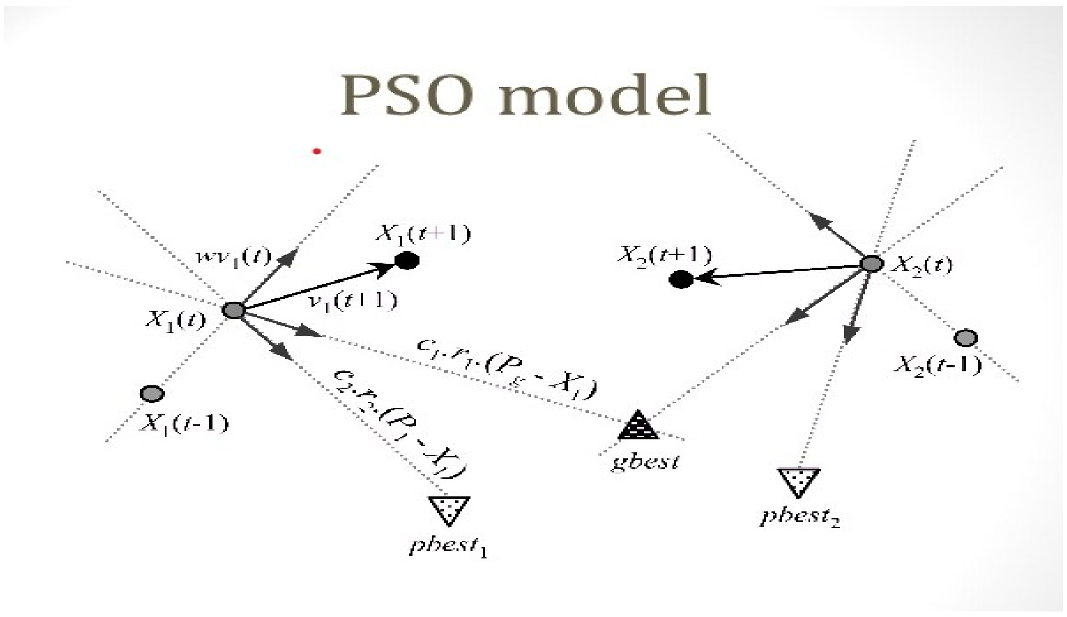
Paris 1 Panthéon-Sorbonne
Ecole d'Economie de la Sorbonne
Master 1: Econométrie Statistiques

THESIS

Particle Swarm Optimization and Different Algorithm Versions:Technical Analysis and Implementation in SAS



BRIDA Kevins
VITHIYANANTHAN Melissa
ZARICINII Xenia

Prefessor : M.DE PERETTI

Paris 2020

Contents

Abstract

Metaheuristics are approached methods that deal with difficult optimization problems. Among these, we will focus on particle swarm optimization, proposed by Kennedy and Eberhart in 1995. This method is based on collaboration between individuals: each particle moves and at each iteration, the closest to the optimum communicates his position to others so they modify their trajectory. The idea is that a group of individuals characterized by limited memory and intelligence can have a complex global organization. PSO has the advantage of being simple to understand, simple to implement, effective on a wide variety of problems without the user having to modify the basic structure of the algorithm. Although recognized as a successful stochastic metaheuristics in solving difficult optimization problems with continuous variables, the particle swarm optimization method has weak points that can discourage some users. One of the major drawbacks of this method is the premature convergence. Since its creation many scientists suggested different version of this algorithm in order to improve algorithm performance. In our work we have chosen a few notable modifications. We reviewed the appearance of inertia weight in the algorithm that controls the particle direction influence on the future relocation. The particle containment was introduced in another version in order to prevent particles from getting out of the search space. Another version is using a constriction factor to control better the divergence. In addition, usage of this method exclude maximum velocity and inertia weight. Then, we reviewed the model with time varying acceleration coefficients $C_1$and C_2 that allow the control of second and third part of the velocity equation on the future particle relocation. Later a new remarkable change was introduced in the algorithm called Fully Informed Particle Swarm where particles take into account not only global best result but also best result found by their neighbors. The last version reviewed is a hybrid strategy using two different algorithms such as differential evolution and particle swarm optimization. This strategy avoids getting into a local optimum and has relatively fast convergence. After presenting these various advancements, we will analyze and compare the performance of the PSO algorithm and its variants: the estimation of the maximum likelihood of the normal distribution.For all the algorithms tested, the convergence speed is practically similar. Asymptotically, the version of the PSO algorithm with inertia weight has the best performance and the version with time varying acceleration coefficients has the worst for the mean argument. In addition, all the PSO versions tested have a feeble performance for the variance argument.

Keywords: PSO Overview, PSO Variants, Optimization, Swarm Intelligence, Continuous Optimization, Metaheuristics

General Introduction

Euler: There is nothing in the world that does not happen without the will to minimize or maximize something. Optimization is a field of mathematics whose importance is growing in terms of its applications. Thus, optimization occurs in many areas such as engineering (design of structures, optimal system design, etc.), operational research (transport problem, economy, stock management, etc.), digital analysis (approximation/resolution of linear, non-linear systems, etc.) but also in statistical analysis for estimating the maximum likelihood of a distribution. An optimization problem is defined by a set of variables, an objective function and a set of constraints. The search space for the best solution(s) is the set of possible solutions of the problem. The optimization problem solving consists in the search for the minimum or maximum of the objective function(s) of the problem posed, while, satisfying a set of defined constraints. In order to solve difficult optimization problems, which includes a large number of sub-optimal solutions, Heuristics and Metaheuristics methods have been developed. These methods identify points that are not strictly optimal, but wich most closely approximates. The majority of algorithms used to solve difficult optimization problems are population-based metaheuristics. Among these methods, we will focus on particle swarm optimization. Particle Swarm Optimization, proposed by Kennedy and Eberhart in 1995, is a biology-based method. This algorithm, inspired by the living world, is based on the principles of self-organization that allow a group of living organisms characterized by limited memory and intelligence to act together in a complex way. In order, to act together in a complex way they follow simple "rules" such as going at the same speed as others, moving in the same direction or staying close to your neighbours. We are talking about collective intelligence. Social behaviour is shaped by a simple mathematical equation that linearly combine three components to guide the particles during their displacement process. The movement of a particle is influenced by: the inertial component (the particle tends to follow its current direction of motion), the cognitive component (the particle tends to move towards the best site through which it has already passed) and the social component (the particle tends to move towards the best site reached by its neighbours). The PSO method is recognized as an effective stochastic metaheuristics in solving difficult optimization problems with continuous variable. This method has the advantage of being simple to understand, simple to implement, effective on a wide variety problems without the user having to modify the basic structure of the algorithm. However, like all metaheuristics, this method has weak points such as premature convergence. PSO has been extensively researched in recent years to improve its performance. The research focuses on the right set of parameters which has to improve the algorithm and many variants of the algorithm have been proposed. For example, Some researchers try to improve it by introducing new parameters like inertia weight, constriction coefficient, particle containment. Other researchers, by introducing time varying acceleration coefficients C_1 and C_2 that allow the control of second and third part of the velocity equation on the future particle relocation. Later a new remarkable change was introduced in the algorithm called Fully Informed Particle Swarm where particles take into account not only global best result but also best result found by their neighbors. In the last few years, more sophisticated particle swarm optimization variants appeared by making a hybrid optimization method using PSO combined with other optimizers. This method completes the advances a little more.

The purpose of this thesis is to give an overview of the PSO algorithm and then propose variants of the PSO algorithm that improve its performance.

The thesis is divided into three sections. The first section is devoted to the presentation of the PSO algorithm : origins, general principle, formalization, configuration of the method and finally we will overview the advantages and the disadvantages of the particle swarm algorithm.

The second section is dedicated to some variants of this algorithm in order to improve the performance. In our thesis we have chosen a few remarkable modifications : the inertia weight, particle containment, the constriction factor x, the

time varying acceleration C_1 and C_2, the fully informed particle swarm and the hybridisation strategy.

The third section is dedicated to the analysis and comparison of the performance of PSO algorithm and its variants from an application: estimation of the maximum likelihood of the normal distribution.

1 Particle Swarm Optimization

1.1 Overview of Particle Swarm Optimization

1.1.1 Origin

Particle Swarm Optimization (PSO) developed by Kennedy (socio-psychologist) and Eberhart (electrical engineer) in 1995, is a stochastic optimization method for non-linear functions. This method is based on a model developed by Craig W.Reynolds in 1986, allowing to simulate the behaviour of a flock of birds in flight. These simulations highlighted the ability of birds in a moving group to maintain an optimal distance between them and to follow a global movement relative to the movements of their neighbours. These simulations also revealed the importance of the competition which opposed birds for food. In fact, birds are looking for food sources that are randomly dispersed in a research space, and when a bird locates a food source, the other birds will try to reproduce it. This social behaviour based on an analysis of the environment and the neighbourhood is then a method of finding optimum by observing the trends of the neighbouring birds. Each bird seeks to optimize its chances by following a trend that it moderates through its own experiences. In this model, each artificial bird or "boid" (bird-oid object), moves randomly following three rules:

- Cohesion: the boids are attracted to the group's average position;

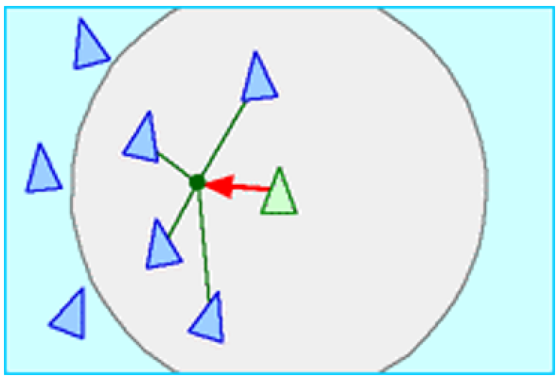


Figure 1: Cohesiokn

- the boids follow the same path as their neighbours;

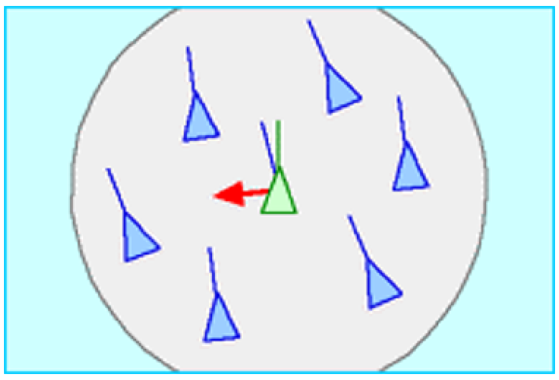


Figure 2: Alignment

- Separation: to avoid collisions, the boids keep a certain distance between them;

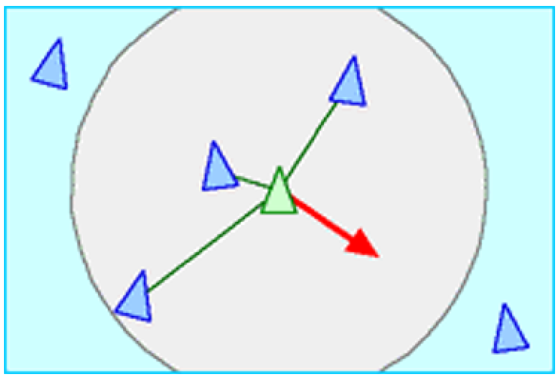


Figure 3: Separation

Kennedy and Eberhart modied these rules and proposed the PSO algorithm. For instance, they included a 'roost' in a simplified Reynolds-like simulation so that:

- Each agent was attracted towards the location of the roost.
- Each agent 'remembered' where it was closer to the roost.
- Each agent shared information with its neighbors about its closest location to the roost.

