UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Master 1 Econométrie Statistiques

2019/2020

Particle Swarm Optimization Algorithm

Authors:
El Mehdi Agunaou
Stojanović Jovan
Ville Solal

Directed by:
Mr. Philippe de Peretti

Programming in:
SAS/IML

February 2020

ABSTRACT

This work aims to introduce the Particle swarm optimization *(PSO)* method, applied on a wide range of real life complex problems.
Depending on the context, PSO outperforms other optimization methods such as gradient based algorithms and other derivative-free algorithms, especially in the case of some non-linear objective function, as they tend to get trapped in local optimums and fail to find the global one.

The core of this paper is to discuss theories behind PSO, and then test them. We will apply PSO algorithm on highly complex non-linear functions and test them using different approaches on each one of the algorithm's parameters, aiming to build an improved version of PSO suitable for each problem.

The empirical approach developed here, brings some good insights on the several shortcomings of PSO that we need to overcome like premature convergence and low accuracy. The characteristics of a function will have some implications on the parameter's settings. Finally, we demonstrate the power of PSO with the right parameters in optimizing difficult functions, reducing computational cost and the risk of returning local optimum.

Keywords: *Particle swarm optimization*, *Sensitivity analysis,Optimization algorithms*, *Parameter testing*, *Complex functions*, *Non-linear optimization*, *Optimization under constraints*, *SAS*

Contents

1 An introduction to the PSO algorithm: a simulation of social behaviour

The Particle Swarm Optimisation is a technique introduced in 1995 by J. Kennedy, a social psychologist, and R. Eberhart, an electric engineer, permitting the resolution of a great number of optimization problems of continuous linear and more importantly nonlinear functions [4]. This is achieved through computer simulations of moving particles inspired by animals social behaviour (in this case birds or fish).

The initial idea behind the discovery of the algorithm was a simulation of the social behaviour of a bird flock, in what is called an artificial life (or A-life) simulation research process. The second main inspiration came from the idea of fish schooling: just like it seems that fish share information amongst each other to help find food, sharing of information between different entities thus seems to create an evolutionary advantage [4]. Therefore, the two ideas put together stimulated authors [3] to create a simulation (called the Cornfield vector simulation) of birds that find quickly any location when searching for food, helped by each other's knowledge: an optimization idea very useful for finding a solution point in a function. The social paradigm behind the reasoning is clear: helping each other by sharing knowledge is an important element when trying to find the optimal position.

In this paper, we will attempt to review the main functioning of the Particle Swarm Optimisation algorithm, with a focus on the parameters and their impact on different optimization problem case, including optimization of different non-linear functions or optimization under constraints. This will permit us to have a deeper understanding of the algorithm so that it can be applied in the proper way to concrete problems of different background. By testing, we will be able to deduce some conclusions on the parameters, though which should be taken with caution as they may not apply to all cases.

Firstly, we will introduce the PSO algorithm, the different mechanisms behind it by presenting the analytical equation and explaining theoretical impacts of each of the parameter. The second part presents the improved version of the algorithm that we have introduced in SAS and that we will use throughout this paper. The main testing of the different parameters will be presented thirdly, when we will be able to deduce first results on the effectiveness of the algorithm. Fourthly, we will observe the functioning of the algorithm on problems of optimization with constraints. Finally, we will use Gradient and Hessian to post-validate PSO's results

2 PSO algorithm: an overview

2.1 The simplified version

We will introduce a simplified version of the PSO algorithm.
Conceptually, two variables are essential: the **pbest**, or the personal best position an individual particle atteins and the **gbest**, the group best result, attained by the best positioned particle (who shares it's knowledge) and which all individuals tend to attain. We need therefore to know only the current position of the particle x, $\mathbf{present_x}$, to write the simplified version.

Nonetheless, some parameters are to be discussed: **c1** and **c2** are two constants introduced whose best fitting value was still not determined at the time [7]. **c1** adds a weight to the individual best result, while **c2** does it for the global best. Proposed values were $\mathbf{c1} = \mathbf{c2} = 2$ because it seemed that the weights of the social and individual knowledge will thus be equilibrated in average.

The simplified version of the PSO algorithm was therefore based on the following equation, defining the computed velocity of particle x, $\mathbf{V_x}$:

$$V_x = V_x + c_1 * rand() * (pbest_x - present_x) + c_2 * rand() * (gbest - present_x) \quad (1)$$

where rand() is a random uniform variable (in the range [0,1]) weighted by the constant and the distance between the goal of the particle and it's current position.
The position of the particle x will thus be:

$$present_x = present_x + V_x \quad (2)$$

Very quickly, were also developed the matrix notations, so that the algorithm may be applicable to a function of any dimension [5]. We take again the two equations defining the PSO dynamic, adding an indicator **i** to note the i-th particle position in a D-dimensional space:

$$X_i = \begin{pmatrix} x_{i1} & x_{i2} & ... & x_{iD} \end{pmatrix} \quad (3)$$

In the same manner, the local best of the particle i, it's velocity and the global best are changed:

$$pbest_i = \begin{pmatrix} p_{i1} & p_{i2} & ... & p_{iD} \end{pmatrix} \quad (4)$$

$$V_i = \begin{pmatrix} V_{i1} & V_{i2} & ... & V_{iD} \end{pmatrix} \quad (5)$$

$$gbest = \begin{pmatrix} p_{g1} & p_{g2} & ... & p_{gD} \end{pmatrix} \quad (6)$$

The equations to be manipulated which are equivalent to equation (1) and (2) of the initial model are:

$$V_{id} = V_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * rand() * (p_{gd} - x_{id}) \quad (7)$$

The position of the particle x will then be:

$$x_{id} = x_{id} + V_{id} \tag{8}$$

Consequently, the authors have shown that the treatment of the problem in a D-dimensional space does not change the problem.

2.2 A discussion on parameters: inertia weight, maximum velocity and the exploration-exploitation trade-off

Still, a very serious discussion needed to take place on the choice of parameters, which were mostly chosen based on experience and application tests of different optimization problems. The biggest issue with finding the right parameter is to resolve the exploration-exploitation trade-off [7]:

- Exploration is the capability of the algorithm to explore new areas in order to locate a proper optimum, which has then a great chance to be the global one.
- Exploitation is based on precision: a candidate solution is searched in detail so that the potential optimum position is located with precision.

A more general solution needed to be found.

2.2.1 The introduction of a new parameter, the inertia weight

R. Eberhart , author of the 1995 founding article for Particle Swarm Optimisation algorithm and Y. Shi proposed a modified version [5] of the initial model based on tests conducted on a number of functions. The main idea behind is the introduction of a new parameter, inertia weight **w**, which was computed after many simulations. The revised initial equations (7) and (8) after the introduction of the inertia weight are:

$$V_{id} = w * V_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * rand() * (p_{gd} - x_{id}) \tag{9}$$

$$x_{id} = x_{id} + V_{id} \tag{10}$$

The role of the inertia weight is thus to balance the effort of the global and local search. This problem is also known as the exploration-exploitation trade-off: focusing on local search may be problematic if the point is not a global optimum, and emphasizing global research may find the global optimum but at the cost of precision.

- A smaller value of w finds faster the global optimum if it finds it, as particles move quickly. However, the PSO resembles more a local search algorithm in the cases when the global optimum needs to be searched more thoroughly.

- A bigger value of w is the opposite: it searches almost constantly for new areas, as a global research method, the particles "flying" in the search space. The issue is that there are more chances of having a great number of iterations or failing to find the global optimum.

With a balanced value of w, the PSO algorithm has a bigger chance of finding the global optimum within a smaller number of iterations, conclude the authors. The authors [5] conclude after a series of test, that the best value to be attributed to w would be in the interval [0.9,1.2]. However, later work by the same authors have led them to believe that w should usually be decreased linearly from 0.9 to 0.4 during a simulation. Some updated work by Clerc [1] showed that it is best to change it to [0.5 + (Rnd/2.0)], a number randomly varying from 0.5 to 1, with a mean of 0.75.

2.2.2 A discussion on the introduction of maximum velocity

In a 2001 article by Eberhart and Shi [2], maximum velocity on each dimension is discussed. The impact of **Vmax**, the maximum velocity is the following:

- setting it too high would be a problem as particles would overreact and fly over good solutions. (more global "exploration")
- setting it too low may stop the exploration of a wider space, and local optimas may trap the particles. (more local "exploitation")

This is important for the exploration-exploitation trade-off. By experience the authors propose setting Vmax to about 10-20% of the dynamic range for each variable x_{id} on each dimension (the dynamic range being the difference between the largest and smallest value taken by the variable x).

2.2.3 The impact of the two constants c_1 and c_2

The impact of c_1 and c_2 (called "acceleration constants") should also be analyzed more deeply. As we have seen, c_1 is weighting the importance of the particles movement towards its personal best, while c_2 is changing attraction towards the global best. There are three possibilities:

- $c_1 > c_2$: the particles would converge more to their personal best than to the global known best. The risk is that they would spend too much time on exploring a non-optimal region.
- $c_1 < c_2$: the particles are attracted more to the global best position than to their personal best. The risk is that they may too soon neglect their region.
- $c_1 = c_2$: the particles are equally attracted to the personal and global best, thus equilibrating their search. The best option according to experience.

