Implementation of wolf pack algorithm for global unconstrained optimization problems

Yash Shethwala
School of Computing and Augmented Intelligence (SCAI)
Arizona State University
Tempe, AZ
yashshethwala@asu.edu

Praveen Tamrakar
School of Electrical, Computer and Energy Engineering (ECEE)
Arizona State University
Tempe, AZ
ptamraka@asu.edu

***Abstract*—This paper mentions one of the bio-inspired optimization algorithms which can be used very efficiently to optimize functions robustly. The algorithm presented here is inspired by the pack of wolf in the nature. The wolfs are one of the best hunters of the nature and they hunt in a group which we can call as a pack.**

Keywords—Wolf pack, algorithm, optimization, metaheuristic, bio-inspired.

I. Introduction

Nowadays, the optimization is becoming a very important part of the areas like engineering, economics, production, etc. Global optimization, especially using classical approaches, is becoming a difficult undertaking as many real-world issues get more complicated [2]. The problems are getting more complex by more dimensions being added up which make it more difficult to solve [1]. Luckily, numerous algorithms inspired by nature have evolved into effective solutions to these issues [3-5]. The general formulation for any unconstrained optimization problem can be seen in equation 1.

$$\text{min or max } f(X), \quad X = (x_1, x_2, x_3, \dots\dots, x_n) \tag{1}$$

One can take inexhaustible inspiration from the magnificent swarm's behaviors that are present in nature like ant colonies, flock of birds, school of fish etc. In recent decades, there were many algorithms formed by many people like in 1995, Particle Swarm Optimization (PSO) inspired by flock of birds was proposed by Kennedy [7]; Dorigo ,inspired from ant colonies, suggested Ant Colony Optimization (ACO) in 1996 [9]; artificial fish swarm algorithm (AFSA) , inspired by foraging behaviour of fish schools, created by Li in 2002 [6]; Karaboga introduced the artificial bee colony (ABC) algorithm in 2005, inspired by honeybee swarms' sophisticated foraging behaviour [8]; rats herds algorithm, mosquito swarms algorithm, and dolphins herds algorithm are some examples of swarm intelligent algorithms proposed by researchers [11]. These animals lack the ability to tackle a complex problem because they don't have a human brain. However, they all have one thing in common i.e., food, which they are attempting to obtain. As a result of the harsh conditions of the environment and natural limits, they created a powerful swarm intelligence, which solved the problem through mutual collaboration and understanding. Came up with new ideas to enable them to hunt or obtain food using this method. We're studying their procedure for obtaining food so that we would implement it to solve our optimization challenge.

The wolf pack is incredible. Their strict organization structure and sophisticated hunting behaviour have evolved over centuries as a result of their harsh living environment and ongoing evolution. The use of wolf techniques is not a new concept; people from various countries are attempting to implement this way in order to combat others. For example, Wolves tactics of Mongolia cavalry in Genghis Khan period, submarine tactics of Nazi Admiral Doenitz in World War II and U.S. military wolves attack system for electronic countermeasures all highlight great charm of their swarm intelligence [12]. To solve the optimization problem, a wolf colony algorithm (WCA) is proposed [10]. However, WCA's precision and efficiency are insufficient, and it is not giving a perfect solution when use for high dimensional functions. Hu-Sheng Wu and Feng-Ming Zhang reanalysed collaborative predation behaviour and prey distribution mode of wolves and proposed a new swarm intelligence algorithm, the wolf pack algorithm (WPA) [12]. In this paper, we are attempting to model wolf tactics, using [12], in such a way that we can obtain a good solution for complex function in 8-10 iterations of the algorithms. We will utilize three different complex and different functions that are difficult to solve in the old algorithms.

II. Wolf Pack System Analysis

The wolf pack is divided into three types of groups: the lead wolf, elite wolves who work as scouts, and ferocious wolves. They are cohabiting, surviving, and hunting together, as well as taking responsibility for their individual groups. They are deeply concerned for one another and their families.

To begin with, the lead wolf is the most important wolf in the pack because it is the sharpest and most ferocious wolf who will give orders to the other wolves and be responsible for all decisions. It is the leader of their clan. It determines when we must hunt, when elite groups must go scouting, and when ferocious wolves must hunt since it has the best ability to smell prey and hunt. It's also in charge of analysing the environment and any potentially dangerous conditions and making decisions that allow them to hunt the prey without danger or risk.

Second, when the leader wolf considers it, it's time to hunt the prey. As a result, the lead wolf dispatches an elite group wolf to track down the prey. They disperse and independently explore over different zones in search of the prey's scent. They decide to go in that way based on where they sense the prey's scent. And they continue to migrate toward the smell's higher concentration. As a result, they locate the prey by heading in the direction of the prey.

Third, once the elite wolf group locates the prey, whichever wolf discovers the prey will howl and alert the lead wolf, who will then arrive at the location and assess the situation. If the situation is not dangerous for the wolf to capture the prey, the lead wolf will summon the ferocious wolf to hunt the prey. If the ferocious wolf is summoned by the lead wolf, the ferocious wolf will arrive quickly, round up the prey, and hunt it down.

Fourth, food is not distributed equally once the wolf pack has hunted down the prey; this is the wolf pack's rule. The rule is that the stronger the wolf, the more food he gets, and the weaker the wolf, the less food he gets. Over completing this process, the weak wolf continues to receive less food, and after a period, those wolves die without nourishment. This process occurred because the strong wolf gained more strength and was able to effortlessly chase down the prey. That is the motivation for the rule.