Birds do not move randomly, they have a goal to achieve. This goal is determined by a function to be optimized or "objective function" which is provided by the user, and which depends on the application concerned.

After, making some modifications of the model developed by Reynolds, Kennedy and Eberhart noticed that the behavior of the group is more similar that of a swarm than of a ock of boids. The new algorithm was called Particle Swarm Optimization

1.1.2 General Principle of PSO

The particle swarm optimization algorithm is based on a set of particles originally randomly and homogeneously arranged, moving in the D-dimension research hyperspace, looking for the global optimum. Each of these particles has a position, that is, its coordinates in the definition set. The location of each particle in the research space represents a potential solution to the optimization problem. The "quality" associated with each of these solutions is quantified by the objective function, optimized little by little according to the positions, more or less optimal. Particles have a neighbourhood, that is, a set of particles that interact directly with the particle. Especially the one has the best position. The particles also have a velocity that allows the particle to move. Thus, at each iteration, each particle changes position. The movement of these particles is managed by simple equation which linearly combines the following three components:

1. A component of inertia: the particle tends to follow its current direction of displacement;
2. A cognitive component: the particle tends to move towards the best site he has already reached;
3. A social component: the particle tends to rely on the experience of its fellow beings and thus to move towards the best site already reached by its neighbours.

From local and empirical optimums, all particles will normally converge towards the global optimal solution.

Two models of PSO have been developed: Global best (Gbest) and Local best (Lbest). These two algorithms differ by their neighbourhood typologies. We will discuss about these two algorithms with more detail in the following sections.

1.1.3 Formalization of Global Best PSO

We present here, the global version of PSO, where all particles of the swarm are considered to be close to particle i. In this model the neighbourhood topology is a star. The position of each particle is influenced by the position of the best known solution throughout the swarm.

A particle i of the swarm in a research space of dimension D is characterized, at the moment t, by:

- x_i :A current position in search space;
- v_i : A current velocity;
- p_i: A personal best position in search space. The personal best position corresponds to the best position that the individual particle has visited since the first time step;
- g:A global best position. The global best position is the best position discovered by any of the particles in the entire swarm.

The particles move taking into account, their current directions of movement, their best position but also the best position of its neighbourhood. In practice, the new velocity of particle i is calculated from the following formula:

$$v_{i,j}^{t+1} = W \cdot v_{i,j}^{t} + C_1 \cdot {r_1}_{i,j}^{t} \left(p_{i,j}^{t} - x_{i,j}^{t}\right) + C_2 \cdot {r_2}_{i,j}^{t} \left(g_j^{t} - x_{i,j}^{t}\right), \quad j \in \{1, 2, \ldots, D\} \tag{1}$$

Where:

- W is a constant, called the inertia weight; Remark: W was introduced historically in a second time;
- $v_{i,j}^{t}$ is the velocity vector of particle i in dimension j at time t ;
- $x_{i,j}^{t}$ is the position vector of particlei in dimension j at time t ;
- $p_{i,j}^{t}$ is the personal best position of particle i in dimension j found from initialization throught time t ;
- g_j^{t} is the global best position of particle i in dimension j found from initialization throught time t ;
- C_1 and $C_2$are positive acceleration constants wich are used to level the contribution of the cognitive and social components respectively ;
- ${r_1}_{i,j}^{t}$ and ${r_2}_{i,j}^{t}$ are random numbers from uniform distribution U(0,1) at time t.

The three components: inertia, cognitive and social are represented in equation [1] by the following terms:

1. $W \cdot v_{i,j}^{t}$ corresponds to the inertial component of the displacement, where the parameter W controls the influence of the direction of displacement on the future displacement;
2. $C_1 \cdot {r_1}_{i,j}^{t} \left(p_{i,j}^{t} - x_{i,j}^{t}\right)$ corresponds to the cognitive component of displacement, where the parameter c1 controls the cognitive behavior of the particle;
3. $C_2 \cdot {r_2}_{i,j}^{t} \cdot \left(g_j^{t} - x_{i,j}^{t}\right)$ corresponds to the social component of displacement, where parameter c2 controls the social aptitude of the particle.

We can then determine the following position of the particle thanks to the speed we have just calculated:

$$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1}, \quad j \in \{1, 2, \ldots, D\} \tag{2}$$

Where : $x_{i,j}^{t+1}$ is the position of the particle at iteration t.

Once the particle movement is completed, the new positions are evaluated and both particle best position $\vec{\vec{\mathrm{Pi}}}$ and global best position $\vec{\mathrm{G}}$ vectors are updated. In case of minimization problems, at the next time step t + 1, they are updated according to the two equations [3] and [4]

$$\vec{P}_i(t+1) = \begin{cases} \vec{P}_i(t), & sif\,(\vec{x}i(t+1)) \geq \vec{P}i(t) \\ \vec{x}_i(t+1), & sinon \end{cases} \tag{3}$$

$$\vec{\mathrm{G}}(t+1) = \arg\min_{\vec{P}i} f\left(\vec{P}i(t+1)\right), \quad 1 \leq i \leq N \tag{4}$$

The essential steps of particle swarm optimization are presented by the following algorithm:

1. Randomly initialize N particles: position and speed.
2. Evaluate particle positions
3. For each particle i, $\vec{P}i = \vec{x}_i$
4. Calculate, $\vec{\mathrm{G}}$, according to [4]
5. Until the stopping criterion is not satisfied do :
    6 Moving particles according to [1] and [2]
    7 Evaluate particle positions
    8 Update $\vec{\vec{\mathrm{Pi}}}$ and $\vec{\mathrm{G}}$ according to [3] et [4]
6. End.

1.1.4 Formalization of local best PSO

In this section, we will present the local best PSO model. The Local best Pso model is characterized by a ring neighborhood topology. Each particle is connected to n particles. The position of each particle is influenced by the best position of its immediate neighbours and not the best position known throughout the swarm.

The velocity of particle i is therefore calculated as follows:

$$v_{i,j}^{t+1} = Wv_{i,j}^{t} + C_1 r_{1\,i,j}^{\,t}\left[p_{i,j}^{t} - x_{i,j}^{t}\right] + C_2 r_{2\,i,j}^{\,t}\left[l_i^{t} - x_{i,j}^{t}\right], j \in \{1, 2, \ldots, D\} \tag{5}$$

Where : l_i^t is the best position that any particle has had in the neighborhood of particle found from initialization through time t.

1.1.5 Configuration of the method

There are several parameters that affect the performance of the PSO method. The choice of these parameters can have a considerable influence on the convergence of the algorithm and generally depends on the problem posed. The particle swarm optimization algorithm is influenced by:

- The swarm size;
- The position of the particles;
- The neighborhood topologies;
- The number of iterations;
- The inertia factor;
- The confidence coefficients;
- The stopping criterion;

1. The Swarm Size

The swarm of particles corresponds to a population of agents, called particles. The swarm is composed of N particles. The size of the swarm is affected by two parameters: the size of the search space, the machine's computing capabilities and the maximum search time. There is no rule to determine this parameter, making numerous tests allows to acquire the necessary experience to apprehend this parameter.

2. The Position of Particles

A uniform particle distribution is recommended. It is usually done randomly according to a uniform distribution on [0,1].

3.The Iteration Number

The number of iterations should not be too low, this could prematurely stop the research process. The number of iterations must not be too high either, this has the consequence of increasing computational complexity per iteration and thus also search time. The number of iterations to obtain a good result is therefore problem-dependent.

4.The Neighborhood Topologies

The choice of a neighbourhood topology has a crucial importance. The topology of the neighbourhood constitutes the structure of the social network and defines with whom each of the particles will be able to communicate. There are many combinations, the following are the most used:

a) Star topology: each particle is connected to all the others, the optimum of the neighbourhood is the global optimum. This topology is referred to as the Gbest PSO. Thanks to a larger particle interconnectivity, Gbest PSO converges faster than the Lbest PSO. However, there is a susceptibility to be trapped in local minima. Figure 2 (a) illustrates the star topology.

b) Ring topology: each particle is connected to n particles (n = 3 in general), this is the most used. Each particle tends to move towards the best in its local neighbourhood. This topology is referred as the Lbest PSO. The Lbest PSO model due to his better exploration abilities, diminished susceptibility to being trapped in local minimal. Figure 2 (b) illustrates the ring topology.

c) Wheel topology: particles communicate with only one central particle. Only the central particle adjusts its position to the best. If there is improvement in its position, the information is then disseminated to its congeners. Figure 2 (c) illustrates the wheel topology.

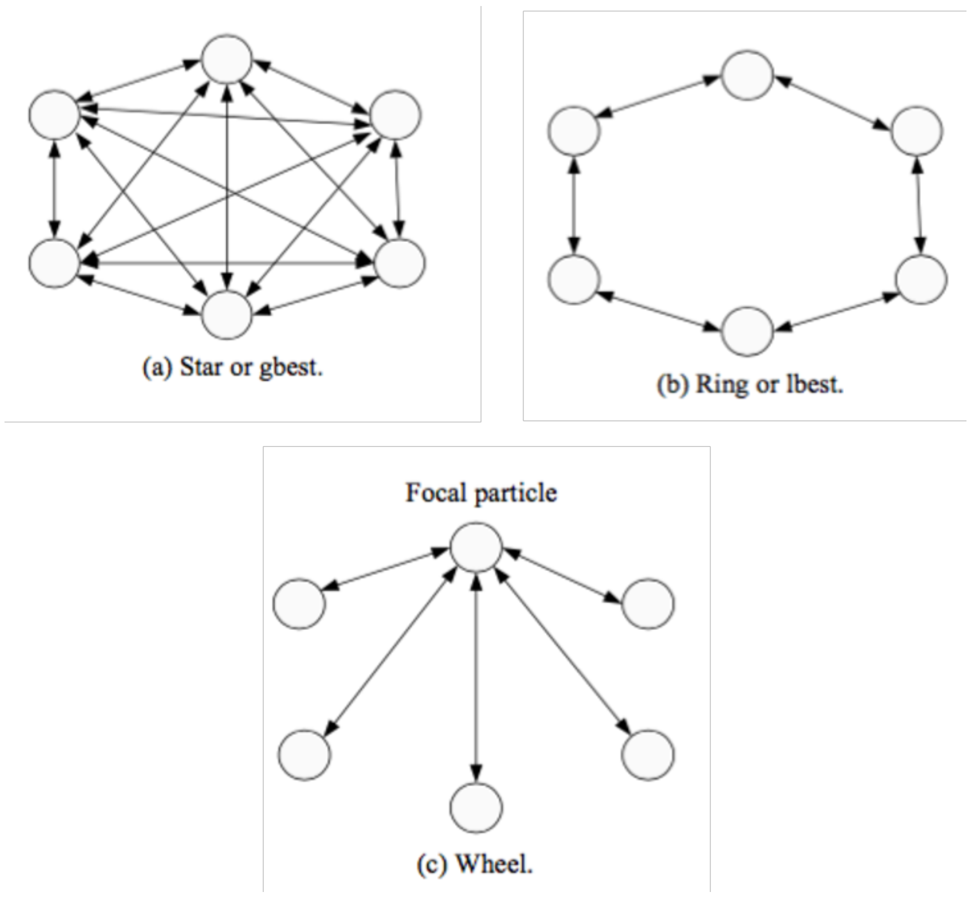


Figure 4: Neighborhood topologies

5. The Inertial Factor

The coefficient of inertia w is used to define the exploration capacity of each particle in order to improve the convergence of the method. Setting this parameter is a compromise between a global exploration $W > 1$ and a local exploration $W < 1$. It represents the adventurous instinct of the particle.