In general, based on experience [2], the following conclusions are made:

- Higher constant values lead to a fiercer movement towards the targeted best position, there is a risk of passing the region if it is set too high.
- Lower values are provoking a slower movement to the target points, especially if far from the target region.

c_1 and c_2 have by experience been set to a value of 2. However, work on the "constriction factor" by Clerc [1] has lead some authors [2] to accept his results: c_1 and c_2 are set by default to 1.49445. This is an analytic result based on mathematical theory. We will focus more on the experimental choice of constants based on tests. In both cases, the best decision still is that both constants should be equal.
In this work, we are going to analyze the impact of the two parameters more closely with applications to functions later.

2.2.4 Defining the number of particles and iterations

When setting up the algorithm, we need to define the swarm population, or number of particles, which will be exploring the search space. Evidently, a great number of particles lead to a better exploration, as most of the space would be covered. The problem is that every iteration would then be more time consuming (and thus more costly) as the complexity of computations rises.
The number of iterations follows the same logic: no doubt that a great number of iterations permits more attempts to find the global optimum and rises the chances we have to do it, but also needs more time which is not always efficient (i.e. if the solution is found quickly and closely enough).
We will focus more on the swarm size and the number of iteration in a test (part 4.3.)

2.3 Theoretical conclusions

In conclusion, was praised from one side the simplicity of the algorithm and from the other its effectiveness when optimization a great number of functions. One of the biggest advantages of the algorithm is that it does not need the objective function to be differentiable (the derivative may not exist), so it includes more functions in it's analysis compared to the more traditional gradient descent optimization method.
Besides, we have seen that the algorithm should be efficient when searching for a global solution. In a unimodal model (e.g. Sphere function), there is only one optimum (minimum or maximum) and the search for local optimums is also a search for the global one. However, in multi-modal problems (e.g. Ackley function), this is not the case: there are multiple local optimums aside from a global solution. The great advantage of the PSO algorithm is that it permits a more concentrated search to find the global optimum compared to other optimization algorithms. [6]
Some applications of the Particle Swarm Optimisation have been [2]:

- Analysis of human tremor (e.g. : Parkinson disease);

- End milling;
- Reactive power and voltage control;

Nonetheless, it could be applied to most problems convertible to an optimization problem.

3 PSO algorithm explanation and details

3.1 Some improvements to the basic algorithm: exiting particles

We used the algorithm as defined before with adding some slight improvements. In linear optimization problems, we observed that the particles had a hard time finding the optimum which is very often at the margin of the search space.
At the same time, a lot of particles were lost by exiting the search space and never getting in again, because of cases with extremely high and gradually increasing velocity.
When a particle leaves the search space, the distance between it and the Group-best and the Personal-best that are inside the search space becomes higher, thus increasing the particle's velocity (according to the equation of velocity: see (9)). When this happens many times to the same particle, the latter's velocity becomes bigger than the length of the search space, therefore it almost never gets back to the search space and frequently flies over it.
Particles leaving the defined range (consisting of an upper and lower bound) were put back in it. There are multiple ways in sending the particles back into the bounded search space : putting the particle at the boundary, this solution works best for linear optimization problems or giving the particles a random position inside the search space.
We also assigned a maximum velocity as 20% of the defined range (20% of the upper bound - lower bound difference), preventing particles from exceeding it, which means that they would not easily fly over the solution or exit the search space. The choice of using 20% of the range was made after testing different percentages on several functions (Ackley, Easom, Sphere) and recording which ones worked better.

Easom function is optimized better (within less iterations) with a Vmax between 10% and 15% as the diameter of its hole is very small and its drop is steep (see Figure 1 bellow), with a Vmax of 20% PSO finds the solution but in more iterations.

Ackley funtion is optimized better with a Vmax of 20% as particles still need to have a sufficient velocity that will allow them to not get trapped in a local optimum (see Figure 2). And finally, Sphere function is optimized better with a Vmax of 20% (see Figure 6).

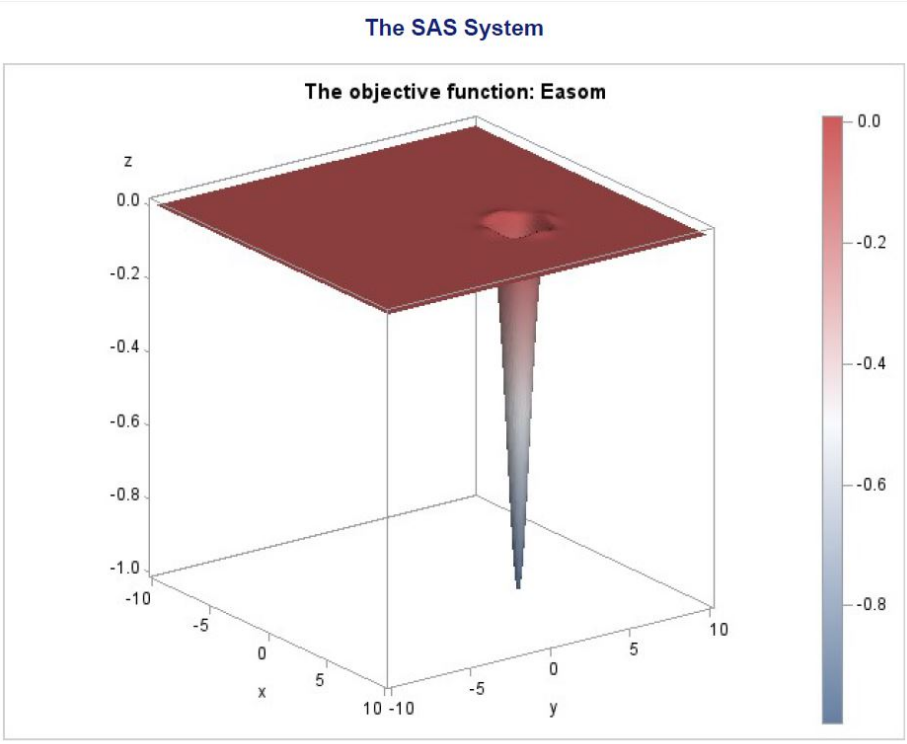
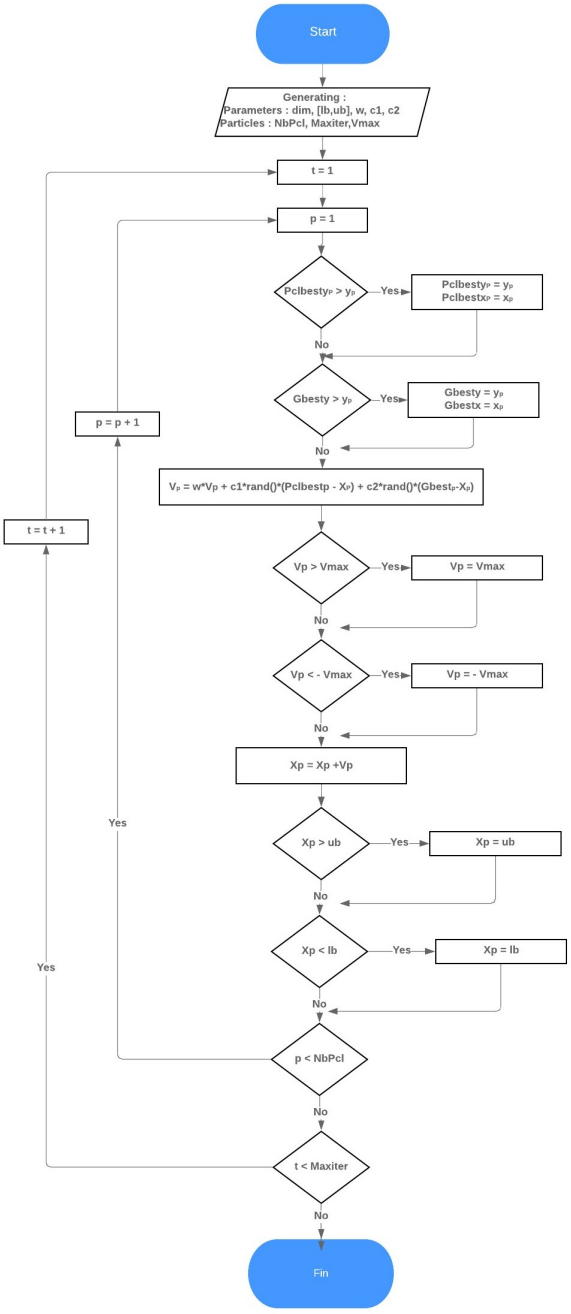


Figure 1: Easom function

After a series of tests on different functions, we perceived that with these changes, PSO performed well on both linear and non-linear functions. All of these improvements have been made in the perspective of solving better all optimization problems with bounded search spaces.