III. Wolf Pack Algorithm

A. Few Definitions

The predatory space of the wolf pack is considered as $N \times D$ Euclidean space, N is the number of wolves and D is the number of dimensions(variables) of the function to be optimized. The position of single wolf i is a vector $X = (x_{i1}, x_{i2}...., x_{id})$ where x_{id} is the dth variable value of the ith artificial wolf. Also, the prey's smell concentration, $Y = f(X)$, perceived by artificial wolves is the objective function's value.

The distance between two wolves p and q can be represented as L(p,q) which we have used is the Euclidean distance as our functions to be optimized are continuous. However, one can use Manhattan distance for continuous function optimization or hamming distance for discrete optimization. Furthermore, while the issues of maximum value and minimal value are interchangeable, we have discussed only maximization problem in the algorithm discussion. Also, the wolves will move by taking different step lengths in different steps. The step lengths will have the following relationship as shown in equation (2). In resolution space, S stands for step coefficient and denotes the finer degree of an artificial wolf seeking for food.

$$step_a^d = \frac{step_b^d}{2} = 2 \cdot step_c^d = S \qquad (2)$$

B. Intelligent Behaviors and rules Desciption of Wolf pack

The collaboration of the lead wolf, scout wolves, and feracious wolves result in virtually perfect hunting, and the wolves can prosper in the correct direction due to the prey distribution from strong to weak. Their 3 intelligent behaviors i.e., scouting, calling and besieging behaviors, and two intelligent rules i.e., winner-take-all generating rule for the lead wolf and stronger-survive renewing rule for the wolf pack, are abstracted from their entire predatory behavior of wolf pack. The description of all the rule and behaviors can be found as below.

1) The winner-take-all generating rule for the lead wolf: The lead wolf is selected who has the best objective function value. The lead wolf's function value is compared to other wolves; if the value of other wolf is better, the lead wolf will be replaced and the best will become lead. The lead will directly go to the next iteration without performing any behaviours if it is not replaced by some other better wolf.

2) Scouting Behavior: The *S_num* of elite wolves except for lead wolves will do the souting behaviour and search for the prey in the predatory space. Here, Y_i will be the concentration of prey's smell perceived by ith wolf and Y_{lead} is the concentratrion perceived by lead wolf.

If $Y_i > Y_{lead}$, the prey is nearer to the scout wolf and can catch prey. Thus, the scout wolf will become lead wolf and $Y_{lead} = Y_i$.

If $Y_i < Y_{lead}$, the scout wolf will search its surroundings by taking a step in h different directions with a step length of $step_a$. the state of the scout wolf after taking a step can be formulated as below:

$$x_{id}^{p} = x_{id} + \sin\left(2\pi \times \frac{p}{h}\right) \times step_a^d, \quad p = \{1, 2..., h\} \qquad (3)$$

Here, h will be randomly selected integer from $[h_{min}, h_{max}]$ and will be different for all the wolves. Now, the current concentration of prey smell perceived by wolf i, Y_{0i} will be compared to Y_{ip} which is the concentration of prey's smell after talking the step in pth direction. If $\max\{Y_{i1}, Y_{i2}..., Y_{ip}\} > Y_{0i}$, the wolf i takes a step in the direction of maximum step and its position Xi is updated using (3). This step will be repeated until $Y_i > Y_{lead}$ or the maximum number of iterations T_{max} is achieved for that behavior.

3) Calling behavior: The lead wolf will summon the *C_num* feracious wolves by howling to gather around the prey. The feracious wolves will accumulate around the lead wolf by taking step size of $step_b$ as the location of lead wolf will be considered as the prey. g_d^k is the lead wolf's position in the dth variable space and at kth iteration. The position of the wolf i at kth iteration will be updated according to the equation (4).

$$x_{id}^{k+1} = x_{id}^{k} + step_b^d \cdot \frac{\left(g_d^k - x_{id}^k\right)}{\left|g_d^k - x_{id}^k\right|} \qquad (4)$$

If $Y_i > Y_{lead}$, the wolf i will become the lead wolf and $Y_{lead}=Y_i$. After this, all the wolves will take the scouting behavior. If $Y_i < Y_{lead}$, the wolves will continue to accumulate towards the lead wolf until $L(i,l)<L_{near}$ and then takes the besieging behavior. L_{near} is the used to judge the condition and $L(i,l)$ is the distance between wolf i and lead wolf.

The notion of social cognition is blended with the idea of calling behavior, which illustrates an information transmitting and sharing process in a wolf pack.

4) Besieging behaviour: After accumulating around the lead wolf, except for the lead wolf all the wolves take the

beseiging behaviour with step length of *step*$_c$ to kill the prey. Now prey's positon will be same as the lead wolf i.e., G_d^k in the dth variable space at kth iteration. The position of wolf i will be updated using equation (5).

$$x_{id}^{k+1} = x_{id}^{k} + \lambda \cdot step_c^d \cdot \left|G_d^k - x_{id}^k\right| \tag{5}$$

Here, λ is randomly generated in uniformly distributed range of [-1,1]. The position of wolf i is updated if and only if $Y_{ik} > Y_{i0}$, where Y_{i0} is the concentration of smell of prey at current position and Y_{ik} is one after taking the step.

