# **Moth Flame Optimization Algorithm**

Rucha Patil, Bau Nguyen, Momoreoluwa Ayinde

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Introduction**

In this project, we implement the Moth-Flame Optimization (MFO) algorithm, which is a nature-inspired algorithm based on the behavior of moths' attraction to flames. This algorithm was proposed by Seyedi Mirjalili in 2015. The MFO algorithm mimics the natural behavior of moths to search for optimal solutions in various optimization problems. It simulates the movement of moths to the brightest flame by iteratively updating the moths' positions based on the light’s intensity. The aim is to converge to the global optimum. We implemented our algorithm with the Python programming language.

**Implementation**

* **Libraries Used**
  + **sys**: For system-specific parameters and functions.
  + **random**: For generating random numbers.
  + **numpy** **(np)**: For numerical operations and array manipulation.
  + **pandas (pd)**: For data manipulation and analysis.
  + **matplotlib.pyplot (plot)**: For data visualization.
  + **tabulate:** For creating formatted tables from tabular data.
* **Classes**
  + **MFO (Moth-Flame Optimization)**:   
    This is the main class that implements the MFO algorithm. It includes methods for initialization, updating moth positions, evaluating fitness functions, and generating optimization reports.
  + **Fitness Functions:**   
    This class defines various fitness functions used in optimization problems. It also provides details such as dimensionality and bounds for each fitness function. Following the paper: we implemented fitness functions f1-f11. These fitness functions include unimodal and multimodal models

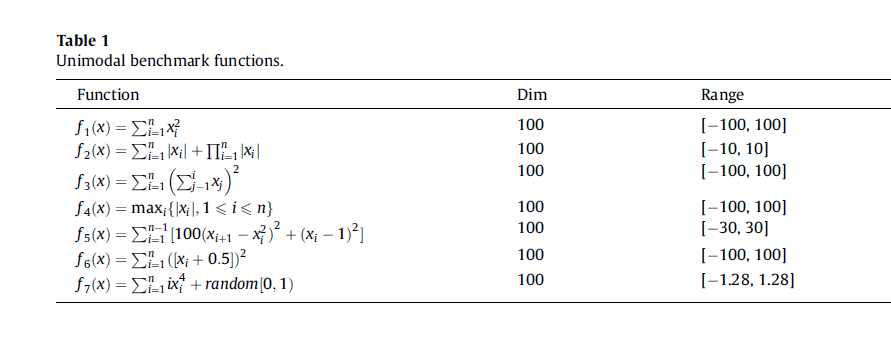
**Detailed Explanation of our Implementation**

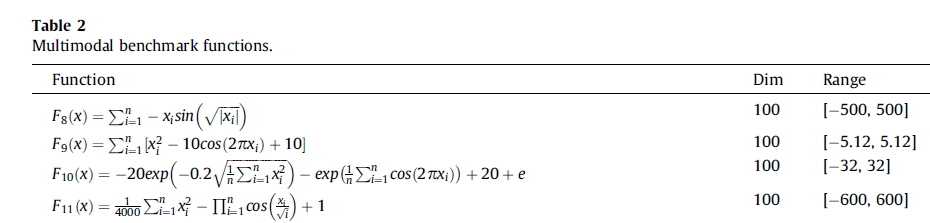
* **Initialization:**
  + The MFO class is initialized with parameters such as maximum iterations, number of moths, fitness function, etc.
  + Fitness functions are retrieved based on their names, and details like dimensionality and bounds are obtained.
* **Position Initialization:**
  + Moth positions are initialized within specified bounds randomly.
* **Fitness Evaluation:**
  + The fitness of moths is evaluated using the specified fitness function.
* **Optimization Process:**
  + The optimization process iterates through a specified number of iterations.
  + Moths’ positions are updated based on the behavior of moths attracted to flames.
  + The best moths (flames) are determined and their fitness values are updated.
  + A convergence curve is generated to visualize the optimization process.
* **Report Generation:**
  + A report containing optimization details such as best scores and positions in each iteration is generated.
  + The report is written into a file for further analysis.
* **Main Functionality:**
  + **Main Function:**
    - The main function orchestrates the execution of the MFO algorithm.
    - It interacts with the user to input parameters such as fitness function, maximum iterations, and number of moths.
    - The optimization process is initiated, and results are visualized and reported.
  + **Utility Functions:**These areprovided for obtaining user inputs and generating reports.

