Name: Jaime Arana-Rochel

3 Methods for Approximating the Root of a Function

Overview

This paper will outline three numerical methods for approximating the real root of a function. These methods are the Regula Falsi Method (also known as the False Position Method), the Birge-Vieta Method, and the Polynomial Deflation Method (also known as the Ruffini-Horner's Method). These three methods all seek to improve upon simpler iteration techniques such as the Bisection, Newton, and Secant methods.

For the purpose of clarity and confidence in the code, each of the techniques will be tested using the function

$$f(x) = x^3 - 8x^2 + x + 42 = (x+2)(x-3)(x-7)$$

which has roots: $x = -2$, $x = 3$, $x = 7$.

I. False Position Method

In the context of a root finding algorithm, the False Position Method is a mixture of the Bisection method and the Secant method. Specifically, it makes the Secant method a bracketing method to ensure that the algorithm converges.

It achieves this in the following way. We take our two initial guesses x_0 and x_1 as in the Secant method, but instead of calculating the derivative using these guesses we check if $f(x_0)$ and $f(x_1)$ have opposite signs. We do this to ensure that our initial guesses bracket the root ($[x_0, x_1]$) by virtue of the intermediate value theorem. Next, we compute our estimated root x_2 in the same way as in the Secant method

$$x_2 = x_1 - f(x_1)\frac{(x_1 - x_0)}{f(x_1) - f(x_0)}$$

Finally, we use the same technique as in the bisection method and choose our next brackets. If $f(x_1)f(x_2)$ is negative, then $[x_1, x_2]$ brackets the root, otherwise $[x_0, x_2]$ brackets the root and we repeat the process with these new guesses.

For example, taking the function defined in the overview section and choosing our bracket to be $[0.0, 5.0]$ the process gives us the following iterations:
(1): $x_0 = 0.0$, $x_1 = 5.0$, $x_2 = 3.0$
(2): $x_0 = 0.0$, $x_1 = 3.0$, $x_2 = 3.0$
which tells us that a root for this function is $x = 3.0$.

The motivation for this technique is to overcome the fact that the Newton and Secant algorithms might produce approximations which diverge away from the real root if our initial guess is not close enough to the root. Those two methods are not bracket methods because there is no guarantee that the two successive approximations will bracket the root. Trying, for example, to calculate the root of

$$f(x) = tan(\pi x) - 6$$

with $x_0 = 0$, $x_1 = 0.48$ as our guesses with the Secant method will lead to diverging estimates with each iteration. This is proven in the provided code. However, the False Position method guarantees that we will converge to a root because we choose our brackets at every iteration, and it does this at a small computational expense. Using the False Position Method, we correctly approximate the real root of the above equation, $x = 0.44743154$.
Of course, that does mean that we would have to have some knowledge of the shape of the function ahead of time in order to properly choose our initial guesses. Without making an educated guess, it might be difficult to choose two initial x_i such that the algorithm converges rapidly and without resorting to choosing a wide bracket. In that respect, none of the other methods presented in this paper have this requirement.

II. Birge-Vieta Method

A major requirement for many algorithms such as the Secant, Bisection, and False Position methods is the evaluation of $f(x)$ at successive approximations x_k. Others like Newton's method requires evaluation of the derivatives of $f(x)$ at each iteration. The Birge-Vieta Method is a way to avoid having to calculate both $f(x)$ and $f'(x)$ at some x explicitly from their equations.

If f(x) is a polynomial $P_n(x)$, which is often the case in many practical situations, then there is an extremely simple technique, known as Horner's Method, to compute $P_n(x)$ and $P'_n(x)$ at $x = x_0$.

Horner's Method

Given a polynomial

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

and our value at which to evaluate the polynomial $x = x_0$, we need to compute $P_n(x_0)$ and $P'_n(x_0)$. We can rewrite $P_n(x)$ as

$$P_n(x) = (x - x_0)Q_{n-1}(x) + b_0$$

where $Q_{n-1}(x) = b_nx^{n-1} + b_{n-1}x^{n-2} + \cdots + b_2x + b_1$.

Therefore, $P_n(x_0) = b_0$.