5) The stronger-survive renewing rule for the wolf pack: In this step, the distribution of prey will be done and the stronger wolves will get more food while weak wolves will get less food which results in their death. Thus, we simulate this behaviour in this step by making dead R wolves while generating new R wolves around the lead wolf. The position of new ith wolf in dth variable space can be calculated using eqaution (6).

$$x_{id} = g_d \cdot (1 + rand), \qquad i = \{1,2\ldots, R\} \tag{6}$$

Here, g_d represents the position of lead wolf in the dth dimension space and *rand* is a uniformly distributed random number between [-0.1, 0.1]. An important thing is never deleting the leader.

The greater the value of R, the better for preserving wolf's variety and allowing the algorithm to open up additional resolution space. However, if R is too big, the process will resemble a random search. As the preys captured are different in every iteration, the weak dead wolves are different and so we need to select R randomly as an integer in the range $[n/(2\cdot\beta), n/\beta]$, where β is the population renewing proportional coefficient.

C. *Description of the Algorithm*

To begin with, the scouting behaviour increases the likelihood that WPA will be able to thoroughly explore the solution space. Second, the winner-take-all rule for creating lead wolves, as well as the calling behaviour, cause the wolves to flock to the lead wolf who is closest to the prey and is most likely to catch it. Because wolves take the biggest step in calling behaviour, the winner-take-all rule and calling behaviour cause wolves to reach in the vicinity of the global optimum only after a few repetitions. Finally, WPA algorithm's besieging behaviour with a little step, *step*$_c$, allows it to open up fresh solution space and carefully explore the global optima in a favourable solution region. The figure 1 shows the flowchart for this algorithm.

IV. RESULTS

The main components of WPA are explained in previous section. The authors have used 3 off-the-shelf test functions Auckley, Booth and Easom which can be found in equation (7), (8) and (9) respectively.

$$f(x,y) = -20e^{-20\sqrt{0.5(x^2+y^2)}} - e^{-0.5(\cos 2\pi x + \cos 2\pi y)} + e + 20 \tag{7}$$

$$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2 \tag{8}$$

$$f(x,y) = -\cos(x)\cos(y)\, e^{-((x-\pi)^2+(y-\pi)^2)} \tag{9}$$

The 3D and 2D graphs for all the functions can be found in figure 2, figure 3 and figure 4 respectively.

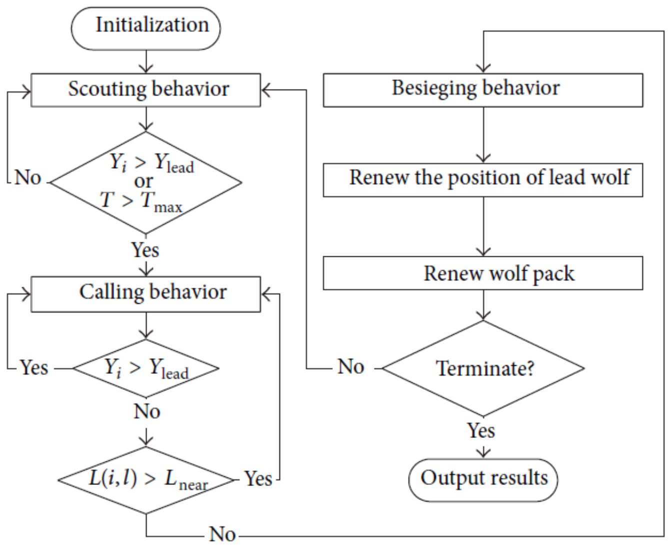

Figure 1. The flowchart for wolf pack algorithm

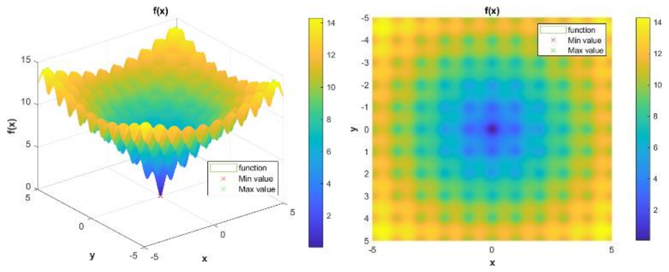

Figure 2. Auckley function in 3D and 2D

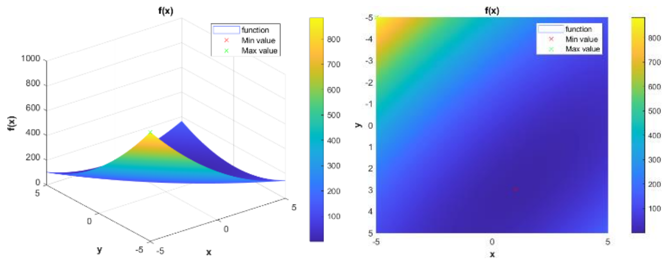

Figure 3. Booth function in 3D and 2D

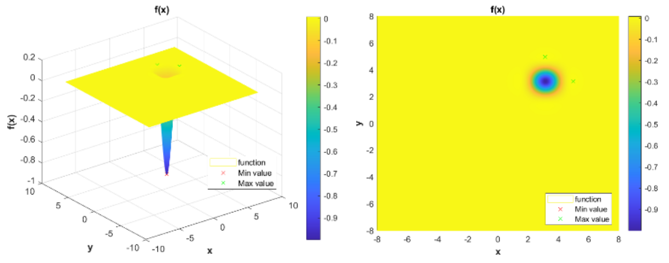

Figure 4. Easom function in 3D and 2D

The wolf pack algorithm was iterated for 30 iterations for all the functions with the parameter values as mentioned in the table 1. Here, the functions are minimized.

