UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Particle Swarm Optimization

Léo Colin, Ezequiel Hurtado, Louis Lebreton
Directed by Philippe De Peretti

Master 1 Econometrics-Statistics 2023-2024

Abstract

Approximative optimization seems to gain on popularity as amounts of data grow substantially and exact methods exhaust computer capacities. The Particle Swarm Optimization algorithm is one of the most popular approximate algorithms so far. We will therefore show the PSO Algorithm, its specifications, how it can be used. We will also compare it in terms of performance to exact algorithms such as the Newton-Raphson algorithm or the Gradient descent method.

Contents

1 Introduction

When we talk about optimization, we should immediately think about the maximization of benefits or minimization of losses. There are two kinds of optimization algorithms that exists. Those that are exact or the ones that are approximative. In one hand, we have the ones named exact algorithms which are based on mathematical procedures. On the other hand, we have approximative algorithms, containing heuristic and metaheuristic optimization algorithms. These can be classified in multiple categories as Jain et al. (2022) states in his paper. Indeed, we have the evolutionary algorithms, the physics and chemistry based algorithms and Swarm Intelligence based algorithms also called collective intelligence.

Metaheuristic optimization is a method used to find approximate solutions to complex problems, often where finding the exact best solution is too difficult or time-consuming with computational effort. Metaheuristics are more flexible and often stochastic (chance-based) approaches. For example, unlike the PSO algorithm, gradient descent and the Newton-Raphson algorithm are not metaheuristics, and therefore not approximative algorithms, because they are specific optimization methods based on differential calculus. These kind of solutions, based on calculus are only feasible for a little amount of data. That is when metaheuristic finds its advantages and utility. In our study we will concentrate on this kind of algorithms, more precisely on the Particle Swarm Optimization Algorithm.

Its popularity has raised since its creation. Simplicity and flexibility are the most notorious qualities that have helped to its success and its family. The utility and acceptance in the academic world and its application on firms and companies evolves and so does the algorithm. For this reason we have chosen this topic.

The proof of its acceptance and relevance is the study from Poli (2007) where he founds that its litterature has grown almost exponentially through the years and that PSO is being applied in over 20 domains. These categories go from health to image analysis passing by robotics. One example of the professional use of PSO algorithm can be the Zemzami et al. (2016) paper.

Our study will be structured on a historic review of the algorithm, the explanation of the algorithm itself accompanied with its evolution, advancements and criteria. We will do a comparison between PSO and exact optimization algorithms, on particular Gradient descent optimization and Newton Raphson algorithm, on its performance and other parameters. We will also show some applications and a final conclusion of the work that we have done in the present study.

Throughout this paper our analyzes and demonstrations will focus on minimization problems but the reasoning would be similar for maximization problems.

2 PSO Algorithm principle

2.1 An algorithm imitating animal behavior

Introduced firstly on Kennedy and Eberhart (1995), PSO (Particle Swarm Optimization) is an approximate optimization algorithm. It finds its origin on metaphors linked with bird flocking and fish schooling. The principal aim was to give a method to optimize nonlinear functions through particle swarm methodology.

The metaphor behind the algorithm begins with a common space where birds try to find food. They would reach their goal to the extent that they communicate with other birds and that they take into account previous self experience. Through these two main factors they would be able to update their position, correct their trajectory and finally they would be able to found their food.

In a proper manner, the algorithms starts with a population of agents randomly placed in a space. Subsequently, they will move and interact in each of the iterations. Communication with other agents and self experience will help agents to determine the direction and pace of movement. Finally, the procedure will be repeated until they converge into a solution. The convergence is progressive and relies on iterations. These iterations formally describe the moment when particles or agents exchange information in order to reach the aimed position.

2.2 Influences of the route

Whilst the particles position is randomly assigned in the first step of the algorithm, direction and pace is not. In fact, the paths of the PSO particles are influenced by 3 components:

- **Personal Best Location (PBest):** It defines the best (minimum) solution found by an individual particle so far. This solution is updated continually as the particle explores.
- **Global Best Location (GBest):** This represents the best solution found by the entire swarm, also updated continuously as any particle in the swarm discovers a better solution.
- **Current Direction (Velocity):** Describes the particle's current momentum, influenced by its previous direction at time t-1. The particle's accessible position is determined by its current velocity.

These criteria allows us to say that PSO method may be a Swarm Intelligence based algorithm. Indeed, it responds to the principles of proximity, quality, diverse response, adaptability and stability.

The quality principle is represented through the particles response to *GBest* and *PBest*, their environment. The fact that there are significant quantities of responses should fulfill the diversity principle. In view of the fact that they change their behavior only in the presence of a "better" *GBest* answers to the stability criteria. Behaviour change in response to a new *GBest* shows the adaptability principle. Finally the fact that distance is taken into account in the algorithm satisfy the proximity standard.

When describing how the particles movement is influenced by the parameters, the word minimum is used due to the fact that our examples aim to minimize equations. It is evident that PSO can be used in maximization problems. In terms of code, the unique change that we should have is the sense of inequation.

That being said, we can introduce the equations defining the Particle Swarm Optimization procedure and its components. As a matter of fact the direction of a particle will be determined by the vector addition of these three

components exposed before. Each of them can be weighted variably. In addition, these directions can be modified using a random component, allowing for stochastic influences on the particles trajectory.

The equations will be composed of these following variables which, together, build the PSO algorithm as we know it:

- x_i: position of the particle i
- v_i: velocity of the particle i
- c_1: cognitive coefficient influence
- c_2: social coefficient influence
- w: inertia weight
- r_i: random number uniformly distributed

The position and velocity of a particle are updated as follows:

$$v_i^{t+1} = wv_i^t + c_1 r_1 (Pbest_i^t - x_i^t) + c_2 r_2 (Gbest^t - x_i^t) \tag{1}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2}$$

The velocity equation is composed by three components. Indeed, these ones are the momentum part, cognitive part and the social part. In Jain et al. (2022) each of the parts is defined and the author says that the momentum part designs as a memory or inertia as it takes into account the previous velocity. The cognitive part is the one that influence the particles to reach their *PBest*. To finish, the social part is the component that influence the particle to reach the *GBest* found by the swarm so far. Through these criteria the position criteria will be updated.

The choice of parameters is fundamental in the comportment of the PSO algorithm. In fact, they will determine the convergence speed of the particles but also the quality of the solution. We can therefore notice that understanding them will be crucial.

These parameters are the number of particles, the number of iterations, the acceleration coefficient and the inertia weight. Our approach will be to explain the debate or trade-off that researchers face when working with PSO algorithm to then explain how each of the components could be parameterized in order to get potentially good solutions.

In optimization there exists a property that aims to create an equilibrium between the exploitation and exploration. Consequently, it also applies to our algorithm. It is called the exploration-exploitation trade-off.

As in Shi et al. (2001), it seems that velocity values tend to get higher rapidly. This condition could lead the PSO Algorithm to divergence as particles would not properly analyze the area they pass by, as they go fast, and we would not found our aimed point. On the other side, if velocity is not significant enough, particles would not move and the exploration area will be reduced, leaving behind parts where we could find our optimum.

This being said, the author introduced some limits on velocity values to avoid higher values and set a maximum, *VMax*. This criteria will be determined on whether we want good exploration, with *VMax* being high, or good exploitation, where *VMax* won't be big. The last exposed frame would focus on a local exploration. Instead, when *VMax* is important, exploration is important but exploitation wouldn't be. These explains the exploitation-exploration trade-off in the PSO Algorithm context.

PSO must balance exploration (searching in new areas) and exploitation (refining the search around the best solutions found). Randomness plays a substantial role to maintain this balance by allowing some flexibility in the movement of particles.

3 Effects of Flexible Criteria

3.1 w: inertia weight

Inertia weight (w) was introduced with the idea to better control exploration and exploitation. Indeed, On Shi et al. (2001), we can find that they have proven a better performance, better results and in less iterations when

introducing the inertia weight. In theory, a huge w coefficients means a preference for exploration and inversely, when w is not that important, the work relies on exploitation.

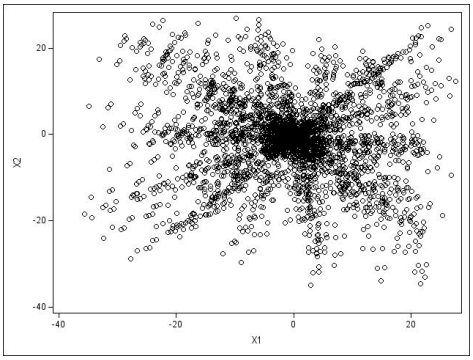

Figure 1: Rastringin; w = 1

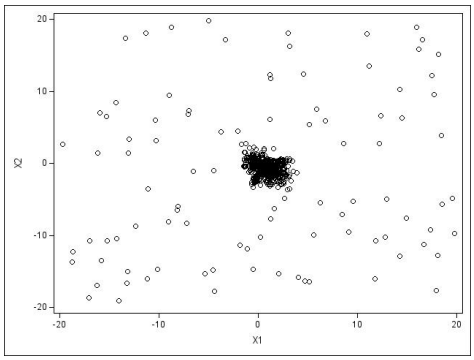

Figure 2: Rastringin; w = 0.1

Figure 1 illustrates that when there is a significant inertia coefficient (w = 1), the PSO algorithm prioritizes maximization of exploration. On the contrary, when the inertia coefficient is low (w = 0.1, as shown in Figure 2), the algorithm aims to maximize exploitation, resulting in the particles from the final iterations remaining concentrated near the global minimum. In practice, it is commonly used with multiple values. By this, we want to explain the fact that it usually begins with a value of 0.9 and decrease until it reaches the value of 0.4.

This could be translated by saying that on a first moment, one could prefer exploration and the more iterations we have the more we will move our preferences to exploitation.

$$w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} iter \tag{3}$$

By searching a harmony between exploitation and exploration, they join the constriction coefficient in the task. The inertia weight w allows us to reach an optimal solution in lesser iterations. This means that convergence will be better and it seems that we get better quality results as Jain et al. (2022) states. In the words of Juneja and Nagar (2016), the inertia weight (w) controls the influence of the velocity of the just last iteration on the actual one.

The equation that we introduce here is determined by the maximum and minimum value that w can take. We can also see that the total number of iteration and the iteration that we realize at the moment are also present on the equation on the term *iter*.

