Fire Hawk Optimizer on Open Source Package

Adel REMADI, Aiza AVILA CAÑIVE, Michele Natacha ELA ESSOLA,
Vanshika SHARMA, Xingshan HE and Hamza LAMSAOUB

December 2022

Contents

1 Introduction

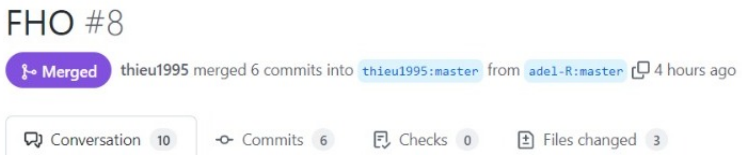
1.1 Selection of the problem

For the project presented here, we decided to collaborate with the open-source package MealPy, which is a package that develops nature-inspired meta-heuristic algorithms in order to implement them on applications such as Deep Learning.

The objective of our deliverable was to create a class in MealPy that would be able to implement the algorithm proposed by the paper "Fire Hawk Optimizer: a novel metaheuristic algorithm"[1]. There are several reasons that got us interested into doing this project. First of all, the opportunity to make an open-source contribution was new to all the members of the group and an exciting experience. Second, among all the proposed algorithms by MealPy's founder, FHO was the one that was published in Springer, a renown a global publisher that serves and supports the research community, notably in Data Science. In addition, although the initial article on FHO had a recent publication date (2022), it has already been cited several times in subsequent publications (despite being recent). Moreover, based on the Authors' extensive description, it seemed to outperform other meta-heuristic algorithms, which made it seem like a meaningful project to us. The Authors of the FHO article are also references in the Data Science field with high h-index, several publications and awards. The last reason of our choice was the intriguing fact that these Hawks exist in nature, in Australia's Northern Territory, and their most common way of hunting is by scaring the preys using fire.

The following report aims to explain what the algorithm is about, the implementation that was carried out in Python and finally a description of how the algorithm was tested on 5 functions in dimensions 2 and 25.

It is to be noted that the code produced by our group was submitted to the MealPy project. Our pull request was accepted, as the owner of the project was satisfied of our contribution: `https://github.com/thieu1995/metaheuristics/pull/8`.



1.2 Metahuristic Algorithms

First we will solve the question: what are metaheuristic algorithms? These are approximate general purpose search and optimization algorithms. They are iterative procedures that guide a subordinate heuristic by intelligently combining different concepts to adequately explore and exploit the search space. The metahuristic is born after having an optimization problem, which in first instance can be solved by two techniques: the exact and the approximate. The exact ones lead us to the optimal solution, which is not always viable (due to computational cost, amount of data, etc.). On the other hand, the approximate techniques do not guarantee reaching the best solution but do provide a solution close to the actual one.

Within this last approach, we have Ad-hoc Heuristics (approximate techniques that are developed specifically for a problem) and Metaheuristics (which are more general techniques that can be applied to different optimization problems). The latter are divided into trajectory-based (which rely on a single candidate solution per iteration) and population-based techniques, which consider multiple candidate solutions at the same time, and therefore allow different strategies to be applied. These strategies have been classified into three categories: "Evolutionary Algorithms", "Swarm Intelligence", and "Physics-Inspired Algorithms".

The algorithm of interest in this report is considered a "Swarm Based" algorithm.

2 Fire Hawks Optimization

2.1 Mathematical model

The algorithm that is implemented (FHO) tries to imitate the behavior of the birds in which it takes inspiration. Initially, a number of candidate solutions (X) are determined, these are the position vectors of the hawks and preys, followed by a random initialization process to identify the initial positions of these vectors on the search space.

$$\mathrm{X} = \begin{bmatrix} \mathrm{X}_1 \\ \mathrm{X}_2 \\ \vdots \\ \mathrm{X}_\mathrm{i} \\ \vdots \\ \mathrm{X}_\mathrm{N} \end{bmatrix} = \begin{bmatrix} \mathrm{x}_1^1 \mathrm{x}_1^2 \cdots \mathrm{x}_1^\mathrm{j} \cdots \mathrm{x}_1^\mathrm{d} \\ \mathrm{x}_2^1 \mathrm{x}_2^2 \cdots \mathrm{x}_2^\mathrm{j} \cdots \mathrm{x}_2^\mathrm{d} \\ \vdots : \vdots : \ddots : \vdots \\ \mathrm{x}_\mathrm{i}^1 \mathrm{x}_\mathrm{i}^2 \cdots \mathrm{x}_\mathrm{i}^\mathrm{j} \cdots \mathrm{x}_\mathrm{i}^\mathrm{d} \\ \vdots : \vdots : \ddots : \vdots \\ \mathrm{x}_\mathrm{N}^1 \mathrm{x}_\mathrm{N}^2 \cdots \mathrm{x}_\mathrm{N}^\mathrm{j} \cdots \mathrm{x}_\mathrm{N}^\mathrm{d} \end{bmatrix}, \begin{matrix} i = 1, 2, \ldots, N. \\ j = 1, 2, \ldots, d. \end{matrix} \tag{1}$$

$$\mathrm{x}_\mathrm{i}^\mathrm{j}(0) = \mathrm{x}_{\mathrm{i,min}}^\mathrm{j} + \mathrm{rand} \cdot \left(\mathrm{x}_{\mathrm{i,max}}^\mathrm{j} - \mathrm{x}_{\mathrm{i,min}}^\mathrm{j} \right), \begin{matrix} i = 1, 2, \ldots, N. \\ j = 1, 2, \ldots, d. \end{matrix} \tag{2}$$

