Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

EARIN

Laboratory 3
Group 21
Variant 1
Evolutionary and Genetic Algorithms

Tymon Żarski 310992
Bartosz Peczyński 310703

WARSAW April 11, 2024

Contents

1. Introduction

Assigned task: Write a Python program to solve the Rosenbrock function using a basic genetic algorithm with Tournament Selection.

Initialize x and y from the range [-5,5], where the equation gives the function:

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

Use the Gaussian operator for mutation and random interpolation for crossover (eg: $x_o = \alpha * x_{p1} + (1-\alpha) * x_{p2}, yo = \alpha * y_{p1} + (1-\alpha) * y_{p2}$, where o, p1, p2 refer to offspring, parent 1 and parent 2, and α is a random number; this version of the crossover operator is practically used for real-valued problems). Assume we always replace the current population with the new offspring.

2. Implementation

The implementation we proposed to solve the following tasks is standard and prepared according to the provided template. To provide data handling and statistical operations, we used **pandas** to do the heavy lifting and increase the readability and performance.

The only custom part of the implementation is the plotting functionality. The function which visualises (**compute_and_prepeare_plots**) the performance and evolution of the Genetic Algorithm (GA) over generations. Here's how the code handles plotting and how you can potentially extend it with additional plotting capabilities:

This function is where the GA's evolution is visualized if *should_plot* is set to True. It creates a figure with four subplots, each dedicated to a specific aspect of the GA's performance:

- Best Solutions X: Plots the first component of the best solutions over generations.
- Best Solutions Y: Plots the second component of the best solutions over generations.
- Best Fitness Value: Shows how the best fitness value evolves, using a logarithmic scale for the y-axis to handle wide value ranges.
- Average Fitness Value: Similar to the best fitness value plot, but for average values across the population.

3. Experiments

3.1. Task 1: Finding genetic algorithm parameters

To find the best parameters of the algorithm, first, we have chosen a set of parameters that we thought might be reasonable. Then, we decreased or increased the value of one parameter at a time to see if the algorithm performed better. If this was the case, we tried to change the chosen parameter even further. After no significant change in outcome, we moved to optimising the next parameter. To determine how well the algorithm operates, we have considered the value of best fitness found by the algorithm under test and the number of generations it needs to find this value. Our results can be seen in Table 3.1 or on the plots on Figure 3.1.