**Results**

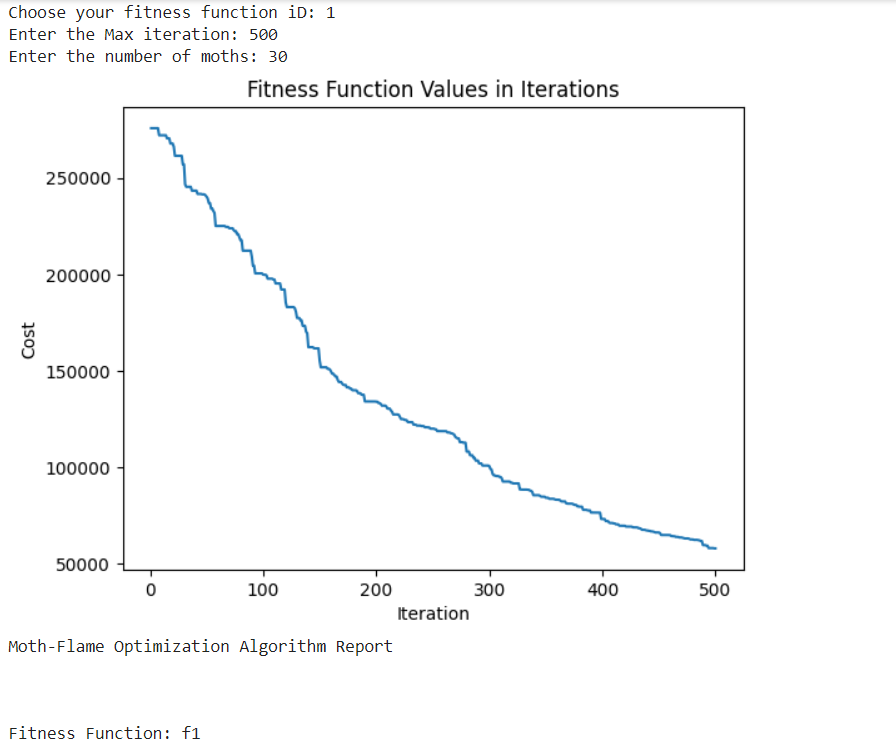
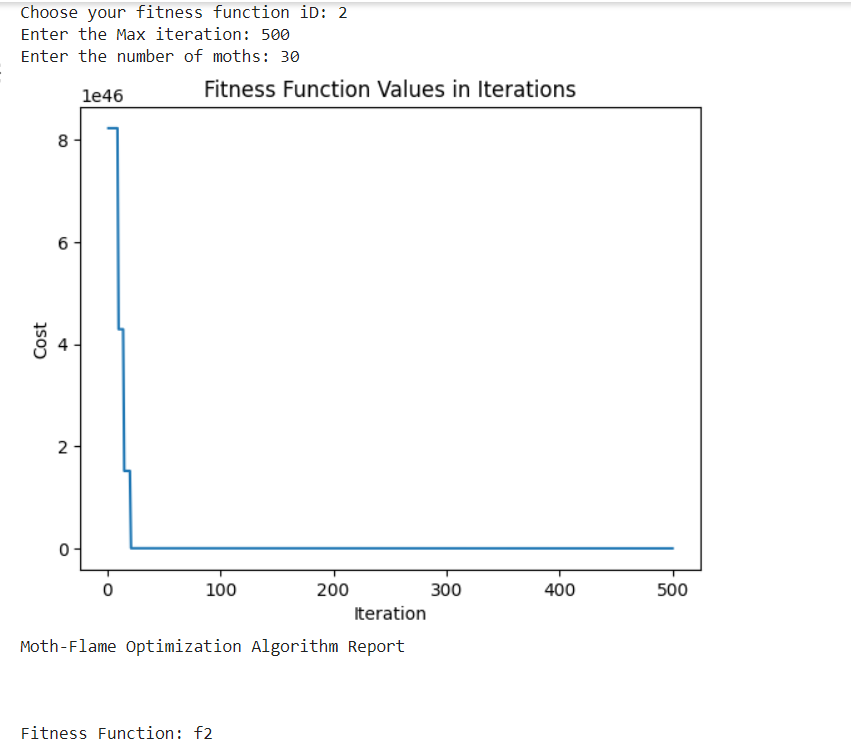
We followed the paper and used the number of moths = 30 and the number of iterations = 500. The number of iterations used in the paper is 1000, but for computation reasons, we limited our experiment to 500 iterations. We used these values for all implemented fitness functions and below we see the visualization of the result. This experiment was carried out on a device with a processor - Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz, RAM- 8.00 GB, and a 64-bit operating system, x64-based processor so results may differ based on the device used for execution.