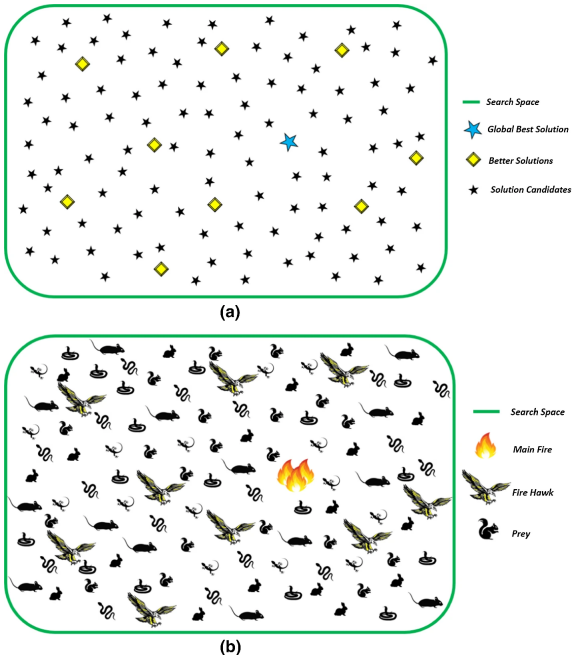
Where: X_i: represents the ith candidate solution in the search space. d: represents the dimension of the considered problem N:the total number of solution candidates x_i^j: the jth decision variable of the ith solution candidate $x_i^j(0)$: the initial position of the solution candidates $x_{i,min}^j$ and $x_{i,max}^j$: the minimum and maximum bounds of the jth decision variable for the ith solution candidate $rand$: a uniformly distributed random number in the range of [0,1].

To determine the locations of the hawks in the search space, the evaluation of the objective function considers some solutions as the hawks (the best solutions), while the rest are considered as "preys". While the best global solution is supposed to be the main "fire" that hawks use to hunt in the search space. This in mathematical representation would look like:

$$\mathrm{PR} = \begin{bmatrix} \mathrm{PR}_1 \\ \mathrm{PR}_2 \\ \vdots \\ \mathrm{PR}_\mathrm{k} \\ \vdots \\ \mathrm{PR}_\mathrm{m} \end{bmatrix}, k = 1, 2 \ldots, m. \tag{3}$$

$$\mathrm{FH} = \begin{bmatrix} \mathrm{FH}_1 \\ \mathrm{FH}_2 \\ \vdots \\ \mathrm{FH}_\mathrm{l} \\ \vdots \\ \mathrm{FH}_\mathrm{n} \end{bmatrix}, l = 1, 2 \ldots, n. \tag{4}$$

where: PR_k: is the kth prey in the search space regarding the total number of m preys FH_l: is the lth fire hawk considering a total number of n fire hawks in the search space.



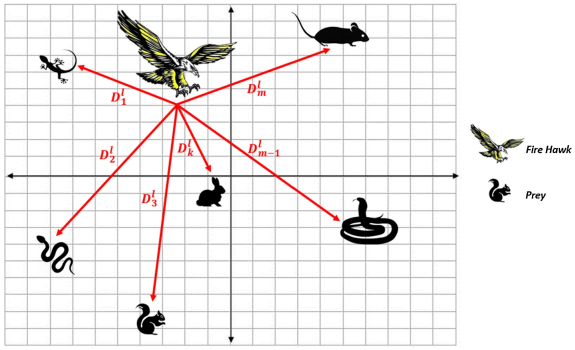
Next, the distance between the hawks and the prey is calculated by:

$$D_k^l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \begin{matrix} l = 1, 2, \ldots, n. \\ k = 1, 2, \ldots, m. \end{matrix} \tag{5}$$

It should be noted that the nearest prey to the first Fire Hawk with the best objective function value is determined, while the territory of the other birds are considered by means of the remaining prey

Where: D_k^l: is the total distance between the lth fire hawk and the kth prey m:is the total number of preys in the search space n: is the total number of fire hawks in the search space (x_1, y_1) and (x_2, y_2): represent the coordinates of the Fire hawks and preys in the search space.

Here is an illustrative representation:



Subsequently, the territory of the hawks is distinguished by means of the prey that is closest to them.
By classifying the hawks and preys, the search process is performed. Where the Fire Hawk with the best objective function value hunts the closest prey, and later the rest of the hawks proceed to hunt their closest prey, as seen in the image:

In the next stage, the hawks collect fire sticks to set the selected area on fire, forcing the prey to flee. And at the same time there are other birds that want to use the fire sticks of the hawks to also attack their territories. Both behaviors can be used as update procedures in the main FHO loop, as expressed in the following equation.

$$FH_l^{new} =, FH_l + (r_1 \times GB - r_2 \times FH_{Near}), l = 1, 2, \ldots, n. \tag{6}$$

where: FH_1^{new}: is the new position vector of the lth Fire Hawk (FH_l) GB: is the global best solution in the search space considered as the main fire FH_{Near}: is one of the other Fire Hawks in the search space r_1 and r_2: are uniformly distributed random numbers in the range of (0, 1) for determining the movements of Fire Hawks toward the main fire and the other Fire Hawks' territories

In the next phase of the algorithm, the movement of the prey inside the territory of each Fire Hawk is considered a key aspect of animal behavior for the position updating process. When a burning stick is dropped by a Fire Hawk, the prey decides to hide, run away, or will run towards the Fire Hawk by mistake. These actions can be considered in the position updating process by using the following equation:

$$PR_q^{new} = PR_q + (r_3 \times FH_l - r_4 \times SP_l, \begin{matrix} l = 1, 2, \ldots, n. \\ q = 1, 2, \ldots, r. \end{matrix} \tag{7}$$

Where: PR_q^{new}: is the new position vector of the qth prey (PR_q) surrounded by the lth Fire Hawk (FH_l) GB: is the global best solution in the search space considered as the main fire SP_l: is a safe place under the lth Fire Hawk territory r_3 and r_4 are uniformly distributed random numbers in the range of (0, 1) for determining the movements of the prey towards the Fire Hawks or the safe place.

