AMS 595/DCS 525 – C++ Project 2

Due: December 5th, 2023 – 11:59 PM

NOTE:. *You must write your code on your own. Do not copy code from any source. To get full credit your codes must be well documented with comments. Submit your codes to GitHub* ***and*** *Brightspace along with brief documentation and any relevant plots. Your codes should have the correct file extensions and should run* ***without*** *any needed modifications.*

Background

The value of π can be computed using the following formula

$$\int_0^1 \sqrt{1-x^2}\, dx = \frac{\pi}{4}\,. \tag{1}$$

To evaluate this integral numerically, we may partition the interval $[0,1]$ as $\{x_k\}_{k=0}^N$ and use the *trapezoidal rule*:

$$\int_a^b f(x)dx \approx \sum_{k=1}^{N} \Delta x_k \Big(\frac{f(x_k)+f(x_{k-1})}{2}\Big) \tag{2}$$

where $\Delta x_k = x_k - x_{k-1}$.

Q1. Approximating π (30 points)

Write a function called `pi_approx` that approximates π using (1) and (2). This function should be written inside a separate source file called `pi_approx.cpp`. The number of intervals, N, should be taken as an input into the function. Your program should return both the computed value of π as well as the corresponding absolute error (the built-in value of π in C++ is `M_PI`).

HINT: Functions in C++ cannot have multiple return values. To circumvent this and return both the approximation and the error at once, create a `struct` called `PiResults`, with the approximation and error values being named `approx` and `error`, respectively. You may find the library `cmath` useful to compute exponents, square roots, absolute values, and the floor function.

Q2. Dynamic Memory, Pointers, and Arrays (30 points)

Write a function called `approximations` that takes a vector of integers (which you may assume to all be positive) as an input. For each integer n_i in this vector, compute an approximation of π using Q1 and n_i intervals. Store these approximations in a *dynamically* allocated C-style array and return this array. Your function should be written inside a separate source file called `approximations.cpp`.

Q3. Header Files and Makefiles (30 points)

Create a source file called `HW2main.cpp` in whose main function you will be calling your functions from Q1 and Q2. In particular,

- Print the π approximation and error for $N = 10000$ using Q1.
- Create a vector with elements $(10^1, 10^2, \ldots, 10^7)$. Print out the elements of the array from Q2 using this vector as an input.

Proper use of header files and header guards are required to receive full credit. Furthermore, write a makefile to compile this program into an executable called `HW2main`. In particular, your makefile should:

- Include a rule to remove all object files in the current directory
- Have proper use of variables (e.g. for compiler type, compiler flags, name of object files...etc).

Optional Bonus (30 Points)

NOTE: *Only those students who did not attempt the python extra credit will receive credit for this problem.*

Write a function called `pi_approx2` that again approximates π using (1) and (2), in particular, you may use your function from Q1. However, this time have the user input their maximum acceptable error. Your program should find the necessary number of intervals such that the error tolerance is respected. As an output, return the error and the number of intervals. For this problem use a `struct` called `Pi2Results` with values `approx` and `intervals`.

There are several ways of achieving this. For example, one could estimate the number of intervals to use with the asymptotic error formula of the trapezoidal rule. But to practice your C++ skills, you should instead use the following approach: first try to evaluate the integral with a small number of intervals, if the resulting error is greater than the tolerance then try again with a larger number of intervals. By going back and forth you can narrow down the necessary number of intervals. You may want to enforce an upper limit for the number of intervals to try. For example, a very large number of intervals will be needed to satisfy any error tolerance less 10^{-8}.

To receive full credit you must do this in an efficient and intelligent manner (i.e. not a brute force solution), and the number of intervals your code outputs should be correct within an error of ± 1 interval.

HINT: To avoid overflow, use appropriate data types and data type modifiers (only where necessary).

Grading Scheme

- 90 points for correctness of code.
- 10 points for documentation, proper coding practices (commenting, ease of reading, etc.), and correct use of Git.