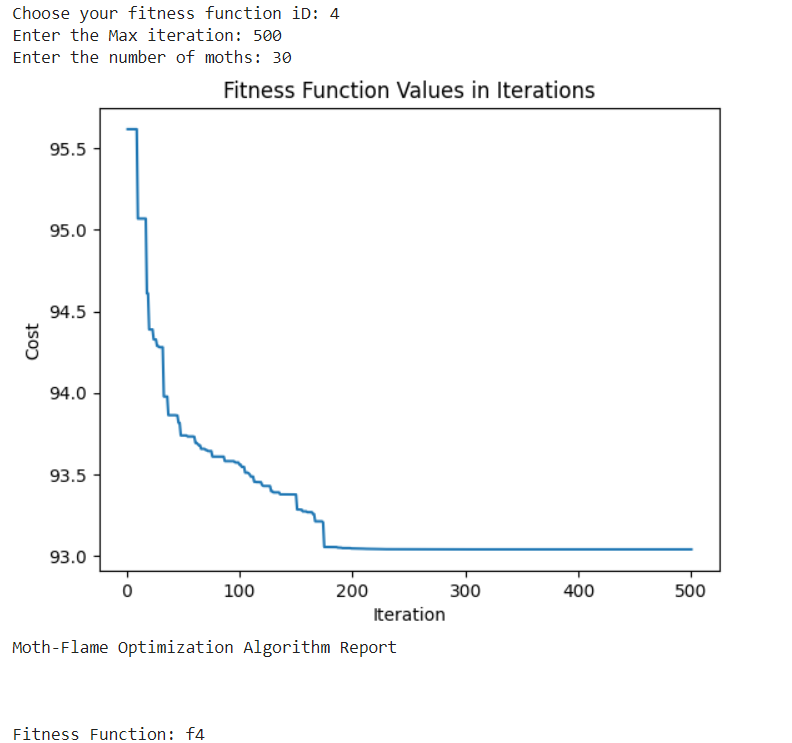
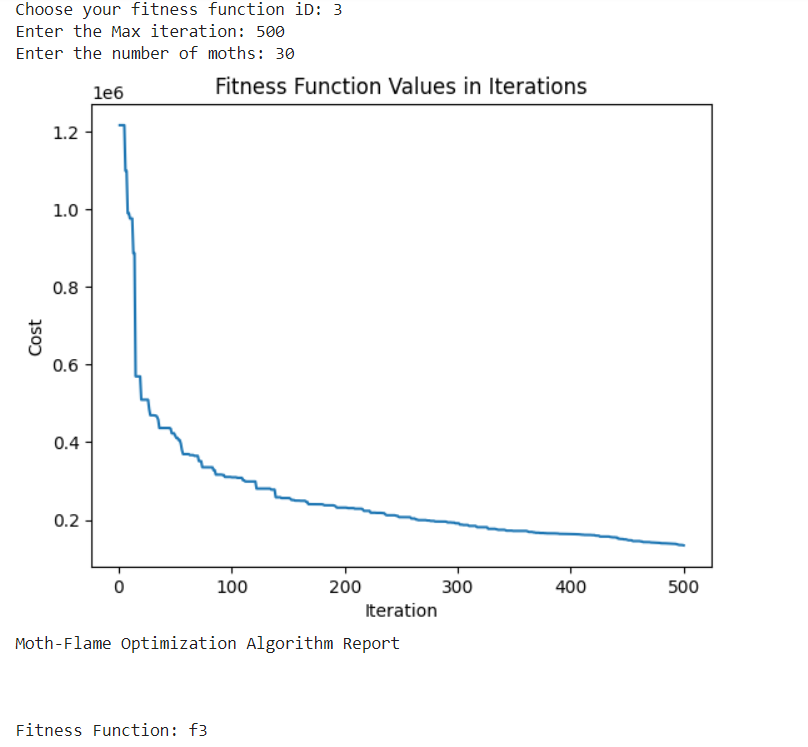
The graphs below represent the convergence curve we got after running our optimization algorithm on the fitness functions. f1-f11 are unimodal and multimodal functions and the full formula is shown below:

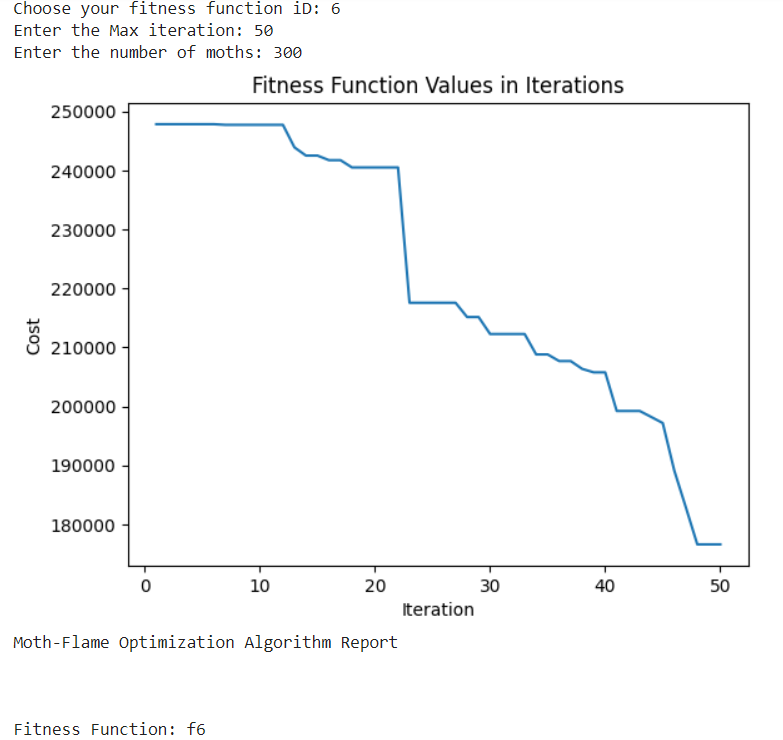
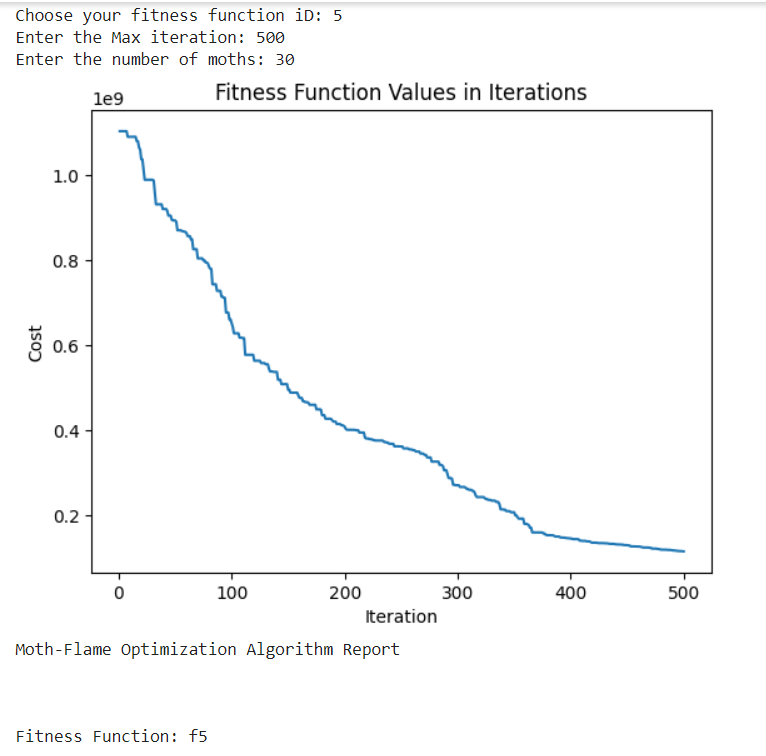