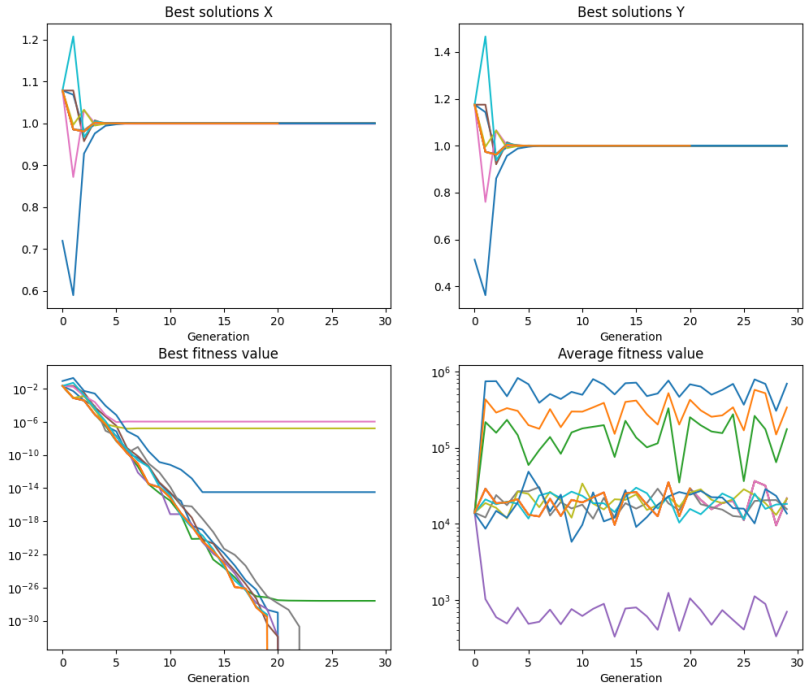

Figure 3.1. Experiments to determine the best parameters

Table 3.1. Findings of the best parameters

	Population size	Mutation rate	Mutation strength	Crossover rate	Number of generations	Tournament size	Seed	Best fitness value	Average fitness value	Best solution
0	1000	0.20	10	0.3	30	30	1	0.000000e+00	690576.824612	(1.0, 1.0)
1	1000	0.10	10	0.3	30	30	1	0.000000e+00	336454.558596	(1.0, 1.0)
2	1000	0.05	10	0.3	30	30	1	2.555909e-28	174959.500184	(1.000000000000016, 1.000000000000032)
3	1000	0.10	5	0.3	30	30	1	0.000000e+00	21667.113819	(1.0, 1.0)
4	1000	0.10	2	0.3	30	30	1	0.000000e+00	704.167197	(1.0, 1.0)
5	1000	0.10	5	0.4	30	30	1	0.000000e+00	21667.113819	(1.0, 1.0)
6	1000	0.10	5	0.2	30	30	1	1.031473e-06	21665.891032	(0.999, 0.998)
7	1000	0.10	5	0.3	30	20	1	0.000000e+00	15631.384229	(1.0, 1.0)
8	1000	0.10	5	0.3	30	40	1	1.578575e-07	21394.763341	(0.9998, 0.9996)
9	2000	0.10	5	0.3	30	30	1	0.000000e+00	18287.026941	(1.0, 1.0)
10	500	0.10	5	0.3	30	30	1	3.255755e-15	13672.260852	(1.0000000545, 1.000000107)
11	1000	0.10	5	0.3	21	30	1	0.000000e+00	29146.441090	(1.0, 1.0)

3.2. Task 2: Randomness in genetic algorithm

This experiment explores the influence of randomness by using different random seeds and population size adjustments on the performance of a genetic algorithm. We experimented by running the algorithm under identical settings, but with different seed values. The aim was to investigate how changes in randomness affect the algorithm's ability to find optimal solutions and the consistency of these solutions across different runs. The parameters used for this evaluation included a population size of **1000**, a mutation rate of **0.1**, a mutation strength of **5**, a crossover rate of **0.3**, **21** generations, and a tournament size of **30**.

The results, as documented in Table 3.2, reveal a significant influence of seed values on the algorithm's performance. Remarkably, for seeds 1 and 291, the algorithm achieved the best fitness value of 0, indicating that it successfully found optimal solutions. The average fitness values varied across different seeds, reflecting the inherent randomness and its effect on the algorithm's search efficiency. We also prepared a plot illustrating the algorithm's operation using different seeds: Figure 3.2.

Table 3.2. Task 2 - different seeds

	Population size	Mutation rate	Mutation strength	Crossover rate	Number of generations	Tournament size	Seed	Best fitness value	Average fitness value	Best solution
0	1000	0.1	5	0.3	21	30	1	0.000000e+00	29146.441090	(1.0, 1.0)
1	1000	0.1	5	0.3	21	30	291	0.000000e+00	15960.319397	(1.0, 1.0)
2	1000	0.1	5	0.3	21	30	1159	2.415887e-30	17910.233567	(0.999, 0.999)
3	1000	0.1	5	0.3	21	30	2605	1.109336e-31	23467.529273	(0.999, 0.999)
4	1000	0.1	5	0.3	21	30	4629	3.867391e-28	22218.647707	(1.0000000000000195, 1.0000000000000389)