3.2 Algorithm's flowchart

The final algorithm with the improvements mentioned above, is described by the following Flowchart :



3.3 Applying PSO to a test function

In this section we applied PSO's algorithm to find the global minimum of a test function. We have chosen Ackley function because it is a typical non-linear function that has several local optimums, as we can see in the following plot. This would permit us to test whether the algorithm is precise enough to find the global solution in the presence of local ones. The equation of Ackley function with two dimension is:

$$f(x,y) = -20e^{-0,2\sqrt{0,5(x^2+y^2)}} - e^{0,5[\cos(2\pi x)+\cos(2\pi y)]} + e + 20 \qquad (11)$$

Here is a representation of Ackley function in 3-D :

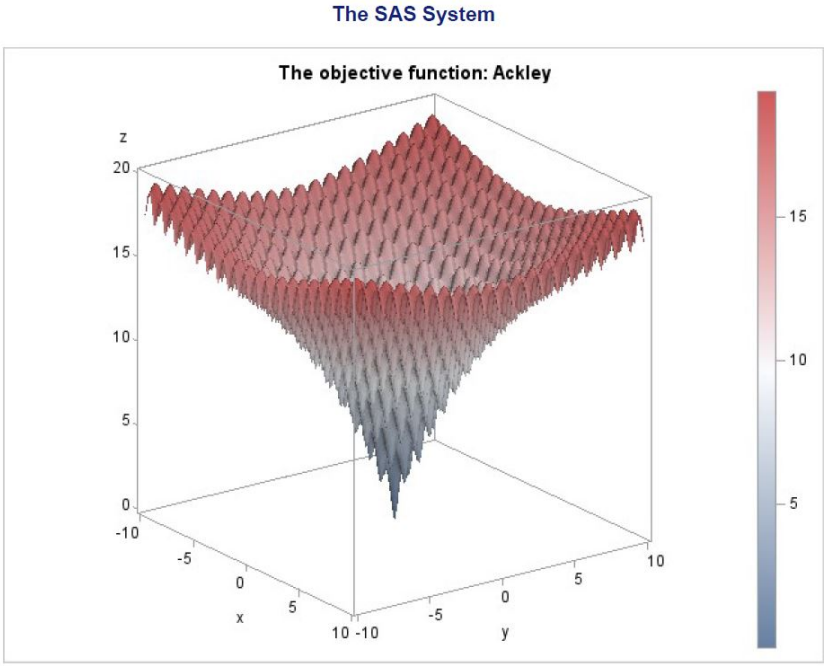


Figure 2: Ackley function

As explained in the flowchart, firstly we must define parameters pertinent to our optimization objective. We used the following parameters for this function:

- Dimensions : 2
- Range : [-5 ; 5]

- Number of Particles : 100
- Number of iterations : 200
- Personal constant (c1) : 2
- Social constant (c2) : 2
- The known global minimum : 0
- inertia weight (w) : [0.2 ; 0.9] (w is gradually descending with every iteration from 0.9 to 0.2)

The inertia weight is changing with a variable (t) representing the iterations number, by a loop with the following equation :

$$w = 0,9 - t * \frac{0,9 - 0,2}{200} \tag{12}$$

The reason we used a changing inertia weight is to balance more exploration and exploitation. We will discuss more on the inertia weight later.

Using our algorithm, the particles converged and gave the following results :

Gbestys	0.000043	9.877E-13	0
3.2210365	0.000043	3.197E-13	0
0.5584335	0.000043	3.197E-13	0
0.5584335	0.000043	3.197E-13	0
0.400437	0.000043	3.197E-13	0
0.400437	0.000043	3.197E-13	0
0.400437	0.000043	1.812E-13	0
0.400437	0.0000202	7.461E-14	0
0.400437	0.0000202	7.461E-14	0
0.400437	0.0000202	7.461E-14	0
0.0857899	0.0000202	6.395E-14	0
0.0857899	0.0000202	3.197E-14	0
0.0857899	6.398E-6	3.197E-14	0
0.0857899	6.398E-6	2.132E-14	0
0.0857899	6.398E-6	1.421E-14	0
0.0857899	6.398E-6	1.421E-14	0
0.0857899	5.4904E-6	1.421E-14	0
0.0857899	5.4904E-6	7.105E-15	0
0.0857899	5.4904E-6	7.105E-15	0
0.0857899	5.4904E-6	3.553E-15	
0.0857899	4.1685E-6	3.553E-15	
0.0857899	4.1685E-6	3.553E-15	
0.0857899	4.1685E-6	0	

Figure 3: Group-best solutions in each iteration

The values in the table above represent some of the 200 group's minimums found by all the particles together (with communication) in each iteration. This test was repeated several times, we noticed that the algorithm always converged towards the global minimum "0". Some "G-best" values (such as 0.000043 here) are repeated several times during the research process, indicating that particles have a risk to be stuck in a local minimum.

The following graph shows the convergence of the group's best solution found in each iteration towards the solution :

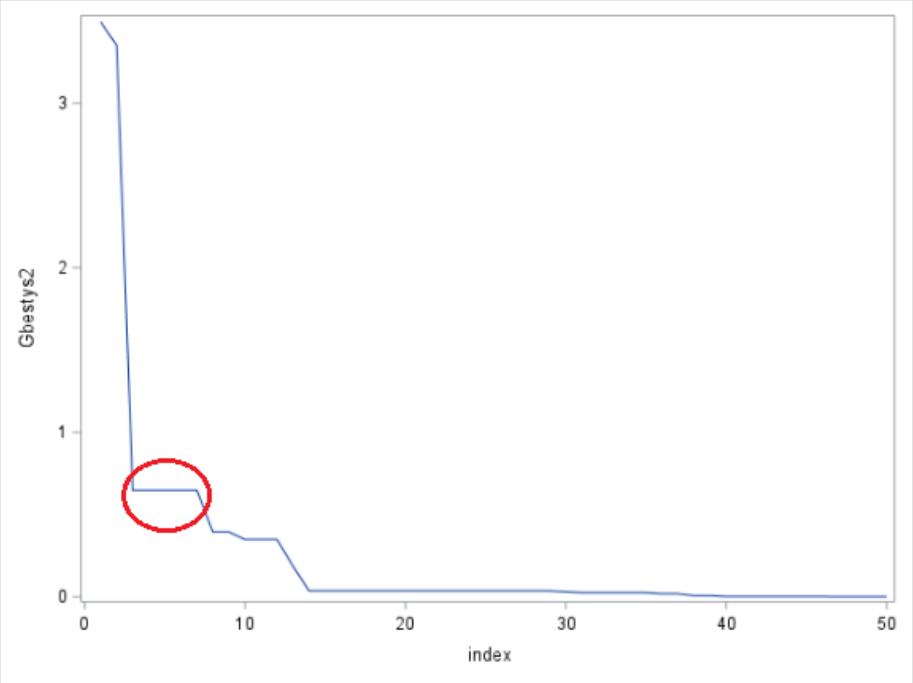


Figure 4: Convergence speed

The horizontal line inside the circle indicates that the particles did not find an improved solution after many iterations. It means that in this test function, the risk of getting stuck in a local minimum is significant.Thus, we have to find an optimal balance between exploration and exploitation of particles such that we diminish the risk of getting trapped in local minimums without degrading the precision of the algorithm.

This can be significantly improved by changing the constants c1 and c2. The usage of 2 as a value for the personal and social constants ($c1 = c2 = 2$) was somehow perceived as arbitrary by the authors developing the algorithm [5], and we think that better values can be assigned to them in order to help improve more the Exploration-Exploitation balance. We will test this possibility.

4 Testing PSO's parameters : Results' sensitivity to parameter change

4.1 The effect of Inertia weight "w"

We discussed before that a bigger value of w makes the particles explore more on the search space (favoring exploration), and a smaller value of w makes particles move by baby-steps (favoring exploitation), as they never fly over the minimum and lead to a faster and more precise convergence.
In most optimization problems we will need both exploration and exploitation, and as mentioned, most recommend an inertia weight "w" between 0.9 and 0.2. We tested all values inside and even some values outside that range [0.2;0.9] and came up with the following results:

- Whenever inertia weight exceeds the defined range mentioned above, particles converge, if they do, way less quickly than when "w" is in the defined range. That is because particles keep having almost the same direction and their velocity becomes less dependent to the social and personal components, thus less stochastic.

Regarding these results we decided to sweep "w" from 0.9 to 0.2 gradually so that in the first iterations particles explore more, and trough iterations they adopt more a behavior of convergence towards the minimum as they become more dependent on the social and personal component and less dependent on their initial velocity.

4.2 Testing the effect of Personal and Social constants (c1,c2) on the results

4.2.1 The effect of c1 and c2 on the convergence and precision rate

In this part we conducted a test on the constants by applying the algorithm many times on test functions and varying c1 and c2 from 0 to 2 by steps of 0.2. To test all combinations, we had to calculate the convergence and precision rate of the 121 combinations. This would then show us what is the best combination of the constants c1 and c2 for the particle swarm optimization.
The functions used to conduct the test were Ackley (multi-modal) and Sphere (uni-modal) function : We used the following parameters for both functions :

- Dimensions : 2
- Range : [-5 ; 5]
- Number of Particles : 100
- Number of iterations : 300
- (c1) (c2) : *The parameters being tested.*

- The known global minimum : 0
- inertia weight (w) : [0.2 ; 0.9] (w is swept decreasingly with every iteration from 0.9 to 0.2)

The test indicators (convergence and precision rates) were calculated in two ways :

First way : using a "Stopping criteria", we programmed the algorithm to stop if the improvement of the Global minimum found within 10 iterations is insignificant ($< 1 \times 10^{-6}$). We then recorded iteration's number where the algorithm stopped in each combination of constants and use it as the color response variable for the heat-map. We used this way on Ackley function.

