Genetic Algorithm Project Report

Course: Artificial Intelligence and Expert Systems
Instructor: Dr. Abdi
Student: Amirhossein Eslami
Semester: Spring 2025

Abstract

This project investigates the application of Genetic Algorithms (GA) to solving systems of linear and nonlinear equations with multiple variables.
The main objective is to design and implement a solver that relies solely on GA concepts, avoiding direct numerical methods.

The implemented process follows these steps:
1. Accept user-defined equations as input.
2. Generate an initial population.
3. Evaluate fitness using an error function.
4. Select parents based on fitness ranking.
5. Apply crossover and mutation operators.
6. Produce successive generations until an acceptable solution is reached.

The project is divided into three stages, addressing systems of two, three, and four variables, respectively. Special attention was given to avoiding local minima and improving convergence speed. Final results were compared with reference examples from the course materials, and several optimization strategies are proposed for future improvement.

Keywords: Genetic Algorithm, Equation Solving, Fitness Function, Crossover, Mutation

Table of Contents

Table of Figures

Chapter 1 : Solving Two-Variable Systems

The first implementation focuses on solving two-variable systems of equations.

- Input Handling: Coefficients for variables (x, y) along with constants (c1, c2) are collected from the user.
- Initial Population: Random chromosomes representing candidate solutions are generated.
- Fitness Function: Each chromosome is evaluated by computing the error of substituted values in the equations.
- Parent Selection: Chromosomes are sorted by fitness, and the top candidates are selected for reproduction.
- Crossover and Mutation:
   * Crossover produces new chromosomes by combining parent genes.
   * Mutation introduces randomness to maintain diversity and avoid premature convergence.
- GA Execution: The algorithm iteratively generates new populations until the solution converges within a predefined accuracy.
- Testing: Several test cases from lecture notes demonstrate that the GA achieved highly accurate solutions.

```python
def get_equation_coefficients() -> tuple[tuple[float, float, float], tuple…:
    print("(a1 * x + b1 * y = c1):")
    a1 = float(x=input(prompt="input a1: "))
    b1 = float(x=input(prompt="input b1: "))
    c1 = float(x=input(prompt="input c1: "))

    print("\n(a2 * x + b2 * y = c2):")
    a2 = float(x=input(prompt="input a2: "))
    b2 = float(x=input(prompt="input b2: "))
    c2 = float(x=input(prompt="input c2: "))

    return (a1, b1, c1), (a2, b2, c2)
```

Figure 1-1 Genetic Algorithm Define Equation's Coefficient

```python
def generate_initial_population(pop_size=100, lower_bound=-100, upper_bound=100) -> list:
    population: list = []
    for _ in range(stop/pop_size):
        x: float = random.uniform(a=lower_bound, b=upper_bound)
        y: float = random.uniform(a=lower_bound, b=upper_bound)
        chromosome: list[float] = [x, y]
        population.append(object/chromosome)
    return population
```

Figure 1-2 Generate Initial Population Function

```python
def fitness(chromosome, eq1, eq2) -> Any:
    x: Any,y: Any = chromosome
    a1: Any, b1: Any, c1: Any = eq1
    a2: Any, b2: Any, c2: Any = eq2

    error1: Any = (a1*x + b1*y) - c1
    error2: Any = (a2*x + b2*y) - c2

    totalSqueredError: Any = error1**2 + error2**2

    return totalSqueredError
```

Figure 1-3 Fitness Function

```python
def select_parents(population, eq1, eq2, num_parents=NUM_PARENTS_SELECTION) -> list:
    population_sorted: list = sorted(iterable/population, key=lambda chromo: fitness(chromosome=chromo, eq1=eq1, eq2=
    return population_sorted[:num_parents]
```

Figure 1-4 Parents Selection Function

```python
def cross_over(parent1, parent2) -> list:
    x: Any = (parent1[0] + parent2[0])/2
    y: Any = (parent1[1] + parent2[1])/2
    return [x,y]
```

Figure 1-5 Cross Over Function

```python
def mutate(chromosome, mutation_rate = MUTATION_RATE) -> Any:

    if(random.random < mutation_rate):
        chromosome[0] += random.uniform(a=MUT_MIN_BOUND, b=MUT_MAX_BOUND)

    if(random.random < mutation_rate):
        chromosome[1] += random.uniform(a=MUT_MIN_BOUND,b=MUT_MAX_BOUND)

    return chromosome
```

