MARIA KOILALOU *JUSTINE LE BOURG*

Artificial Intelligence

Laboratory 3

The Evolutionary Strategy (1+1) algorithm is implemented to optimize the Rosenbrock function in 3D space. The Rosenbrock function is a commonly used benchmark function for optimization algorithms, known for its narrow and curved global minimum. The goal of the algorithm is to find the global minimum of the Rosenbrock function by generating new candidate solutions (offspring) through mutations of the current best solution (parent), and selecting the best one based on the fitness value (the value of the Rosenbrock function).

The algorithm takes the following inputs from the user:

- The range of values for the x and y variables: The minimum and maximum values for x and y.
- The mutation strength (sigma): The magnitude of the Gaussian noise added to the parent's x and y values to generate offspring.
- The mutation probability: The probability of mutating each offspring.
- The number of generations: The number of times the algorithm will generate new offspring and evaluate their fitness.

The algorithm starts by initializing the population with a random value of x and y within the user-specified ranges. Then, it generates new offspring by adding random Gaussian noise to the parent's x and y values, and evaluates their fitness using the Rosenbrock function. If the fitness of the offspring is better than the parent's fitness, the offspring becomes the new parent. The algorithm repeats this process for the specified number of generations, gradually reducing the mutation strength and varying the mutation probability.

After the optimization process, the algorithm returns the best fitness value, the corresponding x and y values, and the optimization time. It also plots the optimization process in 3D space by showing the surface of the Rosenbrock function and the trajectory of the best solution over generations.

The Rosenbrock function is defined in the code as:

```
def rosenbrock(x, y):
    return (1 - x) ** 2 + 100 * (y - x ** 2) ** 2
```

The Evolutionary Strategy (1+1) algorithm is implemented as the evolve() function in the code. It takes the fitness function, initial x and y values, mutation strength, mutation probability, and number of generations as inputs, and returns the best fitness value, best x value, and best y value as outputs.

The evolve() function in the (1+1) evolutionary strategy algorithm is responsible for evolving a single parent solution into a new offspring solution through mutation and evaluating its fitness. The steps performed in the evolve() function are as follows:

1. Mutation: The parent solution, represented as a numpy array parent, is mutated to generate an offspring solution. The mutation is performed by adding a random Gaussian noise vector, scaled by the mutation_step_size, to the parent solution. This is done using the numpy random.normal() function, which generates random numbers from a Gaussian distribution with mean 0 and standard deviation 1, and then scaling it by the mutation_step_size.
2. Boundary Handling: If the offspring solution goes beyond the defined search space boundaries, it is clipped back to the boundary values using the numpy clip() function. This ensures that the offspring solution remains within the valid search space defined by the lower and upper bounds.
3. Evaluation: The fitness of the offspring solution is evaluated using the provided fitness function, which takes the offspring solution as input and returns a scalar fitness value. The fitness function is defined externally and passed as an argument to the evolve() function.
4. Selection: The offspring solution is compared with the parent solution to determine which one has higher fitness. If the offspring solution has higher fitness, it replaces the parent solution as the new solution for the next iteration. Otherwise, the parent solution is retained as the solution for the next iteration.
5. Return: The evolved solution, along with its fitness, is returned as a tuple (offspring, offspring_fitness) from the evolve() function.

The evolve() function is typically called in a loop for a specified number of generations or until a certain convergence criterion is met, in order to continuously improve the solution over iterations and optimize the target fitness function.

The suggested ranges for user input parameters are printed to the console, including the ranges for x and y values, mutation strength, mutation probability, and number of generations.

The user is prompted to enter the values for the parameters, including the minimum and maximum values for x and y, mutation strength, mutation probability, and number of generations.

The population is initialized with random values of x and y within the user-specified ranges, and the optimization time is measured using the time module.

The Evolutionary Strategy (1+1) algorithm is then run with the user-specified parameters using the evolve function, and the best fitness value, best x value, best y value, and optimization time are printed to the console.

The optimization time provides an estimate of the computational cost of running the algorithm, which can be useful for comparing the efficiency of different optimization algorithms or for benchmarking purposes. A shorter optimization time generally indicates a faster algorithm, although it may also depend on various factors such as the hardware used, the problem complexity, and the size of the population and number of generations.

Finally, the optimization process is plotted in 3D using the matplotlib library. The surface of the Rosenbrock function is plotted, and the trajectory of the best solution over generations is shown as a red asterisk. The x, y, and fitness values are labeled on the plot, and the plot is titled "Evolutionary Strategy (1+1) for Rosenbrock Function".