6.The Confidence Coefficients

The coefficients $C_1 r1$ and $C_2 r2$ of the PSO equation are called confidence coefficients, they allow to weigh the tendencies of the particles to follow their instinct of conservation that is the best position by which they have already passed or their panurgism that is to head towards the best position reached by the neighbors.

Coefficients $r1$, $r2$ are random variables evaluated at each iteration according to a uniform distribution on the domain $[01]$.

The coefficients of acceleration C_1, C_2 are constants defined by the relationship $C1 + C2 \leq 4$. Normally, their optimal values are found empirically. However, several empirical studies demonstrate the two acceleration coefficients should be $C1 = C2 = 2$.

7. The Stopping Criterion

The stopping criterion differs depending on the optimization problem and the user's constraints. It is strongly advised to provide the algorithm a stopping criterion. Indeed, convergence towards the global optimal solution is not guaranteed in all cases, even if the experiments indicate the high performance of the method. The algorithm must then run until one of the convergence criteria has been reached:

- a maximum number of iterations;

- the fitness of the solution is sufficient;
- speed ratio is close to 0;

Other stopping criteria can be used depending on the optimization problem and constraints.

1.1.6 Advantages and Disadvantages of PSO

Advantages of the PSO algorithm

- The PSO algorithm conceptually very simple;
- Simple implementation;
- The calculation in PSO algorithm is very simple;
- Easily parallelized for concurrent processing;
- Very efficient global search algorithm;
- Derivative free.

Disadvantages of the PSO algorithm

-Tendency to a fast and premature convergence in local minima points (Gbest Pso)

-PSO algorithm suffers from the partial optimism. The partial optimism degrades the regulation of its speed and direction.

1.2 Implementation of the Particle Swarm Optimization Algorithm in SAS

1.2.1 Notion of likelihood function and maximum likelihood

1. Definition of likelihood function

In order to implement the PSO algorithm in SAS, an objective function is required. In fact, particles do not move randomly, they have an objective to achieve. We have chosen the likelihood function of the normal distribution and we try to find the maximum of this fonction so that the probabilities of achievements observed are also maximum.The objective function of the various PSO improvements presented in the Section 2 are similar to the PSO canonical version of this Section.

The likelihood function plays an important role in statistical inference. The likelihood function measures the goodness of fit of a statiscal model to a sample of data for given values of the unknown parameters. It is formed from joint probability distribution of the sample, but viewed and used as function of the parameters only, thus treating the random variables as fixed at the observed values.

Let $X_1, \ldots, X_1$ an iid sample with probability density function $f(xi;\theta)$, where θ is a $(k \times 1)$ vector of paremeters that characterize $f(xi;\theta)$. The joint density of the sample is, by independence, equal to the product of the marginal densitie:

$$f(x_1, \ldots, x_n; \theta) = f(x_1; \theta) \cdots f(x_n; \theta) = \prod_{i=1}^{n} f(x_i; \theta) \tag{6}$$

The joint density is an n dimensional function of the data $X_1, \ldots, X_1$ given the parameter vector θ

The likelihood function is defined as the joint density treated as a functions of the parameters θ :

$$L\left(\theta|x_1,\ldots,x_n\right)=f\left(x_1,\ldots,x_n;\theta\right)=\prod_{i=1}^{n}f\left(x_i;\theta\right) \tag{7}$$

If the distribution is discrete, f will be the frequency distribution function.

Lets take the case of normal distribution. The normal distibution is an absolutely continuous distibution of probability which depends on two parameters: its expectation, a real number noted m, and its standard deviation, a positive real number noted . The density of probability of the normal law is given by:

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{x-m}{\sigma}\right)^2\right] \tag{8}$$

The likelihood function:

$$L\left(x_1,x_2,\ldots,x_n;m;\sigma\right)=\frac{1}{(\sigma\sqrt{2\pi})^n}\times e^{-\frac{1}{2\sigma^2}\sum_1^n(x_i-m)^2} \tag{9}$$

2. Definition of the Maximum Likelihood Estimator

Formally, the maximum likelihood estimator, denoted $\theta^{\wedge}mle$, is the value of θ that maximizes L(θ|x). That is, $\theta^{\wedge}mle$ solves

$$Max_{\theta}\mathrm{L}(\theta\mathrm{x})$$

It is often quite difficult to directly maximize L(θ|x). It is usually much easier to maximize the log-likelihood function lnL(θ|x). Since ln($\cdot$) is a monotonic function the value of the that maximizes lnL(θ|x) will also maximizeL(θ|x). Therefore, we may also define $\theta^{\wedge}mle$ as the value of θ that solves:

$$Max_{\theta}\mathrm{lnL}(\theta\mathrm{x})$$

1.2.2 Method initialization

In this part we explain the initial part of the code that remains the same for all the versions

```
proc iml;
free TTg Tcv;
do i = 1 to 1000;
free Cgbest ;
/*Data generation.
We start our algorithm with generation of 100 observations of
 uniform distribution (0, 1)*/

donnee = randfun({100 1}, "Normal",0,1);
N  = nrow(donnee); /* Number of observation equal to 100 in our case. */
S  = donnee[+]; /*returns sum of all observations generated*/
S2 = donnee[##]; /*calculates sum of squares of observations*/
pi = constant("pi");


moment = donnee[:]||var(donnee); /*Calculates real parameters*/
moment = abs(round(moment,.1));

/*Bsup determines upper bound of search space at points 50 100
 and binf is a lower bound at points -50 0.00000001.
 Nbpart represents a number of particles that we use
 in search process. */

bsup = {50 100};
binf = {-50 0.0000001};
```

```
nbpart = 30;

/*Initialization of positions for each particle and usage of particle
containment strategy describe in the improvement version of
 particle containment. */

moy = uniform(j(nbpart,1,0))*(bsup[1]-binf[1]) + binf[1];
va = uniform(j(nbpart,1,0))*(bsup[2]-binf[2]) + binf[2];
pos = moy||va;
print pos;

/*Velocity Initialization. */

vit = uniform(j(nbpart,2,1))*10-5;

/*Calculates initial fit values of the likelihood function of
 normal distribution  in stages. */
av =1/(sqrt(2*pi*pos[,2])) ;
ap =(1/(2*pos[,2]))#(S2 - 2*pos[,1]*S + N*(pos[,1]##2));
fit = (av##N)#exp(-ap);

/*Collecting best particles. */
pbest = pos;
/*At the initialization, the best particle position is the initial position
For 30 particles we have 30 fit values. Function fit[<:>], compare
all fit values and chose the greatest value among them and takes the particle
position with this fit value and use it as a best global position*/

gbest = pbest[fit[<:>],]; /*Collectig of position with greatest fit value
to be able to collect particle.
 Cgbest is a table of gbest it means table of best global positions. */
Cgbest = gbest;

/*We set the maximum iteration number at 1000 iterations as one of
stop criteria. */

itermax = 1000;
coun = 0;

/*We determine inertia weight as a constant 0.8 and acceleration
coefficients C1 as 1.5 and C2 as 2.5. */

w = 0.8;
c1 = 1.5;
c2 = 2.5;
```

1.2.3 The PSO algorithm loop in SAS:

The code for the PSO loop is following:

```
do while(coun < itermax);
        r1 = uniform(1);/* r1 and r2 are the random coefficients taken
        from uniform distribution*/
        r2 = uniform(2);
        /* velocity update*/
        vit = w*vit +c1*r1*(pbest - pos) + c2*r2*(gbest - pos);

        /* Position update.*/
        pos = pos + vit;

        /* Count of fit values using new positions.*/
        av =1/(sqrt(2*pi*pos[,2])) ;
        ap =(1/(2*pos[,2]))#(S2 - 2*pos[,1]*S + N*(pos[,1]##2));


        Tfit = (av##N)#exp(-ap);
```

```
    co = fit < Tfit;
    if co[+] ^= 0 then do;/* ^=  means that co is different to 0,
    and if it is the case, the loop is starting to run.
Then we select best position for each particle.*/
            pbest[loc(co),] = pos[loc(co),];/* loc(co),choose particle
            positions with new fit superior to previous fit value.*/
    end;

    /* Selection of best fit values.*/
    fit = fit <> Tfit;/* Vector of best fit values taken from 2 vectors
    fit and Tfit, then it compare all fit values to chose the greatest
    one and use particle position with this fit value as a global best
    position*/
    gbest = pbest[fit[<:>],];

    Cgbest = Cgbest//gbest;/* addition of gbest into table Cgbest*/
            /* Stop Criterion:*/
            Cv_Moy = sqrt(var(pbest[,1]))#(1/pbest[:,1]);
            Cv_Var = sqrt(var(pbest[,2]))#(1/pbest[:,2]);
            Biais = moment-gbest;

            Tcount = coun+1; /* Count iteration number*/
            if nrow(Cgbest) > 9 then do;
                    Tgbest = round(Cgbest[nrow(Cgbest)-9:nrow(Cgbest),],.000000001);
                    /* we take best position and round the position value t with
                    an accuracy of nine decimal places. */

                    if Tgbest = Tgbest[1,] then do; /* If round gbest
                     is equal to real perameter then the loop is stopped and
                     we print the result in the final table*/

                            coun = itermax -1;end;
            end;
            coun = coun+1;
    end;
```

2 PSO algorithm improvements

The Particle Swarm Optimization is a subject of numerous studies. The PSO gained a great popularity because of its' speed of convergence and this algorithm is easily applicable. However, numerous researches have been done in order to improve the algorithm and receive event better algorithm performance. Below, we reviewed some significant modifications of PSO algorithm.

2.1 The inertia weight W.

A parameter inertia weight W was introduced into original particle swarm optimization by Yuhui Shi and Russel Eberhart, scientists in indiana university purdue university indianapolis, in 1998.

The inertia weight W controls the particle direction influence on the future relocation. The W plays the role of balancing the global search and local search. A large value of W facilitates the global search while a small value facilitates a local search.

It was decided that it is better to use the dynamic inertia weight that varies over time to have better performances of the algorithm. The value of W is in range [0.9, 0.4]. With each subsequent iteration, the W is decreasing linearly from 0.9 to 0.4.

