Advanced Object Oriented System Design Using C++

Assignment 6

Foreword

This Assignment's text description is long. It is long to help you understand things. If you've questions, ask! That said, remember what the high-level goal is: to compute the numerical integration of a function. (When you took calculus, you would have done this as integration using the Reimann Sum Approximation, Trapezoid Rule, and/or Simpson's Rule.)

Overview

It is not always possible to symbolically determine the integral of a function, f, but there are many applications that need to compute such an integral. The solution is to compute an approximation of such by numerically integrating it.

This assignment involves writing a function template capable of numerically integrating a function, f, within an interval $[a,b]$ by subdividing it N times. This is known as the Reimann Sum Approximation, i.e.,

$$\int_a^b f(x)dx \approx \sum_{k=1}^n f(c_k)\Delta x_k$$

where c_k is the midpoint of the kth subinterval within $[a,b]$ whose width is Δx_k.

Two functions in this assignment will be numerically integrated:

$$\int_0^1 e^{-x^2} dx$$

which is a function important in statistics which must be approximated since f does not have a closed-form antiderivative in terms of elementary functions; and the other function is:

$$\int_0^1 \frac{1}{1+x^2} dx = \frac{\pi}{4}$$

which converges very slowly --so it is not a good way to compute π but it is useful to demonstrate and test numerical integration.

Additionally, it is common for documentation to give instructions to compile code by defining a macro to control what is produced. This assignment requires macros to be defined when the code is compiled which enables you to explore using some of the preprocessor directives to control this.

Finally, a purpose of this assignment is to see how generics, i.e., C++ templates, can be used to write a "compile-time" polymorphic function capable of numerically integrating *any* function using a simple call. This course's lectures have presented function objects, function pointers, etc. By using templates one can simply say "pass in some function here" and let the compiler sort out its substitution and code generation. In an object-oriented language like Java, one would write abstract methods and classes designed to do the same --except that polymorphism will occur at run-time. Since the C++ template mechanism is 100% compile-time, this leads to more efficient/optimized code --albeit with larger generated binary files (e.g., .exe files) due to the many template instantiations necessary to achieve that kind of compile-time polymorphism.

Before You Start

- This assignment computes the numerical integration of a function in a **memory inefficient** manner. This is intentional and designed to allow one to explore some of additional Standard Library algorithms.
  - Know using a special helper iterator class (not provided in this assignment) one can numerically integrate a function efficiently without using an array/std::vector at all using only one call to std::transform_reduce().
- This assignment uses the rectangle method so the midpoint of each interval is computed.
- This assignment invokes purely sequential code --no parallel algorithms are used.
  - Note: The next assignment will use parallel algorithms and will be similar to (but not completely the same as) this assignment.

- Should you explore changing with the number of intervals to integrate over, be cognizant of the amount of free RAM that is available on the machine you are using since this assignment will allocate RAM proportional to the number of intervals, e.g., on cs340.cs.uwindsor.ca please keep the number of intervals small.
  - You can see the amount of free RAM there is on a Linux machine by running "free -h" at the command line.
- Typically, in C++, preprocessor directives are used for (i) #include guards and (ii) to help make code portable across operating systems and architectures. To explore such one needs access to compilers under difference operating systems and/or for different architectures. Since that is out-of-reach with available resources (especially with everyone at home), this assignment instead uses macros to control which function is numerically integrated instead.

Task

#include files

This assignment will require using a number of #Include files. Look up the #include files required for the following items and ensure your assignment code #includes them.

- std::size_t, std::vector, std::numeric_limits
- std::iota(), std::transform(), std::transform_reduce(), std::exp()
- std::cout, std::scientific, std::setprecision

main()

Your main() function will need to be after the numerical_integration_serial() function below (unless you prototype-declare it before main()). main() is discussed here since it is straight-forward to implement. To write main do the following:

1. Create a for loop using a **std::size_t** counter variable i that starts at 1, keeps iterating while i <= 10000, and each time it iterates i is multiplied by 10. For example, i's values will be 1, 10, 100, 1000, 10000. Everything below is inside this for loop.
2. The first statement inside the for loop must call numerical_integration_serial() passing in these arguments: i, 0.0, 1.0, and a lambda function. (The lambda functions will be described below. Start by writing one of those functions and get your code to work correctly first.)
3. As part of the previous step, store the value returned by numerical_integration_serial() in a variable (so it can be output in the next step).
4. Output using std::cout the following in the order listed: (i) "integration estimate of f over [0,1] using ", (ii) i, (iii) " intervals: ", (iv) use std::scientiic to cause floating-point numbers ot be output using scientific notation, (v) use std::setprecision() passing in the result of std::numeric_limits<double>::digits10+1, and (vi) the variable holding the result described in the previous step (as detailed below).

main(): The Lambda to Compute exp(-x*x)

As mentioned above, start by only writing one function in your code.

