Computing Lab #3: Simple integration algorithm

Due: Tuesday, October 3, 2017 (midnight)

Summary: Introduction to compiled languages and simple algorithmic analysis.

Programming language: Write, compile, and run a Fortran, C or C++ program. Your choice! This lab assumes the C programming language.

Points: 10

I. TASK #1: GETTING STARTED: COMPILE AND RUN A PROGRAM (5 POINTS)

Task: Write, compile, and run the most basic program possible (one which prints "Hello World!"). Then extend this program in a few interesting ways. The program you write in task 1 will be used in task 2.

- **Part a** Write a program to print "Hello World!".

  1. I will give you the code below. Create a file called hello.c and add these contents:

     ```
     // This is a C comment, the compiler will ignore it
     // we include stdio.h = header file for the standard io (Input/Output) library
     // this library defines the function printf
     #include <stdio.h>

     // every C program should have this part, "int main(){"
     int main(){

       printf("Hello World!\n");
       return 0; // every C program should have this part, return 0;
     }
     ```

  2. Compile the source code into an object file (this object file is the program you will run)

     ```
     >>> gcc -o sayHello hello.c
     ```

  3. Now run the program by executing it

     ```
     >>> ./sayHello
     ```

- **Part b** Modify your program to accept command-line input. This input will necessarily be a *string*. In your code, convert the string to an integer, N, output this number to the screen.

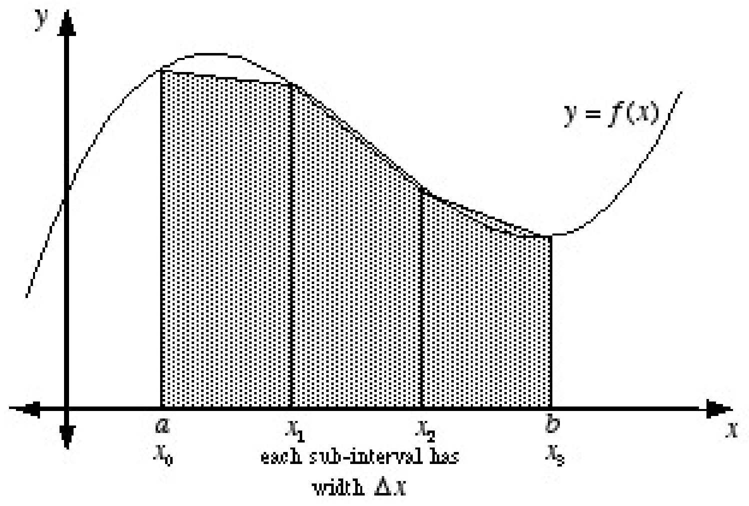
  1. Look through today's lecture notes to see how to pass input into your program (search for "Command-line input"). Pay special attention to the ordering of the argv list: argv[0] is the executable name, argv[1] is the input you will pass. Simply print this input to screen using printf.
  2. Convert this string to an integer (hint: google "string to int" to find out how) and store this in a variable N. This line of code will look something like:

     ```
     int N = // code here to convert input to an integer
     ```

  3. Compile and run your code. Check that you've got a working program

II. TASK #2: TRAPEZOIDAL RULE (5 POINTS)

Learning objective/task: In this second task you will write a program for the Trapezoidal rule. On a computer, this formula is a perfectly reasonable algorithm for computing an integral. Verify theoretical properties of this algorithm. You are encouraged to use the internet and Unix manuals to solve this problem.



The area of the trapezoids (shaded) approximately equals the area bounded by $y = f(x)$.

$$\int_a^b f(x)dx = \frac{\Delta x}{2}\left[f(x_0) + 2f(x_1) + 2f(x_2) + f(x_3)\right].$$

The specific version of the Trapezoidal rule you will code is the so-called extended form (see the above figure). This algorithm was discussed in class, although the formula for this algorithm is all over the internet and you should no problem tracking it down. In particular, you can find it here: https://en.wikipedia.org/wiki/Trapezoidal_rule.

The integration code you will write will numerically compute:

$$\int_{-1}^{1} f(x)dx\,. \tag{1}$$

- If you've completed task 1, you have a program which accepts input and converts it to an integer N.
- If you use N equally space points to sample the interval $[-1, 1]$, what is the spacing between each point? Compute this number and output it's value to the screen. This will be your Δx value used in Trapezoidal rule's formula.
- Add a for-loop to your program to generate the points x_i you need to carry out the Trapezoidal rule. Print these values to screen and verify their correctness.
- Use the Trapezoidal rule to compute

  $$\int_{-1}^{1} dx\,,$$

  and verify the result is 2.
- Use the Trapezoidal rule to compute

  $$\int_{-1}^{1} \frac{1}{1+x^2} dx\,.$$

  Verify the algorithm converges to $\pi/2$ as $N \to \infty$ by running for increasingly larger values of N.
- Modify your code to output the data (N, absolute error) for each run. For example, your output might be

```
>>> ./trap 100
100 .01
```

- Write a Unix script which calls your program for $N = 2^i$ for integers i from 1, 2, 3, ..., 20 and use the Unix redirect operator >> to record this data in a file (just like lab 2). Here's a hint on how to start your script – run it to see what it does:

```
#!/bin/bash

i=1
while [[ i -le 20 ]]
do
    echo $i
    ((i = i + 1))
done
```

- Make a plot of N vs error on a log-log plot. On a log-log scale, your error curve should look like a straight-line, suggesting the error is roughly $E(N) \approx (\Delta x)^C$ for some constant C. Does the value of C you measure match your expectations?

III. UPLOAD YOUR WORK TO GIT

Congratulations! You've completed the lab. Now lets upload your work to git.

1. Open up a terminal and change into your personal git project folder (not the class folder).
2. Create a new directory to place your work

```
>>> mkdir lab3
```

3. Move your answers to task 1 (the source code) and task 2 (the source code, the Unix script, and the figure) into the lab3 folder.
4. Make git aware of this new file by doing

```
>>> ls      # COMMENT: make sure you see "lab3" in the folder
>>> git add lab3
```

5. Commit the files to the git project along with a commit message:

```
>>> git commit -m "Lab 3 answers"
```

6. Upload the files to Bitbucket

```
>>> git push
```

7. Finally, open the web browser and go to your bitbucket page. Check the files have been uploaded.