Figure 1-6 Mutate Function

```python
def genetic_algorithm(eq1, eq2, generations=1000, pop_size=100) -> Any:
    lower: Any, upper: Any = estimate_bounds(eq1=eq1, eq2=eq2)
    population: list = generate_initial_population(pop_size=pop_size, lower_bound=lower, upper_bound=upper)

    for generation in range(stop/generations):
        population: list = sorted(iterable/population, key=lambda chromo: fitness(chromosome=chromo, eq1=eq1, eq2=eq2))
        best_fitness: Any = fitness(chromosome=population[0], eq1=eq1, eq2=eq2)

        if best_fitness < 1e-6:
            print(f"Solution found in generation {generation}")
            return population[0]

        parents: list = select_parents(population=population, eq1=eq1, eq2=eq2, num_parents=20)

        # Generate new population
        new_population: list = []
        while len(obj/new_population) < pop_size:
            parent1: Any = random.choice(seq=parents)
            parent2: Any = random.choice(seq=parents)
            child: list = crossover(parent1=parent1, parent2=parent2)
            child: list = mutate(chromosome=child)
            new_population.append(object/child)

        population: list = new_population

    print("No exact solution found. Best approximation:")
    return sorted(iterable/population, key=lambda chromo: fitness(chromosome=chromo, eq1=eq1, eq2=eq2))[0]
```

Figure 1-7 Implement Genetic Algorithm

```python
eq1: tuple[float, float, float], eq2: tuple[float, … = get_equation_coefficients()
solution: Any = genetic_algorithm(eq1=eq1, eq2=eq2)
print(f"Approximate solution: x = {solution[0]}, y = {solution[1]}")
```

Figure 1-8 Test The Algorithm

```
(a2 * x + b2 * y = c2):
input a2: 4
input b2: 4
input c2: 12
Solution found in generation 8
Approximate solution: x = 2.0002460012081356, y = 0.9998368463088123
```

Figure 1-9 The Given Input Coefficients and its Outcome

$$\begin{cases} x + 2y = 4 \\ 4x + 4y = 12 \end{cases} \implies \begin{cases} x = 2 \\ y = 1 \end{cases}$$

Figure 1-10 Comparison with Lecture Note Example

Chapter 2 : Solving Three-Variable Systems

In this phase, the solver was extended to handle systems with three unknowns.

- Equation Parsing: A parser was implemented to convert user input strings into mathematical expressions.
- Validation: The parser was tested to ensure correctness of interpretation.
- Population Generation and Fitness Evaluation: Similar to Chapter 1, but adapted for three variables.
- Optimization Challenge: The algorithm occasionally converged to local minima. To address this, hyperparameters such as population size and mutation probability were tuned.
- Results:
  * With proper adjustments, the GA produced solutions close to the expected values.
  * However, in some cases, convergence was slower and accuracy lower compared to the two-variable case.
- Improvement: The parsing function was refined using error handling (try/catch) to avoid crashes when input length was zero.
- Verification: With the improved parser, the GA successfully solved example cases from the course materials.

```
[ 86.97729318  55.52272624 -63.15352858]
[ 89.38986747 -54.7790876   20.51801454]
[ 75.41624617  70.6048285   32.71599405]
[ 10.38029788 -59.1987186   80.97408002]
[ 57.81295752 -83.49314924  21.3168993 ]
[ 53.58523012  51.50790263  88.58781311]
[  8.9172475  -28.59502087  -3.64102061]
[  5.11679739 -94.12217514   1.81424585]]
```

Figure 2-1 Example of Initial Population Created

```
Generation 260: Fitness=0.000398 | Solution=[ 0.66666731 -4.99996553  0.75001814]
Generation 270: Fitness=0.000398 | Solution=[ 0.66666731 -4.99996553  0.75001814]
Generation 280: Fitness=0.000398 | Solution=[ 0.66666731 -4.99996553  0.75001814]

Converged at generation 282

Best Solution Found: [ 0.66666731 -4.99996553  0.75001814] | Fitness: 0.00039753
```

Figure 2-2 Example Execution after Parser Correction

$$\begin{cases} 6x - 2y + 8z = 20 \\ y + 8x \times z = -1 \\ 2z \times \dfrac{6}{x} + \dfrac{3}{2}y = 6 \end{cases} \Longrightarrow \begin{cases} x = \dfrac{2}{3} \\ y = -5 \\ z = \dfrac{3}{4} \end{cases}$$

Figure 2-3 Comparison The Output with Lecture Note Example

Chapter 3 : Solving Four-Variable Systems

The final stage scales the GA approach to four-variable systems.