When writing this lambda function as an argument to numerical_integration_serial(), don't write the call and lambda code all in one line --break it up as this will make it more readable, etc.

This lambdas code is as follows:

- This lambda has no captures.
- This lambda has one argument. Per the mathematics given above, the argument is a floating-point number, i.e., double in this assignment, i.e., your argument can be auto const& or double const&.
  - ASIDE: It is good practice to make read-only arguments const even if they don't have to be.
- **Update: The code inside the function must be exp(-x*x)** in order to numerically integrate $\int_0^1 e^{-x^2}\,dx$.

main(): The Pi Estimation Lambda

With the above function written and tested in your code (come back to this after you've done the rest of the assignment), it is straight-forward to have a second function passed as the *same* argument to numerical_integration_serial() as the exp(-x*x); lambda function. This can be achieved using the C/C++ preprocessor. Before adding the preprocessor directives needed, start by writing the Pi Estimation lambda function:

- This lambda has no captures.
- This lambda has one argument. Per the mathematics given above, the argument is a floating-point number, i.e., double in this assignment, i.e., your argument can be auto const& or double const&.
- **Update: The code inside the function must be 1.0 / (1.0 + x*x)**, i.e., the function in $\int_0^1 \frac{1}{1+x^2}\,dx$.

With such, now you can write some preprocessor directives to control the inclusion of one function during compilation which will exclude the other from being compiled. It is assumed below that the $\int_0^1 e^{-x^2}\,dx$ lambda appears before the $\int_0^1 \frac{1}{1+x^2}\,dx$ lambda. To achieve this do the following:

- Before the $\int_0^1 e^{-x^2} dx$ lambda --but *after the comma of the previous argument* to numerical_integration_serial()-- add the preprocessor directive **#ifdef USE_EXP_FUNC** on its own line.
  - If you did not break your function call up in to several lines --you must do this!!!
  - Each preprocessor directive has to be on its own line and there can only be whitespace before the # character.
- After the lambda function but before the closing parenthesis of numerical_integration_serial(), add the preprocessor directive **#endif**.
- Before the **#endif**, add the preprocessor directive **#elif defined(USE_PI_ESTIMATE)** and ensure your definition for the $\int_0^1 \frac{1}{1+x^2} dx$ lambda appears after such (but before the **#endif**).
  - Although there is #ifdef, there is no #elifdef. Instead one needs to use #elif deifned(MACRO_NAME).
  - Instead of #ifdef MACRO_NAME one can use #if defined(MACRO_NAME) instead.
  - Know defined(MACRO_NAME) also permits one to do more than one defined() test as well, e.g., #if defined(NDEBUG) && defined(MACRO_NAME) --but such is not needed here.
- Finally, before the **#endif**, add **#else** followed by **#error "USE_EXP_FUNC or USE_PI_ESTIMATE must be #defined."**.
  - Remember each preprocessor directive must be on its own line!
  - The result is if neither macro is defined, "USE_EXP_FUNC or USE_PI_ESTIMATE must be #defined." will be output in a compiler error message. **Your program must do this!**

To control the output of the variable:

- Use the preprocessor directive **#ifndef USE_PI_ESTIMATE** to output the variable **as-is**.
- Use the preprocessor directive **#else** to output the variable **multiplied by 4** for the π estimate.
- Don't forget to add **#endif** after that.

NOTE: Don't write separate cout statements to do this: use one statement --just break it up over multiple lines and within each preprocessor if section, have the code needed.

TIp: When writing your code, start by writing and using only one function. After that works, add the other function and the preprocessor directives. If you want to test the second function without adding any preprocessor directives, then comment the code out for the function you don't wish to use for that test. :-)

Know that normally preprocessor directives are only used as #Include guards and to handle source inclusion/exclusion for things concerning operating system, compiler, and/or architecture portability --not to control which function gets used when a program is run. But this example allows you to see and explore how one can control which source code lines are compiled by the compiler. :-)

NOTE: Outside of the #include files, no other part of the program uses preprocessor directives.

numerical_integration_serial()

Start by writing the opening of this function:

```
template <typename Domain, typename Func>
auto numerical_integration_serial(
  std::size_t N,
  Domain const& a,
  Domain const& b,
  Func&& f
)
{
```

where N is the number of intervals within the range $[a, b]$ for the function f. The task of this function is to call the needed C++ standard library functions in order to estimate the actual value of the integration of f from a to b.

