Using Genetic Algorithms to Find Roots

Sherman Kettner
University of Colorado Colorado Springs
1420 Austin Bluffs Pkwy, Colorado Springs, CO 80918 USA
skettner@uccs.edu

Abstract

Root finding is a fundamental problem in mathematics with applications spanning science, engineering, and technology. Traditional numerical methods like the Bisection and Newton-Raphson techniques are widely used due to their simplicity and efficiency, but they have limitations when applied to complex or nonlinear systems. This paper explores the use of genetic algorithms (GAs) as an alternative approach for root finding. A genetic algorithm is implemented and compared against the Bisection and Newton-Raphson methods to assess its accuracy, computational efficiency, and scalability. The study demonstrates that while GAs require significantly more computational resources for single-root solutions, they excel in finding multiple roots or solving systems of nonlinear equations. Additionally, combining GAs with Newton-Raphson offers a hybrid method that leverages the strengths of both techniques. The results highlight the complementary role of genetic algorithms in addressing scenarios where traditional methods face challenges, making them a valuable addition to the toolbox for solving mathematical equations.

1 Introduction

Solving for roots is a fundamental concept in mathematics, forming the basis for countless applications in science, engineering, and technology. While many methods exist for finding roots, none are universally perfect; each method has its limitations in terms of applicability, efficiency, or accuracy. This variability sparked my curiosity about whether machine learning techniques, particularly genetic algorithms, could be effectively used to solve these problems.

To explore this, I will implement a genetic algorithm for root finding and compare its performance against two traditional methods: the Bisection method, known for its simplicity, and the Newton-Raphson method, regarded as one of the most efficient and widely-used techniques in numerical analysis. By analyzing the results of my implementation, I aim to assess the potential of genetic algorithms in this domain. Additionally, I will review existing literature to determine whether there is a consensus on the practicality and effectiveness of using genetic algorithms for root finding. Through this study, I hope to provide insights into whether this approach is a valuable alternative for solving roots.

2 Relevant Background

To implement a genetic algorithm for root finding, it is essential to first understand what a genetic algorithm is. One of the foundational works in this field is John H. Holland's Holland [1992], which introduced the concept of genetic algorithms. Holland described these algorithms as computational simulations of natural selection, where "individuals" evolve to solve problems through processes analogous to reproduction and mutation in biological systems.

In a computational context, an individual can be represented as a string (e.g., a binary sequence or a set of parameters) that encodes a potential solution to the problem. The algorithm iterates through generations, with each generation consisting of the following steps:

1. **Evaluation:** Each individual in the population is evaluated for fitness, which measures its performance in solving the problem.

2. **Selection and Crossover:** The fittest individuals are selected to "mate," combining parts of their strings to produce offspring. The idea is to mix traits from different parents to create an even more successful generation.

3. **Mutation:** Random alterations are introduced to the offspring strings, promoting diversity and allowing exploration of new solutions.

Since Holland's early work, research has significantly expanded our understanding of genetic algorithms. Numerous refinements and variations have been developed to improve efficiency and adaptability, as detailed in contemporary studies, such as *Modern Variants of Genetic Algorithms* Katoch et al. [2021]. While these advancements are valuable, this paper focuses on the foundational approach outlined above.

To contextualize the effectiveness of genetic algorithms for root finding, we compare them to two classical methods: the Bisection method and the Newton-Raphson method. A brief overview of these methods is provided below:

- **Bisection Method:** This technique repeatedly divides an interval containing the root into halves. If a sign change is detected in one half (indicating the presence of a root), that subinterval becomes the new search interval. This process continues until the interval is sufficiently small.Chapra [2021]

- **Newton-Raphson Method:** Starting from an initial guess, the method iteratively refines the estimate of the root. At each step, the tangent to the function at the current estimate is drawn, and its intersection with the x-axis provides the next estimate. The process is repeated until convergence.Chapra [2021]

3 Methodology

Now that we understand what a genetic algorithm is, how can it be used to solve for a root? The main customization in applying genetic algorithms lies in defining the evaluation (fitness) function.

In this case, the fitness function is defined as:

$$Fitness = \frac{1}{1 + |functionvalue|}$$

This ensures that solutions closer to a root (where the function value is near zero) are assigned higher fitness values.