- Reused Components: The equation parser and GA framework from previous chapters were adapted.
- Modifications:
  * Initial population generation was updated to accommodate four variables.
  * The fitness function and parent selection methods were extended accordingly.
- Crossover and Mutation: Two-point crossover was employed to enhance genetic diversity.
- Challenges and Debugging:
  * Some initial runs failed due to missing variables in the equation set.
  * After revising the code, the GA produced valid outputs.
- Performance:
  * Convergence was slower than in the previous cases due to increased complexity.
  * Nevertheless, the GA approximated correct solutions with reasonable accuracy.
- Validation: Testing against course-provided examples confirmed that the GA was effective, though requiring more generations to converge.

$$
\begin{cases}
\dfrac{1}{15}x - 2y - 15z - \dfrac{4}{5}t = 3 \\
-\dfrac{5}{2}x - \dfrac{9}{4}y + 12z - t = 17 \\
-13x + \dfrac{3}{10}y - 6z - \dfrac{2}{5}t = 17 \\
\dfrac{1}{2}x + 2y + \dfrac{7}{4}z + \dfrac{4}{3}t = -9
\end{cases}
\Longrightarrow
\begin{cases}
x = -\dfrac{3}{2} \\
y = -\dfrac{7}{2} \\
z = \dfrac{1}{3} \\
t = -\dfrac{11}{8}
\end{cases}
$$

Figure 3-1 Test Example from Lecture Notes

```
Generation 450: Fitness=0.071310 | Solution=[-1.50631039 -3.58116331  0.33279771 -1.20937392]
Generation 460: Fitness=0.069184 | Solution=[-1.50631039 -3.57729725  0.33279771 -1.22949652]
Generation 470: Fitness=0.063313 | Solution=[-1.50631039 -3.57140963  0.33279771 -1.22949652]
Generation 480: Fitness=0.063186 | Solution=[-1.50631039 -3.57140963  0.33279771 -1.23321487]
Generation 490: Fitness=0.063186 | Solution=[-1.50631039 -3.57140963  0.33279771 -1.23321487]

Best Solution Found: [-1.50631039 -3.56552839  0.33279771 -1.82891225] | Fitness: 0.06008997
```

