Data Clustering with Elephant Herding Optimization

Vincent Graham

Abstract

Clustering is a data analysis task that groups objects using a similarity measure. In this paper, I will present an overview of a clustering algorithm based on the herd structure and behavior of the African Savanna Elephant. This will be done using biological analogy and several derivations from other iterative optimization strategies. A gradient-based version of elephant herding optimization will also be discussed and compared against the original on several data sets. A more detailed example is shown for the Fisher Iris data set. This paper will provide a MATLAB implementation of both algorithms.

I. Introduction

Natural computation encompasses three classes of methods: methods which take advantage of patterns in nature; methods that synthesize natural phenomena; and methods that use natural elements to perform computation [6]. It exists at the intersection of natural sciences, computer science, mathematics, and many other fields.

Metaheuristic algorithms grew greatly in popularity during the period from 2005 to 2015, as found by Hussain et al. in 2018 [15]. 2015 saw the largest number of publications. Practical applications contributing to this growth included job scheduling, electronics design, and communications. A plurality of metaheuristics are described by biological metaphor and among them, particle swarm optimization is the most common. Unfortunately, some metaheuristic algorithms are not guaranteed to converge either locally or globally.

Bio-inspired computing, a sub-field of natural computation, includes topics such as artificial neural networks, swarm intelligence, genetic and evolutionary algorithms [7]. The biological behavior of the inspiration is often simplified to decrease computational costs. These methods use only a few key behaviors when building a model. As an example, take emperor penguin optimization introduced by Vidhya and Kumar in [19]. Their model is designed around the position and temperature of the huddle penguins form to keep warm.

II. Particle Swarm Optimization

Advantages of swarm intelligence methods are their speed, parallelizability, and stability [7]. Particle swarm optimization (PSO) was first proposed by Russell Eberhart and James Kennedy in [10]. It has roots in artificial life, in particular swarming theory, and evolutionary programming. The behavior of particles is governed by five principles [10], [20].

1) Proximity: space and time calculations should be simple. A minimal amount of computation should be required for a particle to find its next location.
2) Quality: the population should be able to respond to qualitative properties of their environment such as the nutritional value of nearby food sources.
3) Diverse response: the population should be able to adapt the sudden changes in the environment but not over commit down narrow lines.
4) Stability: not every change in the environment should produce a change in the population's behavior. Some changes in behavior do not result higher rewards.
5) Adaptability: if the rewards of a behavior change outweigh the computational cost, then the population should change its behavior.

Particle swarm optimization follows these five principles [10, p. 1]. A *swarm* is a collection of *particles*, each of which represent a possible solution to a given problem. The particles exist in the *search space* or solution space denoted as Ω. One such problem could be

$$f : \Omega \times A \to \mathbb{R}$$

which maps a particle $x \in \Omega$ and some information relevant to the specific problem in A to a real number. If f only depends on the position of the particle x then $\Omega = \operatorname{dom} f$; the search space and the domain of f are the same. The value of $f(x)$ is used to evaluate the particle x.

Particles are defined by their coordinates in the search space and updates to their position are performed iteratively. They are also able to remember their own previous positions. An iteration, denoted by t, represents the state of the swarm at a given time, or in a *generation* in biologically inspired models. The particles in a swarm are not necessarily unique because, if convergence is to be achieved then all particles will eventually have the same position.

Two positions, p_{best}, and g_{best}, form the qualitative measures that are stored. They can be though of as a personal-best and global-best. Each particle has its own p_{best} that is p_{best} is the best position that the particle has ever been in. Sometimes, several of the top p_{best}, which are called informants, are used to influence the particles position but only one g_{best} exists for the swarm. p_{best} can be any position in the search space; it doesn't have to be currently occupied by a particle. The particle also has a position called l_{best} (local best) which is often implemented as either the p_{best} or g_{best} depending on the desired topology of the problem. If p_{best} is used, then it is assumed that the particles do not jump around the search space too much. PSO algorithms that use multiple p_{best} positions as their local l_{best} are called lbest PSO.

The best solution that has been found by any particle at any generation is recorded as g_{best}. Like p_{best}, the g_{best} position also does not need a particle to currently exist at it. Changes to these values affect behavior of the entire population, while updates to particles other than the three best positions during the current generation do not.

To find the best solutions corresponding to a given particle x_n, the p_{best} can be thought of as a function. Suppose that F is the fitness function. If, at time t, every position x_n's history is currently stored in the array $PB_n = \{pb_{n,1}, pb_{n,2}, \ldots, pb_{nt} : f(pb_{n,i}) \leq f(pb_{n,i+1})\}$, then

$$p_{best} : \Omega \to PB_n, \qquad p_{best}(x_n) = pb_{n,1} \tag{1}$$

In the original PSO algorithm, particle motion is updated by a velocity term [10]. The equations used for particle x_n are

$$v_n^{(t+1)} = c_1 v_n^{(t)} + 2r_1 c_2 \left(pb_{n,1} - x_n^{(t)}\right) + 2r_2 c_3 \left(g_{best} - x_n^{(t)}\right) \tag{2}$$

$$x_n^{(t+1)} = x_n^{(t)} + v_n^{(t+1)} \tag{3}$$

where $r_i \in [0, 1]$ are uniform random variables and c_1, c_2, c_3 are parameters that can be tuned. Each particle is attracted towards both its own best position (the middle term) as well as the global best position (the last term), while keeping its previous velocity $v_n^{(t)}$. The algorithm continues until a set number of iterations is reached.

Particle swarm optimization produces many solution candidates which all seek to approximate an minimum or maximum of a function, called a *fitness function* in this paper. Fitness draws upon the biological analogy of evaluating the health of particles as how good of a solution they are. The fitness function is evaluated for each particle and each particle is then assigned a *fitness*, or *score*.

A. Minimization Example

As an example of a fitness function, suppose we are minimizing the one-dimensional function [5]

$$h(x) := \left[\left(|x|^2 - 1\right)^2\right]^{\frac{1}{8}} + \left(\frac{1}{2}x^2 + x\right) + \frac{1}{2} \tag{4}$$

By construction, the minimum is achieved at $x = -1$. In this example, a particle is in $\mathbb{R}^1$ and its position is determined by its x-coordinate. Since there is only one variable to consider in h, the search space Ω is the same as $\operatorname{dom} h$.

In Figure 1, the particles have been plotted as $(x, h(x))$ but, importantly, the $h(x)$ coordinate is not part of the particle's position. In this one dimensional example, particles are able to overcome a local minimum around $x = 1$ and eventually reach the solution of $x = -1$. Additionally, the particles have formed a tighter swarm as a result of moving towards the best solutions at each iteration. After 50 iterations, the global best solution, g_{best}, is $x = -1$. When the maximum iteration number has been reached, g_{best} is selected as the best solution and returned out of the algorithm.

In two dimensions, the particles can be plotted as along their component axes x_1, x_2. Using the 2D version of the function in Eq. (4) [5],

$$H(\mathbf{x}) = \left[\left(\|\mathbf{x}\|^2 - 2\right)^2\right]^{\frac{1}{8}} + \frac{1}{2}\left(\|\mathbf{x}\|^2 + x_1 + x_2\right) + \frac{1}{2}$$

The search space is $\Omega = \mathbb{R}^2 = \operatorname{dom} H$. For visualization purposes, the particles are plotted on top of the contour plot of H in Figure 2.

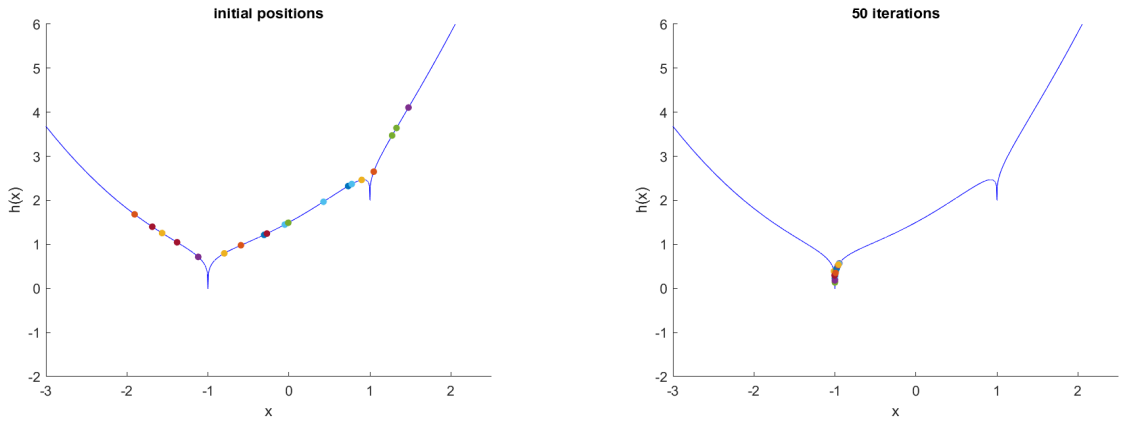

Figure 1: PSO [3] was run to minimize h using 25 particles. The particles, shown as dots, approach the global minimum as the number of iterations increases.

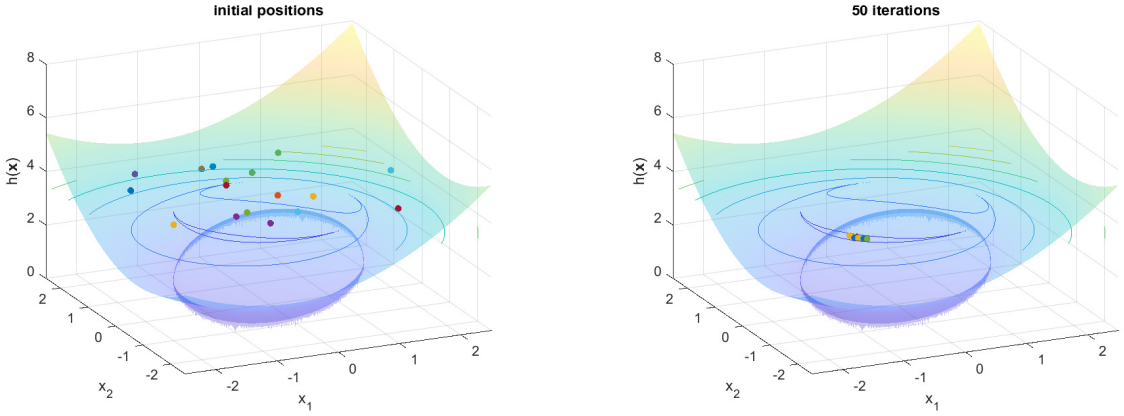

Figure 2: Minimizing H using 25 two-dimensional particles with the algorithm in [3]. The particle position is plotted in the x_1, x_2 plane and is shown on top of the contour plot of H.