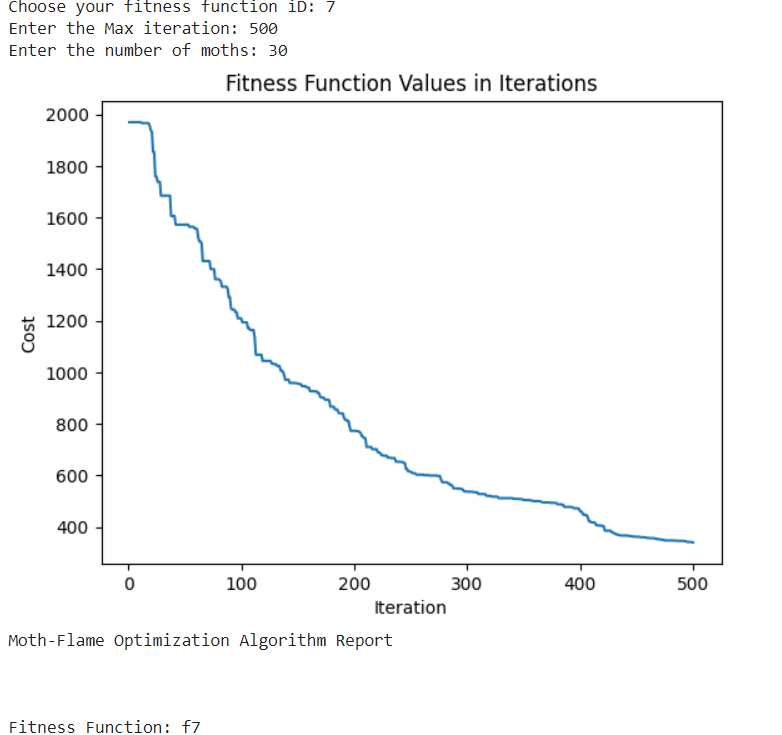
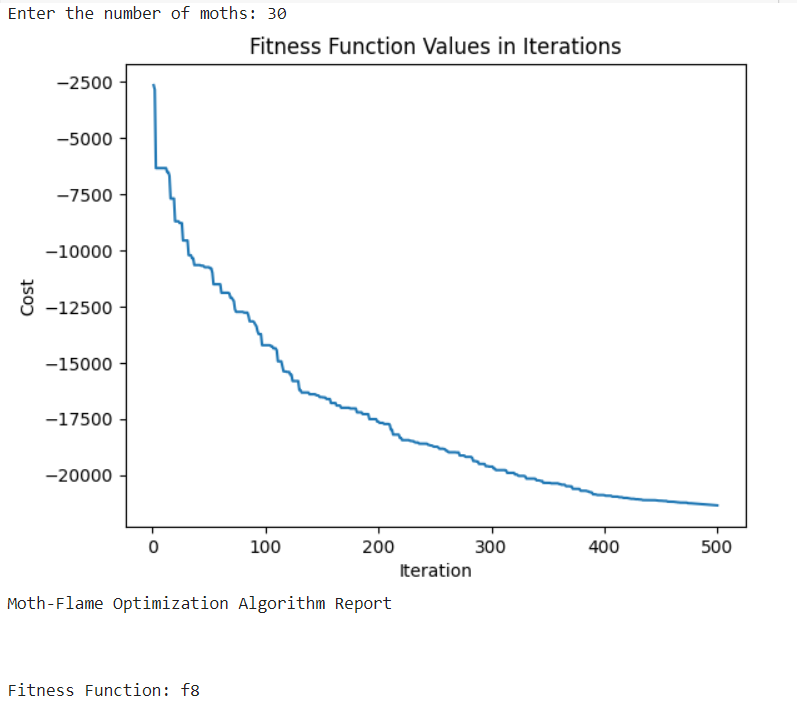


