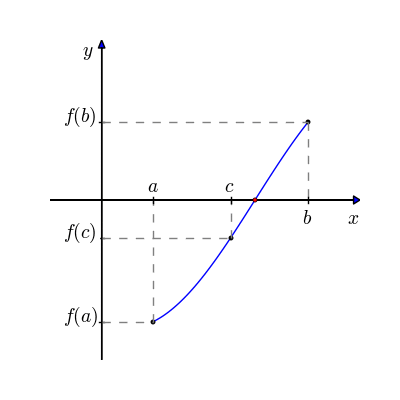
**MAS2806 -Assignment 2**

Ewan Walker 200818067

Graphical user interface, text, application

Description automatically generated**c) The Secant method**

The Secant method basically finds Xn using the previous term (Xn-1) and deducts the function of this previous term multiplied by the difference between the previous two terms over the difference between the previous two functions in the sequence. This method is repeated in a loop, setting a new Xn with each iteration until a near-enough estimate of the root is found (the new iterance is so like the previous that it is a negligible difference) leaving that Xn is almost equal to Xn-1. *Note: the function breaks if the difference of the functions of the two previous terms is 0, resulting in a zero-division error. What this means is that the two outputs are identical so the loop can no longer continue.*

**Comparing with Bisection method:**

The bisection method evaluates the midpoint ((b-a)/2) between two points (a and b) on the x-axis. The function evaluated at this midpoint produces a positive/negative output; then repeat the process using this midpoint and either the upper/lower (respectively) bound as the new bounds (i.e. b and xmid / a and xmid) until the difference between the two points (a and b) is sufficiently close to zero. The last midpoint is the root estimation.

The bisection method is like the secant method in that they both iterate using two previous values of x and the loop continues until the difference between these two values is extremely close to 0.

This differs to the secant method because the bisection method does not iterate using a function of the previous terms where the secant method does include this. In addition, the bisection method has no risk of zero-division errors.

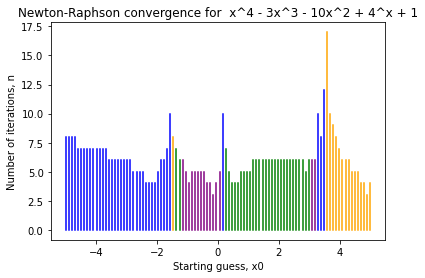
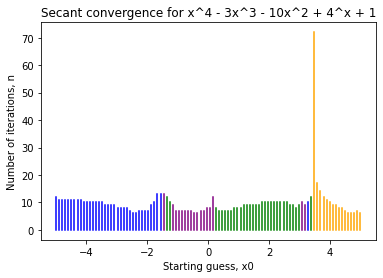
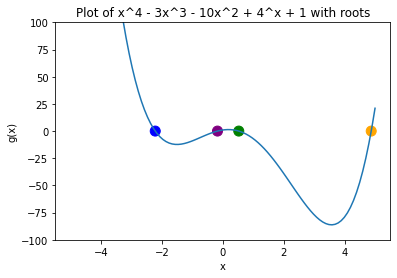
Graphical user interface, text

Description automatically generated**Comparing with Newton-Raphson method:**

The Newton-Raphson method loops the previous x value (Xn-1) and deducts the function evaluated at this previous x value over the derivative of the function evaluated at this same previous x value. Once the function of the previous x value is close enough to 0, then the deduction is negligible, leaving that the Xn is essentially equal to Xn-1. At this point the loop breaks and the last Xn is the root. *Note: This method considers that the local min/max (where the gradient evaluated at that point is 0) may not cross the axis or may touch the axis exactly. In the event of this occurring, the loop breaks with a zero-division error. The function must be smooth everywhere.*

The Newton-Raphson method is similar to the secant method as the iterations both start with the previous x term (Xn-1) and then deduct off some calculated multiple of the function of the Xn-1 (i.e. they are both proportional to the function of the previous term). This also leads them to both have potential zero-division errors. This method is more similar to the N-R than the bisection method.

However, this method differs to the N-R method in the sense that it includes a term of a derivative of the function, where the N-R method does not. It also only used the previous term, where the N-R uses two consecutive previous terms.

**f) Plotting**

**g) The hybrid method**

My plan to find the entire set of what is asked is as follows:

1. Using N-R method, find out which initial guess is not closest to the nearest root and record the initial root in an array.
2. Using S method, find out which initial guess is not closest to the nearest root and record the initial root in an array.
3. Compare the arrays and create a new array, consisting of the all the elements that are in both these arrays ((1) and (2)).
4. Use the hybrid method to create an array of the roots using the elements from the (3) array.
5. For each element in the array, run 2 loops. In one, set the lower to be the relevant x0 and the upper to be 5, and in the other set the upper to be the relevant x0 and the lower -5. Record the number of iterations until the root is found for each loop.
6. The lower number of iterations finds the closest available root fastest.

However, the question only asks for a single example. Thus, I can look at the very nicely colour coordinated plots above to see coherent inconsistencies. Quite clearly, there is an out-of-place purple/blue region just before the massive spike (between 2 and 4). So, this means that I am looking for negative roots in the arrays amongst a sea of positive roots. I quickly print the two arrays of roots to see that these intersect between elements roots[80] and roots[82] (for a set of 100 linearly spaced x0s). These are where x0 = 3.0808, 3.1818 and 3.2828.

I conduct step 5 and 6 in the entire set plan above.

Graphical user interface, text

Description automatically generatedBelow is the code asked for, for the hybrid method:

Graphical user interface

Description automatically generated

The code returns the closest positive value for the **x0 = 3.0808**, and it is the correct root of 4.87451013…

Thus, the bounds of [3.0808,5] returns the correct answer and is the closest answer as [-5, 3.0808] does not produce a result within time, meaning that it iterates seemingly indefinitely.