EARIN

Lab 3 – variant 1

Evolutionary and genetic algorithms

Oleg Kim 309025

Seniv Volodymyr 309358

Date: 16/4/2023

1. Task Description:

Write a program that solves a given 2D function using a genetic algorithm. Variant 1: Optimize Rosenbrock function: $f(x, y) - \ (1 - x)^2 + 100(y - x^2)^2$. Use Evolutionary Strategy (1+1).

2. Launch instruction:

Open file, run script. Matplotlib package required. You will be asked to provide data related to calculations.

3. Theoretical background:

The (1+1) Evolutionary Strategy is a variant of evolutionary algorithms that can be used to optimize the Rosenbrock function. It is a simple and effective algorithm that utilizes selection and mutation to iteratively improve candidate solutions until an optimal solution is found.

The (1+1) Evolutionary Strategy algorithm's working process can be described in such sequence:

1.Initialization -> 2.Evaluation -> 3.Mutation -> 4.Evaluation -> 5.Selection ->

Repeat step 3-5 until stopping criteria is met. In our case program will stop when a specified by user number of generations is traversed.

- **Initialization** - Choose a random point in the domain of the Rosenbrock function as the initial candidate solution.
- **Evaluation** - Evaluate the fitness of the candidate solution by computing the value of the Rosenbrock function at that point.
- **Mutation** - Generate a mutated solution by adding a small random value to each of the candidate solution's coordinates. This introduces a small perturbation to the candidate solution, allowing the algorithm to explore new regions of the search space.
- **Selection** - If the mutated solution has a better fitness value than the candidate solution, replace the candidate solution with the mutated solution. Otherwise, keep the candidate solution and discard the mutated solution. This step is crucial for converging towards the optimal solution, as it allows the algorithm to focus on promising solutions and discard unpromising ones.

By repeating this process, the algorithm can iteratively improve the candidate solution and converge towards the optimal solution, which is at the minimum value of the Rosenbrock function. It is important to note that the performance of the (1+1) ES algorithm can be influenced by the specific choice of mutation operator and step size, which can affect the algorithm's exploration and exploitation abilities.

4. Solution:

To optimize Rosenbrock function, we wrote a small program in Python that operates in such manner:

After the program/script is run, the user is asked to provide some parameters requirement for calculations – x_min, x_max, y_min, y_max, mutation strength, mutation probability, number of generations.

- **x_min, x_max, y_min, y_max** – specifies the range in which randomly our x or y value will be chosen.
- **mutation strength (sigma)** - sigma is the mutation strength or mutation step size. It determines the size of the mutation when a new candidate solution is generated.

  The value of sigma controls how much the current solution will be perturbed to generate a new candidate solution. If sigma is large, the new candidate solution will be generated far from the current solution, while if sigma is small, the new candidate solution will be generated close to the current solution. Therefore, sigma plays a crucial role in determining the exploration-exploitation trade-off in the search process. A large sigma value will result in a more explorative search, while a small sigma value will result in a more exploitative search.

- **mutation probability** - variable represents the probability of mutation. It is used to determine whether a mutation should occur or not.

  It determines the likelihood that the algorithm will explore new solutions in the search space. If the value is low, then the algorithm will explore the search space less and stick to the current solution more. If the value is high, then the algorithm will explore the search space more and be more likely to find a better solution. The value should be carefully chosen to balance exploration and exploitation of the search space.