Second way : applying PSO with 300 iterations in each combination of constants and recorded the number of iterations needed to reach a precision of 10^{-6} (without stopping the optimization program), as for example for Sphere function the algorithm will record the number of iterations needed to find a solution lower than or equal to 10^{-6}, we then multiply that by the group-best solution found in the last iteration (300th iteration) (*so that the closer the value is to zero, the faster and more precise we consider the algorithm is*), and for scaling purposes we take the logarithm of the result as the color response variable for the heat-map. (*So the more negative the value of the logarithm is, the faster and more precise the algorithm is*)

There is a disadvantage to the second way, if the global optimum is equal to "0" and the algorithm is precise enough to return a solution exactly equal to "0", then we cannot calculate the precision rate. In practice, when optimizing Ackley function, this problem may occur as the algorithm returns a solution of 0, and that is why we applied this second way only on the Sphere function where the algorithm returns solutions with a precision between 10^{-20} and 10^{-90} but never exactly "0".

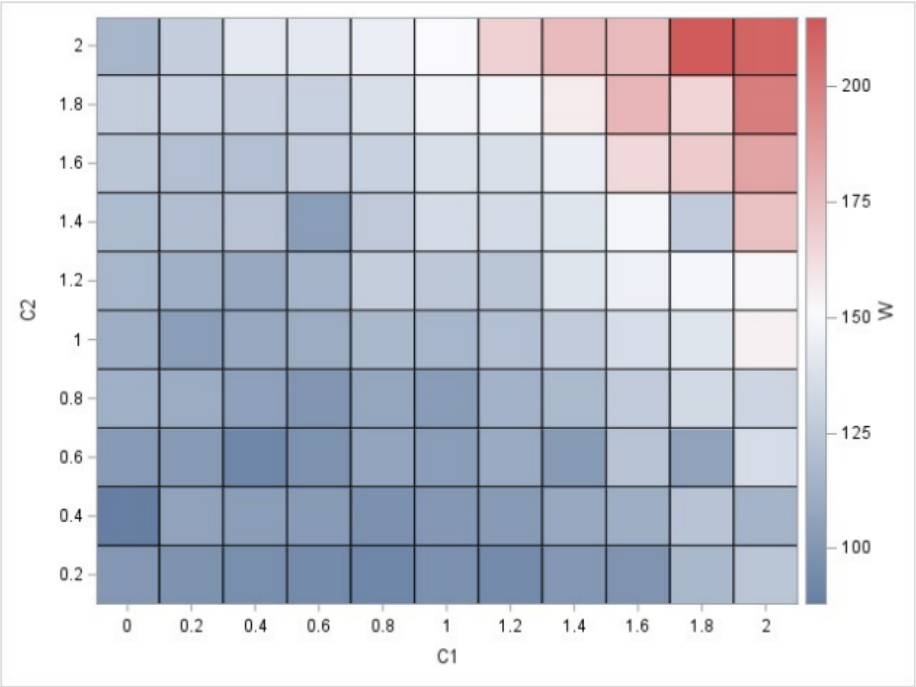
The second indicator takes into account two principal components :

- Speed of convergence : The number of iterations needed to find an acceptable solution (precision of 10^{-6}).
- Precision of the solution : The precision of the Global solution found by the algorithm.

Before showing the heat map, we can discuss what we would expect to happen :

- We would expect the convergence speed to decrease when c1 increases (Particles explore more).
- We would expect the convergence speed to increase when (c2-c1) increases (Particles converge more).

The results produced the following heat-maps :



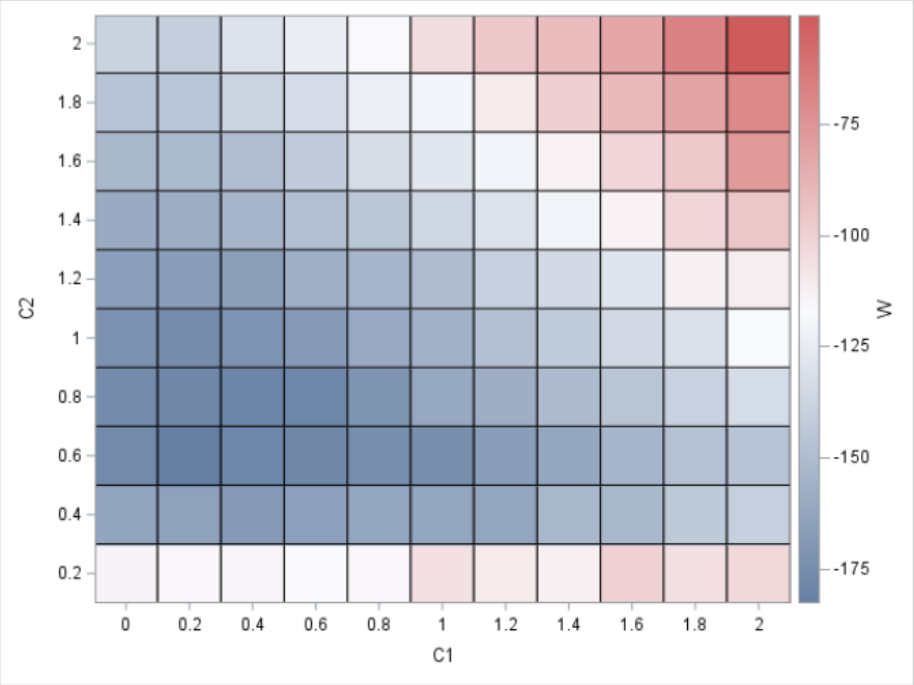
Convergence speed : Ackley function

The indicator "W" is the number of iterations needed to reach the global minimum "0". First thing we notice is that c2 = 0 doesn't appear in the heat map, in all combinations where c2 = 0 particles don't converge, the minimum is not reached. We can also see that, as predicted, when c1 increases "W" increases so the convergence speed decreases (there is a horizontal tendency [2]). But contrary to our prediction, when (c2-c1) increases "W" increases, so the convergence speed decreases, it means that for a fixed c1, the convergence speed decreases when c2 increases (there is a vertical tendency [1]) .These two tendencies create a visual diagonal tendency in the heat-map. This tendency (or relationship) is due to the velocity's big value when the constants increase. Particles get very close to the global optimum, but because of their large velocities, instead of stopping at it directly, they exceed it and then roam around it for some iterations before stopping at it, thus increasing the number of iterations needed to get the global optimum.

Globally, does this tendency mean that it is always better to use small constants?
Well no, it depends. In this case, small constants converged faster than bigger

[1] Vertical tendency : The behavior of c2 for a fixed c1 in the heat-map.
[2] Horizontal tendency : The behavior of c1 for a fixed c2 in the heat-map.

ones also because of the small range of the search space [-5;5], and because the solution is in the center of the search space, so particles didn't need to explore more to find it. Bigger constants give bigger velocities to the particles so they cover more of the search space (convenient for a large range search) and reduce the risk of getting trapped in local optimums, made at the expense of their convergence speed.

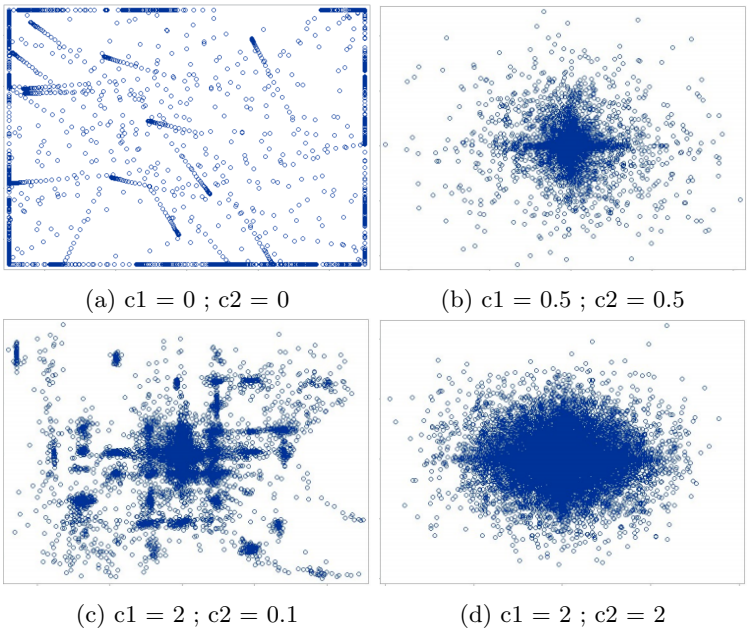


Convergence speed and precision : Sphere function

The factor "W" here indicates convergence speed and precision of the algorithm for each combination of constants. The smaller "W" is, the faster the algorithm converges and more precise it is. Firstly, we notice that this heat-map has the same tendency as the first one, convergence speed and precision are deteriorated with the increase of the constants. Except this time, when $c2 < 0.4$ the algorithm doesn't perform well, its social component is too small, and particles move very marginally towards the group-best. Thus, the algorithm is not strongly converging, and not precise enough : The global minimum found by the combination (c1=0.2,c2=0.6) was $2*10^{-82}$ and the one found by the combination (c1=1,c2=0.2) was $2*10^{-23}$. When particles move by baby-steps the algorithm becomes more precise, but the risk of getting trapped in a local optimum is higher.

4.2.2 Visualizing particles convergence in average and extreme cases

In this section, we launched the program and recorded the particles position in each iteration. Afterwards, we could plot the position history to show the way particles eventually converge in each combination of constants. We can learn a lot from the behaviour of particles shown by these plots.



