EECS402 Winter 2025 Project 1

One very important aspect of calculus is being able to compute the area between a curve and the X-axis. There are different ways to compute this value, and for this project, you will implement an algorithm to approximate this area (similar to, but a little different than a definite integral). You do not need to have a strong calculus background to perform this project – it is just specified this way to give you a feel of how programming can be applied to a real-world problem. These specifications look long, but the reason is that the details are fully described to make the project easier.

Submission and Due Date

You will submit your program using an email-based submission system by attaching *one C++ source file* only (named exactly "project1.cpp"), and *one script file* only (named exactly "typescript"). Do not attach any other files with your submission – specifically, do *not* include your compiled executable, etc. The due date for the project is **Tuesday, January 28, 2025 at 4:30pm**. Early submission bonus will be applied as described in the course syllabus.

No submissions will be accepted after the submission final deadline. As discussed in lecture, please double check that you have submitted the correct files (the correct version of your *source code* file named exactly as specified above and the typescript file, correctly generated showing you building your project executable and running valgrind). Submission of the "wrong files" will not be grounds for an "extension" or late submission of the correct files, and will result in a score of 0.

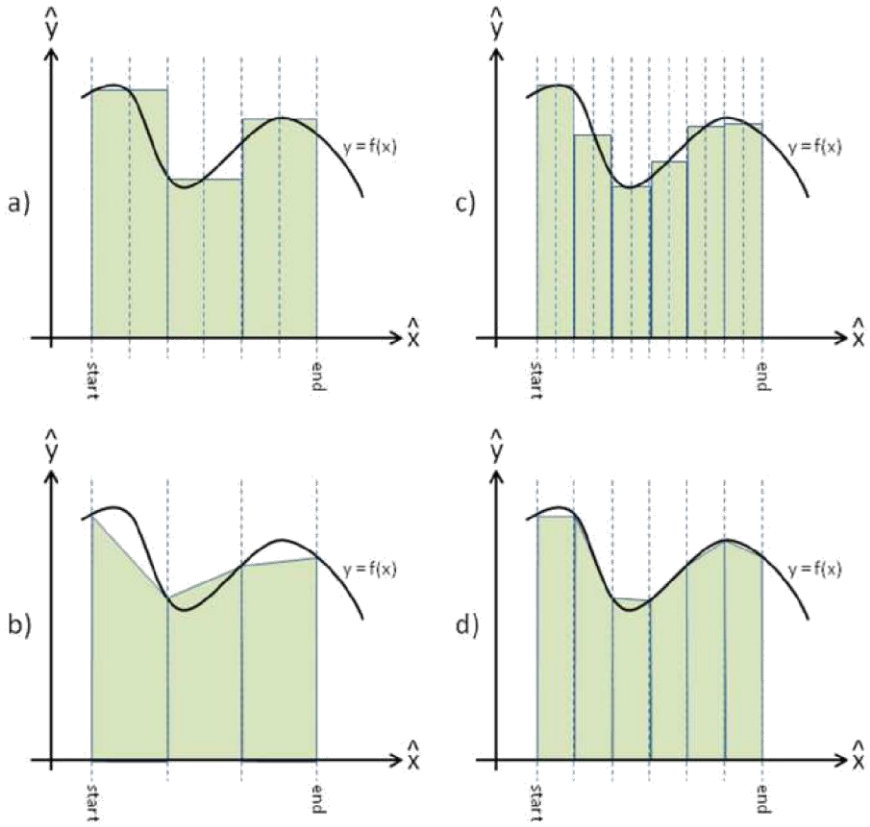
High-Level Description

One way to approximate the area between a curve and the X-axis is to fit many polygons between the curve and the axis. By choosing a polygon that has an area that is easily computed, we can approximate the actual area by adding the area of all the polygons. Remember that area is a positive value always, so in this project, we aren't quite computing an integral, but rather an "enclosed area". What we are computing could be better described as the area under the curve when the curve is above the x-axis, summed with the area above the curve when the curve is below the x-axis.

There are numerous polygons that can be used. For this project, we will use rectangles since they are simple to work with. When fitting rectangles, it must be decided how to determine the height of the rectangle. There are three widely used choices. The first possibility evaluates the function describing the curve at the rectangle's left-most x location. The second uses the rectangle's midpoint along x. The third uses the rectangle's right-most x value to determine the height. For this project, you need to only worry about the second possibility - that is, using the midpoint. The width of the rectangle is determined by taking into account the X-interval of the area of interest and the number of rectangles that need to be fit into that interval.

The below diagram shows 4 examples of approximating the area using polygons. Example (a) uses 3 rectangles to approximate the area between the curve y=f(x) and the X-axis from x = start to x = end.