III. Clustering

Clustering is an unsupervised data analysis task which detects existing structures in the data which give rise to clusters. A cluster is a collection containing data with similar characteristics. Particle swarm optimization is able to be applied to clustering tasks. In this paper, the number of clusters we are looking for, k, is a pre-selected parameter.

A data set $\mathbb{X}$ contains data *points* x_j with each point having d dimensions. For an array X of n real valued, multidimensional data points, we have $X \subseteq \mathbb{X}$, $X \in \mathbb{R}^{n\times d}$, and each $x_j \in \mathbb{R}^d$. The set $\mathbb{R}^d$ can be called the *data space*. When data is incorporated into PSO, the fitness function is $f : \Omega \times \mathbb{R}^{n\times d} \to \mathbb{R}^n$. Given a particle p, f is defined by $f(p, X) = s$ for some $s \in \mathbb{R}$.

In centroid based clustering, there are several *centroids* in the data space; $\mathsf{c}_\ell \in \mathbb{R}^n$. These centroids are positions and are not necessarily in the data set. Data points are assigned to the closest centroid under a distance function, $D(x, \mathsf{c}_\ell)$.

The centroid c_ℓ is the center of the cluster $\mathbf{C}_\ell$ to which the points have been assigned. A similar calculation can be done in the reverse order. If we know a cluster $\mathbf{C}_\ell$, the centroid is the average position of the data points assigned to $\mathbf{C}_\ell$. The symbol $\in_C$ denotes a cluster assignment relation and $|\mathsf{C}_\ell|$ is the number of points assigned to the cluster.

$$\mathsf{c}_\ell = \frac{1}{|\mathbf{C}_\ell|} \sum_{x \in_C \mathbf{C}_\ell} x$$

A particle represents k centroids, so for data in $\mathbb{R}^d$, the search space is $\Omega = \mathbb{R}^{k\times d}$. Suppose we are clustering a one dimensional data set and we assume that there 2 underlying clusters. The search space is $\mathbb{R}^{2\times 1}$ and the data space

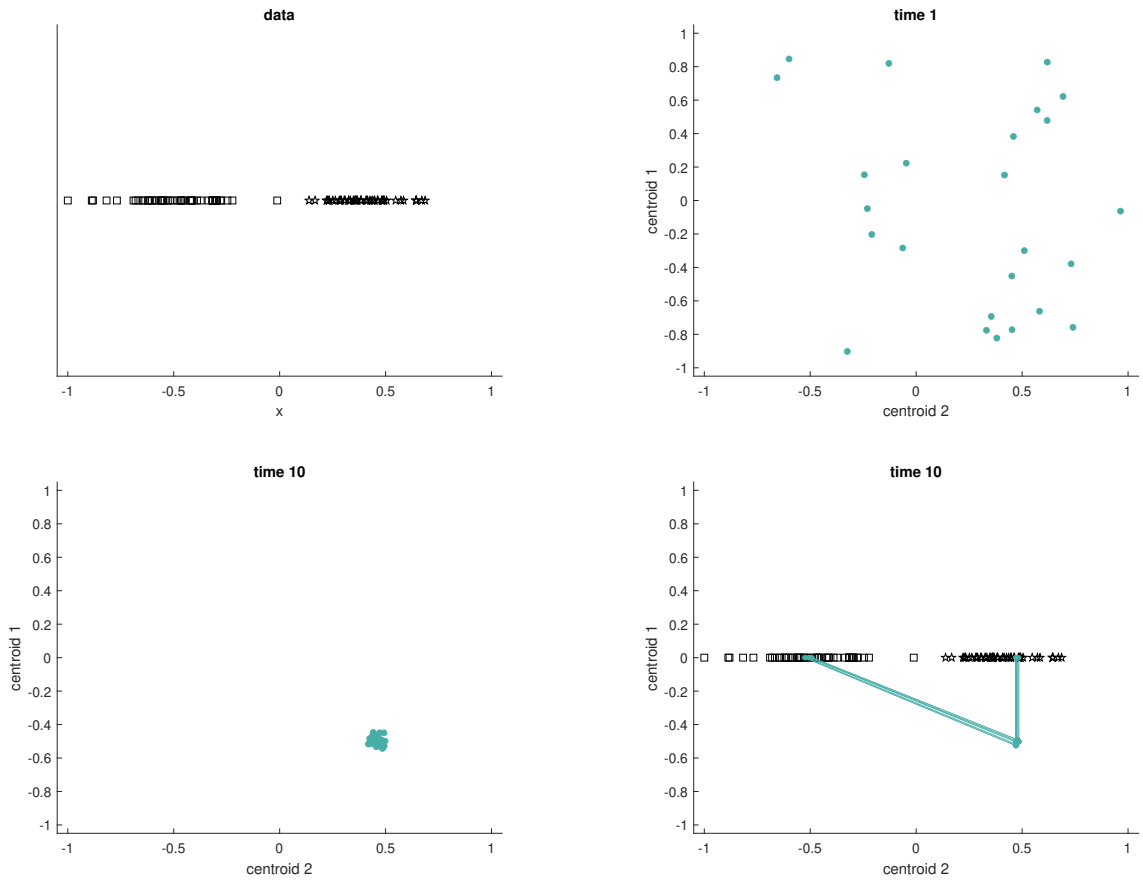

Figure 3: 2D particles coordinates in the search space correspond to the x coordinate of the two centroids in the 1D data set in the top left figure. The initial particle positions are shown in the top right. After 10 iterations, the swarm has become much tighter. In the bottom right, the data is overlayed so a line can be drawn between the particle location in the search space and the corresponding centroid locations in the data space. Only 5 particles from the swarm are plotted in the bottom right image.

is $\mathbb{R}^1$. Since the dimensionality of the search space is low, the particles can be plotted more easily. Their position and the data is shown in Figure 3.

A. *Cluster Evaluation*

Once we have found some centroids, we can score their clustering with a fitness function. One such function is the sum of squared errors, calculated with $D(x, y) = \|x - y\|_2^2$, which measures within-cluster variance. The function is defined as

$$\min_{\mathsf{c}_1,\ldots,\mathsf{c}_k} SSE = \min_{\mathsf{c}_1,\ldots,\ \mathsf{c}_k} \sum_{\ell=1}^{k} \sum_{x \in \mathbf{C}_\ell} D(x, \mathsf{c}_\ell)^2 \tag{5}$$

where $x \in \mathbf{C}_\ell$ is all data points assigned to cluster ℓ. If the within-cluster variance is zero, every point in the cluster is identical to the center. We seek to minimize the SSE to find the tightest clustering.

Another method, proposed by Rousseeuw [22], is to construct a silhouette of the clusters. Given a clustering $\mathbf{C}$ and a point x_j, we compute two dissimilarity values using the metric $D(x, y) = \|x - y\|_2^2$. The first is within the cluster, $x_j \in \mathbf{C}_i$

$$a_j = \frac{1}{|\mathbf{C}_i| - 1} \sum_{y \in \mathbf{C}_i, y \neq x_j} D\left(x_j,\, y\right)$$

Thus, a measures how far the data point is from all other points in its cluster. We set $a_j = 0$ if x_j is the only member of C_i The next calculation is between clusters,

$$b_j = \min_{\ell \neq i} \frac{1}{|\mathsf{C}_\ell|} \sum_{y \in \mathsf{C}_\ell} D(x_j, y)$$

b_j is the minimum of the average distances between x_j and every point in another cluster.

Combining a_j and b_j into s_j gives the score of the point x_j.

$$s_j = \frac{b_j - a_j}{\max(a_j, b_j)}$$

We can score a clustering solution by taking the average score over the entire data set of n data points.

$$score = \frac{1}{n} \sum_{j=1}^{n} s_j$$

A good clustering has a higher score, so the silhouette score is maximized to obtain the best solution.

Similar to the demonstration in [22], the labeled Fisher Iris data set can also be scored using different amounts of clusters. The average silhouette score in each cluster for $k = 2$ and $k = 3$ are shown in Figure 4. However, this is not the best silhouette score possible even though it is the true data classifications.

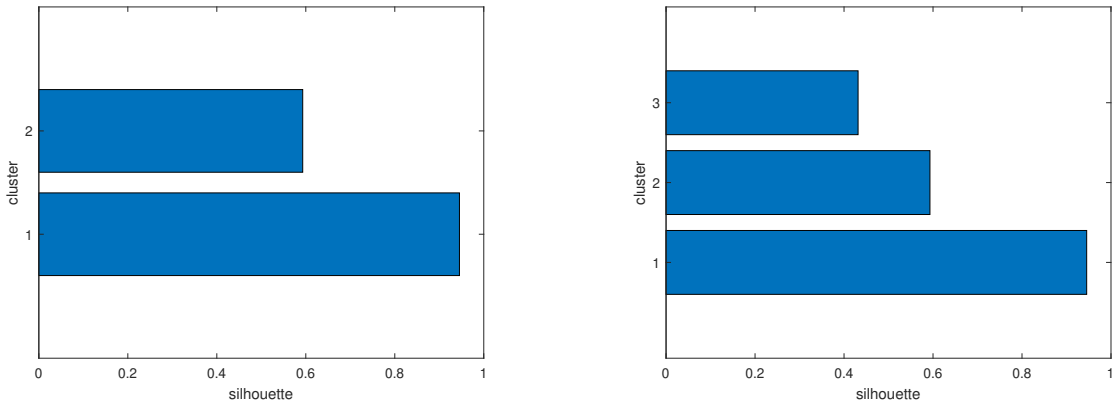

Figure 4: Average silhouettes of the iris data set using 2 and 3 clusters. The total score for two clusters is 0.7692 while the total score for three clusters is 0.6567. This indicates that using two clusters will provide a better representation of the data's structure.

