HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

# PROJECT REPORT

## Solving Elementary Algebraic Equation Using Genetic Algorithm

| PHAM QUANG TRUNG | LANG VAN QUY | NGAN VAN THIEN |
|---|---|---|
| 20214935 | 20214928 | 20215647 |

**Course: Project I (IT3910E)**

**Supervisor:** Associate Professor Tran Dinh Khang ________________

Signature

**Department:** Computer Science

**School:** Information and Communication Technology

**HANOI, 12/2023**

# *Solving Elementary Algebraic Equation Using Genetic Algorithm*

## Abstract

This project explores the application of Genetic Algorithms (GAs) for solving elementary algebraic equations. While traditional methods have long been established, GAs provide an innovative and automated approach to finding solutions. The study investigates the effectiveness of GAs in handling a variety of elementary algebraic equations, showcasing their adaptability to different problem structures.

The project outlines the foundational principles of Genetic Algorithms, drawing parallels with natural selection and genetic structures. It delves into the unique perspective offered by GAs in autonomously generating and evolving mathematical expressions to find solutions. The research aims to contribute to the growing body of knowledge on computational intelligence applications in elementary algebra, highlighting the potential of GAs as valuable tools in algebraic problem-solving.

# Acknowledgements

We extend our heartfelt gratitude to all those who played a pivotal role in the realization of this project. Our sincere appreciation goes to Assoc. Prof. Tran Dinh Khang for his unwavering guidance, invaluable insights, and suggestions that significantly enriched our understanding of the Genetic Algorithms domain, contributing immensely to the successful completion of this project.

Special recognition is also due to our fellow teammates, whose enthusiastic contributions and strong collaboration were instrumental in bringing this project to fruition. The collective effort and dedication of everyone involved have been indispensable in achieving commendable outcomes.

With sincere thanks,

# TABLE OF CONTENTS

# 1 Introduction

## 1.1 Problem description

In the realm of mathematical problem-solving, the application of innovative techniques has always been essential in addressing complex challenges. One such approach that has garnered increasing attention in recent years is Genetic Programming (GP). Originally inspired by the principles of natural selection and genetics, GP has found success in various fields, including computer science, optimization, and now, algebraic problem-solving.

This project delves into the application of Genetic Programming as a powerful tool for solving elementary algebraic equations. Elementary algebra, serving as the foundation for more advanced mathematical concepts, often involves the manipulation and simplification of equations to find unknown variables. Traditional methods such as factoring, completing the square, or using the quadratic formula have long been the standard approaches for solving these equations. However, with the advent of computational intelligence techniques like Genetic Programming, a new avenue for efficient and automated problem-solving has emerged.

The fundamental idea behind Genetic Programming is rooted in the principles of evolution. By mimicking the processes of natural selection, crossover, and mutation, Genetic Programming evolves a population of candidate solutions over successive generations. This evolutionary approach allows the algorithm to explore a vast solution space, adapt to the problem at hand, and converge towards an optimal or near-optimal solution.

In the context of solving elementary algebraic equations, Genetic Programming offers a unique perspective. Rather than relying on predefined algorithms or manual intervention, GP can autonomously generate and evolve mathematical expressions to find solutions. This approach proves particularly valuable when dealing with equations of varying complexity, where traditional methods may become cumbersome or impractical.

This project aims to explore the effectiveness of Genetic Programming in solving a range of elementary algebraic equations, showcasing its ability to adapt to different problem structures and provide solutions in an automated fashion.

# 2 The Method

## 2.1 Input Handling

The input from the user is essentially a string representation of an equation. This input serves as the basis for the fitness evaluation, as detailed in the subsequent subsection. The evaluation process comprises two primary phases: first, the analysis of the user's input, followed by the calculation of the output.

### 2.1.1 Shunting Yard Algorithm

The Shunting Yard Algorithm is a powerful and efficient method for parsing mathematical expressions written in infix notation – the conventional way mathematical expressions are written with operators placed between operands. Proposed by Edsger Dijkstra in 1961, this algorithm is particularly useful for converting infix expressions into a format that is easier to evaluate, such as Reverse Polish Notation (RPN).

Here's an overview of how the Shunting Yard Algorithm works:

**Tokenization:** The expression is first broken down into tokens, which are the smallest units of the expression. Tokens include numbers, operators, parentheses, and any other relevant elements.

**Shunting Yard Process:**

1. The algorithm uses two stacks: one for operators and another for output.
2. It iterates through each token from left to right in the infix expression.

**Operator and Operand Handling:**

1. If a token is an operand (a number), it is immediately added to the output queue.
2. If the token is an operator, the algorithm checks the operator stack:
   - If the operator stack is empty, the current operator is pushed onto the stack.
   - If the stack is not empty, and the precedence of the current operator is greater than the operator at the top of the stack, the current operator is pushed onto the stack.
   - If the precedence is equal or lower, operators from the stack are popped and added to the output queue until the conditions are met for pushing the current operator.