3.2 Number of particles and search space

The quantity of particles that we will display on our search space is an essential parameter on the PSO algorithm. It seems to depend on the problem we are facing as in Juneja and Nagar (2016). It seems that an optimal quantity of particles on the literature could be between 20 and 50. The more we have particles, the more the search space is covered. This will results on lesser iterations to get to the optimal solution. Nevertheless, the computational complexity rises as the particles becomes more and more important on quantities.

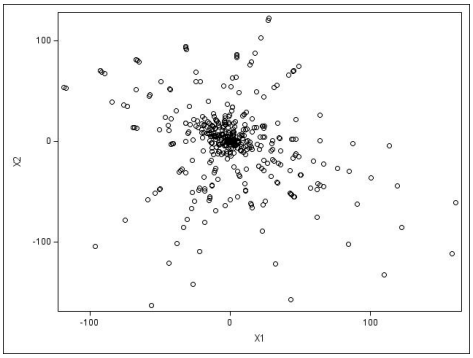

Figure 3: Rastringin; 10 particles

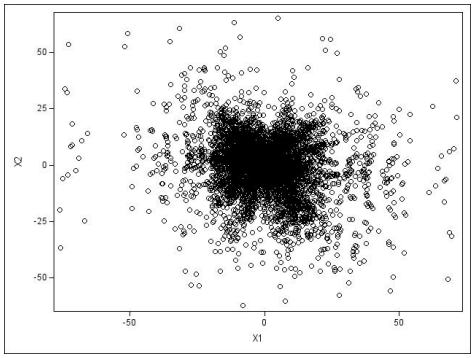

Figure 4: Rastringin; 100 particles

Figures 3 and 4 highlight the importance of using an adequate number of particles in the PSO algorithm to ensure effective convergence towards the global minimum. In Figure 3, we utilize 10 particles, but it is determined that the optimal count for achieving more precise results is beyond 50 particles.

3.3 Number of iterations and stopping criteria

The determination of the number of iterations follows the same logic as the one on the number of particles. Having more iterations will add

precision but also computational complexity, while having a lesser quantity of iterations will be less computational costly but it exist a possibility that the PSO algorithm stops before reaching the global optimum.

When we talk about the implication of having a huge number of iterations, we are talking of one specific case study. This is when the termination criterion is to realize all the iterations. The stopping criteria will be essential to find the optimum point lesser iterations.

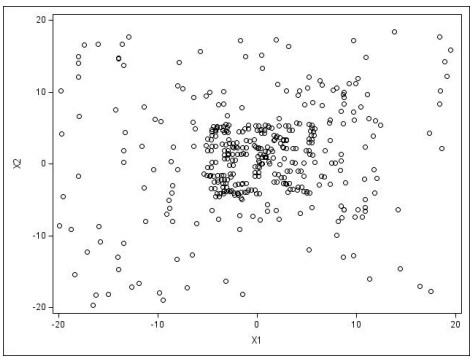

Figure 5: Rastringin; 5 iterations

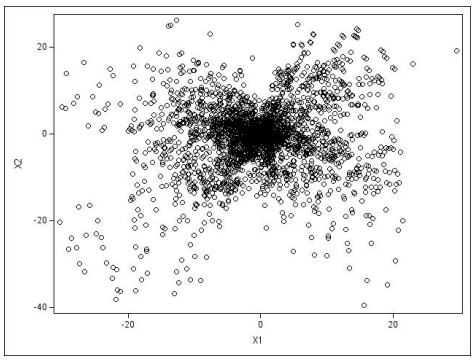

Figure 6: Rastringin; 50 iterations

Figures 5 and 6 demonstrate the necessity of having a sufficient number of iterations for the particles to effectively approach the global minimum. While Figure 6 employs 50 iterations for better visualization, the ideal number of iterations is identified as 500.

3.4 Acceleration Coefficients

The acceleration coefficients are c_1 and c_2 in our formula. They will determine how the cognitive and social part of our equation intervene in the velocity of the particles. It is important to remark that it is not just up to them to determine the velocity as they are accompanied by the random coefficient r_i. They give to PSO algorithm the meta-heuristic characteristic.

Understanding how the acceleration coefficients work may result interesting as they play an essential role on the performance of the algorithm. We will

explain the different cases that one could face on whether the values of c_i are.

- $c_1 = c_2 = 0$: The velocity would be independent from both, the cognitive and the social part of the PSO Velocity equation. We can conclude that velocity would only be determined by the inertia weight coefficient and the immediately past velocity. This means that in absence of the inertia weight, the velocity would be constant all throughout the realization of the algorithm.
$$v_i^{t+1} = v_i^t \tag{4}$$

- $c_1 = c_2$: When both of the coefficients are equal, the weight of each part of the equation is equal. Therefore, the particles will be attracted to both of the average optimal points that they know, *PBest* and *GBest* in the same measures.

- $c_1 > 0$ and $c_2 = 0$: In this case, there won't be interaction between particles as $c_2 = 0$. The *Gbest* won't be relevant. This means that all the particles will be independent and the determinants of velocity will be *PBest* and the velocity in the preceding iteration .
$$v_i^{t+1} = v_i^t + r_1 c_1 (PBest_i^t - x_i^t) \tag{5}$$

- $c_2 > 0$ and $c_1 = 0$: In this manner, the *PBest* does not play a determinant role of the velocity of the particles. Only the *GBest* or the Interactive part of the equation will be "active". The movement, as Jain et al. (2022) says, will be determined by the best position of our neighbors.
$$v_i^{t+1} = v_i^t + r_2 c_2 (GBest^t - x_i^t) \tag{6}$$

- $c_2 >> c_1$: When c_2 is significantly superior to c_1, we will be in presence of a premature convergence as the other's position has more influence than the cognitive part.

- $c_1 >> c_2$: Here, we can see that it is the exactly opposite to the last item, here explained. Therefore, when c_1 is bigger than c_2 the cognitive part will be more influential than the interaction part of our velocity equation. This means that the particles will move in an itinerant way.

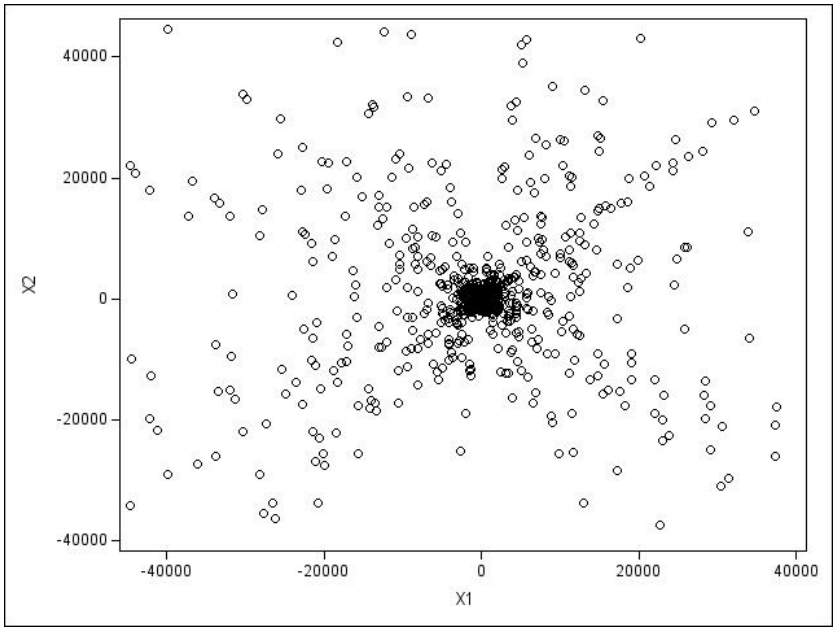

Figure 7: Rastrigin function ; c_1=0.5 ; c_2=4

From Figure 7, we observe that when c_1 is set to 0.5 and c_2 to 4, the PSO algorithm does not converge for the Rastrigin function. This phenomenon can be attributed to the high value of c_2 relative to c_1, which overly emphasizes social learning or collective behavior. In such a scenario, particles are heavily influenced by their neighbors positions and tend to converge prematurely. In this case, we can potentially be missing the global optimum, especially in complex multimodal functions like the Rastrigin function.

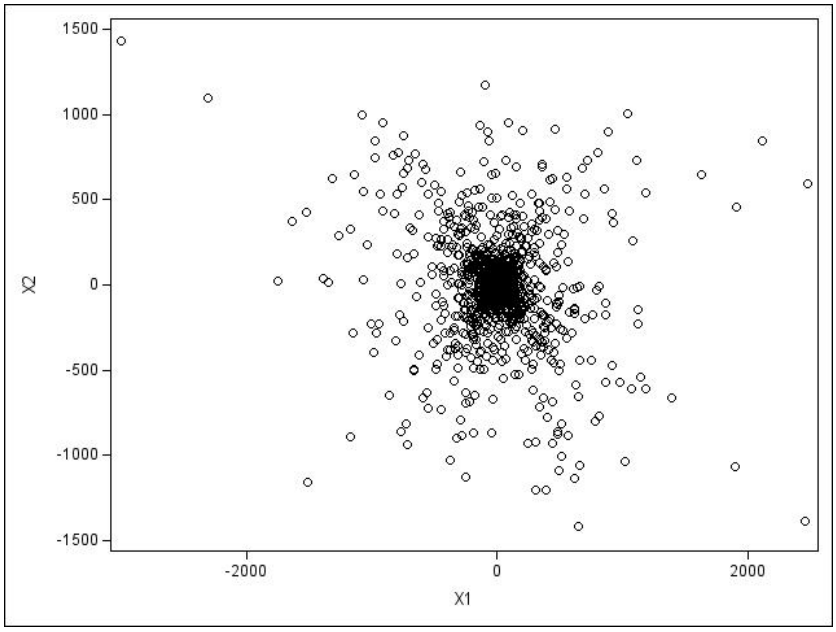

Figure 8: Rastrigin function ; c_1=4 ; c_2=0.5

On the contrary, as shown in Figure 8, when c_1 is increased to 4 and c_2 is reduced to 0.5, the PSO algorithm successfully converges. This configuration, with a higher c1 and a lower c2, allows particles to rely more on their own experience and exploration capabilities. This increased emphasis on individual exploration helps the swarm to more effectively search the complex landscape of the Rastrigin function.