IV. Data

The Iris Data Set [12] is a widely used data set for classification and clustering and is the primary data set used in this paper. It contains 150 4 dimensional iris flower entries. Each data point represents a flower through four measurements: sepal length, sepal width, petal length, and petal width, measured in centimeters. The parts of the flower are shown in Figure 5.

The data is dived evenly into three classes by the flower's species, *Iris Setosa*, *Iris Versicolor*, and *Iris Virginica*. Two clusters are linearly separable [12], one cluster of *Iris Setosa* and another cluster of both *Iris Versicolor* and *Iris Virginica*. This is most clear when viewing the data from the third and fourth dimensions as shown in Figure 5.

V. Elephant Herd Behavior

Elephant herding optimization, a variant of particle swarm optimization, was first introduced by Wang *et al.* in [25]. Elephants are the largest land mammal, largest among them is the African Savanna Elephant. Other species include the African Forest Elephant and the Asian Elephant The herd behaviour in this paper is based on the African Savanna Elephant.

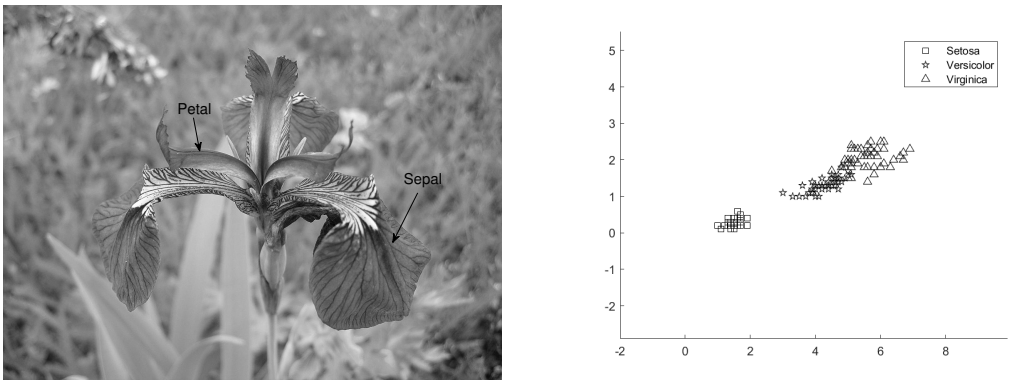

Figure 5: An example of an *Iris Setosa* from [21] and the Iris Data Set projected along the 3rd and 4th dimensions. Data points in the Iris Data Set record the petal and sepal dimensions.

An elephant *population* lives in a *home range* and can contain many different herds. Elephants feed for 18 hours a day, consuming hundreds of pounds of plants. They often come into competition with humans for land and resources [27]. Each individual elephant lives in a *herd*, sometimes called a clan, and is led by a *matriarch*. Herds exhibit coordinated behaviour [16] and contain female elephants and calves. While female elephants remain with the herd their entire lives [27], male elephants eventually leave the herd and live alone or in small, loosely associated groups.

These behaviours can be modeled to conform to the principles from [10] that particle swarm optimization is based on. The principles of quality and stability are captured by the movement of the herd based on the matriarch and the herd's coordinated behaviour. Since an elephant herd is not located at a single point in space, rather, it occupies a small area. The solutions are not committed down too narrow a path.

To model the elephant herds, the particle swarm is substituted for a population of elephants, which are considered the solution candidates. The variant of particle swarm optimization that uses the elephant behavior is called elephant herding optimization [25]. It takes advantage of the elephant movement being coordinated by the matriarch and by having the male elephants leave the herd. The matriarch corresponds to the local best solution, while p_{best} was used in the original PSO from Section II.

To minimize the clustering fitness function $F : \Omega \times \mathbb{R}^N \to \mathbb{R}$, we create the elephants in the search space Ω and herds such that

$$\text{elephant}_j \in \text{herd}_\ell \subseteq \Omega$$

Since this is an iterative algorithm, one iteration is used to represent a generation of elephants. Every generation, the worst solution or male elephant is removed from the herd and replaced by a new randomly generated elephant calf. Their replacement corresponds to the principle of adaptability as the reward for removing the solution with the worst fitness is higher than the cost of doing so.

The home range's population supporting resources is analogous to the search space containing the target extrema. Elephants will remain in a region constructed from $[x_{min}, x_{max}]$ [25]. In clustering tasks, the space used is a multi-dimensional region using coordinate-wise minima and maxima of the data space. Because the elephant represents multiple centroids, every centroid is restricted to fall within $[x_{min}, x_{max}]$. So for k clusters, the search space is $[x_{min}, x_{max}]^k$. Additional padding can be added to each coordinate to artificially expand the region, demonstrated in Figure 6.

Instead of performing position updates on the entire swarm of particles, the elephant population is first evenly divided into m herds and updated separately. Every herd, denoted by , contains n_{ci} elephants while the population size is n_{total}. The jth elephant of the cith herd is written as $\text{elephant}_{ci,j} \in \text{herd}_{ci}$ and $\text{elephant}_{ci,best}$, $\text{elephant}_{ci,worst}$ denote the matriarch and the worst elephant in the herd respectively.

The population $\{\text{herd}_1, \text{herd}_2, \ldots, \text{herd}_m\}$ can be initialized randomly. Let $r \in [0,1]^{n_{total} \times \dim(x)}$, be a vector with random entries. For each elephant in the population, the initialization is done by

$$\text{elephant} = x_{min} + r \circ (x_{max} - x_{min}) \tag{6}$$

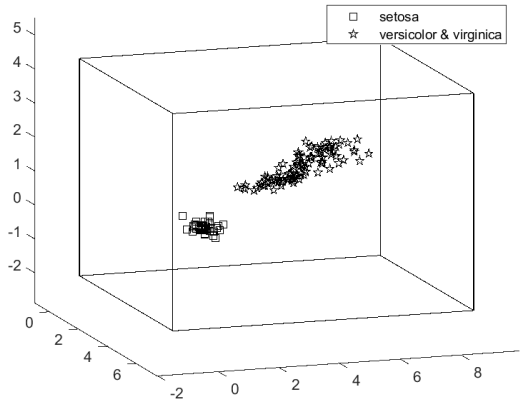

Figure 6: The search region containing the data points defined coordinate-wise by $[x_{min} - 2, x_{max} + 2]$. The 2nd, 3rd, and 4th dimension of the data points are shown

We next update the elephant positions. If the elephant is not a matriarch, then it's movement will be influenced by the matriarch of its herd. At iteration t, a non-matriarch elephant is updated as

$$\text{🐘}_{ci,j}^{(t+1)} = \text{🐘}_{ci,j}^{(t)} + \alpha r \left(\text{🐘}_{ci,best}^{(t)} - \text{🐘}_{ci,j}^{(t)} \right) \tag{7}$$

The parameter α can be tuned depending on how much the matriarch should influence the elephant's position. The uniform random variable $0 \leq r \leq 1$ is used to slightly perturb the solution.

The same step cannot be used to update the matriarch. Instead, the center of the elephant herd as the environmental qualitative factor that influences her behaviour. The amount of influence can be tuned through the parameter β. We calculate

$$\text{🐘}_{ci,best}^{(t+1)} = \text{🐘}_{ci,best}^{(t)} + \beta \left(\left(\frac{1}{n_{ci}} \sum_{j=1}^{n_{ci}} \text{🐘}_{ci,j}^{(t)} \right) - \text{🐘}_{ci,best}^{(t)} \right) \tag{8}$$

To prevent the elephants from leaving the home range, they are constrained to the search space by setting each coordinate to

$$\text{🐘} = \min \left(\max \left(\text{🐘}, x_{max} \right), x_{min} \right) \tag{9}$$

In the original [25, Eq. (3)] and subsequent papers [18], [9], the matriarch is updated as

$$\text{🐘}_{ci,best}^{(t+1)} = \beta \frac{1}{n_{ci}} \sum_{j=1}^{n_{ci}} \text{🐘}_{ci,j}^{(t)}$$

However, using this update with $\beta \ll 1$ quickly moves the elephant population towards 0 and we cannot assume that 0 is the actual solution. To continue to have the herd's center influence the matriarch, without necessarily moving her closer to 0, the update in Eq. (8) is used instead.

The separating operator is defined next. Although male elephants leave the close-knit herd, they remain in the home range. We can't lower the number of elephants in each herd because doing so would decrease the amount of solution candidates. Thus the update to the worst elephant is

$$\text{🐘}_{ci,worst}^{(t+1)} = x_{min} + r_n \left(x_{max} - x_{min} \right) \tag{10}$$

where $r_n \in [0, 1]$ is a normal random variable [25].

VI. Gradient based optimizer

A further modification to particle swarm optimization is the introduction of a two operators, a gradient search rule (GSR) and a locally escaping operator (LEO) [2]. Using two more operators improves the diversity and qualitative response principles of particle swarming. At the same time, it increases both the complexity of spatial computation and the amount of factors influencing particle movement.

The locally escaping operator uses the direction of the gradient at the position of a solution produce a new solution which is, locally, more optimal. In this section, *solution* is used instead of *elephant* as this approach applies more generally EHO. Accordingly, x denotes a solution in the swarm, x_n is the position of x at the nth iteration, and f is the function we are optimizing.

1) Newton's Method: Newton's method is root finding algorithm with the recursive formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{11}$$

The local linear model [26, Eq. 3.2]

$$M_n(x) = f(x_n) + f'(x_n)(x - x_n) \tag{12}$$

If the derivative of f is not available in analytic form, the numerical gradient can be used instead. In the cluster analysis problems in this paper, the fitness function uses the numerical gradient approach [2]. The centered difference approximation of the first derivative, $f'(x)$, can be derived from the Taylor series of $f(x + \Delta x)$ and $f(x - \Delta x)$.

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}$$

After re-writing Eq. (11) with the numerical gradient, the iterative step is

$$x_{n+1} = x_n - \frac{f(x_n)2\Delta x}{f(x + \Delta x) - f(x - \Delta x)} \tag{13}$$

In a minimization problem, the iteration result x_n should be approaching the minimum of f as n increases.