These functions are optimized using wolf pack algorithm and one can see best, worst and average fitness in figure 5, figure 6 and figure 7 for auckley, booth, and easom functions respectively.

Table 1. Hyper-parameter values

Parameter	**Value**
Number of wolves (N)	20
Max no. of iterations (k_{max})	30
Step coefficient (S)	0.12
L_{near}	0.3
T_{max}	8
β	2

One can notice from the results that in every function the algorithm gets close to global optima in almost first 10 iterations. This means that it can be exploration phase. After that, the value gets close to the optimum value as the algorithm goes into exploitation phase.

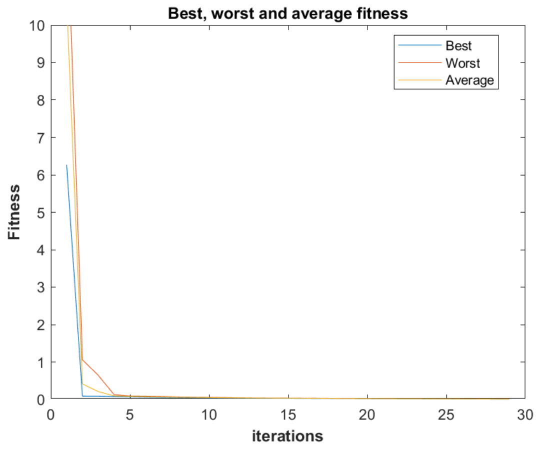

Figure 5. Auckley function's best, worst and average fitness graph for all iterations

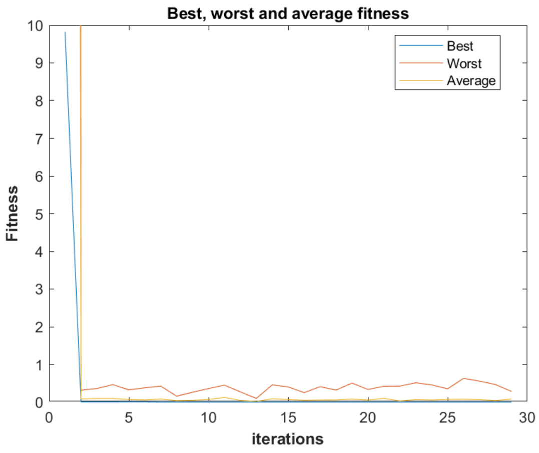

Figure 6. Booth function's best, worst and average fitness graph for all iterations

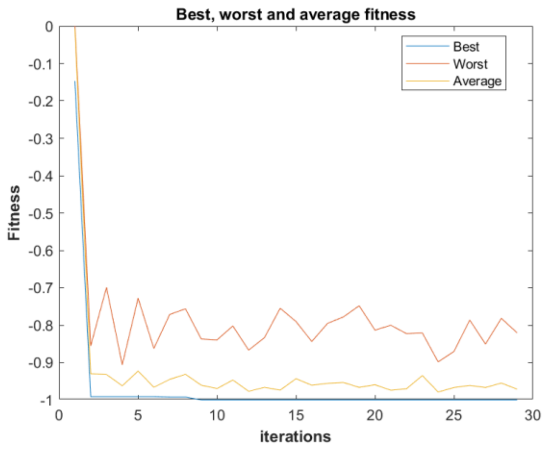

Figure 7. Easom function's best, worst and average fitness graph for all iterations

V. Conclusion and Discussion

We can see from figure 5 that the best, worst and average individuals become same after iteration 15 as there is gradual decrease in auckley function equally form all sides. Moreover, the algorithm doesn't get trapped in local optima due to the swarm nature of algorithm. The figure 6 suggests that the booth function's worst fluctuates as there is unequal gradual decrease so the individual can acquire value around minima easily. The figure 7 has much more fluctuations than other two functions as the easom function has a sharp fall near the global minima. If the individual has slight change in its value, then there will be huge change in its function value.

By doing this experiment, we can conclude that the values of hyper-parameters N, S and L_{near} would affect the performance of wolf pack algorithm. More the number of wolves (N), it will take more time to converge but we will get greater accuracy. On the other hand, if we decrease the number of wolves (N), it will take less time to converge but it will also decrease the accuracy. The hyper-parameter L_{near} will increase the exploitation if we decrease its value and vice-versa. With lower value of L_{near}, it goes into such a state where it will take a lot of time to converge, or it will never converge as it won't be exploring new spaces. No doubt the higher value of S will make the convergence faster but when it reaches to besieging behavior, it may oscillate around the prey. Also, if we keep the value of S smaller it will take higher time to converge and also to complete besieging behavior. Thus, an optimal value of S needs to be found out and for this case one can find it in table 1.

References

[1] C. Grosan and A. Abraham, "A novel global optimization technique for high dimensional functions," International Journal of Intelligent Systems, vol. 24, no. 4, pp. 421–440, 2009.

[2] F. Kang, J. Li, and Z.Ma, "Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions," *Information Sciences*, vol. 181, no. 16, pp. 3508–3531, 2011.

