POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Bio-Inspired Modifications of the PSO Algorithm

Relatori
prof. Marco Scianna

Candidato
Melissa Cannas

Anno Accademico 2022-2023

Contents

Introduction

Most real-world problems are not deterministic in nature and, therefore, require stochastic techniques to find solutions. To achieve this, it is convenient to rely on stochastic optimization algorithms. Although they are efficient in finding solutions, they can lead to significant computational efforts and may fail as the complexity of the problem increases. To address these issues, bio-inspired stochastic algorithms and population-based techniques have been developed and gained importance due to the improvement in computational efficiency.
Particle Swarm Optimization (PSO) is an example of such algorithms, as it is indeed a population-based stochastic optimization algorithm that exploits the concepts of social behavior observed in animals like insects, herds, birds, and fish, in the search of the best possibile solution/s to a given problem. The algorithm was first introduced by James Kennedy and Russell Eberhart in their 1995 paper titled "Particle Swarm Optimization" (Kennedy and Eberhart [1995]).
The basic idea behind PSO is to simulate the cooperative behavior observed in nature, where animals in a swarm share information about their local surroundings and collectively navigate towards better conditions.
In PSO, a population of individuals, referred to as "particles", move through the search space, adjusting their positions and velocities based on both their individual experiences and the experiences of the swarm as a whole. This combination of individual exploration and swarm cooperation helps guide the particles towards optimal solutions. This algorithm has several advantages: it is easy to describe and implement, requires a relatively small number of function evaluations to converge, and boasts a fast rate of convergence. It has undergone numerous variations and improvements, including modifications to the update equations, incorporation of constraints, and hybridization with other optimization techniques.
In this thesis, we will introduce bio-inspired modifications to the PSO algorithms, following the considerations given in Section 1.3. While Particle Swarm Optimization is highly effective and suitable for modeling swarms of animals, from a biological perspective, this algorithm needs to undergo slight changes, that will be presented and described in detail in Chapter 2 .
The rest of the thesis is organized as follows. The formalization of a generic PSO will be presented in Chapter 1, including a description of the algorithm, and comments on the component ingredients and parameters.
As previously introduced, Chapter 2 will be dedicated to presenting our proposed method. In Chapter 3 we will describe our two objective functions and numerical settings used in our simulations. Moreover, we will explore how different values of the model coefficients affect simulation outcomes, specifically the method's ability to reach convergence.
Finally, in Chapter 4, with the introduction of the concept of "neighbourhood", we will present a variation of our proposed method, with slight changes in some terms of the algorithm.

Chapter 1

The Particle Swarm Optimization Model

1.1 Definition of the problem

Let us introduce a given high dimensional "objective" function of d variables:

$$F(\mathbf{x}) : \mathbf{X} \subseteq \mathbb{R}^d \to \mathbb{R}. \tag{1.1}$$

An optimization problem consists in finding $\mathbf{x}^* \in \text{argmin}_{\mathbf{x}\in\mathbf{X}} F(\mathbf{x})$, i.e, in finding the points of the domain $\mathbf{X}$ where F attains the minimum value.
In this respect, the domain $\mathbf{X}$ of the function is often called *acceptable region*, or search space, while each point $\mathbf{x} = \{x_1, x_2, ..., x_d\}^T \in \mathbf{X}$ is typically referred to as *admissible* or *candidate solution*.

The Particle Swarm Optimization algorithms solve the above minimization problem by employing a population of simple entities, called particles, which move in the search space $\mathbf{X}$ according to a specified set of behavioural rules. Their positions indeed represent candidate solutions of the problem.

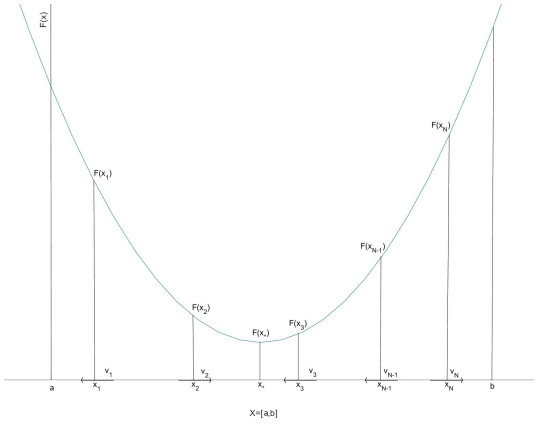

Figure 1.1. Representative system setting.
The objective function $F(x, y) = (x - 3.14)^2 + (y - 2.72)^2 + \sin(3x + 1.41) + \sin(4y - 1.73)$ is defined on the one-dimensional domain $X = [a, b]$.
A population of N particles, with position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$ is then allowed to move along X in order to find out $\mathbf{x}^* = \text{argmin}_{\mathbf{x}\in\mathbf{X}} F(\mathbf{x}) \in X$.

1.2 Original PSO

As previously seen, Particle Swarm Optimization algorithms typically take into account a set of N particles: each generic agent i is represented by a material point with unitary mass, with actual position $\mathbf{x}_i(t) \in \mathbf{X} \subseteq \mathbb{R}^d$ and velocity $\mathbf{v}_i(t) \in \mathbb{R}^d$, being $\mathbf{X}$ the domain of the objective function F defined in (1.1), and $t \in T = [0, t_f]$ is indeed the time variable, with t_f the final observation time.
In the perspective of numerical implementation, we hereafter refer to the discretized version of the time domain T.

In the original PSO algorithm, starting with initially assigned values

$$\begin{cases} \mathbf{x}_i(0) = \mathbf{x}_i^0; \\ \mathbf{v}_i(0) = \mathbf{v}_i^0, \end{cases} \tag{1.2}$$

for any agent $i = 1, .., N$, the system is updated as follows:

$$\begin{cases} \mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \\ \mathbf{v}_i(t+1) = \underbrace{w\mathbf{v}_i(t)}_{\textit{inertia}} + \underbrace{c_1\mathbf{R}_1^i(t) \cdot (\mathbf{p}_i(t) - \mathbf{x}_i(t))}_{\textit{"individual knowledge" awareness}} + \underbrace{c_2\mathbf{R}_2^i(t) \cdot (\mathbf{g}(t) - \mathbf{x}_i(t))}_{\textit{"social interactions" transmission of information}}. \end{cases} \tag{1.3}$$

Regarding the last equation:

- $\mathbf{p}_i(t)$ identifies the best position, in the perspective of minimization of F, found by the i-th particle up to t-th iteration.
  It is initialized as
  $$\mathbf{p}_i(0) = \mathbf{x}_i^0,$$
  and updated with rule:
  $$\mathbf{p}_i(t+1) = \begin{cases} \mathbf{p}_i(t), & \text{if} \quad F(\mathbf{x}_i(t+1)) \geq F(\mathbf{p}_i(t)); \\ \mathbf{x}_i(t+1), & \text{if} \quad F(\mathbf{x}_i(t+1)) < F(\mathbf{p}_i(t)). \end{cases} \tag{1.4}$$
  The vector $\mathbf{p}_i(t)$ is usually referred to as the *local best position*.