Figure 3-2 Final Result for Third Example in Lecture Notes

```
Generation 0: Fitness=127.004842 | Solution=[ 6.16553091 16.16945556 -2.15658104 -2.21036724]
Generation 10: Fitness=19.530861 | Solution=[ -3.38730352 -25.48260198   0.30490369  44.36894754]
Generation 20: Fitness=17.734695 | Solution=[ -3.25552233 -23.58713551   0.30490369  40.0093181 ]
Generation 30: Fitness=15.804868 | Solution=[ -3.00343208 -21.396381     0.30490369  35.63696285]
Generation 40: Fitness=13.963144 | Solution=[ -2.86966231 -19.34966321   0.30490369  31.22518035]
Generation 50: Fitness=12.185051 | Solution=[ -2.71653881 -17.26514116   0.30490369  27.05279947]
Generation 60: Fitness=10.541389 | Solution=[ -2.61584543 -15.18854255   0.30490369  22.92924996]
Generation 70: Fitness=8.915023 | Solution=[ -2.39846992 -13.68131644   0.30490369  18.96829001]
Generation 80: Fitness=7.024927 | Solution=[ -2.178662   -11.51478753   0.30490369  14.99521919]
Generation 90: Fitness=5.675548 | Solution=[-1.96316009 -9.96722624  0.30490369 11.68997507]
Generation 100: Fitness=4.443619 | Solution=[-1.96930911 -8.42904099  0.30490369  8.34435485]
Generation 110: Fitness=3.463299 | Solution=[-1.78012915 -7.4584595   0.30490369  6.3945639 ]
Generation 120: Fitness=2.707981 | Solution=[-1.78012915 -5.98230302  0.30490369  4.3280128 ]
Generation 130: Fitness=2.075754 | Solution=[-1.69943568 -5.86872632  0.33210244  3.46091749]
Generation 140: Fitness=1.791165 | Solution=[-1.69399958 -5.29630913  0.33210244  2.5992575 ]
Generation 150: Fitness=0.972814 | Solution=[-1.59689126 -4.61441552  0.33210244  0.84216495]
Generation 160: Fitness=0.750487 | Solution=[-1.5741883  -4.31151642  0.33210244  0.36968075]
Generation 170: Fitness=0.706840 | Solution=[-1.56917638 -4.31120867  0.33210244  0.25358975]
Generation 180: Fitness=0.637297 | Solution=[-1.56283654 -4.22204948  0.33210244  0.11303156]
Generation 190: Fitness=0.305283 | Solution=[-1.52535303 -3.85114949  0.33210244 -0.69067063]
Generation 200: Fitness=0.230173 | Solution=[-1.52535303 -3.74908846  0.33210244 -0.88105148]
Generation 210: Fitness=0.216269 | Solution=[-1.51886519 -3.74908846  0.33210244 -0.88105148]
Generation 220: Fitness=0.186251 | Solution=[-1.51886519 -3.70867263  0.33210244 -0.94337448]
Generation 230: Fitness=0.177784 | Solution=[-1.51886519 -3.6960978   0.33210244 -0.96940369]
Generation 240: Fitness=0.172078 | Solution=[-1.51682874 -3.6960978   0.33210244 -0.97938776]
Generation 250: Fitness=0.169025 | Solution=[-1.51682874 -3.68805657  0.33210244 -1.01415944]
Generation 260: Fitness=0.166817 | Solution=[-1.51598519 -3.68805657  0.33210244 -0.98607502]
Generation 270: Fitness=0.163448 | Solution=[-1.51598519 -3.6862653   0.33210244 -1.00507325]
Generation 280: Fitness=0.155574 | Solution=[-1.51598519 -3.67575464  0.33279771 -1.01539342]
Generation 290: Fitness=0.150896 | Solution=[-1.51598519 -3.66878945  0.33279771 -1.02942929]
Generation 300: Fitness=0.134583 | Solution=[-1.51316211 -3.647512    0.33279771 -1.09923119]
Generation 310: Fitness=0.121490 | Solution=[-1.51316211 -3.63394698  0.33279771 -1.09923119]
Generation 320: Fitness=0.119422 | Solution=[-1.50841471 -3.63394698  0.33279771 -1.10835896]
Generation 330: Fitness=0.113487 | Solution=[-1.50841471 -3.62511255  0.33279771 -1.11559108]
Generation 340: Fitness=0.095443 | Solution=[-1.50841471 -3.6084463   0.33279771 -1.15239723]
Generation 350: Fitness=0.094143 | Solution=[-1.50841471 -3.6084463   0.33279771 -1.16243583]
Generation 360: Fitness=0.090322 | Solution=[-1.50841471 -3.60146772  0.33279771 -1.16345611]
Generation 370: Fitness=0.085354 | Solution=[-1.50777777 -3.59753991  0.33279771 -1.176509  ]
Generation 380: Fitness=0.081243 | Solution=[-1.50777777 -3.59312712  0.33279771 -1.18984082]
Generation 390: Fitness=0.079363 | Solution=[-1.50777777 -3.58861645  0.33279771 -1.18984082]
Generation 400: Fitness=0.078794 | Solution=[-1.50777777 -3.58861645  0.33279771 -1.19162429]
Generation 410: Fitness=0.077344 | Solution=[-1.50631039 -3.58861645  0.33279771 -1.20232895]
Generation 420: Fitness=0.074585 | Solution=[-1.50631039 -3.58030492  0.33279771 -1.20232895]
Generation 430: Fitness=0.073424 | Solution=[-1.50631039 -3.58435532  0.33279771 -1.20937392]
Generation 440: Fitness=0.073424 | Solution=[-1.50631039 -3.58435532  0.33279771 -1.20937392]
Generation 450: Fitness=0.071310 | Solution=[-1.50631039 -3.58116331  0.33279771 -1.20937392]
```

Figure 3-3 Convergence Path Toward Final SolutionFigure 3-2

Conclusion

This project demonstrated the feasibility of using Genetic Algorithms to solve multi-variable systems of equations without relying on direct numerical methods.
While the GA successfully solved systems of two, three, and four variables, performance issues such as local minima and slow convergence in higher-dimensional cases highlight areas for improvement.

Future Work

- Adaptive mutation rates to reduce premature convergence.
- Hybrid approaches combining GA with classical optimization methods.
- Parallelization of population evaluation to reduce runtime.

References

1. Course Lecture Notes
2. Problem Set Handouts
3. ChatGPT (AI Assistant)