Since N might be set to zero, if N is zero, set it to one. This simplifies subsequent code since it does not have to deal with division by zero. :-)

The C++ Standard Library does not have a special iterator type that appears to point to an array of numbers from 0 to N without requiring any storage. Thus, you will need to populate a std::vector containing N std::size_t values, starting at 0 and ending at N-1. Do this as follows:

- Declare the vector and name it indices.
  - Use the constructor to create N values --don't call reserve(). The std::iota() function requires the elements to already be in the container.
- Call std::iota() to populate the vector with values starting at 0.

To make it easier to write some of the code below, define the following:

- Declare **delta_x** to have type **Domain const** and set its value to be $(b - a) \div N$.
- Declare **half_delta_x** to have type **Domain const** and set its value to be **delta_x / 2**.

With indices defined, compute the midpoint of each interval and store all results in a vector as follows:

- Declare a vector containing type Domain values. The vector has N elements.
  - Use the constructor to create N values.
- Call std::transform to map each indices element to that interval's midpoint (i.e., "x") value as follows:
  - Look up std::transform() and read what its arguments are.
  - The "output" iterator for std::transform will be the begin() of your vector of domain values.
  - Pass in a lambda function as the fourth argument as follows:
    - This lambda will capture variables by reference.
    - This lambda has one argument (per std::transform() documentation).
      - While you can use **auto const&**, its type is actually std::size_t because that is the type of what is in indices. If this is not clear, re-read the description of what std::transform() does. Do look up std::transform in TCPPSL.
    - The body of this function needs to compute the indices element multiplied by delta_x + half_delta_x.

With the "x" midpoint of each interval computed, one can now numerically integrate the function, f in a single return statement as follows:

- Write return.
- Call std::transform_reduce().
  - As always, look up std::transform_reduce() read about it and understand what it does.
  - Another name for transform() is "map". So this call is a "map-reduce" call.
  - Mapping maps a set of values to another set of values.
    - e.g., converting index values to interval midpoint values.
  - A reduction applies a function over a set of values which results in a single value.
    - e.g., summing up a list of numbers produces one number (that is the sum).

At first glance, std::transform_reduce() might seem overwhelming. It is not --re-read its description again and think about what it is doing. It requires more inputs because it is doing transform() calls followed by reduce() calls --in a more efficient, combined manner. The arguments for std::transform_reduce() you want are as follows:

- The first two iterators is the iterator range of your interval midpoint vector.
- The third argument is: decltype(f(a)){}.
  - What is this, you ask? When decltype is used in C++, one can write code inside of it, i.e., f(a), and it will return the resulting type of that code. In this case, f and a are the arguments that were passed to numerical_integration_serial(), so the resulting type of f(a) is the type that f(a) returns. The {} after the decltype default constructs that type. That is, this argument represents the starting value of the summation needed to compute the integration result, i.e., a "zero" value is needed for whatever type f(a) returns. :-)
  - ASIDE: In C++, default constructing a type can be thought of as creating the "zero" value of that type. With all numeric types, this is that type's zero value.
- The fourth argument should be a lambda as follows:
  - No captures. It has two arguments. Return the sum of the arguments.
  - i.e., this is the reduction function: summation.
- The fifth argument should be a lambda as follows:
  - Capture by reference, one argument c.
  - Returns f(c) * delta_x, i.e., $f(c_k)\Delta x_k$.
    - Earlier you computed c, i.e., the midpoint, this now finally computes the value that is summed.

Whew but that is it! This function is three calls: std::iota(), std::transform(), and std::transform_reduce() + the code to set up the vectors' data needed for those calls.

Something To Ponder

Programming with generics/templates is a different way of thinking --and it can be very powerful. This function template can numerically integrate any function. While the functions this assignment used double values, there is no reason why std::complex, mathematical vector/matrix types, etc. could not have been used. The only requirements are that Domain is the type of f's argument, that the return type's "zero" value is created by its default constructor and the return type's values can be meaningfully summed.

Sample Program Output

This assignment will be compiled twice as follows:

- g++-10.2.0 -std=c++20 -Wall -Wextra -O2 -DUSE_EXP_FUNC -o a6-exp.exe a6.cxx
- g++-10.2.0 -std=c++20 -Wall -Wextra -O2 -DUSE_PI_ESTIMATE -o a6-pi.exe a6.cxx

When ./a6-exp.exe is run, the program must produce as output:

```
integration estimate of f over [0,1] using 1 intervals: 7.7880078307140488e-01
integration estimate of f over [0,1] using 10 intervals: 7.4713087774799747e-01
integration estimate of f over [0,1] using 100 intervals: 7.4682719849231971e-01
integration estimate of f over [0,1] using 1000 intervals: 7.4682416346904890e-01
integration estimate of f over [0,1] using 10000 intervals: 7.4682413311899509e-01
```