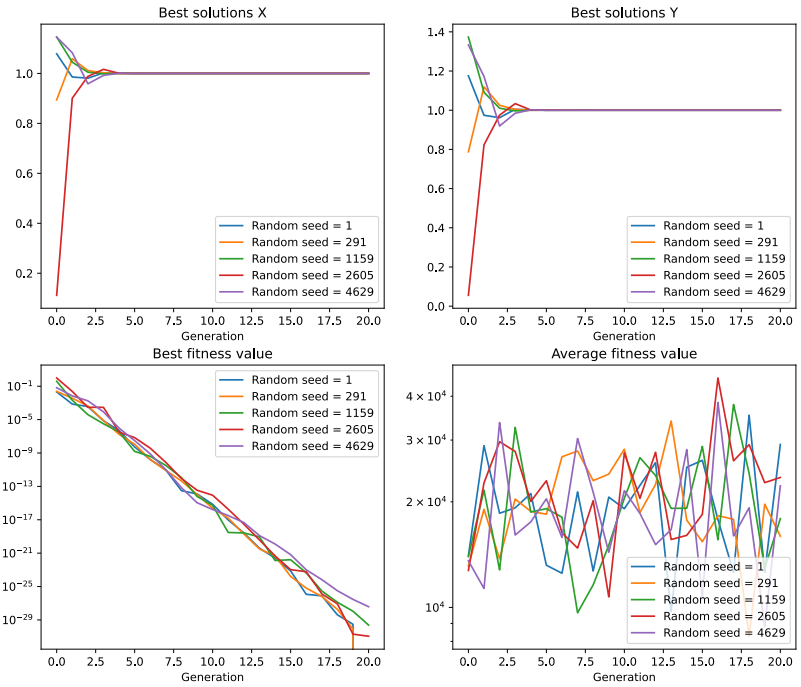

Figure 3.2. Task 2 - influence of seed

Further analysis was conducted to explore the effects of decreasing the population size on the genetic algorithm's efficiency. The same seed (1) was used across all runs to eliminate variability due to randomness and focus solely on the impact of population size. The population sizes tested were 1000 (original size), 500, 250, and 100, with all other parameters remaining constant, which could be seen in the Table 3.3.

In the Table 3.3, we can find the outcomes of this part of the experiment. As the population size decreased, a gradual increase in the best fitness value was observed, from an optimal 0 in the entire population size to 0.7945987 in the smallest population size tested. This indicates a deterioration in the algorithm's ability to find the optimal solution as the population size diminishes. The average fitness values also varied, generally showing an increase with the decrease in population size, suggesting that a smaller population size negatively impacts the algorithm's overall performance and solution accuracy.

Then, we rerun the algorithm with decreasing population size for the seed equal to 1. We found out that for half of the population size, the algorithm still provides quite an accurate solution (fitness value equal to 3.256e-15), but by decreasing this value further, the error increases significantly. This is presented on the Figure 3.3

Table 3.3. Task 2 - Decreasing population

	Population size	Mutation rate	Mutation strength	Crossover rate	Number of generations	Tournament size	Seed	Best fitness value	Average fitness value	Best solution
0	1000	0.1	5	0.3	21	30	1	0.000000e+00	29146.441090	(1.0, 1.0)
1	500	0.1	5	0.3	21	30	1	3.255755e-15	24160.817061	(1.0000000545, 1.000000107)
2	250	0.1	5	0.3	21	30	1	6.918811e-03	21224.292087	(1.083, 1.173)
3	100	0.1	5	0.3	21	30	1	7.945987e-01	4590.207699	(0.153, -0.004)

In the end, we calculated, that the average fitness value across all seeds was around **7.785e-29**, and its standard deviation was **1.727e-28**. Although not that significant (worst fitness value equal to **3.867e-28)**, we can see some overfitting of our parameters, as when using different seeds, the solutions found differ.

3.3. Task 3: Crossover impact

In this experiment, the genetic algorithm was subjected to multiple runs with variations in the crossover rate while keeping other parameters constant. The objective was to observe how changes in the crossover rate affect the algorithm's performance regarding both the best and average fitness across generations. Results were averaged across multiple seed values to ensure reliability. From two parents there are made two children. Values of crossover impact are complementary. For example value 0.1 produces the same children, as the value 0.9, because only the order of those children is changes, proportions stay the same.

