integration-using-simpson-1-3-r
htm
- https://www.codewithc.com/c-program-for-simpson-1-3-rule/
- https://en.wikipedia.org/wiki/Simpson%27s_rule#Simpson'
s_1/3_rule