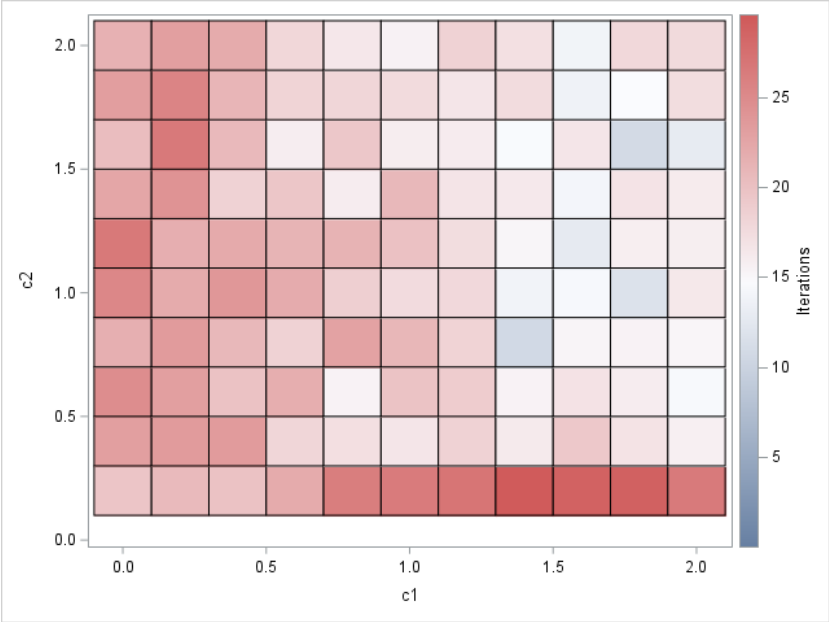

Figure 9: PSO algorithm on Rastrigin function

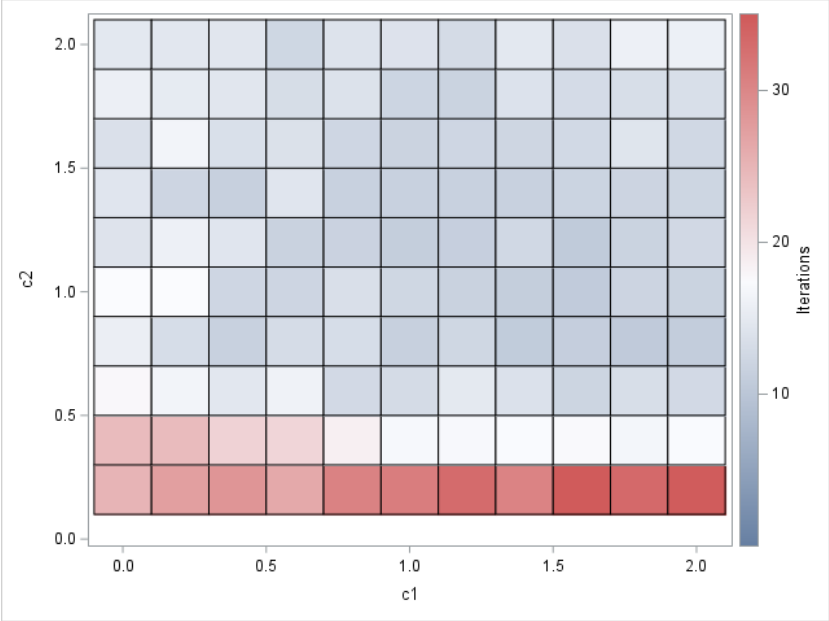

Figure 10: PSO algorithm on Ackley function

Through the use of heatmaps, Figures 9 and 10 demonstrate the number of iterations required for the PSO algorithm to converge for various combinations of c_1 and c_2 values. These data was derived from averages obtained through Monte Carlo simulations. In these figures, the stopping criterion is based on $\|\mathbf{GlobalBest}\|^2 < \epsilon$. These visualizations reveal distinct patterns for the two different functions analyzed. However, a general trend can be observed: when c_2 is less than 0.5, the algorithm struggles to converge.

$$c_1 = 2 - \frac{iter \times (2 - 0.1)}{iter\_max}; \tag{7}$$

$$c_2 = 0.1 + \frac{iter \times (2 - 0.1)}{iter\_max}; \tag{8}$$

From a practical standpoint, it is highly advantageous to dynamically adjust the values of c_1 and c_2 throughout the iterations of the PSO algorithm. Initially, a higher c_1 and a lower c_2 should be set to prioritize exploration, granting individual particles greater autonomy to traverse the search space. This approach ensures a broad and diverse exploration in the early stages. As the iterations progress, gradually increasing c_2 while decreasing c1 shifts the focus towards exploitation. This gradual transition leverages the collective intelligence of the swarm, allowing particles to converge more effectively on optimal or near-optimal solutions. This adaptive strategy balances exploration and exploitation, optimizing the algorithm's performance over its runtime.

It seems that there is also an important relation that should be considered and it is the relation between c_1 and c_2. If we refer to Carlisle, Dozier et al. (2001), we can see that they did several experimentation on this concern. They conclude that in general terms, acceptable values for c_1 and c_2 should be 2.8 for c_1 and 1.3 for c_2. They also tried to explain that an optimal guideline is to have $c_1 + c_2 = 4.1$ in general .

3.5 Constriction coefficient

The constriction coefficient is meant to assure the convergence on the PSO algorithm. It allows us to avoid using $VMax$ and w, the inertia weight. There are different cases when using the constriction factor depending on whether it is accompanied by the velocity clamping or the coefficient value

itself. When using the constriction coefficient, the velocity equation update as follows :

$$v_{ij}^{t+1} = \chi[wv_i^t + c_1 r_1(PBest_i t - x_i^t) + c_2 r_2(GBest^t - x_i^t)] \tag{9}$$

When we have a proximity between the last *PBest* and the *GBest*, the constriction coefficient will perform an intensive local search. If not, the algorithm will perform a global search. Along with assuring the convergence, the constriction coefficient can prevent the algorithm from collapse.

When we use it with the inertia weight, the convergence seems to be even faster as Talukder (2011). In addition, this paper also states that in algebraic terms, the velocity equation will give the same results on whether we use the inertia weight or the constriction coefficient.

3.6 Constraints

There is a possibility that the swarm explodes. This means that some particles can go beyond the search space. This could potentially invalidate our results. Therefore, We will have to look for a solution. The Velocity clamping, inertia weights w and the constriction coefficient seems to be the potential solutions. Nevertheless, these procedures do not told us about how the particles will remain on our search space.

As in Talukder (2011), this represents a problem that should be addressed. The solution that is given is to take into account the boundary conditions. This solutions has diverse categories that we can use. These can be :

- ABS - Absorbing boundary Conditions : This means that by the moment the particle goes outside of our search space, we will take it and place it at the frontier of the space with its velocity in this dimension being zero.
- RBC - Reflecting boundary conditions : Here what will happens is that once the particle goes outside of our search space, we will reintroduce it in the wall but the velocity will be the opposite of the one when this particle went out of our search space in this dimension.
- DBC - Damping boundary conditions : In this case, it will be similar to the one explained just above but the reflecting velocity won't be the

same due to the random factor that will reflect only a part of this total velocity.

- IRBC - Invisible Reflecting boundary conditions: The particles won't be reintroduced at the frontier but the velocity will change on the dimension as the RBC case when the fitness value will be taken into account.

- IDBC - Invisible dampling boundary conditions : Just as in IRBC, the particles will stay out of the search space and the velocity will change on the dimension where the particle went out. This velocity will be influenced by the sign and a random coefficient.

- The Penalty Function Approach - We also tested a method which consists of penalizing the objective function each time a particle leaves the definition space. The further a particle moves away from this space, the more the objective function is penalized. This penalty manifests itself as an artificial increase in the function image at points that have exceeded the search space. The score of these particles will then be large (which is "bad" in our case of minimization), the particles will therefore no longer have any incentive to search in this area and they should tend to return inside the definition space. In this scenario, the function that we consider is no longer only the objective function but the following "total" function which brings together the objective function and the penalty:

$$TF(X) = f(X) + \alpha \times Penalty(X) \tag{10}$$

With *f(X)*, the objective function, α a penalty coefficient that must be large enough in order to make particles converge quickly to the search space and *Penalty*, a function which penalizes particles proportionally to their deviation from the search space. Therefore, a particle that lies in the search space isn't penalize.

Figure 11 is a one-dimensional graphical representation of the effect of the penalty on the objective function.

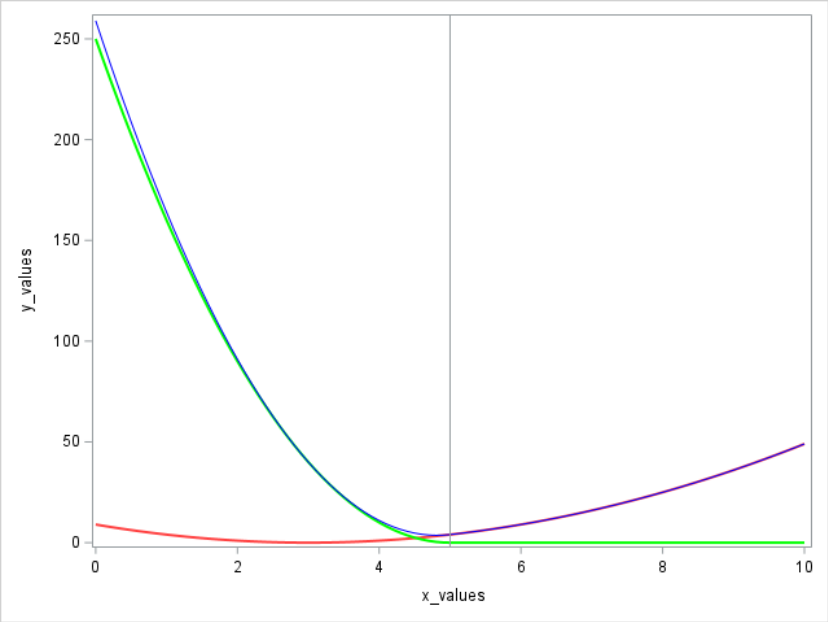

Figure 11: Objective Function with Penalty

3 curves are represented on this graph, in red the objective function, in green the penalty function and in blue the "total" function. The constraint represented is x≥5. We see that the total function is penalized when the constraint is not respected.

In our case, for example, assuming that the i-th coordinate of each particle x_j must be between two values $lb \leq x_{j,i} \leq ub$, we have (considering here that lb is negative and ub positive) :

$$\text{Penalty}(x_{j,i}) = \begin{cases} x_{j,i} - lb & \text{si } x_{j,i} \leq lb, \\ x_{j,i} + ub & \text{si } x_{j,i} \geq ub, \\ 0 & \text{si } lb \leq x_{j,i} \leq ub. \end{cases}$$

We created our algorithm so that the greater the number of coordinates of a particle leaving the search space, the more the particle are penalized.
This technique amounts to injecting the constraint into the augmented objective function.