Table 3.4. Average values of experiments with different seeds for analyzing crossover impact

	Population size	Mutation rate	Mutation strength	Crossover rate	Number of generations	Tournament size	Seed	Best fitness value	Average fitness value	Best solution - X	Best solution - Y
0	1000.0	0.1	5.0	0.2	21.0	30.0	1737.0	2.905383e-05	21732.151117	0.997415	0.994850
1	1000.0	0.1	5.0	0.3	21.0	30.0	1737.0	7.785318e-29	21740.634207	1.000000	1.000000
2	1000.0	0.1	5.0	0.4	21.0	30.0	1737.0	3.097371e-21	21740.634207	1.000000	1.000000
3	1000.0	0.1	5.0	0.6	21.0	30.0	1737.0	4.996066e-14	21740.634459	1.000000	1.000000
4	1000.0	0.1	5.0	0.9	21.0	30.0	1737.0	1.107729e-03	21827.585699	1.015183	1.031754

Observations:

- **Best Fitness Value**: Across all runs, the best fitness value varied significantly depending on the crossover rate. Crossover rates from the range [0.3, 0.6] resulted in significantly better fitness values

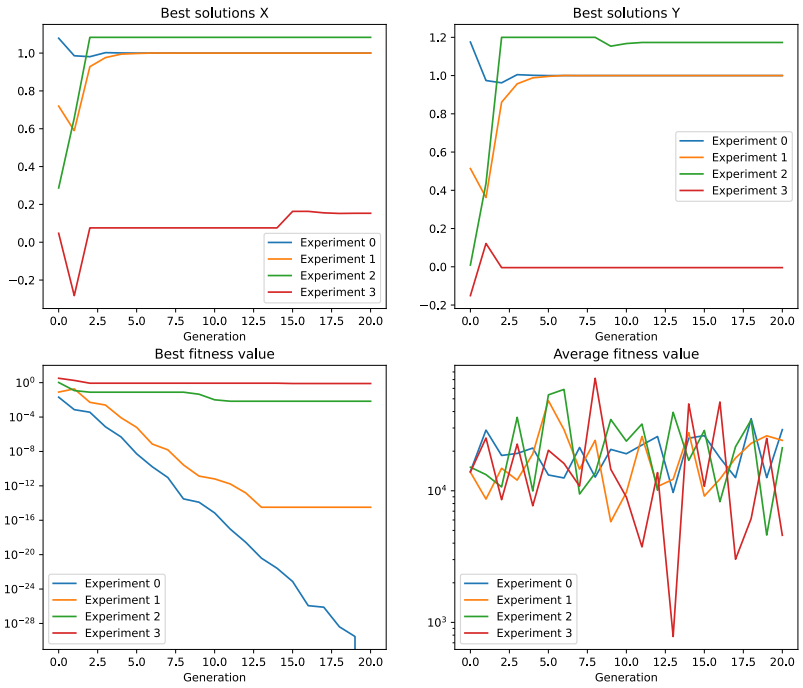

Figure 3.3. Task 2 - influence of decreasing population

than 0.2, 0.9, which produce offspring, in which each individual is very similar to one of the parents, and is only by small extend similar to the other.

- **Average Fitness Value**: Similarly, the average fitness value remained relatively consistent for crossover rates between 0.3 and 0.6, indicating stable performance. However, a noticeable decline was observed at a crossover rate of 0.9.
- **Best Solution**: The best solution (X and Y coordinates) converged to optimal values (close to 1) for crossover rates ranging from 0.3 to 0.6. However, at a crossover rate of 0.9 and 0.2, the best solution deviated from the optimal values, indicating suboptimal convergence.

To conclude, the crossover rate is crucial in determining the genetic algorithm's performance. Moderate crossover rates (0.3 to 0.6) yield optimal results regarding best and average fitness values and convergence to optimal solutions. Crossover rates favoring one parent (0.9, 0.2) may lead to suboptimal convergence and degradation in performance. Further analysis with additional parameters may provide deeper insights into the interaction between crossover rate and other factors in genetic algorithm optimization.

