Laboratory 3

Variant #3
Mariya Zacharneva and Ruslan Melnyk

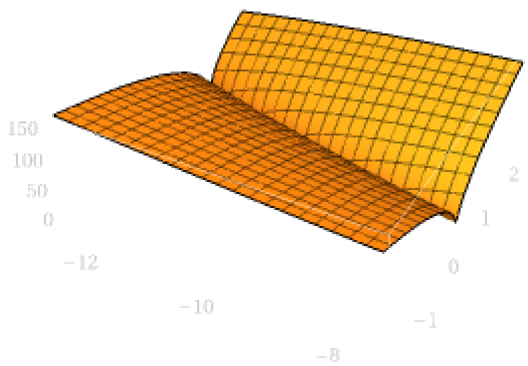
Introduction

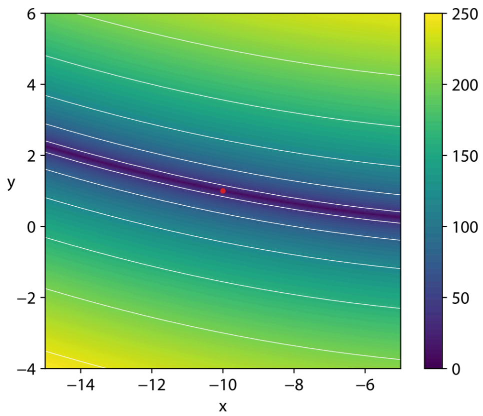
For this lab, we have to optimize Bukin function using Tournament Selection:

$$f(x, y) = 100\sqrt{|y - 0.01x^2|} + 0.01|x + 10|.$$

where x is in range [-15, 5] and y is in range [-3, 3].

Bukin function is one of the specifically designed test functions, also known as artificial landscapes, which are used for evaluating optimization algorithms. This exact function has its global minimum at the point (-10, 1).





Tournament selection

Tournament selection is a method that is used in evolutionary algorithms to choose individuals from a population for reproduction. In this method, a small group is randomly selected from the population, and then the candidate with the highest fitness is chosen as a parent. This process is repeated until the needed number of parents is reached.

After selecting parents, we run the crossover, or recombination stage. At this stage, the genetic information of two parents is being combined, producing the offspring with mixed features. We are using the Gaussian operator:

$$x_o = \alpha * x_{p1} + (1 - \alpha) * x_{p2}$$

$$y_o = \alpha * y_{p1} + (1 - \alpha) * y_{p2}$$

The next stage of the algorithm is the mutation: at this stage, random mutations are applied to a limited ratio of the population. In our algorithm, mutation means that a random value is being added to both X and Y coordinates.

In total, 5 parameters can be set to obtain the best results:

1. Population size

2. Mutation rate - the ratio of individuals that will be mutated
3. Mutation strength - how big will the changes be when we apply the mutation
4. Crossover rate - the ratio of individuals that will go through the crossover phase
5. The number of generations

Experiments

Best parameters

First of all, we tried a few combinations by hand, and found one which gives a fairly good result:

```
population_size=50,
mutation_rate=0.1,
mutation_strength=0.5,
crossover_rate=0.7,
num_generations=100,
```

With such parameters, the best fitness obtained is 0.1958.

Now, let's run a wider experiment with many parameter combinations, and see what gives the best results:

1. Population size values: 20, 50, 100, 200
2. Mutation rate values: 0.01, 0.05, 0.1, 0.2, 0.3
3. Mutation strength: 0.1, 0.5, 1, 3
4. Crossover rate: 0.2, 0.5, 0.8, 1
5. Generations: 20, 50, 100, 200

After running a simulation with all of these values, we have found that the 10 best results were obtained with the following parameters:

	Population Size	Mutation Rate	Mutation Strength	Crossover Rate	Generations	Score
887	100	0.2	3.0	0.5	200	0.000982
886	100	0.2	3.0	0.5	100	0.000982
885	100	0.2	3.0	0.5	50	0.000985
871	100	0.2	1.0	0.5	200	0.003310
870	100	0.2	1.0	0.5	100	0.003310
869	100	0.2	1.0	0.5	50	0.003310
868	100	0.2	1.0	0.5	20	0.008065
837	100	0.2	0.1	0.5	50	0.011812
839	100	0.2	0.1	0.5	200	0.011812

838	100	0.2	0.1	0.5	100	0.011812

What we can see from this table:

1. For some parameters, there are clearly winning settings: for example, all top 10 results have population size 100, mutation rate 0.2, and crossover rate 0.5. However, for the other two parameters, very good results can be obtained with very different settings.