- $\mathbf{g}(t)$ is instead the best position found among all the particles up to iteration t.
  In this respect, we have that:
  $$\begin{cases} \mathbf{g}(0) = \text{argmin}_{i=1,..,N}\{F(\mathbf{x}_i(0) = \mathbf{x}_i^0)\}; \\ \mathbf{g}(t+1) = \text{argmin}_{i=1,..,N}\{F(\mathbf{p}_i(t+1))\}. \end{cases} \tag{1.5}$$
  The vector $\mathbf{g}(t)$ is also called *global best position*.

- The terms $\mathbf{R_j^i}(t)$, with j = 1,2, indicate two d-dimensional diagonal matrices. In particular, they contain random numbers uniformly distributed in the interval [0,1], and that are generated, for any iteration, for each agent, i.e.,
  $$(\mathbf{R}_1^i)_{jj}, (\mathbf{R}_2^i)_{jj} \in U([0,1]), \forall j = 1, .., d, t \in T, i = 1, ..., N.$$

Such random contribution may be also given as vectors, i.e. $\mathbf{r}_j^i(t)$, with j = 1,2, for any time $t \in T$ and agent $i = 1, ..., N$.
In this case, we have to introduce in (1.3) the Kronecker product, denoted with $\otimes$:

$$c_1\mathbf{r}_1^i(t) \otimes (\mathbf{p}_i(t) - \mathbf{x}_i(t))$$

and

$$c_2\mathbf{r}_2^i(t) \otimes (\mathbf{g}(t) - \mathbf{x}_i(t)),$$

being

$$\mathbf{a} \otimes \mathbf{b} = (a_1b_1, ..., a_nb_n),$$

with $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$.

- w, c_1, $c_2 \in \mathbb{R}^+$ are finally a sort of acceleration coefficients.

1.3 Observations and critical points

In the perspective of an application of PSO algorithm to biological scenarios, it is necessary to make some considerations:

- Random movement is self-generating, i.e. it is independent from other behavioural stimuli and for this reason we have to consider a separate term in the velocity to represent them.

- A cell can only interact directly with those in the surrounding, i.e. the global best position has to be specified for each particle:

$$\mathbf{g}(t) \to \mathbf{g}_i(t).$$

- In the original PSO all behavioural traits can be simultaneously maximized or minimized and this is not possible considering a biological point of view. Furthermore the agent speed is not differentiated from the movement direction and it is necessary to take into account this difference in a biology perspective.

- Cells move in a highly viscous environment, i.e. characterized by low Reynolds' number. In this respect, inertia can be considered negligible.

Chapter 2

Proposed method

Basing on the considerations of the previous section, let us now propose our version of the PSO method.
In particular, in perspective of numerical implementation, we here after consider one-dimensional settings.
In this respect, the objective function is defined on a one-dimensional domain, i.e.,

$$F(\mathbf{x}) : \mathbf{X} \subseteq \mathbb{R} \to \mathbb{R}. \tag{2.1}$$

We define the following notation for unit vectors,

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{|\mathbf{a}|},$$

with $\mathbf{a} \in \mathbb{R}$, and then introduce a "new" actual velocity for the probing particles:

$$\mathbf{v}_i(t+1) = v_i \hat{\mathbf{w}}_i(t+1), \tag{2.2}$$

where $v_i \in \mathbb{R}_+$ is an individual speed/motility, that may account for physiological limitation, while

$$\mathbf{w}_i(t+1) = \alpha(\mathbf{p}_i(\widehat{t) - \mathbf{x}_i}(t)) + \beta(\mathbf{g}(\widehat{t) - \mathbf{x}_i}(t)) + \gamma \hat{\mathbf{r}}_i(t), \tag{2.3}$$

for any $t \in T$ and $i = 1, ..., N$. Agent position is instead updated exactly as in (1.3).

In the above equation (2.3):

- $\mathbf{p}_i(t)$ and $\mathbf{g}(t)$ are respectively the individual best position and the global best position, as previously defined in section (1.2).

- $\mathbf{r}_i(t) \in \mathbb{R}$ implements cell Brownian crawling. In this respect, a wide range of sophisticated or application-related laws may be employed. However, for the sake of simplicity, we opt to set $\mathbf{r}_i(t)$ as a random variable that takes the values -1 or +1 with probability $\frac{1}{2}$.

- The acceleration coefficients α, β, $\gamma \in \mathbb{R}^+$ are then set to be subjected to the following constraint:

  $$\alpha + \beta + \gamma = 1. \tag{2.4}$$

  In this respect, they can be interpreted as weights that define the relative importance of each migratory contribution in (2.3), i.e. in affecting cell probing activity.

Some comments on the proposed version of the algorithm:

- it consistently decouples cell speed (v_i) and direction of movement ($\mathbf{w}_i$). The former quantity in fact is essentially determined by intracellular pathways involving molecules such as Roc, Rho,... that affect membrane ruffles and fluctuations. The latter is instead established by the polarization of the cell cytoskeleton , which is influenced by internal and external signals, able to activate action-filament rearrangements.

- The first term at the right hand side of (2.3) may be defined of mesenchymal nature, as it implements a single-cell mode of migration, i.e. independent from the presence of other individuals. Coherently, the second contribution has an epithelial nature, as it depends on intercellular communication.
  In this respect, the assumption that $\mathbf{g}$ is in common for all agents has the underlying implication that each cell is able to exchange information with any other group mate, regardless its distance.
  From a biological perspective, we are indeed assuming the possibility of a long-range cell-cell trasmission of signals, that may rely upon release and absorption of selected chemical factors (and not only upon the activity of cadherins that are instead involved in short-range cell-cell contact interactions).