[3] Y. Yang, Y.Wang, X. Yuan, and F. Yin, "Hybrid chaos optimization algorithm with artificial emotion," Applied Mathematics and Computation, vol. 218, no. 11, pp. 6585–6611, 2012.

[4] W. S.Gao andC. Shao, "Pseudo-collision in swarm optimization algorithm and solution: rain forest algorithm," Acta Physica Sinica, vol. 62, no. 19,Article ID 190202, pp. 1–15, 2013.

[5] Y. Celik and E. Ulker, “An improved marriage in honey bees optimization algorithm for single objective unconstrained optimization,” The Scientific World Journal, vol. 2013,Article ID 370172, 11 pages, 2013.

[6] M. Dorigo, Optimization, learning and natural algorithms [Ph.D. thesis], Politecnico di Milano, Milano, Italy, 1992.

[7] D. Karaboga, “An idea based on honeybee swarm for numerical optimization,” Tech. Rep. TR06, Computer Engineering Department, Engineering Faculty, Erciyes University, Kayseri, Turkey, 2005.

[8] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in Proceedings of the IEEE International Conference on Neural Networks, pp. 1942–1948, Perth, Australia, December 1995.

[9] X.-L. Li, Z.-J. Shao, and J.-X. Qian, “Optimizing method based on autonomous animats: Fish-swarm Algorithm,” System Engineering Theory and Practice, vol. 22, no. 11, pp. 32–38, 2002.

[10] C.-G. Liu, X.-H. Yan, and C.-Y. Liu, “The wolf colony algorithm and its application,” Chinese Journal of Electronics, vol. 20, no. 2, pp. 212–216, 2011.

[11] J. A. Ruiz-Vanoye, O. D´ıaz-Parra, F. Coc´on et al., “Meta-Heuristics algorithms based on the grouping of animals by social behavior for the travelling sales problems,” International Journal of Combinatorial Optimization Problems and Informatics, vol. 3, no. 3, pp. 104–123, 2012.

[12] Hu-Sheng Wu, Feng-Ming Zhang, "Wolf Pack Algorithm for Unconstrained Global Optimization", Mathematical Problems in Engineering, vol. 2014, Article ID 465082, 17 pages, 2014. https://doi.org/10.1155/2014/465082

APPENDIX

1) The wolf pack algorithm MATLAB code for Aukley function optimization.

```
clc;
clear all;
close all;

% Auckley fucntion
auckley = @(x1,x2) (-20*exp(-
0.2*sqrt(0.5.*(x1.^2+x2.^2)))-
exp(0.5*(cos(2*pi*x1)+cos(2*pi*x2)))+exp(1)
+20);

plotfn(auckley, [-5,5], [-5,5]);
plotfn(auckley, [-5,5], [-5,5]);
view(0,-90)

%%% Wolf pack algorithm parameters
initialization
% number of wolfs
N = 20;
% number of dimensions
D = 2;
% maximum number of iterationsw
kmax = 30; k=1;
% step coefficient
S = 0.5;
% Distance determinant coefficient
Lnear = 0.3; %0.08
% max number of  iterations in scouting
behaviour
Tmax = 8; T=0;
% population renewing proportional constant
beta = 2;
% steps
stepa = S; stepb = 2*S; stepc = S/2;
% h limits
hmin = 10; hmax= 20;
% R to delete wolves from end
Rmin = round(N/(2*beta)); Rmax =
round(N/beta);

%%% initial position
xmin = -5;
xmax = 5;
X = (xmax-xmin).*rand(N,D) + xmin;
m = X(:,1); n = X(:,2);
hold on
p = plot(m, n, "or");
hold off

p.XDataSource = "m";
p.YDataSource = "n";

%%% starting iterations
while k<kmax
    step = 1; overfl=0;
    while ~overfl
        if step==1
            % finding lead
            fitness =
auckley(X(:,1),X(:,2));
            [lead, idx_lead] =
min(fitness);

            % data for plots
            [best(k), idx_best] =
min(fitness);
            worst(k) = max(fitness);
            avg(k) = mean(fitness);

            step=2;
        elseif step==2
            % scouting
            brkfl = 1;
            while T<Tmax && brkfl
                for i=1:N
                    if i~=idx_lead
                        h =
randi([hmin,hmax], 1);
                        movefl = 0; x1new =
-6; x2new = -6; fitnessnew = fitness(i);
                        for j=1:h
                            X1new = X(i,1)
+ (stepa * sin(2*pi*(j/h)));
                            X2new = X(i,2)
+ (stepa * sin(2*pi*(j/h)));
                            if
auckley(X1new,X2new)<fitnessnew
```

```
                                x1new =
X1new; x2new = X2new; fitnessnew =
auckley(X1new,X2new);