3.4. Task 4: Mutation and the covergence

3.4.1. Mutation rate

In this experiment, the genetic algorithm was executed multiple times with increasing mutation rates while keeping other parameters constant. The objective was to analyze how changes in the mutation rate

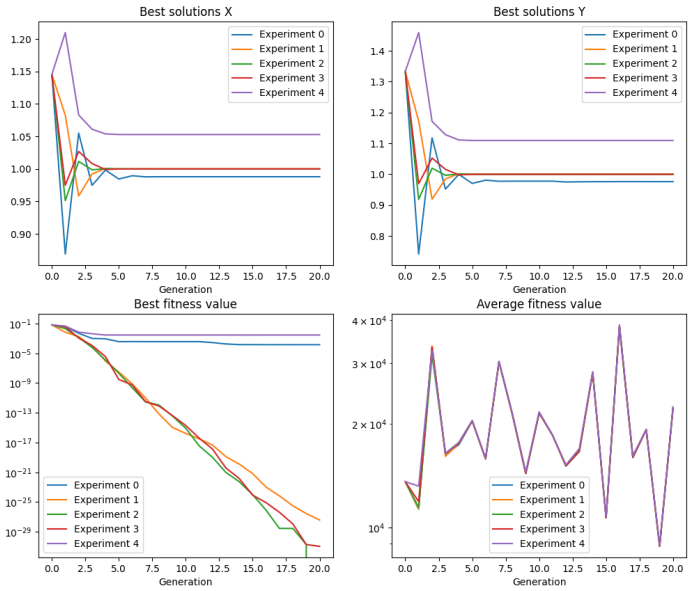

Figure 3.4. Influence of crossover rate - average results between seeds

influence the algorithm's performance, mainly focusing on the best and average fitness across generations. Results were averaged across multiple seed values to minimize the randomness factor.

Table 3.5

	Population size	Mutation rate	Mutation strength	Crossover rate	Number of generations	Tournament size	Seed	Best fitness value	Average fitness value	Best solution - X	Best solution - Y
0	1000.0	0.150	2.0	0.3	30.0	30.0	1737.0	0.000000e+00	860.935269	1.000000	1.000000
1	1000.0	0.225	2.0	0.3	30.0	30.0	1737.0	2.017371e-17	1381.633399	1.000000	1.000000
2	1000.0	0.300	2.0	0.3	30.0	30.0	1737.0	0.000000e+00	1927.994143	1.000000	1.000000
3	1000.0	0.450	2.0	0.3	30.0	30.0	1737.0	0.000000e+00	2936.011362	1.000000	1.000000
4	1000.0	0.300	2.0	0.3	30.0	30.0	1737.0	0.000000e+00	1927.994143	1.000000	1.000000
5	1000.0	0.600	2.0	0.3	30.0	30.0	1737.0	1.878265e-25	3957.210271	1.000000	1.000000
6	1000.0	0.750	2.0	0.3	30.0	30.0	1737.0	1.579988e-04	4951.744222	1.000715	1.002070
7	1000.0	0.900	2.0	0.3	30.0	30.0	1737.0	5.589648e-02	5557.935803	0.979764	1.006109
8	1000.0	1.500	2.0	0.3	30.0	30.0	1737.0	1.176453e-01	6408.659240	1.098484	1.270901

Observations:

- **Best Fitness Value**: Increasing the mutation rate led to significantly varying the best fitness value. Lower mutation rates (0.150 to 0.300) resulted in near-zero best fitness values, indicating better convergence towards optimal solutions. However, at higher mutation rates (0.600 to 1.500), the best fitness values increased, suggesting less effective optimization,
- **Convergence to Optimal Solutions**: The mutation rate has a significant impact on the algorithm's ability to converge to optimal solutions. Lower mutation rates (0.150 to 0.300) resulted in near-zero best fitness values, indicating effective convergence towards optimal solutions. The best solutions consistently approached ideal values equal to 1 for X and Y coordinates, demonstrating high convergence.
- **Effect of Higher Mutation Rates**: As the mutation rate increased (0.600 to 1.500), the algorithm's convergence to optimal solutions became less effective. Best fitness values increased, indicating

a divergence from optimal solutions. Consequently, the best solutions deviated from ideal values, suggesting reduced convergence and suboptimal performance.