The equation of the inertia weight changing in time is following:

$$W = W_{\min} + (W_{\max} - W_{\min}) \cdot \left(\frac{iter}{maxiter}\right) \tag{10}$$

Where $iter$ is the number of iteration at the moment, $maxiter$ is the maximum number of iterations and W_{max} and W_{min} are the values maximum and minimum of W. In general, W_{max} and W_{min} are in range [0, 1]

The equation of particle velocity including inertia weight W is following:

$$v_i^{t+1} = W \cdot v_i^t + C_1 \cdot r_1^t \left(p_i^t - x_i^t\right) + C_2 \cdot r_2^t \cdot \left(g^t - x_i^t\right) \tag{11}$$

The equation to determine a new position is following:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{12}$$

The code in SAS is following:

Beginning of the algorithm loop is the same in all improovements. We start with generation of r_1 and r_2. C_1 and C_1 are constants. However we also define W_{min} and W_{max} initial values.

```
c1 = 2.05;
c2 = 2.05;
wmin =0.4;
wmax =1.2;
do while(coun < itermax);
        r1 = uniform(1);
        r2 = uniform(2);
        coef = uniform(3);
```

W represents an equation that changes on each iteration using W_{min} and W_{max} predefined earlier and then update the velocity equation. This process is running until the stop criterion is reached and results are displayed.

```
        w = wmin + ( wmax -wmin)*(coun/itermax);

        vit = w*vit +c1*r1*(pbest - pos) + c2*r2*(gbest - pos);
```

2.2 Particle containment

One of ideas in order to improve algorithm results a new strategy of particle containment was introduced. This strategy is useful when a particle escapes from a search area. In this case, we put back escaped particle into search space.

Some different variants of particle containment exist. Particularly in our algorithm the system calculates if with the next particle velocity applied this particular particle will get out of the search space. To determine if it is going to get out of the search space, we calculate the value v_{max1} equal to the difference between upper bound and particle actual position and if particle velocity is superior to this value v_{max1}, then we consider that particle is going to get out of the search space. If it is the case, then the system cancels the velocity before the particle gets out of the search space and assigns a new velocity to this particle in order to leave it in the search space. To determine new velocity we subtract from upper bound actual particle position and multiply this difference by a coefficient chosen randomly in range (0, 1).

The code in SAS is following:

```
do while(coun < itermax);
        r1 = uniform(1);
        r2 = uniform(2);
        coef = uniform(3);

        vit = w*vit +c1*r1*(pbest - pos) + c2*r2*(gbest - pos);

        vmax1 = bsup-pos;
        co1 = vit > vmax1;
        if co1[+,1] ^= 0 then do;
                 vit[loc(co1[,1]),1] = coef*(bsup[1]-pos[loc(co1[,1]),1]);end;
        if co1[+,2] ^= 0 then do;
                 vit[loc(co1[,2]),2] = coef*(bsup[2]-pos[loc(co1[,2]),2]);end;
        vmax2 = pos - binf;
        co2 = vit < vmax2;
        if co2[+,1] ^= 0 then do;
                 vit[loc(co2[,1]),1] = -coef*(pos[loc(co2[,1]),1] - binf[1]);end;
        if co2[+,2] ^= 0 then do;
                vit[loc(co2[,2]),2] = -coef*(pos[loc(co2[,2]),2] - binf[2]);end;
        pos = pos + vit;
```

2.3 The constriction factor X

The constriction factor X is one of the Evolution algorithms of classical PSO proposed by Clerc and Kennedy in 2002. It is an adaptive PSO model that uses a new parameter 'X' named the constriction factor. The Inertia weight W and the Maximum Velocity V_{max} are excluded from this model. The use of the constriction factor X allows better control of the divergence. Using the constriction factor the equation for the particle 'i' for velocity is :

$$v_i^{t+1} = X\left[v_i^t + C_1 \cdot \varphi_1 \cdot \left(p_i^t - x_i^t\right) + C_2 \cdot \varphi_2 \cdot \left(g^t - x_i^t\right)\right] \tag{13}$$

The equation to determibe a new position is following:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{14}$$

With:

$$X = \frac{2}{\varphi - 2 + \sqrt{\varphi^2 - 4\varphi}}, \varphi = \varphi_1 + \varphi_2, \varphi > 4 \tag{15}$$

In general, we use ϕ=4.1 and ϕ1=ϕ2 so the constriction factor 'X'=0.7298844

The constriction factor gives a better convergence rate without fixing a maximum velocity. However, in some cases single constriction factor does not lead to an acceptable solution. In this case it can be more interesting to determine V_{max} at the same time with the constriction factor $V_{\max} = \frac{(X_{\max} - X_{\min})}{2}$ to increase the global performances of the algorithm.

The code in SAS is following:

```
itermax = 1000;
     coun = 0;²

     c1 = 2.05;
     c2 = 2.05;
     x = 0.7298844;

     do while(coun < itermax);
             r1 = uniform(1);
             r2 = uniform(2);
             coef = uniform(3);

             vit = x*(vit +c1*r1*(pbest - pos) + c2*r2*(gbest - pos));
```

2.4 Time varying acceleration coefficients C_1 and C_2

The acceleration coefficients are used to control second and third parts of velocity equation and control particle velocity itself. Usually C_1 used should be equal to C_2 ($C_1 = C_2$) and both are in range [0, 4].

In this modification model, C_1 and C_2 are suggested varying in time. The idea if this modification is to boost the global search over the whole search space at the beginning and closer to the end of the search to converge the system to global optima. The value of C_1 is changing from 2.5 to 0.5 and vice versa for C_2, which is changing from 0.5 to 2.5. Authors of this version of algorithm choose those values to realize their idea.

The equations for C_1 and C_2 coefficients are following:

$$C_1 = (C_i - C_f)\frac{iter}{maxiter} + Cf \tag{16}$$

$$C_2 = (C_f - C_i)\frac{iter}{maxiter} + Ci \tag{17}$$

Where C_f and C_i are constants. *maxiter* is a number of iterations predefined by the author in the model and *iter* is a current iteration number.

The code in SAS is following:

```
do while(coun < itermax);
     r1 = uniform(1);
     r2 = uniform(2);
     coef = uniform(3);
     c1 = (ci-cf)*(coun/itermax)+ cf;
     c2 = ci + ( cf -ci)*(coun/itermax);

     vit = w*vit +c1*r1*(pbest - pos) + c2*r2*(gbest - pos);
```

2.5 Fully Informed Particle Swarm

Fully Informed Particle Swarm is a new method proposed by James Kennedy and Rui Mendes in 2004 that uses the known Gbest topology.

In canonical particle swarm algorithm the best performer among all neighbors influences each individual. At the Fully Informed Particle Swarm, a particle is influenced not only by the best neighbor, but also by all the neighbors around this particular particle. Thus, all particles are the source of influence on the velocity.

In this model, neighborhood size is important. It determines how diverse influences are. This concept can be thought as social network.

The new velocity equation is following:

$$v_i^{t+1} = X \left(v_i^t + \sum_{n=1}^{N_i} \frac{U(0,\phi)\left(p_{nbr(n)}^t - x_i^t\right)}{N_i} \right) \tag{18}$$

Where v_i^{t+1} is a velocity calculated using X which is a constriction factor, v_i^t a previous velocity, $nbr(n)$ is a particle's neighbor n for the particle i, N_i is a neighborhood size represented by a number of neighbors chosen to be able to influence the particle's velocity and ϕ is an acceleration coefficient which controls the convergence of the system ans is equal to 4.1 according to previous studies.

The code in SAS for FIPS is following:

```
x = 0.7298844;
c = 4.1;
u = uniform(c);

do while(coun < itermax);
r1 = uniform(1);
r2 = uniform(2);
coef = uniform(3);

vit = x*(vit + (u/nbpart)*(pbest[+]-(nbpart*pos)));

vmax1 = bsup-pos;
        co1 = vit > vmax1;
        if co1[+,1] ^= 0 then do;
          vit[loc(co1[,1]),1] = coef*(bsup[1]-pos[loc(co1[,1]),1]);end;
         if co1[+,2] ^= 0 then do;
                 vit[loc(co1[,2]),2] = coef*(bsup[2]-pos[loc(co1[,2]),2]);end;
         vmax2 = pos - binf;
         co2 = vit < vmax2;
         if co2[+,1] ^= 0 then do;
                 vit[loc(co2[,1]),1] = -coef*(pos[loc(co2[,1]),1] - binf[1]);end;
         if co2[+,2] ^= 0 then do;
                 vit[loc(co2[,2]),2] = -coef*(pos[loc(co2[,2]),2] - binf[2]);end;

         pos = pos + vit;

         av =1/(sqrt(2*pi*pos[,2])) ;
         ap =(1/(2*pos[,2]))#(S2 - 2*pos[,1]*S + N*(pos[,1]##2));

         Tfit = (av##N)#exp(-ap);

         co = fit < Tfit;
         if co[+] ^= 0 then do;
                pbest[loc(co),] = pos[loc(co),];
                end;

         fit = fit <> Tfit;
         gbest = pbest[fit[<:>],];

         Cgbest = Cgbest//gbest;
         Cv_Moy = sqrt(var(pbest[,1]))#(1/pbest[:,1]);
         Cv_Var = sqrt(var(pbest[,2]))#(1/pbest[:,2]);
         Biais = moment-gbest;

         Tcount = coun+1;
         if nrow(Cgbest) > 9 then do;
                Tgbest = round(Cgbest[nrow(Cgbest)-9:nrow(Cgbest),],.000000001);
                if Tgbest = Tgbest[1,] then do;
                                coun = itermax -1;end;
          end;
        coun = coun+1;
end;
```

2.6 The Hybridization strategy : Combination of Particle Swarm Optimization and Differential Evolution algorithms.

For this paper, we have chosen a hybrid strategy that combines a modified particle swarm optimization with a differential evolution called DEMPSO (Differential Evolution + Modified Particle Swarm Optimization).

The aim of such strategy as hybridization is to combine the advantages of Differential Evolution and Particle Swarm Optimization. For Differential Evolution it is an efficient global optimization that prevents us from falling onto local optimum. For Particle Swarm Optimization it is fast convergence speed.

The Differential Evolution Optimization is an algorithm that simulates a biological evolution process. This algorithm is capable to solve optimization problems for nondifferential, multimodal objective and nonlinear functions. According to the classification of optimization methods, Differential Evolution method belongs to the class of stochastic methods, since it uses a random number generator in the solution's search. For this optimization algorithm, an initial population must be randomly generated. Then the next generation is generated through mutation, crossover and selection operations.

In DEMPSO Modified Particle Swarm Optimization (PSO) uses W (inertia weight described in evolution model 2.1, C_1 and C_2 (acceleration coefficients described in evolution model 2.4 that allow greater optimization performance of the algorithm.

At the first stage of the DEMPSO we use DE algorithm to reduce the search space, and then obtained populations are used as an initial population for PSO algorithm to get fast convergence to global optimum. Thus, DEMPSO avoids falling onto local optimum.

The DEMPSO algorithm's procedure is following:

1. Population initialization.

At this stage we initialize the population with DE algorithm. The initial population is generated randomly in the range of search space. The associated velocities are generated randomly as well in the D-dimension space.

The equations to express the initial individuals and associated velocities are following:

$$X_i^0 = \left\{x_{i,1}^0, x_{i,2}^0, \dots, x_{i,D}^0\right\} \tag{19}$$

$$x_{i,j}^0 = x_{\min} + r_{i,j}(0,1) \cdot (x_{\max} - x_{\min}) \tag{20}$$

$$V_i^0 = \left\{v_{i,1}^0, v_{i,2}^0, \dots, v_{i,N_p}^0\right\} \tag{21}$$

Where X is an individual, Np is the population number,$1 \leq i \leq Np, 1 \leq j \leq D$,$D$ is a dimension, $[Xmin, Xmax]$ is a range of search space and $r(0,1)$ is a random number in range $(0,1)$.