                                movefl = 1;
                            end
                        end
                        if movefl
                            X(i:1) = x1new;
X(i:2) = x2new;
                            fitness(i) =
fitnessnew;
                            m = X(:,1); n =
X(:,2);
                            refreshdata
                            drawnow
                        end
                        if
auckley(X(i,1),X(i,2))<lead
                            lead =
auckley(X(i,1),X(i,2));
                            idx_lead = i;
                            brkfl=0;
                            break
                        end
                    end
                end
                T = T + 1;
            end
            step=3;
        elseif step==3
            % Calling
            distfl = 1;
            while distfl
                moveable_wolves = [];
                for i=1:N
                    dist(i) =
sqrt((X(idx_lead,1)-X(i,1))^2 +
(X(idx_lead,2)-X(i,2))^2);
                    if dist(i)>Lnear
                        moveable_wolves =
[moveable_wolves i];
                    end
                end
                if isempty(moveable_wolves)
                    disfl=0;
                    break
                end
                for
i=1:length(moveable_wolves)
                    x1old =
X(moveable_wolves(i),1); x2old =
X(moveable_wolves(i),2);
                    x1new = X(idx_lead,1);
x2new = X(idx_lead,2);
                    X(moveable_wolves(i),1)
= x1old + stepb * ((x1new-x1old)/abs(x1new-
x1old));
                    X(moveable_wolves(i),2)
= x2old + stepb * ((x2new-x2old)/abs(x2new-
x2old));
                    m = X(:,1); n = X(:,2);
                    refreshdata
                    drawnow
                    if
auckley(X(moveable_wolves(i),1),
X(moveable_wolves(i),2))<lead
                        lead =
auckley(X(moveable_wolves(i),1),
X(moveable_wolves(i),2));
                        idx_lead =
moveable_wolves(i);
                        step=2;
                        distfl = 0;
                        break
                    end
                end
            end
            if step==3
                step=4;
            end
        elseif step==4
            % Besieging
            for i=1:N
                if i~=idx_lead
                    x1old = X(i,1); x2old =
X(i,2);
                    x1lead = X(idx_lead,1);
x2lead = X(idx_lead,2);
                    x1new = x1old +
(2*rand-1) * stepc * (abs(x1lead-x1old));
                    x2new = x2old +
(2*rand-1) * stepc * (abs(x2lead-x2old));
                    if auckley(x1new,
x2new)<auckley(x1old, x2old)
                        X(i,1) = x1new;
X(i,2) = x2new;
                        m = X(:,1); n =
X(:,2);
                        refreshdata
                        drawnow
                        if auckley(x1new,
x2new)<lead
                            lead =
auckley(x1new, x2new);
                            idx_lead = i;
                        end
                    end
                end
            end
            step=5;
        elseif step==5
            % stronger surviving renewing
            fitness = 
auckley(X(:,1),X(:,2));
            [~, idx_sorted] = sort(fitness, 
"descend");
            R = randi([Rmin, Rmax], 1);
            for i=1:R
                    X(idx_sorted(i),1) = 
X(idx_lead, 1) * (1+(0.2*rand-0.1));
                    X(idx_sorted(i),2) = 
X(idx_lead, 2) * (1+(0.2*rand-0.1));
                    m = X(:,1); n = X(:,2);
                    refreshdata
                    drawnow
            end
            overfl=1;
        end
    end

    k = k + 1;
    fprintf("iteration: %d\n", k)
end

%%% Plotting fitness over time
figure;
plot(best);
hold on
plot(worst); plot(avg);
hold off
ylim([0, 10])
title("Best, worst and average fitness");
xlabel("iterations","FontWeight", "bold");
ylabel("Fitness", "FontWeight", "bold");
legend("Best", "Worst", "Average", 
"Location", "best");
```