Chapter 3

Numerical settings

In the forthcoming sections, we will analyze the ability of the proposed method to solve minimization problems upon variations in selected model components and parameters.
In particular, numerical tests will involve the following two objective functions, both evaluated in the closed domain $\mathbf{X} = [-30,30]$:

$$F_1(x) = x^2, \tag{3.1}$$

$$F_2(x) = (2 - cos(x))(x - 3)^2. \tag{3.2}$$

As reproduced in Figure 3.1, the former function is a parabola with the global minimum in the vertex, i.e., in $x = 0$. The latter is instead a function characterized by a global minimum in $x = 3$ and several local minima.

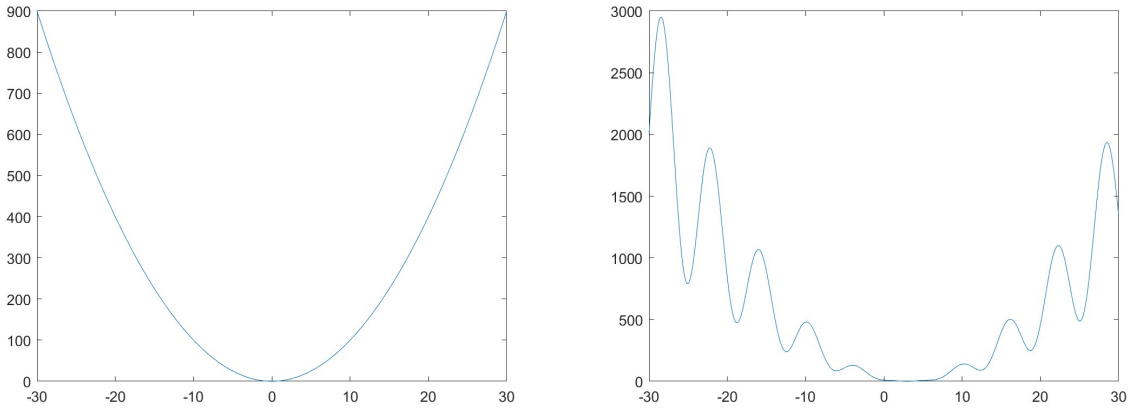

Figure 3.1. Graphical representation of the objective functions defined in (3.1) and (3.2), both evaluated in the domain $\mathbf{X} = [-30,30]$.

In particular, for any numerical setting, the output of the algorithm will be classified according to the mean value of the following quantities, calculated over 7 independent realizations:

$$d_{min} = |\mathbf{g}(t_f) - \underset{x \in X}{\operatorname{argmin}} F(x)|, \tag{3.3}$$

$$d_{max} = \max_{i=1,\dots,N} |\mathbf{p}_i(t_f) - \underset{x \in X}{\operatorname{argmin}} F(x)|. \tag{3.4}$$

In this respect, we will distinguish four different scenarios:

- S_1, when d_{min} is larger than 0, regardless the value of d_{max}. It is the worst situation since it implies that no agent is able to find the target point.
- S_2, when $d_{min} = 0$ but $d_{max} > 0$: in this case, only a subset of particles is able to find the point of interest.
- S_3, when $d_{min} = d_{max} \neq 0$, i.e., all particles get stuck in one or more local minima.
- S_4, when $d_{min} = d_{max} = 0$ finally implies that all particles converge to the desired point. From an algorithmic point of view, it is the best situation, as the minimization problem is solved by the entire population of individuals.

3.1 Parameters

For any forthcoming simulation setting, the cell population size N will be constantly set equal to 50.
For any cell $i = 1, ..., N$, we will set $v_i(0) = 0$ and randomly established the initial position, i.e., to avoid biases deriving from the specific initial configuration.
As a boundary condition we employ the *Absorbing Walls* (Robinson and Rahmat-Samii [2004]): when a cell hits the border of the domain, its velocity is set to zero.
Finally, for the sake of simplicity, the individual speed v_i will be set equal to 1 and the final observation time t_f will be constantly fixed to 5000 iterations.

3.2 Varying model coefficients

In this section, we will analyze how variations in the coefficients α, β and γ will affect the simulation outcomes, in terms of ability of the method to eventually solve the problem.

For this purpose, we will illustrate our results using two graphs relative to the two objective functions, and four different colors, one for each scenario described before.
In particular, the green points stand for S_1, the light-blue points represent S_2, the red ones are used for S_3, and finally the orange ones stand for S_4.

***Function* $\mathbf{F_1}$**

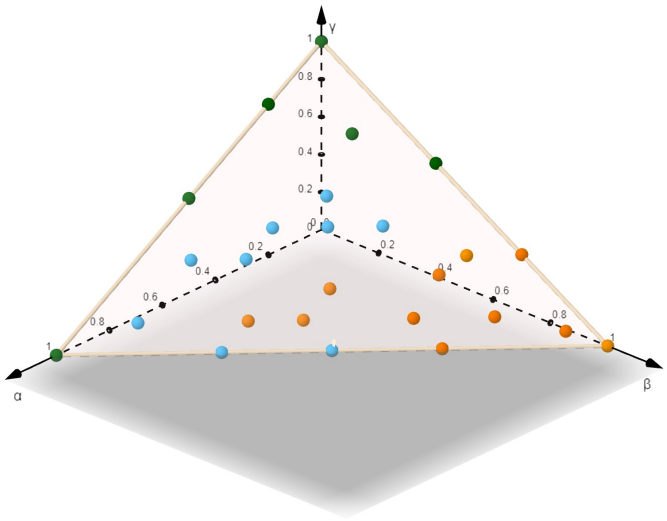

Figure 3.2. Behavior of the proposed method in case of the objective function F_1 upon variations in the parameters α, β and γ.

First of all, we can obviously observe that the third scenario S_3 does not exist for the parabola because there is only the global minimum, which is the vertex.
When γ assumes high values (i.e., $\in [0.5, 1]$), regardless the value of α and β, no particle reaches the minimum (Scenario S_1).
Conversely, if γ assumes low values (i.e., $\in [0, 0.4]$), three different situations may emerge, depending on the value of α.
For $\alpha \in [0.9, 1]$, and therefore low values of β, no particle is capable of reaching the minimum, thus leading to scenario S_1. Instead, when α assumes intermediate values (i.e, $\in [0.3, 0.8]$), we have that a subset of particles successfully reaches the minimum (Scenario S_2).
Lastly, if $\alpha \in [0, 0.2]$, and therefore β assumes sufficiently high values, all particles find the global minimum (scenario S_4).

For the sake of completeness, in Fig. 3.3, 3.4 and 3.5, we show a representative time-lapse sequence of particle dynamics for each of the above discussed scenarios.

