Numerical Methods of Finding Roots

CSC 301 - Scientific Programming

May 22, 2019

Abtahi Chowdhury

Professor Erik Grimmelmann

Introduction

A common problem in numerical analysis is the problem of finding the roots of a given function, f(x). Formally, the roots of a given function f(x) are the values of x such that f(x) = 0. However, finding these values of x can become more complicated as the function itself becomes more complicated. For example, the roots of a polynomial of degree 2 in the form $ax^2+bx+c=0$ can be found using the formula:

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

Another example would be that the roots of a polynomial of degree 3 in the form $ax^3+bx^2+cx+d=0$ can be found using the formula shown below. [1]

$$x = \sqrt[3]{\left(\frac{-b^3}{27a^3} + \frac{bc}{6a^2} - \frac{d}{2a}\right) + \sqrt{\left(\frac{-b^3}{27a^3} + \frac{bc}{6a^2} - \frac{d}{2a}\right)^2 + \left(\frac{c}{3a} - \frac{b^2}{9a^2}\right)^3}}$$

$$+ \sqrt[3]{\left(\frac{-b^3}{27a^3} + \frac{bc}{6a^2} - \frac{d}{2a}\right) - \sqrt{\left(\frac{-b^3}{27a^3} + \frac{bc}{6a^2} - \frac{d}{2a}\right)^2 + \left(\frac{c}{3a} - \frac{b^2}{9a^2}\right)^3}} - \frac{b}{3a}$$

These are two examples of analytical methods of finding the root of function and as shown, an increase of 1 in the degree of the polynomial can exponentially increase the difficulty of the problem. In fact, it was proven by Abel and Galois that it is impossible to find the roots of a polynomial of degree 5 or higher analytically. This is known as the Abel-Ruffini theorem which states the following:

Let $n \geq 5$. Then there exist $a_0, \ldots, a^{n-1}$ in $\mathbb{C}$ such that no root in $\mathbb{C}$ of the equation $x^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$ can be obtained from $\{0, 1, a_0, \ldots, a_{n-1}\}$, in a finite number of steps, using the operations $+, -, \cdot$ and $/$, and $()^{1/r}$ (with choice) for all $r \geq 1$ in $\mathbb{Z}$. [2]

Due to this fact, the roots of many functions can only be found numerically, making many methods for finding their roots iterative.

Applications of Numerical Root Finding

Numerical root finding can be applied to many different fields of science and engineering, such as physics, chemistry, aerospace engineering, etc. The following are a few common applications of numerical root finding.

Finding Eigenvalues

One example of a situation in which the use of a numerical method to find the roots of a function is needed is finding the eigenvalues of a matrix. When finding the eigenvalues of a n-by-n matrix, the resulting function after taking the determinant will be a function of degree n. If n happens to be greater than 4, then because of the Abel-Ruffini theorem, it is required to use a numerical method to solve for the eigenvalues.

Optimization

Another example of a situation we would need to the find the roots of a function would be in optimization problems. Many of us have seen these types of questions, usually in calculus

one, but those tend to be simpler, where the function is at most a polynomial of degree two. However, in more complex situations where the function has degree 5 or higher, or possibly even be a logarithmic or trigonometric function, you will need to resort to a numerical method. In this cause, we must use a numerical root finding method to find the answer to the question.

Projectile and Orbital Motion

Another example in which finding the roots of an equation is necessary is in projectile motion. A very common physics problem is finding the position of where a ball lands after it was thrown at some angle θ. In most cases, the ball follows a parabolic motion and the position can be found analytically. However, if we scale this up to orbital motion, we no longer have a simple parabolic function, but a trigonometric function. This is seen when wanting to find the x and y coordinates of a planet orbiting a star. These can be found using the following formulas: [3]

$$x = a(cos(E) - e)$$

$$y = asin(E)\sqrt{1 - e^2}$$

where a is the semi-major axis, e is the eccentricity of the ellipse, and E is the eccentric anomaly. However, to calculate this, we must first calculate E using Kepler's equation of motion: [3]

$$M = E - esin(E)$$

where M is the mean anomaly. Using this, we get the trigonometric equation:

$$M - E + esin(E) = 0$$

The solution to such an equation can then be found using numerical root finding methods.

