Comments on Solving Nonlinear Systems in One Variable

Dennis C. Martin

March 2023

1 Introduction

Computers and humans indeed share a symbiotic relationship. The human brain, through the process of evolutionary refinement, has become exceedingly effective at solving trivial symbolic systems, to the extent that many systems can be solved almost immediately.

However, the human brain is not well-suited for the symbolic computation of very complex linear systems, and that is where the utility of the computer is shown. A computer, which can perform massive sets of basic calculations very efficiently, is well-suited for the process of forming numerical solutions to a system.

The difference between the symbolic and numerical solutions of a system is effectively a question of tolerance. A symbolic solution works by manipulating the representative variables of a problem based on certain axiomatic constraints to produce a symbolic solution. A good example of a trivial symbolic solution is to consider the derivative of the function $2x^2$, which the human brain can solve immediately as being $4x$. However, for more complicated functions, performing actions such as finding the root on a given interval becomes much more difficult, and this is where numerical solutions are important.

Numerical solutions, which are calculated by some sort of algorithm implemented by the computer, find a solution to a given problem to a specific tolerance. The tolerance may be on the error of the solution relative to a known exact *symbolic* solution, or on some residual value of a function. No matter what the tolerance is measured upon, the important fact is that numerical solutions are never perfectly accurate, insofar as they are incapable of generating an exact solution, the way that a symbolic method does. However, in the case of many problems in Computer Science and Numerical Analysis, this is an acceptable tradeoff for the ability to generate some sort of solution to a complex problem.

2 Root-Finding

A common problem in mathematics is that of finding the root of a function. A root is defined, in layman's terms, as the point such that

$$f(x) = 0 \tag{1}$$

meaning, visually, the point on a graph where a function crosses the x-axis. Symbolically, it is possible to compute the roots of many functions realtively easily, such as through the process of factoring, but for complex functions, the use of factoring or other such symbolic methods may not be easily available. For these problems, the use of numerical solutions is possible. There are several methods that can yield numerical solutions, but first, a brief digression must be made to explain the Intermediate Value Theorem, an important tool in the use of numerical solutions for root finding.

2.1 Intermediate Value Theorem

The Intermediate Value Theorem, also known as IVT, is a crucial compenent of many numerical root finding methods. The IVT gives us a useful definition for whether or not it is possible to find the root of a function on a given interval. The theorem is based on the corrolary known as Bolzano's theorem, which states that if a continuous function has values of opposite sign inside an interval, it is possible to find the root of a function. Defined more formally, the IVT states that

$$min(f(a), f(b)) < u < max(f(a), f(b)) \tag{2}$$

In practice, we can implement IVT in numerical methods to determine if a method could even produce the root of a function at all. The following Python code demonstrates this.

```
if( f(a) * f(b) > 0):
    print("Error: f(a) * f(b) must be negative")
```

The above script is designed such that it will fail if IVT is not possible, which is to say that the product of $f(a)$ and $f(b)$ must be less than zero in order for a root to exist. By implementing IVT, we can design numerical methods that more efficiently solve roots, by adding conditions to the methods that ignore unsolvable problems.

2.2 Bisection

One of the most basic methods of determining the root of a function is the Bisection method. This method works by constricting an interval until the interval itself converges to the root of a function. In Python, Bisection works as follows:

```
for k in range(k_max):
    center = (a + b) / 2
    f_middle = f(center)
    f_left = f(a)

    if(f_middle * f_left > 0):
        a = center
    else:
        b = center
```