Scenario $\mathbf{S_1}$

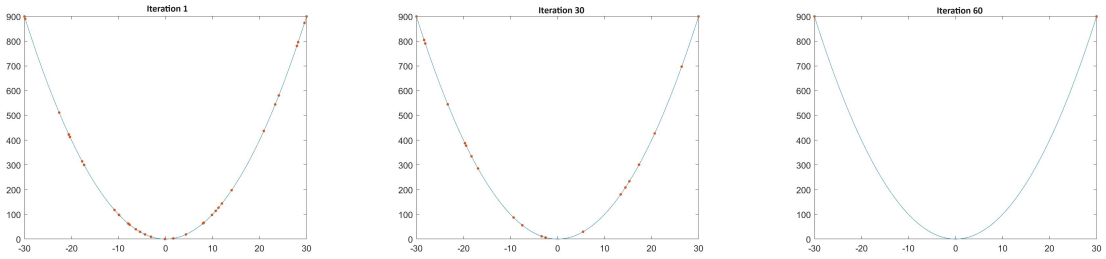

Figure 3.3. Representation of particle dynamics on function F_1, at iteration 1, 30 and 60, with chosen coefficients $\alpha = 0.1$, $\beta = 0.3$ and $\gamma = 0.6$.

Scenario $\mathbf{S_2}$

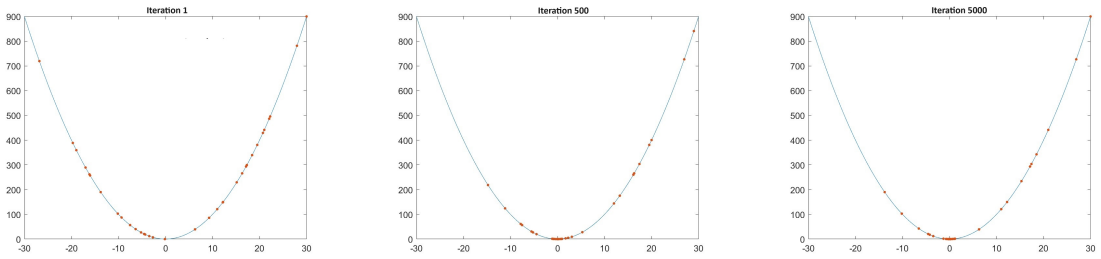

Figure 3.4. Representation of particle dynamics on function F_1, at iteration 1, 500 and 5000, with chosen coefficients $\alpha = 0.5$, $\beta = 0.2$ and $\gamma = 0.3$.

Scenario $\mathbf{S_4}$

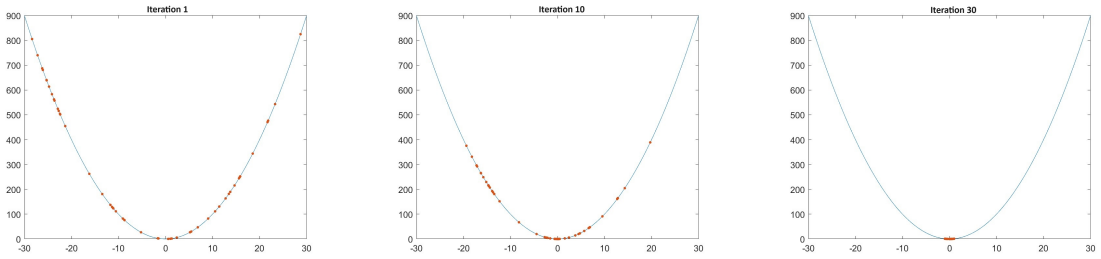

Figure 3.5. Representation of particle dynamics on function F_1, at iteration 1, 10 and 30, with chosen coefficients $\alpha = 0.1$, $\beta = 0.8$ and $\gamma = 0.1$.

Function $\mathbf{F_2}$

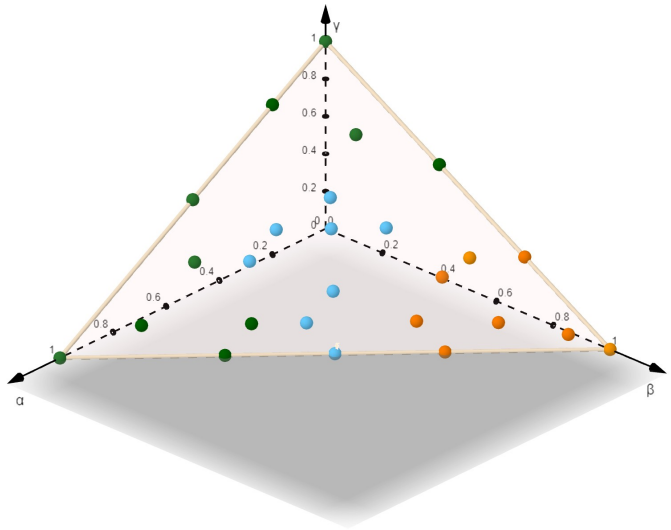

Figure 3.6. Behavior of the proposed method in case of the objective function F_2 upon variations in the parameters α, β and γ.

As in function F_1, when $\gamma \geq 0.5$, no particle is able to find the minimum (scenario S_1). When we decrease the value of γ, specifically when $\gamma \in [0, 0.4]$, we have three different scenarios depending on α, as before.
When $\alpha \in [0.6, 1]$, and consequently β assumes low values, we are in the worst scenario, S_1. Conversely, if α falls in an intermediate range (i.e., $\in [0.3, 0.5]$), we are in scenario S_2.
Finally, when $\alpha \in [0, 0.2]$, and therefore β assumes high values, the entire population of particles reaches the global minimum (scenario S_4).
It is interest to notice that we do not observe scenario S_3 for any tested parameter setting.

As for function F_1, in Fig. 3.7, 3.8 and 3.9, we show a representative time-lapse sequence of particle dynamics for each of the above discussed scenarios.

Scenario $\mathbf{S_1}$

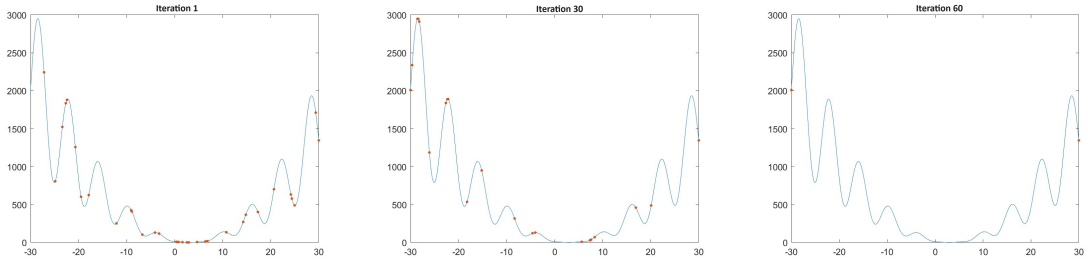

Figure 3.7. Representation of particle dynamics on function F_2, at iteration 1, 30 and 60, with chosen coefficients $\alpha = 0.1$, $\beta = 0.3$ and $\gamma = 0.6$.

