Narrowing the searching space increases convergence of PSO

The Particle Swarm Optimization

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Master 1 Econométrie et Statistiques
Janvier 2021

Marie-Lou Baudrin
Serena Gruarin
Théo Lorthios

Directed by Philippe De Peretti

Abstract

This work aims at creating a Particle Swarm Optimization (PSO) algorithm and at modifying some parameters to enhance it. PSO algorithms usually outperforms other optimization methods such as gradient based algorithms, but also other ones, especially in the case of non-linear objective function.
We aimed at improving the basic PSO algorithm as many authors already did, and thus we found new optimal values for the parameters that were not mentioned in other papers that we know of, with the help of Monte Carlo simulations.
There are many way to improve the PSO algorithm and we found a new one. We also improved the initial algorithm by introducing optimization under constraint and we made a Machine Learning Process to found the optimal global solution without approximate the Hessian matrix.
Finally, we applied our enhanced algorithm to five complex functions and managed to find the optima.

Summary

1 Introduction

In the nineties, several studies have been carried out on the social behavior of animals in groups. They showed that some like birds or fish were able to communicate and thus share information among their group.
That ability confers them a survival advantage. It is, inspired by these research, that Kennedy and Eberhart proposed in 1995 the Particle Swarm Optimization (PSO) algorithm.
The PSO algorithm is a metaheuristic and an optimization algorithm. It was mostly inspired by social behavior of flocks of birds. As it was initially created to study flocks of birds searching for food, applied to functions, it allows to find global optimum, either for minimization or maximization problems.
But how were they inspired by these research on animals social behavior to create such an optimization algorithm?
Let's discuss the behavior of a flock of birds: For example, a flock of birds flying over a forest must find a piece of food. The place to land is thus a complex problem as it depends on different issues, it has to minimize the risk of encountering predators and maximize the amount of food so the whole flock can eat. The birds synchronously move until they find the spot to land and then they all land at the same place at the same time. They can only do so by sharing information among their group otherwise they would probably land at different places and time.
Research in the nineties demonstrated that each bird of the flock knows the group's best place to land and balances its individual and social knowledge in order to land optimally. This problem to find the best place to land is an example of an optimization problem. Each bird searches and assesses different spots using many surviving criteria as discussed earlier. The PSO algorithm mimics this behavior.

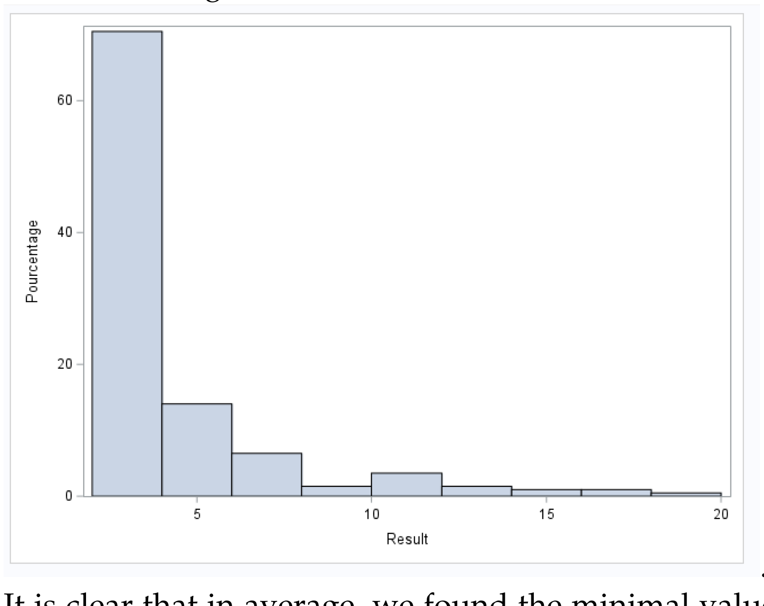
The classic version of the PSO algorithm was proposed in 1995, but since then, many other versions have been proposed, we will discuss some in this paper. This essay aims to study this algorithm and its recent developments, then we will particularly insist on the choices of the parameters and try to improve our PSO algorithm by modifying their values.

2 Extension of the Particle Swarm Optimization

2.1 Monte-Carlo Simulations:

Our initial PSO code works pretty well to find the minimal, or maximal value. However, as we saw, in the equation of the particles next position, the particles move quite randomly because of the two random variables : r_1 and r_2. Then, we can say that there is still hazard in the $gbest$ that we find. It can thus be quite different from the minimal, or maximal, value that we want to find.
It is because of this random situation that we coded a different PSO for which we used Monte Carlo simulations. This runs the PSO a pre-defined number of time and the $gbest$ will simply be the mean value of all the $gbest$ found by the simulations.

For example, we run the first Monte Carlo simulations for the simple function : $f(x,y) = 2 + x^2 + y^2$. We put twenty particles and fifteen iterations for two hundred simulations. That mean that the PSO will run 200 times times 15 iterations. For the other parameters, we choose value that authors recommended, that is $c_1 = c_2 = 2$, and $(\omega_{min}, \omega_{max}) = (0.4, 0.9)$. We found that the average minimal value found is : 2.5.
Here is a histogram of all the $gbest$ found:



.
It is clear that in average, we found the minimal value, 2. But we also see that there is some very different values, like 10, 15 or 20. That means that if we work only with the initial code of PSO, such value can be found because of the random part that we will discuss later. By using the Monte Carlo simulations, we then find a $gbest$ equal to 2.5, which is not too far from the minimal value, unlike results found with the initial code.

As the objective of our work is to find the optimal values for all the parameters, being the one that minimize the number of iterations, we used the Monte Carlo simulations code to find these.

The Monte Carlo simulation code allows the PSO to work a number of times that we set. We can then see the most found values with a histogram. This can help to find optimal values for the parameters by changing them and see if the minimum (for a minimization problem) or maximum (for a maximization one) value is more found with different values of the parameters.

We are going to work in three phases. In the first one, we will describe the possible modifications of the parameters, then we will look at the empirical results obtained with SAS by means of Monte-Carlo simulations. Lastly we will describe the optimal algorithm's convergence framework towards the solution.
Indeed, if we decide to optimize the simple function : $f(x, y) = 2 + x^2 + y^2$. So our optimisation program without constraint would be :