Root Finding Methods

When considering various methods to find the roots of an equation, there are two very important constraints to consider. These are the stopping criteria of the algorithm (i.e. when to stop iterating through the loop) and the convergence rate of the algorithm (i.e. how fast will the algorithm converge to the root, if it does converge at all). The latter of the two can usually be found through testing, but the former must be defined into the algorithm and therefore we will define the stopping criteria to be as such:

C_k is a root of the function f(x) if one of the following is satisfied:

1. $\left|f(C_k)\right| \leq \varepsilon$: The value of the function at C_k is within the tolerance.
2. $\frac{|C_{k-1}-C_k|}{|C_k|} \leq \varepsilon$: The rate of change of C_k is less then or equal to the tolerance.
3. The number of iterations exceeded the maximum allowed number of iterations.

With these criteria in mind, we will now begin looking at various root finding methods. For all of our examples, we will be using the polynomial f(x) = (x-3)*(x-1)*(x+1)*(x+3) as our function with known roots at -3, -1, 1, and 3.

Bisection Method

Regarding the Bisection Method

One method to find the roots of an equation would be to use the bisection method. The theory behind it is quite simple, though there is one restriction for the bisection method. Given a

function f(x) and two points a and b, then $f(a)*f(b) \leq 0$. What this means is that either one of f(a) or f(b) is positive while the other is negative because of the fact that the product of a positive and a negative is negative. This also implies that at some point c between a and b, f(c) = 0. The algorithm follows as such:

1. Compute c to be the midpoint of a and b
2. If $|f(c)| \leq \epsilon$, then c is a root of f(x)
3. If $f(c)*f(a) < 0$, then b = c, else a = c
4. Iterate until step 2 is satisfied

Discussion on the Bisection Method

Using this algorithm on f(x) with a being -5 and b being -2, we see in figure 1 that the algorithm is narrowing in on some point between -4 and -2. If we zoom in, we can see in figure 2 that it is narrowing in on a point between -3.2 and -2.8. We also notice that it is bouncing back and forth within that interval. Zooming in further, we see in figure 3 that the algorithm bounces back and forth quite a bit. Eventually, after 15 iterations, it settles on -3.000030517 which is the root of the function within our tolerance of 10^{-5}. In figure 4, we run bisection method with a fixed value of b and various values of a and find that as the range of a and b increases, the number of iterations required also increases but at a logarithmic rate.

Flaws of the Bisection Method

One downside to this method is the requirement of needing two points a and b that satisfy $f(a)*f(b) \leq 0$. In order to combat this, it is possible to write a function, say find_points, which iterates through some interval of f(x) in search of an a and b value what holds the stated property.

However, there are many issues with this. The most obvious being that the root is in an interval not covered by find_points might not contain the root. There is also the case when you have functions such as $f(x) = x^2$, which has a root at x = 0, that will not work with the bisection method. More generally, any function with a positive range will not work with this algorithm because $f(a)*f(b) \leq 0$ will never hold true for any value of a and b.

Newton Raphson Iteration

Regarding the Newton Method

Another method that can be used to find the roots of a function is the newton raphson iteration or the newton method. This method is a form of fixed point iteration, meaning it uses the following general formula:

$$f(x) = x - g(x) = 0$$

where g(x) is some function such that x-g(x) is 0. In the newton method, this g(x) is then

$$g(x) = x - (f(x) * (f'(x))^{-1}).$$

An important fact about this method is that it also requires an initial guess x for the value of the root. If this guess is *significantly* close to the actual root of the function and it is guaranteed to converge. The algorithm follows as such:

1. Compute $f(x)$ *and* $f'(x)$
2. Compute $x = x - \frac{f(x)}{f'(x)}$
3. If $|f(x)| \leq \varepsilon$ then x is a root of f(x), else iterate

Discussion on the Newton Method

Using this algorithm on f(x) and making our initial guess -5, we can see in figure 5 that it converges towards the root quickly. Looking closer, in figure 6 we observe that unlike the bisection method, there is no bouncing back and forth. Instead, the algorithm moves towards the root quickly in the beginning and as it closes in on the answer, it will slow down. Eventually after 6 interactions, it settles on -3.000079847237657 which is the root of the function within our tolerance of 10^{-5}. Looking at figure 7, we can see that as our initial guess gets farther away from the actual root, the number of iterations also increases in a logarithmic rate. One thing to note however is that for initial guess higher than -10^{8}, the algorithm does not converge.