Scenario $\mathbf{S_2}$

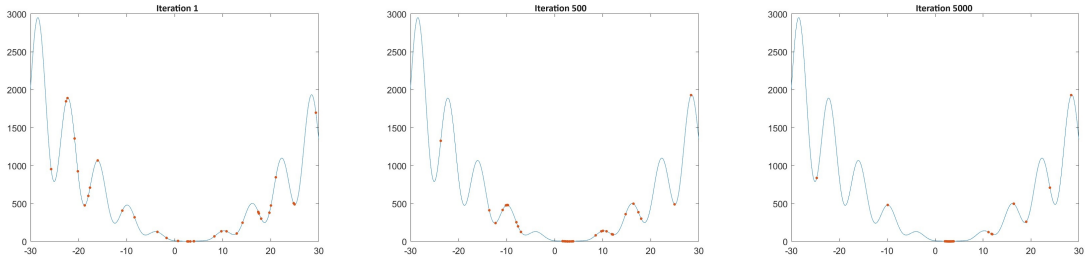

Figure 3.8. Representation of particle dynamics on function F_2, at iteration 1, 500 and 5000, with chosen coefficients $\alpha = 0.5$, $\beta = 0.3$ and $\gamma = 0.2$.

Scenario $\mathbf{S_4}$

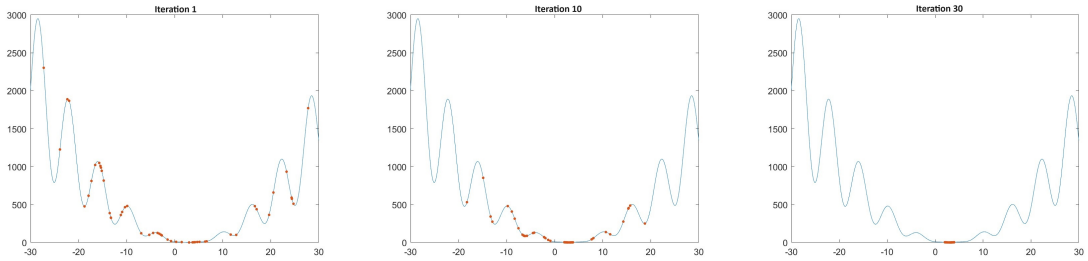

Figure 3.9. Representation of particle dynamics on function F_2, at iteration 1, 10 and 30, with chosen coefficients $\alpha = 0.1$, $\beta = 0.8$ and $\gamma = 0.1$.

Comparison

By comparing the above results, we can make the following comments:

- A substantially high relevance of randomness (i.e., $\gamma \geq 0.5$) disrupt the possibility of the agent population to reach the desired point, i.e. regardless the value of α and β.

- For sufficiently low values of γ, the algorithm behaviour relies on the ratio between α and β. In this respect, the ability of the agent population to converge to the solution of the problem emerges only for high values of β, i.e. for high relevance of the social component in individual exploratory behaviour. Furthermore, slight differences emerge between the two tested objective functions: in the case of F_1, the range of values of β leading to scenario S_4 is larger than that observed in the case of F_2. This is due to the smoothness of function F_1 compared to F_2, as F_1 lacks local minima: this fact makes it easier for particles to reach the global minimum relying solely on their individual knowledge.
  It is clear that the two functions yield qualitatively equivalent results, but it's evident that as the complexity of the functions increases, a higher value of β is required to achieve convergence.
  In conclusion, β is indeed the key coefficient to reach the solution.

In this respect, we analyze the number of iterations needed to achieve convergence in selected numerical realizations.

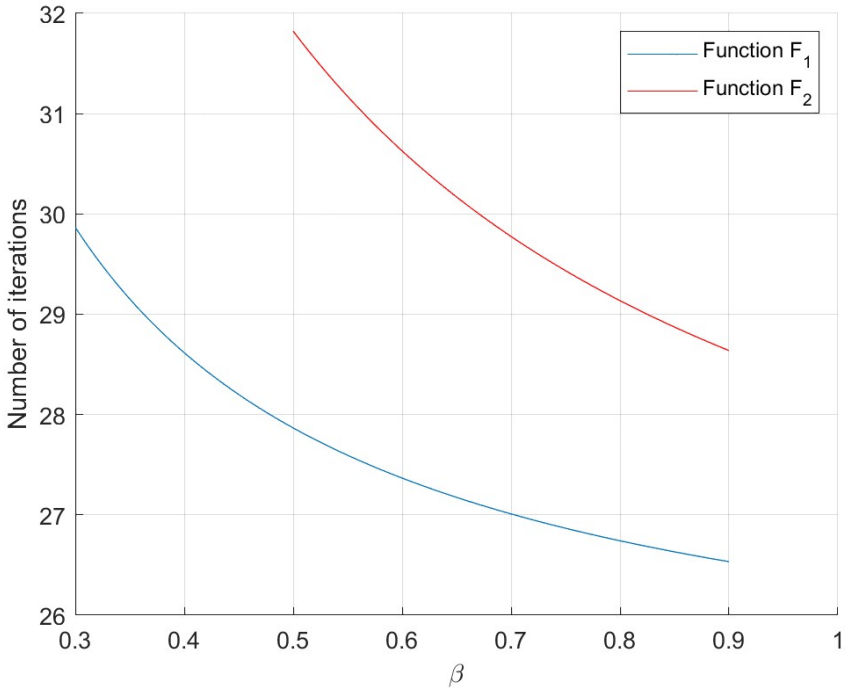

Figure 3.10. Fixing $\gamma = 0.1$, we show the number of iterations needed to reach convergence, varying the value of β.

As shown in Figure 3.10, it becomes evident that as β increases, the number of iterations required to achieve convergence decreases for both functions F_1 and F_2, i.e., the more efficient is the information transmission across the population of agents, the quicker is the

algorithmic convergence to the solution.
From the same graph, we can observe a saturation effect, as the iterations required to converge to the solution decrease up to a certain point and then stabilize.
It is finally useful to observe that for a given parameter setting, the amount of iterations needed for convergence is higher for F_2 compared to F_1, and this is determined by the different complexity of the two functions.