We tried three ways to penalize the algorithm so that it satisfied the imposed constraints:

- the Penalty Function Approach
- the ABS criterion
- a complete reset of the particle coordinates

We choose to mainly use the last technique. It seems to be the most appropriate due to its simplicity and its performance (will be demonstrated below). The second method is limited, if a particle reaches a global best on the edge of the domain which is not the optimal solution, the particles may tend to converge on the edge of the domain and therefore explore less and/or less well the entire searching space. Indeed, many of them will stage at a specific point on the edge of the domain and therefore there won't be more exploration or exploitation. The first technique is just as limited because it does not prevent particles in the absolute from leaving the search space, which can allow the algorithm to find an optimal solution that does not respect the constraints (the particle comes out, updates the global best and subsequently even if it returns to the search space the algorithm does not find a better solution).

Let's apply these three constraints to PSO algorithm on Rastrigin function and then conduct Monte Carlo simulations with 100 iterations. This will allow us to compare the average number of iterations required for the algorithm to converge. Here the stopping criterion will be $\|\mathbf{GlobalBest}\|^2 < \epsilon$.

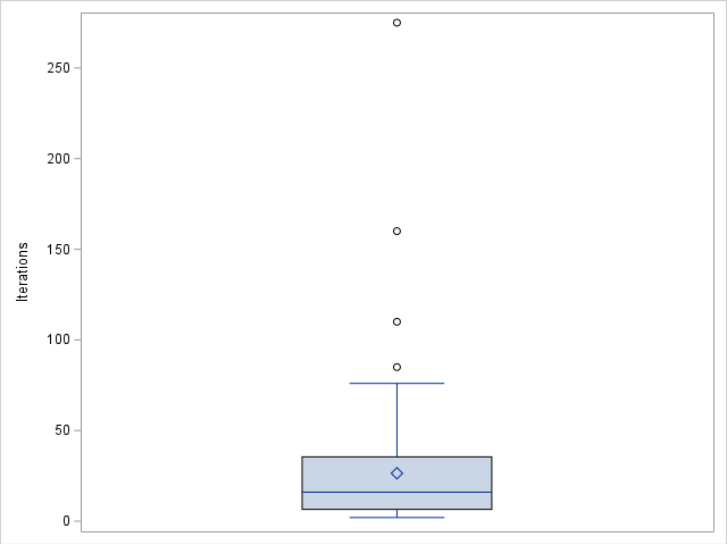

Figure 12: Monte Carlo simulation on Rastrigin function without constraints ; 500 iterations ; 100 particles ; Boxplot

Mean	Median	Variance	Sd
26.41	16	1272.7494	35.675613

Figure 13: Monte Carlo simulation on Rastrigin function without constraints ; 500 iterations ; 100 particles ; Descriptive statistics

When no constraints are applied to our PSO algorithm, the average number of iterations is 26.41, with a median of 16. We observe that there are extreme values exceeding 100 iterations, which skew the average and increase the standard deviation.

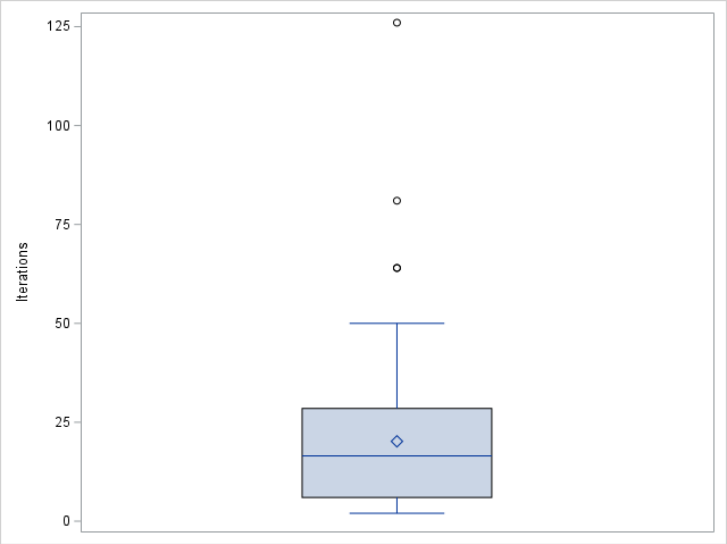

Figure 14: Monte Carlo simulation on Rastrigin function under Absorbing boundary Conditions ; 500 iterations ; 100 particles ; Boxplot

Mean	Median	Variance	Sd
20.18	16.5	370.41172	19.246083

Figure 15: Monte Carlo simulation on Rastrigin function under Absorbing boundary Conditions ; 500 iterations ; 100 particles ; Descriptive statistics

Let's now implement the Absorbing Boundary Conditions. With this approach, when a particle exits the search space, it is repositioned at the boundary of this space. Thanks to this constraint, there appears to be a reduction in extremely high values. This has effectively lowered the average number of iterations.

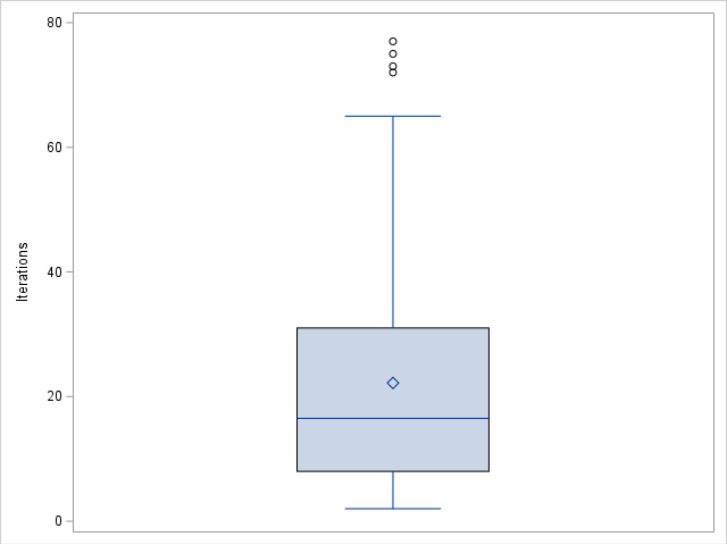

Figure 16: Monte Carlo simulation on Rastrigin function with a penalty ; 500 iterations ; 100 particles ; Boxplot

Mean	Median	Variance	Sd
22.18	16.5	362.83596	19.048253

Figure 17: Monte Carlo simulation on Rastrigin function with a penalty ; 500 iterations ; 100 particles ; Descriptive statistics

We can now introduce a penalty to the Rastrigin function, designed to arbitrarily increase the value of the image outside the search space. Consequently, when a particle exits the search space, it incurs a penalty. This penalty further reduces the number of extreme iterations. With this penalty applied, we achieve an average of 22.18 iterations.

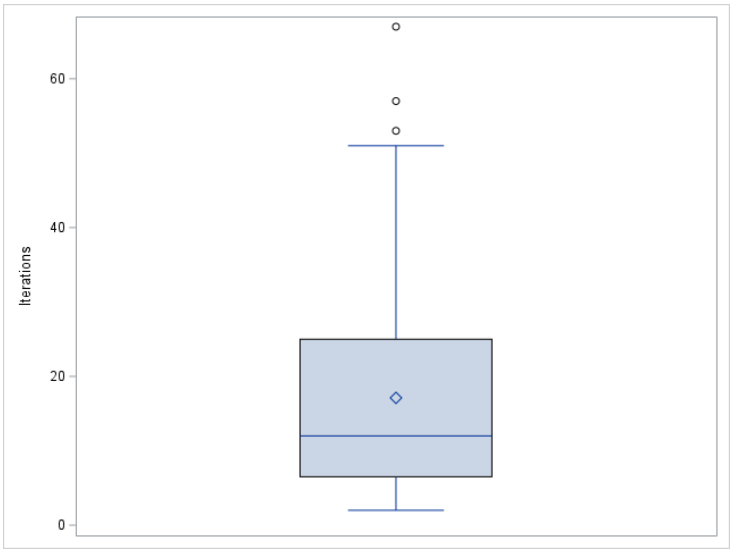

Figure 18: Monte Carlo simulation on Rastrigin function with complete reset of the particle coordinates that diverge excessively ; 500 iterations ; 100 particles ; Descriptive statistics

Mean	Median	Variance	Sd
17.11	12	212.48273	14.576787

Figure 19: Monte Carlo simulation on Rastrigin function with complete reset of the particle coordinates that diverge excessively; 500 iterations ; 100 particles ; Boxplot

Let's now apply the latest constraint technique. Under this condition, when a particle exits the search space, it is randomly repositioned within the search area. This constraint significantly reduces the number of extreme values. With this constraint in place, we achieve the lowest average number of iterations (17.11), making it the most effective constraint compared to the previous ones.

It is also possible to enter a parameter, *Vmax*, that refers to the velocity clamping: it imposes a limit on the particle's movement speed to prevent particles from diverging. This constraint ensures that particles do not stray

too far from the potential solution space, aiding in the convergence of the algorithm. That is how the three components listed before in section 2.2 influence the direction of the swarm.

3.7 Stopping Criteria

- Maximum number of iterations : Setting a maximum number of iterations as a stopping criterion is essential for a few reasons. Firstly, it ensures computational efficiency, preventing the algorithm from running indefinitely and consuming excessive resources. Secondly, it helps in assessing convergence. Oftentimes, beyond a certain point, further iterations don't significantly improve the solution. Finally, it aids in practical experimentation, allowing for consistent comparisons across different runs and adjustments based on observed performance. This approach strikes a balance between finding an adequate solution and managing computational constraints.

- Global best static during several iterations: This criterion halts the algorithm if the global best solution remains unchanged over a predetermined number of iterations. The reasoning behind this is that if the best solution found by the swarm does not improve for a consecutive number of iterations, it is likely that the algorithm has reached a plateau or is trapped in a local optimum. This criterion ensures that computational resources are not wasted unnecessarily on iterations that do not contribute in an improved solution. It is important to choose the number of static iterations carefully, as setting it too low may result in premature termination, while setting it too high may lead to unnecessary computation.

- $\|\mathbf{GlobalBest}\|^2 < \epsilon$: This stopping criterion is based on the precision of the solution. The algorithm stops when the sum of the squares of the global best solution's components falls below a specified threshold, in this case, ϵ. This threshold represents the desired level of precision or accuracy of the solution. This criterion is particularly useful when the problem requires a high degree of accuracy, and it ensures that the algorithm does not terminate before reaching a sufficiently precise solution. However, it also requires careful calibration to avoid excessively long run times or premature termination.