A. Modifications of Newton's method

We start with three ways to approximate a definite integral: the rectangular rule, Eq. (14), the trapezoidal rule, Eq. (15), and the midpoint rule, Eq. (16), and work towards two additional versions of Newton's methods. Consider the following formulas, which are derivable from the Taylor expansion of f'

$$\int_{x_n}^{x} f'(\lambda)d\lambda = f'(x_n)(x - x_n) + \frac{1}{3}h^3 f'''(\xi) \tag{14}$$

$$\int_{x_n}^{x} f'(\lambda)d\lambda = \frac{1}{2}(x - x_n)(f'(x_n) + f'(x)) - \frac{1}{12}h^3 f'''(\xi) \tag{15}$$

$$\int_{x_n}^{x} f'(\lambda)d\lambda = (x - x_n)f'\left(\frac{x_n + x}{2}\right) + \frac{1}{3}f'''(\xi) \tag{16}$$

The error term is also included in each formula, ξ being a number in the interval between x_n and x.

Substituting the rectangular rule into the right hand side of Eq. (12) and ignoring the approximating error term, the local linear model can be immediately written as

$$M_n(x) = f(x_n) + \int_{x_n}^{x} f'(\lambda)d\lambda$$

Next, the rectangular rule is swapped with the trapezoidal rule to approximate the integral term. Setting $M_n(x) = 0$, by solving for $x_{n+1} = x$, the following reformulation [26] is obtained

$$\begin{aligned} 0 &= f(x_n) + \frac{1}{2}(x_{n+1} - x_n)\left[f'(x_n) + f'(x_{n+1})\right] \\ x_{n+1} &= x_n - \frac{2f(x_n)}{f'(x_n) + f'(x^*_{n+1})} \end{aligned} \tag{17}$$

We can also use the midpoint rule [28] for the integral term in Eq. (11).

$$x_{n+1} = x_n - \frac{f(x_n)}{f'\left(\frac{x_n + x^*_{n+1}}{2}\right)} \tag{18}$$

To avoid the implicit use of x_{n+1} on the right hand side, x^*_{n+1} is calculated with any different version of Newton's method.

B. Gradient search rule

Newton's method, Eq. (13), can next be adapted for the clustering task. Clustering uses a fitness function $f(x)$ which is optimized in order to find the best cluster centroids, x. Suppose we wish to minimize the fitness function. We assume that $f(x_n + \Delta x) > f(x_n - \Delta x)$ [2, p. 134]. That is, $x_n - \Delta x$ has a better (lower) fitness than $x_n + \Delta x$. Since clustering often uses an expensive fitness function, we do not want to evaluate it any more than necessary. If $f(x_n \pm \Delta x)$ can be removed from the formula, we no longer need to evaluate the fitness function at $x_n \pm \Delta x$, eliminating a time consuming computation [2].

Luckily, there is already a list of solutions with fitness values better or worse than x_n that has already been calculated. In Particle Swarm Optimization algorithms, the fitness of all possible solutions must be calculated every iteration to update g_{best}. Instead of needing to find $f(x_n \pm \Delta x)$, we pick the x_{best} and x_{worst} solutions in the neighborhood of x_n, which have a better and worse fitness value respectively. For simplicity, the entire space is used as the neighborhood of x_n, allowing us to pick the overall best and worst particles of the current iteration. We substitute x_{best} for $f(x_n - \Delta x)$ and x_{worst} for $f(x_n + \Delta x)$ into Eq. (VI-1) and obtain the first gradient search rule (GSR) [2, Eq. (14)] for x_n.

$$GSR_C = randn \times \rho \times \frac{2x_n \Delta x}{x_{worst} - x_{best} + \varepsilon} \tag{19}$$

where ρ is a parameter, $randn$ is a normal random number, and $\varepsilon \in [0, 0.1]$ is small.

Even though the fitness does not have to be computed, we still require a Δx. Following [2], [9], several parameters are defined in order to form the Δx. Let $r \in [0, 1]$ be a uniform random number, T be the maximum number of iterations, t be the current iteration number, and $\varepsilon > 0$ be a small value to avoid dividing by zero.

$$\rho_i = 2r\alpha - \alpha \tag{20}$$

$$\alpha = \left| \beta \sin\left[\frac{3\pi}{2} + \sin\left(\beta \frac{3\pi}{2}\right)\right]\right| \tag{21}$$

$$\beta = \beta_{min} + (\beta_{max} - \beta_{min})\left[1 - \left(\frac{t}{T}\right)^3\right]^2 \tag{22}$$

Equations (21) and (22) act as a learning rate scheduler, varying based on the current iteration number. ρ will be used multiple times but each different ρ_i calculated with a unique uniformly drawn random value r. For each point x_n, the Δx is calculated as the distance from x_n, to the average position of four randomly chosen points, $x_{r1}, x_{r2}, x_{r3}, x_{r4}$. R is a vector in the data space with random entries between 0 and 1.

$$\delta = 2r\left(\left|\frac{x_{r1} + x_{r2} + x_{r3} + x_{r4}}{4} - x_n\right|\right)$$

$$\Delta x = R\left|\frac{x_{best} - x_{r,1} + \delta}{2}\right| \tag{23}$$

The direction of motion parameter [9, Eq. (18)] is defined as the direction when moving towards the best solution,

$$DM = \rho_2 r(x_{best} - x_n) \tag{24}$$

Next, we calculate the midpoint Newton's method, but substituting $z_{n+1} = x^*_{n+1} = GSR_C$ in Eq. (18), giving us the second GSR formula in Eq. (26) [9], [2].

$$y = \frac{z_{n+1} + x_n}{2}$$

$$GSR_M = \rho_1 r \frac{2x_n \Delta x}{(r_1 y + r_2 \Delta x) - (r_3 y - r_4 \Delta x) + \varepsilon} \tag{25}$$

GSR_M is used to calculate the first new candidate solution $X1_n$.

$$X1_n = x_n - GSR_M + DM$$
$$X1_n = x_n - \rho_1 r \frac{2x_n \Delta x}{(r_1 y + r_2 \Delta x) - (r_3 y - r_4 \Delta x) + \varepsilon} + \rho_2 r(x_{best} - x_n) \tag{26}$$

The variables $r_i \in [0, 1]$ have a uniform distribution. To approximate f', the centered difference formula is used. The same updating step can be applied to x_{best}, creating another candidate $X2_n$ as

$$X2_n = x_{best} - \rho_1 r \frac{2x_n \Delta x}{(r_1 y + r_2 \Delta x) - (r_3 y - r_4 \Delta x) + \varepsilon} + \rho_2 r(x_{r1} - x_{r2}) \tag{27}$$

It can be noted that without the random variables the denominator in both Eq. (26) and Eq. (27) are reduced to ε. In Eq. (27), the DM term has been changed to use the difference between two randomly chosen x solutions instead of x_{best}. x_n is updated by a combination of $X1_n$, Eq. (26) and $X2_n$, Eq. (27) and uniform random values $r_1, r_2 \in [0, 1]$. We take a weighted average of the two candidates $X2$ and $X1$ to produce an $X3$ and the updated position X_m.

$$X3 = x_n - \rho_1 (X2 - X1) \tag{28}$$
$$X_m = r_1 \left[r_2 X1_n + (1 - r_2) X2_n \right] + (1 - r_1) X3_n \tag{29}$$

C. Locally escaping operator

Using the gradient search rule, we can construct a better solution that is able to escape a local minimum.

Following [2], we start with two parameters μ_1, μ_2 and two corresponding binary variables L_1, L_2.

$$L_i = \begin{cases} 0 & \mu_i < 0.5 \\ 1 & \mu_i \geq 0.5 \end{cases}$$

Three additional variables that depend on L are also defined.

$$u_1 = 2L_1 r + (1 - L_1) \tag{30}$$
$$u_2 = L_1 r + (1 - L_1) \tag{31}$$
$$u_3 = L_1 r + (1 - L_1) \tag{32}$$

Taking a random particle in the swarm, x_p, and a random solution generated by

$$x_r = x_{min} + r \left(x_{max} - x_{min} \right)$$

the last solution x_k is computed.

$$x_k = L_2 x_p + (1 - L_2) x_r \tag{33}$$

The LEO from [2] can now be assembled from the candidate solutions in Equations (29), (26), (27) and (33). This calculation is similar to the velocity term in the PSO algorithm from [10] in Eq. (2). Let $\varphi_1 \sim \mathcal{U}([-1, 1])$ and $\varphi_2 \sim \mathcal{N}(0, 1)$.

$$X_{LEO} = X_m + \varphi_1 \left(u_1 g_{best} - u_2 x_k \right) + \frac{1}{2} \varphi_2 \rho_1 \left(u_3 \left(X2 - X1 \right) + u_2 \left(x_{r1} - x_{r2} \right) \right) \tag{34}$$

To further the diversity of the swarm, a conditional parameter pr is provided to change the probability that X_{LEO} will be instead calculated by

$$X_{LEO} = g_{best} + \varphi_1 \left(u_1 g_{best} - u_2 x_k \right) + \frac{1}{2} \varphi_2 \rho_1 \left(u_3 \left(X2 - X1 \right) + u_2 \left(x_{r1} - x_{r2} \right) \right) \tag{35}$$

X_{LEO} can be used as a replacement for the particle $x_n^{(t)}$.

Even though using a locally escaping operator will help the swarm converge to the optimal solution [2], it requires that we calculate the fitness function more often. Each X_{LEO} needs to have its fitness compared against the $x_n^{(t)}$ that it is generated from. The X_{LEO} is only considered for replacement if it has a better fitness.

In terms of the five principles from Section II, using a locally escaping operator has broken at least two. The complexity of a particle update now requires the fitness evaluation and a much longer chain of calculations for X_{LEO} than the simpler velocity term in Eq. (2). X_{LEO} also depends on more environmental factors than the normal particles. Several additional random members of the swarm are required to affect the new position, rather than only the three types of best particle in PSO.