(a) c1 = 0 ; c2 = 0 (b) c1 = 0.5 ; c2 = 0.5

(c) c1 = 2 ; c2 = 0.1 (d) c1 = 2 ; c2 = 2

Figure 5: Particle's position history in extreme and average cases

The first thing we notice in Figure (a) is that particles don't converge when c2 = c1 = 0 as there is no communication between them, we can see that they keep their first randomly assigned velocity and always move towards the same direction until they exit the search space. When this happens, in our version, they are put back in the boundary which creates a square of particles around the search space.
The essence of a swarm intelligence algorithm is communication, in this case it is represented by the social constant "c2", and without it, the optimization algorithm is ineffective.

We also notice that the particles cover more the search space whenever c1 and c2 are bigger. We can see that when comparing Figures (b) and (d), particles converged to the global minimum of Ackley function "0", but those of Figure (d) covered more the search-space than particles of Figure (b).

Figure (c) confirms what we have concluded with heat-maps, when c2 is very low

(c2<0.4), a lot of particles are stuck in local optimums, very few converged and gave a global minimum with a precision of 10^{-12}, unlike cases where (c2>0.4) where that gave very precise solutions, "0" in this case (Ackley function).

4.3 The effect of swarm size and number of iterations

The swarm size is the number of particles in the communicating swarm. The choice of the swarm size depends on the size of the search space (upper bound - lower bound), a bigger swarm covers more of the search space per iteration and can also reduces the number of iterations needed to find the global optimum.
The maximum number of iterations chosen for an optimization problem is also very important, the algorithm has a significant risk returning a non-satisfying local optimum when the number of iterations is not sufficient, in this case particles don't cover enough of the search space.
Some complex non-linear functions with a lot of local optimums and a wide search space like Ackley (see plot and expression in 3.3.) or the Rastrigin function (see plot and expression in Appendix) may need a bigger swarm size, and most importantly, a large number of iterations.
The risk of getting trapped in a local optimum is thus significant in Ackley or Rastrigin function, we need more iterations than less complex functions in order to give particles more chance to leave their positions around local optimums and find the global optimum.
We will test the impact of the number of particles in relation to the number of iterations for the previous two functions. To do so, we have fixed a maximum iteration number at 1000, varying only the number of particles. The range was set [-10;10], the rest of the parameters (apart from the constants c1 and c2) did not change from those in the 3.3. example. We repeated the search under these conditions for 30 times. Was deduced then the maximum, minimum and average number of iterations necessary for the PSO algorithm to converge towards the global optimum.
The following results were found:

Table 1: PSO - Iteration number and swarm size performance (c1=2, c2=2)

Number of iterations for :	Ackley function			Rastrigin function		
Number of Particles	Average	Minimum	Maximum	Average	Minimum	Maximum
15	924	852	991	765	689	859
30	887	824	940	734	679	771
60	854	777	905	699	631	770

Table 2: PSO - Iteration number and swarm size performance (c1=0.8, c2=0.8)

Number of iterations for :	Ackley function			Rastrigin function		
Number of Particles	Average	Minimum	Maximum	Average	Minimum	Maximum
15	378	339	425	277	254	307
30	363	326	424	261	233	291
60	353	328	381	251	216	275

The test shows that complex non-linear functions (with multiple local optimums) optimization depends mostly on a good choice of the constant parameters, c1 and c2. The only difference between table 1 and 2 is the choice of the value for the two constant parameters, which permitted that the number of iterations to found the optimum was divided by around 3 times. We conclude that there is a huge impact depending on the choice of the constants.
This conclusion was also deduced from the heath maps created when testing the effect of the two constants, but appears more strikingly in this example.
In relation to the swarm size, and independently of the choice of other parameters, we observe that increasing the number of particles reduces the number of iterations needed to find the global optimum, in both functions. We confirm therefore our theoretical conclusions on the swarm size.

4.4 Results and conclusion

The choice of the best fitted values for the constants in an optimization problem is an arbitrage between many factors, namely : the existence of local optimums, the range of the search space and the precision we look for.

Was concluded from the tests above that one of the main parameters in this optimization algorithm is the social component (c2), its value should be at least (c2$\geq$0.4). We also learned from the Vertical and Horizontal tendencies of the heat-maps, that when one constant is greater than the other, the algorithm takes more time to converge than when they are equal. So we need: |c2-c1| $\approx$ 0.

And finally, if we want to reduce the risk for the algorithm to return a local optimum, we should choose bigger values for the constants, although that would decrease the convergence speed. In general, the values of c1 and c2 should be proportionate to the number of local optimums we suspect would exist in the function we wish to optimize, but also they should not exceed a certain value: $c1 = c2 = 2$ appears to be a good combination for the maximum, being big enough to avoid local optimums and not that big to deeply curb the convergence speed.

Therefore, if we want to optimize a function that we know have few or no local optimums, we better choose small values for the constants within the following range : c1 $\in$ [0.2 ; 1] and c2 $\in$ [0.6 ; 1.2]. These ranges were concluded from the heat-map above, they correspond to the dark-blue triangle created in the bottom left corner.
The swarm size has a smaller effect on the convergence speed but certainly reduces it when the number of particles is bigger.

5 Optimization under constraints

5.1 Analytical approach

A very common optimization practice is to put constraints on an optimization problem which reflects the conditions of the problem.
For instance, in economics, the level of certain inputs cannot be negative in the production process. This would mean that we can put -10L of water to produce a certain quantity of beer. This is not possible, the constraint should be that input(water)$\geq$0. This is called an inequality constraint.
We can also have equality constraints. A minimization (equivalent to doing -maximisation) problem formulated in a general form with q inequality and p equality conditions therefore becomes:

$$\begin{aligned} \min \quad & f(x_1, ..., x_n) \\ \text{u.c.} \quad & g_i(x1, ..., x_n) \geq 0, i = 1, 2, ..., q \\ & h_i(x1, ..., x_n) = 0, i = 1, 2, ..., p \end{aligned} \tag{13}$$

To test if our PSO algorithm respects constrained optimization, we are going to use the Sphere function.

5.2 Test: the constrained Sphere function

$$f(x, y) = x^2 + y^2 \tag{14}$$

The representation of the Sphere function is the following:

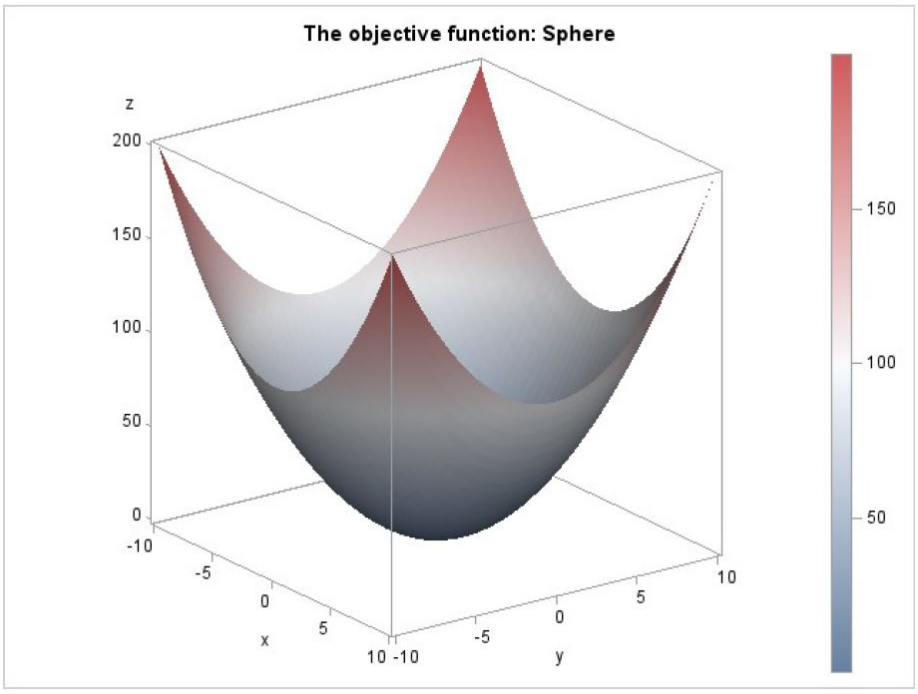


Figure 6: Sphere Function

This is a unimodal function: there is only one solution, global as well as local. The solution is, of course : $(x, y) = (0, 0)$ and the global minimum is 0. The PSO algorithm founds easily the preceding solution, but let us see what will happen if we add the following constraint:

$$x \neq y \tag{15}$$

The optimal solution is no longer feasible, and the algorithm needs to find another one. We used the following parameters for the test:

- Dimensions : 2
- Range : [-10 ; 10]
- Number of Particles : 15, 30 and 60
- Number of iterations : 100
- Personal constant (c1) : 2
- Social constant (c2) : 2

- inertia weight (w) : [0.2 ; 0.9] (w is gradually descending with every iteration from 0.9 to 0.2)

There are several possibilities to include constraints when testing a function. A common solution is to exclude the constrained area such as non-defined. This would have an impact on the particles movement, because all particles leaving the defined space will have to be replaced inside, which can be done by different methods (See section 3.1).
To avoid this problem, one of the most efficient solution is to define a penalty for a result that does not meet the demands of the constraint(s). In our case, when $x \neq y$, we will penalize the result if the particles are located in this non-permitted (though still defined) area. The penalty will be the following:

$$\textbf{if } x \neq y \textbf{ the output will be } f(x, y)$$
$$\textbf{else the output will be } 200$$