2. The number of generations seems to be the least important setting: we can see all tested values in the top 10 table. Even in the top 3, the number of generations is the only parameter that is changing.

3. Even though good results can be obtained with different mutation strengths, the best results were achieved when it was relatively high.

Now we can tune our parameters even more precisely. We will leave the number of generations and mutation strength as it was before, however, for the other 3 parameters, we will test results in a close range to our previous winning numbers:

1. **Population size values: 90, 100, 110, 120**
2. **Mutation rate values: 0.18, 0.2, 0.22, 0.25**
3. Mutation strength: 0.1, 0.5, 1, 3
4. **Crossover rate: 0.45, 0.5, 0.55, 0.6**
5. Generations: 20, 50, 100, 200

Results:

	Population Size	Mutation Rate	Mutation Strength	Crossover Rate	Generations	Score
1003	120	0.25	1.0	0.55	200	0.000776
1002	120	0.25	1.0	0.55	100	0.000776
1001	120	0.25	1.0	0.55	50	0.000781
375	100	0.20	3.0	0.50	200	0.000982
374	100	0.20	3.0	0.50	100	0.000982
373	100	0.20	3.0	0.50	50	0.000985
969	120	0.25	0.1	0.55	50	0.001025
971	120	0.25	0.1	0.55	200	0.001025
970	120	0.25	0.1	0.55	100	0.001025
750	110	0.25	1.0	0.60	100	0.001431

Our previous assumption, that the number of generations doesn't matter too much, is confirmed here as well. The same applies to mutation strength; we can see all options in the leaderboard. For the last 3 parameters, we can see that there definitely are dominant values, but within such a small range, the difference is not that dramatic.

Therefore, our best achieved result is:

Parameters	Results

`population_size=120,`
`mutation_rate=0.25,`
`mutation_strength=1,`
`crossover_rate=0.55,`
`num_generations=200`
	`Best solution: Individual(`
	`x=np.float64(-9.922362064024572),`
	`y=np.float64(0.9845326892959396)`
	`)`
	`Best fitness: 0.0007763793597542801`
	`Average fitness: 20.166185695386353`

Randomness

With the obtained best parameters, let's try to use different random seeds to see if the results are stable:

For seed 18	best solution: [-3.344 0.112]	best fitness: 0.069	average fitness: 22.681
For seed 28	best solution: [-6.942 0.482]	best fitness: 0.031	average fitness: 24.043
For seed 45	best solution: [-1.977 0.039]	best fitness: 0.08	average fitness: 20.905
For seed 180	best solution: [-1.727 0.03]	best fitness: 0.083	average fitness: 21.819
For seed 2390	best solution: [-11.435 1.308]	best fitness: 0.026	average fitness: 22.852

As we can see, different random seeds can actually change the result a lot, and optimal parameters only work for a given seed. Even though the fitness is very small for all of the results, the found solution is often far from the actual minimum value.

Now we can try to find such a combination of parameters, which would give us good results for all different random seeds, even if it will not be super-precise for any of them in particular. To do that, we will run the result again as in the previous section, but this time each result will be run 5 times for different seeds, and the sum of all fitnesses will be used for evaluation.

	Population Size	Mutation Rate	Mutation Strength	Crossover Rate	Generations	Score
1003	120	0.25	1.0	0.55	200	0.003882
1002	120	0.25	1.0	0.55	100	0.003882
1001	120	0.25	1.0	0.55	50	0.003906
375	100	0.20	3.0	0.50	200	0.004910
374	100	0.20	3.0	0.50	100	0.004910
373	100	0.20	3.0	0.50	50	0.004927

969	120	0.25	0.1	0.55	50	0.005127
971	120	0.25	0.1	0.55	200	0.005127
970	120	0.25	0.1	0.55	100	0.005127
750	110	0.25	1.0	0.60	100	0.007156

The results are not very different from the results with a single seed value.

Let's also rerun the algorithm with decreasing population size and different seed values. In each cell, you can see the best solution, the best fitness, and the average fitness. Results, which are the closest to the actual minimum value, are marked green.