Example (b) uses 3 trapezoids. Example (c) uses 6 rectangles to get a better approximation, and example (d) uses 6 trapezoids. The trapezoid examples are just to illustrate how the approximated area might differ with different polygons. You are *not* expected to implement area approximation with trapezoids for this project!



The first choice the user of your program will have is to determine the approximate area between a cubic function and the X-axis using some number of rectangles. The user must enter 4 coefficients for a cubic equation (I.e. $y = a * x^3 + b * x^2 + c * x + d$, where a, b, c, and d are the coefficients provided), the x start and end values for the area of interest's interval, and the number of rectangles to be fit within the interval.

When dealing with approximations, there is usually some sort of tolerance, or precision you are willing to accept. For example, one engineer might be willing to accept up to 0.05 error in an area approximation, while another engineer working on a space vehicle launch mechanism might only be willing to accept an error of 0.000001. In order to allow the user to better understand how many rectangles are needed in general for a good approximation, you will allow the user another choice. In this case, the user will provide the correct (known) answer to the integration, and provide the tolerance they are willing to accept. Your job is to tell the user how many rectangles were needed to achieve that tolerance. For example, if I'm willing to accept a tolerance of 0.005, then how many rectangles would I need to use to get within 0.005 of the correct answer? Since the user may provide an incorrect "correct

answer" you should stop trying to achieve the specified tolerance after trying 100 rectangles, and report that you were unable to reach the specified tolerance using the maximum number of rectangles.

Requirements

The following global functions are required for this project. You may not modify the function prototype as given for ANY reason. This includes changing the order of parameters, the types of parameters or return values, or renaming the function in ANY way. When describing functions, words in "quotation marks" usually refer to the parameters that are passed in.

`void printMenu();`

This function prints out the possible choices the user has, in a menu format. For the exact format expected, see the sample outputs on the project's Canvas page. No input is done in this function!

`double toThePower(const double val, const int power);`

This function raises "val" to the power "power" and returns the result. For example, toThePower(4.0, 3) would return 64.0. No input or output is done in this function! You can assume all powers will be greater than 0. Remember, you may NOT use anything from the math library, so don't call the "pow" function in your implementation, even if you know how.

`bool evaluateCubicFormula(const double aCoeff, const double bCoeff, const double cCoeff, const double dCoeff, const double xValue, double &resultVal);`

This function evaluates the cubic formula specified as:

y = aCoeff * xValue^3 + bCoeff * xValue^2 + cCoeff * xValue + dCoeff

at the x value specified as "xValue". The resulting y value is output back to the calling function as the reference parameter "resultVal". The function will return false if the resulting y value is a negative value, or true if the resulting value is greater or equal to zero.

`double approximateAreaWithRectangles(const double aCoeff, const double bCoeff, const double cCoeff, const double dCoeff, const double startX, const double endX, const int numRects);`

This function will approximate the area between the X-axis and the curve defined by the formula:

y = aCoeff * x^3 + bCoeff * x^2 + cCoeff * x + dCoeff

Rectangles are used to approximate the area. The rectangle's width will be determined by considering the interval of interest and number of rectangles that are to be used. The rectangle's height is determined by evaluating the function at the rectangle's midpoint along x (width). The area of interest's interval is from "startX" to "endX", and "numRects" rectangles should be used to do the approximation. No input or output is done from this function! Since the exact number of rectangles is passed in as input, you should use a count-controlled loop to iterate that number of times. Prior to iterating, determine the width of the rectangles (do this only one time). To determine the x coordinate of the rectangles within the loop, simply add the width each iteration of the for loop over the number of rectangles. If you follow

this design, your results should match the sample output's exactly. For example, if the X-interval is from 4.0 to 10.0 and you need to fit 5 rectangles, you would compute each rectangle's width as (10.0 – 4.0) / 5, which is 1.2. Then, your first rectangle would span X = 4.0 to X = (4.0 + 1.2 = 5.2), and its X midpoint would be X = (4.0 + 5.2) / 2 = 4.6. The next rectangle would span X = (4.0 + 1.2 = 5.2) to X = (5.2 + 1.2 = 6.4) with a midpoint of X = (5.2 + 6.4) / 2 = 5.8. (etc.) The last rectangle would span X=8.8 to 10.0 with a midpoint of 9.4.

`int main();`

The main function will loop until the user chooses to exit. Each iteration, the user will be prompted for input of a choice from the menu that will be printed. Once a choice is made, the user will be prompted for all necessary input, and the requested operations should be performed. Appropriate output should be printed to the screen when the operation completes. The return value of main should be 0, representing the program exited in a normal fashion. The program ends when the user chooses to exit via the menu.