While the general structure of genetic algorithms is similar across applications, the specific details can vary. Below is an overview of the implementation used in this study:

###Inputs:

- **function**: The target function whose root is to be found.
- **populationSize**: The number of individuals in each generation. Larger populations require more resources but may improve convergence.
- **crossoverRate**: The proportion of genetic material exchanged between parents during crossover (default: 0.7).
- **mutationRate**: The probability of mutation occurring in an individual (default: 0.01).
- **maxGenerations**: The maximum number of generations before the algorithm terminates.
- **minX, maxX**: The range of values for initializing and evaluating the population.
- **tolerance**: The acceptable error threshold for determining convergence.

Algorithm Workflow: The genetic algorithm consists of the following main steps:

1. **Initialization:** Create an initial population by randomly selecting values within the range $[minX, maxX]$.
2. **Evaluation:** Compute the fitness of each individual in the population.
3. **Selection:** Retain the top 50% of the population based on fitness.
4. **Crossover:** For each pair of parents, create two children by combining their genetic material with a random weighting based on the **crossoverRate**.
5. **Mutation:** Introduce random changes to the genetic material of some individuals, with the probability determined by the **mutationRate**.
6. **Termination:** Check if a solution within the specified **tolerance** has been found or if the maximum number of generations has been reached. If neither condition is met, return to step 2.

The best solution from the final generation is returned if no exact root is found.

Pseudocode: Here is the pseudocode for the implementation:

```
function GeneticAlgorithm(function, populationSize, crossoverRate,
        mutationRate, maxGenerations, minX, maxX, tolerance):
    Initialize population with random values between minX and maxX
    for generation in 1 to maxGenerations:
        Evaluate fitness of each individual
        if bestFitness >= (1 / (1 + tolerance)):
```

```
            return bestIndividual
        Select top 50% of the population
        Create newPopulation with:
            Best individual (elitism)
            for remaining individuals:
                parent1, parent2 = Randomly select two parents
                child1, child2 = Crossover(parent1, parent2, crossoverRate)
                Mutate(child1, mutationRate, minX, maxX)
                Mutate(child2, mutationRate, minX, maxX)
                Add child1 and child2 to newPopulation
        population = newPopulation
    return bestIndividual from final generation

function Crossover(parent1, parent2, crossoverRate):
    alpha = Random(0, crossoverRate)
    child1 = alpha * parent1 + (1 - alpha) * parent2
    child2 = (1 - alpha) * parent1 + alpha * parent2
    return child1, child2

function Mutate(individual, mutationRate, minX, maxX):
    if Random(0, 1) < mutationRate:
        mutationAmount = Random(-0.01 * (maxX - minX), 0.01 * (maxX - minX))
        individual += mutationAmount
```

Key Details:

- **Initialization:** The population is initialized with random values within $[minX, maxX]$.
- **Selection:** The top 50% of the population is retained for breeding.
- **Crossover:** Children are created by interpolating between two parents using a random weighting factor (**alpha**) derived from the **crossoverRate**.
- **Mutation:** Random adjustments are applied to some individuals to maintain genetic diversity.

By following this workflow, the algorithm iteratively refines its population toward a solution that meets the specified tolerance.