- *Gradient* $< \epsilon$: The implementation of a gradient-based stopping criterion involves halting the algorithm when the gradient of the global best solution is less than a specified threshold, such as ϵ. This criterion is based on the concept that a small gradient indicates a flat or level area in the search space, suggesting that the algorithm is close to an optimum (local or global). Using the gradient as a stopping criterion is effective in preventing unnecessary computations once the solution has settled in an optimum region. However, it requires the problem to be differentiable and may not be suitable for all types of PSO applications.

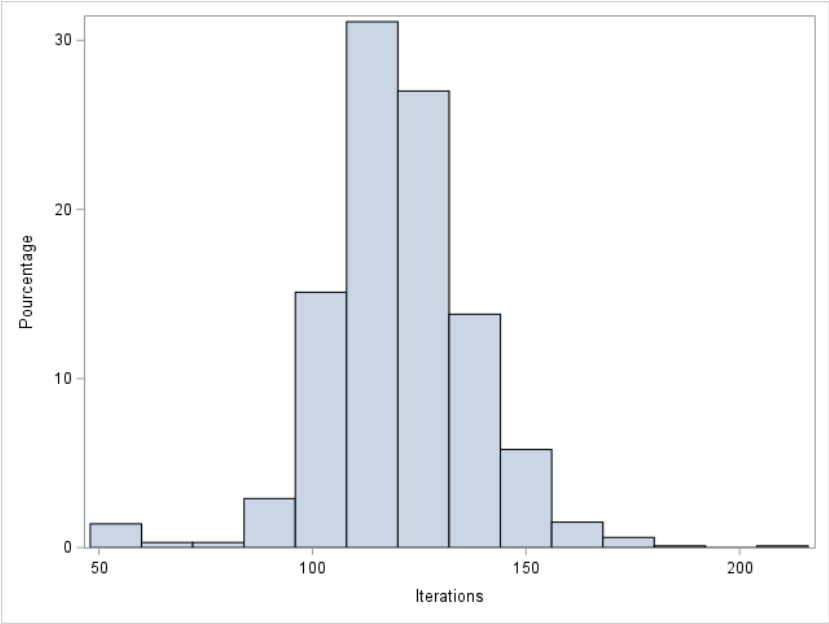

Figure 20: Monte Carlo simulation on a simple function ; Gb static 50 times

Figure 20 presents an example of a Monte Carlo simulation applied to a basic convex function, formulated as $2x^2 + 10$. This figure highlights the number of iterations required for the "Global Best" to remain unchanged for 50 consecutive iterations. It is observed that the majority of the implemented PSO (Particle Swarm Optimization) algorithms require between 100 and 150 iterations to achieve this consistency in the "Global Best". Furthermore, the distribution of results from different simulations appears to follow a normal distribution.

4 Applications of PSO algorithm

4.1 Applications on some test functions for optimization

In this subsection, we will apply our PSO algorithm on optimization test functions in order to verify its effectiveness. These functions are useful in applied mathematics to evaluate characteristics of optimization algorithms, such as convergence rate, precision, robustness or general performance.

For the following functions evaluated, we employed the parameters:

- Starting range: $[-100, 100]$
- Iterations: 500
- Particles: 100
- c1: 2
- c2: 2
- w: gradually decreasing from 0.9 to 0.4 over the course of iterations.

Easom function

$$f(X) = -\cos(X_1)\cos(X_2)\exp(-((X_1-\pi)^2 + (X_2-\pi)^2)) \qquad (11)$$

Its global optimum point is $f(\pi, \pi) = -1$.

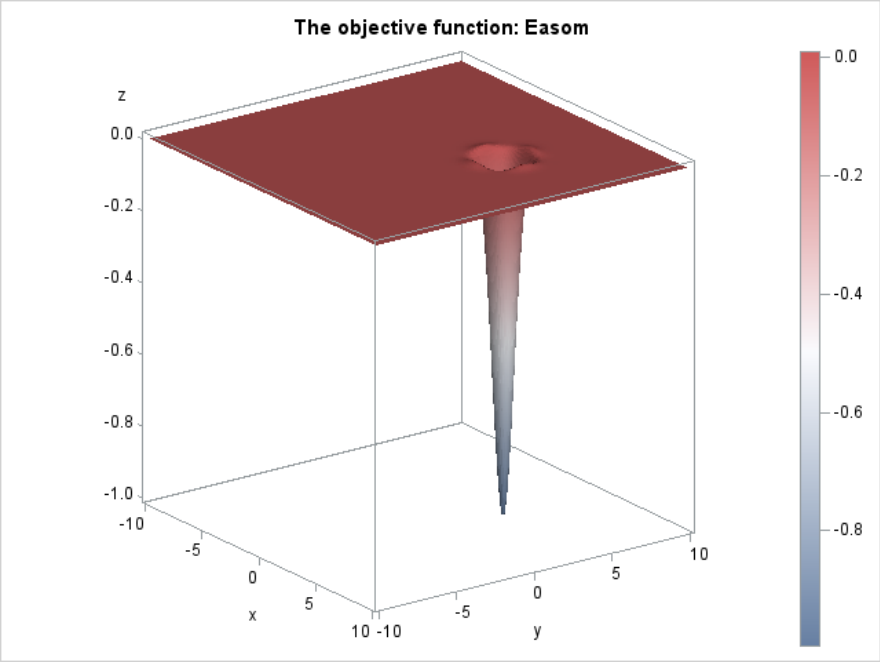

Figure 21: Easom function

Gbest_during_convergence		
1	-2.787459	8.9026396
50	3.1265264	3.1098492
100	3.1428894	3.141736
150	3.1415881	3.1415393
200	3.1415924	3.1415927
250	3.1415927	3.1415927
300	3.1415927	3.1415927
350	3.1415927	3.1415927
400	3.1415927	3.1415927
450	3.1415927	3.1415927
500	3.1415927	3.1415927

Figure 22: PSO algorithm on Easom function (2 dim) ; Gbest for each iteration

Let's apply the PSO algorithm to the Easom function using the previously mentioned parameters. The figure 22 illustrates that from the 100th

iteration, the Global Best (Gbest) of our particles is already very close to our global minimum, with an accuracy of about 10^{-2}. This indicates that the algorithm has performed effectively.

Sphere function

$$f(X) = (X_1 - 1)^2 + (X_2 - 1)^2 + (X_3 - 1)^2 - 4 \tag{12}$$

Its global optimum point is $f(1, 1, 1) = -4$.

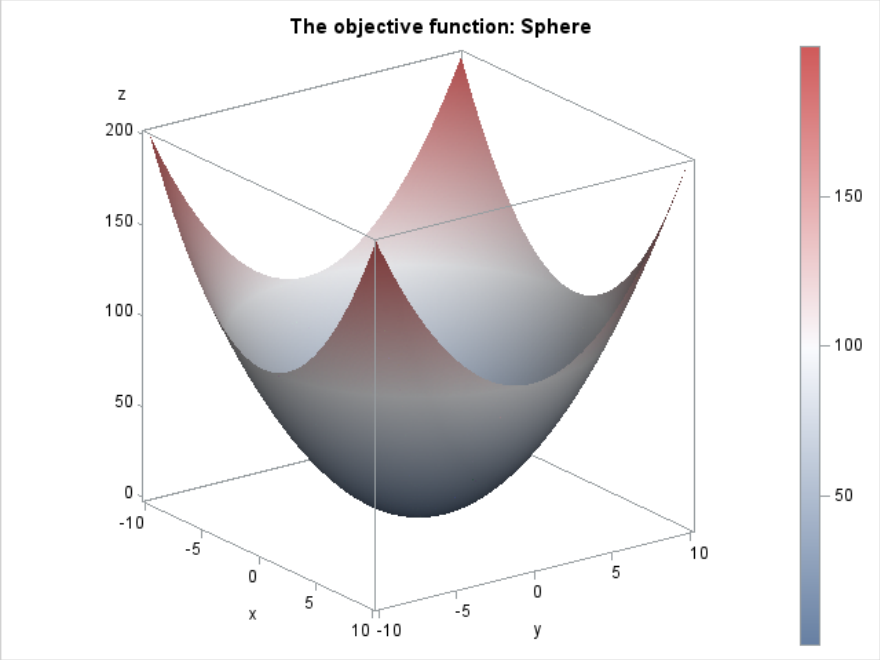

Figure 23: Sphere function

Gbest_during_convergence			
1	-8.50005	-36.70257	-20.67555
50	0.3587347	1.6903587	0.15544
100	0.7466269	1.1391553	1.4843261
150	0.9947948	0.8954422	1.2458609
200	1.0691022	0.8114228	1.1255342
250	1.0064037	0.9996638	1.0023474
300	1.0005582	0.9981813	1.0000772
350	1.0000317	1.0000052	1.0000212
400	1	1	1
450	1	1	1
500	1	1	1

Figure 24: PSO algorithm on Sphere function (3 dim) ; Gbest for each iteration

Regarding the 3-dimensional Sphere function, the figure 24 indicates that from the 250th iteration, the Global Best (Gbest) of our particles is already very close to our global minimum, with an accuracy of approximately 10^{-2}. Once again, this demonstrates a successful convergence.

Rastrigin function

$$f(X) = 20 + (X_1^2 - 10\cos(2\pi X_1)) + (X_2^2 - 10\cos(2\pi X_2)) \qquad (13)$$

Its global optimum point is $f(0, 0) = 0$.