The choice of the constant punishment value is arbitrary and depends on the function properties. Here, knowing that the minimum is 0, and the maximum is 200 (the tested range being from -10 to 10), fixing the value at 200 permit us to exclude the constraint from every feasible minimisation solution. The particles, following a global and personal best, shall want to leave this area which is inefficient. We can visualise the situation with the following projection:

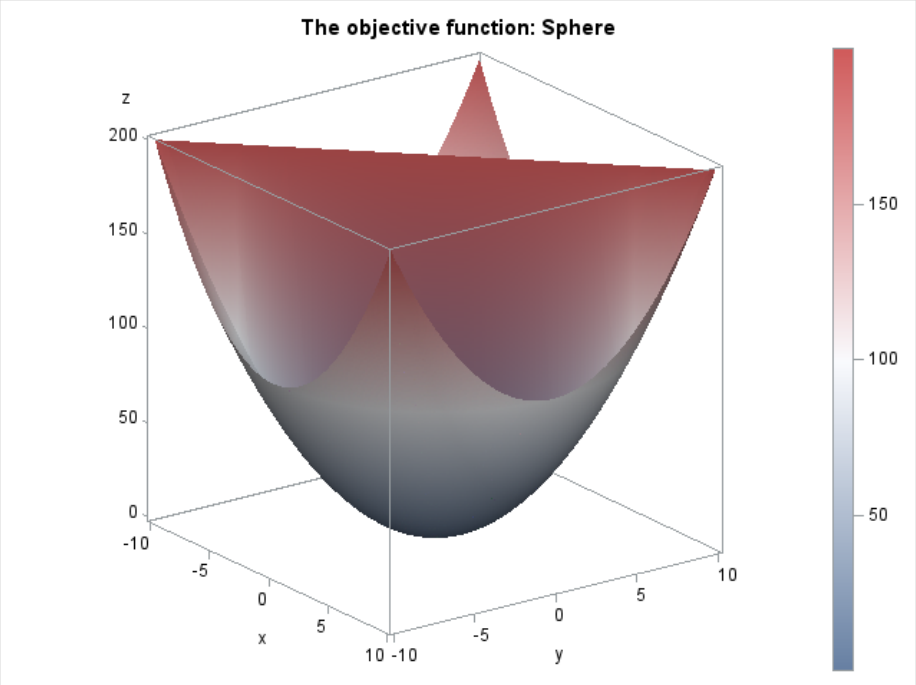


Figure 7: Sphere Function with penalty constraints

In the annex, the code for the previous projection and the way to include the constrained penalty is explicited.

5.3 Results and conclusion

The following results of the test were obtained:

Table 3: Sphere Function under constraint

			(x*,y*) and f(x*,y*)			
Number of Particles	Average		Minimum (or best)		Maximum (or worst)	
15	(0.0134, 0.0518)	0.0165	(0.00105, 0.00009)	1.1172E-6	(-0.166, 0.249)	0.08963
30	(0.0145, 0.0158)	0.00264	(-0.00047, 0.00006)	2.2933E-7	(0.0939, 0.0148)	0.0091
60	(0.0106, 0.0173)	0.00183	(0.00632, -0.00153)	4.22E-5	(0.0682, 0.0688)	0.0094

The number of iterations (100) was not changed independently of the number of particles. Every test was reinitialized 20 times.

From the results of the test, we can deduce that the constraint has been respected and that the algorithm has no problem of finding a solution close to the previous optimum. In average, having more particles lead to having a better precision, though there is a possibility to reach a good position with less particles. This is due to the fact that the search is stochastic, and it depends also on the initial position of the particles (which is a random occurrence).
In conclusion, it seems that PSO algorithm remains very efficient with constrained models when the penalty solution is applied.

6 Using Gradient and Hessian

6.1 Using Gradient and Hessian to post-validate Algorithm's results

The Gradient and Hessian matrix (corresponding to the first and second partial derivatives) of a function can be used to check if the optimum returned by Particle swarm optimization algorithm is correct, and to get information about its nature.

We applied Global PSO on Schwefel function that has a known global minimum "0" at points (x=420.9687 , y=420.9687).

$$f(x,y) = 418.9829 * 2 - x\sin\left(\sqrt{|x|}\right) - y\sin\left(\sqrt{|y|}\right) \tag{16}$$

Here's a 3D representation of Shwefel function :

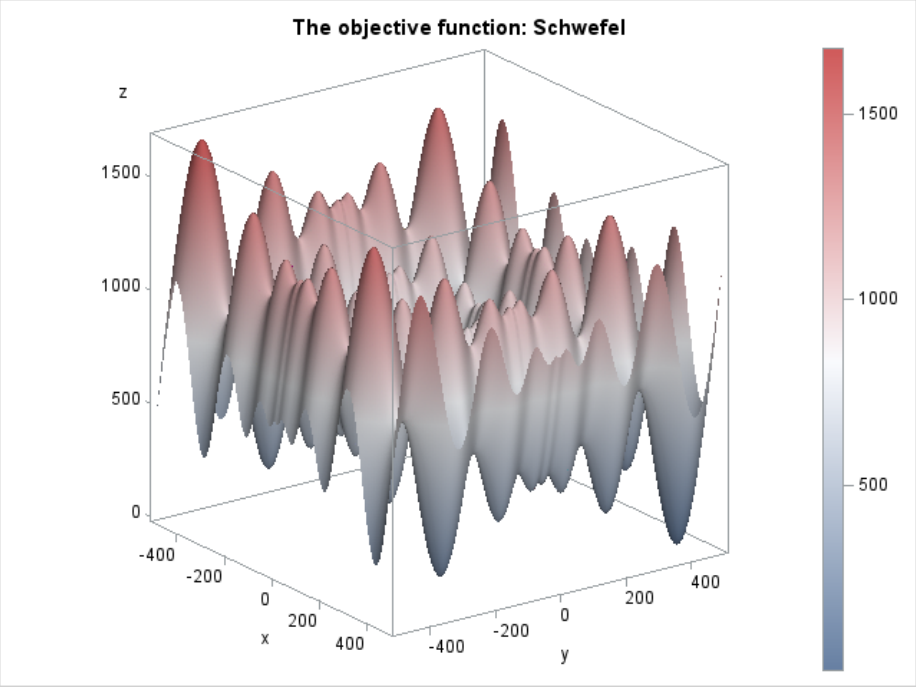


Figure 8: Schwefel Function

We then calculate the first partial derivatives to form the Gradient.

$$\frac{\partial f(x,y)}{\partial x} = -\sin\left(\sqrt{|x|}\right) - \frac{x^2\cos\left(\sqrt{|x|}\right)}{2\,|x|^{\frac{3}{2}}} \tag{17}$$

$$\frac{\partial f(x,y)}{\partial y} = -\sin\left(\sqrt{|y|}\right) - \frac{y^2\cos\left(\sqrt{|y|}\right)}{2\,|y|^{\frac{3}{2}}} \tag{18}$$

$$\mathrm{G} = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

We then calculate the second partial derivatives to form the Hessian.

$$\frac{\partial^2 f(x,y)}{\partial x^2} = \frac{x\,|x|^{\frac{7}{2}}\sin\left(\sqrt{|x|}\right) - 3x^3\,|x|\cos\left(\sqrt{|x|}\right)}{4\,|x|^{\frac{9}{2}}} \tag{19}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} = \frac{y\,|y|^{\frac{7}{2}}\sin\left(\sqrt{|y|}\right) - 3y^3\,|y|\cos\left(\sqrt{|y|}\right)}{4\,|y|^{\frac{9}{2}}} \tag{20}$$

$$\frac{\partial^2 f(x,y)}{\partial x \partial y} = 0 \tag{21}$$

$$\frac{\partial^2 f(x,y)}{\partial x \partial y} = 0 \tag{22}$$

$$\mathrm{H} = \begin{pmatrix} \frac{\partial^2 f(x,y)}{\partial x^2} & \frac{\partial^2 f(x,y)}{\partial x \partial y} \\ \frac{\partial^2 f(x,y)}{\partial x \partial y} & \frac{\partial^2 f(x,y)}{\partial y^2} \end{pmatrix}$$

We then applied the positions of the found solution on the Gradient and Hessian. The algorithm gave the following output :

Le Système SAS

Gbesty
0.0000255

GBestx	
420.96875	420.96875

Gradient	
1.1514E-8	4.9633E-8

Hessienne	
0.2523671	0
0	0.2523671

Figure 9: Gradient and Hessian

First of all, we notice that the Gradients values are very close to 0, therefore the point is an optimum. Secondly, the Hessian's determinant is positive, and its principal diagonal terms are positive, thus we can say that the function has a convex solution. In conclusion, the point : (x=420.9687 , y=420.9687) is a minimum of Schwefel function.

There is another way to post-validate our results, SAS/IML has a subroutine called **NLPFDD**, it uses the double-dogleg method to solve a nonlinear optimization problem. We will use it to post-validate our results again.