2. At this stage we make a loop of DE algorithm to determine a new population.

X_i^t denote a mutation process at time t. The algorithm chooses randomly 3 individuals from previous population and generates a mutant individual V_i^{t+1} according to a following equation :

$$V_i^{t+1} = X_{r1}^t + F\left(X_{r2}^t - X_{r3}^t)\right) \tag{22}$$

Where F is a differential weight in range $(0,1)$ which is a zoom factor to control the amplification of mutation operation. In this equation $r1$, $r2$ and $r3$ are random integer numbers selected from the range $(1, Np)$. Then a crossover operation is starting. This operation is constructing a new population $U_{i,j}^{t+1}$ using a current population and mutant population to increase the diversity of generated individuals.

The process is presented by following equations:

$$U_i^{t+1} = \left\{u_{i,1}^{t+1}, u_{i,2}^{t+1}, \ldots, u_{i,D}^{t+1}\right\} \tag{23}$$

$$u_{i,j}^{t+1} = \begin{cases} v_{i,j}^{t+1}, & ifr(0,1) \leq C_r \quad orj = jrand \\ x_{i,j}^{t}, & ifnot \end{cases} \tag{24}$$

Where j_{rand} is an integer randomly chosen from the range $(1, D)$, Cr is a random crossover parameter in range $(0,1)$ and $r(0,1)$ is a random number chosen from the range 0 to 1.

Next operation is the selection operation that uses a crossover vector U_i^{t+1} and compare it to the target vector X_i^t by evaluating the fitness function value. For the next generation we select a vector with a smaller fitness value.

$$X_i^{t+1} = \begin{cases} U_i^{t+1}, & iff\left(X_i^t\right) \geq f\left(U_i^{t+1}\right) \\ X_i^t, & ifnot \end{cases} \tag{25}$$

We define a stopping criterion and when the global best part with a minimum fitness value achieves our criterion the iteration is stopping and then we move on to the next stage.

The code in SAS for DE loop is following:

```
do while(coun < (itermax/4));
cr = uniform(1);
Spos = pos;
Tpos= j(nrow(pos),ncol(pos),0);
do i = 1 to nrow(pos);
 if i = 1 then do; p = pos[2:nrow(pos),];end;
 if (1<i)&(i <nrow(pos))  then do; p = pos[1:(i-1),]//pos[(i+1):nrow(pos),];end;
 if i = nrow(pos) then do; p = pos[1:nrow(pos)-1,];end;
 ind = ceil(uniform(3)*nrow(p));
 if ind < (nrow(p)-1) then
 Tpos[i,] = p[ind,] + cr*(p[(ind+1),] - p[(ind+2),]);
 else Tpos[i,] = p[ind,] + cr*(p[(ind+2-nrow(p)),] - p[(ind+3-nrow(p)),]);
end;

do j = 1 to ncol(pos) ;
        Trang = ceil( uniform(j(nrow(pos),1,4))*ncol(pos) ) = j ;
        Tcr = uniform(j(nrow(pos),1,4)) <= cr;
        Tcom = Trang|Tcr;
        ppos = Spos[,j];
        if j = 2 then do; Tcom = Tcom&(Tpos[,j]>0);end;
        ppos[loc(Tcom = 1)] = Tpos[,j][loc(Tcom = 1)];
        Spos[,j] = ppos;
end;


av =1/(sqrt(2*pi*Spos[,2])) ;
ap =(1/(2*Spos[,2]))#(S2 - 2*Spos[,1]*S + N*(Spos[,1]##2));
Tfit = (av##N)#exp(-ap);

co = fit < Tfit;
if co[+] ^= 0 then do;
        pos[loc(co),] = Spos[loc(co),];
end;

 coun = coun+1;
end;
```

3. At this stage we run PSO algorithm using the population received at the stage 2 until the stopping criterion is achieved and Global optimum is reached.

For the PSO algorithm we use the algorithm described earlier that uses following equations:

$$v_i^{t+1} = W \cdot v_i^t + C_1 \cdot r_1^t \left(p_i^t - x_i^t\right) + C_2 \cdot r_2^t \cdot \left(g^t - x_i^t\right) \tag{26}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{27}$$

Where x is a new particle position and v is an associated velocity.

The code in SAS for PSO loop is following:

```
av =1/(sqrt(2*pi*pos[,2])) ;
ap =(1/(2*pos[,2]))#(S2 - 2*pos[,1]*S + N*(pos[,1]##2));
fit = (av##N)#exp(-ap);

pbest = pos;
gbest = pbest[fit[<:>],];

Cgbest = gbest;


coun = 0;

wmin =0.4;
wmax =1.2;
ci =0.5;
cf =2.5;

do while(coun < itermax);
        r1 = uniform(1);
        r2 = uniform(2);
        coef = uniform(3);
        c1 = (ci-cf)*(coun/itermax)+ cf;
        c2 = ci + ( cf -ci)*(coun/itermax);
        w = wmin + ( wmax -wmin)*(coun/itermax);

        vit = w*vit +c1*r1*(pbest - pos) + c2*r2*(gbest - pos);

        vmax1 = bsup-pos;
        co1 = vit > vmax1;
        if co1[+,1] ^= 0 then do;
          vit[loc(co1[,1]),1] = coef*(bsup[1]-pos[loc(co1[,1]),1]);end;
         if co1[+,2] ^= 0 then do;
                  vit[loc(co1[,2]),2] = coef*(bsup[2]-pos[loc(co1[,2]),2]);end;
         vmax2 = pos - binf;
         co2 = vit < vmax2;
         if co2[+,1] ^= 0 then do;
                  vit[loc(co2[,1]),1] = -coef*(pos[loc(co2[,1]),1] - binf[1]);end;
         if co2[+,2] ^= 0 then do;
                  vit[loc(co2[,2]),2] = -coef*(pos[loc(co2[,2]),2] - binf[2]);end;

        pos = pos + vit;


        av =1/(sqrt(2*pi*pos[,2])) ;
        ap =(1/(2*pos[,2]))#(S2 - 2*pos[,1]*S + N*(pos[,1]##2));
                Tfit = (av##N)#exp(-ap);

        co = fit < Tfit;
        if co[+] ^= 0 then do;
            pbest[loc(co),] = pos[loc(co),];
        end;

        fit = fit <> Tfit;
        gbest = pbest[fit[<:>],];

        Cgbest = Cgbest//gbest;
        Cv_Moy = sqrt(var(pbest[,1]))#(1/pbest[:,1]);
        Cv_Var = sqrt(var(pbest[,2]))#(1/pbest[:,2]);
        Biais = moment-gbest;

        Tcount = coun+1;
        if nrow(Cgbest) > 9 then do;
                Tgbest = round(Cgbest[nrow(Cgbest)-9:nrow(Cgbest),],.000000001);
                if Tgbest = Tgbest[1,] then do;
                        coun = itermax -1;end;
        end;
        coun = coun+1;
end;
```

3 Results of analysis and comparison

3.1 Performance analysis of the PSO algorithm versions

3.1.1 Presentation of indicators used for the analysis.

The indicators, used for the analysis of PSO algorithm convergence and its improvements, are following:

Indicators in finite time: the algorithm is executed only one time.

```
Tcount : the number of iterations of the algorithm in order to
converge.

V_moy : the standard deviation of the optimum found by the swarm for
the mean argument of the likelihood function of the normal
distribution.

V_var : the standard deviation of the optimum found by the swarm for
the variance argument of the likelihood function of the normal
distribution.

Biais_moy : the difference between the optimum of the mean found
by the best particle of the swarm and the true optimum of the mean
of the likelihood function.

Biais_  Var  : the difference between the optimum of
the variance found
by the best particle of the swarm and the true optimum of the
variance of the likelihood function.
```

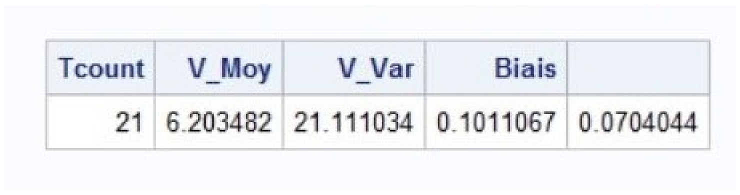
The indicators in asymptotic: The algorithm is executed 1000 times successively.

```
Moy_IterAs : asymptotic mean of the number of iterations for a
given algorithm.

Cv_Iteras : asymptotic coefficient of variation of the number
of iterations.

Ascv_moy : the asymptotic coefficient of variation of the optimum
found by the swarm for the mean argument of the likelihood
function of the normal distribution.

Ascv_var : the asymptotic coefficient of variation of the optimum
found by the swarm for the variance argument of the likelihood
function of the normal distribution.
```

```
Asbiais_moy: the asymptotic bias of the mean.

Asbiais_var: the asymptotic bias of the vareance.
```

3.1.2 Inertia weight

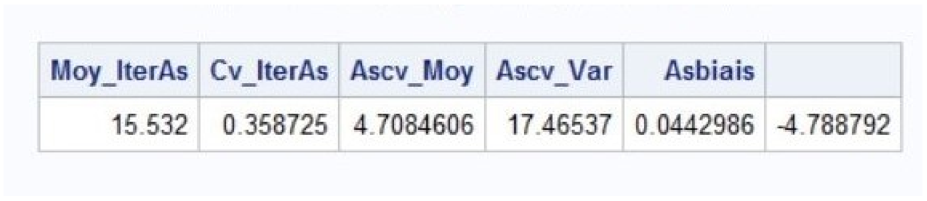
Finite time: 1 time execution



Tcount	V_Moy	V_Var	Biais	
21	6.203482	21.111034	0.1011067	0.0704044

For this version, according to the chosen stopping criterion, the algorithm needs 21 iterations to converge. However, the standard deviation of particles (it means standard deviation at the level of their arguments mean (6.20) and variance (21.11)) shows that the swarm remains dispersed when the algorithm stops. However, the best particle is close to the function optimum, because its bias vector is only (0.1, 0.07) respectively for its mean and variance argument.

Asymptotic : 1000 times execution

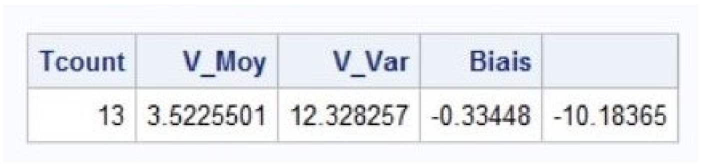


Moy_IterAs	Cv_IterAs	Ascv_Moy	Ascv_Var	Asbiais	
15.532	0.358725	4.7084606	17.46537	0.0442986	-4.788792

Asymptotically, on average, the algorithm needs 16 iterations to converge. The mean argument of the best particle is even more precise with a bias of 0.04, but the one that concerns variance deviates by about 5 points from the optimum.

