Atefe Rajabi 40230563

1. Functionality
   a. Objective functions module
   b. Evolutionary strategy module
   c. Parameters

2. Configurations and Results
   a. Schwefel Function
      i. 3-dimensional space vs. 6-dimensional space
      ii. Adaptive vs. Non-Adaptive
      iii. Elitism vs. Generational
      iv. Different test for best parameters (available in Jupyter notebook)
   b. Ackley Function
      i. 2-dimensional space vs. 5-dimensional space
      ii. Adaptive vs. Non-Adaptive
      iii. Elitism vs. Generational
      iv. Different test for best parameters (available in Jupyter notebook)
3. Overall Conclusion

Note: Jupyter notebook also has descriptive format.

1.Functionality

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.

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: 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: 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: 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.

Parameters:

Survival_method:

Explanation: Determines the strategy used for selecting individuals to survive from the current generation to the next.

Impact: Choosing between "elitism" and "generational" affects how the population evolves over generations. "Elitism" retains the best individuals, favoring exploitation, while "generational" replaces the entire population, allowing for more exploration.

Max_generations:

Explanation: The maximum number of generations the algorithm will run.

Impact: Affects the total number of iterations the algorithm performs. Higher values allow for more exploration but may lead to longer runtimes.

Population_size:

Explanation: The number of individuals in each generation.

Impact: Larger populations generally increase diversity but also computational cost. Too small a population may hinder exploration.

Num_offspring:

Explanation: The number of offspring generated in each generation.

Impact: Higher values promote more exploration but increase computational cost. The balance depends on the problem's complexity and the desired exploration-exploitation trade-off.

mutation_function_config:

Explanation: Configuration for the mutation function.

Impact: Parameters like `**mutation_enable**`, `**mutation_probability**`, `**mean_log**`, and `**sigma_log**` affect the exploration-exploitation trade-off and the diversity of the population.

In the case where mutation_enable is set to False, two methods are implemented: one for not adapting the sigma values, and another for employing creep mutation, is not called, which adds a number from a lognormal distribution to each gene if it satisfies the mutation probability. The parameters **mean_log** and **sigma_log** are used for configuring the lognormal distribution. The first method is employed when mutation_enable is set to False.

Sigma_config:

Explanation: Configuration for the sigma (step size) used in mutation.

Impact: The method for initializing sigma (`initialization_sigma_method`), the range for random initialization (`random_sigma_range`), and the threshold (`sigma_threshold`) affect the exploration behavior and convergence speed.

The (`sigma_threshold`) plays a crucial role in controlling the step size of mutations in the algorithm. If it's too high, it might lead to mutations that are too large, hindering the algorithm's ability to make small, incremental improvements, especially when the solution is close to the optimum.

Although, the ES adapts sigma by itself, the initialization of sigma can follow two approach one using default initialization, another using a random initialization within a provided range. Random initialization provides an initial level of diversity, and adaptive mechanisms help adjust the exploration-exploitation balance based on the algorithm's experience.

A larger sigma corresponds to a larger step size, leading to more significant changes in the parameter values during mutation. Consequently, larger sigma values promote more exploration in the search space.

Convergence_threshold:

Explanation: A threshold for determining convergence based on the average fitness change.

Impact: Controls when the algorithm considers the population to have converged. Lower values increase the required convergence, potentially leading to longer runtimes.

No_improvement_threshold:

Explanation: The number of generations with no improvement to trigger termination.

Impact: Controls when the algorithm terminates due to a lack of improvement. Higher values may allow for longer exploration before termination.

Objective_function_config:

Explanation: Configuration for the objective function.

Impact: Parameters like `objective_function`, `range`, and `chromosome_length` define the optimization problem. Different objective functions and problem dimensions will impact algorithm performance.

Configurations:

Note: All configurations are available in the 'configs' directory with their respective names. In this summary, only the best configuration is provided. Other configurations that are not considered the best are identifiable in the directory by having '-2' at the end of their names.