In addition, the prey may move into the territory of other hawks, hide, or approach the hawk. And all these positions are location updates, which can be represented by:

$$PR_q^{new} = PR_q + (r_5 \times FH_{Alter} - r_6 \times SP, \quad \begin{matrix} l = 1, 2, \ldots, n. \\ q = 1, 2, \ldots, r. \end{matrix} \tag{8}$$

Where PR_q^{new}: is the new position vector of the qth prey (PR_q) surrounded by the lth fire hawk (FH_l) FH_{Alter}: is one of the other fire hawks in the search space SP: is a safe place outside the lth Fire Hawk's territory r_5 and r_6: are uniformly distributed random numbers in the range of (0, 1) for determining the movements of preys toward the other Fire Hawks and the safe place outside the territory.

The algorithm also takes into account that in nature, in order to be protected, animals must remain together, representing this with:

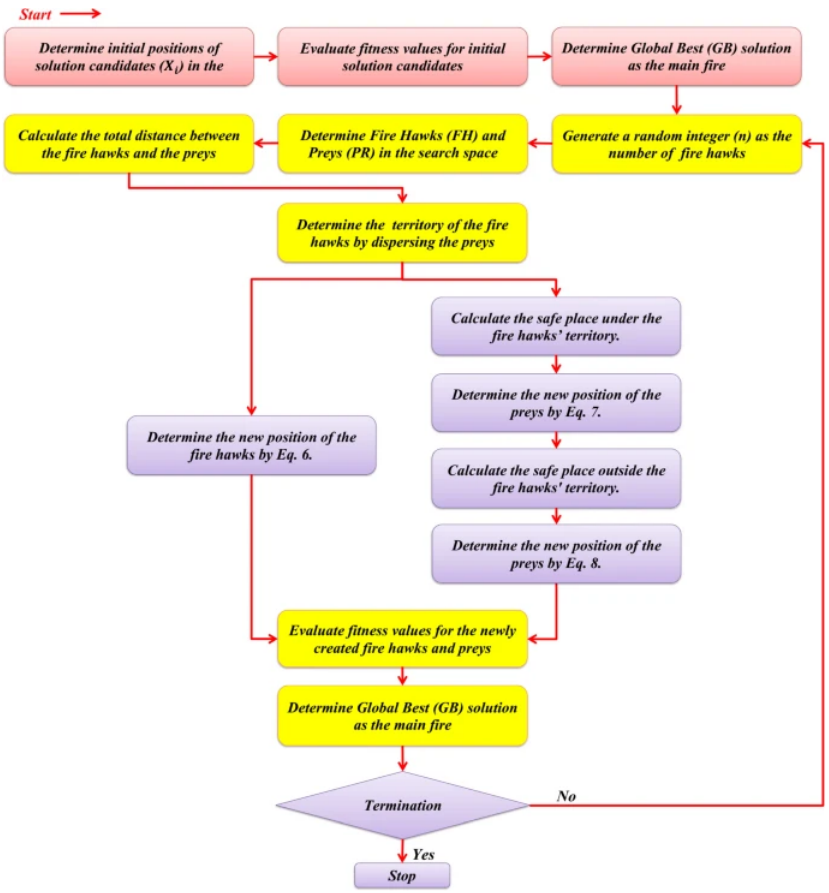
$$SP_l = \frac{\sum_{q=1}^{r} PR_q}{r}, \quad \begin{matrix} l = 1, 2, \ldots, n. \\ q = 1, 2, \ldots, r. \end{matrix} \tag{9}$$

$$SP = \frac{\sum_{k=1}^{m} PR_k}{m}, \quad k = 1, 2, \ldots, m \tag{10}$$

Where: PR_q: is the qth prey surrounded by the lth fire hawk (FH_l) PR_k: is the kth prey in the search space.

For this implementation, it is assumed that the territory of each hawk is circular, and the definition of the territories is dependent on the distances between the prey and the hawk in question. This is that when a prey is placed in the territory of a specific hawk, it is assumed that said prey will only be considered for the hawk of the territory in which it is. Therefore the number of preys and their distances from the hawks determine the limits of the hawks' territory. On the other hand, the possibility of the preys being outside their own territory is also considered in the position updating process regarding the fact that the preys should be affected by the fire hawks from other territories. The number of preys in each search loop is the total number of solution candidates minus the number of fire hawks determined randomly through the Brownian motion with a Gaussian distribution.

Here a more condensed flowchart of the process explained:



2.2 Implementation of the algorithm

For the implementation of the algorithm, a class called FHO was created. And within, different functions were defined to determine the territories, minimize the objective function of the hawks and plot the costs of the algorithm.

To define the territories, we followed the implementation described in the previous section. First, the number of preys available and the range of territories are defined depending on the number of fire hawks and the latters' distance to the best solution known at a given time. This was implemented in an helper function as follows:

```
def territories(self,Fire_Hawks,Preys):
#Computing territories using the euclidien distance
preys_left=Preys.copy()
territories={i:np.array([]) for i in range(len(Fire_Hawks))}
for i in range(len(Fire_Hawks)):
    #distance with respect to Fire hawk i
    D=np.linalg.norm(Fire_Hawks[i]-preys_left,axis=1)
    #Get territory of fire Hawk i
    sorted_preys_idx=np.argsort(D)
    alpha=np.random.randint(1,len(preys_left)-1) if len(preys_left)-1>1 else 1
    my_preys=sorted_preys_idx[:alpha]
    territories[i]=preys_left[my_preys]
    preys_left=preys_left[sorted_preys_idx[alpha:]]
    if len(preys_left)==0:
        break
if len(preys_left)>0:
    territories[len(Fire_Hawks)-1]=np.array(list(territories[len(Fire_Hawks)-1])+list(preys_left))
return territories
```

However, the central function within the class is the one that is in charge of performing the minimization of the cost function, and it has been called minimize_FHO. It takes as parameters the minimum and maximum bounds for each dimensions in which the function will be evaluated (any mathematical funtion), the desired number of solution candidates, the cost function to be optimized and the maximum number of function evaluations to perform over all iterations.