$$Min_{xy} f(x, y) = 2 + x^2 + y^2 \tag{1}$$

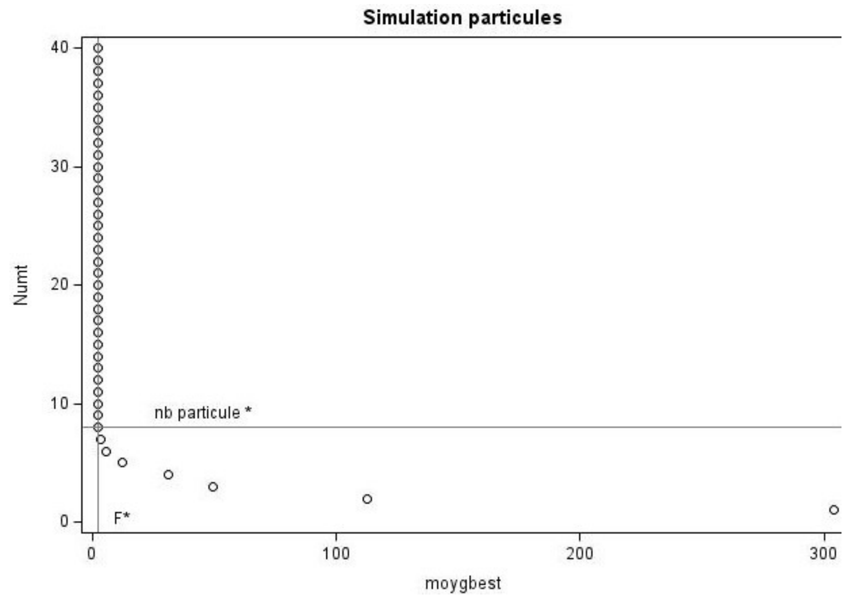
It is very clear that the minimum is 2, for $x^* = y^* = 0$ and the image is : $\arg\min_{x,y} f(x, y) = f(x^*, y^*) = 2$.

We are going to compare our results to this global minimum to comprehend the efficiency of the different parameters.
Among our results, we are going to find $itbest$ which is the average number of iterations at which the $gbest$ is found by the algorithm. For example if, for a precise parameter, we have $itbest$ at forty, it means that our configuration on two hundred simulations gives us forty iterations on average to converge towards the solution. So, these simulations enable us to optimise the speed of convergence but also the accuracy of our solution.
We will obviously also use the value of this $gbest$ to check that the algorithm is properly converging towards the optimal solution. With a coherent $gbest$, we are therefore going to try to find the most optimal configuration to minimize $itbest$. In order to do that, we are going to work on several parameters including :

- The number of iterations
- The values of c_1 and c_2
- The inertia weight
- The number of particles
- The range of the initial particles' positions

The most obvious parameters are the number of iterations and the number of particles. In deed, we have reasons to believe that the higher they are, the more accurate the convergence will be. Although our objective is to reduce these two parameters to a minimum.

In the first place, we are going to ask to our algorithm to replicate 200 simulations of 500 iterations each with the number of particles that goes from one to forty particles. We obtain this graphic:



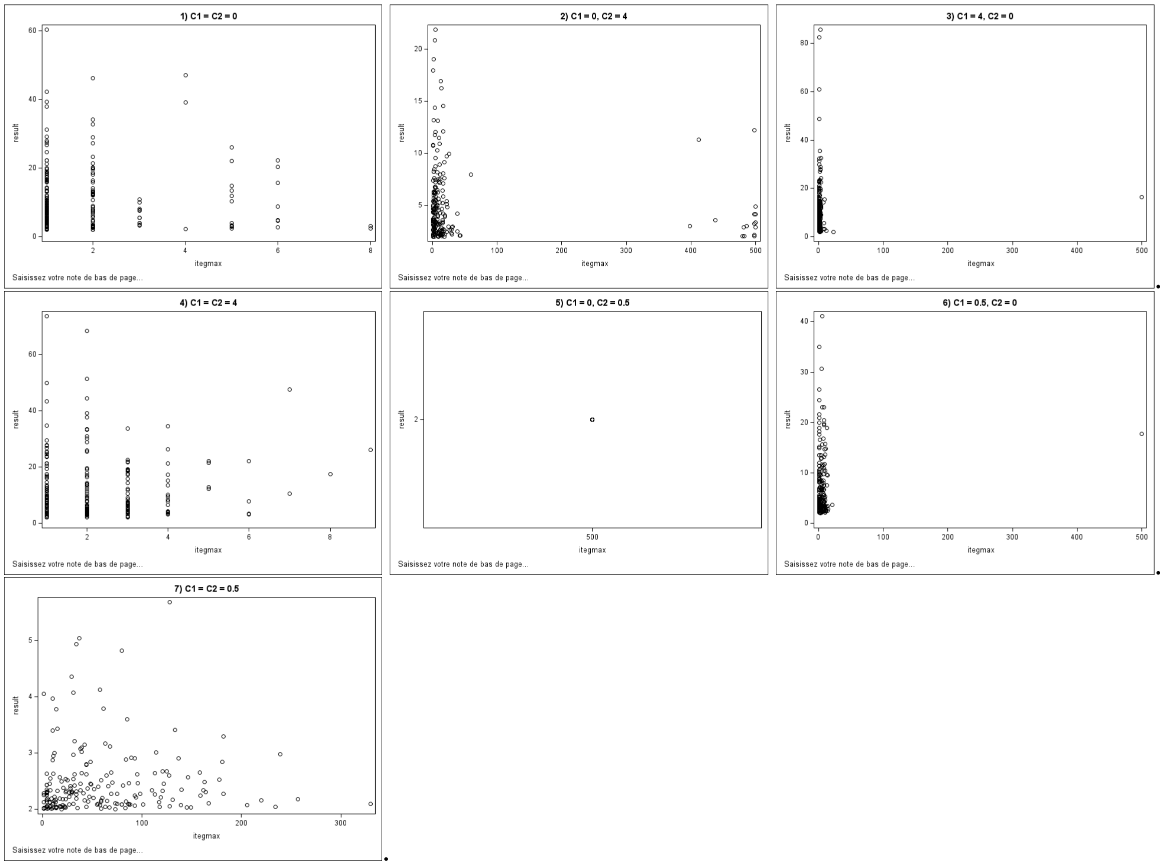
The intuition is right, the number of particles is very important for the convergence of the algorithm.
The optimal number of particles is twenty. It is with this number of particles that we are now going to search for the best possible combination for c_1 and c_2, that is, the combination that find the optimal solution with the lowest number of iterations possible.

We chose seven different combinations :

1. $c_1 = c_2 = 0$
2. $c_1 = 0$ and $c_2 = 4$
3. $c_1 = 4$ and $c_2 = 0$
4. $c_1 = c_2 = 4$
5. $c_1 = 0$ and $c_2 = 0.5$
6. $c_1 = 0.5$ and $c_2 = 0$
7. $c_1 = c_2 = 0.5$

First of all, we know we will not keep combination 1) because in this case the particles move only with the velocity, so the $gbest$ found will never be the optimal one, the one that minimizes our function.
By intuition, we can also predict that combinations 3) and 6) will not be optimal because the degree of dependence, c_2 is set to 0, particles will only follow their personal best position, $pbest$, without acknowledging the group's knowledge. So, we run the Monte Carlo simulations for each combination, with the number of particles set to 20 and the

number of iterations set to 500.