Seed	Population=120	Population=60	Population=30	Population=10
18	[-3.344 0.112] 0.069 22.681	[-9.508 0.904] 0.005 20.966	[-7.498 0.562] 1.522 20.165	[-6.003 0.36] 0.901 19.836
28	[-6.942 0.482] 0.031 24.043	[-1.541 0.024] 0.094 21.376	[-10.258 1.052] 0.815 22.978	[-11.81 1.39] 6.968 15.771
45	[-1.977 0.039] 0.08 20.905	[-12.979 1.684] 0.03 23.582	[-13.927 1.94] 0.04 16.947	[-11.647 1.356] 0.696 13.166
180	[-1.727 0.03] 0.083 21.819	[-2.733 0.075] 0.073 21.754	[-7.922 0.628] 0.063 18.609	[-7.879 0.621] 1.665 16.527
2390	[-11.435 1.308] 0.026 22.852	[-8.163 0.667] 1.354 22.82	[1.287 0.017] 0.274 17.395	[-10.016 1.007] 6.585 13.973

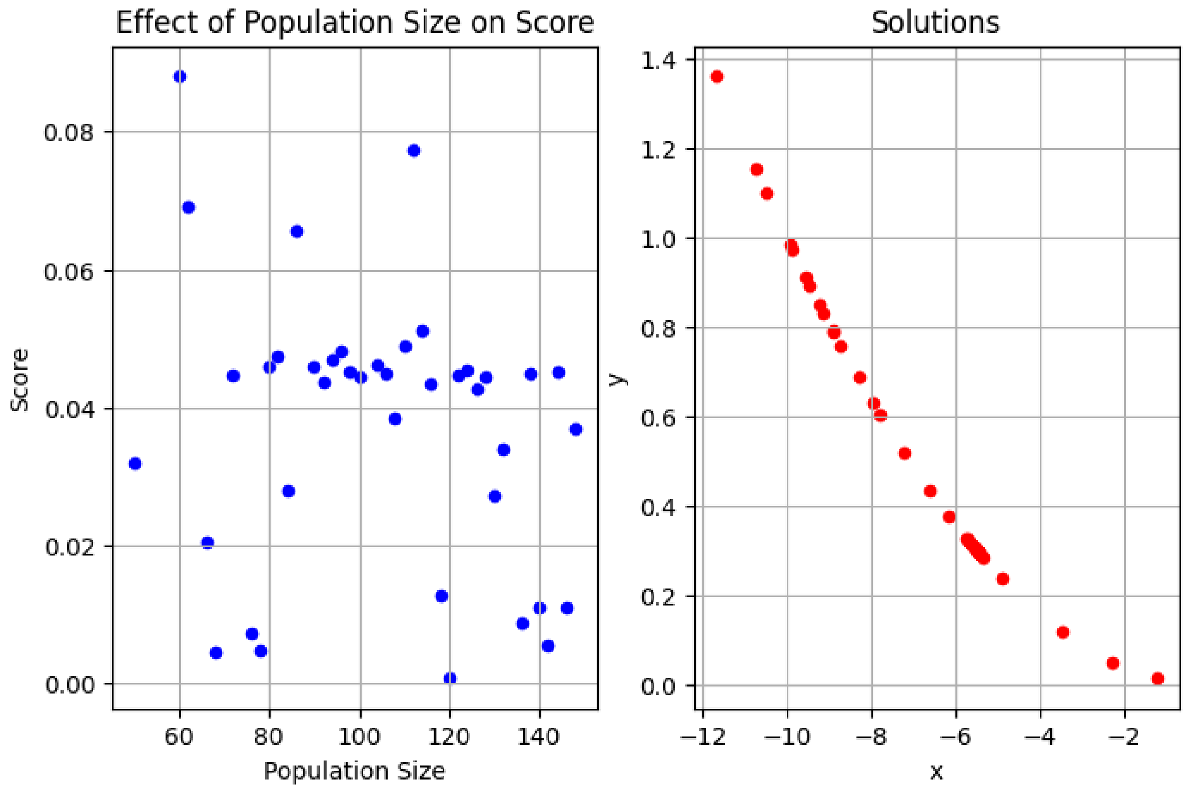
Changing each parameter separately

Let's take our best shot and try to change each parameter individually, to see how the results will change. For the fixed values, we will take:

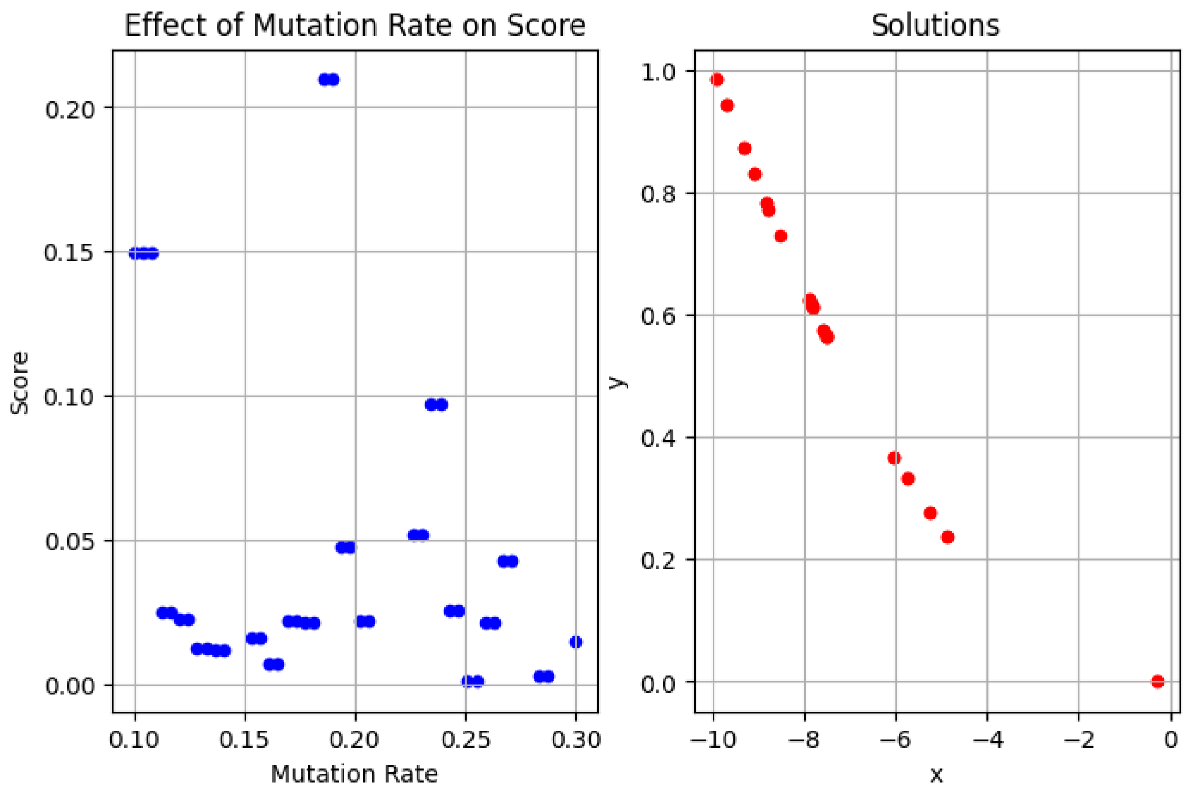
```
population_size=120,
mutation_rate=0.25,
mutation_strength=1,
crossover_rate=0.55,
num_generations=100
```

To get more accurate results, we will drop the worst 10 from each run. This will allow us to see the area near zero more closely and compare close values more precisely.