4 Experiment Design

To test the GeneticAlgorithm I had ChatGPT generate a bunch of function to test it with:

```
\item \textbf{Original Polynomial Function:}
\[
f(x) = 0.0074x^4 - 0.284x^3 + 3.355x^2 - 12.183x + 5
\]
```

```
\[
f'(x) = 4 \cdot 0.0074x^3 - 3 \cdot 0.284x^2 + 2 \cdot 3.355x - 12.183
\]

\item \textbf{High-Degree Polynomial:}
\[
f_1(x) = 0.001x^6 - 0.05x^5 + 0.5x^4 - 2x^3 + 3x^2 - x + 10
\]
\[
f_1'(x) = 0.006x^5 - 0.25x^4 + 2x^3 - 6x^2 + 6x - 1
\]

\item \textbf{Trigonometric-Linear Mix:}
\[
f_2(x) = \sin(x) - \frac{x}{10}
\]
\[
f_2'(x) = \cos(x) - 0.1
\]

\item \textbf{Exponential Function:}
\[
f_3(x) = e^x - 5
\]
\[
f_3'(x) = e^x
\]

\item \textbf{Complex Polynomial:}
\[
f_4(x) = x^5 - 5x^4 + 10x^2 - x + 1
\]
\[
f_4'(x) = 5x^4 - 20x^3 + 20x - 1
\]

\item \textbf{Rational Function:}
\[
f_5(x) = \frac{x^3 - 2x + 1}{x^2 + 1}
\]
\[
f_5'(x) = \frac{(3x^2 - 2)(x^2 + 1) - (x^3 - 2x + 1)(2x)}{(x^2 + 1)^2}
\]

\item \textbf{Logistic-Type Function:}
```

```
\[
f_6(x) = \frac{1}{1 + e^{-x}} - 0.5
\]
\[
f_6'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}
\]

\item \textbf{Higher-Degree Polynomial:}
\[
f_7(x) = 0.002x^7 - 0.01x^6 + 0.5x^3 - x + 2
\]
\[
f_7'(x) = 0.014x^6 - 0.06x^5 + 1.5x^2 - 1
\]

\item \textbf{Trig-Polynomial Mix:}
\[
f_8(x) = x\sin(x) - 0.5
\]
\[
f_8'(x) = \sin(x) + x\cos(x)
\]

\item \textbf{Exponential-Polynomial Mix:}
\[
f_9(x) = x^2e^{-x} - 0.1
\]
\[
f_9'(x) = e^{-x}(2x - x^2)
\]

\item \textbf{Logarithmic-Quadratic Mix (x > 0):}
\[
f_{10}(x) = \ln(x) - x^2 + 3
\]
\[
f_{10}'(x) = \frac{1}{x} - 2x
\]

\item \textbf{Flat Derivative Near Zero:}
\[
f(x) = x^3, \quad f'(x) = 3x^2
\]

\item \textbf{Piecewise Function with Discontinuity:}
```

```
\[
f(x) =
\begin{cases}
  -1 & x < 0 \\
   1 & x \geq 0
\end{cases}
\]
\[
f'(x) = 0 \text{ (undefined at } x = 0\text{)}
\]

\item \textbf{Highly Oscillatory Function:}
\[
f(x) = \sin(10x), \quad f'(x) = 10\cos(10x)
\]

\item \textbf{Rational Function with Asymptote:}
\[
f(x) = \frac{1}{x - 1}, \quad f'(x) = -\frac{1}{(x - 1)^2}
\]

\item \textbf{Exponential Plateau Near Zero:}
\[
f(x) = e^{-x} - 0.1, \quad f'(x) = -e^{-x}
\]

\item \textbf{Exponential Decay Function:}
\[
f(x) = e^{-x}, \quad f'(x) = -e^{-x}
\]
```

These functions are then tested in GeneticAlgorithm, Bisection, and NewtonRaphson

5 Results

Taking the data generated from the testing and graphing it produces:

Please refer to figure 1

and

Please refer to figure 2

The graphs illustrate the performance of the Genetic Algorithm (GA), Newton-Raphson, and Bisection methods in terms of error rates and computational time. The first graph shows that both GA and Newton-Raphson achieve error rates close to zero, demonstrating their accuracy in solving for roots. However, GA exhibits significantly longer computational times. In contrast, the Bisection method, while having a higher error rate, remains consistent in computational efficiency across different scenarios. The second graph, a boxplot of errors by method, highlights the variability in

