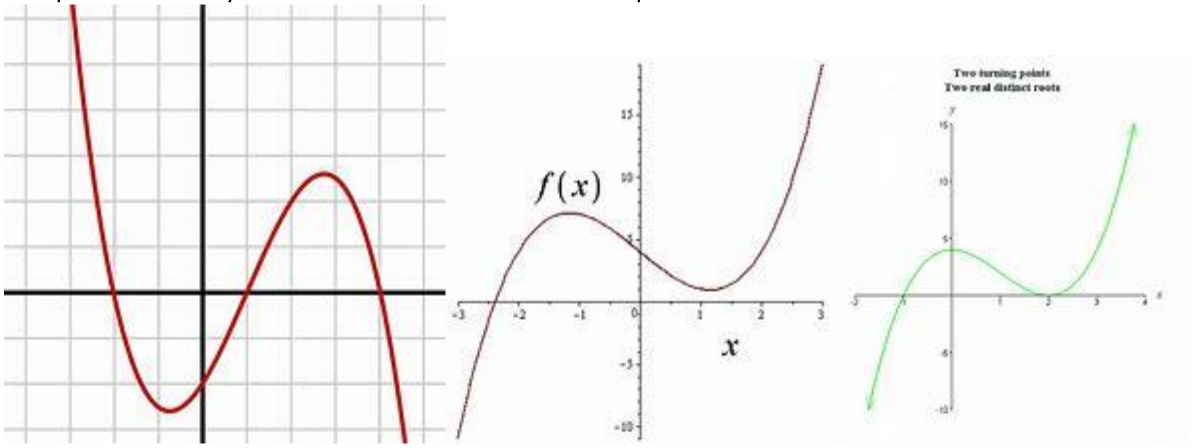
ENGR 102
Lab and Assignment #6

Activity #1: Root Finding – To be done as a team

As a team, you are to write a program that will find a root of a cubic polynomial. A cubic polynomial is of the form:

$$f(x) = Ax^3 + Bx^2 + Cx + D$$

A root of the polynomial is a value, x, such that *f(x)*=0. For a generic cubic polynomial, there will be one local maximum, and one local minimum, and the curve will go off to negative infinity in one direction, and positive infinity in another. Here are three examples:



The polynomial has some number of real roots: the points at which the curve crosses the x-axis. A cubic curve will have either 3 real roots (as above at left), 1 root (above, middle), or in rare cases 2 roots (as above at right). Note that roots are typically single roots, and for single roots, the curve is negative on one side of the root, positive on the other (a double root results in a tangent to the curve, like that above at right).

You should write a program that takes in the coefficients of the polynomial: A, B, C, and D, along with a bound on one single root: *a*,*b*. The user should be expected to input *a* and *b* such that *a*<*b* and there is exactly one single root of the polynomial between *a* and *b*. You should report the value of the root, accurate to within 10^{-6}.

Before developing your code, your team should come up with at least 4 test cases. Test cases should indicate a curve (the 4 coefficients for the curve), as well as a large starting bound on the root (values of *a* and *b*, with *b* >= *a*+1). Your code should include documentation of these 4 (or more) test cases in comments, near the top of the program (after your standard header).

To do this, note that you can use the following procedure, known as bisection. It is a form of what is more generally known as binary search:

- Because the root is a single root, either *f(a)* or *f(b)* will be positive, and the other will be negative.
- You can generate the midpoint of the interval from *a* to *b*, by finding the value halfway between *a* and *b*. Call this point *p*.
- If you evaluate *f(p)*, it will be either positive or negative (or possibly, be the root itself, if *f(p)*=0).

- This will enable you to narrow the interval to either [*a*,*p*] or to [*p*,*b*].
- You can continue doing this until your interval is less than 10^{-6} wide.

As a team, you should create a program that performs bisection to determine the root. More specifically, your program should do the following:

- Read in the coefficients of the polynomial from the user
- Read in the upper and lower bounds around a single root of the polynomial
- Determine the value of that root to within 10^{-6}
- Print the result of that root finding, as a single number
- Print out how many iterations it took to find that root.

Be sure to include comments in your code.

Challenge: Cubic polynomials always have one root that can be found through bisection. There also are ways to find maximum and minimum bounds on the sizes of real roots of polynomials (you will need to research this). Modify your program so that it takes in only the coefficients of the polynomial, and reports one of the real roots to within 10^{-6}.

Activity #2: Taking limits to compute derivatives – To be done as a team

In an earlier lab, we observed how we could have a function that is undefined at some value (such as (sin x)/x at the point x=0), but could come arbitrarily close to it by successively evaluating smaller and smaller numbers (i.e. taking a limit). For example, we might evaluate at x=0.1, x=0.01, x=0.001, etc. until we have come very close to the value. Taking limits like this, numerically, is commonly done when functions are too complicated to evaluate analytically. You will write a program to compute a derivative as a numerical limit. This activity has a few parts:

You may reuse code from activity #1 if it is helpful.

a) Evaluating a polynomial limit analytically
   You should have learned by now the process for finding the derivative of a polynomial (as another polynomial). Write a program that will read in from the user a cubic polynomial *f(x)* (as a set of 4 coefficients), and use this to compute the derivative polynomial (i.e. compute the three coefficients of the derivative *f'(x)*). Then, read in a value for *x* from a user, and evaluate the derivative polynomial at that x. Print out that value.

b) Evaluating a polynomial derivative numerically
   For a function *f(x)*, the derivative of the function at a value *x* can be found by evaluating $\frac{f(x+a)-f(x)}{a}$ and finding the limit as *a* gets closer and closer to 0. Using the same polynomial as the user entered in part (a), and for the same value of *x* as entered in part (a), compute the limit numerically. That is, start with an estimate by evaluating $\frac{f(x+a)-f(x)}{a}$ using a value for *a* such as 0.1. Then, repeatedly halve the value of a until the difference between successive evaluations of $\frac{f(x+a)-f(x)}{a}$ is less than some small value, such as 10^{-6}. Print the result, along with the number of evaluations it took. Calculate how close that result is to the actual answer, computed in part (a).

Challenge: Derivatives can also be estimated by computing the limit $\frac{f(x)-f(x-a)}{a}$ or $\frac{f(x+a)-f(x-a)}{2a}$. Try computing each of those, and calculate how many iterations you need to converge to the limit. Do you get different results with any of them, or does any of them take fewer steps to get an answer?

c) Evaluating a more complex function.
In your own code, come up with four more complex functions (not a polynomial – e.g. use sin/cos/tan/exp/log/powers/etc.), that you do not know how to compute the derivative for analytically, but that you can evaluate. For each function, using the same process as in part (b), calculate the derivative of that function at some value. Write a line of output describing each function, and stating what the computed derivative for it is, along with the number of steps needed to compute the derivative.

Be sure to include appropriate comments in your code, and to use descriptive input and output statements.