D. Elephant herding LEO

Returning to the elephant herding algorithm, for each elephant, we construct a locally escaping operator. However, to avoid the computationally expensive fitness calculation for every elephant, a regulatory parameter psr is used

$$🐘_{ci,j,LEO} = \begin{cases} 🐘_{ci,j,LEO} & rand < psr \\ 🐘_{ci,j} & rand \geq psr \end{cases} \tag{36}$$

$$🐘_{ci,j,LEO} = \begin{cases} 🐘_{M} + \varphi_1 \left(u_1 🐘_{gbest} - u_2 🐘_{k}\right) + \frac{1}{2}\varphi_2\rho_1 \left(u_3 \left(🐘_{2} - 🐘_{1}\right) + u_2 \left(🐘_{r1} - 🐘_{r2}\right)\right) & rand < pr \\ 🐘_{gbest} + \varphi_1 \left(u_1 🐘_{gbest} - u_2 🐘_{k}\right) + \frac{1}{2}\varphi_2\rho_1 \left(u_3 \left(🐘_{2} - 🐘_{1}\right) + u_2 \left(🐘_{r1} - 🐘_{r2}\right)\right) & rand \geq pr \end{cases} \tag{37}$$

One more modification is made for the gradient based optimizer to be used with elephant herding. Instead of using Eq. (7) to update the elephants, if $r \sim \mathcal{U}[0,1]$, $r < psr$, then the update uses a similar process to the velocity equation from Eq. (2) [9].

$$🐘_{ci,j} = 🐘_{ci,j} + r_1 \left(\frac{🐘_{ci,j,pbest} + 🐘_{ci,j,pbest}}{2} - 🐘_{ci,j}\right) + r_2 \left(\frac{🐘_{ci,j,pbest} - 🐘_{gbest}}{2} - 🐘_{ci,j}\right) \tag{38}$$

VII. Results

Both the elephant herding optimization [25] and the gradient based elephant herding optimization [9] algorithms were used to cluster the iris data set. The cluster results are compared against $k$means and a particle swarm clustering [3]. Fifty particles were used in each algorithm. In the two elephant based algorithms, there are five herds of ten elephants. All algorithms were run for 30 iterations.

The results of gradient based elephant herding optimization are shown in Figure 7. The SSE fitness of g_{best} in the GBEHO algorithm is plotted over time and the lowest fitness is 6.9822 which is also the fitness that $k$means has. In comparison, the unmodified elephant herding optimization from [25] have a fitness of 7.1773 and are shown in Figure 8. While PSO is only slightly worse in fitness, it produces many outliers. A box plot of GBEHO, EHO, PSO, and $k$means is also shown in the top left of Figure 8.

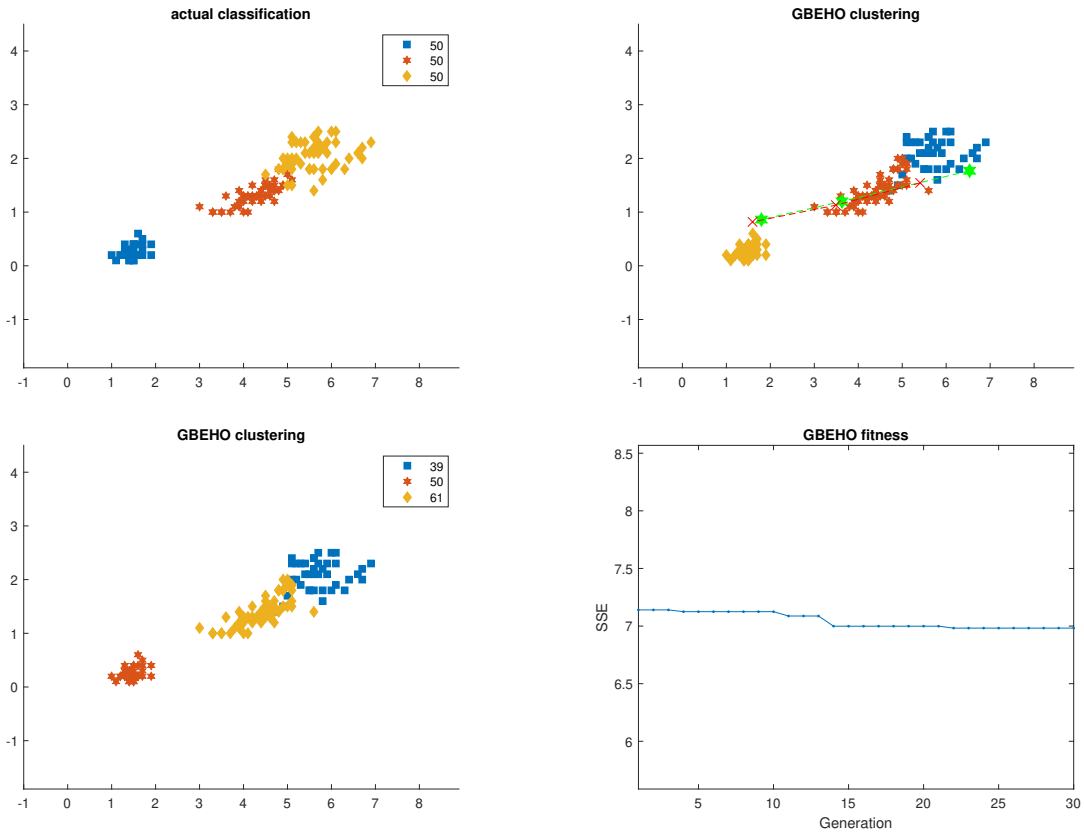

Figure 7: In the top left the true clustering from the labeled data set is shown. The top right is the position of the g_{best} (green) and g_{worst} (red) elephants. The clustering found by the GBEHO algorithm is in the bottom left and the bottom right is the graph of fitness over time. GBEHO achieves a fitness of 6.9822 after 30 generations.

Two other data sets, Wine and seeds [8], were also clustered. The Wine Data Set has 178 wines with 12 characteristics of each wine. The seeds Data Set is 12 dimensional and has 210 entries. Both of them are labeled, like the Fisher Iris Data Set, and contain three different categories. The box plot of the different algorithms discussed is shown in Figure 10 and compared against $k$means. $k$means gives a solution worse than GBEHO by about 10^{-14}, GBEHO has 49.8491813679964 and $k$means is 49.8491813679965. This could be due to floating point precision instead of the solutions being different.

The silhouette fitness function was also evaluated but did not produce results for $k = 3$ clusters. GBEHO found the two linearly separable clusters in the Fisher Iris Data Set when $k = 3$ was passed as a parameter. The clusters and fitness graph for the silhouette fitness function is in Figure 12.

VIII. Conclusion

Gradient Based Elephant Herding Optimization is at least as good as $k$means when clustering the Fisher Iris, Wine, and seeds Data Sets [8]. On smaller data sets such as the Wine and seeds data sets, both elephant herding algorithms take more iterations and more computation time to create a clustering than $k$means does. While Gradient Based Elephant Herding Optimization offers an additional improvement over regular Elephant Herding Optimization, it is more expensive to compute. Elephant herding optimization, on the other hand, maintains the simplicity of particle movement calculations and performs better than PSO.

The computation time is dominated by the fitness function evaluation step. Using either a different fitness function or one that can be computed in parallel could provide a significant increase in speed but this was not tested. Another performance consideration in the implementation is the use of a sorting function to find the g_{best} and g_{worst} particles and the two matriarchs of the herd. Particle swarm optimization algorithms are flexible so it would be easy to substitute another fitness function for the SSE or Silhouette ones used in this paper.

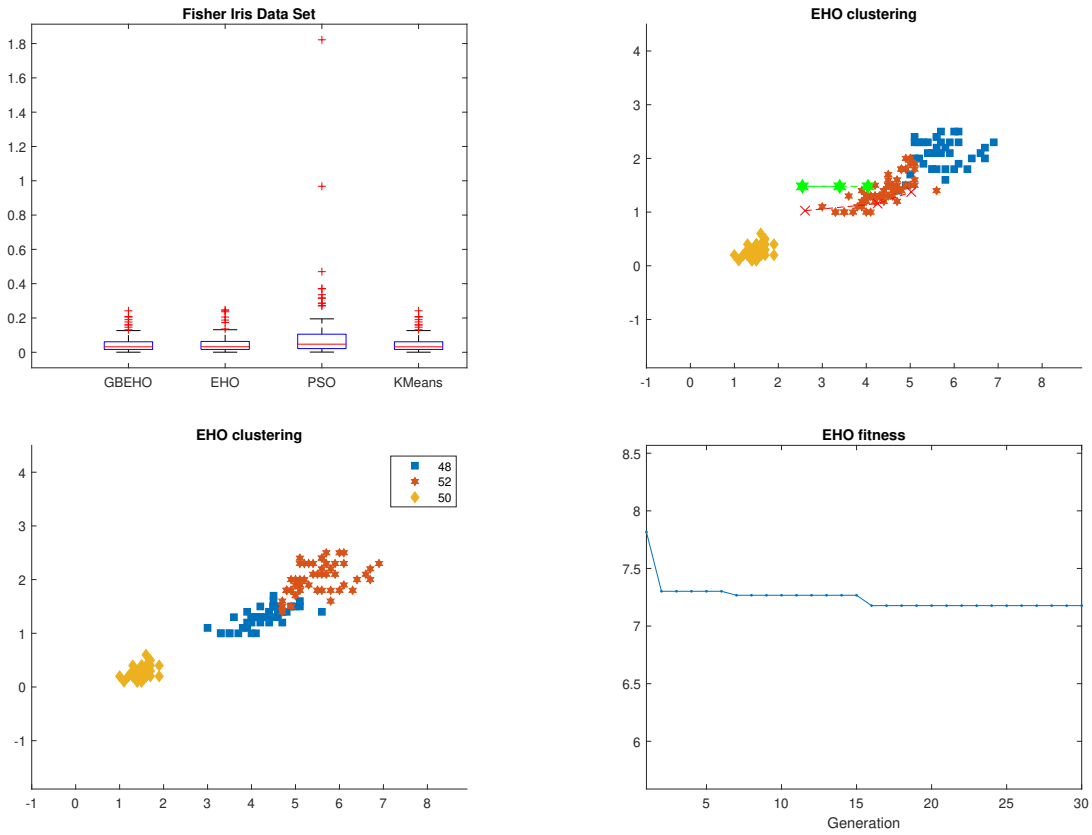

Figure 8: The top left is a box plot comparing the three algorithms based on particle swarm optimization and $k$means. The GBEHO and $k$means are the best and achieve the same fitness of 6.9822 while EHO is slightly worse at 7.1773. PSO is the worst at 14.2745. EHO was run with 5 herds of 10 elephants each and 30 generations. The clustering found is in the bottom left while the g_{best} and g_{worst} solutions are shown in the top right as green and red lines respectively.