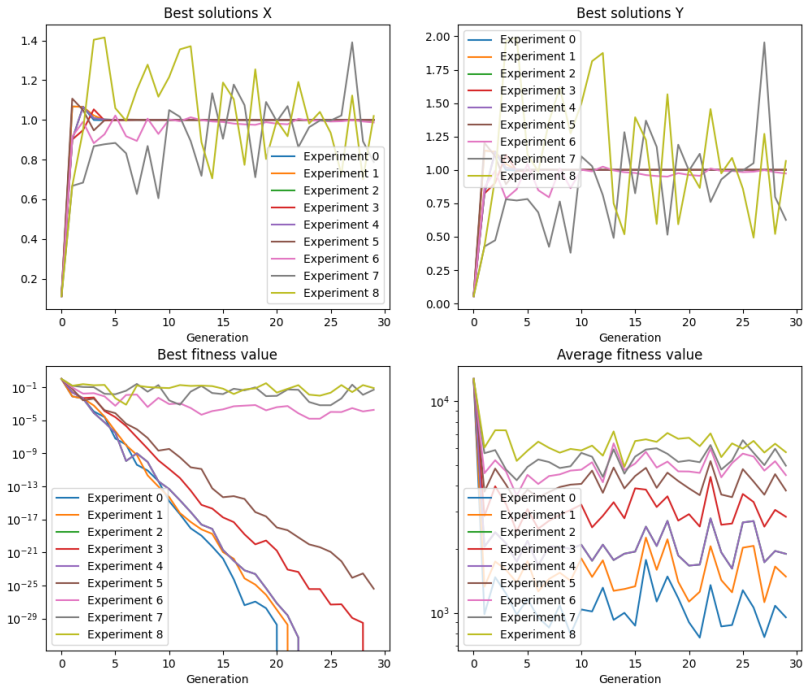

Figure 3.5. Influence of decreasing population mutation rate - average results between seeds

Ultimately, the mutation rate significantly influences the genetic algorithm's performance, particularly its ability to converge to optimal solutions. Lower mutation rates tend to yield better results in terms of both best and average fitness values, leading to more effective optimization. Conversely, higher mutation rates may impede convergence and result in suboptimal solutions. Careful consideration of the mutation rate is crucial for achieving optimal performance in genetic algorithms.

3.4.2. Mutation strength

In this experiment, the genetic algorithm was executed multiple times with increasing mutation strengths while keeping other parameters constant. The primary objective was to analyze how changes in mutation strength affect the algorithm's performance, particularly focusing on the best and average fitness across generations. Results were averaged across multiple seed values to ensure reliability.

1. **Impact on Fitness Values**: The mutation strength has a significant impact on the fitness values of the population. As the mutation strength increases, the best and average fitness values show a notable rise. This suggests that higher mutation strengths allow for greater exploration of the solution space, leading to improved fitness outcomes. However, it's essential to note that the best fitness value remains at 0.0 throughout all runs, indicating convergence to optimal solutions despite increased mutation strength.

Table 3.6

	Population size	Mutation rate	Mutation strength	Crossover rate	Number of generations	Tournament size	Seed	Best fitness value	Average fitness value	Best solution - X	Best solution - Y
0	1000.0	0.3	1.0	0.3	30.0	30.0	1737.0	0.0	2.309768e+02	1.0	1.0
1	1000.0	0.3	1.5	0.3	30.0	30.0	1737.0	0.0	7.553114e+02	1.0	1.0
2	1000.0	0.3	2.0	0.3	30.0	30.0	1737.0	0.0	1.927994e+03	1.0	1.0
3	1000.0	0.3	3.0	0.3	30.0	30.0	1737.0	0.0	8.094331e+03	1.0	1.0
4	1000.0	0.3	4.0	0.3	30.0	30.0	1737.0	0.0	2.372947e+04	1.0	1.0
5	1000.0	0.3	10.0	0.3	30.0	30.0	1737.0	0.0	8.478769e+05	1.0	1.0
6	1000.0	0.3	16.0	0.3	30.0	30.0	1737.0	0.0	5.494539e+06	1.0	1.0
7	1000.0	0.3	26.0	0.3	30.0	30.0	1737.0	0.0	3.813417e+07	1.0	1.0