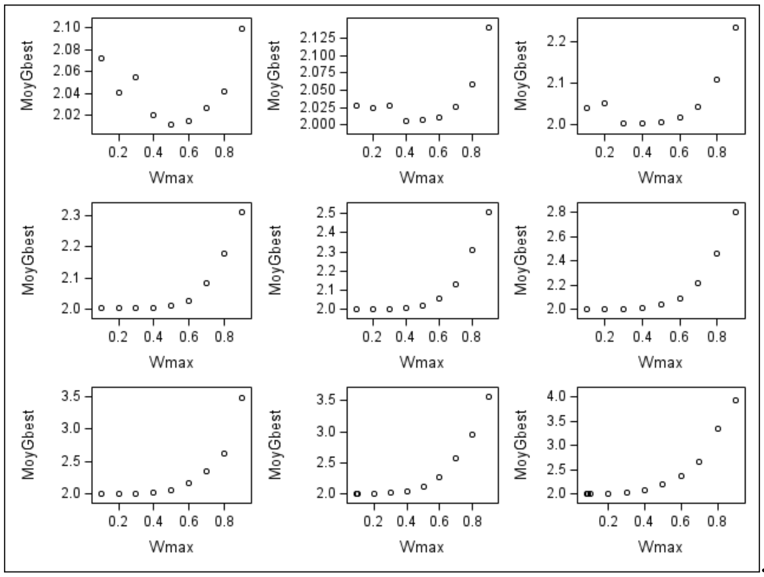


As we predicted, combination 1) is not optimal because all the $gbest$ are just randomly found. For the same reason, we can reject combination 4). However, this was not predictable because this combination of c_1 and c_2 is not so different from the one authors usually suggest being $c_1 = c_2 = 2$.
We also know that combinations 3) and 6) cannot be optimal as particles do not follow the group's knowledge. Finally, it seems that combination 5) is optimal but the number of iteration is way too high. Combination 7) seems to be able to find the minimum value in a smaller number of iterations.

We run the Monte Carlo simulations with these two combinations while changing the number of iterations to see which one converges faster.
It appears that combination 5) is the most optimal one. It is quite intuitive: the "self-centered" part of the particles is null, they only follow the $gbest$. The parameter associated to the $gbest$, c_2, is here quite low thus the particles do not move too fast and they therefore do not surpass the minimum value, like they did in combination 2).
However, this combination is only a first step to find the real optimal combination. By running the Monte Carlo around this value, we find that the optimal combination is :

$c_1 = 0$ and $c_2 = 0.575$.

Now, we are going to search for the optimal inertia weight. As we saw, this weight is an evolutionary function which is composed of two parameters : ω_{min} and ω_{max}. Here again, our problem is to find the optimal combination.
We run the Monte Carlo simulations with the optimal value for the number of particles, for c_1 and c_2, and with ten iterations. We test all the possible combinations within the interval $[0.1, 0.9]$ for each parameters.
Before that, by intuition, we can predict that combinations with ω_{min} higher than ω_{max} are not optimal. We store the average value of all the $gbest$ found, those results give us the following graph.



.

We move ω_{max} with a constant value for ω_{min}. From up to the bottom, and from left to right, we have the following combinations :

- $\omega_{min} = 0.1$
- $\omega_{min} = 0.2$
- $\omega_{min} = 0.3$
- $\omega_{min} = 0.4$
- $\omega_{min} = 0.5$
- $\omega_{min} = 0.6$
- $\omega_{min} = 0.7$
- $\omega_{min} = 0.8$
- $\omega_{min} = 0.9$

.
So, as we predicted, we can see that the three last combinations, that is, when $\omega_{min} = 0.7, 0.8$ and 0.9, the final $gbest$ are quite distant from our minimal value, 2.
Finally, we find smaller values in the first three graphs, when $\omega_{min} = 0.1, 0.2$ and 0.3. We can see that when $\omega_{max} = 0.5$ in the first graph, the final $gbest$ is at a minimum : we found that on average, $gbest$ is equal to 2.011. We can find an even smaller one with the combination $(0.3, 0.4)$, the average $gbest$ is then equal to 2.0030724.
We run the simulations for five iterations and we have the following results :

- when $(\omega_{min}, \omega_{max}) = (0.2, 0.4)$, $gbest \approx 2.1972$
- when $(\omega_{min}, \omega_{max}) = (0.3, 0.4)$, $gbest \approx 2.2398$

We are now going to zoom into this two combinations in order to find the optimal one.
Finally, by running the Monte Carlo simulations, we find it: $(\omega_{min}, \omega_{max})$ =(0.230,0.411).

We now have the optimal values for the parameters, the one allowing the PSO to find the minimum, or maximum in a smaller number of iterations.

To summarise, we have:

- Number of particles, $nbpart = 20$
- The two acceleration factors, $c_1 = 0$ and $c_2 = 0.575$
- The weight of velocity, $\omega_{min} = 0.230$ and $\omega_{max} = 0.411$

However, it is better to work with the Monte Carlo code and not with the first PSO algorithm that we created, that is, we run the PSO for a certain number of simulations, and we take the average value of all the $gbest$ found during the simulations.

2.2 The Zoom Effect:

The most flexible parameter is still the interaction zone of particles. We name this zone, the particles randomization interval. For the first step of PSO we are asked to choose the range $[a, b]$ in which the particles were to be distributed. This initial position turns out to be essential for the rapid and efficient research of global optima of our function. In deed, the bigger the basic interval is, the more chance we will have to tend towards a global optimum and not a local one. However the bigger it is, the longer it takes to converge towards this optimum. We therefore have to compromise between accuracy and speed.
We can have two possible solutions to resolve this compromise:

- The evolution of the parameters with the interval of randomization: We are going to increase the velocity of our particles within a large interval of speed. The limit is obviously the precision bias that we will get with a high speed. We will therefore opt for the second solution which is the zoom effect.

- The zoom effect: It allows, with pre-defined parameters, to find pretty quickly and in a few steps, a solution. We are going too it now.

The zoom effect is an optimization method that allows to localize the optimal interval pretty quickly. The principle is therefore to choose a sufficiently wide basic interval to cover as many values as possible. Then in a second step, we will reduce this interval by looking at the results given by the algorithm. In a minimization, we want our results to decrease with the widening of the interval [1].By repeating this step several times, we arrive at equilibrium, the results will be stable. It is the stationary point: it is the optimum of our function. We can quickly test this zoom effect with our minimization program.