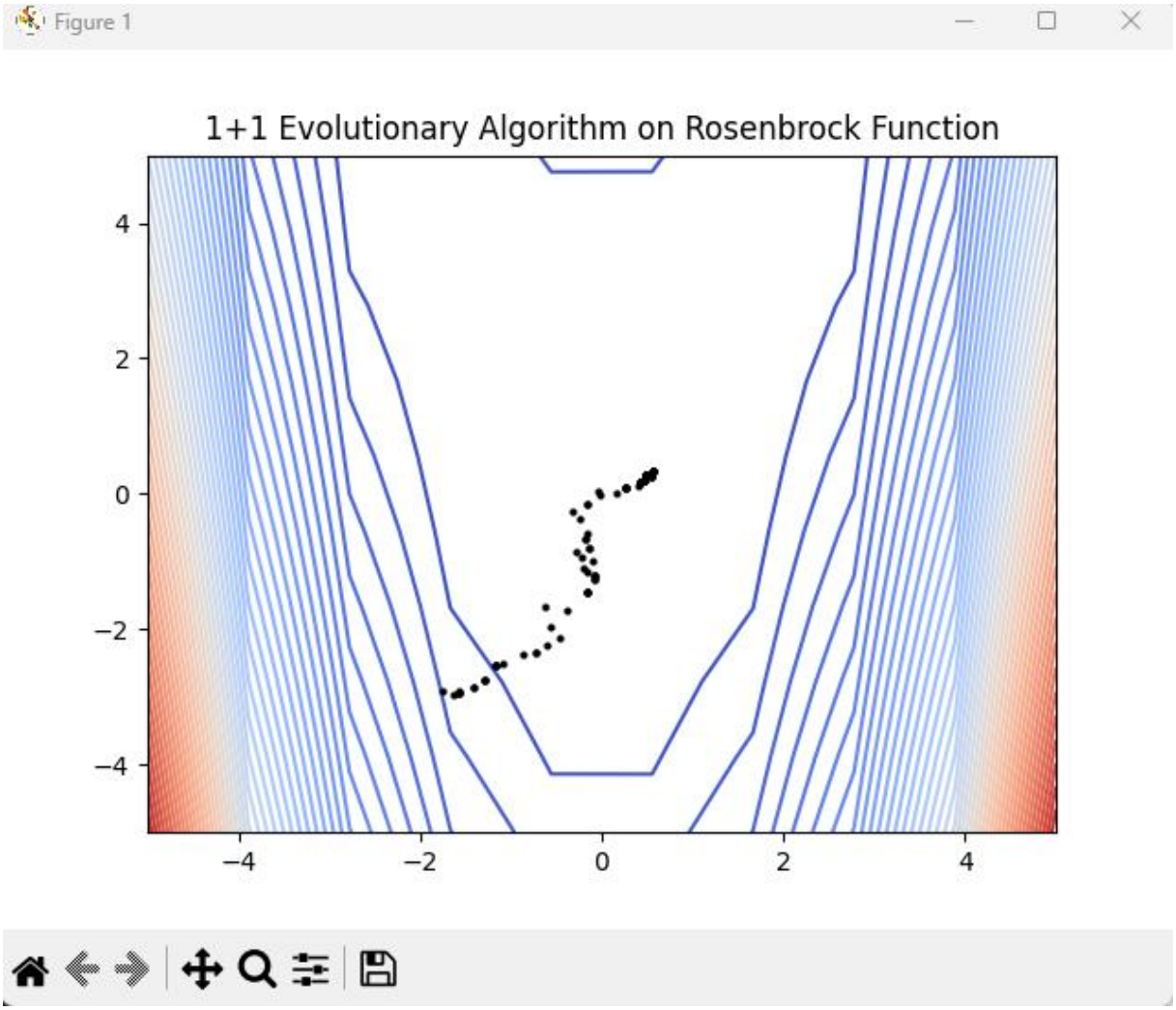
- **Number of generations** - variable represents the number of generations (or iterations) that the algorithm will run. It determines how many times the main loop will be executed, where a mutation is performed, and the fitness of the mutated point is evaluated.

The results of this algorithm will depend on the input values provided by the user, such as the mutation strength (sigma), mutation probability (mu_prob), and the number of generations (n_generations).

In general, the algorithm attempts to find the minimum value of the Rosenbrock function within the specified bounds of x and y. The algorithm starts by initializing a random point within the given bounds and then mutates that point using the mutation operator. The mutated point's fitness is evaluated using the Rosenbrock function, and if the mutated point is better than the current point, it replaces the current point. This process continues for a specified number of generations.