Flaws of the Newton Method

The newton method has two big issues. The first is one that was observed in testing, which is that if the initial guess isn't close enough to the actual root, then the algorithm will not converge. Although in the testing, the range of proximity to the root is about 10^{8}, which in most practical situations shouldn't be an issue, that number might change for different functions. The other flaw has to do with the fact that the process requires division. This means this algorithm will not work if in any iteration x is a critical point of f(x). In our example, if we choose our initial guess to be 0, the algorithm will throw a divide by zero error.

Muller Method

Regarding the Muller Method

Another method that can be used to find the roots of a function is the Muller method. What makes this method unique is that given a function f(x) and three real initial guesses x_0, x_1, x_2, this method can converge towards complex roots as well as real root. These three guesses are used to create a parabola and the quadratic formula is then used on it to find the x values to use for the next iteration. This algorithm converges very fast, however it is only when the initial guesses are near the root. Otherwise, the algorithm has a high probability of diverging. The algorithm follows as such:

1. Compute $f(x_0)$,$f(x_1)$,$f(x_2)$, and set $c = f(x_2)$
2. Compute $u = x_0 - x_2$, $v = x_1 - x_2$, $y = f(x_0) - c$, and $z = f(x_1) - c$
3. Compute $a = \frac{uz - yu}{uv(v-u)}$ and $b = \frac{yv^2 - u^2 z}{vu(v-u)}$
4. Compute $x_3 = x_2 + \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$
5. If $\left| \frac{x_3 - x_2}{x_2} \right| < \varepsilon$ then x_3 is a root of f(x)
6. Set $x_0=x_1$, $x_1=x_2$, $x_2=x_3$
7. Iterate until step 4 is satisfied

Discussion on Muller Method

Using the algorithm on f(x) and having the initial guesses to be -8, -7, and -9, we can see in figure 8 that the algorithm does converge to the root x=-1. Looking at figure 9, we see that

with each iteration, a complex number is generated. However the complex portion of the number does converge towards 0, meaning that the given function has a real root. In fact the does converge to the root -1.000000011687454 - 5.982693679519721e-9i in 12 iterations. If we test this algorithm on a function that has complex roots, say $f(x) = x^4+x^2+5$, we can see in in figures 10 and 11 that the algorithm does converge towards a complex number root 0.931683416590672 - 1.1696298511710068i in 9 iterations. One thing to note in figure 12 is that the algorithm converges really quickly when the guesses are close to the roots, however when the range of the guesses exceed 2e4, the algorithm no longer converges.

Flaws of the Muller Method

There are a couple issues with the Muller method. The first most prominent one is that the algorithm doesn't converge at all if the guesses are not within a certain range. This range is seen from the testing to be very small, 2e4 in the cause when f(x)=(x-3)*(x-1)*(x+1)*(x+3). The other issue is that it normally requires a 2x2 matrix to be solved. However, this isn't a big issue as it can be avoided by using the formulas mentioned earlier as a substitute.

Conclusion

Out of the three methods test, each has its uses given a certain circumstance. The bisection method will always converge (slowly) regardless of the initial range as long as it meets the criteria of f(a)*f(b)<0. It is useful when the general intervals of when a function is positive and negative are known. The newton method converges much faster than the bisection method, but it is computationally expensive due to having the need to calculate the derivative at each

iteration. The initial guess also needs to be relatively close to the root, or else it will not converge. This algorithm is useful when the general location of the root is known. The muller method converges even faster than the newton method, however the initial guesses need to be much closer to the roots than the newton method in order to converge. However the main application of this method is that it can converge towards complex roots which is something that the other two algorithms cannot do. Overall, all three of these algorithms have their own uses given the situation.

Bibliography

[1] E. Schechter, *The Cubic Formula*. [Online]. Available at: https://math.vanderbilt.edu/schectex/courses/cubic/. [Accessed: 14-Apr-2019].

[2] Edixhoven, B. (2013). *Galois theory and the Abel-Ruffini theorem*. [PDF] p.2. Available at: http://www.math.leidenuniv.nl/~edix/talks/2013/indonesia/yogya/workshop.pdf [Accessed 14 Apr. 2019].

[3] Klioner, S. (2011). *Lecture Notes on Basic Celestial Mechanics*. [PDF] p.18. Available at: https://arxiv.org/pdf/1609.00915.pdf [Accessed 24 Apr. 2019].