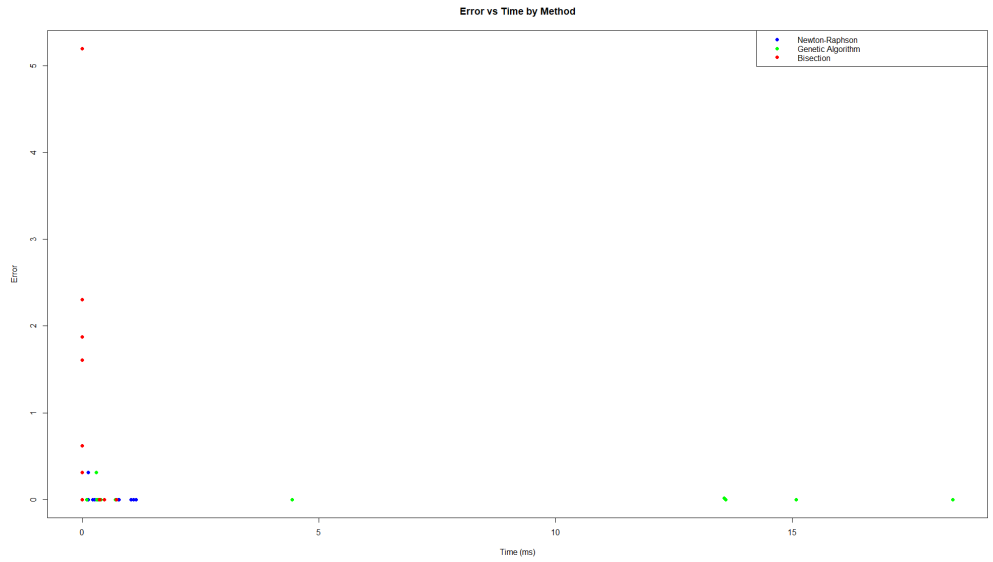

Figure 1: Comparison of Error vs. Time for Newton-Raphson, Genetic Algorithm, and Bisection Methods: Visualizing the trade-offs between accuracy and computational time across different root-finding techniques.

performance among the three techniques. The Bisection method has a wider range of errors, including outliers, compared to the highly consistent error rates of GA and Newton-Raphson, which both cluster near zero. This suggests that while GAs are not always computationally efficient, they maintain accuracy, particularly for complex or nonlinear systems, making them a valuable complement to traditional methods.

6 Discussion

Others have done this in the past with similar results, such as Bhasin, Harsh, and Bhatia, Surbhi article ”Use of Genetic Algorithms for Finding Roots of Algebraic Equations.”Bhasin and Bhatia [2011] Another would be Kuri, Angel in the article, ”Solution of Simultaneous Non-Linear Equations using Genetic Algorithms.” Kuri [2003] While both papers show that it does work, they, however, did not compare their solutions to other methods.

Adding on to that Hassan, Osama Farouk and Jamal, Amani and Abdel-Khalek, Sayed found in there publication, ”Genetic algorithm and numerical methods for solving linear and nonlinear system of equations: a comparative study,” that there are instance where a well set up GA can find solutions where Newton’s Method will fail.Hassan et al. [2020]

Ryuji Koshikawa, Akira Terui, and Masahiko Mikawa paper on ”Solving System of Nonlinear Equations with the Genetic Algorithm and Newton’s Method” Koshikawa et al. [2020], did compare to other methods, most notably Newtons Method, where they found Newtons Method slightly more accurate, and while they do not report the data they found GA much less efficient. However, this paper’s main focus was actually on combining GA and Newton’s Method. By doing so, they achieved even better results than any single application of either.

This finding is backed up in C.L. Karr and Barry Weck and L.M. Freeman work, ”Solutions

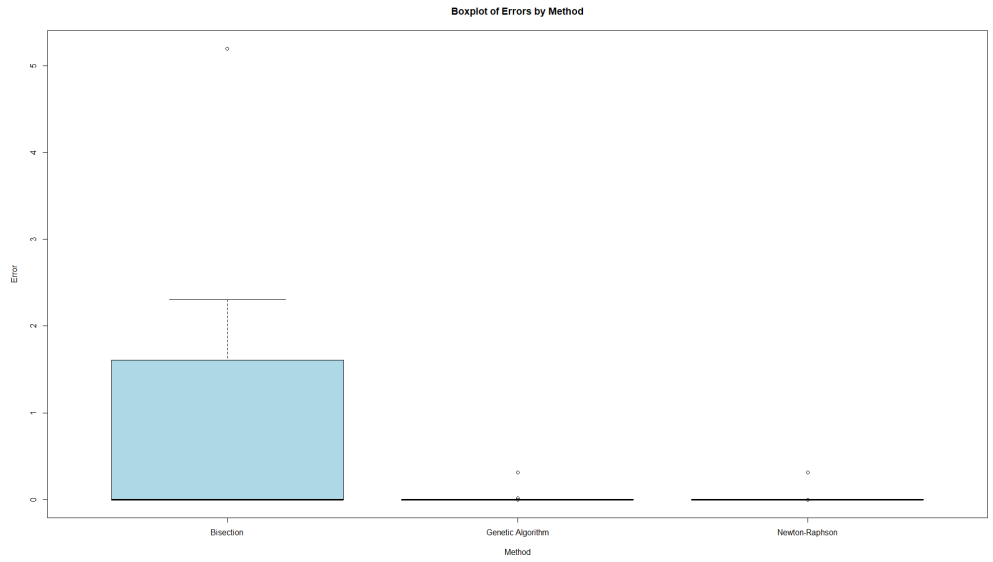

Figure 2: "Boxplot of Errors by Method: Comparing the error distributions of Bisection, Genetic Algorithm, and Newton-Raphson methods, highlighting differences in performance variability and outliers."

to systems of nonlinear equations via a genetic algorithm." Karr et al. [1998] Where once more they find that using GA plus Newton Method reduces the exponential increase in calculation time associated with the number of equations.