It is to be noted that the below code presents a modification compared to the initial algorithm. This modification has been commented in the code and allowed the algorithm to converge faster and more reliably in high dimensions.

```
def minimize_FHO(self):
## Fire hawk algorithm to minimize the cost function
n_dims=self.n_dims
Pop=self.Pop
pop_size=self.pop_size
max_generations = self.max_generations
min_bounds=self.min_bounds
max_bounds=self.max_bounds
cost_function = self.cost_function
#Evaluate the cost function for all candidate vectors
cost= np.array([cost_function(Pop[i]) for i in range(pop_size)])
#Randomly set a number of Hawks between 1 and 20% of pop_size
num_Hawks = np.random.randint(1,int(pop_size/5)+1) if 1<int(pop_size/5)+1 else 1
#Ordering candidates
Pop = Pop[np.argsort(cost)]
cost.sort()
SP=Pop.mean(axis=0)
#Select fire hawks
Fire_Hawks= Pop[:num_Hawks]
#Select the Preys dim(pop_size-num_Hawks,n_dims)
Preys = Pop[num_Hawks:]
#get territories
territories=self.territories(Fire_Hawks,Preys)
#update best
GB=cost[0]
Best_Hawk=Pop[0]
self.path.append(Best_Hawk)
#Counter
FEs=pop_size
```

```
## Main Loop
while FEs < max_generations:
    Pop_Tot=[]
    cost=[]
    #Movement of Fire Hawk for all territories
    for i in territories:
        PR=territories[i].copy()
        FHl=Fire_Hawks[i].copy()
        SPl=PR.mean(axis=0) if len(territories[i]) > 0 else np.zeros(FHl.shape)
        a,b=np.random.uniform(0,1,size=2)
        FHnear  =Fire_Hawks[np.random.randint(num_Hawks)]
        FHl_new =FHl+(a*GB-b*FHnear)
        FHl_new = np.maximum(FHl_new,min_bounds)
        FHl_new = np.minimum(FHl_new,max_bounds)
        Pop_Tot.append(list(FHl_new))
        #Movement of the preys following Fire Hawks movement
        for q in range(len(PR)):
            a,b=np.random.uniform(0,1,size=2)
            PRq_new1=PR[q].copy()+((a*FHl-b*SPl))
            PRq_new1= np.maximum(PRq_new1,min_bounds)
            PRq_new1 = np.minimum(PRq_new1,max_bounds)
            Pop_Tot.append(list(PRq_new1))
            #Movement of the preys outside of territory
            a,b      =np.random.uniform(0,1,size=2)
            FHAlter  =Fire_Hawks[np.random.randint(num_Hawks)]
            PRq_new2 =PR[q].copy()+((a*FHAlter-b*SP));
            PRq_new2 = np.maximum(PRq_new2,min_bounds)
            #The following line for PRq_new2 differs from original algorithm in matlab code
            # (max instead of min):
            # Effects observed through our testing:
            # 1/ It converges faster and the costs of the subsequent iterations will tend to
            # decrease (less chaotic behavior than with np.minimun)
            # 2/ In higher dimensions, it converge to the right solution! (while not np.minimum)
            PRq_new2 = np.maximum(PRq_new2,max_bounds)
            Pop_Tot.append(list(PRq_new2))
    Pop_Tot=np.array(Pop_Tot)
    for i in range(len(Pop_Tot)):
        cost.append(cost_function(Pop_Tot[i]))      #Get costs
        FEs = FEs+1
    #Create a new population of Hawks and Preys
    order_idx=np.argsort(cost)
    cost.sort()
    Pop_Tot=np.array(Pop_Tot)[order_idx]
    num_Hawks = np.random.randint(1,int(pop_size/5)+1) if 1<int(pop_size/5)+1 else 1
    Best_Pop=Pop_Tot[0]
    SP=Pop_Tot.mean(axis=0)
    Fire_Hawks=Pop_Tot[:num_Hawks]
    Preys=Pop_Tot[num_Hawks:]
    #Get new territories
    territories=self.territories(Fire_Hawks,Preys)
    # Update Global Best cost (if relevant)
    if cost[0]<GB:
        Best_Position=Best_Pop
        GB=cost[0]
        self.best_costs.append(GB)
        self.minimal_p=Fire_Hawks[0]
        self.path.append(Best_Position)
    else:
        self.best_costs.append(GB)
    #Track the iteration calculated cost
    self.costs_iter.append(cost[0])
#Return Global Best and argmin
return (GB,self.minimal_p)
```

2.3 Results and performance

In order to evaluate the algorithm, we tested it with different functions in multiple dimensions. Namely we tested the algorithm on :