2) The wolf pack algorithm MATLAB code for Booth function optimization.

```
clc;
clear all;
close all;

% Booth function
booth = @(x1,x2) (x1+2*x2-7).^2 + (2*x1+x2-
5).^2;

plotfn(booth, [-5,5], [-5,5]);
plotfn(booth, [-5,5], [-5,5]);
view(0,-90)

%%% Wolf pack algorithm parameters 
initialization
% number of wolfs
N = 20;
% number of dimensions
D = 2;
% maximum number of iterations
kmax = 30; k=1;
% step coefficient
S = 0.12;
% Distance determinant coefficient
Lnear = 0.3; %0.08
% max number of  iterations in scouting 
behaviour
Tmax = 8; T=0;
% population renewing proportional constant
beta = 2;
% steps
stepa = S; stepb = 2*S; stepc = S/2;
% h limits
hmin = 10; hmax= 20;
% R to delete wolves from end
Rmin = N/(2*beta); Rmax = N/beta;

%%% initial position
xmin = -5;
xmax = 5;
X = (xmax-xmin).*rand(N,D) + xmin;
m = X(:,1); n = X(:,2);
hold on
p = plot(m, n, "or");
hold off

p.XDataSource = "m";
p.YDataSource = "n";

%%% starting iterations
while k<kmax
    step = 1; overfl=0;
    while ~overfl
        if step==1
            % finding lead
            fitness = booth(X(:,1),X(:,2));
            [lead, idx_lead] = 
min(fitness);

            % data for plots
            [best(k), idx_best] = 
min(fitness);
            worst(k) = max(fitness);
            avg(k) = mean(fitness);

            step=2;
        elseif step==2
            % scouting
            brkfl = 1;
            while T<Tmax && brkfl
                for i=1:N
                    if i~=idx_lead
                        h = 
randi([hmin,hmax], 1);
```

```
                        movefl = 0; x1new =
-6; x2new = -6; fitnessnew = fitness(i);
                        for j=1:h
                            X1new = X(i,1)
+ (stepa * sin(2*pi*(j/h)));
                            X2new = X(i,2)
+ (stepa * sin(2*pi*(j/h)));
                            if
booth(X1new,X2new)<fitnessnew
                                x1new =
X1new; x2new = X2new; fitnessnew =
booth(X1new,X2new);
                                movefl = 1;
                            end
                        end
                        if movefl
                            X(i:1) = x1new;
X(i:2) = x2new;
                            fitness(i) =
fitnessnew;
                            m = X(:,1); n =
X(:,2);
                            refreshdata
                            drawnow
                        end
                        if
booth(X(i,1),X(i,2))<lead
                            lead =
booth(X(i,1),X(i,2));
                            idx_lead = i;
                            brkfl=0;
                            break
                        end
                    end
                end
                T = T + 1;
            end
            step=3;
        elseif step==3
            % Calling
            distfl = 1;
            while distfl
                moveable_wolves = [];
                for i=1:N
                    dist(i) =
sqrt((X(idx_lead,1)-X(i,1))^2 +
(X(idx_lead,2)-X(i,2))^2);
                    if dist(i)>Lnear
                        moveable_wolves =
[moveable_wolves i];
                    end
                end
                if isempty(moveable_wolves)
                    disfl=0;
                    break
                end
                for
i=1:length(moveable_wolves)
                    x1old =
X(moveable_wolves(i),1); x2old =
X(moveable_wolves(i),2);
                    x1new = X(idx_lead,1);
x2new = X(idx_lead,2);
                    X(moveable_wolves(i),1)
= x1old + stepb * ((x1new-x1old)/abs(x1new-
x1old));
                    X(moveable_wolves(i),2)
= x2old + stepb * ((x2new-x2old)/abs(x2new-
x2old));
                    m = X(:,1); n = X(:,2);
                    refreshdata
                    drawnow
                    if
booth(X(moveable_wolves(i),1),
X(moveable_wolves(i),2))<lead
                        lead =
booth(X(moveable_wolves(i),1),
X(moveable_wolves(i),2));
                        idx_lead =
moveable_wolves(i);
                        step=2;
                        distfl = 0;
                        break
                    end
                end
            end
            if step==3
                step=4;
            end
        elseif step==4
            % Besieging
            for i=1:N
                if i~=idx_lead
                    x1old = X(i,1); x2old =
X(i,2);
                    x1lead = X(idx_lead,1);
x2lead = X(idx_lead,2);
                    x1new = x1old +
(2*rand-1) * stepc * (abs(x1lead-x1old));
                    x2new = x2old +
(2*rand-1) * stepc * (abs(x2lead-x2old));
                    if booth(x1new,
x2new)<booth(x1old, x2old)
                        X(i,1) = x1new;
X(i,2) = x2new;
                        m = X(:,1); n =
X(:,2);
                        refreshdata
                        drawnow
                        if booth(x1new,
x2new)<lead
```

```
                            lead =
booth(x1new, x2new);
                            idx_lead = i;
                        end
                    end
                end
            end
            step=5;
        elseif step==5
            % stronger surviving renewing
            fitness = booth(X(:,1),X(:,2));
            [~, idx_sorted] = sort(fitness,
"descend");
            R = randi([Rmin, Rmax], 1);
            for i=1:R
                    X(idx_sorted(i),1) =
X(idx_lead, 1) * (1+(0.2*rand-0.1));
                    X(idx_sorted(i),2) =
X(idx_lead, 2) * (1+(0.2*rand-0.1));
                    m = X(:,1); n = X(:,2);
                    refreshdata
                    drawnow
            end
            overfl=1;
        end
    end

    k = k + 1;
    fprintf("iteration: %d\n", k)
end

%%% Plotting fitness over time
figure;
plot(best);
hold on
plot(worst); plot(avg);
hold off
ylim([0, 10])
title("Best, worst and average fitness");
xlabel("iterations","FontWeight", "bold");
ylabel("Fitness", "FontWeight", "bold");
legend("Best", "Worst", "Average",
"Location", "best");
```