This list of functions is completely exhaustive. You may not implement *any* additional functions for *any* reason. You also may *not* choose to leave out any number of the required functions for any reason.

Special Testing Requirement

In order to allow the course staff to easily test parts of your implementation using our own main() function in place of yours, some special preprocessor directives will be required. Don't worry about what all this means right now – it should not affect anything that you do or how your program acts, but is useful for course staff during grading. Immediately before the definition of the main() function, (but after ***all*** other prior source code), include the following lines EXACTLY:

```
#ifdef ANDREW_TEST
#include "andrewTest.h"
#else
```

and immediately following the main() function, include the following line EXACTLY:

```
#endif
```

Therefore, your source code should look as follows:

```
library includes
program header
constant declarations and initializations
global function prototypes with comments
#ifdef ANDREW_TEST
#include "andrewTest.h"
#else
int main()
{
  implementation of main function
}
#endif
```

```
global function definitions
```

Lines above in red are to be used exactly as shown. Other lines simply represent the location of those items within the source code file.

Additional Information

- The program must exit "gracefully", by reaching the ONE return statement that should be the last statement in the main function - do not use the "exit()" function or additional “return”s in main.
- Any floating point values you need should be declared of type "double" Do NOT use any "float" variables in this program.
- You don’t need to worry about number formatting. Don’t use setprecision or anything similar. Just let cout print the values the way it wants to.
- The only library you may #include is <iostream>. No other header files may be included, and you may not make any call to any function in any other library (even if your IDE allows you to call the function without #include'ing the appropriate header file). Be careful not to use any function located in <cmath>.
- Input and/or output should only be done where it is specified. See the function descriptions above to determine which functions can do input and/or output.
- When implementing your solution, use only topics that we’ve discussed in lecture by the date that the project was posted, or that course staff has explicitly allowed in writing. Experienced programmers may identify ways to solve problems in the project using more advanced topics that had not been discussed in lecture by the date the project was posted, but you may not utilize those techniques!
- Remember, “magic numbers” are bad and need to be avoided
- Remember, “duplicated code” is bad and needs to be avoided
- Remember, identifier naming is important. Name your variables, constants, etc., using a descriptive name using the style described in lecture
- Remember, you must ensure your program builds successfully using the c++98 standard specifically
- Remember, you must run valgrind to ensure it does not identify any issues in the execution of your program

Error Checking

You need not worry about error checking the type of the user input in this program. You **may** assume that the user will enter data of the correct data type when prompted (in other words, when the program expects the user to enter an integer value, we will only enter integer values, etc.). Note: This is usually a horrible assumption. As you learn error handling techniques in this class, you will be required to use them, but for this project, just concentrate on implementing the functions as described. During your testing, if you accidentally input a value with an incorrect type, your program will likely go into an infinite loop and print a ton of stuff to the screen. That is “normal” and we will learn how to overcome that later in the course. If this happens to you during your testing, just press “Control-c” to kill the

execution of your project. Please note that it might take a while for the "Control-c" to take effect after you issue it.

You are required to perform error check on certain values that the user enters as input. For example, the user is prompted for a menu choice for the main menu. We promise that we will only enter integer values at that prompt, but the integer value we enter may not be a valid menu option, and your program needs to recognize that situation. Similarly, when providing a range in X, two floating point values are requested (x min and x max for the range). Again, we will input floating values as expected, but you need to perform some error checking to ensure that the x min value is less than the x max value. Finally, the number of rectangles must be a positive value, and your program needs to check for that as well. The posted sample output 3 shows examples of these situations.

Design and Implementation Details

As mentioned above, for this project, you are required to implement the project *exactly* as described here using the *exact* output strings, including punctuation, provided in the sample outputs. Make sure you implement and utilize the exact functions specified and do not add any additional functions, parameters, etc. You will have more "freedom" in your design and implementation as the course progresses. This effectively means that you do not have any control over the general program design for this project.

It is strongly recommended that you implement this program in a piece-wise fashion. That is, start with the printMenu() function, which should be the simplest one to implement. Write a main() to go along which calls the function to see that it works as expected (i.e. write a "driver program" for the sole purpose of testing your printMenu function). Test your program at this point. Once this is all working, add the loop structure to main so it exits the program appropriately. Note: if following this suggested implementation, you have not yet written any of the other functions that will be called by main, you are simply setting up a framework for the program and making sure it works as you go. Once the looping structure is added, run your program and type in user options to ensure your program keeps looping until you specify the option to quit.

Once that is fully working implement another function (or, if the function is relatively complex, implement a part of a function) and then test the new functionality. Continue implementing and testing little bits of additional functionality at a time until the project is completed. To help you get used to this, I've described the functions in the order I would recommend implementing them.