	$k$means	PSO	EHO	GBEHO
SSE	6.9822	14.2745	7.1773	6.9822
$\lvert\mathcal{C}_1\rvert$	61	80	52	61
$\lvert\mathcal{C}_2\rvert$	50	49	50	50
$\lvert\mathcal{C}_3\rvert$	39	21	48	39

Figure 9: The lowest fitness of solution in every algorithm and the number of data points assigned to each cluster by the best solution. The clusters have been reordered to fit in the table since the algorithms are not classifying the data.

The use of a locally escaping operator is not only expensive but it breaks the particle swarm principles. One principle of particle swarm optimization algorithms is the simplicity in particle position and velocity updates. The algorithm is more confusing to use due to the lack of biological analogy of the LEO elephant. Although the foundations are built upon multiple applications of different forms of Newton's method, the LEO includes many random variables and parameters which must be tuned. A benefit of particle swarm optimization algorithms is their robustness against parameter choice so to promote a behavior change, there are many parameters to consider. It is difficult to exhaust the combinations of parameters and find which ones produce a satisfactory result.

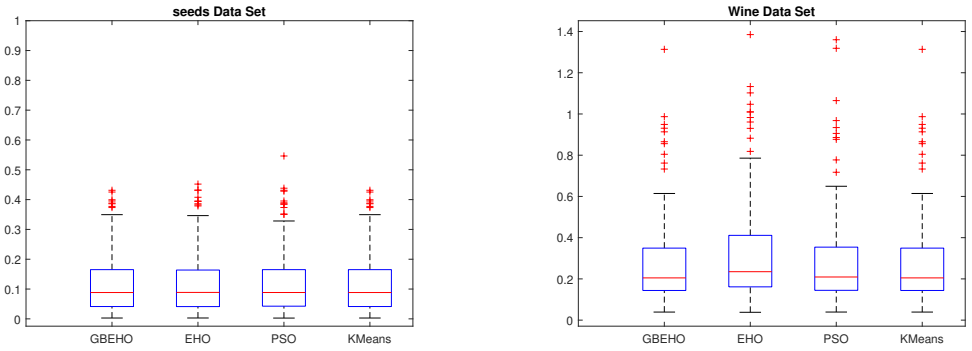

Figure 10: Resulting box plots of the seeds and Wine Data Sets. The clustering found by GBEHO is the best but is only marginally better than $k$means.

	$k$means	PSO	EHO	GBEHO
seeds	24.4759	24.7316	24.5911	24.4759
Wine	49.8492	51.1744	57.3073	49.8492

Figure 11: The best solution's fitness in the four algorithms using the seeds and Wine Data Sets. EHO performs the worst in the Wine Data Set. The $k$means and GBEHO achieve equal results for both data sets.

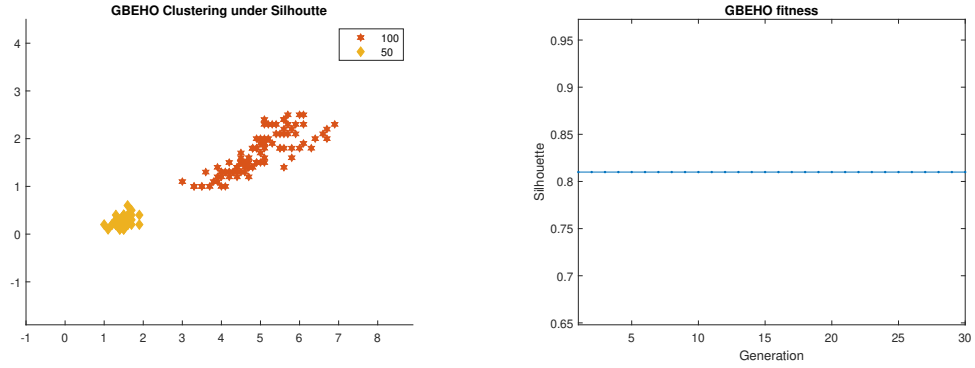

Figure 12: The clustering found with GBEHO using silhouette fitness, which only found two clusters. The silhouette score is 0.81, with 1 being the best.

References

[1] I. Ahmadianfar, O. Bozorg-Haddad, and X. Chu, *Gradient Based Optimizer*, version 1.0.0, 2021. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/96997-gradient-based-optimizer-gbo

[2] ——, "Gradient-based optimizer: A new metaheuristic optimization algorithm," *Information Sciences*, vol. 540, pp. 131–159, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520306241

[3] A. L. Ballardini, "A tutorial on particle swarm optimization clustering," 2018. [Online]. Available: https://arxiv.org/abs/1809.01942

[4] Z. Beheshti and S. M. H. Shamsuddin, "A review of population-based meta-heuristic algorithms," *International Journal of Advances in Soft Computing and its Applications*, vol. 5, no. 1, pp. 1–35, 2013.

[5] H.-G. Beyer and S. Finck, "Happycat – a simple function class where well-known direct search algorithms do fail," in *Parallel Problem Solving from Nature - PPSN XII*, C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 367–376.

[6] L. D. Castro, "Fundamentals of natural computing: an overview," *Physics of Life Reviews*, vol. 4, no. 1, pp. 1–36, Mar. 2007. [Online]. Available: https://doi.org/10.1016/j.plrev.2006.10.002

[7] A. Darwish, "Bio-inspired computing: Algorithms review, deep analysis, and the scope of applications," *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 231–246, Dec. 2018. [Online]. Available: https://doi.org/10.1016/j.fcij.2018.06.001

[8] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[9] Y. Duan, C. Liu, S. Li, X. Guo, and C. Yang, "Gradient-based elephant herding optimization for cluster analysis," *Applied Intelligence*, vol. 52, no. 10, pp. 11 606–11 637, Jan. 2022. [Online]. Available: https://doi.org/10.1007/s10489-021-03020-y

[10] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.

[11] R. Fisher, "The use of multiple measurements in taxonomic problems," pp. 179–188, Sept. 1936.

[12] ——, "Iris," UCI Machine Learning Repository, 1988.

[13] H. Han, W. Lu, L. Zhang, and J. Qiao, "Adaptive gradient multiobjective particle swarm optimization," *IEEE Transactions on Cybernetics*, vol. 48, no. 11, pp. 3067–3079, Nov. 2018. [Online]. Available: https://doi.org/10.1109/tcyb.2017.2756874

[14] D. Hoskins, "Two gray elephant on brown sand," 2020. [Online]. Available: https://images.pexels.com/photos/3684839/pexels-photo-3684839.jpeg

[15] K. Hussain, M. N. M. Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: a comprehensive survey," *Artificial Intelligence Review*, vol. 52, no. 4, pp. 2191–2233, Jan. 2018. [Online]. Available: https://doi.org/10.1007/s10462-017-9605-z

[16] M. P. Kahl and C. Santiapillai, "A glossary of elephant terms," 2004. [Online]. Available: https://www.asesg.org/PDFfiles/Gajah/23-01-Glossary.pdf

[17] L. E. King, I. Douglas-Hamilton, and F. Vollrath, "African elephants run from the sound of disturbed bees," *Current Biology*, vol. 17, no. 19, pp. R832–R833, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960982207017186

[18] J. Li, H. Lei, A. H. Alavi, and G.-G. Wang, "Elephant herding optimization: Variants, hybrids, and applications," *Mathematics*, vol. 8, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2227-7390/8/9/1415

[19] V. M and S. K. R, "An emperor penguin optimization and particle swarm optimization control algorithm for PV with ZSI in grid connected system," Apr. 2021. [Online]. Available: https://doi.org/10.21203/rs.3.rs-440660/v1

[20] M. M. Millonas, "Swarms, phase transitions, and collective intelligence," 1993. [Online]. Available: https://arxiv.org/abs/adap-org/9306002

[21] Денис Анисимов, "Irissetosa1," 2017. [Online]. Available: https://commons.wikimedia.org/wiki/File:Irissetosa1.jpg

[22] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042787901257

[23] D. van der Merwe and A. Engelbrecht, "Data clustering using particle swarm optimization," in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, vol. 1, 2003, pp. 215–220 Vol.1.

[24] ——, "Data clustering using particle swarm optimization," in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, vol. 1, 2003, pp. 215–220 Vol.1.

[25] G.-G. Wang, S. Deb, and L. d. S. Coelho, "Elephant herding optimization," in *2015 3rd International Symposium on Computational and Business Intelligence (ISCBI)*, 2015, pp. 1–5.

[26] S. Weerakoon and T. Fernando, "A variant of newton's method with accelerated third-order convergence," *Applied Mathematics Letters*, vol. 13, no. 8, pp. 87–93, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893965900001002

[27] Elephant. World Wildlife Fund. [Online]. Available: https://www.worldwildlife.org/species/elephant

[28] A. Özban, "Some new variants of newton's method," *Applied Mathematics Letters*, vol. 17, no. 6, pp. 677–682, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893965904901048