3) The wolf pack algorithm MATLAB code for Easom function optimization.

```
clc;
clear all;
close all;

% Easom function
easom = @(x1,x2) (-cos(x1).*cos(x2).*(exp(-
((x1-pi).^2+(x2-pi).^2))));

plotfn(easom, [-8,8], [-8,8]);
plotfn(easom, [-8,8], [-8,8]);
view(0,90)

%%% Wolf pack algorithm parameters
initialization
% number of wolfs
N = 20;
% number of dimensions
D = 2;
% maximum number of iterations
kmax = 30; k=1;
% step coefficient
S = 0.12;
% Distance determinant coefficient
Lnear = 0.3; %0.08
% max number of  iterations in scouting
behaviour
Tmax = 8; T=0;
% population renewing proportional constant
beta = 2;
% steps
stepa = S; stepb = 2*S; stepc = S/2;
% h limits
hmin = 10; hmax= 20;
% R to delete wolves from end
Rmin = N/(2*beta); Rmax = N/beta;

%%% initial position
xmin = -5;
xmax = 5;
X = (xmax-xmin).*rand(N,D) + xmin;
m = X(:,1); n = X(:,2);
hold on
p = plot(m, n, "or");
hold off

p.XDataSource = "m";
p.YDataSource = "n";

%%% starting iterations
while k<kmax
    step = 1; overfl=0;
    while ~overfl
        if step==1
            % finding lead
            fitness = easom(X(:,1),X(:,2));
            [lead, idx_lead] =
min(fitness);

            % data for plots
            [best(k), idx_best] =
min(fitness);
            worst(k) = max(fitness);
            avg(k) = mean(fitness);

            step=2;
        elseif step==2
            % scouting
            brkfl = 1;
            while T<Tmax && brkfl
                for i=1:N
                    if i~=idx_lead
                        h =
randi([hmin,hmax], 1);
                        movefl = 0; x1new =
-6; x2new = -6; fitnessnew = fitness(i);
                        for j=1:h
                            X1new = X(i,1)
+ (stepa * sin(2*pi*(j/h)));
                            X2new = X(i,2)
+ (stepa * sin(2*pi*(j/h)));
                            if
easom(X1new,X2new)<fitnessnew
                                x1new =
X1new; x2new = X2new; fitnessnew =
easom(X1new,X2new);
                                movefl = 1;
                            end
                        end
                        if movefl
                            X(i:1) = x1new;
X(i:2) = x2new;
                            fitness(i) =
fitnessnew;
                            m = X(:,1); n =
X(:,2);
                            refreshdata
                            drawnow
                        end
                        if
easom(X(i,1),X(i,2))<lead
                            lead =
easom(X(i,1),X(i,2));
                            idx_lead = i;
                            brkfl=0;
                            break
                        end
                    end
                end
                T = T + 1;
            end
            step=3;
        elseif step==3
            % Calling
            distfl = 1;
            while distfl
                moveable_wolves = [];
                for i=1:N
                    dist(i) =
sqrt((X(idx_lead,1)-X(i,1))^2 +
(X(idx_lead,2)-X(i,2))^2);
                    if dist(i)>Lnear
                        moveable_wolves =
[moveable_wolves i];
                    end
                end
                if isempty(moveable_wolves)
                    disfl=0;
                    break
                end
                for
i=1:length(moveable_wolves)
                    x1old =
X(moveable_wolves(i),1); x2old =
X(moveable_wolves(i),2);
                    x1new = X(idx_lead,1);
x2new = X(idx_lead,2);
                    X(moveable_wolves(i),1)
= x1old + stepb * ((x1new-x1old)/abs(x1new-
x1old));
                    X(moveable_wolves(i),2)
= x2old + stepb * ((x2new-x2old)/abs(x2new-
x2old));
                    m = X(:,1); n = X(:,2);
                    refreshdata
                    drawnow
                    if
easom(X(moveable_wolves(i),1),
X(moveable_wolves(i),2))<lead
                        lead =
easom(X(moveable_wolves(i),1),
X(moveable_wolves(i),2));
                        idx_lead =
moveable_wolves(i);
                        step=2;
                        distfl = 0;
                        break
                    end
                end
            end
            if step==3
                step=4;
            end
        elseif step==4
            % Besieging
            for i=1:N
                if i~=idx_lead
                    x1old = X(i,1); x2old =
X(i,2);
                    x1lead = X(idx_lead,1);
x2lead = X(idx_lead,2);
                    x1new = x1old +
(2*rand-1) * stepc * (abs(x1lead-x1old));
                    x2new = x2old +
(2*rand-1) * stepc * (abs(x2lead-x2old));
                    if easom(x1new,
x2new)<easom(x1old, x2old)
                        X(i,1) = x1new;
X(i,2) = x2new;
```

```
                        m = X(:,1); n =
X(:,2);
                        refreshdata
                        drawnow
                        if easom(x1new,
x2new)<lead
                            lead =
easom(x1new, x2new);
                            idx_lead = i;
                        end
                    end
                end
            end
            step=5;
        elseif step==5
            % stronger surviving renewing
            fitness = easom(X(:,1),X(:,2));
            [~, idx_sorted] = sort(fitness,
"descend");
            R = randi([Rmin, Rmax], 1);
            for i=1:R
                    X(idx_sorted(i),1) =
X(idx_lead, 1) * (1+(0.2*rand-0.1));
                    X(idx_sorted(i),2) =
X(idx_lead, 2) * (1+(0.2*rand-0.1));
                    m = X(:,1); n = X(:,2);
                    refreshdata
                    drawnow
            end
            overfl=1;
        end
    end

    k = k + 1;
    fprintf("iteration: %d\n", k)
end

%%% Plotting fitness over time
figure;
plot(best);
hold on
plot(worst); plot(avg);
hold off
ylim([-1, 0])
title("Best, worst and average fitness");
xlabel("iterations","FontWeight", "bold");
ylabel("Fitness", "FontWeight", "bold");
legend("Best", "Worst", "Average",
"Location", "best");
```

4) The plot function used in above MATLAB codes

```
function plotfn(fn, xlim, ylim)
    % plotting function
    figure;
    [ptsx, ptsy] =
meshgrid(linspace(xlim(1),xlim(2),800),
linspace(ylim(1),ylim(2),800));
    fn_vals = fn(ptsx,ptsy);
    [min_val] = min(fn_vals,[], "all");
    [max_val] = max(fn_vals,[], "all");
    mesh(ptsx ,ptsy, fn_vals);
    colormap default; colorbar
    hold on
    plot3(ptsx(fn_vals ==
min_val),ptsy(fn_vals == min_val),
fn_vals(fn_vals == min_val), "xr");
    plot3(ptsx(fn_vals ==
max_val),ptsy(fn_vals == max_val),
fn_vals(fn_vals == max_val), "xg");
    title("f(x)");
    xlabel("x","FontWeight", "bold");
ylabel("y", "FontWeight", "bold");
zlabel("f(x)", "FontWeight", "bold");
    legend("function", "Min value", "Max
value", "Location", "best");
end
```