3.1.3 Particle containment

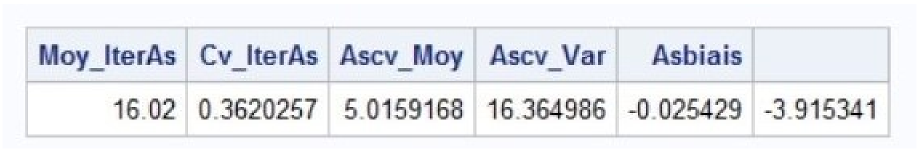
Finite time: 1 time execution



Tcount	V_Moy	V_Var	Biais	
13	3.5225501	12.328257	-0.33448	-10.18365

The algorithm with particle containment strategy needs 13 iterations to converge. The standard deviations in terms of their arguments are respectively (3.52) for the mean and (12.33) for the variance. The restoring force of this modification is twice as large as the previous version. We can observe it with standard deviations of the particles which have fallen half. The mean argument of the best particle remains very close to the optimum, but that of the variance shows low performance with a bias of 10 points.

Asymptotic : 1000 times execution

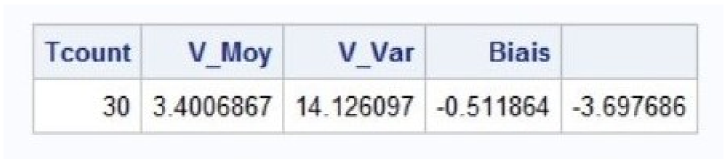


Moy_IterAs	Cv_IterAs	Ascv_Moy	Ascv_Var	Asbiais	
16.02	0.3620257	5.0159168	16.364986	-0.025429	-3.915341

Asymptotically, on average, the algorithm needs 16 iterations to converge. The asymptotic case shows that the best particle always finds the optimum for the mean, but the variance remains biased by deviating from its optimum by 4 points.

3.1.4 The constriction factor X.

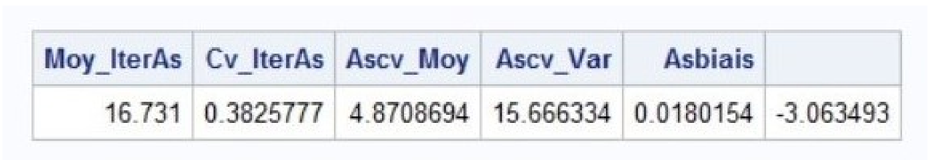
Finite time: 1 time execution



Tcount	V_Moy	V_Var	Biais	
30	3.4006867	14.126097	-0.511864	-3.697686

The algorithm with constriction factor needs 30 iterations to converge. The standard deviations in terms of their arguments are respectively (3.40) for the mean and (14.13) for the variance. The standard deviations of the particles show that the swarm remains relatively dispersed when the algorithm is stopped. However, the swarm converges more than the first inertia weight modification towards the function optimum. In addition, the bias vector of the particle with the best position is equal to (-0.5; -3.7) respectively for its mean and variance argument. The particle with the best position in the search space is close to the optimum for the mean and far from the optimum for the variance.

Asymptotic : 1000 times execution

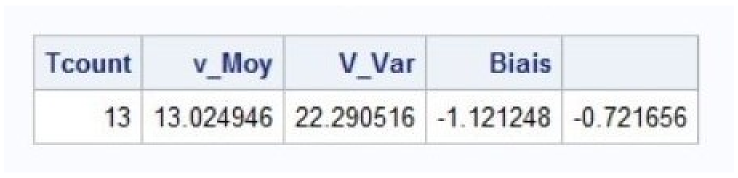


Moy_IterAs	Cv_IterAs	Ascv_Moy	Ascv_Var	Asbiais	
16.731	0.3825777	4.8708694	15.666334	0.0180154	-3.063493

Asymptotically, on average, the algorithm needs 17 iterations to converge.The bias vector of the particle with the best position is equal to (0.02, -3.06) respectively for its mean and variance argument. The asymptotic case shows that the best particle is very close to the optimum for the mean, but it remains far from its optimum for the variance.

3.1.5 Time varying acceleration coefficients C_1 and C_2

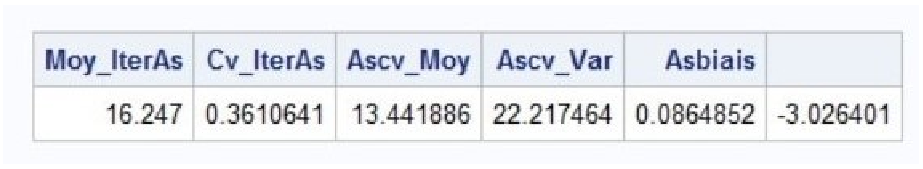
Finite time: 1 time execution



Tcount	v_Moy	V_Var	Biais	
13	13.024946	22.290516	-1.121248	-0.721656

The algorithm with time varying acceleration coefficients C_1 and C_2 needs 13 iterations to converge. The standard deviations in terms of their arguments are respectively (13.02) for the mean and (22.3) for the variance. The standard deviations are high and the swarm has not converged towards the optimums. However, the mean and variance arguments of the best particle are close to the optimum. The bias vector of the particle with the best position is (-1.12; -0.72).

Asymptotic : 1000 times execution

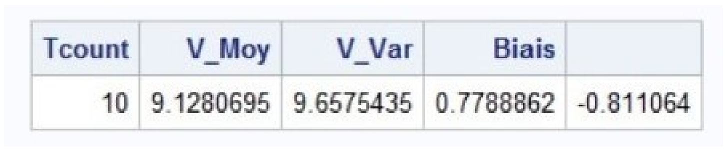


Moy_IterAs	Cv_IterAs	Ascv_Moy	Ascv_Var	Asbiais	
16.247	0.3610641	13.441886	22.217464	0.0864852	-3.026401

Asymptotically, on average, the algorithm needs 16 iterations to converge. The bias vector of the particle with the best position is equal to (0.08, 3) respectively for its mean and variance arguments. The asymptotic case shows that the best particle is very close to the optimum for the mean, but it remains far from its optimum for the variance.

3.1.6 Fully Informed Particle Swarm

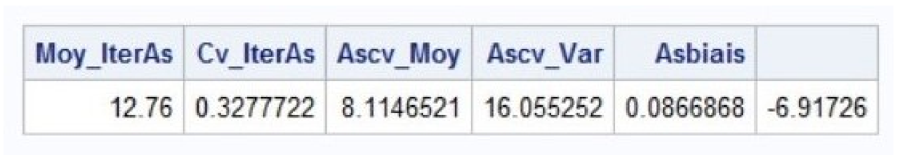
Finite time: 1 time execution



Tcount	V_Moy	V_Var	Biais	
10	9.1280695	9.6575435	0.7788862	-0.811064

The algorithm with Fully Informed Particle Swarm needs 13 iterations to converge. The standard deviation is around 9 for the mean and variance arguments. High standard deviations show that the swarm remains dispersed after stopping the algorithm. The best particle is very close to the optimum, with a bias vector of only (0.78; -0.81).

Asymptotic : 1000 times execution

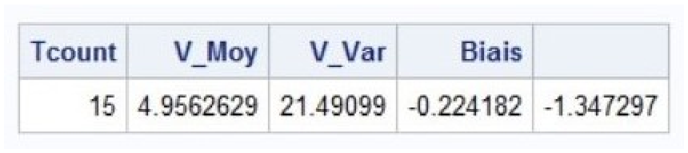


Moy_IterAs	Cv_IterAs	Ascv_Moy	Ascv_Var	Asbiais	
12.76	0.3277722	8.1146521	16.055252	0.0866868	-6.91726

Asymptotically, on average, the algorithm needs 13 iterations to converge. Also, the bias vector of the particle with the best position is equal to (0.09; -7) respectively for its mean and variance argument. The asymptotic case shows that the best particle always finds the optimum for the mean, but for the variance, it is too far from its optimum.

3.1.7 The Hybridization strategy : Combination of Particle Swarm Optimization and Differential Evolution algorithms.

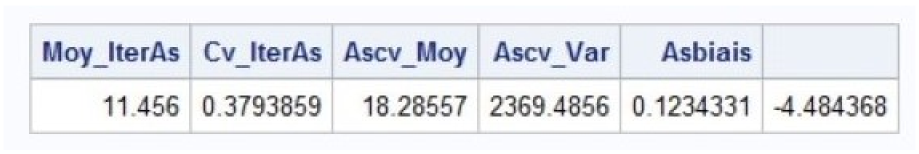
Finite time: 1 time execution



Tcount	V_Moy	V_Var	Biais	
15	4.9562629	21.49099	-0.224182	-1.347297

The algorithm with Hybridization strategy needs 15 iterations to converge. However, the standard deviation of particles (it means standard deviation at the level of their arguments mean (5) and variance (2)) shows that the swarm remains dispersed when the algorithm stops. However, the best particle is close to the function optimum, because its bias vector is only (-0,22 , -1,34) respectively for its mean and variance argument.

Asymptotic : 1000 times execution



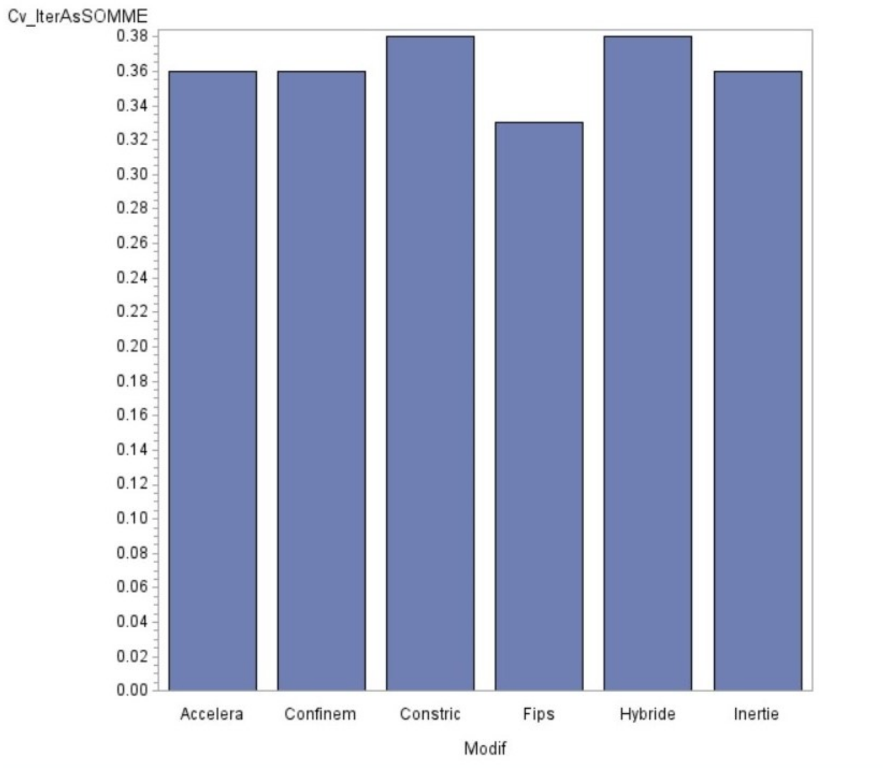
Moy_IterAs	Cv_IterAs	Ascv_Moy	Ascv_Var	Asbiais	
11.456	0.3793859	18.28557	2369.4856	0.1234331	-4.484368

Asymptotically, on average, the algorithm needs 11 iterations to converge. Moreover, the mean argument of the best particle is even more precise with a bias of 0.14, but the one of the variance deviates by just over 4 points from the optimum.

