EECS402, Winter 2013, Project 1

Overview:
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 a few different algorithms 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.

All necessary input will come from the user, via the keyboard. The complete specifications for this project are given below.

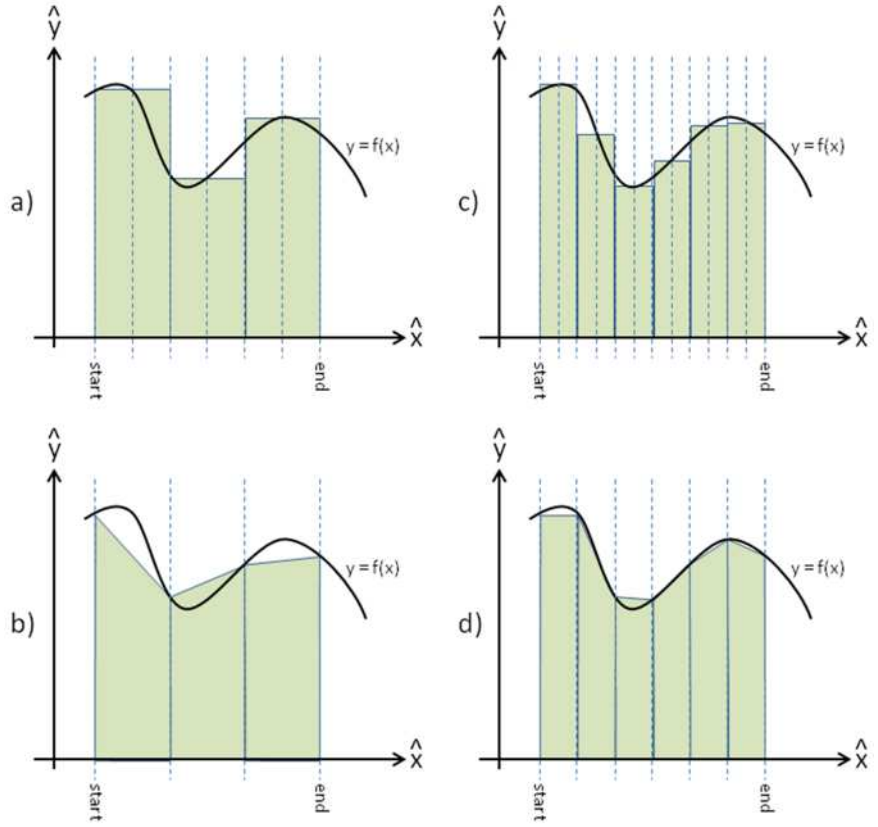
Due Date and Submitting:
This project is due on **Wednesday, January 30 at 5:30pm**. Late submissions will not be accepted for any reason.

For this project, you must submit your project in a single file. The file submitted must be named by appending the project number to your uniqname. For example, my file would be "morgana1.cpp".

Detailed 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 and trapezoids. 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 (and trapezoid as well) is determined by taking into account the X-interval of the area of interest and the number of rectangles (or trapezoids) 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 first two choices the user of your program will have is to determine the approximate area between a cubic function and the X-axis using either some number of rectangles or trapezoids. The user must enter 4 coefficients for a cubic equation, the x start and end values for the area of interest' interval, and the number of polygons to be fit in 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 polygons are needed in general for a good approximation, you will allow the user two more choices. In these cases, 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 (or trapezoids) were needed to achieve that tolerance. Since the user may provide an incorrect "correct answer" you should stop trying to achieve the specified tolerance after trying 100 polygons, and report that you were unable to reach the specified tolerance using the maximum number of polygons.

Global Functions:
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.

double toThePower(double val, 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.

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 at the end of the spec. No input is done in this function!

double approximateAreaWithRect(double aCoeff, double bCoeff, double cCoeff, double dCoeff, double startX, double endX, int num);
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 "num" 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 4.6. The next rectangle would span X=(4.0 + 1.2 = 5.2) to (5.2 + 1.2 = 6.4) with a midpoint of 5.8. (etc) The last rectangle would span X=8.8 to 10.0 with a midpoint of 9.4.

double approximateAreaWithTrap(double aCoeff, double bCoeff, double cCoeff, double dCoeff, double startX, double endX, int num);
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.

Trapezoids are used to approximate the area. The trapezoid's width will be determined by considering the interval of the area of interest and number of trapezoids that are to be used. The height of each side of the trapezoid is determined by evaluating the function at the trapezoid's minimum x value and maximum x value. The area of interest's interval is from "startX" to "endX", and "num" trapezoids should be used to do the approximation. No input or output is done from this function! The design of this function (looping mechanism, etc.) should be the same as for approximateAreaWithRect, as described above.

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.

Additional Specifications / 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.

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 computing a trapezoid's area, there's an odd case where one side of the trapezoid will end up above the X axis and the other side ends up below. This doesn't really form a normal looking trapezoid. You should NOT do anything special to handle this case. Instead, always use this formula for the area of a trapezoid, regardless of the signs of yVal1 and yVal2 (the values of the function at the startX and stopX of the trapezoid):

```
trapArea = (1.0 / 2.0) * xInterval * (yVal1 + yVal2)
```

We certainly could come up with a better way to handle this, but we're trying to keep this project simple.

Error Checking Requirements:

You need not worry about user input error checking in this program. You may assume that the user will enter data of the correct data type when prompted. 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.

Design and Implementation Details:

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 and requests a choice from the user. 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.

Next, implement ONE of the required functions. Once you have it all typed in (or even just a tiny portion that you want to test), save it, and run your program (the menu is already in place from above). Run the program numerous times, testing the new function you just implemented with MANY different inputs that test the different cases that might come up. Once you have exhaustively tested that function and are satisfied with the results, start implementing the next function, and so on.

After you have implemented and tested all the functions, run your program several more times, re-testing all the functions. Note that this final set of test cases can be fairly brief since you have already tested the

functions and were happy with the results. It is always good to do a final test which checks the functionality to make sure everything is still in place as expected.

"Specific Specifications"
These "specific specifications" are meant to state whether or not something is allowed. A "no" means you definitely may NOT use that item. In general, you can assume that you should not be using anything that has not yet been covered in lecture (as of the first posting of the project).

- Use of Goto: No
- Global Variables / Objects: No
- Global Functions: Yes (as specified only)
- Use of Friend Functions: No
- Use of Friend Classes: No
- Use of Structs: No
- Use of Classes: No
- Public Data In Classes: No
- Use of Inheritance: No
- Use of Polymorphism: No
- Use of Arrays: No
- Use of C++ "string" Type: No
- Use of C-Strings: No
- Use of Pointers: No
- Use of STL Containers: No
- Use of User-Defined Header Files: No
- Use of Multiple Source Code Files: No
- Use of Makefile: No
- Use of exit(): No
- Use of overloaded operators: No
- Use of float type: No
- Use of double type: Yes

Sample Program Output:
Sample outputs are provided on the course website that will allow you to see what is expected and to compare your program's output to. Important Note: The sample outputs show you what values are expected for a very limited number of cases. If your program produces the exact same output, that does NOT mean that you are finished. There are many cases that were not tested in the provided output, such as curves that extend below the x-axis, etc. Since computers differ slightly, you may notice your output values look very slightly different from mine. As long as the difference is extremely minimal, this will not be a problem (and in fact, will likely match exactly when I run your program on my computer) – please do not try to "correct" this by forcing your outputs to have a specified number of digits of precision, etc.