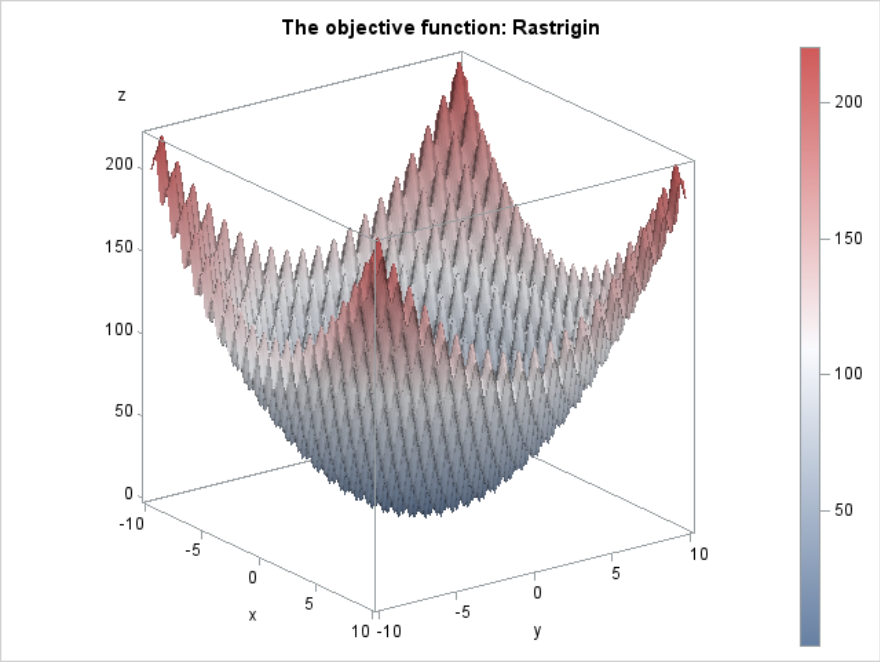

Figure 25: Rastrigin function

Gbest_during_convergence		
1	-5.306242	-1.107476
50	-1.053437	-0.018628
100	-1.053437	-0.018628
150	-1.053437	-0.018628
200	-1.053437	-0.018628
250	-1.008173	0.0360387
300	0.0000782	0.0001867
350	-1.067E-7	2.8838E-9
400	2.334E-10	1.0549E-9
450	2.334E-10	1.0549E-9
500	2.334E-10	1.0549E-9

Figure 26: PSO algorithm on Rastrigin function (2 dim) ; Gbest for each iteration

For the Rastrigin function, it is evident from the 300th iteration (as shown in the figure 26) that there is indeed a convergence of the particles.

Ackley function

$$f(X) = -20\exp\left[-0.2\sqrt{0.5(X_1^2 + X_2^2)}\right] - \exp\left[0.5(\cos(2\pi X_1) + \cos(2\pi X_2))\right] + e + 20 \tag{14}$$

Its global optimum point is f(0,0) = 0.

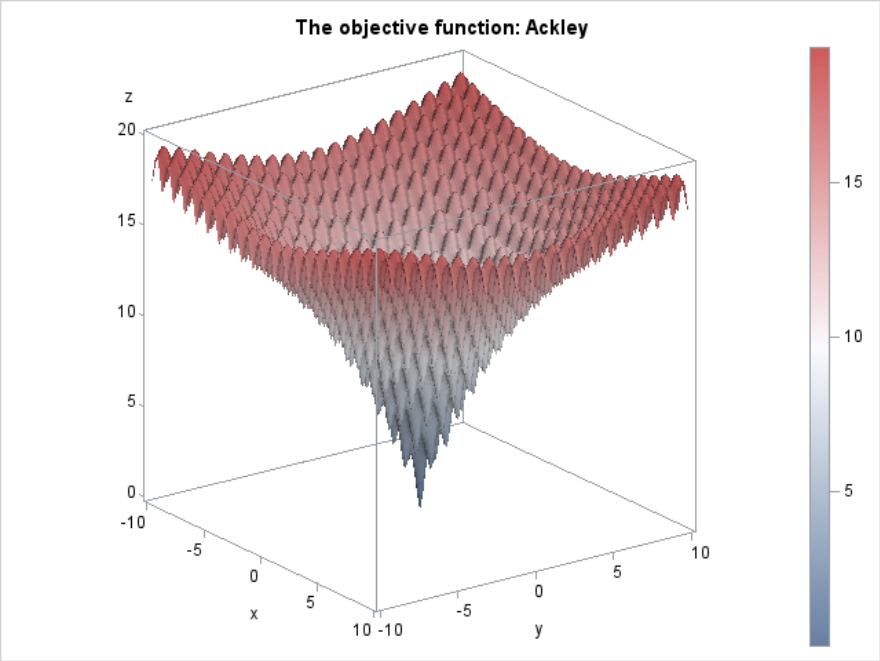

Figure 27: Ackley function

Gbest_during_convergence		
1	-11.68974	2.3823751
50	-0.015423	0.1048427
100	-0.000926	0.0056407
150	-0.001408	-0.000581
200	-0.001266	-0.000054
250	-0.000011	0.0000143
300	3.5512E-9	2.9064E-9
350	1.468E-11	-9.8E-11
400	1.118E-14	-8.32E-16
450	-2.89E-16	-1.85E-16
500	-2.89E-16	-1.85E-16

Figure 28: PSO algorithm on Ackley function (2 dim) ; Gbest for each iteration

For the Ackley function, it is observable from the 100th iteration (as illustrated in the figure 28) that the Global Best (Gbest) closely approximates the global minimum, with a precision of about 10^{-2}.

Rosenbrock Function

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \tag{15}$$

Its global optimum is obtained when we have $f(1, 1) = 0$

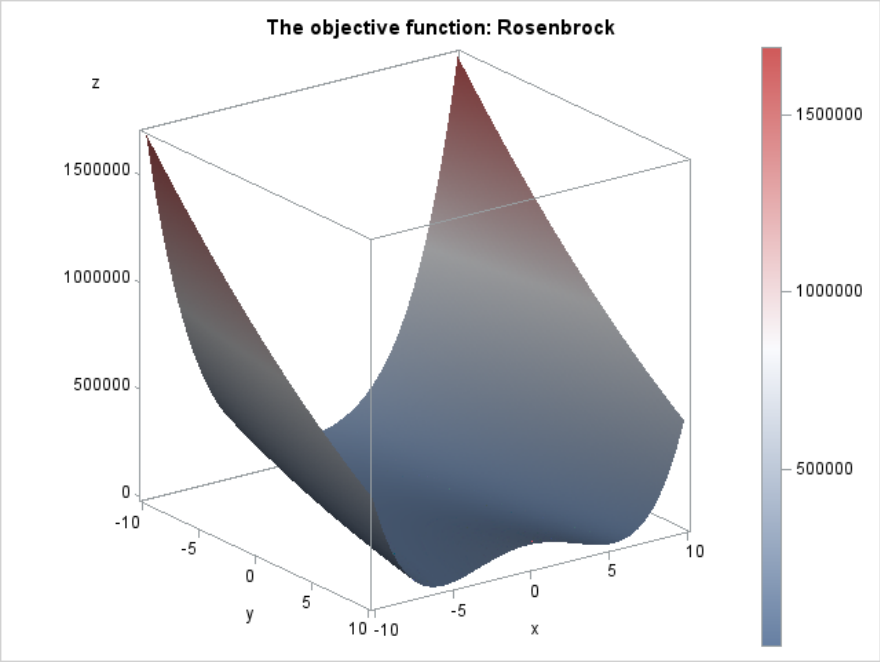

Figure 29: Rosenbrock Function

Gbest_during_convergence		
50	2.4863763	6.2124566
100	2.492717	6.2169694
150	2.3578149	5.5344187
200	1.4571362	2.1190924
250	1.3601643	1.8459128
300	0.9993732	0.9988201
350	0.9999965	0.9999924
400	1	1
450	1	1
500	1	1

Figure 30: PSO Algorithm on Rosenbrock Function; Gbest for each iteration

As we can see, the Rosenbrock Function approaches the optimal point at its 300th iteration. A more precise point is found at its 400th iteration where we have the optimal point.

In this section we have sought to find a global optimum, but we could also use PSO to find all the local optima of a function. All we would have to do is looking at all the points that cancel the gradient and then analyse the Hessian matrix to determine the nature of that point (minimum, maximum, saddle point, plateau).

4.2 Performance under n dimensions

The PSO algorithm can be applied to functions of any dimensionality.

Absolute value function in n dimensions

$$f(x) = \sum_{i=1}^{n} |x_i|$$

Dim	f_Gbest	CPU_Time
1	0.0000	4.04
2	0.0000	4.71
3	0.0000	6.31
5	0.0000	10.51
10	0.0001	19.26
20	0.1609	43.76
50	3.6779	69.46
100	15.4479	87.93

Figure 31: Comparison of the number of dimensions with the Gbest obtained and the necessary CPU time for the Absolute value function

Figure 31 illustrates the performance of applying the PSO algorithm to a multi-dimensional Absolute Value function. For 1000 iterations and 1000

particles, we observe that as the number of dimensions increases, the final global best deviates from the global minimum, indicating reduced precision of the algorithm. Furthermore, the required CPU computation time increases as the dimensions expand.

When working with a function in a 100-dimensional space, the accuracy of convergence can be enhanced by, for instance, increasing the number of particles and the number of iterations. For example, for this specific function, multiplying the number of particles and iterations results in a Gbest value closer to 0 albeit with a naturally increased CPU time.

Sphere function in n dimensions

$$f(x) = \sum_{i=1}^{n} (x_i - 1)^2$$

Dim	f_Gbest	CPU_Time
2	0.0000	8.45
3	0.0000	9.89
5	0.0000	18.11
10	0.0000	27.90
20	0.0000	65.93
50	1.9651	134.62
100	21.5677	186.67

Figure 32: Comparison of the number of dimensions with the Gbest obtained and the necessary CPU time for the Sphere function

Based on the figure 32, let's examine the performance and computation time on a Sphere-type function (a more complex function) of n dimensions, whose global minimum is 0. We observe the same phenomenon here as well.

We can conclude that the PSO algorithm seems to remain relevant and effective as long as the objective function is of dimension less than 50. Beyond

that, the algorithm necessarily loses precision but can still remain useful if we use different parameters such as a higher number of particles and iterations.

4.3 Application under constraints

We will see if PSO is still as efficient when it optimizes a constrained function. As it seems to be the most effective criterion, we use the particle reset technique as a constraint criterion. In this subsection we show that the PSO algorithm is efficient by applying it to two constrained functions.

We first tried it on the following program:

$$\begin{cases} f(x,y) = x^2 + y^2 \\ x + y - 10 = 0 \end{cases}$$

Solving this program is simple, we can find the solution in particular by the Lagrange multiplier method. The derivatives of the Lagrangian give us that x=y and therefore by reinjecting into the constraint we obtain that (x,y)=(5,5).

Here are the results of our PSO algorithm:

The SAS System

nbr_iter	X		Gb		Gb_image
200	17.689317	-7.689317	4.9927409	5.0072591	50.000105
	8.7798671	1.2201329			

Figure 33: Results of PSO on the previously defined function

With only two particles and therefore in a very low calculation time the algorithm converges perfectly.