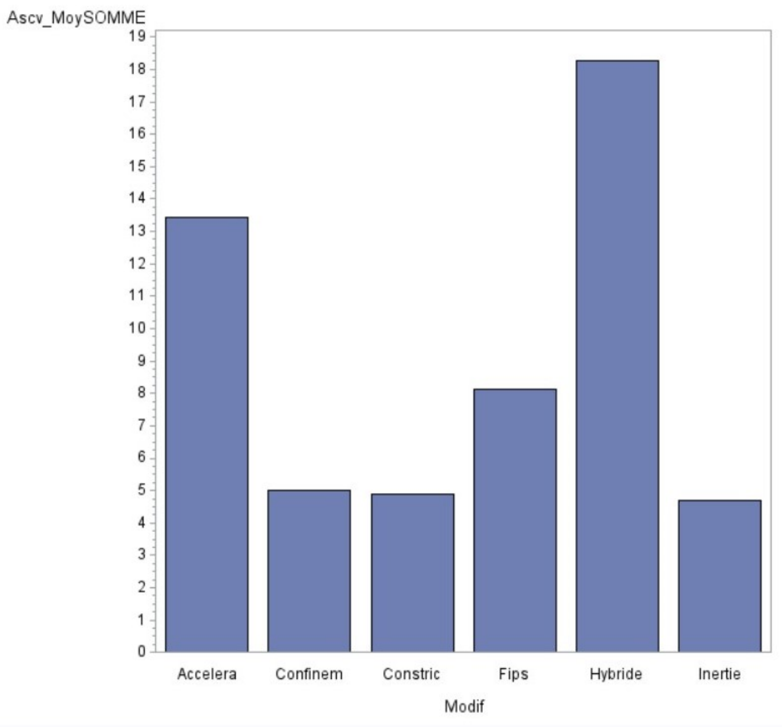
In this analysis, we did not present the results of canonical PSO version, because it doesn't use particle containment strategy and the particles are going out of the search space. Because of this situation, the results can't be compared. In this case, we obtained a negative variance.

3.2 Results comparison

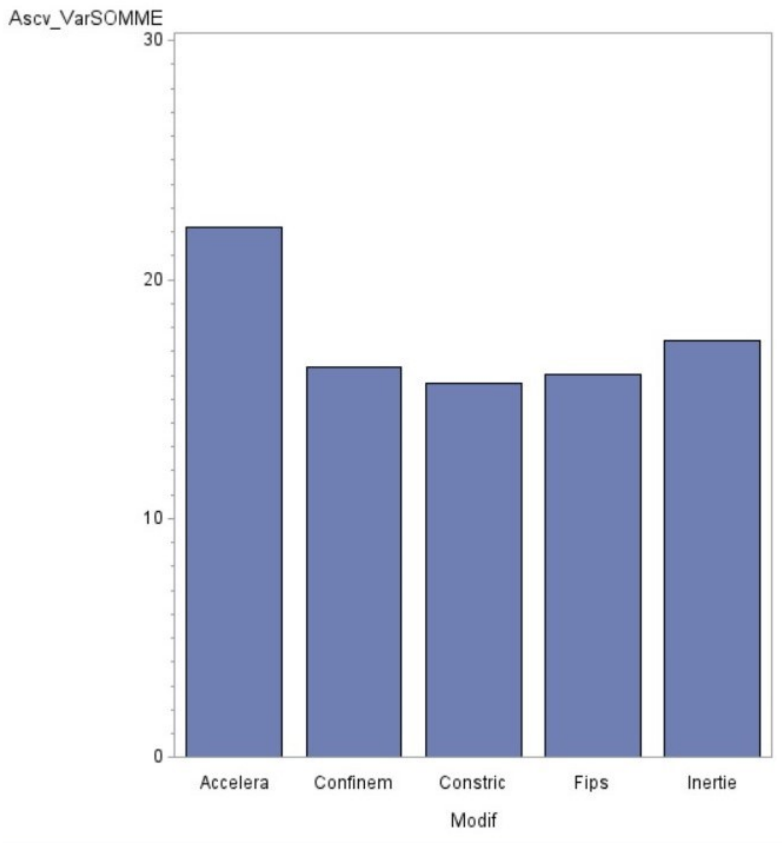
In terms of the variability of the number of iterations which allows to the algorithm to converge, the dispersion is feeble and almost the same (0.37) for all the modifications except algorithm FIPS, which has a slightly smaller dispersion of its iterations (0.34).



The asymptotic convergence of the swarm for the mean argument is strongly varying from one algorithm to another. The version, which introduced inertia weight, shows the best performance with the variation coefficient equal to 4.7 and the version with time varying acceleration coefficients C_1 and C_2 shows the lowest score with coefficient equal to 13.44.



However, observing the argument variance, all the algorithms show a low performance (minimum variation coefficient is equal to 16).



We should note that PSO hybrid strategy shows an anomaly as to the convergence of the variance argument, which is equal to 22369, despite the systematic bias of its best particle, which is relatively low (-4, 4).

Conclusion

In the first section of this thesis, we introduced the Particle swarm optimization algorithm. The method of PSO is used in order to find optimal values in a search area. It is inspired by the principle of foraging behaviour of swarm of birds. Particle swarm optimization consist swarm of particles and these particles represent a potential solution called better condition. Particle will move through a multidimensional search space to find the best position in that space. PSO has many advantages and disadvantages. Different variants have been designed to verify effects of various parameters on the performance of PSO.

In the second section of our thesis, we have reviewed the major improvements since pso algorithm creation. In original PSO, there was no inertia weight but to improve the performance researchers introduced the inertia weight. The appearance of inertia weight allows to control the particle direction influence on the future relocation. Then, they tried to improve the performance by introducing the particle containment in order to prevent particles from getting out of the search space. Some researchers intoduced a constriction factor to control better the divergence. However it is important to note that the usage of this method exclude maximum velocity and inertia weight. Other researchers introduced the model with time varying acceleration coefficients C_1 and C_2. This model allows the control of second and third part of the velocity equation on the future particle relocation. Later a new remarkable modification was introduced in the algorithm called Fully Informed Particle Swarm. where particles take into account not only global best result but also best result found by their neighbors. In the last few years, more sophisticated particle swarm optimization variants appeared by making a hybrid optimization method using PSO combined with other optimizers.This strategy avoids getting into a local optimum and has relatively fast convergence.

In the third section, we have analyzed and compared the performance of the PSO algorithm and its variants through an application: estimation of the maximum likelihood of the normal distribution. For all the algorithms tested, the convergence speed is practically similar. Asymptotically, the version of the PSO algorithm with inertia weight has the best performance and the version with time varying acceleration coefficients has the worst for the mean argument. In addition, all the PSO versions tested have a feeble performance for the variance argument.

There are promising directions for future work. Future works need to understand where the algorithm suffers in order to understand limitations of PSO. Studies can be conducted out toward neighborhood topologies, parameter adjustment, initial distribution of particles and methods to deal with stagnation.

References

[1] Abbas El Dor. *Perfectionnement des algorithmes d'optimisation par essaim particulaire : applications en segmentation d'images et en electronique.(French)* University Paris-Est,France,2012.

[2] Bingyan Mao, Zhijiang Xie,Yongbo Wang, Heikki Handroos and Huapeng Wu. *A Hybrid Strategy of Differential Evolution and Modified Particle Swarm Optimization for Numerical Solution of a Parallel Manipulator.* State Key Laboratory of Mechanical Transmission, Chongqing University,China, Lappeenranta University of Technology,Finland,2018.

[3] Craig, W.Reynolds, Flocks, Herds and Schools. *A Distributed Behavioral Model Published in Computer Graphics,* 21(4), July 1987, pp. 25-34.

[4] Fran van den Bergh. *An Analysis of Particle Swarm Optimizers.* University of Pretoria etd, 2006.

[5] Nadia Smairi. *Optimisation par essaim particulaire : adaptation de tribes a l'optimisation multiobjectif.(French)* Universite Paris-Est; Ecole Nationale des Sciences de l'Informatique (La Manouba, Tunisie), 2013.

[6] Rui Mendes,Member, IEEE, James Kennedy and José Neves. *The Fully Informed Particle Swarm: Simpler, Maybe Better* IEEE Transactions on Evolutionary Computation, Volume 8,No.3,June 2004.

[7] Saptarshi Sengupta. *Particle Swarm Optimization: A survey of historical and recent developments with hybridization perspectives.* Vanderbilt University

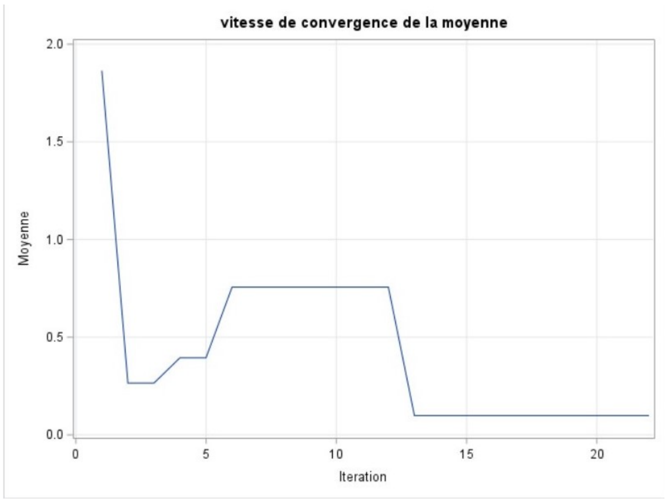
[8] Swagatam Das, Ajith Abraham, Amit Konar. *Particle Swarm Optimisation and Differential Evolution Algorithms: Technical Analysis, Applications and Hibridization Perspectives,* Department of Electronics and Telecomunication Engineering, Jadavrup University, India,Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology,Norway,2008.

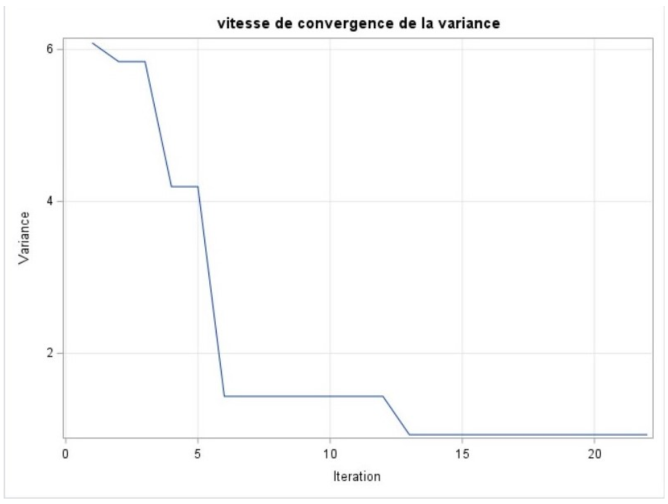
[9] Variants of PSO Algorithm,United Kingdom,2018.
`https://www.ukdiss.com/examples/variants-of-pso-algorithm.php?vref=1`

[10] Yuhui Shi and Russel Eberhart *A modified Particle Swarm Optimizer.* Department of Eletrical Engineering, Indiana University- Purdue University, Indianapolis, IN 46202-5160,1998.