Schwefel function:

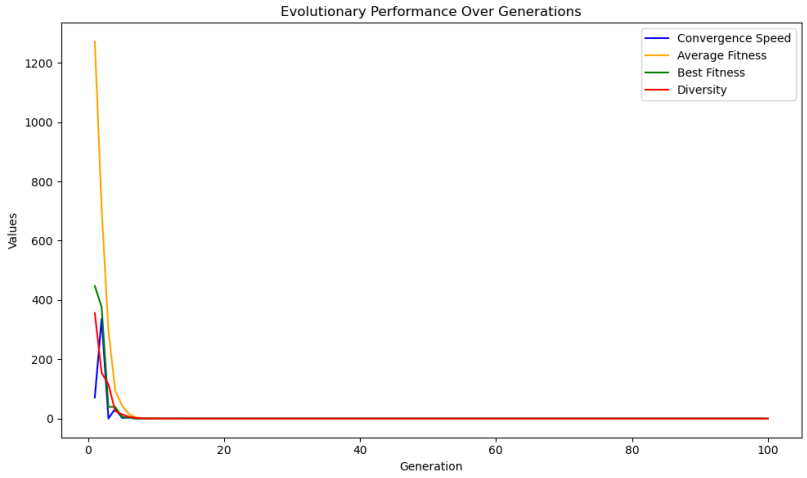
i. 3-dimensional space vs 6-dimensional space

With an increase in dimensional space, exploration should be increased since we are struggling with a more complex problem. One can increase the population size and the number of generated offspring or change the survival method or employ many other approaches to adjusting the exploring rate. It is preferable to increase the population size and the number of generated offspring while keeping the elitism strategy.

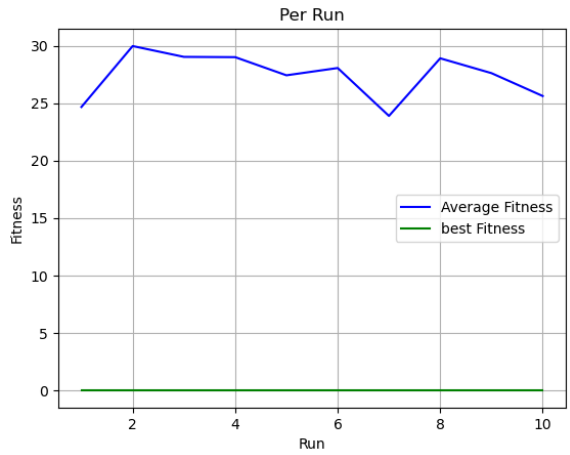
3- dimensions: (elitism)

Best result:

this is run number: 7
solution fitness: 3.818269897237769e-05
solution fitness: [420.9687468453309, 420.9687466074493, 420.96874584057014]



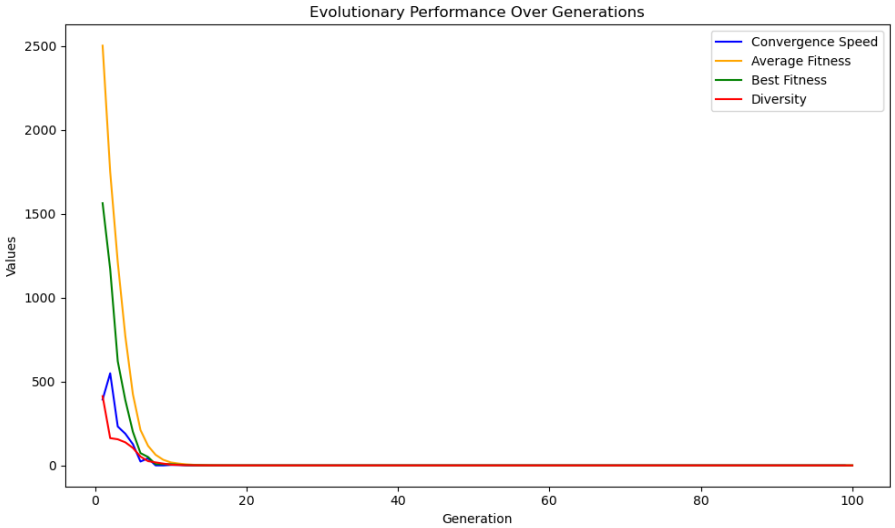
Results for 10 run in 3-dimension problem:



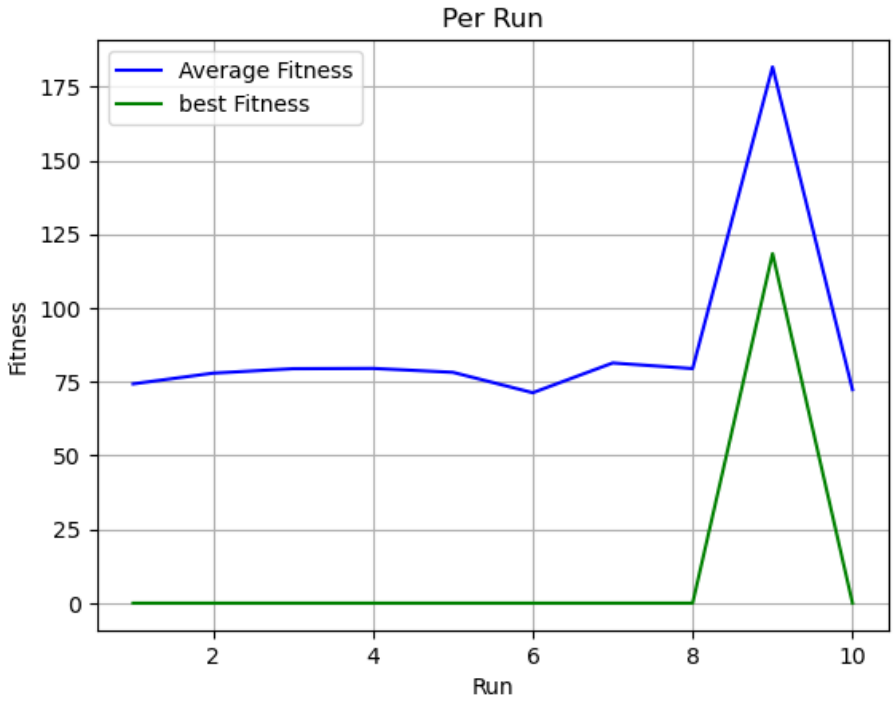
6- dimensions: (elitism)

Best result:

this is run number: 6
solution fitness: 7.636539930899744e-05
solution fitness: [420.9687449359284, 420.96874602936276, 420.96874642742443, 420.9687494806889, 420.96874762373017, 420.96874770544395]



Results for 10 run in 6-dimension problem:



ii. Adaptive vs. Non-Adaptive

After recombination, it checks whether mutation is enabled (mutation_enable). If so, it performs mutation using the uncorrelated_mutation_n_sigma method. If mutation is disabled, it uses non_adaptive_mutation_n_sigma. This method performs non-adaptive mutation using a fixed strategy parameter (self.strategy_parameters) for the entire chromosome. It adds random values sampled from a standard normal distribution (np.random.randn) multiplied by the strategy parameter to each gene in the chromosome.

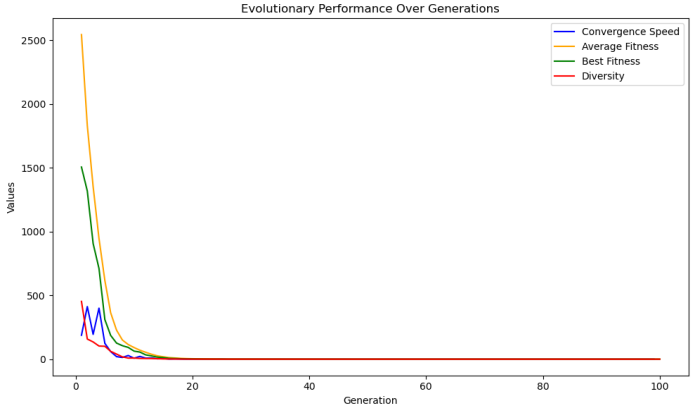
Adaptive:

The configuration previously included details about the adaptive part. (6- dimension)

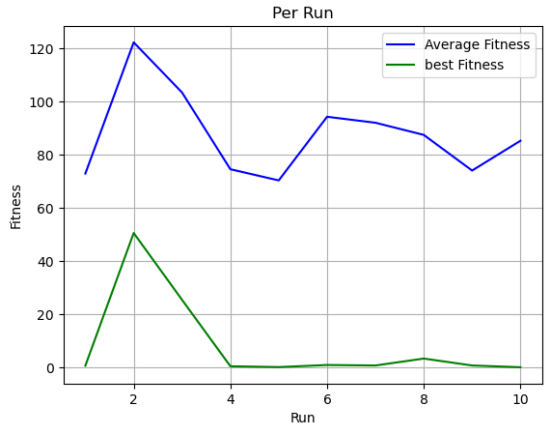
Non-Adaptive:

Best Result:

this is run number: 10
solution fitness: 0.06777694150241587
solution fitness: [421.31554038002184, 420.9534911100793, 421.1383374748764, 420.7138838150187, 420.8718635578487, 420.40929520904587]



Results for 10 run in 6-dimension problem:



Based on these results, we conclude that an Evolutionary Strategy (ES) without adaptive mutation is not efficient, and recombination does not have a significant effect on convergence.

iii. Generational vs. Elitism:

1.Generational Strategy:
Exploration vs Exploitation: Generational strategies are often associated with a higher level of exploration. Since the entire population is replaced, it allows for a more diverse set of solutions to be explored in each generation.
Advantages: It can help escape local optima and encourage a broader exploration of the solution space.

2. Elitism Strategy:
Exploration vs Exploitation: Elitism is often associated with a higher level of exploitation. By preserving the best individuals, it intensifies the focus on the promising areas of the solution space.
Advantages: It helps in converging towards optimal solutions, especially in situations where promising solutions are identified.

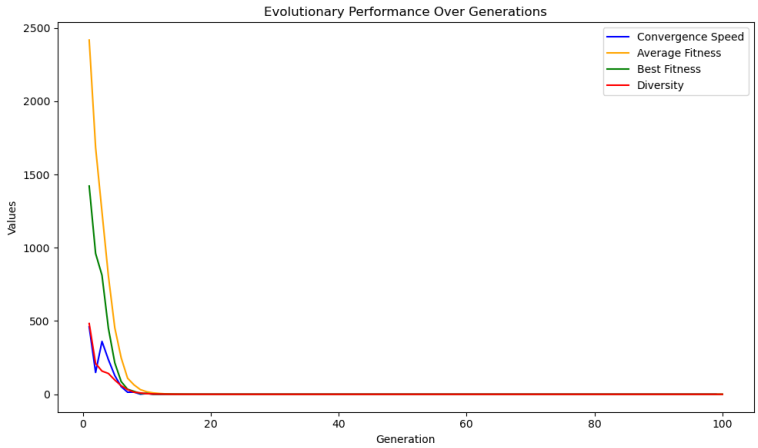
Elitism:

The previous configuration was with 6 dimensions and elitism.

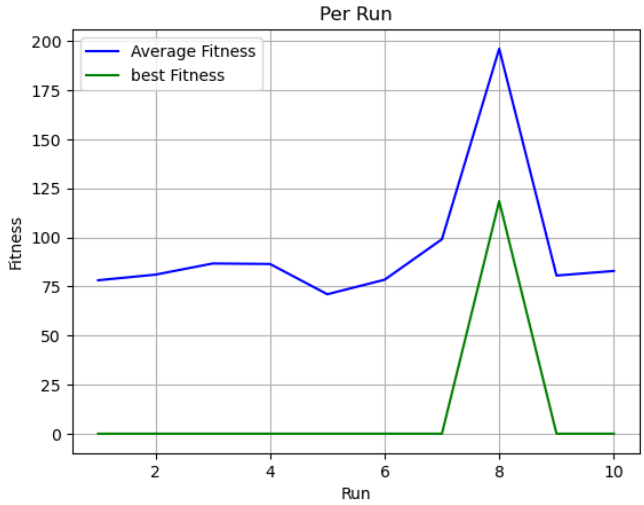
Generational:

Best result:

this is run number: 5
solution fitness: 7.636541977262823e-05
solution fitness: [420.9687432855832, 420.9687516988137, 420.9687540667513, 420.9687430539504, 420.9687544589369, 420.96874760414164]



Results for 10 run in 6-dimension problem:



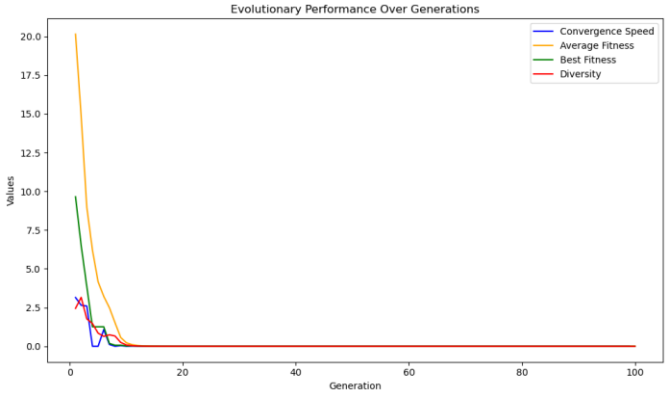
Ackley function:

i. 2-dimension vs 5-dimension

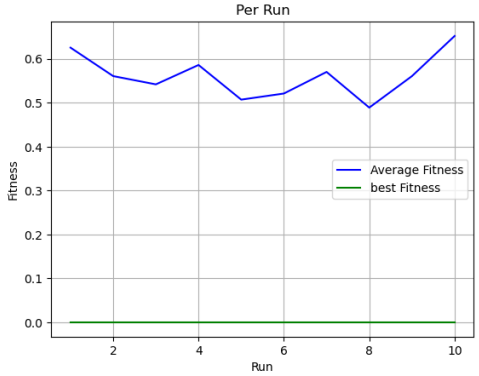
2-dimension:

Best result:

this is run number: 1
solution fitness: 4.440892098500626e-16
solution fitness: [-3.2851364593544174e-16, -1.1634770896898895e-16]



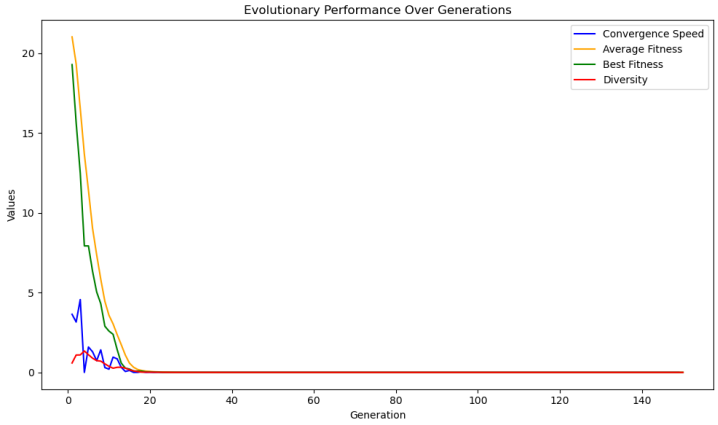
Results for 10 different run:



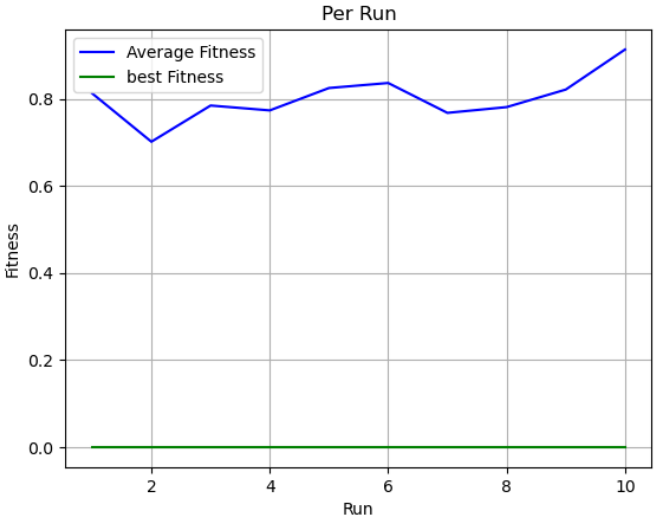
5-dimension:

Best Result:

this is run number: 1
solution fitness: 4.440892098500626e-16
solution fitness: [8.537211498626934e-17, 6.258530587852057e-17, 1.6256347668808719e-16, 1.8972182208099455e-17, 1.6163208617970615e-16]



Results for 10 different run:



ii. Generational vs Elitism:

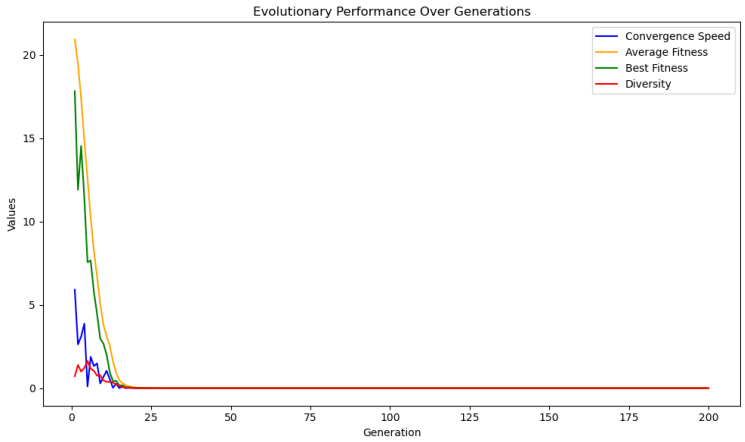
Elitism:

(Previous configuration)

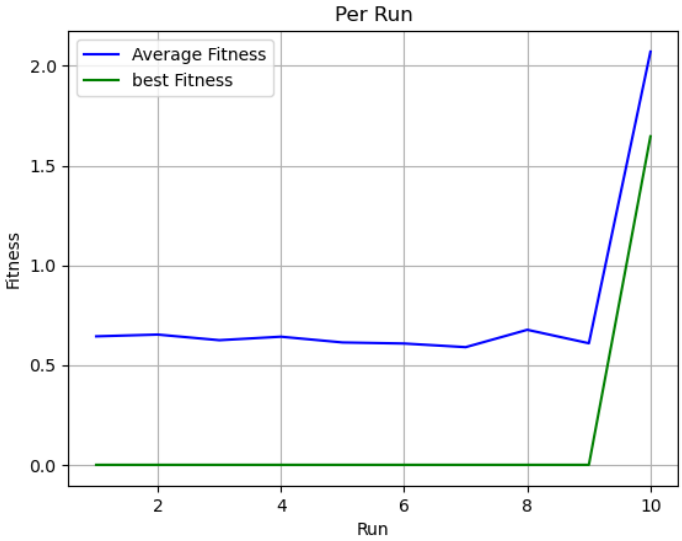
Generational:

Best Result:

this is run number: 1
solution fitness: 4.440892098500626e-16
solution fitness: [1.1636909201999552e-16, 7.648148193212447e-17, 6.473606916082792e-18, -4.497928905301464e-16, -3.053478139501094e-16]



Result for different 10 run:



iii. Non-Adaptive vs. Adaptive:

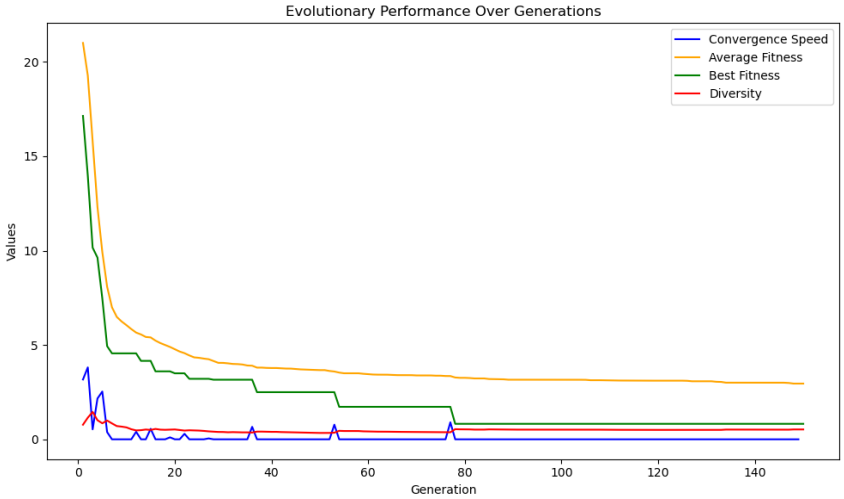
Adaptive:

(Previous Configuration)

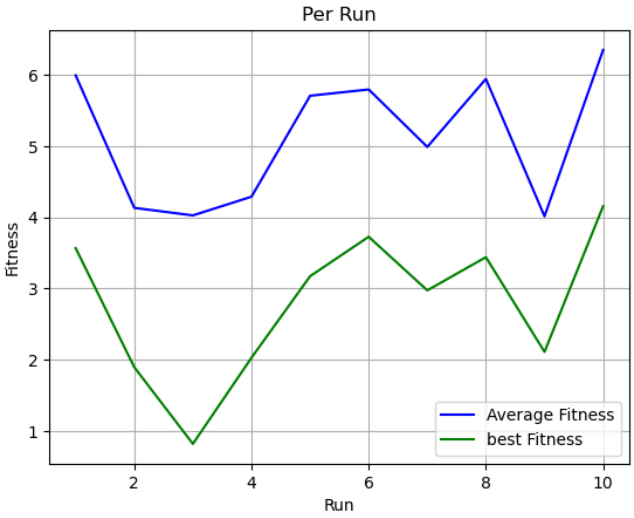
Non-Adaptive:

Best Result:

this is run number: 3
solution fitness: 0.8201868411803699
solution fitness: [0.11382350140808262, 0.05701548100959197, 0.081111009276074, 0.15258463794597643, 0.027511651487749228]



Result for 10 different run:



Overall result:

1. Adaptive Mutation:
   - Adaptive mutation is critical in ES, particularly for problems with complex landscapes like the Schwefel function. It allows the algorithm to adjust mutation rates dynamically, balancing exploration (searching new areas) and exploitation (refining current solutions).
   - For both Schwefel and Ackley functions, adaptive mutation seems to perform better than non-adaptive strategies, which is evident from the superior solution quality (lower fitness values) in the adaptive case.

2. Selection Methods:
   - The $(\mu + \lambda)$ selection method, which includes parents and offspring in the selection pool, may lead to a faster exploitation of promising search areas but could also reduce diversity over time.
   - The (μ, λ) selection method, which selects only from offspring, promotes exploration and can maintain or increase diversity, potentially preventing premature convergence to local optima.

3. Dimensionality:
   - Increasing the dimensionality of the problem space, as seen in the Schwefel function from 3D to 6D, requires adjustments in the algorithm to maintain effective exploration and convergence rates.
   - In higher-dimensional spaces, strategies like increasing the population size and the number of offspring, as well as maintaining elitism, can help cover the broader search space.

5. Objective Functions:
   - For the Ackley function, which typically has a unimodal landscape, global recombination methods are effective.
   - For multimodal functions like Schwefel, local recombination is preferred due to the need for more targeted exploration around multiple local optima.

6. Sigma Threshold:
   - The sigma threshold plays a crucial role in controlling mutation depth. A suitably small threshold allows the ES to perform fine-grained search, which is necessary for problems where the solution involves very small numbers, such as the Ackley function with a solution magnitude of 10^{-16}.

7. Algorithm Performance:
   - The performance of the ES is evaluated through metrics like convergence speed, solution quality, and robustness. The configuration parameters are fine-tuned to optimize these performance metrics.