We then tried the Rosenbrock function constrained to a disk centered at zero and with radius $\sqrt{2}$ as follows:

$$\begin{cases} f(x,y) = (1-x)^2 + 100(y-x^2)^2 \\ x^2 + y^2 \leq 2 \\ -1.5 \leq x \leq 1.5 \\ -1.5 \leq y \leq 1.5 \end{cases}$$

This constraint makes it possible to concentrate the search around a point. The known global solution of this program is (x,y)=(1,1). The algorithm also converges correctly.

The SAS System

nbr_iter	Gb		Gb_image
200	0.999351	0.9986689	5.3328E-7

Figure 34: Results of PSO on Rosenbrock function constrained

5 Comparison with Gradient descent and Newton-Raphson algorithm

5.1 Presentation of Gradient descent method

5.1.1 Gradient descent principles

- **Concept:** Gradient descent is a fundamental optimization algorithm. It is particularly effective for problems where the goal is to find the global optimum of a function. The basic principle behind gradient descent is to iteratively adjust the parameters of the function by moving in the direction opposite to the gradient of the function at the current point. The gradient of a function at any point gives the direction of

the steepest ascent. Therefore, by moving in the opposite direction (steepest descent), one can gradually reach the local minimum of the function.

- **Algorithm:** The parameters are updated iteratively as follows:

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla f(\theta_{\text{old}}) \tag{16}$$

where θ represents the parameters, α is the learning rate, and ∇f is the gradient of the function f. The choice of learning rate is crucial. A rate too small leads to slow convergence, while a rate too large can cause overshooting. There are methods to control this rate during the optimization process which are not studied here.

5.1.2 Advantages and limitations of this method

Gradient Descent has several advantages, firstly this method is simple to understand and relatively easy to implement (the machine only needs to calculate the gradient of the function at a point). It is therefore appreciated in applications on large data sets due to its low calculation time. Its main disadvantages are its slow convergence (which depends on the complexity of the search space), its sensitivity to the starting point and the difficulty of finding the right learning rate.

5.2 Presentation of Newton-Raphson method

5.2.1 Newton-Raphson method principles

- **Concept:** The Newton-Raphson method is used for function optimization. This approach is utilized to find points where a function reaches its minimum or maximum. It starts with an initial estimate, θ_0, and uses the Hessian matrix to enhance the efficiency of the optimum search.

- **Algorithm:** The approximation of θ is updated iteratively as follows:

$$\theta_j = \theta_{j-1} - \alpha \times H(\theta_{j-1})^{-1} \times \nabla f(\theta_{j-1}) \tag{17}$$

where θ_j is the current estimate, α is the learning rate, $H(\theta)$ is the Hessian matrix of the function f at point θ, and $\nabla f(\theta)$ is the gradient

of f at θ. After each update, the stopping criterion is calculated:

$$sc = \sum \|\theta_{ij} - \theta_{i(j-1)}\| \tag{18}$$

where sc is the sum of the norms of the differences between successive estimates of θ. If sc is less than a threshold ϵ, the algorithm stops, indicating that the approximation has converged to an optimum. Of course, we can use other stopping criteria which are mentioned above, in section 3.

5.2.2 Advantages and limitations of this method

The Newton-Raphson method, for its part, generally offers faster convergence than gradient descent, in particular for well-conditioned problems. It (can) automatically adjust the step size (no need to specify a learning rate). It is particularly effective for small data sets. However, unlike gradient descent it requires the calculation of second derivatives (of the Hessian matrix), which is more computationally expensive, especially for complex functions. It is therefore necessary to have class C2 type functions (continuous and differentiable first derivative and continuous second derivative functions) which can be restrictive. It is also sensitive to initialization and can diverge if the Hessian is not positive definite. Just like gradient descent, it has more difficulty converging towards the optimal solution in the presence of non-convex problems and can converge towards local minima or saddle points in the case of non-convex objective functions.

When using these types of algorithms (gradient descent or Newton-Raphson) it is necessary to try different initial parameters to ensure convergence towards the correct solution. It is also possible to noise this solution in order to observe if the algorithm converges again towards this same solution. If the algorithm finds different values, they must be compared and put into perspective within the framework of use to determine which is the best solution.

5.3 Comparison of these methods with PSO

5.3.1 Classical methods may be more effective than PSO algorithm

We first started by testing the different algorithms on an econometrics problem. We used the following data generating process:

$$Y(t) = 0.7X1(t) + 1.2X2(t) - 0.4X3(t) + 0.3 + eps(t) \tag{19}$$

To find the parameters $\hat{\beta}_i$ of the DGP, the objective is then to minimize the following sum of squares:

$$\sum_{t=1}^{T}(y(t) - \hat{\beta}_1 x_1(t) - \hat{\beta}_2 x_2(t) - \hat{\beta}_3 x_3(t) - \hat{\beta}_4)^2 \tag{20}$$

By successively applying our three algorithms we had the following results:

- All three methods found the correct parameters
- The Newton-Rapson method proved to be the most effective, followed by gradient descent. The PSO algorithm converged much more slowly to the optimal solution.

To represent these results we have the following graphs which allow us to compare the convergence speeds:

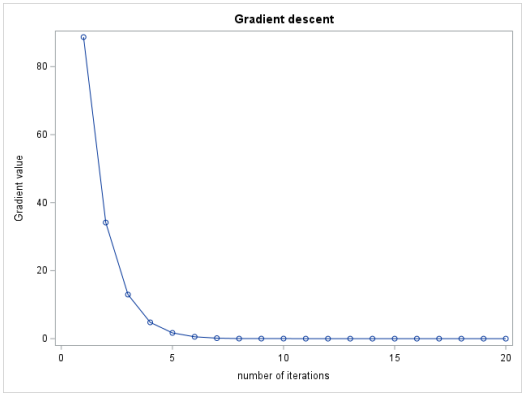

Figure 35: Speed of convergence of Gradient descent method

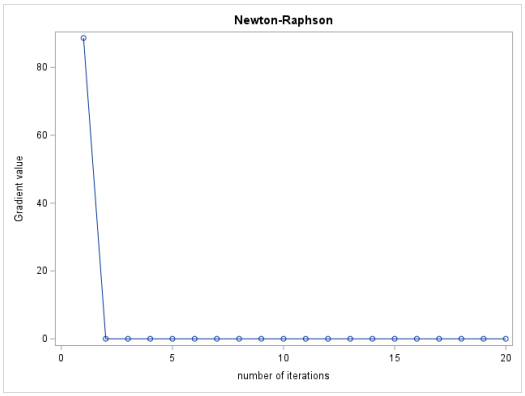

Figure 36: Speed of convergence of Newton-Raphson method

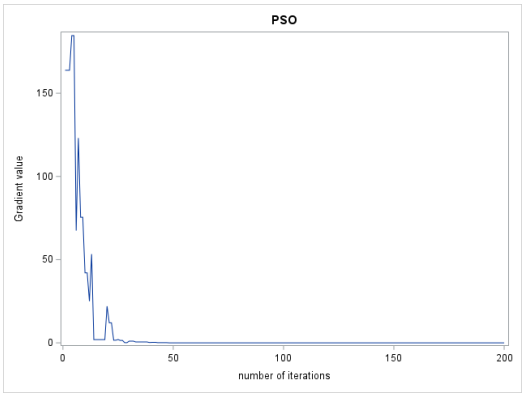

Figure 37: Speed of convergence of PSO method

These graphs show the number of iterations required for the different algorithms to cancel the gradient and find the right solution. It takes 3 iterations to cancel the gradient for the Newton-Raphon method, about 10 for gradient descent, and at least 50 for PSO.
This shows that on simple convex functions of the Mean Square error (or Sum of Square) type, the classsical methods are more efficient than the PSO algorithm, they converge more quickly. This is the reason why the Newton-Raphson method is very popular in econometrics.

5.3.2 Highlighting the difficulties of gradient descent and Newton Raphson methods

The objective here is to show that on more complex functions, such as functions which are not convex, classical methods do not find the optimal solution of the problem. To illustrate this point, we used the following non-convex function:

$$f(x) = \sin(3x) + 0.5x^2 \tag{21}$$

We will see that the effectiveness of these classic methods is dependent on having a good initial point for the iterative process. We have the following case of study:

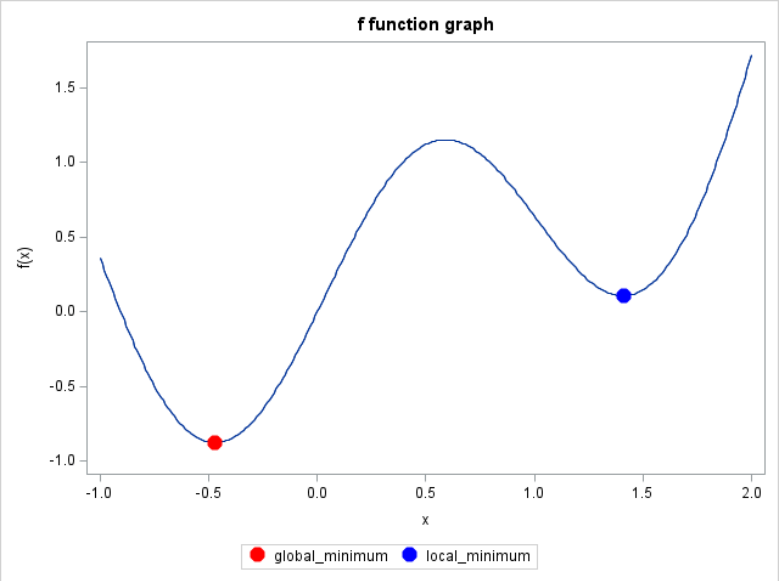

Figure 38: f function

By setting the initial point x to 0, the gradient descent (using NLPFDD[1] with a step of 0.003) and Newton-Raphson algorithms (using NLPNRA[2]) converge to the global optimum which is -0.47. Of course, PSO also converges correctly within a few iterations.

[1]Method that uses finite differences to approximate derivatives

[2]Newton-Raphson algorithm which also uses finite differences, with a learning rate that evolves over time and a technique which allows the optimization to be carried out even in the presence of a Hessian matrix which is not positive definite

However, by setting the initial point at 2, the two classical methods converge towards a local optimum which is the point (1.41, 0.11). This shows the limit of these methods, these algorithms can easily find themselves trapped in local optima. We therefore easily understand the interest of a PSO algorithm. It is much more robust and efficient.

There are variations of these methods created to deal with problems related to non-convex functions. These variants can help avoid getting trapped in local minima and explore the search space more efficiently. They are not discussed here. As explained before, to check if the algorithm converges towards a global optimum, we can change initial parameters or noise its results to see if it always converges towards the same extremum.