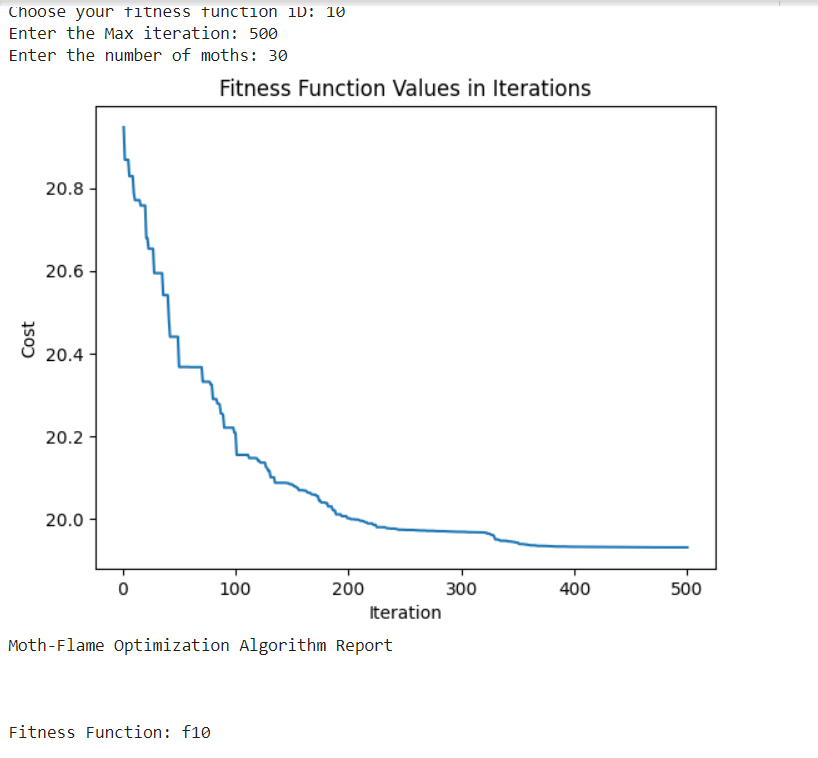
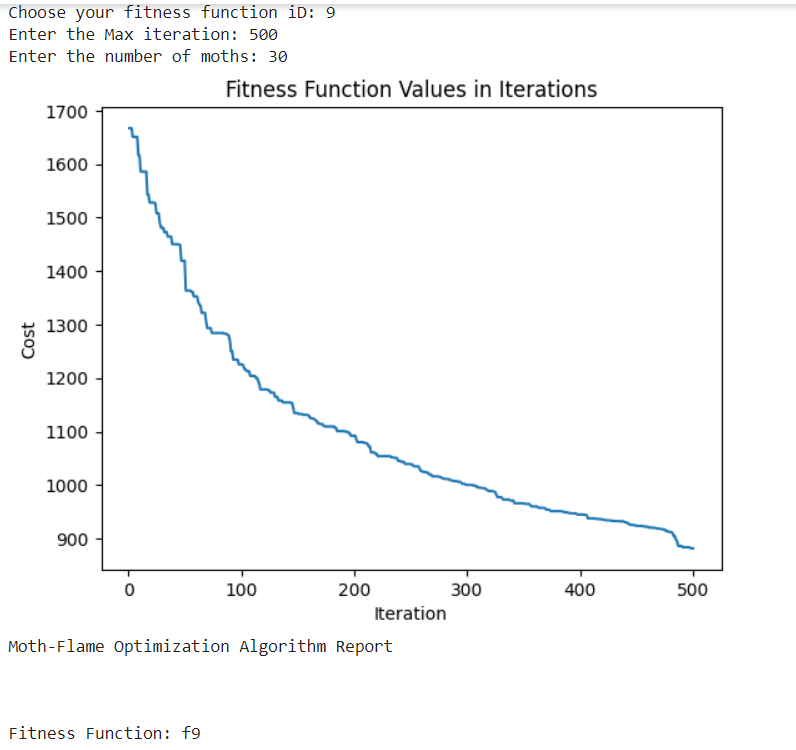
**Convergence Curves:**