When ./a6-pi.exe is run, the program must produce as output:

```
integration estimate of f over [0,1] using 1 intervals: 3.2000000000000002e+00
integration estimate of f over [0,1] using 10 intervals: 3.1424259850010978e+00
integration estimate of f over [0,1] using 100 intervals: 3.1416009869231249e+00
integration estimate of f over [0,1] using 1000 intervals: 3.1415927369231262e+00
integration estimate of f over [0,1] using 10000 intervals: 3.1415926544231265e+00
```

Should you wish to try out running the program with a larger number of intervals (on your own machine, should you have enough RAM), this output is provided:

```
$ ./a6-exp.exe
integration estimate of f over [0,1] using 1 intervals: 7.7880078307140488e-01
integration estimate of f over [0,1] using 10 intervals: 7.4713087774799747e-01
integration estimate of f over [0,1] using 100 intervals: 7.4682719849231971e-01
integration estimate of f over [0,1] using 1000 intervals: 7.4682416346904890e-01
integration estimate of f over [0,1] using 10000 intervals: 7.4682413311899509e-01
integration estimate of f over [0,1] using 100000 intervals: 7.4682413281549509e-01
integration estimate of f over [0,1] using 1000000 intervals: 7.4682413281245852e-01
integration estimate of f over [0,1] using 10000000 intervals: 7.4682413281238069e-01
integration estimate of f over [0,1] using 100000000 intervals: 7.4682413281262860e-01
$ ./a6-pi.exe
integration estimate of f over [0,1] using 1 intervals: 3.2000000000000002e+00
integration estimate of f over [0,1] using 10 intervals: 3.1424259850010978e+00
integration estimate of f over [0,1] using 100 intervals: 3.1416009869231249e+00
integration estimate of f over [0,1] using 1000 intervals: 3.1415927369231262e+00
integration estimate of f over [0,1] using 10000 intervals: 3.1415926544231265e+00
integration estimate of f over [0,1] using 100000 intervals: 3.1415926535981313e+00
integration estimate of f over [0,1] using 1000000 intervals: 3.1415926535898606e+00
integration estimate of f over [0,1] using 10000000 intervals: 3.1415926535897034e+00
integration estimate of f over [0,1] using 100000000 intervals: 3.1415926535894174e+00
```

The more intervals there are the more accurate the values computed are. How quickly a function converges to an accurate value will vary with the function and the technique (e.g., Simpson's rule) used.

Submission

When done, upload your a6.cxx program! :-)

Submission status

Attempt number	This is attempt 1.
Submission status	Submitted for grading
Grading status	Not marked
Due date	Thursday, 1 April 2021, 11:59 PM
Time remaining	5 days 5 hours

Grading criteria

This rubric is a general rubric for compiled programming languages.

Normally the marker will adhere to the rubric, but, the marker reserves the right to add/subtract marks based on the quality of the work provided. (Such overrides will be noted in the feedback comments.)

Compiler Warnings	Too many compiler warnings. *0 points*	No more than two (2) compiler warnings. *1 points*	No compiler warnings. *2 points*
Compiler Errors (Deduction)	Code fails to compile due to compiler errors. *-5 points*	No compiler errors. *0 points*	
Possible Run-Time Errors	Too many run-time errors occur or can occur. *0 points*	Only a few minor run-time errors occur or can occur. *1 points*	No run-time errors occur or can occur. *2 points*
Code Structure, Names, and Comments	Poor code structure, meaningful symbol names, and comments. *0 points*	Satisfactory code structure, meaningful symbol names, and comments. *1 points*	
Packaging	Not all files are provided and/or not all targets are built cleanly and completely. *0 points*	All files are provided. All targets are built cleanly and completely. *1 points*	
Unit Tests	Too many tests (provided or not) don't pass having input or output processing and correctness issues. *0 points*	Nearly all tests (provided or not) pass and have no input or output processing and correctness issues. *1 points*	All tests (provided or not) pass and have no input or output processing and correctness issues. *2 points*
Assignment Requirements Conformance	Not all instructions were followed. *0 points*	Most instructions were followed. *1 points*	All instructions were followed. *2 points*
Late Penalty (Deduction)	Late. All marks deducted. *-10 points*	Late but submitted within 24 hours of deadline or late without penalty deadline. *-2 points*	Not late or late without penalty (as decided by or with permission of instructor). *0 points*

Last modified	Saturday, 27 March 2021, 6:12 PM	
File submissions	a6.cxx	27 March 2021, 6:12 PM
Submission comments	Comments (0)	

Edit submission Remove submission

You can still make changes to your submission.

◄ Week 9 Lecture Files

Jump to...

Floating-Point Accuracy ►