The demonstration made previously makes us understand to what extent it will be complicated for classical methods to find the global optimum in the presence of very complex functions, like some of those mentioned in section 4(those which are not convex). For example, we can show that these methods are unable to find the global solution of a Rastrigin function which is a function which admits a lot of local extrema. The following results show that the algorithms remain stuck on a local optimum (second column indicates that the gradient cancels to 0) close to the initialization point without ever being able to leave it.

The SAS System

resuNR				resuDG			
1	10.000018	5	5	1	10.000018	5	5
2	0.0415297	4.9747973	4.9747973	2	1.8335186	4.9699999	4.9699999
3	2.2347E-6	4.9746914	4.9746914	3	0.3171572	4.9755005	4.9755005
4	0	4.9746913	4.9746913	4	0.0557557	4.974549	4.974549
5	0	4.9746913	4.9746913	5	0.0097763	4.9747163	4.9747163

The SAS System

resuNR				resuDG			
1	1.999994	-1	-1	1	1.999994	-1	-1
2	0.0003327	-0.994959	-0.994959	2	0.3801428	-0.994	-0.994
3	5.9755E-8	-0.994959	-0.994959	3	0.0720981	-0.99514	-0.99514
4	5.9755E-8	-0.994959	-0.994959	4	0.0136824	-0.994924	-0.994924
5	5.9755E-8	-0.994959	-0.994959	5	0.0025963	-0.994965	-0.994965

Figure 39: Gradient descent and Newton-Raphson algorithms on Rastrigin function

Moreover, as the PSO algorithm does not use derivatives, it is more able to find good solutions to problems when we have discontinuous functions or constraints. On the first hand, classical methods may persist in looking for points for which the gradient is zero. However, it may be that the optimal solution lies on an edge of the definition set and the global optimum does not cancel the gradient. On the other hand, traditional methods can also persist in saturating constraints. However, the solution does not always saturate all the constraints. The algorithm then seems to saturate constraints randomly (so sometimes this gives the right solution, other times no).

Considering the following minimization program:

$$\begin{cases} \min_{x,y} \ f(x,y) = xy \\ \ y + x \geq 2 \\ -10 \leq x \leq 10 \\ -8 \leq y \leq 14 \end{cases}$$

It is easy to understand that the global solution is $(x, y) = (-10, 14)$. Indeed, we choose extreme values, one negative ($x = -10$), the other positive ($y = 14$) so that their multiplication gives a large negative number ($xy = -140$). This solution checks that the sum of the two variables is greater than 2 ($x + y = 14 - 10 = 4$)
By applying PSO to this problem we realize that the algorithm has no difficulty in finding a solution very close to the global optimum. Conversely, by applying Newton-Raphson under constraint (using NLPNRA call), the algorithm often converges on solutions that saturate the constraints but do not meet the minimization objective. The algorithm only converges to the correct solution when the initial parameters x and y verify $x < y$.

x_initial	y_initial	x_final	y_final
-10	14	-10.00	14.00
0	0	1.00	1.00
-1	1	-10.00	14.00
1	-2	10.00	-8.00
2	2	0.99	1.01
2	3	-10.00	14.00
14	-10	10.00	-8.00

Figure 40: Table showing the results of Newton-Raphson optimization for different initial parameters

Conclusion

To conclude, the PSO algorithm can be fine-tuned using various parameters such as c_1, c_2, w, number of particles or iterations along with constraints and penalties. The algorithm is sensitive to these parameters, and the difficulty lies in finding the right combination of them for the problem in question. The aim is often to maximize exploration in the early iterations, then improve exploitation in the later stages of the algorithm. Indeed, the ultimate goal is always to find the most accurate optimum possible, without becoming trapped in a local optimum.

Practically speaking, our observations have shown that it's essential for the social coefficient c_2 to be greater than 0.5, as values below this threshold render the algorithm inefficient. Among the types of constraints tested, we use the particle reset technique because it appears to be the most effective in reducing the number of iterations required for convergence. This condition involves re-implementing a particle at a random location when it exits the search space. In terms of the number of dimensions in the objective function, the greater this number is, the more CPU time is required and the slower the convergence becomes. In addition, to maximize its performance, the scope of the PSO algorithm must be known and precisely defined. In particular, reducing the search space and the number of constraints may generate significant efficiency gains.

The effectiveness of one algorithm over another often depends on the specific nature of the optimization problem. It has been observed that in the case of standard functions, PSO tends to be less efficient compared to algorithms like Gradient Descent or Newton-Raphson, as it necessitates a greater number of iterations to converge to a solution. Nevertheless, thanks to its stochastic process, PSO has demonstrated its robustness to any type of problem, making it a very interesting algorithm. It excels with complex functions, finding the global optimum in fewer iterations and thus requiring less computational effort.

References

Carlisle, Anthony, Gerry Dozier et al. 2001. "An off-the-shelf PSO." 1:1–6.

Jain, Meetu, Vibha Saihjpal, Narinder Singh, and Satya Bir Singh. 2022. "An overview of variants and advancements of PSO algorithm." *Applied Sciences* 12 (17):8392.

Juneja, Mudita and SK Nagar. 2016. "Particle swarm optimization algorithm and its parameters: A review." :1–5.

Kennedy, James and Russell Eberhart. 1995. "Particle swarm optimization." 4:1942–1948.

Poli, Riccardo. 2007. "An analysis of publications on particle swarm optimization applications." *Essex, UK: Department of Computer Science, University of Essex* .

Shi, Yuhui et al. 2001. "Particle swarm optimization: developments, applications and resources." 1:81–86.

Talukder, Satyobroto. 2011. "Mathematicle modelling and applications of particle swarm optimization." .

Zemzami, Maria, Norelislam Elhami, Abderahman Makhloufi, Mhamed Itmi, and Nabil Hmina. 2016. "Application d'un modèle parallèle de la méthode PSO au problème de transport d'électricité." *OpenScience-ISTE Science Publishing* .

6 Appendix - Our main PSO code – SAS IML

```
            /****************************************
             ************ PSO Algorithm ************
             ****************************************/

proc iml;


/*** defining some test functions used in this paper
   ***/


start Rastrigin(X);
        y=2*10 + (x[,1]**2-10*cos(2*constant("pi")*x
           [,1])) +
        (x[,2]**2 - 10*cos(2*constant("pi")*x[,2]));
        return y;
finish Rastrigin; /* solution ={0,0} */

start Sphere(X);
    y=(X[, 1] - 1) ** 2 + (X[, 2] - 1) ** 2+ (X[, 3] -
       1) ** 2 - 4;
        return y;
finish Sphere;  /* solution ={1,1,1} */

/*** Initialization phase ***/

dim=2; /* Indicate the dimension of the function */
S=100; /* The size S of the swarm */
ub=5; /* Define the upper band */
min_position = j(1, dim, -ub);
max_position = j(1, dim, ub); /* Define the limits of
   the initial position */
X = j(S, dim, .); /* Initialize an array to store the
   generated values */
```

```
/* Loop to generate particles automatically */
do j=1 to S;
        do i = 1 to dim;
        X[j, i] = min_position[i] + randfun(1, "Uniform
           ") * 2 * max_position[i];
        end;
end;

Pb = X;/* Assign each particle its personal best
   position, which is its current position in the first
   step  */
V = j(dim, S, 0)‘;/* Initialize the particle velocity
   vector to zero */

/* Define the global best Gb for the first stage */
Gb_image = rastrigin(X[1,]);
do i = 2 to S;
     if rastrigin(X[i,]) < Gb_image then do;
        Gb_image = rastrigin(X[i,]);
        Gb = X[i,];
    end;
end;

/* Functions */

/* Function that updates the Pb : Personnal Best (from t
    to t+1)*/
start update_Pb(X,Pb,S);
        do i = 1 to S;
                if rastrigin(X[i,]) < rastrigin(Pb[i,])
                   then do;
                Pb[i,] = X[i,];
                end;
        end;
        return(Pb);
finish update_Pb;
```

```
/* Function that updates the Gb : Global Best (from t to
    t+1)*/
start update_Gb(Gb,Pb,S);
        do i = 1 to S;
                if rastrigin(Pb[i,]) < rastrigin(Gb)
                   then do;
                Gb = Pb[i,];
                end;
        end;
        return(Gb);
finish update_Gb;

/* Function that updates the velocity of the particles (
   from t to t+1) */
start update_V(w,c1,c2,X,V,Pb,Gb);
        r1=RAND('UNIForm');
        r2=RAND('UNIForm');
        V_updated = w*V+ c1*r1*(Pb-X) + c2*r2*(Gb-X); /*
            update of the velocity  */
        return(V_updated);
finish update_V;


/* Function that updates the position of X (the
   particles)(from t to t+1) */
start update_X(X,V_updated);
   X_updated = X + V_updated; /* update of the position
      */
   return X_updated;
finish update_X;

/* Penalty function that resets the coordinates of
   particles that exceed the search space */
/* This function is much more complicated in the case of
    problems with constraints */
```

```
start Penalty(X,dim,S,ub);
    do j = 1 to dim;
        do i = 1 to S;
            if abs(X[i,j]) > ub then X[i,j] = (-ub) +
               randfun(1, "Uniform") * 2 * ub;
        end;
    end;
    return(X);
finish;

/*** Final loop with a stopping criterion based on a
   maximum number of iterations***/

nbr_iter_max =500;/* max iteration criterion*/
nbr_iter=1;

do while(nbr_iter < nbr_iter_max);
        /* Initialize dynamic parameters */
        w = 0.9 - nbr_iter*(0.9-0.4)/nbr_iter_max;
        c1 = 2 - nbr_iter*(2-0.1)/nbr_iter_max;
        c2 = 0.1 + nbr_iter*(2-0.1)/nbr_iter_max;
        /* Reinitialization of V, X, Pb, Gb */
        V = update_V(w,c1,c2,X,V,Pb,Gb); /* new V*/
        X = update_X(X,V); /* new X */
        X = Penalty(X,dim,S,ub); /* X penalized */
        Pb = update_Pb(X,Pb,S); /* new Pb */
        Gb = update_Gb(Gb,Pb,S); /* new Gb */

        nbr_iter = nbr_iter + 1; /* count of iteration
           */
end;

y_Gb = rastrigin(Gb); /* Compute the image of the global
    best*/
call NLPFDD(f,gradient,h,"rastrigin",Gb); /* Compute the
    gradient of the global best */
print Gb y_Gb gradient ; /* Display the gb found by the
   algorithm, as well as its image and its gradient by
   the function considered. */
quit;
```