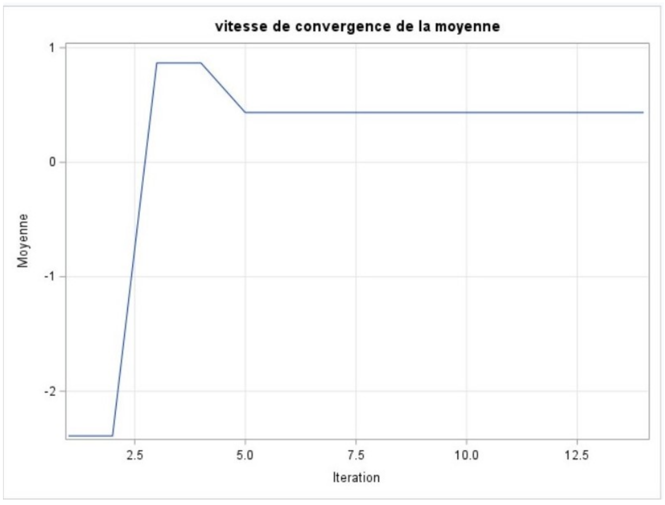
Annex

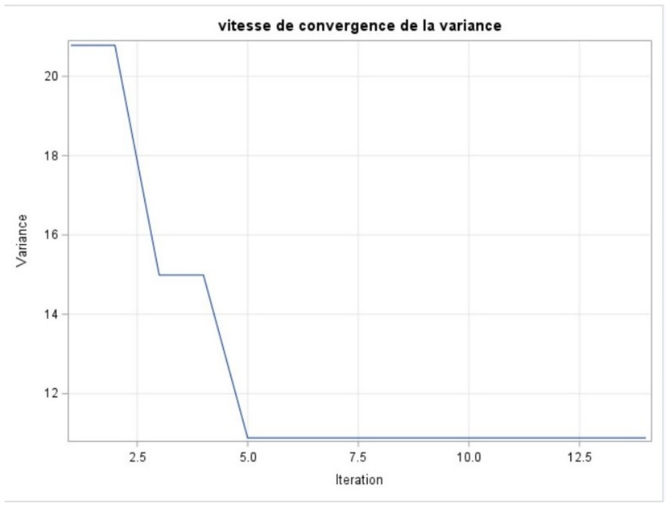
Annex 1 : Inertia weight



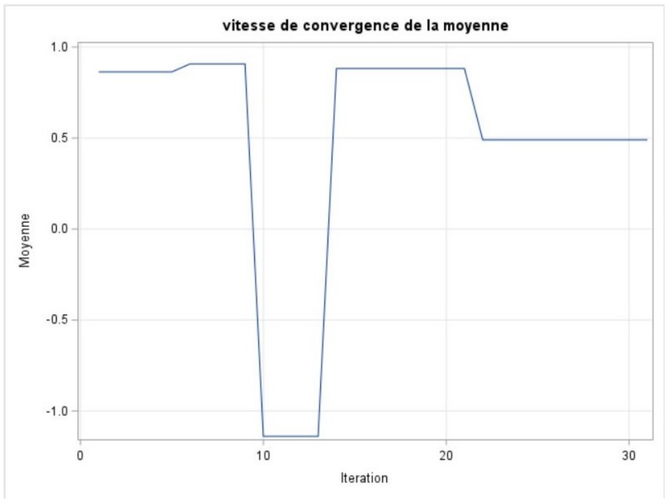


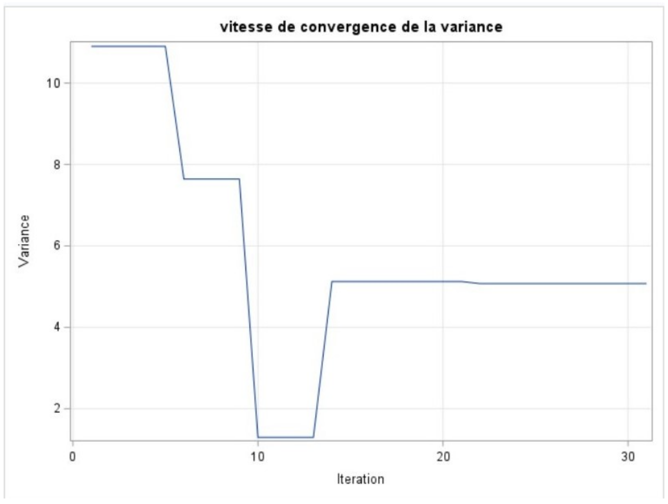
Annex 2 : Particle containment



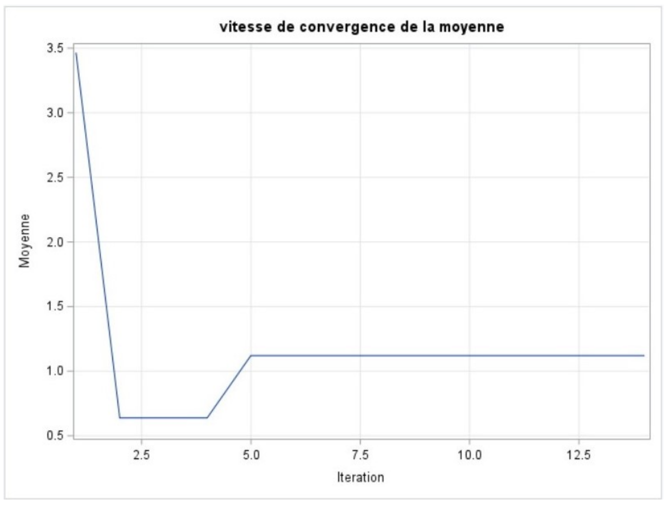


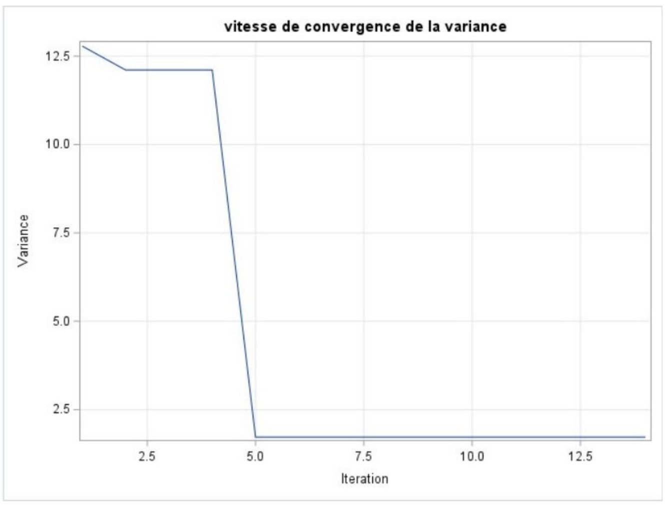
Annex 3 : The constriction factor X



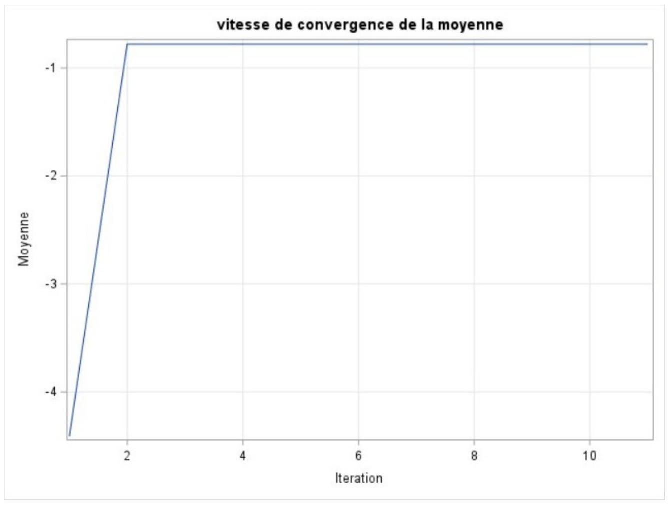


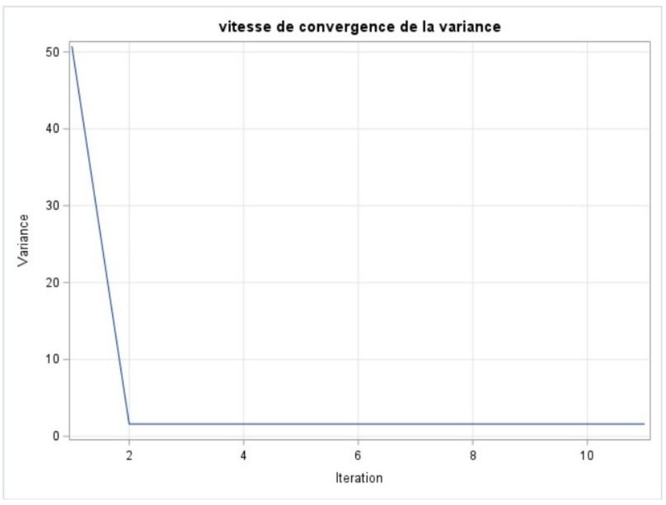
Annex 4 :Time varying acceleration coefficients C_1 and C_2



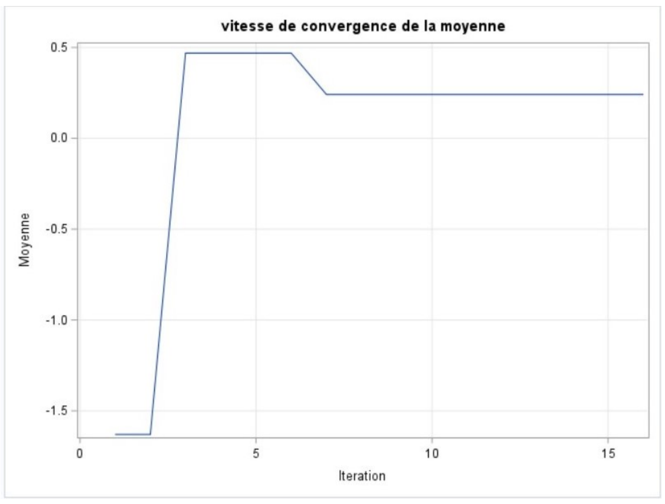


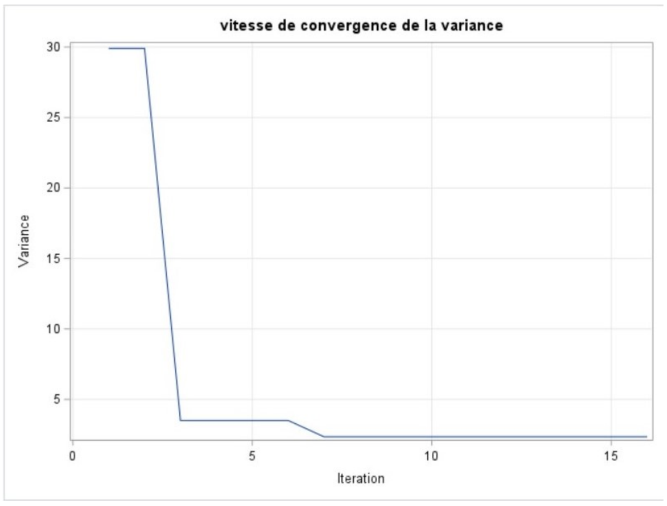
Annex 5 : Fully Informed Particle Swarm





Annex 6 : The Hybridization strategy


vitesse de convergence de la moyenne
Moyenne
Iteration


vitesse de convergence de la variance
Variance
Iteration