However, this is not the only way for GA to be used effectively, as shown in Grosan, Crina, and Abraham, Ajith's article, "A New Approach for Solving Nonlinear Equations Systems." Grosan and Abraham [2008] Here, they show that by looking for multiple answers, GA can be more efficient than all other models. Since, for example, Newton's Method can only solve one solution at a time. However, GA has no such limitations if set up correctly.

Both of these findings add up to make papers like Sašo Karakatič, "Optimizing nonlinear charging times of electric vehicle routing with genetic algorithm," where they are trying to solve the real-world problem of electric vehicle charging plus the routing to achieve this. The best results come from an algorithm that uses GA. Karakatič [2021] Namely a two-layer genetic algorithm with multiple crossovers, which, while outside the scope of this paper to explain, does demonstrate that GA does have a place in solving some equations within real-world situations.

7 Conclusion

In conclusion, genetic algorithms prove to be a viable approach for solving roots, offering accurate results but requiring significantly more computational power when solving for a single root then the more traditional techniques in numerical analysis such as Newton-Raphson. However, their advantages become increasingly evident when tackling multiple roots or systems of nonlinear equations, where traditional methods often face limitations. Moreover, combining genetic algorithms with the Newton-Raphson method can yield the best of both worlds, leveraging the exploratory nature of genetic algorithms and the efficiency of Newton-Raphson.

While genetic algorithms may not always be the most practical choice for root solving, they undoubtedly have a valuable role in cases where traditional methods struggle or when diverse solutions are required. As such, genetic algorithms should be considered a complementary tool, capable of addressing challenges that other root-solving methods might not effectively handle.

References

Harsh Bhasin and Surbhi Bhatia. Use of genetic algorithms for finding roots of algebraic equations. *International Journal of Computer Science and Information Technologies (IJCSIT)*, 2(4):1693–1696, 2011. ISSN 0975-9646. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0125215b5d63be4139f3fc97279e3d8aa3e5c8e6. Accessed: October 27, 2024.

Steven C. Chapra. *Applied Numerical Methods with Python for Engineers and Scientists*. McGraw-Hill Higher Education (US), 2021. Available from: University of Colorado Colorado Springs.

Crina Grosan and Ajith Abraham. A new approach for solving nonlinear equations systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 38(3):698–714, 2008. doi: 10.1109/TSMCA.2008.918599.

Osama Farouk Hassan, Amani Jamal, and Sayed Abdel-Khalek. Genetic algorithm and numerical methods for solving linear and nonlinear system of equations: a comparative study. *Journal of Intelligent & Fuzzy Systems*, 38(3):2867–2872, 2020. doi: 10.3233/JIFS-179572. URL https://web-p-ebscohost-com.uccs.idm.oclc.org/ehost/pdfviewer/pdfviewer?vid=0&sid=f687a827-2f64-4b18-baef-9afa18ca1eb3%40redis. Accessed: October 27, 2024.

John H. Holland. Genetic algorithms. *Scientific American*, 267(1):66–73, 1992. ISSN 00368733, 19467087. URL http://www.jstor.org/stable/24939139.

Sašo Karakatič. Optimizing nonlinear charging times of electric vehicle routing with genetic algorithm. *Expert Systems with Applications*, 164:114039, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2020.114039. URL https://www.sciencedirect.com/science/article/pii/S0957417420308083.

C.L. Karr, Barry Weck, and L.M. Freeman. Solutions to systems of nonlinear equations via a genetic algorithm. *Engineering Applications of Artificial Intelligence*, 11(3):369–375, 1998. ISSN 0952-1976. doi: https://doi.org/10.1016/S0952-1976(97)00067-5. URL https://www.sciencedirect.com/science/article/pii/S0952197697000675.

Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, February 2021. doi: 10.1007/s11042-020-10139-6. URL https://link.springer.com/article/10.1007/s11042-020-10139-6. Accessed: October 27, 2024.

Ryuji Koshikawa, Akira Terui, and Masahiko Mikawa. Solving system of nonlinear equations with the genetic algorithm and newton's method, 2020. URL https://arxiv.org/abs/2007.05159.

Angel Kuri. Solution of simultaneous non-linear equations using genetic algorithms. *WSEAS Transactions on Systems*, 2, 01 2003.