We started from a wide interval, which is $[-150; 150]$ with an average, for 200 PSO, of 24.54 which is pretty far from the solution, two. We therefore take the lower and upper interval to look at the direction of variation of these results. And we get:
For the upper interval : $[-200; 200] => 47.417$
For the lower interval :$[-100; 100] => 13,03$

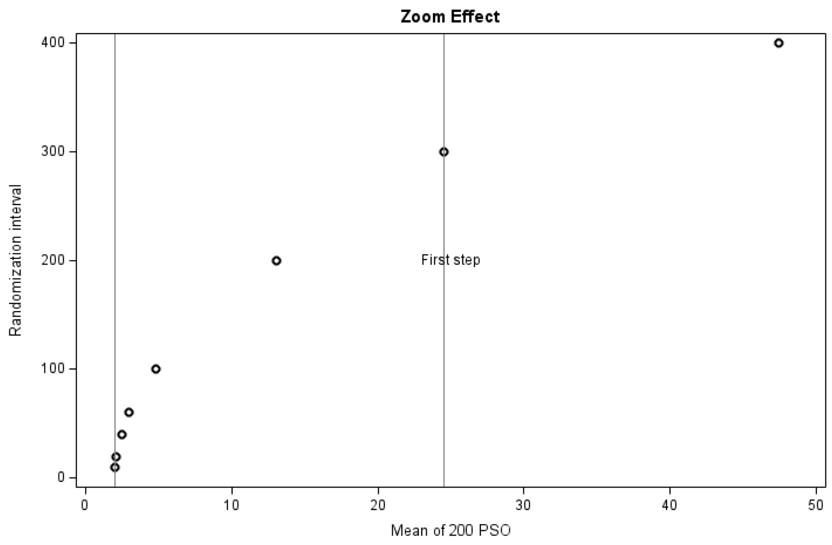
We therefore understand that the more reduced the interval is while converging towards zero, the lower the result. By calculating the extent [2] :

$$b - a \tag{2}$$

[1]While in the case of a maximization, we would want for the value to increase with the narrowing of the interval

[2]of the interval $[a; b]$

Thus we decide to represent on a graph the connection between the results of the PSOs and the different values of the ranges:



We notice that the convergence in two is slower and slower as our interval precisely frames our solution.

To apply the zoom effect on our PSO, we made a machine leaning code. This smart algorithm will learn the evolution of the solution with the decrease of the interval. The goal is to found the global optimum without the Hessian matrix.

This method is efficient for global optima in free optimization. Adding a constraint will allow us to directly find the program's constraint border: the zoom effect will therefore be useless.

2.3 Optimization of programs under constraints :

The addition of a constraint is naturally done in a market economy, for example when we are interested in the maximization of utility under budget constraint. We can also have programs with constraints concerning time, space, technologies etc. Our goal here is therefore to solve this problems as quickly as possible accurately. In order to do this, we reuse the optimal configuration of the PSO parameters and add two while loops inside the basic algorithm.

The theoretical resolution of this kind of optimization program is done in two ways:

- By substituting variables: We manage to express one variable with respect to another in the constraint that we then implement in the objective function: we thus remove the constraint and a variable. This brings us back to a free optimization.
- The other possibility is to use the Lagrangian function which gather the objective function and the multiplied constraint function to the Lagrangian multiplier. By correctly optimizing this new function, we should solve our problem.

The principle of our algorithm differs from its two theoretical processes with one condition: the belonging of the particle to all the possibilities of the constraint. Very simply, our constrained algorithm asks for two pieces of information:
1- The inequality constraint.
2- The constraint frontier.

To calculate the latter, it is sufficient to note in the algorithm the maximum values that our variables can take to saturate the constraint. Consider an example of a constrained optimization program:
Consider an agent who wishes to maximize his utility by consuming two goods (x, y) at the price of 100 and 200 euros respectively. Knowing that this same agent has a budget of 1500 euros and a Cobb-Douglas utility function. We can write our program as follows:

$$\begin{cases} Max_{x} *,_{y} * \quad U(x,y) = 3x^{\frac{1}{4}}y^{\frac{3}{4}} \\ \qquad\qquad s.t \\ 100 \times x + 200 \times y \leq 1500 \end{cases} \tag{3}$$

In our example, we can calculate the constraint frontier in the following way:

$$\begin{cases} 100 \times max_x + 0 \times 200 = 1500 \Rightarrow max_x = 15 \\ 100 \times 0 + max_y \times 200 = 1500 \Rightarrow max_y = 7.5 \end{cases}$$

We will thus take the maximum between max_x and max_y, this value will be the projection limit of our initial positions, so here our particles will be distributed between 0 and 15. In our example, we have a problem with a non-negativity constraint: x and y cannot be negative because they are quantities of goods. But we can easily imagine constraints with negative values, it will then be necessary to put as lower limit projection the minimum value of x and y that saturate the constraint. The advantage of this constrained program is that we do not need to use the zoom method because the area of projection of the initial particle positions is delimited by the boundary of the saturated constraint.

The principle of optimization under constraints is the following:
We will implement a $While$ condition before the calculation of the images of the objective function. The condition will check that the positions (x, y) of each particle respect the constraint. If it is not the case, we will redefine the position of the particles in the constraint with as limits:

- Lower bound $= arg_{Min(x^{st}, y^{st})}$
- Upper bound $= arg_{Max(x^{st}, y^{st})}$

At each iteration, the program checks the correct positioning of the particles and makes them move forward, the particle's memory on the $gbest$ will necessarily attract them towards the optimal point remembered by the particles. We find the result extremely quickly with a very satisfactory accuracy.
For the program resolution, the following optimal solutions are obtained:
$x_{st}^* \approx 3.74$
$y_{st}^* \approx 5.6298$

Knowing that the theoretic solution is:
$x_{st}^* = 3.75$
$y_{st}^* = 5.625$

The result remains very close to the real one with a low number of iterations. The randomness seen at the end of the part about Monte-Carlo simulations also applies in this algorithm. It is therefore important to carry out several simulations to remove the randomness and recover approximations of the expected solutions.

3 An application of the PSO

3.1 PSO applied to complex functions

Now that we have optimal parameters, we can apply our PSO algorithm to resolve complex function.
We will test five functions designated as difficult to globally optimize, especially with the presence of a multitude of local optima. These five functions are defined on a precise interval, outside this interval, they are continuous and monotonic functions, the optima will therefore be $+\infty$ for the maximization and $-\infty$ for the minimization. Here, we will seek the global minimum for each of these functions over the pre-defined interval. In order to check our results, we are not going to use gradient descents or second rank conditions by the miners of the Hessian matrix. Indeed, it would be counterproductive to use these while the very principle of the PSO algorithm is to find the optima with as little information as possible on a function and to avoid heavy derivative calculations or numerical approximations that other algorithms offer and do it more efficiently.
Following the logic of the PSO algorithm, we are going to code a new loop in our algorithm allowing us to represent the zoom effect previously mentioned in our Monte-Carlo simulations. The principle is therefore to progressively reduce the interval to which they belong by x and y in order to compare the results of the different PSOs with each other.
We will therefore teach the machine the different local optima of the function and its response will be the lowest optimum, ie the global optimum. In this way, we cover all the function's minima with a very fast convergence speed due to our optimal configuration of the initial parameters.

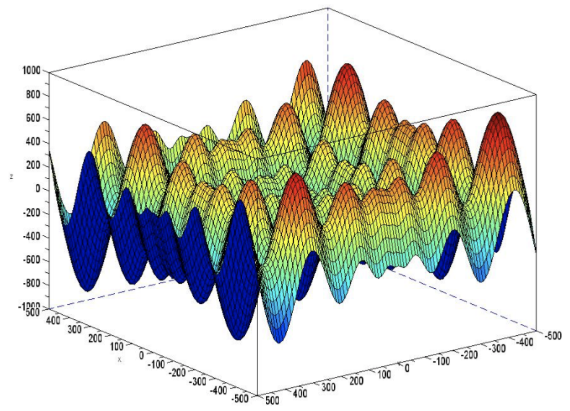
We are therefore going to use the constrained optimization algorithm with a machine learning process. For each function, we will present its theoretical form, our application, its graphical representation, the results of our algorithm and thus, the optimal solution.