5. Tests:

Test 1



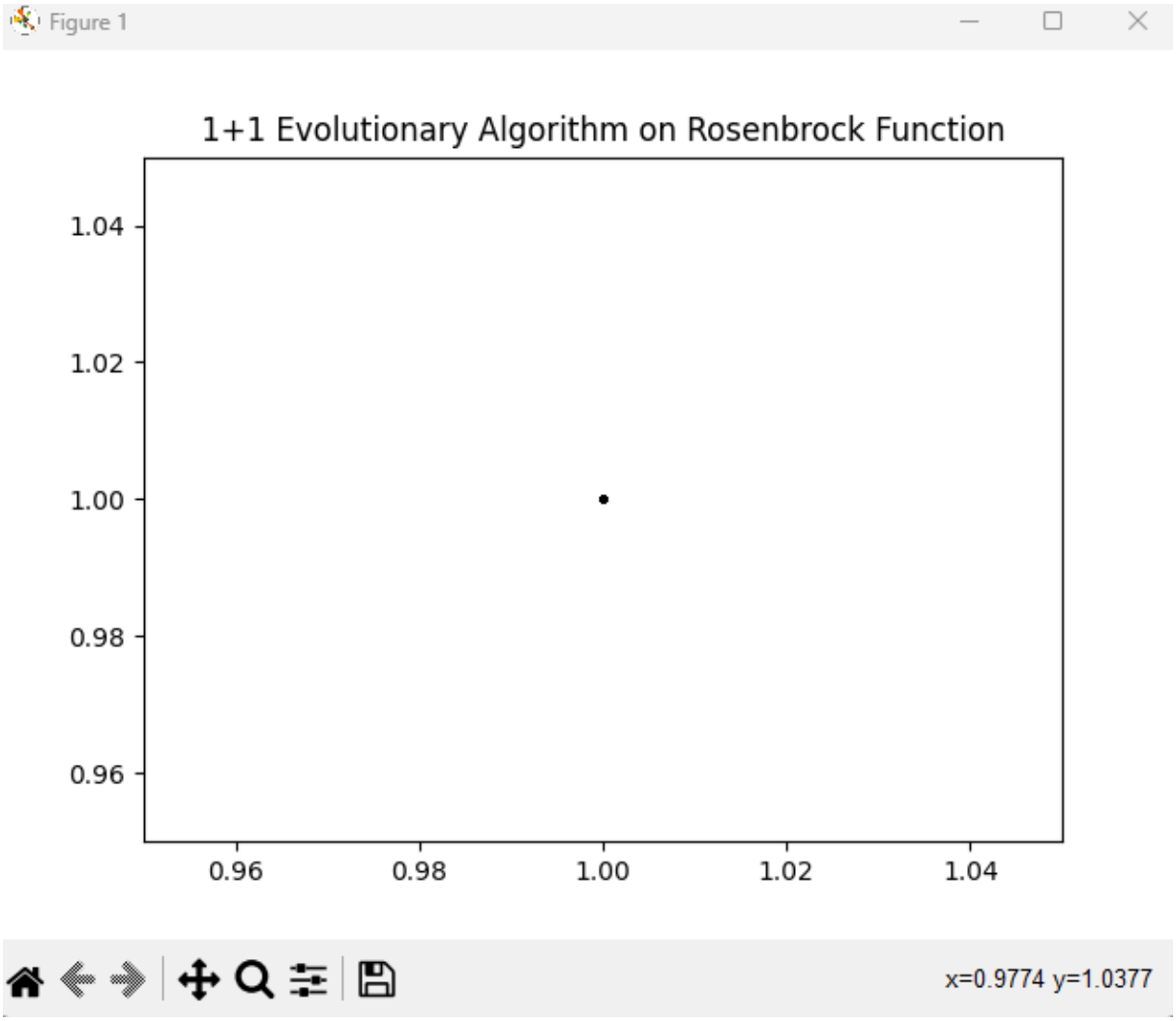
Input:

x_min: -5, x_max: 5,
y_min: -5, y_max: 5,
mutation strength: 0.1,
mutation probability: 0.2,
number of generations: 1000

output:

Optimal solution: x=0.5621738432757428, y=0.3247718432575347, f=0.19931724753120966
Optimization time: 0.0009968280792236328 seconds