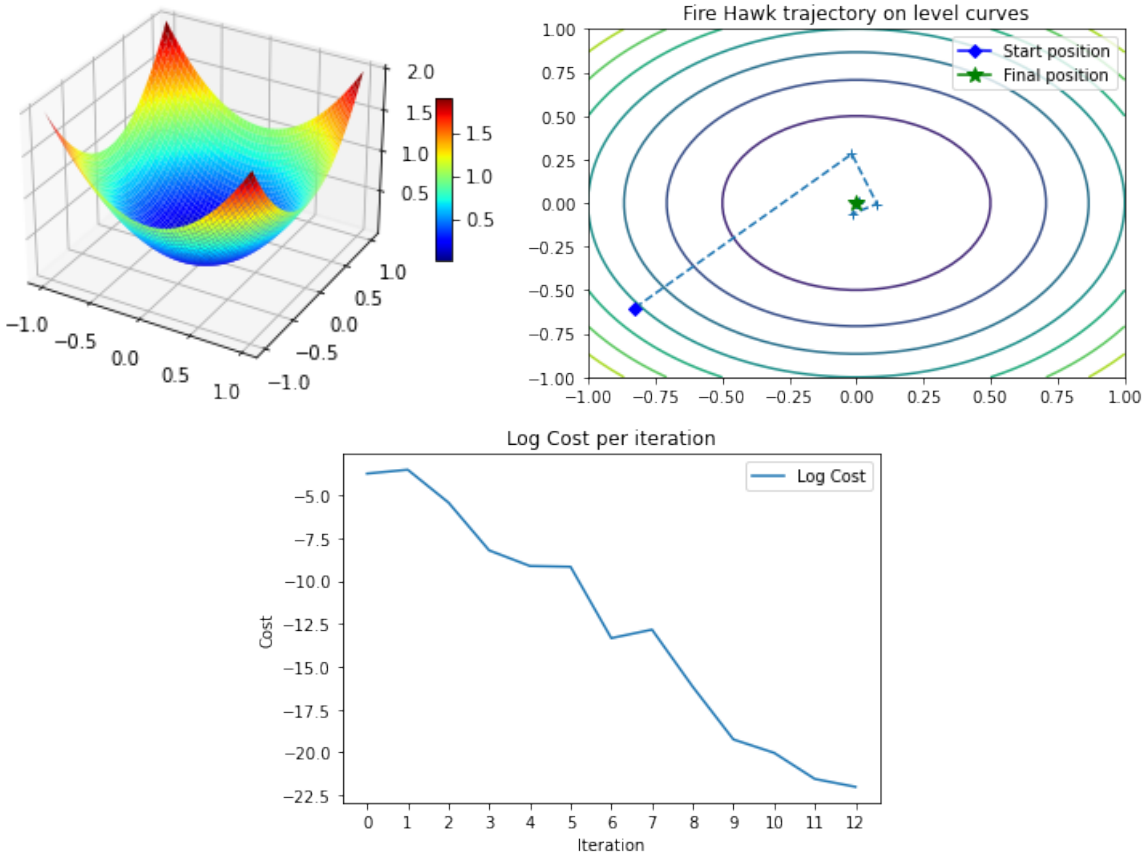
- Sphere function (2D and 25D) : Continuous, Differentiable, Separable, Scalable
- Exponential function (2D and 25D): Continuous, Differentiable, Non-Separable, Scalable, Multimodal
- Ackley 1 function (2D and 25D): Continuous, Differentiable, Non-Separable, Scalable, Multimodal
- Becker-lago function (2D and 25D): Separable
- Bird function (2D and 25D): Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal

2.3.1 Evaluation on a Sphere function

The sphere function is a convex problem and has therefore a unique global minimum.

Analysis in 2 dimensions

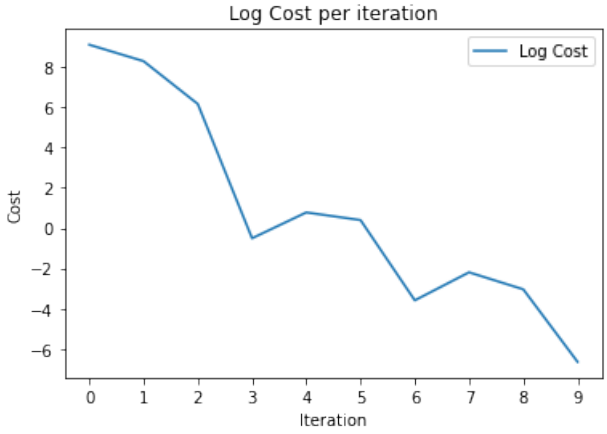
- Cost function:$f(x) = x_1^2 + x_2^2$
- Global minimum reached in 4.5517289560593654e-08
- Evaluated on [-2.09280659e-04 -4.14595622e-05]



We can observe a practically linear convergence on the side of the log cost function graph.