Here's NLPFDD's output :

Le Système SAS

Double Dogleg Optimization

Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)

Without Parameter Scaling

Gradient Computed by Finite Differences

Parameter Estimates	2

Optimization Start			
Active Constraints	0	Objective Function	0.0000254551
Max Abs Gradient Element	1.7176449E-6	Radius	1

Optimization Results			
Iterations	0	Function Calls	2
Gradient Calls	2	Active Constraints	0
Objective Function	0.0000254551	Max Abs Gradient Element	1.7176449E-6
Slope of Search Direction	-1.79769E308	Radius	1

ABSGCONV convergence criterion satisfied.

xres	
420.96875	420.96875

Figure 10: The NLPFDD subroutine *SAS/IML subroutine*

We notice that when we used the positions found earlier **(x=420.9687 , y=420.9687)** in the NLPFDD subroutine, the latter didn't need any more iterations to find the global minimum *Iterations = 0*, so the position we gave it at first was the global minimum. Also, and without any surprise, the Objective Function's result is the same we found using PSO.

In this case we have prior knowledge of the function so we know the global optimum, but what happens if we knew nothing about the function?
In the next section we will use PSO, this time to try and check if the found minimum is the global one.

6.2 Using Gradient and Hessian to determine local optimums and saddle points

In this section we used Gradient and Hessian in the optimization algorithm in order to find local minimums and maximums. That would be helpful as a prior understanding of the function to determine the best fitted parameters for the optimization algorithm.

Instead of using PSO algorithm to optimize the objective function (Schwefel) we will use it to minimize the sum of the absolute values of each term of the function's gradient. The function to minimize will then be :

$$f = |\frac{\partial f(x,y)}{\partial x}| + |\frac{\partial f(x,y)}{\partial y}| \tag{23}$$

By taking the absolute values we know that this function reaches a minimum of 0 when

$$\frac{\partial f(x,y)}{\partial x} = 0 \qquad \text{and} \qquad \frac{\partial f(x,y)}{\partial y} = 0.$$

We also know that it will contain as many 0's as there is optimums *(Minimums, Maximums and Saddle points)* in the objective function.

We apply PSO 30 times, then we record the following information in each launch:

- The Gradient value to check if it's null.
- The positions (x,y) that canceled the Gradient.
- The positions value in the function f(x,y).
- The Hessian's determinant, if it is negative then it's a saddle point.
- The Hessian's principal diagonal value signs, if they're both negative, then it's a maximum, if they're both positive then it's a minimum.

Here's SAS output for the results mentioned above :

Results Optimums f(x,y)	Position(x)	Position (y)	f `(x)	f `(y)	nature Nature of the points
513.33829	-302.5249	-25.87742	0	3.331E-16	Minimum
837.9658	-124.8294	124.82936	-3.33E-16	-3.33E-16	saddle point
964.78728	-5.239199	124.82936	-2.22E-16	-3.33E-16	Maximum
422.92821	-5.239199	420.96875	-2.22E-16	1.776E-14	saddle point
809.93754	5.2363065	-25.87742	-0.001168	9.8017E-8	Minimum
415.03761	5.2391993	420.96875	-2.22E-16	1.821E-14	Minimum
916.93284	-203.8143	-124.8294	2.887E-15	-3.33E-16	saddle point
897.65548	5.2391993	-65.54787	-2.22E-16	2.998E-15	saddle point
739.17259	-124.8294	25.877417	-3.33E-16	3.331E-16	saddle point
897.20699	124.82936	65.547865	1.179E-11	2.998E-15	saddle point
758.99876	203.81425	124.82936	2.887E-15	-3.33E-16	saddle point
956.89667	124.82936	5.2391993	-3.33E-16	-2.22E-16	saddle point
837.9658	25.877417	-25.87742	3.331E-16	3.331E-16	saddle point
837.9658	-25.87742	25.877417	-1.44E-13	1.651E-11	saddle point
886.13172	25.877417	25.877417	3.331E-16	3.331E-16	Maximum
750.24786	65.547865	-25.87742	2.998E-15	3.331E-16	Minimum
865.99406	25.877417	-5.239199	3.331E-16	-2.22E-16	Maximum
1043.7543	-203.8143	-5.239199	2.887E-15	-2.22E-16	Maximum
691.00667	-124.8294	-25.87742	-3.33E-16	3.331E-16	Minimum
1083.7181	124.82936	124.82936	-3.33E-16	-3.33E-16	Maximum
632.17728	5.2391993	203.81425	-2.22E-16	2.887E-15	Minimum
865.99406	25.877417	-5.239199	3.331E-16	-2.22E-16	Maximum
845.8564	-5.239199	-5.239199	-2.22E-16	-2.22E-16	Maximum

Figure 11: Results

The particles initial position is randomized in each PSO launch so that the algorithm gives a different result each time we launch it. These results can be useful after filtering them and calculating the number of distinct minimums found by the algorithm. We can filter them by the following code :

```
data Maximums ; set Results ; if Nature = "Maximum";run;
data Minimums ; set Results ; if Nature = "Minimum";run;
proc sort data=Maximums ; by LocaloptimumYs ; run;
proc sort data=Minimums ; by LocaloptimumYs ; run;

proc freq data=Minimums nlevels noprint;
tables LocaloptimumYs /
   out=uniqueMin ;
run;
proc freq data=Maximums nlevels noprint;
tables LocaloptimumYs /
   out=uniqueMax ;
run;
proc means data=uniquemax ;run;
proc means data=uniquemin ;run;
```

We used the procedure Freq to create a table with unique values of the previous data set. We can get the number of distinct minimums or maximums with the procedure means or contents, this information will help us choose best fitted parameters for the Particle swarm optimization algorithm.

Optimizing the absolute value of a function's Gradient with multiple PSO can be helpful in general, and show the positions where the function is cancelled.

Another way can be used to determine local optimums, launching multiple swarms in defined sub-ranges of the search space, it gives the following results :

Results f(x,y)	Pos(x)	Pos(y)	f `(x)	f `(y)	Nature of the points	range	
1675.9316	-420.9687	-420.9687	1.776E-14	1.776E-14	Maximum	-400	-500
236.87669	-302.5249	-302.5249	0	0	Minimum	-300	-400
1241.6522	-203.8143	-203.8143	2.887E-15	2.887E-15	Maximum	-200	-300
592.21345	-124.8294	-124.8294	-3.33E-16	-3.33E-16	Minimum	-100	-200
877.51782	-25.87742	-65.54787	3.331E-16	2.998E-15	saddle point	1	-100
798.41378	25.877417	65.547865	3.331E-16	2.998E-15	saddle point	100	1
760.84444	200	124.82936	-0.964854	-3.33E-16	saddle point	200	100
434.27936	203.81425	203.81425	2.887E-15	2.887E-15	Minimum	300	200
1439.0549	302.52494	302.52494	0	0	Maximum	400	300
0.0000255	420.96875	420.96875	1.776E-14	1.776E-14	Minimum	500	400

Figure 12: Sub-ranges

Using this way we get the global minimum which is in the range [400 ; 500]

7 PSO algorithm on many dimensions

In all our experiments on the Particle swarm optimization algorithm we used 2 dimensional functions $f(x, y)$. We made that choice for simplification purposes and to be able to provide clear plots of the particles and of the functions themselves.

PSO algorithm is not limited by the function's dimension, it can solve optimization problems with as many dimensions as we want.

In this section we will apply PSO on some of the functions we applied it on in the previous sections, but this time with many dimensions.
We will test computational time needed to find the optimum for each number of dimensions.

Gbestys	Dimensions	CPU_Time
1.585E-74	2	1.9
1.305E-55	5	2.15
8.939E-22	10	2.17
2.0082E-8	20	2.75
0.000045	30	3.71
0.0095616	40	3.84
0.2188299	50	6.32

Figure 13: Dimensions, Accuracy and Computation time

We notice that when we increase the number of dimensions, PSO's accuracy decreases, and Computational time increases. Up to 30 dimensions, PSO's accuracy is still acceptable with 200 iterations, maybe we will need more iterations to be accurate at more than 30 dimensions.
All in all, the PSO algorithm gives satisfying results even with a big multidimensional problem.

8 Conclusion

In general, we can conclude from our tests that the Particle swarm optimization algorithm is very efficient in global optimization problems. The algorithm doesn't require the function to be derivable nor being run multiple times to find the solution like in the gradient descent method.Actually, the communication between particles (*Represented by the social component of the formula*) makes it possible to find the global optimum in one launch of PSO.

In **Section 4** the sensitivity analysis of parameter change gave some interesting results :

Firstly, was concluded from constants testing that the social component is the essence of the algorithm, without it the algorithm becomes useless and it should be at least **(c2≥0.4)**.
Secondly, was concluded from the heat-maps and the visualisation of particles that lower constants (c1,c2) give faster convergence, better precision, higher risk of local optimum trap and smaller search space. On the contrary, bigger constants give slower convergence, worse precision, very low risk of local optimum trap and bigger search space.
Thridly, was concluded from testing swarm size and number of iterations that, a bigger swarm makes particles cover more the search space per iteration, thus makes them find the minimum faster (*and in less iterations*).

In **Section 5** we showed that, on constrained optimization problems, PSO algorithm remains very efficient when **the penalty solution** is applied.

In **Section 6** we post validated PSO's results using Gradient and Hessian manually, then using SAS's NLPDD subroutine.

In the last section **Section 7**, we showed that PSO performs well in multiple dimensions optimization problems, it gives accurate solutions up to a certain level of dimensions. Was concluded from our tests that, when we increase the number of dimensions, PSO's accuracy decreases, and computational time increases.

The results above show that, the algorithm's parameters setting is very important, the choice of the best suited parameters should depend on the problem's characteristics, which requires a prior deep knowledge of the objective function to optimize.