This is clear once we see that the term $(x - x_0)$ becomes zero in the rewritten equation after we substitute in x_0 for x. We can recursively calculate the coefficients b_k in $Q_{n-1}(x)$ to get the value b_0 that we want. Comparing the coefficients of like powers of x from both sides of

$$a_0 + a_1x + a_2x^2 + \cdots + a_nx^n = (x - x_0)(b_nx^{n-1} + b_{n-1}x^{n-2} + \cdots + b_2x + b_1) + b_0$$

we obtain

$$b_n = a_n$$

$$b_{n-1} - b_nx_0 = a_{n-1}$$

$$\Rightarrow b_{n-1} = a_{n-1} + b_nx_0$$

$$\cdots$$

and so on. Comparing all the coefficients, we see that $b_k - b_{k+1}x_0 = a_k$. From this, we get our recursive relation

$$b_k = a_k + b_{k+1}x_0 \qquad k = n-1\,, n-2\,, \ldots, 1, 0$$

Therefore, knowing the coefficients of our polynomial $P_n(x)$, we can calculate the coefficients of $Q_{n-1}(x)$ recursively, starting with $b_n = a_n$, until we get $b_0 = P_n(x_0)$.

In a similar fashion, we can calculate $P'_n(x_0)$ using the same technique but instead using the b_k coefficients and calculating a separate sequence of coefficients c_k.

Birge-Vieta Algorithm

Using Horner's Method, we can compute the value of our function at a given point knowing only the coefficients of the polynomial. The Birge-Vieta algorithm then is simply a combination of the Newton method and Horner's Method for evaluating polynomials. Using the same Newton iteration function for finding a root of $f(x) = 0$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

we substitute $f(x)$ for our polynomial $P_n(x)$

$$x_{k+1} = x_k - \frac{P_n(x_k)}{P_n'(x_k)} = x_k - \frac{b_0}{c_1}$$

As described, Horner's method lets us evaluate the polynomial: $P_n(x_k) = b_0$, $P_n'(x_k) = c_1$.

The largest advantage of this method is that we avoid having to calculate a polynomial and its derivative at each iteration explicitly from their equations. If we have a long function with many terms raised to extremely high powers, we risk not being able to properly calculate the function and its derivative due to computer value overflow and rounding. This method sidesteps that by calculating the value using the coefficients, doing only n multiplications and n additions to evaluate any n^{th} degree polynomial. This method also converges rather quickly because are using the same techniques as the Newton method.

Note however that we cannot use Birge-Vieta for functions that are not polynomials. In that case, we need to resort back to other methods which do not have this requirement. Further, if the polynomial is simple enough with relatively few terms raised to small powers, then there is not much of a computational advantage in using this method. In such cases, the Newton and Secant methods will do just as well if not better. Also, because Birge-Vieta is not a bracketing method, we again run the risk of not converging upon a root. So this method is really only for those situations in which we have to find roots of complex polynomials.

III. Polynomial Deflation (Ruffini-Horner Method)

This method is a technique to compute the other roots of $f(x) = 0$, once an approximate real root or a pair of complex roots are computed. Recall that, if given a polynomial $P_n(x)$,

we can rewrite $P_n(x)$ as

$$P_n(x) = (x - x_0)Q_{n-1}(x) + b_0 = (x - x_0)Q_{n-1}(x) + P_n(x_0)$$

Then, if $x = x_0$ is an approximate root of $P(x) = 0$, we can write

$$P_n(x) \approx (x - x_0)Q_{n-1}(x)$$

where $Q_{n-1}(x)$ is a polynomial after we factor out $(x - x_0)$ from our equation. This new equation is referred to as the "deflated polynomial". The coefficients of $Q_{n-1}(x)$ can be calculated using the synthetic division technique developed by Ruffini and implemented by Horner's method. We can then use a root finding method to calculate the approximate real root, $x = x_1$ of this new polynomial $Q_{n-1}(x)$. Applying this algorithm repeatedly we get

$$P_n(x) = (x - x_0)(x - x_1)\dots(x - x_{k-1})Q_k(x)$$

We keep synthetically dividing our original polynomial $P_n(x)$ using our estimated roots until we have calculated all possible real roots.

We see that this methods allows us to find all of the real roots of a polynomial without having to repeatedly make initial guesses as to where they might be. The method is also fairly quick if we use the Newton method to evaluate the subsequent polynomials created after the synthetic divisions. The difficulty is that our choice of root finding method will compound errors if applied repetitively. The larger our k, the more poorly will x_{k+1}, which is a root of the deflated polynomial $Q_k(x) = 0$, will approximate a root of $P(x) = 0$. There is also the fact that this method only works for functions which happen to be polynomials. More complex equations will require using some of the other root finding methods with repeated initial guesses to calculate roots.