**Parentheses Handling:**

1. If a token is an open parenthesis, it is pushed onto the operator stack.
2. If a close parenthesis is encountered, operators are popped from the stack and added to the output queue until an open parenthesis is reached. Both the open parenthesis and close parenthesis are then discarded.

**Output Generation:** After processing all tokens, any remaining operators on the stack are popped and added to the output queue.

**Result:** The output queue now contains the expression in Reverse Polish Notation, which is more amenable to straightforward evaluation using a stack-based approach.

The Shunting Yard Algorithm provides a systematic way to convert infix expressions to postfix notation, making it easier to evaluate expressions and implement calculators. Its efficiency lies in its ability to handle operator precedence and parentheses, ensuring the correct order of operations.

### 2.1.2 Evaluating Reversed Polish Notation

Calculating the output from Reverse Polish Notation (RPN), also known as postfix notation, involves a straightforward and efficient process that doesn't require parentheses or explicit knowledge of operator precedence. In RPN, the operands appear before their respective operators, making the evaluation process more predictable and systematic.

Here's a step-by-step guide to calculating the output from an expression in Reverse Polish Notation:

**Initialize a Stack:** Create an empty stack to hold operands during the evaluation process.

**Scan the Expression from Left to Right:** Iterate through each token in the RPN expression.

**Operand or Operator Handling:**

1. If the token is an operand (a number), push it onto the stack.
2. If the token is an operator, pop the required number of operands from the stack (typically two for binary operators).

**Perform the Operation:**

1. Apply the operator to the popped operands.
2. Push the result back onto the stack.

**Result:** - Once the entire expression is evaluated, the final result will be the only remaining item on the stack.

Example:

Consider the RPN expression: ‘$34 + 5 * 2 -$ ‘

1. Push 3 and 4 onto the stack for ‘3 4 +‘.
2. Pop 4 and 3, add them, and push 7 onto the stack.
3. Push 5 onto the stack for ‘7 5 *‘.
4. Pop 5 and 7, multiply them, and push 35 onto the stack.
5. Push 2 onto the stack for ‘35 2 -‘.
6. Pop 2 and 35, subtract them, and push -33 onto the stack.

The final result is -33, which represents the calculated value of the original expression in RPN.

## 2.2 Genetic Algorithm

Genetic Algorithms (GAs) are adaptive heuristic search algorithms categorized within the broader family of evolutionary algorithms. Grounded in an analogy with the genetic structure and behavior of chromosomes within a population, the foundation of GA is established upon the following principles:

Within a population, individuals engage in competition for resources and opportunities to mate. The success of individuals, often measured by their fitness, determines their likelihood of reproducing and producing a greater number of offspring than their counterparts.

The individuals deemed the fittest through this process engage in mating, leading to the creation of offspring. The genetic information from these fittest parents is passed on to the next generation. Notably, there is a potential for genetic recombination and mutation during this transmission, resulting in offspring that may exhibit improved characteristics compared to either parent.

This iterative cycle, where fitter individuals contribute to the next generation, leads to a continuous refinement of the population. Over successive generations, the collective genetic makeup becomes increasingly adapted to the prevailing environment, fostering an evolutionary progression where the population becomes better suited to its surroundings.

### 2.2.1 Encoding Method

The search space in our context refers to the domain where the population of individuals is situated. Each individual within this population serves as a potential solution to a given equation. In the representation of these solutions, each individual is encoded as a 32-bit binary number. Here, each bit in the binary representation corresponds to a gene within the analogy of a chromosome.

This binary coding scheme allows us to express a wide range of potential solutions within a finite sequence of 32 bits. The combination of these bits forms a unique genetic makeup for each individual, analogous to the genetic information encapsulated in chromosomes. The variation in the binary representation among individuals within the population reflects the diversity of potential solutions explored in the search space.

In the context of our genetic algorithm, a fitness score is assigned to each individual, serving as an indicator of an individual's competitive capability. The objective is to identify individuals with optimal or near-optimal fitness scores.

### 2.2.2 Fitness Evaluation

For our specific case, the fitness score is determined by taking the inverse of the difference between the left-hand side and the right-hand side of a given equation. This approach aligns with the notion that individuals producing solutions where the equation's sides are closer together are more desirable. The higher the fitness score an individual attains, the greater its likelihood of being selected for further genetic operations or reproduction.

In essence, this fitness scoring mechanism directs the genetic algorithm towards prioritizing individuals that exhibit solutions where the equation is better balanced. By favoring individuals with superior fitness scores, the algorithm aims to iteratively refine the population, progressively converging towards solutions that align closely with the desired equation.