References

[1] M. Clerc. "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization". In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. Vol. 3. July 1999, 1951–1957 Vol. 3. DOI: 10.1109/CEC.1999.785513.

[2] Eberhart and Yuhui Shi. "Particle swarm optimization: developments, applications and resources". In: 1 (May 2001), 81–86 vol. 1. DOI: 10.1109/CEC.2001.934374.

[3] Frank Heppner and U. Grenander. "Ubiquity of Chaos, Chapter A Stochastic Nonlinear Model for Coordinated Bird Flocks". In: *The Ubiquity of Chaos, Chapter A Stochastic Nonlinear Model for Coordinated Bird Flocks* (Jan. 1990), pp. 233–238.

[4] J. Kennedy and R. Eberhart. "Particle swarm optimization". In: 4 (Nov. 1995), 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[5] Y. Shi and R. Eberhart. "A modified particle swarm optimizer". In: (May 1998), pp. 69–73. DOI: 10.1109/ICEC.1998.699146.

[6] Satyobroto Talukder. "Mathematical Modelling and Applications of Particle Swarm Optimization". In: *Master's Thesis, Mathematical Modelling and Simulation* 2010:8 (2010), pp. 5, 10–11.

[7] Ioan-Cristian Trelea. "The particle swarm optimization algorithm: convergence analysis and parameter selection". In: *Information Processing Letters* 85.6 (2003), pp. 317–325. DOI: 10.1016/S0020-0190(02)00447-7. URL: https://hal.archives-ouvertes.fr/hal-01313364.

Appendix

Here's our main PSO algorithm version in SAS/IML :

```
proc iml;
/****************** Defining objective funtions ********************/

Start Ackley(x);
y=-20*exp((-0.2*sqrt(0.5*X*X`))) - exp(0.5*(cos(2*constant("pi")*X[,1]) +
        cos(2*constant("pi")*X[,2])))+constant("e")+20 ;
return (y);
finish Ackley;

Start Sphere(x);
y=x*x`;
return (y);
finish Sphere;

start Rosenbrock(X);
y=(100*(X[,2]-(X[,1])**2)**2+(x[,1]-1)**2);
return y;
finish Rosenbrock;

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;

Start Easom(x);
y=-cos(x[,1])*cos(x[,2])*exp(-((x[,1]-constant("pi"))**2 +
        (x[,2]-constant("pi"))**2));
return (y);
finish Easom;

start Bukin(X);
y=100*abs(x[,2]-0.01*x[,1]**2)+0.01*abs(x[,1]+10);
return(y);
finish Bukin;



/************ Defining Algorithm's parameters **************/

dim = 2;
ub = j(1,dim,5);
lb = j(1,dim,-5);
```

```
Nbpcl = 100;
Maxiter=300;
Maxiner= 0.9;
Mininer=0.2;
Vmax = j(1,dim,0.2*(ub[1]-lb[1]));
a= 0.8;
b= 0.8;

/************* Generating Particles (Position & Velocity) *************/
r = j(Nbpcl, dim, 0);
call randgen(r, "UNIFORM");
Xpcl= (ub-lb)#r+lb;

v = j(Nbpcl, dim, 0);
call randgen(v, "UNIFORM");
Velpcl = (2*Vmax)#v-Vmax;


Pclbestx = J(Nbpcl,dim,0);
GBestx = j(1,dim,0);
Pclbesty = j(Nbpcl,1,9**10);
Gbesty = 9**10;

/********************** Main loop ************************/

do t=1 to Maxiter;
        do i=1 to Nbpcl;
                currentX = Xpcl[i,];
                currentY = Ackley(currentX) ;
                if currentY < Pclbesty[i] then do;
                Pclbestx[i,] = currentX; Pclbesty[i] = currentY;
                end;

                if currentY < Gbesty then do;
                Gbestx = currentX; Gbesty = currentY;
                end;

                w = Maxiner - t*((Maxiner-Mininer)/Maxiter);

                Velpcl[i,]=w*Velpcl[i,] + a*v[i,]#(Pclbestx[i,]-Xpcl[i,]) +
                b*v[i,]#(GBestx - Xpcl[i,]);

                do vv=1 to dim;
                        if Velpcl[i,vv] > Vmax[,1] then do;
                        Velpcl[i,vv]= Vmax[,1];
                        end;
```

```
                        if Velpcl[i,vv] < - Vmax[,1] then do;
                        Velpcl[i,vv]= - Vmax[,1];
                        end;
                end;

                Xpcl[i,] = Xpcl[i,] + Velpcl[i,];

                do bb=1 to dim;
                        if Xpcl[i,bb] > ub[bb] then do;
                        Xpcl[i,bb]= ub[,bb];
                        end;
                        if Xpcl[i,bb] < lb then do;
                        Xpcl[i,bb]= lb[,bb];
                        end;
                end;
        end;

call randgen(v, "UNIFORM");
Gbestys= Gbestys // Gbesty;

end;



/****** Showing results ******/

print "The Global Minimum is : ";
print Gbesty;


/****** Showing Group-Minimum evolution ******/

print "The Group Minimum per iteration is : ";
print Gbestys;
```

The following SAS code is used to 3D-plot functions :

```
/*
Inspired by :
https://blogs.sas.com/content/iml/2015/10/12/create-surface-plot-sas.html
*/

/* Plotting the functions in 3D */
proc template;
/* surface plot with continuous color ramp */
define statgraph SurfaceTmplt;

/* dynamic variables */
dynamic _X _Y _Z _Title;
 begingraph;

 /* specify title at run time (optional) */
 entrytitle _Title;
  layout overlay3d;

  /* specify variables at run time */
    surfaceplotparm x=_X y=_Y z=_Z /
       name="surface"
       surfacetype=fill
       colormodel=threecolorramp /* or =twocolorramp */
       colorresponse=_Z;

        /* prior to 9.4m2, use SURFACECOLORGRADIENT= */
    continuouslegend "surface";
  endlayout;
endgraph;
end;
run;

/* Plot Objective function Ackley with no constraints */
%let Step = 0.1;
data A;
do x = -10 to 10 by &Step;
do y = -10 to 10 by &Step;
z = -20*exp((-0.2*sqrt(0.5*(x**2+y**2)))) -
                exp(0.5*(cos(2*constant("pi")*x) +
                cos(2*constant("pi")*y)))+constant("e")+20;
      output;
   end;
end;
run;
```

```
proc sgrender data=A template=SurfaceTmplt;
   dynamic _X='X' _Y='Y' _Z='Z'
        _Title="The objective function: Ackley";
run;

/* Plot Objective function Sphere with a constraint */
%let Step = 0.05;
data A;
do x = -10 to 10 by &Step;
do y = -10 to 10 by &Step;
        if y^=x
                then z = x**2+y**2 ;
                else z = x**2+y**2+200;
      output;
   end;
end;
run;

proc sgrender data=A template=SurfaceTmplt;
   dynamic _X='X' _Y='Y' _Z='Z'
        _Title="The objective function: Sphere";
run;
```

The Rastrigin function (used for testing in part 3.1) has the following expression:

$$f(x, y) = 2 \times 10 + (x^2 - 10 \times cos(2\pi x)) + (y^2 - 10 \times cos(2\pi y)) \tag{24}$$

This function has the following shape in 3D:

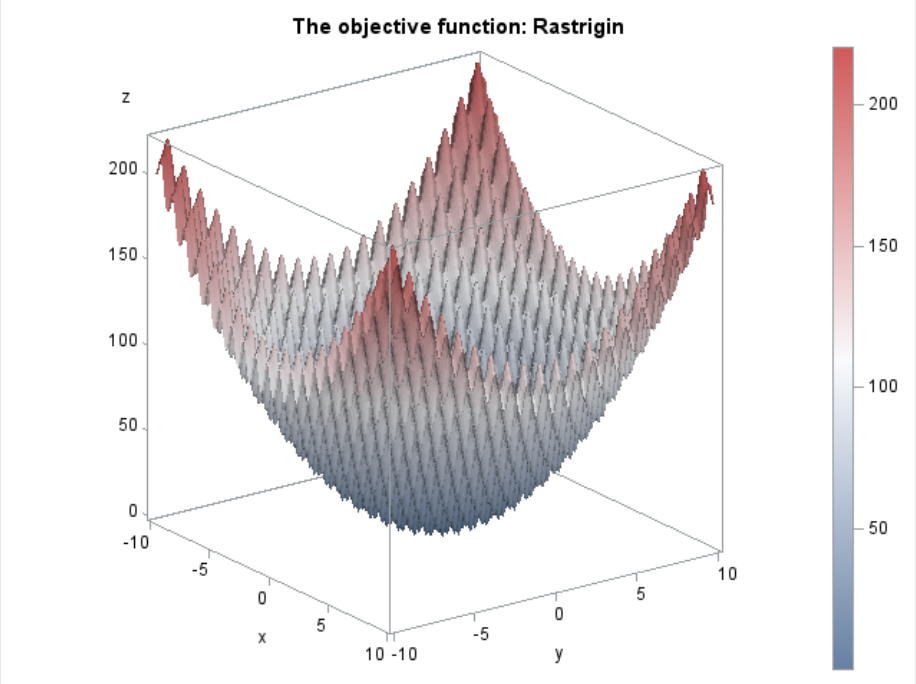


Figure 14: Rastrigin function