Analysis in 25 dimensions

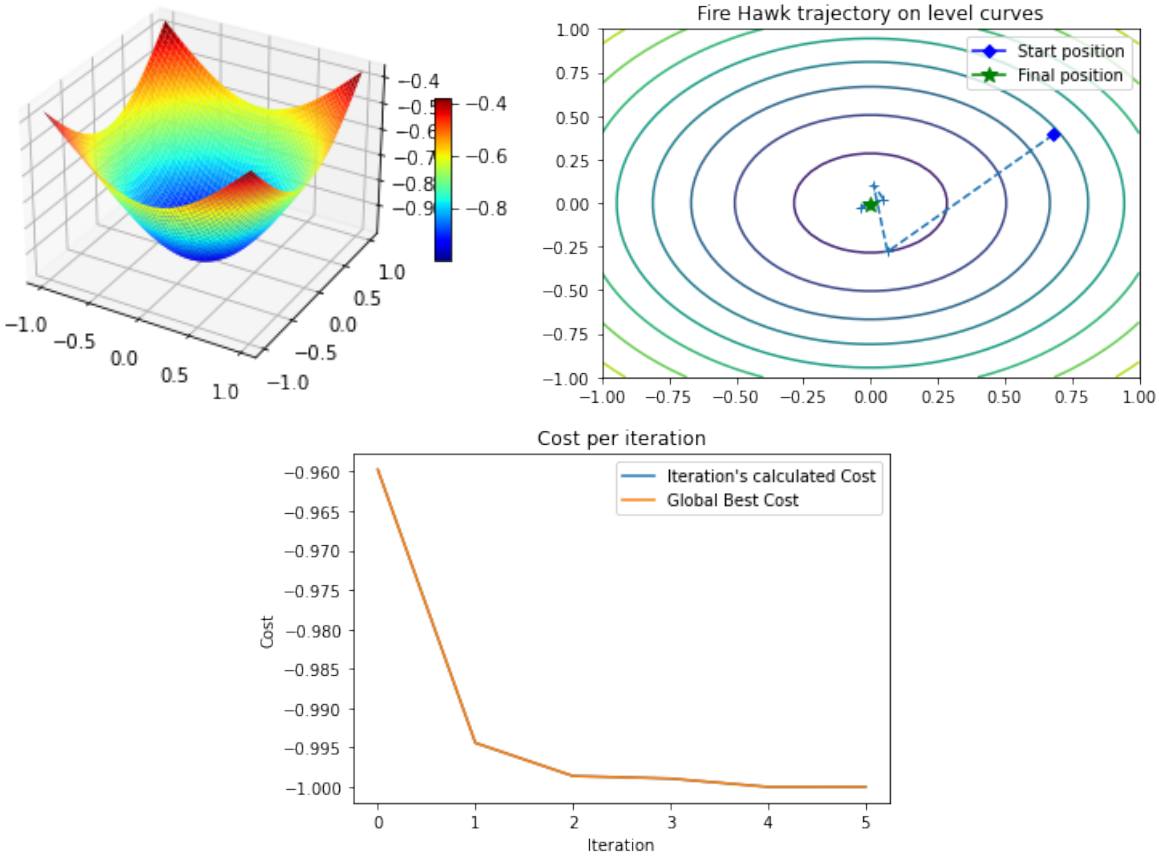
- In 25-D, the algorithm also converges well and provides an approximation of the global minimum of 0.001301. The minimal point, a 25-D vector, found by the algorithm has all its components in the order 10^{-2} or 10^{-3} at most.



2.3.2 Evaluation on the Exponential function

Analysis in 2 dimensions

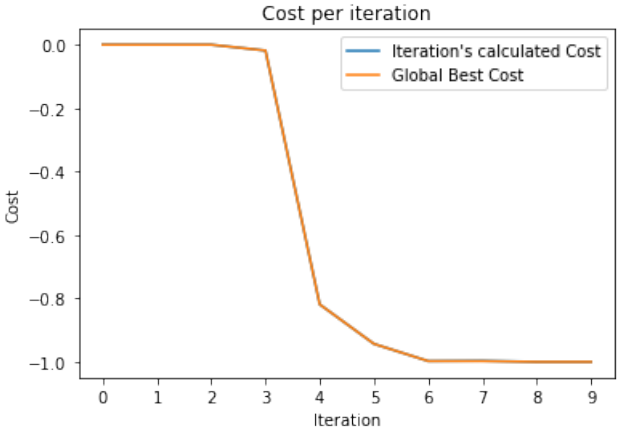
- Cost function: $f(x) = e^{\frac{-1}{2}(x_1^2+x_2^2)}$
- Global minimum reached in -0.999
- Evaluated on [-0.00365899 -0.00485181]



We can observe a good behavior of the algorithm to converge towards the optimum point.

Analysis in 25 dimensions

- In 25-D, the algorithm also converges well and provides an approximation of the global minimum of -0.9994. Similarly to the Sphere's case, all the components of the minimal point are in the order 10^{-2} or 10^{-3} at most.



2.3.3 Evaluation on the Ackley 1 function

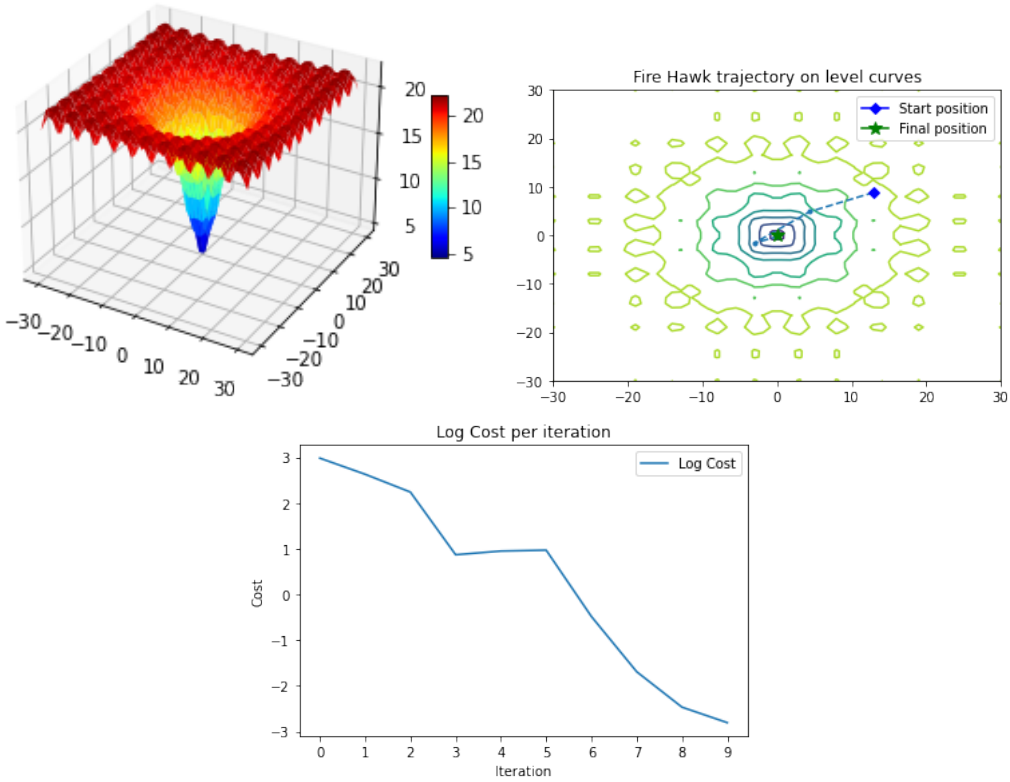
Now, to make it a challenge, we tested the "Fire Hawks Optimizer" on more challenging functions, including the Ackley 1 function, defined as follows in 2-D:

$$f(x,y) = -20(exp[-0.2\sqrt{0.5(x^2+y^2)}] - exp[0.5(cos2\pi x + cos2\pi y)] + e + 20 \tag{11}$$

The theoretical global optimum point is reached in $f(0,0) = 0$, but the present problem is highly non convex and present multiple-local minima. This could notably represent a challenge to gradient based algorithms.