### 2.2.3 Mutation Method

In Genetic Algorithms (GA), the bit flip mutation method is a genetic operator used to introduce diversity in the population by randomly altering individual genes. This mutation technique is particularly applied to binary-encoded individuals, where each bit represents a specific gene within the chromosome.

The process of bit flip mutation involves randomly selecting one or more bits within an individual's binary representation and flipping their values. The term "flip" refers to changing a 0 to a 1, or vice versa. The objective is to mimic the occurrence of genetic variations, akin to the natural mutation process in biological evolution.

### 2.2.4 Crossover Method

Single Point Crossover, also known as one-point crossover, is a fundamental genetic operator used in Genetic Algorithms (GAs) for recombining genetic material between two parent individuals to generate offspring. This operator is particularly applied to binary-encoded individuals, where each bit represents a gene in the chromosome.

**Algorithm 1** Genetic Algorithm

```
Initialization: Generate an initial population of individuals
Evaluate the fitness of each individual in the population
while Termination Condition not met do
    Select parents based on fitness
    Apply crossover to create offspring
    Apply mutation to offspring
    Evaluate fitness of new individuals
    Select individuals for the next generation
end while
```

## 3 Evaluation

The developed system has demonstrated effectiveness in successfully finding roots for complex equations, especially those challenging to solve using traditional methods. With a computational time ranging from approximately 100-200 milliseconds for solvable equations, the system showcases relatively fast performance. However, it is important to address certain challenges and limitations observed during testing:

**No Solution Condition:** In instances where the given functions approach 0 at infinity but are not strictly equal to 0, the system may output large values as answers. For example, equations like $1/x = 0$, where $x$ approaches infinity, don't have real roots, yet the system may provide a substantial result. This issue arises due to the current stopping conditions, which focus on search time and improvement only.

**Slow Convergence:** The speed of convergence is highly dependent on the search range. In scenarios where the solution tends towards the upper limit of the search range, the algorithm may take an extended amount of time to locate this solution. This emphasizes the need for careful consideration and optimization of the search range to improve overall convergence speed.

Acknowledging these challenges, future enhancements to the system could involve refining stopping conditions to better handle cases where the solution approaches infinity without reaching 0 and optimizing the algorithm's performance in scenarios with slow convergence. Additionally, exploring strategies to dynamically adjust search ranges based on the characteristics of the equations may contribute to further improving the system's efficiency.

## 4 Conclusion

The application of the Genetic Algorithm has demonstrated effectiveness in addressing single-variable equations. However, extending its utility to multi-variable equations or systems involving vector quantities demands further refinement. To enhance the algorithm's overall performance, particular attention needs to be directed towards optimizing the fitness evaluation and population initialization mechanisms, both of which play pivotal roles in reducing search times.

Moreover, a critical aspect of future improvements lies in refining the conditions governing the scenario of "No Solution." An enhanced and more nuanced treatment of situations where a solution is unattainable can significantly contribute to the algorithm's accuracy and broaden its applicability across diverse problem domains.

Looking ahead, potential advancements could encompass refining the result visualization component, ensuring clearer representation of the obtained solutions. Designing a more user-friendly interface is another avenue for improvement, making the algorithm more accessible to a broader audience and facilitating its adoption in various domains.

Furthermore, adapting the existing framework to handle differential equations is a promising direction for future research. This expansion would not only present a solution to a wider array of mathematical challenges but also open up opportunities for tackling complex and dynamic systems that involve rates of change.

In summary, future work on the Genetic Algorithm should focus on extending its capabilities to handle multi-variable equations, improving the efficiency of fitness evaluation and population initialization, refining the treatment of "No Solution" scenarios, enhancing result visualization, designing a user-friendly interface, and adapting the algorithm to address the complexities posed by differential equations. These advancements collectively aim to elevate the algorithm's versatility and applicability across a spectrum of mathematical problem-solving scenarios.

## References

[1] Eberhart, R., Simpson, P., & Dobbins, R. (1996). *Computational Intelligence PC Tools*. AP Professionals.

[2] Reeb, J., & Leavengood, S. (1998). *Using the graphical method to solve system of linear equations*. Operation Research, October.

[3] Balagurusamy, E. (1999). *Numerical Methods*. Tata McGraw Hill, New Delhi.

[4] Spivak, R. (2015). *Let's Build a Simple Interpreter. Part 7: Abstract Syntax Tree*. Ruslan's Blog, December.

[5] Mastorakis, N. E. (2005). *Solving Non-linear Equations via Genetic Algorithms*. Hellenic Naval Academy, Terma Hatzikyriakou 18539, Piraeus, Greece, June.