1. ***Schwefel function*** : $f(x) = 418.9829d - \sum_{i=1}^{d} x_i \sin\sqrt{|x_i|}$.

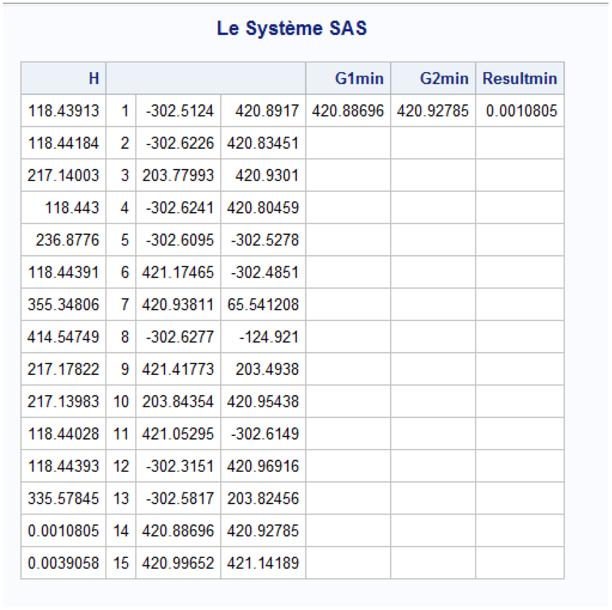
   We have here 2 variables, so the function can be written in our case like : $f(x,y) = 418.9829 \times 2 - x\sin\sqrt{|x|} - y\sin\sqrt{|y|}$

   This equation is defined on :
   $x_i \in [-500; 500]$

   Graphically:



   With our algorithm we find the solution in only twelve iterations.



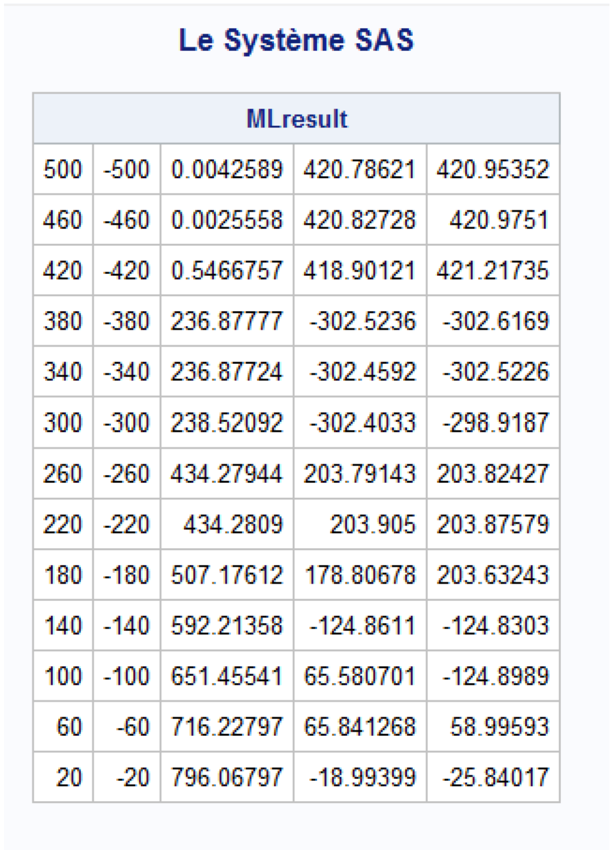
Le Système SAS

H				G1min	G2min	Resultmin
118.43913	1	-302.5124	420.8917	420.88696	420.92785	0.0010805
118.44184	2	-302.6226	420.83451			
217.14003	3	203.77993	420.9301			
118.443	4	-302.6241	420.80459			
236.8776	5	-302.6095	-302.5278			
118.44391	6	421.17465	-302.4851			
355.34806	7	420.93811	65.541208			
414.54749	8	-302.6277	-124.921			
217.17822	9	421.41773	203.4938			
217.13983	10	203.84354	420.95438			
118.44028	11	421.05295	-302.6149			
118.44393	12	-302.3151	420.96916			
335.57845	13	-302.5817	203.82456			
0.0010805	14	420.88696	420.92785			
0.0039058	15	420.99652	421.14189			

   The solution is therefore:

$$f(x^*, y^*) = 0 \quad \text{avec} \quad x^* = y^* = 420.8896 \tag{4}$$

We are going to compare and confirm this result with our smart algorithm, the results are as follows:



Le Système SAS

MLresult

500	-500	0.0042589	420.78621	420.95352
460	-460	0.0025558	420.82728	420.9751
420	-420	0.5466757	418.90121	421.21735
380	-380	236.87777	-302.5236	-302.6169
340	-340	236.87724	-302.4592	-302.5226
300	-300	238.52092	-302.4033	-298.9187
260	-260	434.27944	203.79143	203.82427
220	-220	434.2809	203.905	203.87579
180	-180	507.17612	178.80678	203.63243
140	-140	592.21358	-124.8611	-124.8303
100	-100	651.45541	65.580701	-124.8989
60	-60	716.22797	65.841268	58.99593
20	-20	796.06797	-18.99399	-25.84017

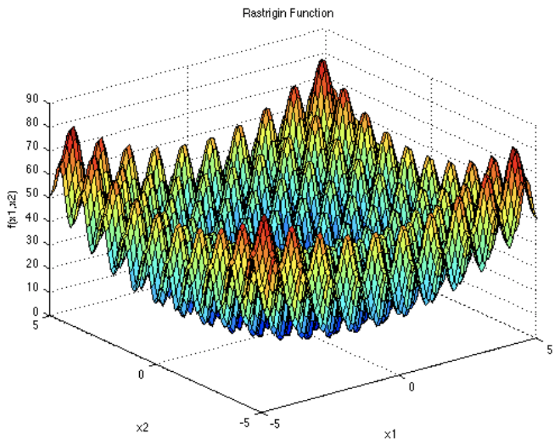
We can note that the solution found using Monte Carlo simulations was correct, we do not find another interval with a lower solution. Our algorithm manages to successfully converge towards the solution with a low number of iterations which is very satisfactory.