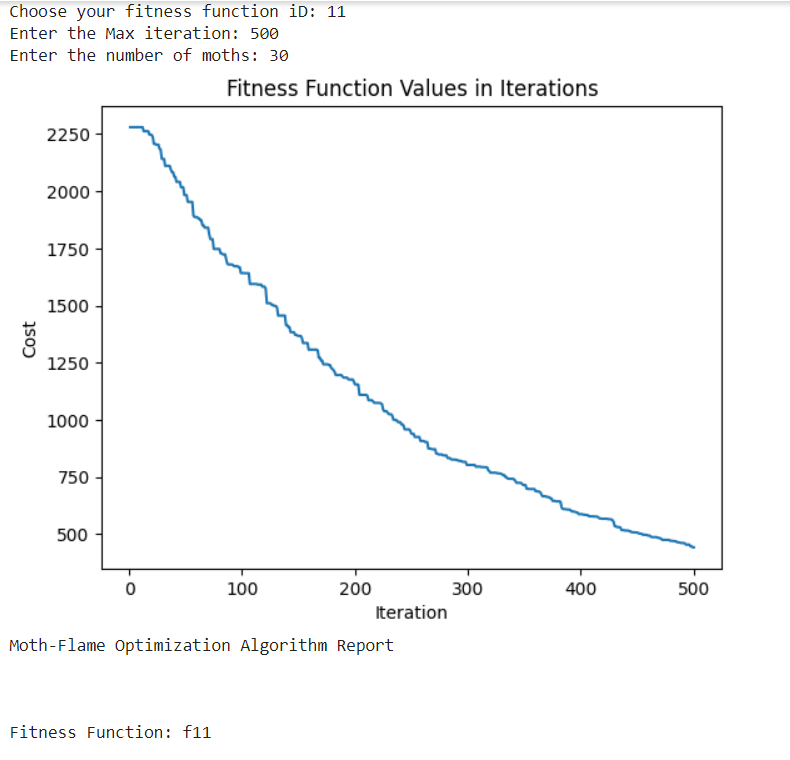
 





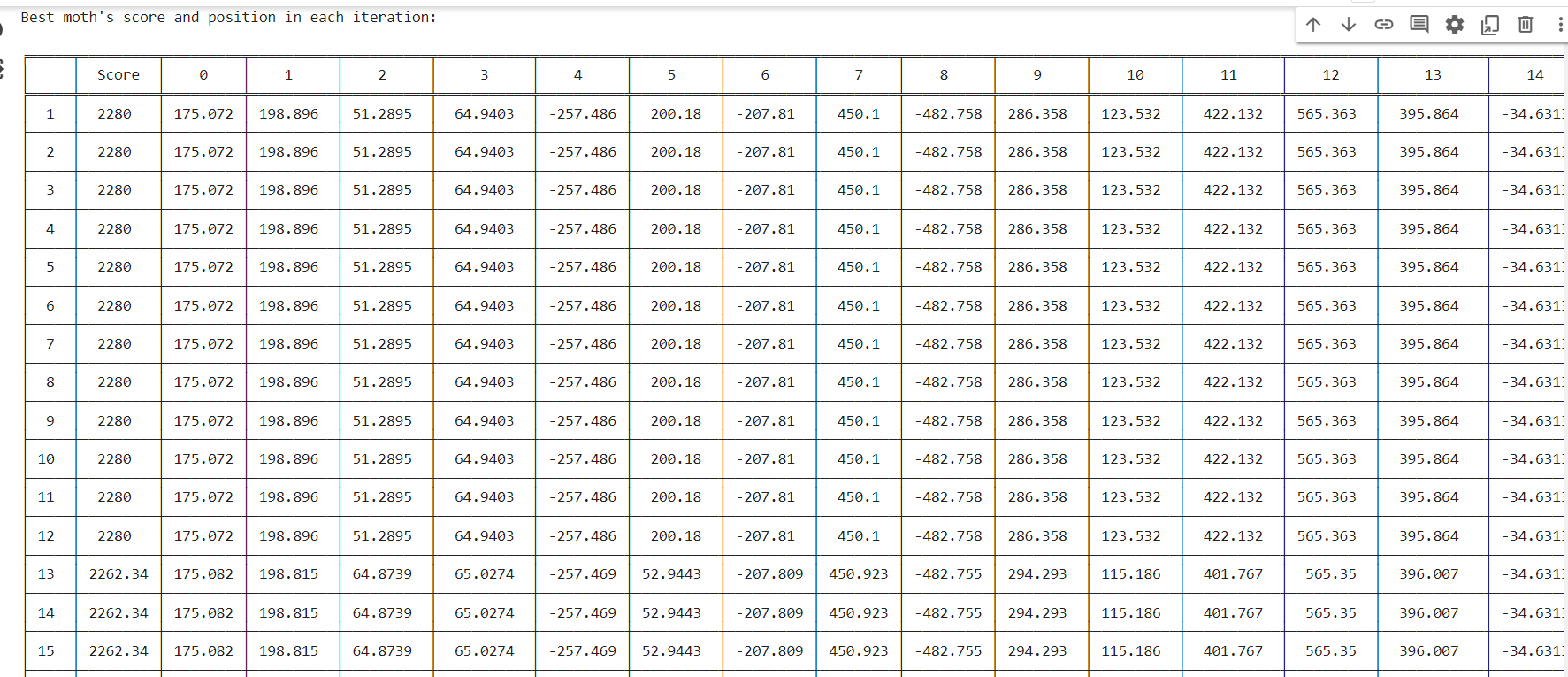
 





We see that amongst the unimodal functions, the MFO algorithm reached the optimal value the fastest with the f2 fitness function and amongst the multimodal functions, it reached the optimal value the fastest with the f10 fitness function. The convergence curves represent how gradually the moth moves towards the flames or light source.

In implementing this algorithm, we also generated a report with size dimensions\*iterations containing the fitness value for each iteration. This is a snippet of the table generated when we run the f11 function. As we can see the score at the 0’th iteration is 2280 and this value slowly decreases as it reaches the optimal value. The value for the scores is plotted in the graphs above.



**Conclusions**

In summary, this project implements and analyzes the Moth-Flame Optimization algorithm for solving optimization problems. It demonstrates how nature-inspired algorithms can be applied to solve real-world problems efficiently. The code is structured and well-documented, making it easy to understand and modify for optimization tasks.

**References**

Seyedali Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm", Knowledge-Based Systems, Volume 89, 2015, Pages 228-249, ISSN 0950-7051, <https://doi.org/10.1016/j.knosys.2015.07.006>