Appendix

A. Particle swarm optimization

```
%% see https://arxiv.org/pdf/1809.01942.pdf for parameter selection reasons
function [swarm_pos,gbest] = pso(x)
rng(0)

k = 3;
n_p = 50; %
iterations = 30;

[n, d] = size(x);

w  = 0.72;
c1 = 1.49;
c2 = 1.49;

swarm_vel = rand(k, d, n_p) * 0.1;
swarm_pos =  rand(k, d, n_p);
swarm_best = zeros(k, d);

c = zeros(n, n_p);

swarm_pos = swarm_pos .* repmat(max(x)-min(x),k,1,n_p) + repmat(max(x)-min(x),
   k,1,n_p);
swarm_fitness(1:n_p) = 10000;
gbest_fitness_history = zeros([iterations, 1]);


for i = 1:iterations
    Dd = zeros(n, k, n_p); % distance matrix
    for p = 1:n_p % particle p
        for j = 1:k % centroid j
            D_j = zeros(n,  1);
            for ell = 1:n % datapoint ell
                D_j(ell) = norm(swarm_pos(j, :, p) - x(ell,:));
            end
            Dd(:, j, p) = D_j;
        end
    end

    for p = 1:n_p
        [~, idx] = min(Dd(:, :, p), [], 2);
        c(:, p) = idx;
    end

    delete(pc); delete(txt);
    pc = []; txt = [];

    %% fitness
    avg_fitness = zeros(n_p, 1);
    for p = 1:n_p
        [avg_fitness(p), ~] = fitness(swarm_pos(:, :, p), x);
```

```
        avg_fitness(p) = -avg_fitness(p);

        if avg_fitness(p) < swarm_fitness(p)
            swarm_fitness(p) = avg_fitness(p);
            swarm_best(:, :, p) = swarm_pos(:, :, p);
        end
    end

    [~, idx] = min(swarm_fitness);
    swarm_pose = swarm_pos(:, :, idx);

    gbest = swarm_pos(:, :, idx);

    %% update swarm
    r1 = rand; r2 = rand;
    for p = 1:n_p
        inertia = w * swarm_vel(:, :, p);
        cognitive = c1 * r1 * (swarm_best(:, :, p) - swarm_pos(:, :, p));
        social = c2 * r2 * (swarm_pose - swarm_pos(:, :, p));
        vel = inertia + cognitive + social;

        swarm_pos(:, :, p) = swarm_pos(:, :, p) + vel;
        swarm_vel(:, :, p) = vel;
    end
end

end
```

```
function [swarm, swarm_history, swarm_pos_t] = pso_min(F, x_min, x_max, T_max,
    n_p)
% F is function being minimized
% T_max is maximum iterations
% n_p is number of particles

d = numel(x_min); % dimension
velocity = rand(d, n_p) * 0.1;
swarm =  x_min + (x_max - x_min) .* rand(d, n_p); % particle position


informants = 3; % number of local best to use
swarm_best = inf([d, n_p, informants]); % best

swarm_fitness = F(swarm);

swarm_history = zeros([size(swarm), T_max]);

phi = 2.07;
c1 = 1 / (phi - 1 + sqrt(phi^2 - 2 * phi));
c_max = phi * c1;

for t = 1:T_max
    c2 = rand*c_max;
    c3 = rand([1, 3]) * c_max;

    swarm_history(:, :, t) = swarm;

    swarm_fitness_new = F(swarm);
    for j=1:n_p
        if swarm_fitness_new(j) <= swarm_fitness(j)
            swarm_fitness(j) = swarm_fitness_new(j);

            informant_idx = F(swarm(:, j)) <= F(squeeze(swarm_best(:, j, :)));
            [~, j_max] = max(informant_idx); % first index
            swarm_best(:, j, j_max) = swarm(:, j); % best position in a
               particles history (pbest). local best unused
        end
    end

    [~, i_min] = min(swarm_fitness); % global best
    swarm_pose = swarm(:, i_min);


    for j=1:n_p
        vel = c1 * velocity(:, j) + c2 * (swarm_pose - swarm(:, j));
        for k = 1:informants
            if swarm_best(:, j, k) < inf
                vel = vel + c3(k) * (squeeze(swarm_best(:, j, k)) - swarm(:, j
                   ));
            end
        end
```

```
        swarm(:, j) = swarm(:, j) + vel ;
        velocity(:, j) = vel;
    end
end
end
```

B. Elephant herding optmization

```
function [c, pos, population, i_ci, gbest, gbest_fitness,
    gbest_fitness_history] = elephant_1d(x, I, plotdata, plotpos)
rng(0)
% x = normalize(x, 'range', [-1 1]);
if nargin < 2
    plotdata = false;
end
if nargin < 3
    plotpos = false;
end

[n, d] = size(x);
x_max = max(x)+0.1;
x_min = min(x)-0.1;
beta = 1.5;
alpha = 0.5;
max_iter = 30;
k = 3;
m = 5;
n_ci = 10;
n_e = m*n_ci;

population = repmat(x_min, [k, 1]) + rand(k, d, n_ci * m) .* (repmat(x_max, [k
    , 1]) - repmat(x_min, [k, 1]));
i_ci = ones(1, m*n_ci);
% herd index
for i = 2:m
    i_ci((i-1)*n_ci+1:i*n_ci) = i;
end
x_min = repmat(x_min, [k, 1]);
x_max = repmat(x_max, [k, 1]);



colorv2 = rand([1, 3, k]);


% initial fitness calculation
[population, i_ci, population_fitness, i_c] = fitness(population, i_ci, x);
gbest = population(:, :, 1);
gbest_fitness = population_fitness(1);
cbest = i_c(:, 1); % indices of best solution
gbest_fitness_history = zeros(1, max_iter);


for t=1:max_iter
    gbest_fitness_history(t) = gbest_fitness;

%     population_center = mean(population, 3);

    for ci = 1:m
```

```
        herd = population(:, :, i_ci==ci);
        population_center = mean(herd, 3);
        herd(:, :, 1) = herd(:, :, 1) + beta * (population_center - herd(:, :,
            1));  % goes towards  center
        herd(:, :, end) = x_min + rand(size(x_min)) .* (x_max - x_min);
        for j = 2:n_ci
            herd(:, :, j) = herd(:, :, j) + alpha * rand * (herd(:, :, 1) -
               herd(:, :, j));
        end

        for j = 1:n_ci
            e = herd(:, :, j);
            e(e<x_min) = x_min(e < x_min);
            e(e>x_max) = x_max(e > x_max);
            herd(:, :, j) = e;
        end

        population(:, :, i_ci==ci) = herd;
    end

    [population, i_ci, population_fitness, i_c] = fitness(population, i_ci, x)
       ;

    if max(population_fitness) >= gbest_fitness
        gbest = population(:, :, 1);
        gbest_fitness = population_fitness(1);
        cbest = i_c(:, 1);
    end

end

c = cbest;
pos = gbest;
end

%% sorted fitness function and centroids
% i_c(j) is cluster indices using centroids in elephant(j)
function [population, i_ci, scores, i_c] = fitness(population, i_ci, x)
[n, d] = size(x);
[k, d, n_e] = size(population);
D = zeros(n, n_e); i_c = zeros(n, n_e); scores = zeros(1,n_e);
for j = 1:n_e
    [D(:, j), i_c(:, j)] = pdist2(population(:, :, j), x, 'squaredeuclidean',
       'Smallest', 1);

    D2 = zeros([length(x), 1]);
    cl_scores = zeros(1, k);
    for ell = 1:k
        x_in_c = x(i_c(:, j)==ell, :);
        cl_center = mean(x_in_c);
        cl_scores(ell) = sum((x_in_c - cl_center).^2, 'all');
```

```
        D2(i_c(:, j)==ell) = sum((x_in_c-cl_center).^2, 2);
    end
    scores(j) = -sum(cl_scores);
end
scores(isnan(scores)) = -1;
[~, i_sort] = sort(scores, 'descend');
scores = scores(i_sort);
population = population(:, :, i_sort);
i_ci = i_ci(i_sort);
i_c = i_c(:, i_sort);
end
```