2. ***Rastrigin function*** : $f(x) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\cos 2\pi x_i]$

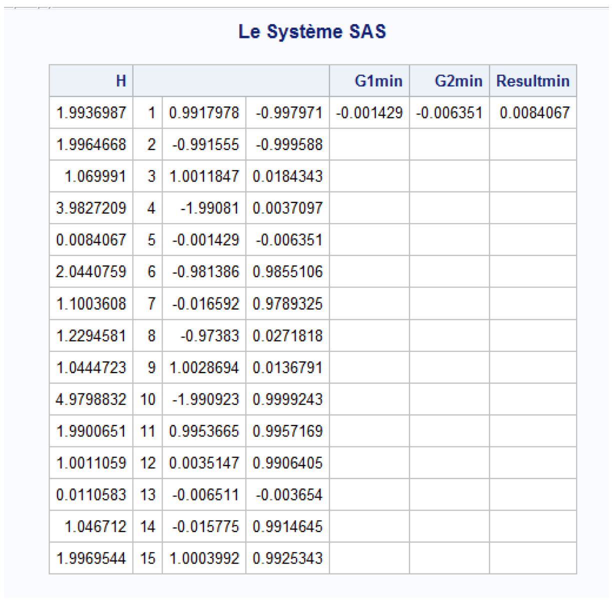
   Here again, with our 2 variables, the function is given by: $f(x,y) = 2 \times 10 + (x^2 - 10 \times \cos 2\pi x) + (y^2 - 10 \times \cos 2\pi y)$

   Just as in the case of the Schwefel function, the values of x and y are set to the interval :
   $x, y \in [-5.12; 5.12]$

   We have the following graphic representation of this function :



   By keeping our optimal parameters, we find these in twelve iterations :



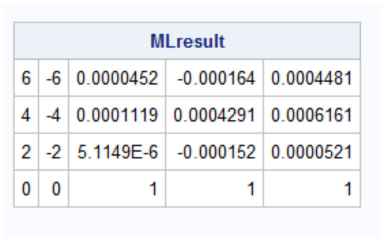
Le Système SAS

H				G1min	G2min	Resultmin
1.9936987	1	0.9917978	-0.997971	-0.001429	-0.006351	0.0084067
1.9964668	2	-0.991555	-0.999588			
1.069991	3	1.0011847	0.0184343			
3.9827209	4	-1.99081	0.0037097			
0.0084067	5	-0.001429	-0.006351			
2.0440759	6	-0.981386	0.9855106			
1.1003608	7	-0.016592	0.9789325			
1.2294581	8	-0.97383	0.0271818			
1.0444723	9	1.0028694	0.0136791			
4.9798832	10	-1.990923	0.9999243			
1.9900651	11	0.9953665	0.9957169			
1.0011059	12	0.0035147	0.9906405			
0.0110583	13	-0.006511	-0.003654			
1.046712	14	-0.015775	0.9914645			
1.9969544	15	1.0003992	0.9925343			

The solution is therefore:

$$f(x^*, y^*) = 0 \quad \text{avec} \quad x^* = y^* = 0 \tag{5}$$

This function is defined on a small interval. Our intelligent algorithm may be useless :



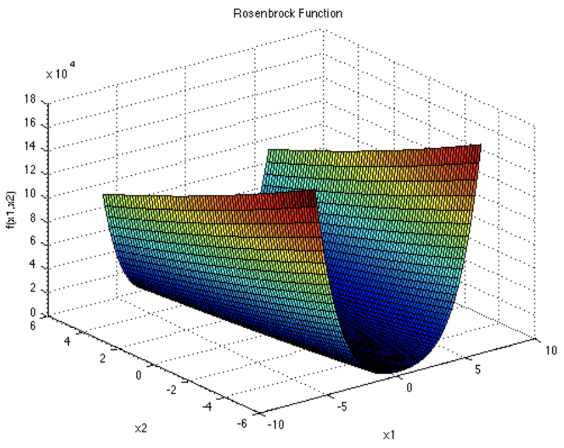
MLresult				
6	-6	0.0000452	-0.000164	0.0004481
4	-4	0.0001119	0.0004291	0.0006161
2	-2	5.1149E-6	-0.000152	0.0000521
0	0	1	1	1

As we can see, our solution seems optimal with $x, y \in [-5, 5]$. The minimum value on the *MLresult* is on the interval [-2,2] but if we go out of the interval for x and y, we can find other solutions. The PSO appears very effective with sinusoidal function.

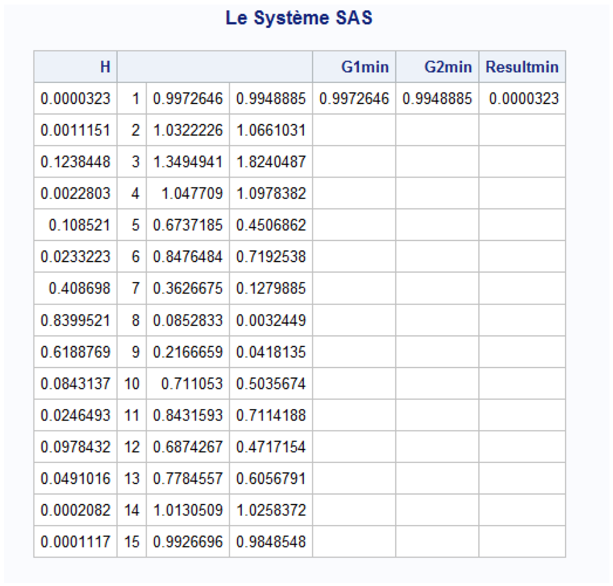
3. ***Rosenbrock function*** :$f(x) = \sum_{i=1}^{d-1}[100(x_{i+1} - x_i^2) + (x_i - 1)^2]$

   With our two variables: $f(x,y) = 100 \times \left((y-x^2)^2\right) \times 2 + (x-1)^2$
   The values of x and y are set to the interval :
   $x, y \in [-10; 10]$

   Graphically :



   Still in twelve iterations, we have the following results on SAS:



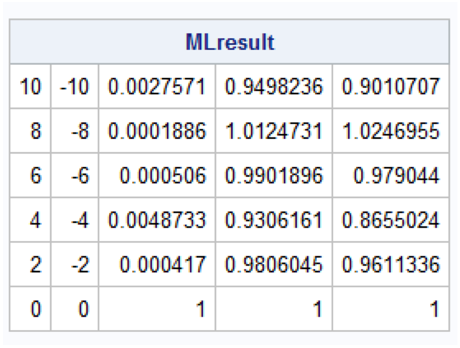
Le Système SAS

H				G1min	G2min	Resultmin
0.0000323	1	0.9972646	0.9948885	0.9972646	0.9948885	0.0000323
0.0011151	2	1.0322226	1.0661031			
0.1238448	3	1.3494941	1.8240487			
0.0022803	4	1.047709	1.0978382			
0.108521	5	0.6737185	0.4506862			
0.0233223	6	0.8476484	0.7192538			
0.408698	7	0.3626675	0.1279885			
0.8399521	8	0.0852833	0.0032449			
0.6188769	9	0.2166659	0.0418135			
0.0843137	10	0.711053	0.5035674			
0.0246493	11	0.8431593	0.7114188			
0.0978432	12	0.6874267	0.4717154			
0.0491016	13	0.7784557	0.6056791			
0.0002082	14	1.0130509	1.0258372			
0.0001117	15	0.9926696	0.9848548			

   The solution seems to be:

$$f(x^*, y^*) = 0 \quad \text{with} \quad x^* = y^* = 1 \tag{6}$$

We have to check this with the second algorithm, we obtain:



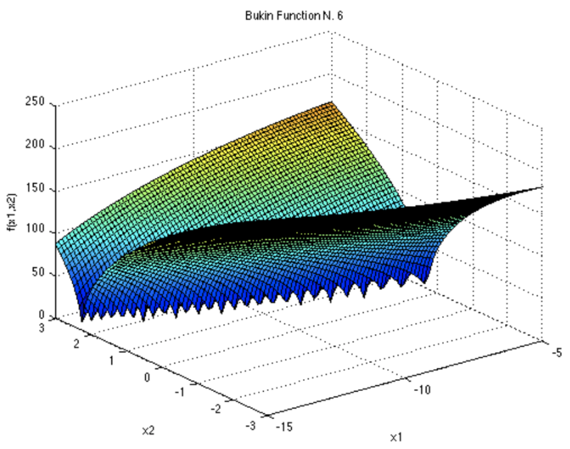
MLresult				
10	-10	0.0027571	0.9498236	0.9010707
8	-8	0.0001886	1.0124731	1.0246955
6	-6	0.000506	0.9901896	0.979044
4	-4	0.0048733	0.9306161	0.8655024
2	-2	0.000417	0.9806045	0.9611336
0	0	1	1	1

This solution is the global minimum on this interval. Our algorithm is actually not in difficulty.

4. ***Bukin function*** : $f(x,y) = 100 \times \sqrt{|y - 0.01 \times x^2|} + 0.01 \times |x+10|$

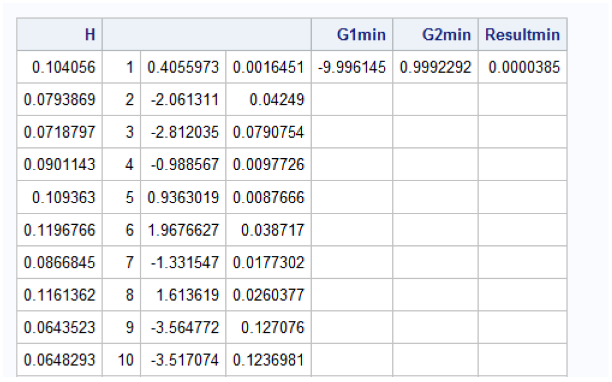
   The values of x and y are set to the interval :
   $x, y \in [-16; 16]$

   Graphical representation of the Burkin function:



   This function seems much more complicated for our particles, we had to increase the number of iterations to a hundred per particle and we also increased their initial velocity by distributing them between $[-2000; 2000]$.

   The final solution seems to be :



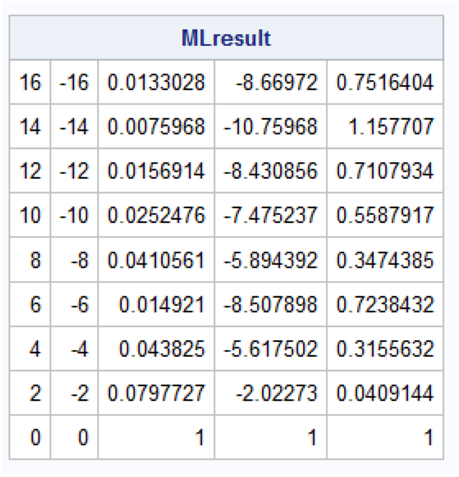
H				G1min	G2min	Resultmin
0.104056	1	0.4055973	0.0016451	-9.996145	0.9992292	0.0000385
0.0793869	2	-2.061311	0.04249			
0.0718797	3	-2.812035	0.0790754			
0.0901143	4	-0.988567	0.0097726			
0.109363	5	0.9363019	0.0087666			
0.1196766	6	1.9676627	0.038717			
0.0866845	7	-1.331547	0.0177302			
0.1161362	8	1.613619	0.0260377			
0.0643523	9	-3.564772	0.127076			
0.0648293	10	-3.517074	0.1236981			

   It may be inferred that the solution would be :

$$f(x^*, y^*) = 0 \quad \text{with} \quad x^* = -10, y^* = 1 \tag{7}$$

This function shows us the limit of our algorithm, we can explain this by the hard convexity of this function. The value on the hessian matrix may be very high in absolute value and positive.

Let’s check the optimality of this solution with our machine learning program.



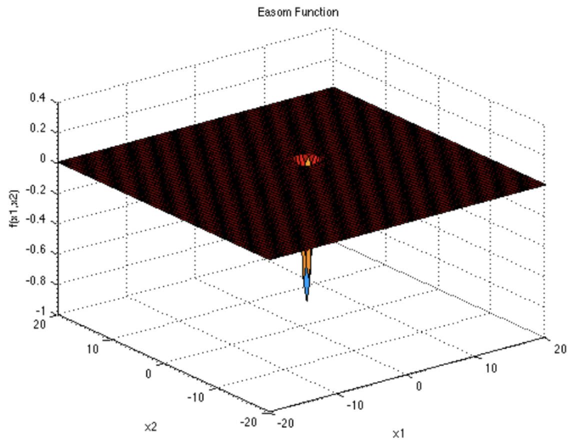
MLresult				
16	-16	0.0133028	-8.66972	0.7516404
14	-14	0.0075968	-10.75968	1.157707
12	-12	0.0156914	-8.430856	0.7107934
10	-10	0.0252476	-7.475237	0.5587917
8	-8	0.0410561	-5.894392	0.3474385
6	-6	0.014921	-8.507898	0.7238432
4	-4	0.043825	-5.617502	0.3155632
2	-2	0.0797727	-2.02273	0.0409144
0	0	1	1	1

Our solution is the lower one. We got our minimal solution.

5. ***Easom function*** $f(x,y) = -\cos x \times \cos y \times \exp\Big(-(x-\pi)^2 + (y-\pi)^2\Big)$

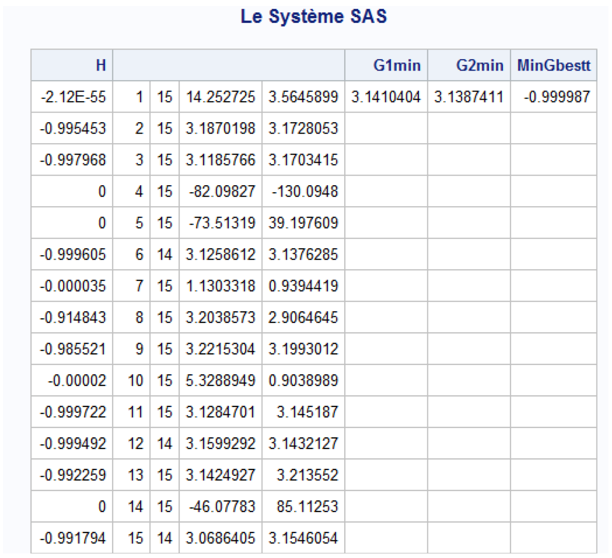
The values of x and y are set to the interval :
$x, y \in [-100; 100]$

Graphically :



$$f(\mathbf{x}) = -\cos(x_1)\cos(x_2)\exp\left(-(x_1-\pi)^2 - (x_2-\pi)^2\right)$$

Our PSO's result with fifteen iterations :