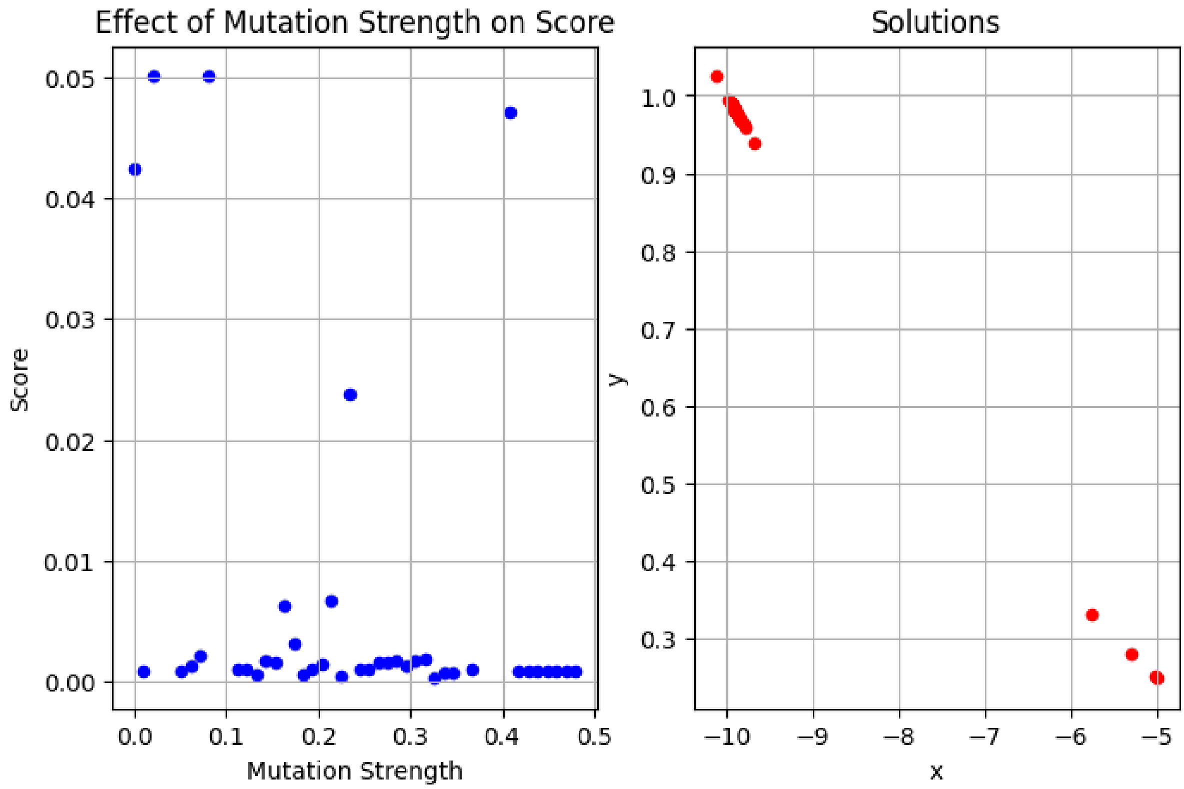
1. Population size: test_values = [x for x in range(50, 150, 2)]



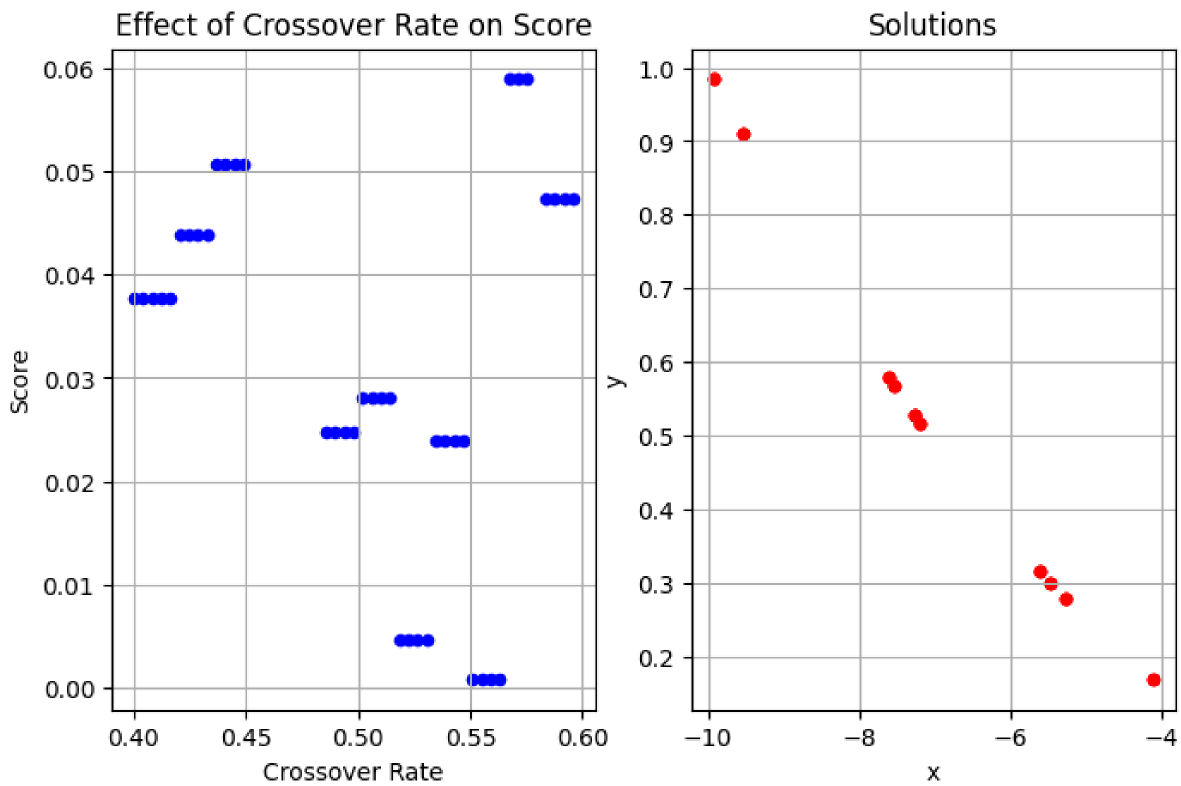
2. Mutation rate: test_values = [x for x in np.linspace(0.1, 0.3, 50)]