C. Gradient based elephant herding optimization

```
% algorithm from the "elephant herding optimization"
% and "Combined Elephant Herding Optimization Algorithm
% with K-means for Data Clustering"
% cbest is clustering from gbest (best solution)
function [swarm_pos, gbest, cbest] = gbeho(x)
rng(0)

k = 3; % number of clusters
m = 5; % number of herds
n_ci = 10; % number of elephants per herd
n_e = m*n_ci; % population size
i_m = zeros(1, m); % best elephant index for each herd
i_ci = ones(n_e, 1); % index assigns elephants to herds
for j = 2:m
    i_ci(n_ci*(j-1)+1:n_ci*j) = j;
end

psr = .2; % parameter
beta_min = 0.2; beta_max = 1.2;
T = 30; % max iterations
tol = 0.09; % end if fitness is high enough (silhouette only?)


[n, d] = size(x);

% bound each centroid in a solution
x_min = repmat(min(x), [k, 1])-1;
x_max = repmat(max(x), [k, 1])+1;

swarm_pos =  rand([k, d, n_e]);

swarm_pos = swarm_pos .* repmat((max(x)-min(x)),k,1,n_e) + repmat(min(x),k,1,
    n_e);
% [~, swarm_pos(:, :, 1), ~] = kmeans(x, k); % isnt using k-means to
    initialize cheating?
% [i_ci, ~] = kmeans(reshape(swarm_pos, [n_e, k*d]), m); % i_ci is herd
    indices



[swarm_fitness, c] = fitness(swarm_pos, x);
[swarm_pos, i_ci, swarm_fitness, ~] = sort_swarm(swarm_pos, i_ci,
    swarm_fitness, c);
gworst = swarm_pos(:, :, end); gworst_fitness = swarm_fitness(end); %
    iteration worst
gbest = swarm_pos(:, :, 1); gbest_fitness = swarm_fitness(1); % global best
pbest = swarm_pos; pbest_fitness = swarm_fitness;
last_best = gbest;
```

```
gbest_fitness_history = zeros([T, 1]); % plot the gbest best fitnesses

% from Y. Duan et al.
for t = 1:T
    beta = beta_min + (beta_max - beta_min) * (1 - (t / T)^3)^2;
    alpha = abs(beta * sin(3*pi/2 + sin(beta * 3*pi/2)));

    % update elephants
    for ci = 1:m
        i_m(ci) = find(i_ci==ci, 1);
        n_ci = sum(i_ci==ci);
        herd = swarm_pos(:, :, i_ci==ci);

        % update best
        if t / T < .1
            C_sigma = trnd(1, k, d);
            herd(:, :, 1) = herd(:, :, 1) + C_sigma; % cauchy random (34)
%                 herd(:, :, 1) = herd(:, :, 1) + beta * (herd(:, :, 1) - mean
    (herd, 3));
        else
%             herd(:, :, 1) = herd(:, :, randi([1, n_ci], 1)) + 2 * (rand(size
    (herd(:, :, 1))) - 0.5) .* (herd(:, :, randi([1, n_ci], 1)) - herd(:, :,
    randi([1, n_ci], 1)));
                herd(:, :, 1) = herd(:, :, 1) + beta * (mean(herd, 3) - herd
                    (:, :, 1));

        end

        for j = 2:n_ci
            if rand < psr
%                 herd(:, :, j) = herd(:, :, j) + rand(size(herd(:, :, j))) .*
     ( (pbest(:, :, j) + gbest) / 2 - herd(:, :, j)) + rand(size(herd(:, :, j)
    )) .* ((pbest(:, :, j) - gbest) / 2 - herd(:, :, j));
                herd(:, :, j) = herd(:, :, j) + rand(size(herd(:, :, j))) .* (
                     (last_best + gbest) / 2 - herd(:, :, j)) + rand(size(herd
                    (:, :, j))) .* ((last_best - gbest) / 2 - herd(:, :, j));

            else
                herd(:, :, j) = herd(:, :, j) + alpha * rand(size(herd(:, :, j
                    ))) .* (herd(:, :, 1) - herd(:, :, j));
            end
        end

        swarm_pos(:, :, i_ci==ci) = herd;
    end


    % update worst
    for ci = 1:m
        K = (-1 + 2*rand([k, d])) .* exp(-2*t / T);
        herd = swarm_pos(:, :, i_ci==ci);
        herd(:, :, end) = herd(:, :, end) + .1 * (x_max - x_min) .* rand([k, d
            ]) + K;
```

```
        swarm_pos(:, :, i_ci==ci) = herd;
    end

    [swarm_fitness, c] = fitness(swarm_pos, x);
    [swarm_pos, i_ci, swarm_fitness, c, i_sort] = sort_swarm(swarm_pos, i_ci,
       swarm_fitness, c);
    pbest = pbest(:, :, i_sort); pbest_fitness = pbest_fitness(i_sort);

    for j = 1:n_e
        if rand < .5
            x_leo = LEO(swarm_pos, j, gbest, gworst, alpha, x_min, x_max);
            [x_leo_f, c2] = fitness(x_leo, x);
            if x_leo_f >= swarm_fitness(j)
                swarm_pos(:, :, j) = x_leo;
                swarm_fitness(j) = x_leo_f;
                c(:, j) = c2;
            end
        end
        if swarm_fitness(j) >= gbest_fitness
                gbest_fitness = swarm_fitness(j);
                gbest = swarm_pos(:, :, j);
                cbest = c(:, j);
        end
        if swarm_fitness(end) <= gworst_fitness
            gworst_fitness = swarm_fitness(end);
            gworst = herd(:, :, end);
        end
    end

    for j = 1:n_e
            elephant = swarm_pos(:, :, j);
            elephant(elephant<x_min) = x_min(elephant<x_min);
            elephant(elephant>x_max) = x_max(elephant>x_max);
            swarm_pos(:, :, j) = elephant;
    end


    [swarm_fitness, c] = fitness(swarm_pos, x);
    [swarm_pos, i_ci, swarm_fitness, c, i_sort] = sort_swarm(swarm_pos, i_ci,
       swarm_fitness, c);
    pbest = pbest(:, :, i_sort); pbest_fitness = pbest_fitness(i_sort);
    pbest(:, :, swarm_fitness>=pbest_fitness) = swarm_pos(:, :, swarm_fitness
       >=pbest_fitness);

    gbest_fitness_history(t) = gbest_fitness;

    % check if gbest is changing
%     if abs(last_fitness - swarm_fitness(1)) < tol
%         [t -gbest_fitness]
%         break
%     end

    last_best = swarm_pos(:, :, 1);
```

```
    if mod(t, 10) == 0
        [t -gbest_fitness] % print progress
    end
end

end

%% fitness/error functions
function [swarm_pos, i_ci, swarm_fitness, c, i_sort] = sort_swarm(swarm_pos,
    i_ci, swarm_fitness, c)
    [swarm_fitness, i_sort] = sort(swarm_fitness, 'descend');
    swarm_pos = swarm_pos(:, :, i_sort);
    i_ci = i_ci(i_sort);
    c = c(:, i_sort);
end
```

```
function x_leo = LEO(swarm_pos, n, gbest, gworst, alpha, x_min, x_max)
%LEO  returns an operator that moves a solution from local optima
%   beta_min = 0.2, beta_max = 1.2 as described in the paper
% Inputs:
%   swarm_pos : potential solutions
%   n             : solution index (swarm(:, :, n) = x_n)
%   gbest        : the (possibly global) best solution
%   gworst      : worst solution
%   [min, max]   : of the dataset X
%   t             : iteration
%   T             : max iterations
%
% Outputs:
%   x_leo        : local escaping operator applied to x
%
%

    epsilon = .01*rand;
    rho1 = 2 * rand * alpha - alpha;
    rho2 = 2 * rand * alpha - alpha;

    [k, d, N] = size(swarm_pos); dim = k*d; % solutions are k*d dimension
    x_n = reshape(swarm_pos(:, :, n), [dim, 1]);
    x_best = reshape(gbest, [dim, 1]);
    x_worst = reshape(gworst, [dim, 1]);
    x_min = reshape(x_min, [dim, 1]);
    x_max = reshape(x_max, [dim, 1]);

    r = randperm(N, 4); % (17)
    x_r = reshape(swarm_pos(:, :, r), [dim, 4]); % pick 4 random solutions

    %% X1
    delta = 2 * rand * abs(mean(x_r, 2) - x_n);
    step = ((x_best - x_r(:, 1)) + delta) / 2;
    delta_x = rand([dim, 1]) .* abs(step);

%     z_n = x_n - randn([dim, 1]) .* (2 * delta_x .* x_n) ./ (x_worst - x_best
    + epsilon);
    z_n = x_n - (rho1 * randn([dim, 1]) .* (2 * delta_x .* x_n) ./ (x_best -
       x_worst + epsilon)) + rand * rho2 * (x_best - x_n); % (22)
    dm = rand * rho2 * (x_best - x_r(:, 1)); % (18, 21)
    yp = rand * (x_n + z_n) ./ 2 + rand * delta_x;
    yq = rand * (x_n + z_n) ./ 2 - rand * delta_x;

    % the gsr uses randn in I. Ahmadianfar et al. (original GSR paper) but not
        in the elephant paper
    gsr = randn .* rho1 * (2 * delta_x .* x_n) ./ (yp - yq + epsilon); % (14)

    X1 = x_n - gsr + dm;

    %% X2
    delta = 2 * rand * abs(mean(x_r, 2) - x_n);
    step = ((x_best - x_r(:, 1)) + delta) / 2;
```

```
    delta_x = rand([dim, 1]) .* abs(step);

%     z_n = x_n - randn * (2 * delta_x .* x_n) ./ (x_worst - x_best + epsilon)
    ; % (22)
    z_n = x_best - (randn([dim, 1]) .* (2 * delta_x .* x_n) ./ (x_best -
        x_worst + epsilon)) + rand * rho2 * (x_r(:, 1) - x_r(:, 2)); % (22)
    yp = rand * (x_n + z_n) ./ 2 + rand * delta_x;
    yq = rand * (x_n + z_n) ./ 2 - rand * delta_x;
    dm = rand * rho2 * (x_r(:, 1) - x_r(:, 2));
    gsr = randn * rho1 * (2 * delta_x .* x_n) ./ (yp - yq + epsilon);
    X2 = x_best - gsr + dm;

    %% X3 and new solution x^{m+1}_n
    rho3 = 2 * rand([dim, 1]) .* alpha - alpha;
    X3 = x_n - rho3 .* (X2 - X1);
    ra = rand([dim, 1]); rb = rand([dim, 1]);
    x_leo = ra .* (rb .* X1 + (1 - rb) .* X2) + (1 - ra) .* X3; % (24)

    %% LEO
    pr = .5; % parameter
    if rand < pr
        f1 = -1 + 2*rand([dim, 1]); f2 = randn([dim, 1]);
        rho1 = 2 * rand([dim, 1]) * alpha - alpha;
        L1 = rand([dim, 1]) < .5; L2 = rand([dim, 1]) < .5;
        u1 = L1 * 2 .* rand([dim, 1]) + (1 - L1);
        u2 = L1 .* rand([dim, 1]) + (1 - L1);
        u3 = L1 .* rand([dim, 1]) + (1 - L1);

        kay = floor(unifrnd(1, length(swarm_pos))) + 1;

        x_k = reshape(swarm_pos(:, :, kay), [dim, 1]);
        x_rand = x_min + rand([dim, 1]) .* (x_max - x_min);

        x_p = L2 .* x_k + (1 - L2) .* x_rand;

        if u1 < .5
            x_leo = x_leo + f1 .* (u1 .* x_best - u2 .* x_p) + f2 .* rho1 .* (
                u3 .* (X2 -X1) + u2 .* (x_r(:, 1)- x_r(:, 2))) ./ 2;
        else
            x_leo = x_best + f1 .* (u1 .* x_best - u2 .* x_p) + f2 .* rho1 .*
                (u3 .* (X2 -X1) + u2 .* (x_r(:, 1)- x_r(:, 2))) ./ 2;
        end
    end

    x_leo = reshape(x_leo, [k, d]);
end
```

D. Other

```
function [scores, C, D2, thetas] = fitness(centroids, x, options)
%FITNESS returns the f_name score for clustering x using centroids c
% Inputs:
%   centroids            : centroids
%   x                        : datapoints (rows)
%   f_name               : fitness function name (CH, Silhouette, SSE)
%
% Outputs:
%   scores  : fitness score

    [k, d, n_e] = size(centroids, [1, 2, 3]); % n_e elephants, k clusters, d
       dimensions
    [n, ~] = size(x); % n data points

   %% assign to clusters
    D = zeros(n, n_e); C = zeros(n, n_e);
    D_full = zeros(n, k, n_e); C_full = zeros(n, k, n_e);
    counts = zeros(k, n_e);
    scores = zeros(n_e, 1);

    for j = 1:n_e
        % k smallest distances
        [D_j, c_j] = pdist2(centroids(:, :, j), x, 'squaredeuclidean', '
           Smallest', k);
        D(:, j) = min(D_j, [], 1);
        C(:, j) = c_j(1, :);
        D_full(:, :, j) = D_j';
        C_full(:, :, j) = c_j';
        for dim = 1:size(c_j, 1)
            counts(dim, j) = sum(C(:, j)==dim);
        end
    end

    for j = 1:n_e
            D2 = zeros([length(x), 1]);
            cl_scores = zeros(1, k);
            for ell = 1:k
                x_in_c = x(C(:, j)==ell, :);
                cl_center = mean(x_in_c);
                cl_scores(ell) = sum((x_in_c - cl_center).^2, 'all');
                D2(C(:, j)==ell) = sum((x_in_c-cl_center).^2, 2);
            end
            scores(j) = -sum(cl_scores); %% !uses negative SSE!
    end
end
```