Analysis in 2 dimensions

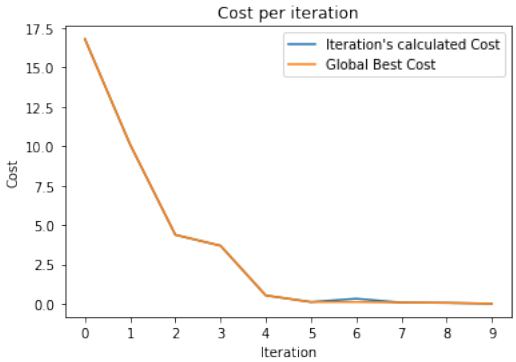
- Global minimum approximation: 0.0283
- Minimal point: [-0.00651668 -0.00651668]



In this challenging case, the algorithm still managed to arrive to a stable result extremely close to the global minimum, even after tweaking the minimum and maximum bounds to increase the likelihood of a far away initialization.

Analysis in 25 dimensions

- Global minimum reached in 0.01744
- Evaluated on [6.56011395e-03, 5.59099587e-04,...,1.15676186e-03]



In 25 dimensions, all the entries of the minimum point approximation are really close to zero, and the cost function plot shows a quick convergence.

2.3.4 Evaluation on the Becker-lago

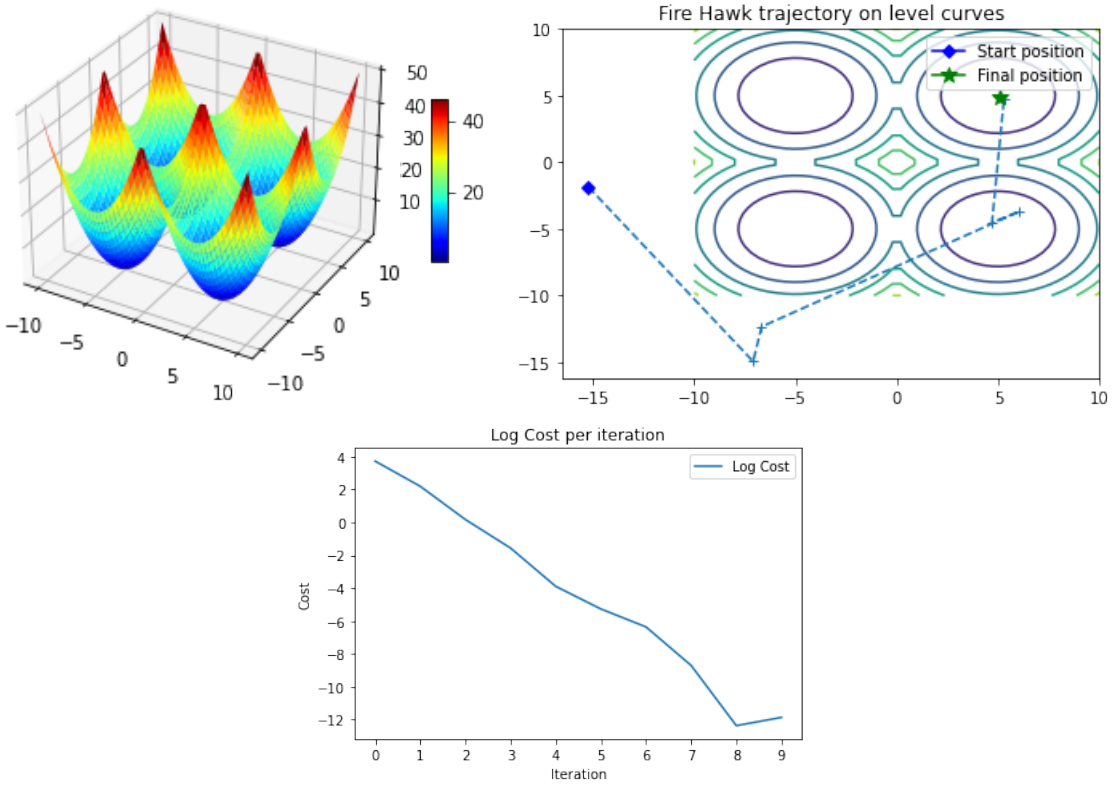
The Becker-Lago function has the following expression:

$$f(X) = (\mid x \mid -5)^2 + (\mid y \mid -5)^2 \tag{12}$$

It has global minimum $f(x, y) = 0$ when $(x, y) \in \{(-5, -5), (-5, 5), (5, -5), (5, 5)\}$

Analysis in 2 dimensions

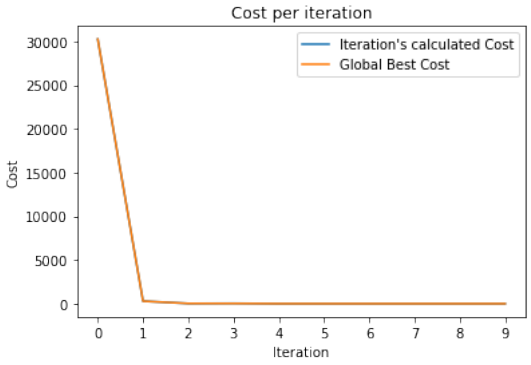
- Global minimum reached is 4.255e-06
- Evaluated on [-5.00199164 -5.00053723]



Again the algorithm provide a good approximation and converges well.