Chapter 4

Variations to the proposed method

The model proposed in Chapter 2 implies that all agents interact with the entire population, regardless their position. However, it is important to note that a cell can only interact directly with its immediate neighborhood.

To model this aspect, we have first to define a "global" best position that is different for each individual, i.e.,

$$\mathbf{g}(t) \to \mathbf{g}_i(t),\ \forall i = 1, ..., N,\ t \in T.$$

We then introduce the Euclidean distance function:

$$|\cdot| : \mathbf{X} \times \mathbf{X} \to \mathbb{R}_+ \cup \{0\}, \tag{4.1}$$

where $\mathbf{X} \subseteq \mathbb{R}$ is the usual one-dimensional domain.

We can define the neighbourhood of the generic particle i as:

$$N_i(t) = \{j = 1, ..., N, j \neq i : |\mathbf{x}_i(t) - \mathbf{x}_j(t)| < n, n \in \mathbb{R}_+\}, \tag{4.2}$$

being n the interaction radius.

In this respect, for any agent i, its global best position is:

$$\mathbf{g}_i(t) = \underset{j \in N_i(t) \cup i}{\operatorname{argmin}} \{F(\mathbf{x}_j(t))\}, \tag{4.3}$$

which is the global best position within the neighbors.

We recall the actual velocity for the probing particles:

$$\mathbf{v}_i(t+1) = v_i \hat{\mathbf{w}}_i(t+1), \tag{4.4}$$

where $v_i \in \mathbb{R}_+$ is an individual speed/motility.

Regarding $\mathbf{w}_i(t+1)$, we substitute the global best position $\mathbf{g}(t)$ with $\mathbf{g}_i(t)$:

$$\mathbf{w}_i(t+1) = \alpha(\mathbf{p}_i(\widehat{t) - \mathbf{x}_i}(t)) + \beta(\mathbf{g}_i(\widehat{t) - \mathbf{x}_i}(t)) + \gamma \hat{\mathbf{r}}_i(t). \tag{4.5}$$

In the above equation (4.5), the agent position $\mathbf{x}_i(t)$, the individual best position $\mathbf{p}_i(t)$ and the random component $\mathbf{r}_i(t)$ are updated exactly as in Chapter 2.

4.1 Numerical setting

For any forthcoming simulation setting, we will employ the test objective functions 3.1 and 3.2, evaluated in the usual closed domain $\mathbf{X} = [-30,30]$, and analyze the algorithm behaviour upon variations in the extension of the interaction neighbourhood.
In this respect, the numerical outputs will be classified according to the above-introduced scenarios, i.e. based on the mean, calculated over 7 independent realizations, of d_{min}(3.3) and d_{max} (3.4).

The cell population size N will be constantly set equal to 50, and for any cell $i = 1, ..., N$, we will set $v_i(0) = 0$ and the initial position is established according to the uniform distribution. For the sake of simplicity, the individual speed v_i will be set equal to 1 and the final observation time t_f will be constantly fixed to 5000 iterations.

In all simulations, we fix the following triplet of parameters, that have been observed to lead to the best possible scenario for both the tested functions, (3.1) and (3.2):

$$\begin{cases} \alpha & = 0.1, \\ \beta & = 0.8, \\ \gamma & = 0.1. \end{cases}$$

4.2 Results

Varying the distance parameter $n = 1, ...,30$, for every agent $i = 1, ..., N$, we analyze the relative scenario obtained.

We have the following two tables, one for each objective function.

n	1	3	5	10	20	30
Scenario	S_1	S_2	S_2	S_4	S_4	S_4

Table 4.1. Function F_1.

n	1	3	5	10	20	30
Scenario	S_1	S_2	S_2	S_2	S_2	S_4

Table 4.2. Function F_2.

Looking at the tables, we can make some considerations:

- For low values of n (i.e., $n < 3$), no particle is able to reach the optimal solution of the problem, in the case of both tested function F_1 and F_2.

- For intermediate values of n, a subset of particles is able to reach the minimum point of the objective functions.

- For high values of n, the entire system of particles converge to the solution of the problem for both F_1 and F_2. In particular, the value of n leading to convergence to the optimal solution is $n = 10$ in the case of F_1, and $n = 30$ in the case of F_2.

We can indeed comment that the more the tested function is complex, the more important is the information transmission across the population of agents. These numerical outcomes are consistent with those obtained in the previous sections, by varying the value of β

Chapter 5

Conclusions

In this thesis, we have modified the original Particle Swarm Optimization algorithm to obtain a more suitable version in a biological perspective.
The two new methods introduced aim to overcome some issues relative to the update of the velocity in the case of cells, which moves in a viscous environment, have a speed that is differentiated from the direction of movement, have a random movement that is self-generating and have a limited possibility to exchange information within all the group.
We have obtained quite satisfying and coherent results, in both methods we have implemented.

For both methods, all the numerical simulations were run using two objective functions in a one-dimensional setting, but it will be very interesting to increase the dimension of the domain and studied the problem in two or three dimensions.
In this respect, when particles convergence to the solution in a one-dimensional domain, they overlap in the minimum point, and this is not realistic. This issues could be overcome using a domain with higher dimension.

In the first proposed method, the implementation of the velocity worked quite well, and the results obtained underline how important is the social component to reach the solution, and this is coherent with the biological phenomena we have in nature.
In the second proposed method, we have introduced the concept of neighbourhood to take into account the communication possibility of the cells, which have not an infinite radius of interactions. These changes has produced coherent solution, but there is a problem from a biological point of view: in nature, cells communicate with each other using the surrounding space. When we have defined the neighborhood, we used the Euclidean norm and we did not introduce a unit of measure: this is important from a biological point of view.

Looking at the choice for the model coefficients, the balance between α, β and γ was fundamental to reach the solution and, although the random component (and the relative model coefficient γ) seems to be a problem if assumes very high values, the choose of γ is extremely important when we work with complex functions that present one or more local minima.
Looking at the choice of the random variable, in our model we choose a uniform distribution, where the random variables assume the values -1 and 1 with probability $\frac{1}{2}$.
In a perspective of future work, this distribution could be modified, for example taking into consideration the Levy's distribution.