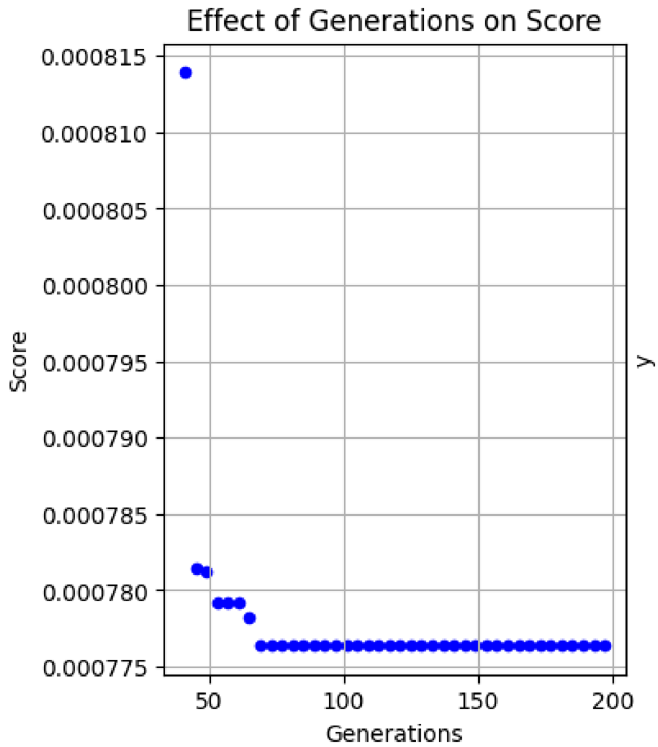
3. Mutation strength: test_values = [x for x in np.linspace(0, 0.5, 50)]

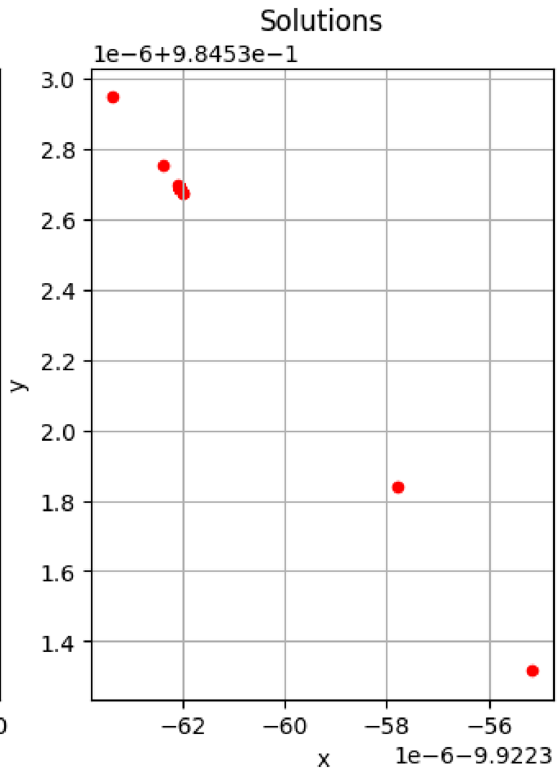


4. Crossover rate: test_values = [x for x in np.linspace(0, 0.5, 50)]



5. Generations: test_values = [x for x in range(1, 200, 4)]





Conclusion

In this lab, we tested how different parameters affect the performance of an evolutionary algorithm. The main parameters we tested were population size, mutation rate, mutation strength, crossover rate, and the number of generations. The results were shown using tables and graphs to help understand the impact of each setting.

From the experiments, we observed that certain values gave better results than others. For example, a moderate mutation rate and crossover rate helped the algorithm find good solutions without too much randomness. However, other parameters, such as population size, did not show a clear effect on the results. These insights can help when tuning an algorithm for better performance.

The best score was achieved with the following parameter settings:

Population Size: 120
Mutation Rate: 0.25
Mutation Strength: 1
Crossover Rate: 0.55
Generations: 200
Score: **0.000776**
Solution: **[-9.922, 0.984]**