CSE 13S Winter Quarter 2022
Assignment 2: Numerical Integration

Description of the Program:

The goal of this assignment is to create a math library of e^x, $sin(x)$, $cos(x)$, $\sqrt{x}$, $log(x)$. Additionally, we will need to find the integral for certain values for various functions that are pre determined.

Files to be included in the “asgn1” directory

- integrate.c
    - This is created by us to find the values of different integrals specified by the start value, the end value, the function, and the precision via number and therefore size of the rieman intervals e^x, $sin(x)$, $cos(x)$, $\sqrt{x}$, $log(x)$
- mathlib.c
    - Recreating mathlib.h using methods specified in the assignment doc. Requires us to approximate values for
- functions.c
    - This is provided and gives us a link from integrate.c to mathlib.c
- functions.c
    - This is also provided and completes the link from integrat.c to mathlib.c
- mathlib.h
    - This is provided and allows us to compare our own mathlib to the mathlib provided.
- Makefile
    - This is used to help compile our code.
- README.md
    - This is a file written in markdown format to explain the purpose of the code and how to build it, in addition to any errors we found.
- DESIGN.pdf
    - Gives insight into some of the processes of coming up with the code, along with the purpose of the assignment.
- Writeup.pdf
    - Explains the assignment further and our thoughtprocess. Additionally, shows graphs and explanation into our work.

Pseudocode:

Python pseudocode for e^x, $sin(x)$, $\sqrt{x}$, $log(x)$ all found in the asgn2 doc.

```
def cos(x, epsilon = 1e-14):
        s, v, t, k, = 1, 1, x, 2.0
        while abs(t) > epsilon:
        t = t * (x * x) / ((k -1) * k)
        s = -s
        v += s* t
        x += 2.0
return v

def simpsonthird(function, start, end, partitions):
        avg = (end - start) / partitions
        integral = 0
        for (i in range(1, partitions, 2):
                integral += 2 * function(avg * i + start)
        for (i in range(1, partitions, 2):
                integral += 4 *  function(avg * i + start)
        integral += function(start)
        integral += function(end)
        integral /= integral / 3
        yield integral

def a(x):
        return sqrt(1 - x * x)

def b(x):
        return 1 / math.log(x)
def c(x):
        return e ** (-x * x)
def d(x):
        return math.sin(x * x)
def e(x):
        return math.cos(x * x)
def f(x):
        return math.log(math.log(x))
def g(x):
        return math.sin(x) / x
def h(x):
        return e ** - x / x
```

```
def i(x):
        return e ** (x * x)
def j(x):
        return sqrt(math.sin(x) ** 2 + math.cos(x) ** 2)
python version of getopt
def sum(f, l, h, n, function, start, end, partitions):
        f = input(“enter function: “)
        l = input(“low range: “)
        h = input(“high range: “)
        n = input(“partitions: “)
        if f == a:
                return simpsonthird(a, l, h, n)
        if f == b:
                return simpsonthird(b, l, h, n)
        if f == c:
                return simpsonthird(c, l, h, n)
        if f == d:
                return simpsonthird(d, l, h, n)
        if f == e:
                return simpsonthird(e, l, h, n)
        if f == f:
                return simpsonthird(f, l, h, n)
        if f == g:
                return simpsonthird(g, l, h, n)
        if f == h:
                return simpsonthird(h, l, h, n)
        if f == i:
                return simpsonthird(i, l, h, n)
        if f == j:
                return simpsonthird(j, l, h, n)
        if f == h:
                print(“usage and synopsis”)
```

Notes about Pseudocode:

- for the cosine, we need to use an absolute value to essentially do a modulus 2π
- for the simpsons integral, we can just use yield because we are in python, however, in c, we need to use a map and a print_array function as shown in Eugene’s section

- for getopt, we were shown to use atoi, for the low and high, however atof is what we should be using, as atoi only returns integers, and floating point numbers are possible inputs for the low and high ranges, such as π.
- Additionally, we need to use a loop in each of the (if f == *) to print each output from the integrate function, because we want to see the individual outputs for every single partition count.

Inefficiencies:

- I tried to use the map and print_array functions like Eugene did in his section, however, I wasn't able to figure it out with a pointer in the integrate function. Therefore, I had to create the same two for loops in every single getopt link, which I know is a big inefficiency and could have saved me about 50 or so lines of code.

Credit:

- I attended Eugene's section and got more insight on how to link functions with #include and got some insight on how to code the portions in c
- The Friday lecture gave insight on how to approximate each of these values, however, most of it was covered again in the assignment doc