Appendix A

Particle Swarm Algorithm

```
close all
clear all
clc

CostFunction = @(x) ((2 - cos(x)).*(x-3).^2);
%(x.^2);
OptimumPos = 3;
nVar = 1;             % Number of Decision Variables

VarSize = [1 nVar];   % Size of Decision Variables Matrix

 VarMin = -30;          % Lower Bound of Variables
 VarMax = 30;          % Upper Bound of Variables
%
 x=linspace(-30,30,1000);
 %y=x.^2;
 y=(2 - cos(x)).*(x-3).^2;

 MaxIt = 5000;       % Maximum Number of Iterations

nPop = 50;         % Population Size (Swarm Size)

% PSO Parameters
%% Run 7 different simulations with fixed acceleration coefficients (j
    =1:7)
n=7;
output_min=zeros(n,1);
output_max=zeros(n,1);


alpha = 0.1;     % Cognitive Coefficient
beta = 0.8;        % Social Coefficient
gamma =0.1;        % Randomization Coefficient
% Velocity Limits
VelMax = 0.1*(VarMax-VarMin);
VelMin = -VelMax;

%% Initialization

empty_particle.Position = [];
empty_particle.Cost = [];
empty_particle.Velocity = [];
```

```
empty_particle.Best.Position = [];
empty_particle.Best.Cost = [];

particle = repmat(empty_particle, nPop, 1);

GlobalBest.Cost = inf;

for i = 1:nPop

    % Initialize Position
    particle(i).Position = unifrnd(VarMin, VarMax, VarSize);

    % Initialize Velocity
    particle(i).Velocity = zeros(VarSize);

    % Evaluation
    particle(i).Cost = CostFunction(particle(i).Position);

    % Update Personal Best
    particle(i).Best.Position = particle(i).Position;
    particle(i).Best.Cost = particle(i).Cost;

    % Update Global Best
    if particle(i).Best.Cost<GlobalBest.Cost

        GlobalBest = particle(i).Best;

    end

end

BestCost = zeros(MaxIt, 1);
maxdist = zeros(nPop,n);
v_loc=zeros([nPop,VarSize]);
v_glob=zeros([nPop,VarSize]);
random=rand([nPop,VarSize])*2-1;
for j=1:n
for it = 1:MaxIt
    for i = 1:nPop

        % Update Velocity
       if particle(i).Best.Position==particle(i).Position
           v_loc(i,:)=0;
       else
            v_loc(i,:)=(particle(i).Best.Position-particle(i).Position)./
    norm(particle(i).Best.Position-particle(i).Position,2);
       end
       if GlobalBest.Position==particle(i).Position
           v_glob(i,:)=0;
       else
           v_glob(i,:)=(GlobalBest.Position-particle(i).Position)./norm(
    GlobalBest.Position-particle(i).Position,2);
       end

       particle(i).Velocity = (alpha.*v_loc(i,:)+ beta.*v_glob(i,:)+gamma
    .*(random(i,:)./norm(random(i,:),2)))...
            ./norm(alpha.*v_loc(i,:)+ beta.*v_glob(i,:)+gamma.*(random(i
    ,:)./norm(random(i,:),2)),2);
       % Apply Velocity Limits
        particle(i).Velocity = max(particle(i).Velocity, VelMin);
        particle(i).Velocity = min(particle(i).Velocity, VelMax);
```

```
99
100         % Update Position
101         particle(i).Position = particle(i).Position + particle(i).
    Velocity;
102
103         % Velocity Mirror Effect
104         IsOutside = (particle(i).Position<VarMin | particle(i).Position>
    VarMax);
105         particle(i).Velocity(IsOutside) = -particle(i).Velocity(IsOutside
    );
106
107         % Apply Position Limits
108         particle(i).Position = max(particle(i).Position, VarMin);
109         particle(i).Position = min(particle(i).Position, VarMax);
110
111         % Evaluation
112         particle(i).Cost = CostFunction(particle(i).Position);
113
114         % Update Personal Best
115         if particle(i).Cost<particle(i).Best.Cost
116
117             particle(i).Best.Position = particle(i).Position;
118             particle(i).Best.Cost = particle(i).Cost;
119
120             % Update Global Best
121             if particle(i).Best.Cost<GlobalBest.Cost
122
123                 GlobalBest = particle(i).Best;
124
125             end
126
127         end
128         xn(i)=particle(i).Position(1);
129         yn(i)=(2 - cos(xn(i))).*(xn(i)-3).^2;
130        %yn(i)=(xn(i)).^2;
131
132         figure(2);
133         plot(x,y)
134         hold on;
135         plot(xn,yn,'.','markersize',10,'markerfacecolor','g');
136         drawnow;
137         hold off
138 if it==MaxIt
139     maxdist(i,j)=norm(particle(i).Best.Position-OptimumPos,1);
140 end
141     end
142
143     BestCost(it) = GlobalBest.Cost;
144
145     disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))
    ]);
146
147 end
148 output_min(j)=norm(GlobalBest.Position-OptimumPos,2);
149 output_max(j)=max(maxdist(:,j));
150 end
151 d_min=mean(output_min);
152 d_max=mean(output_max);
153 BestSol = GlobalBest;
154
155 %% Results
```

```
156
157 figure;
158 %plot(BestCost, 'LineWidth', 2);
159 semilogy(BestCost, 'LineWidth', 2);
160 xlabel('Iteration');
161 ylabel('Best Cost');
162 grid on;
```