2. **Exploration vs. Exploitation**: Higher mutation strengths facilitate more extensive exploration of the solution space, enabling the algorithm to discover diverse solutions. However, maintaining convergence to optimal solutions demonstrates a balance between exploration and exploitation, ensuring that while exploring new solutions, the algorithm does not deviate from the optimal path.

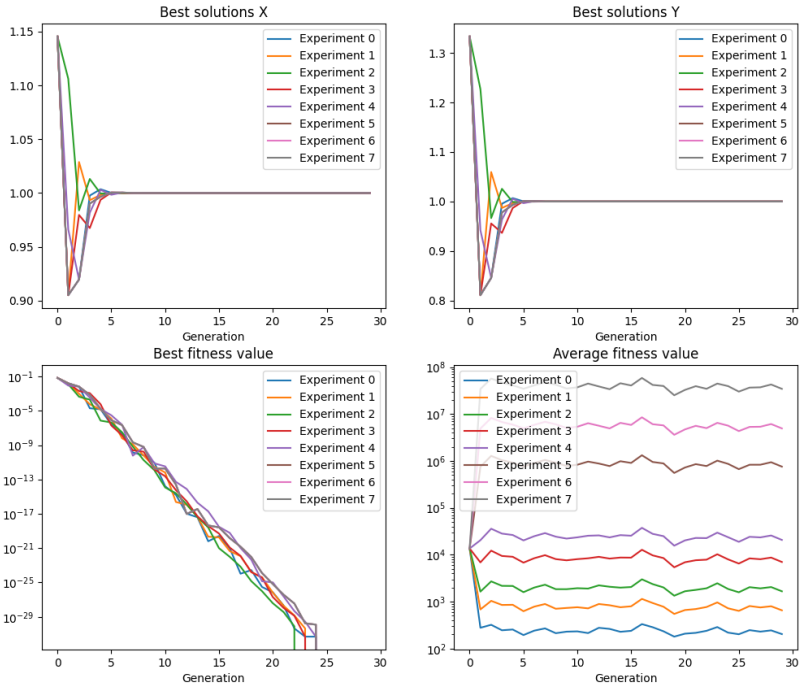

Figure 3.6. Influence of decreasing population mutation strength - average results between seeds

Mutation strength plays a crucial role in determining the genetic algorithm's performance, influencing both exploration and exploitation of the solution space. Higher mutation strengths enable the algorithm to explore a broader range of solutions, leading to improved fitness outcomes. However, maintaining convergence to optimal solutions highlights the algorithm's ability to balance exploration and exploitation effectively. Careful consideration of mutation strength is essential for achieving optimal performance in genetic algorithm optimization, ensuring a balance between exploration and exploitation for efficient convergence to desired solutions.

4. Summary

This study on genetic algorithms (GAs) emphasizes the critical influence of parameter optimization, randomness, crossover, and mutation mechanisms on GA performance. Through systematic experimentation, it was found that optimal parameter settings are essential for maximizing algorithm efficiency, while randomness significantly impacts solution consistency. Moderation of crossover rates proved optimal for balancing solution exploration with exploitation, enhancing algorithmic convergence.

Additionally, our findings suggest that lower mutation rates favour convergence, whereas higher mutation strengths encourage a broader exploration of the solution space without detracting from convergence efficiency. This research advances the academic understanding of genetic algorithms, highlighting the importance of strategic parameter adjustment and the balanced implementation of genetic operations for improving GA performance in complex optimization contexts.