Test 2



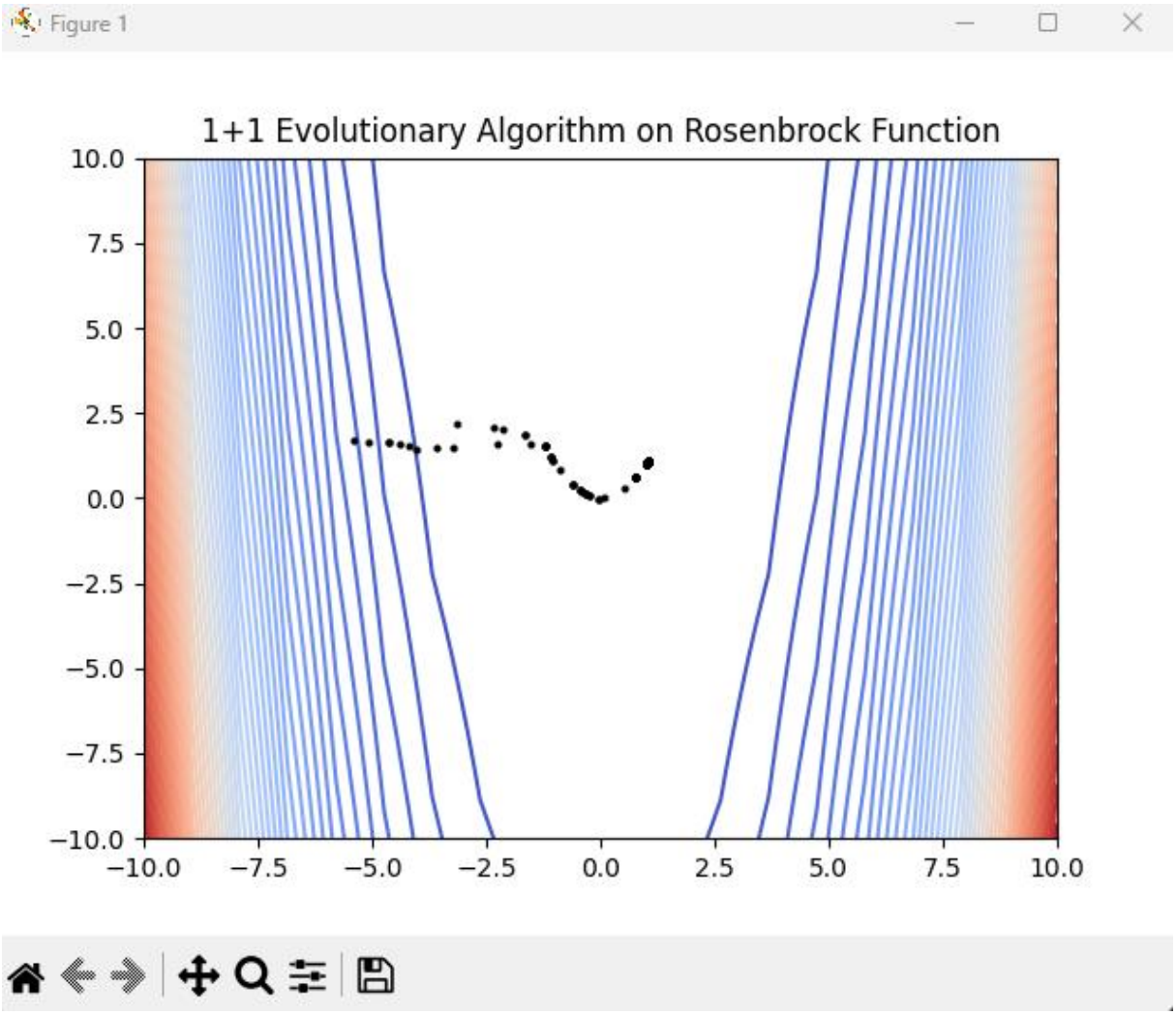
Input:

x_min: 1, x_max: 1,
y_min: 1, y_max: 1,
mutation strength: 0.1,
mutation probability: 0.2,
number of generations: 1000

output:

Optimal solution: x=1.0, y=1.0, f=0.0
Optimization time: 0.0009913444519042969 seconds

Test 3



Input:

x_min: -10, x_max: 10,
y_min: -10, y_max: 10,
mutation strength: 0.2,
mutation probability: 0.4,
number of generations: 2000

output:

Optimal solution: x=1.0346181703836324, y=1.0707745726390951, f=0.001209965086440352
Optimization time: 0.0029993057250976562 seconds

6. What can be improved:

Visualization: the graph can be visualized better; the plot overall can be clearer. The final values could be displayed on the graph, not in console.

GUI can be added. All interactions are made in console and the data that should be provided can come with an instruction to it.

The number of Input parameters can be lowered, or the algorithm restructured in such way, for it to be easier to interact with.

7. Conclusion:

Overall, the (1+1) ES algorithm is a simple yet effective method for optimizing the Rosenbrock function and can be a useful tool for tackling other optimization problems as well.

From the tests we have made we can assume that independently of the input data chosen the output x and y are converging towards the value of 1 and the output value of function itself (f) is converging to 0.

Even if we provide same input data the output can be different since our program choses starting x and y in specified range randomly.

We can judge that the algorithm is working correctly, from test 2. Since the correct output of Rosenbrock function optimization at points 1, 1 should be 0.

Overall, the results of the algorithm will depend on the specific input values provided and how well the mutation probability is tuned.