Appendix B

Particle Swarm Algorithm with Neighbourhood

```
    close all
clear all
clc

CostFunction = @(x)  ((2 - cos(x)).*(x-3).^2);

%(x.^2);


OptimumPos = 3;
nVar = 1;            % Number of Decision Variables

VarSize = [1 nVar];   % Size of Decision Variables Matrix

VarMin = -30;         % Lower Bound of Variables
VarMax = 30;         % Upper Bound of Variables
%
x=linspace(-30,30,1000);
%y=x.^2;
y=(2 - cos(x)).*(x-3).^2;

MaxIt = 5000;      % Maximum Number of Iterations

nPop =50;        % Population Size (Swarm Size)

% PSO Parameters
%% Run 7 different simulations with fixed acceleration coefficients (j
    =1:7)
n=1;
output_min=zeros(n,1);
output_max=zeros(n,1);


alpha = 0.1;     % Cognitive Coefficient
beta = 0.8;        % Social Coefficient
gamma =0.1;        % Randomization Coefficient
% Velocity Limits
VelMax = 0.1*(VarMax-VarMin);
VelMin = -VelMax;
```

```
39
40 %% Initialization
41
42 empty_particle.Position = [];
43 empty_particle.Cost = [];
44 empty_particle.Velocity = [];
45 empty_particle.Best.Position = [];
46 empty_particle.Best.Cost = [];
47
48 particle = repmat(empty_particle, nPop, 1);
49
50
51 for i = 1:nPop
52
53     GlobalBest.Cost(i) = inf;
54
55     % Initialize Position
56     particle(i).Position = unifrnd(VarMin, VarMax, VarSize);
57
58     % Initialize Velocity
59     particle(i).Velocity = zeros(VarSize);
60
61     % Evaluation
62     particle(i).Cost = CostFunction(particle(i).Position);
63
64     % Update Personal Best
65     particle(i).Best.Position = particle(i).Position;
66     particle(i).Best.Cost = particle(i).Cost;
67
68     %Initialize best neighbourhood
69     GlobalBest.neigh=[];
70
71     % Update Global Best
72     if particle(i).Best.Cost<GlobalBest.Cost(i)
73
74         GlobalBest.Cost(i) = particle(i).Best.Cost;
75         GlobalBest.Position = particle(i).Best.Position;
76     end
77 end
78
79 BestCost = zeros(MaxIt, 1);
80 maxdist = zeros(nPop,n);
81 v_loc=zeros([nPop,VarSize]);
82 v_glob=zeros([nPop,VarSize]);
83 random=rand([nPop,VarSize])*2-1;
84 neighbourhood=zeros(nPop,nPop);
85 for j=1:n
86 for it = 1:MaxIt
87     for i = 1:nPop
88
89         % Initialize neighbourhood
90     for t=1:nPop
91         if abs(particle(i).Position-particle(t).Position)<31
92             neighbourhood(i,t)=particle(t).Position;
93         else
94             neighbourhood(i,t)=inf;
95         end
96     end
97
98     % Initialize an array to store absolute differences
99     if neighbourhood(i,:)~= inf
```

```
100     differences = abs(neighbourhood(i, :) - OptimumPos);
101
102     % Find the index of the minimum difference
103     [minDifference, minIndex] = min(differences);
104
105     % Get the value of the nearest neighbor
106     nearestNeighborValue(i) = neighbourhood(i, minIndex);
107
108     % Store the nearest neighbor value in GlobalBest.neigh(i)
109     GlobalBest.neigh(i) = nearestNeighborValue(i);
110     else
111         GlobalBest.neigh(i)= particle(i).Position;
112     end
113
114         % Update Velocity
115        if particle(i).Best.Position==particle(i).Position
116            v_loc(i,:)=0;
117        else
118             v_loc(i,:)=(particle(i).Best.Position-particle(i).Position)./
    norm(particle(i).Best.Position-particle(i).Position,2);
119        end
120        if GlobalBest.Position==particle(i).Position
121            v_glob(i,:)=0;
122        else
123            v_glob(i,:)=(GlobalBest.neigh(i)-particle(i).Position)./norm(
    GlobalBest.neigh(i)-particle(i).Position,2);
124        end
125
126        particle(i).Velocity = (alpha.*v_loc(i,:)+ beta.*v_glob(i,:)+gamma
    .*(random(i,:)./norm(random(i,:),2)))...
127             ./norm(alpha.*v_loc(i,:)+ beta.*v_glob(i,:)+gamma.*(random(i
    ,:)./norm(random(i,:),2)),2);
128        % Apply Velocity Limits
129         particle(i).Velocity = max(particle(i).Velocity, VelMin);
130         particle(i).Velocity = min(particle(i).Velocity, VelMax);
131
132         % Update Position
133         particle(i).Position = particle(i).Position + particle(i).
    Velocity;
134
135         % Velocity Mirror Effect
136         IsOutside = (particle(i).Position<VarMin | particle(i).Position>
    VarMax);
137         particle(i).Velocity(IsOutside) = -particle(i).Velocity(IsOutside
    );
138
139         % Apply Position Limits
140         particle(i).Position = max(particle(i).Position, VarMin);
141         particle(i).Position = min(particle(i).Position, VarMax);
142
143         % Evaluation
144         particle(i).Cost = CostFunction(particle(i).Position);
145
146         % Update Personal Best
147         if particle(i).Cost<particle(i).Best.Cost
148
149             particle(i).Best.Position = particle(i).Position;
150             particle(i).Best.Cost = particle(i).Cost;
151
152             % Update Global Best
153             if particle(i).Best.Cost<GlobalBest.Cost
```

```
154
155                 GlobalBest = particle(i).Best;
156
157             end
158         end
159
160 % To draw the agents:
161 %          xn(i)=particle(i).Position(1);
162 %          yn(i)=(2 - cos(xn(i))).*(xn(i)-3).^2;
163 %          yn(i)=(xn(i)).^2;
164 %
165 %          figure(2);
166 %          plot(x,y)
167 %          hold on;
168 %          plot(xn,yn,'.','markersize',10,'markerfacecolor','g');
169 %          drawnow;
170 %          hold off
171 if it==MaxIt
172     maxdist(i,j)=norm(particle(i).Best.Position-OptimumPos,1);
173 end
174     end
175     BestCost(it) = min(GlobalBest.Cost);
176     disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))
    ]);
177 end
178
179 output_min(j)=norm(GlobalBest.Position-OptimumPos,2);
180 output_max(j)=max(maxdist(:,j));
181 end
182
183 d_min=mean(output_min);
184 d_max=mean(output_max);
185 BestSol = GlobalBest;
186
187 %% Results
188
189 figure;
190 %plot(BestCost, 'LineWidth', 2);
191 semilogy(BestCost, 'LineWidth', 2);
192 xlabel('Iteration');
193 ylabel('Best Cost');
194 grid on;
```

Bibliography

Mingfu He, Mingzhe Liu, Ruili Wang, Xin Jiang, Bingqi Liu, and Helen Zhou. Particle swarm optimization with damping factor and cooperative mechanism. *Applied Soft Computing*, 76:45–52, 2019. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2018.11.050. URL https://www.sciencedirect.com/science/article/pii/S1568494618306823.

Mostapha Kalami Heris. Particle swarm optimization in matlab. In *Yarpiz*, 2015.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995. doi: 10.1109/ICNN.1995.488968.

Jacob Robinson and Yahya Rahmat-Samii. Particle swarm optimization in electromagnetics. *IEEE transactions on antennas and propagation*, 52(2):397–407, 2004. doi: 10.1109/TAP.2004.823969.

Wang W. Research on particle swarm optimization algorithm and its application. *Southwest Jiaotong University, Doctor Degree Dissertation.*, pages 36–37, 2012.