Analysis in 25 dimensions

- Global minimum reached in 0.00098
- Evaluated on [5.00697284, 5.00577378,..., 5.006105]



The convergence and approximation provided by the algorithm stay good in 25 Dimensions.

2.3.5 Evaluation on Bird function

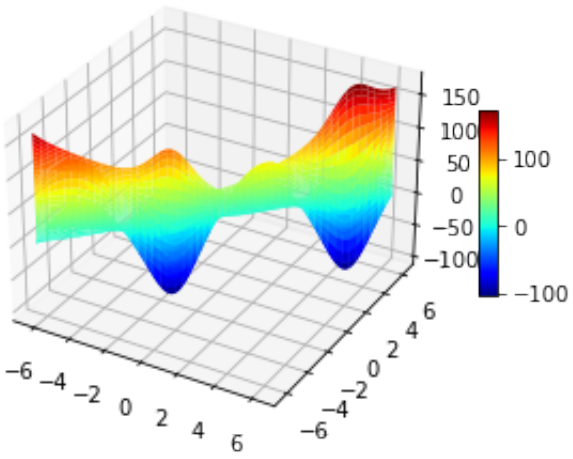
The Bird function has the following expression:

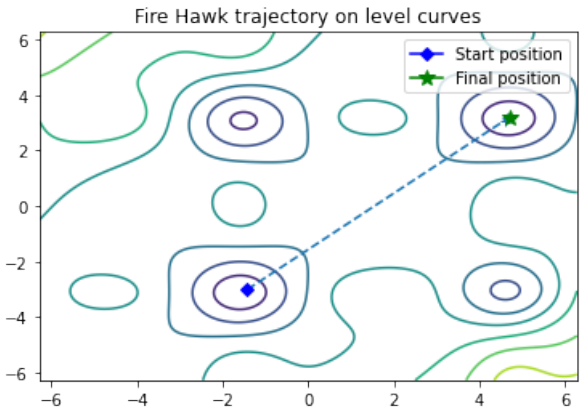
$$f(x,y) = sin(x)\exp(1-cos(y))^2 + cos(y\exp(1-sin(x))^2 + (x-y)^2 \tag{13}$$

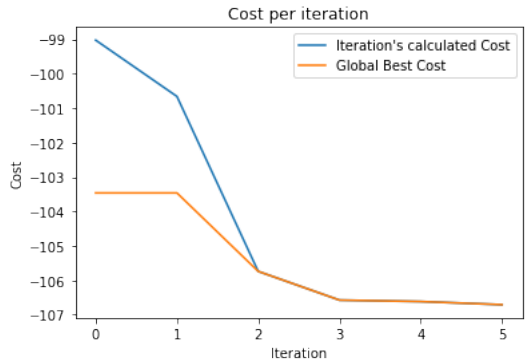
It has two global minima $f(x,y) = -106.764537$ in the interval $[-2\pi, 2\pi]$ at (4.70104 ,3.15294) and (-1.58214 ,-3.13024).

Analysis in 2 dimensions

- Global minimum reached in -106.70790
- Evaluated on [4.72360977 3.16883743]







For this function, the algorithm also displayed consistent results.

3 Contributions of each member

Adel REMADI

- Searched for issues on Github to discuss with the team
- Opened communication with mealPy developer anduploaded the pull request
- Wrote/implemented the code for the Fire Hawk implementation, letting the team know about the issues and findings
- Reviewed the Fire Hawks report and added complementary insights

Aiza AVILA CAÑVE

- Researched literature and the COCO stucture for first approach of comparing Bayes opti and GridSearch optimizers using COCO (this implementation was discarded, since results weren't satisfactory)
- Implemented pseudo-code for COCO comparison between Bayes optimization and GridSearch (this implementation was discarded, results weren't satisfactory)
- Researched the literature regarding the Fire Hawks Optimizer implementation
- Wrote down the report for the Fire Haws Optimizer implementation

Vanshika SHARMA

- Searched for issues on Github to discuss with the team
- Researched literature for Bayes optimizer to apport structure to the comparison with COCO (this implementation was discarded, results weren't satisfactory)
- Researched on the GMM implementation, had open conversations with the developer on Github and uploaded the request
- Implemented 1/2 of the code for the GMM implementation

Michel Natacha ELA ESSOLA

- Searched for issues on Github to discuss with the team
- Researched literature for GridSearch optimizer to apport structure to the comparison with COCO (this implementation was discarded, results weren't satisfactory)
- Researched literature for GMM optimizer implementation
- Wrote down the report for the GMM implementation

Xingshan HE

- Researched literature and the COCO stucture for first approach of comparing Bayes opti and GridSearch optimizers using COCO (this implementation was discarded, since results weren't satisfactory)
- Implemented pseudo-code for COCO comparison between Bayes optimization and GridSearch (this implementation was discarded, results weren't satisfactory)
- Research the literature regarding the GMM implementation
- Implemented 1/2 of the code for the GMM implementation

Hamza LAMSAOUB

- Searched for issues on Github to discuss with the team
- Researched the literature regarding the Fire Hawks Optimizer
- Wrote/implemented the code for the Fire Hawk implementation, letting the team know about the issues and findings
- Reviewed the Fire Hawks report and added complementary insights

References

[1] Siamak Talatahari Amir H. Gandomi Mahdi Azizi. "Fire Hawk Optimizer: a novel metaheuristic algorithm". In: (June 2022). URL: https://link.springer.com/article/10.1007/s10462-022-10173-w#Sec18.