Atefe Rajabi 40230563

1. Functionality
   1. Objective functions module
   2. Evolutionary strategy module
2. Configurations and Results
   1. Schwefel Function
      1. 3-dimention vs 6-dimention
      2. Adaptive vs. Non-Adaptive
      3. Elitism vs. Generational
      4. Different test for best parameters
   2. Ackley Function
      1. 2-dimention vs 5-dimention
      2. Adaptive vs. Non-Adaptive
      3. Elitism vs. Generational
      4. Different test for best parameters

**1.Functionality**

1. **Objective functions module**

This Python module offers a set of objective functions commonly employed in optimization problems. The module is designed to be adaptable, allowing users to easily switch between different objective functions for optimization tasks.

## Libraries Used

- NumPy: It is used for efficient array operations and mathematical computations.

The `ObjectiveFunction` class is initialized with the following parameters:

- name (str): Name of the objective function. Currently supports "schwefel\_function" and "ackley\_function".

- dimension (int): Dimensionality of the function.

- range\_x (tuple): Range of values for the input variables, represented as a tuple (min\_value, max\_value).

**### Method: evaluate**

The `evaluate` method is used to evaluate the objective function for a given input vector.

**### Method: evaluate\_list**

The `evaluate\_list` method evaluates the objective function for a list of input vectors (population).

**### Internal Method: \_evaluate\_schwefel**

The `\_evaluate\_schwefel` method is an internal function used to evaluate the Schwefel function for a given input vector.

**### Internal Method: \_evaluate\_ackley**

The `\_evaluate\_ackley` method is an internal function used to evaluate the Ackley function for a given input vector.

1. **Evolutionary Strategy Module**

## Libraries Used

- json: This library is employed for reading and parsing configuration files.

- ObjectiveFunction: A custom module providing objective functions for optimization.

- NumPy (np): It is widely used in the Evolutionary Strategy algorithm for random number generation, array manipulation, and mathematical operations.

- Matplotlib.pyplot (plt): It enables the generation of graphs to illustrate the convergence and diversity of the evolving population over generations.

**### Method: initialize\_population**

This method generates the initial population of candidate solutions with random parameter values within the specified range.

**### Method: generate\_offspring**

This method generates offspring using local discrete recombination and optional mutation, depending on the configuration.

**### Method: non\_adaptive\_mutation**

(creep\_mutation which is a commonly used mutation for real-value representation)

The `non\_adaptive\_mutation` method introduces non-adaptive mutations to the input chromosome, modifying individual genes based on a log-normal distribution. The mutation process is influenced by the `mutation\_probability`, `mean\_log`, and `sigma\_log` parameters, allowing for controlled and variable changes to the chromosome.

- Mutation Process:

- For each gene in the chromosome, a random value is generated using `np.random.rand()`, which is uniformly distributed between 0 and 1. If the generated random value is less than the specified `mutation\_probability`, a mutation is applied to the corresponding gene.

- The mutation is performed by adding a value sampled from a log-normal distribution with parameters `self.mean\_log` and `self.sigma\_log`.

- The larger the `self.mean\_log`, the more the distribution is shifted towards larger changes.

- The larger the `self.sigma\_log`, the more variability is introduced, allowing for a wider range of changes.

- Log-Normal Distribution:

- The log-normal distribution is characterized by a shift towards larger values when `self.mean\_log` is larger.

- The variability of the mutations is controlled by `self.sigma\_log`, where a larger value increases the range of possible mutation magnitudes.

**### Method: global\_discrete\_recombination**

This method performs global discrete recombination on the given population, creating a new individual by selecting genes from random parents. However, it is worth noting that this method is not utilized in the Evolutionary Strategy (ES) structure. In the case of the **Schwefel function**, employing a multi-parent implementation of recombination is not considered ideal due to its **rugged** and **multi-modal** fitness landscape.

**### Method: local\_discrete\_recombination**

This method performs local discrete recombination on the given two parents, creating a new individual by selecting genes from randomly chosen parents. This recombination **is a better choice** compared to intermediate recombination since it truly recombines the good features of both parents, whereas intermediate recombination simply averages them and may eliminate the distinct attributes of each parent.

**### Method: generate\_offspring\_hybrid**

This method generates offspring using a hybrid recombination strategy, which combines global and local recombination based on the current generation number. It's important to note that this method is not utilized in the Evolutionary Strategy (ES) structure.

**### Method: hybrid\_recombination**

This method performs hybrid recombination based on the current generation number, choosing between global and local recombination strategies. Additionally, it's important to note that this method is not used in the Evolutionary Strategy (ES) structure.

**### Method: uncorrelated\_mutation\_n\_sigma**

This method performs uncorrelated mutation on the given chromosome and sigma parameters, introducing random changes with varying magnitudes.

**### Method: elitism\_survival\_selection**

This method performs elitism survival selection, selecting the best individuals from the combined population of parents and offspring.

**### Method: generational\_survival\_selection**

This method performs generational survival selection, selecting the best individuals from the offspring population.

**### Method: check\_solution**

This method evaluates the best solution in the current population, considering the objective function.

**### Method: run**

This method executes the Evolutionary Strategy algorithm and returns the results, including the best solution, convergence details, and performance metrics.

**### Method: terminate**

This method checks whether the termination criteria for the Evolutionary Strategy algorithm are met.

**### Method: has\_converged**

This method checks whether the algorithm has converged by analyzing the change in best fitness values over recent generations.

**### Method: no\_improvement**

This method checks whether there has been no improvement in best fitness values over a specified number of recent generations.

**### Method: plot\_fitness\_performance**

This method generates plots to visualize the fitness performance of the algorithm, including average and best fitness values over generations.

**### Method: plot\_diversity**

This method generates plots to visualize the diversity of the population over generations, based on the standard deviation of fitness values.

1. **Configurations:**
2. **Schwefel function:**