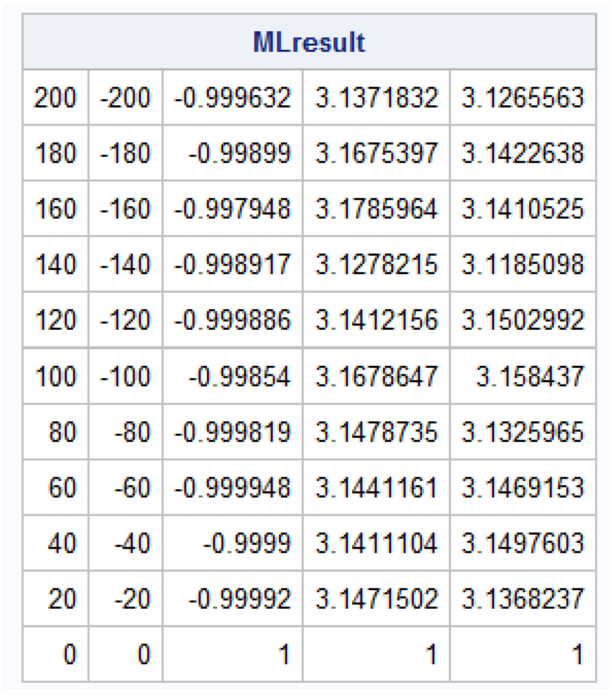
Le Système SAS

H					G1min	G2min	MinGbestt
-2.12E-55	1	15	14.252725	3.5645899	3.1410404	3.1387411	-0.999987
-0.995453	2	15	3.1870198	3.1728053			
-0.997968	3	15	3.1185766	3.1703415			
0	4	15	-82.09827	-130.0948			
0	5	15	-73.51319	39.197609			
-0.999605	6	14	3.1258612	3.1376285			
-0.000035	7	15	1.1303318	0.9394419			
-0.914843	8	15	3.2038573	2.9064645			
-0.985521	9	15	3.2215304	3.1993012			
-0.00002	10	15	5.3288949	0.9038989			
-0.999722	11	15	3.1284701	3.145187			
-0.999492	12	14	3.1599292	3.1432127			
-0.992259	13	15	3.1424927	3.213552			
0	14	15	-46.07783	85.11253			
-0.991794	15	14	3.0686405	3.1546054			

The optimal solution is :

$$f(x^*, y^*) = -1 \quad \text{with} \quad x^* = y^* = \pi \tag{8}$$

Let's check the machine learning output :



MLresult

200	-200	-0.999632	3.1371832	3.1265563
180	-180	-0.99899	3.1675397	3.1422638
160	-160	-0.997948	3.1785964	3.1410525
140	-140	-0.998917	3.1278215	3.1185098
120	-120	-0.999886	3.1412156	3.1502992
100	-100	-0.99854	3.1678647	3.158437
80	-80	-0.999819	3.1478735	3.1325965
60	-60	-0.999948	3.1441161	3.1469153
40	-40	-0.9999	3.1411104	3.1497603
20	-20	-0.99992	3.1471502	3.1368237
0	0	1	1	1

We can conclude that our solution is the global minimum of this function on the interval set. We could understand this with the graph of the function too.

4 Conclusion

The aim of this paper was to understand the principle of the PSO algorithm, to create one and to try and improve it.
In order to do this, we presented the principle of the algorithm, then we coded it on SAS IML. With this code, our objective was simply to understand and create a basic PSO algorithm.
In a first approach, we thus used for the parameters the values recommended by many authors, including the values that the first authors of the PSO, Kennedy, Eberhart and Shi used in their original paper in 1995.
However, we wanted to improve our algorithm by searching for optimal values for those parameters. In order to do that, we used Monte Carlo Simulations.

Thus we coded a Monte Carlo with the help of our original code of PSO algorithm. After a multitude of simulations, we found values for the parameters than can help the algorithm to find the minimum or the maximum in a smaller number of iterations. We found then : an optimal value for the number of particles which is twenty.
The optimal value for the acceleration factors : $c_1 = 0$ and $c_2 = 0.575$. Finaly, we found the weight for the velocity : $\omega_{min} = 0.230$ and $\omega_{max} = 0.411$.

There is also another parameter that can help optimizing the algorithm : the particles' randomization interval. However, contrarily to the previous parameters, this interval cannot be found in the same way, because while the other optimal values are the same whatever the function is , this interval has to change with the function.
We used what we called the Zoom Effect. The principle is to begin with a large interval, and reducing it until we find the optimal one. It may be long, but it is an essential parameter for the speed of convergence of this algorithm.

We also introduced constraints. We started from the initial code and we improved it. We tested this algorithm with a classic microeconomics problem, that is, the maximization of a lifetime utility under a budget constraint and got a nice result.

Finally, we tested the algorithm on five complex functions : the Schwefel function, the Rastrigin function, The Rosenbrock function, The Bukin function and the Easom one. Those functions are too complex to use the initial Monte Carlo code, so we used a new code that integrated the zoom effect. With this new code, we were able to find the minimum solution in a small number of iterations. Furthermore, this new code allowed us to find a $gbest$ that is a global minimum, instead of a local one.

Since the beginning of this algorithm in 1995, many development have been proposed by different authors. We only mentioned a few in this paper, but as we did in our work we can see there are many way to improve this algorithm and there is considerable room for improvement.

5 SAS's Code

5.1 PSO'code:

Ask to the authors :

5.2 Monte-Carlo's code:

Ask to the authors :

5.3 PSO under constraint:

Ask to the authors :

5.4 Zoom effect machine learning's code:

Ask to the authors :

References

[1] Abbas EL DOR. *Perfectionnement des algorithmes d'optimisation par essaim particulaire. Application et segmentation d'images et en électronique.*.Thèse,Archives-ouvertes, 2019.

[2] Bruno Seixas Gomes de Almeidia and Victor Coppo Leite. *Particle Swarm Optimization : a powerful technique for solving engineering problems.*,Swarm Intelligence - Recent Advances, New Perspectives and Applications,December 2019

[3] R.H. Al-Rubayi, M. Kdair Abd, F.M.F. Flaih *New Enhancement on PSO Algorithm for Combined Economic-Emission Load Dispatch Issues*,The International Journal of Intelligent Engineering Systems, October 28, 2019

[4] Russel C. Eberhart and Yuhui Shi *Comparison between genetic algorithms and Particle Swarm Optimization*,LNCS, volume 1447,December 2005

[5] Russel C. Eberhart and Yuhui Shi *Parameter selection in Particle Swarm Optimization*,LNCS, volume 1447,pp 591-600, December 2005

[6] Gregory Watson. *Particle Swarm Optimization in SAS*.UCLA Department of Biostatistics, Los Angeles, CA.

[7] . Maria Zemzami and al. *Electrical power transmission optimization based on a new version of PSO algorithm.* ,ISTE OpenScience, 2016

[8] *Virtual library of simulation experiments ; test functions and datasets* ,https://www.sfu.ca/ ssurjano/

[9] Weng Kee Wong and al. *A modified PSO technique for finding optimal designs for mixture models.* ,PLoS ONE,June 2015