PARTICLE SWARM OPTIMIZATION (PSO): AN ALTERNATIVE METHOD FOR COMPOSITE OPTIMIZATION

María Guillermo Alonso¹, Pierre Duysinx²

¹,² Université de Liège, Liège, Belgium, {mguillermo, p.duysinx }@ulg.ac.be

1. Abstract
The aim of this article is to present the background, improvements and variants of the particle swarm optimization algorithm (PSO) based on the movement of a bird flock, as an alternative method of optimization for composite structures. The strong point of this algorithm is its simplicity and adaptability to all kind of problems and fields. The first part of this article consists in a compilation of different papers that deal with problems encountered while applying PSO and their solutions. The article discusses about the way to deal with large design spaces, about the convergence problem and local vs. global minima.

In the second part of the article, the PSO algorithm is compared to the classical genetic algorithms used for optimization in the software BOSS Quattro. Other kinds of problems that can appear are the constrained ones. In order to solve them, a study of the Augmented Lagrange Multiplier Method is considered. In this case, the PSO is used to optimize the Augmented Lagrangian function giving satisfactory results for common problems.

As future prospects, the study of different ways to improve the algorithm is considered. For example, the way to update the particles has been studied: choosing to update the worse particle, the best one or a percentage of the best/worst population. Future studies could include the way the particles are chosen. As last results, industrial applications will be tested in different software.

2. Keywords: PSO, composite structure optimization, composite materials, genetic algorithms

3. Introduction
Particle Swarm Optimization (PSO) method is an alternative method for optimization that was first introduced by Eberhart and Kennedy [1]. It is inspired by the social movement of a swarm of birds searching for food. As in other optimization metaheuristics [13], like the evolutionary algorithms ([16]-[18]), simulated annealing ([14], [15]), or ant colony optimization (ACO) ([19]-[21]), the search is an iterative process based in random decisions. In the PSO algorithm the swarm of individuals, called particles and characterized by a location and a velocity, explores the search space iteratively looking for the global minimum or maximum of a function. At each iteration, their positions and velocities are updated. After the particles’ velocities have been updated, they move to their new position.

Although this kind of algorithms usually needs a higher number of function evaluations than the gradient-based algorithms, they provide attractive characteristics that make them an interesting point of study. These algorithms are generally easy to implement and they are better suited for finding global solutions. However, they should be used with precaution and only when the gradient-based algorithms are not suitable for the problem, like discrete and discontinuous problems.

The objective of this research is to introduce this method into the software BOSS Quattro, an open object optimization oriented platform, used for optimization, and to study how this method can be applied to composite structure optimization to obtain for instance an optimum stacking sequence.

This paper is organized as follows. First the PSO method is explained and several improvements introduced by different authors are presented. Second, some applications of the PSO method are performed with the software BOSS Quattro Optimization tool. Then, an investigation of the update process of the particles is presented. Finally, the constrained problem is studied and the Augmented Lagrangian method for PSO is described with some applications.

4. Particle Swarm Optimization
The Particle Swarm Optimization (PSO) algorithm is based on the social behavior of the nature swarm’s individuals while searching for promising food. The individuals explore the space keeping in memory the promising spots already found. They also share that information with the rest of the swarm, which adapts its trajectory using all the information available: the individual memory as well as the knowledge gained by the population.

Each individual is named as a ‘particle’, which is a candidate solution to the problem. Each particle is a point in the
search space. It keeps the track of the coordinates with the best solution that it has ever been found until that moment. This one, which is evaluated through an objective function or fitness, has also stored as information of the particle. The value of the coordinates is stored as “pbest”. Another value stored is the best value found by the whole swarm and its location. This value is stored as “gbest”. Finally, PSO algorithm attributes a velocity to each particle in order to update their positions. The position of each individual of the swarm is based on the social behavior of the swarm, which will find the solution returning to promising points of the search space which were previously discovered.

4.1. Description of the basic algorithm
In the numerical approach of the algorithm the position \( x \) of a particle \( i \) is updated for iteration \( k+1 \) as shown by Kenney and Eberhart [1] in Eq.(1)

\[
x_{k+1}^i = x_k^i + v_{k+1}^i \Delta t
\]

where \( x_{k+1}^i \) is the position of the particle \( i \) at iteration \( k+1 \); \( v_{k+1}^i \) is the velocity vector at iteration \( k+1 \) and \( \Delta t \) is the time step value considered unitary for this paper.

The method to update the velocity vector presented in this paper is based on the formula presented by Shi and Eberhart [2]:

\[
v_{k+1}^i = w v_k^i + c_1 r_1 \frac{(p_k^i - x_k^i)}{\Delta t} + c_2 r_2 \frac{(p_g^i - x_k^i)}{\Delta t}
\]

In Eq. (2) \( v_k^i \) is the velocity vector at iteration \( k \); \( r_1 \) and \( r_2 \) are random numbers between 0 and 1; \( p_k^i \) represents the best position found by particle \( i \) so far, and \( p_g^i \) corresponds to the global best position in the swarm up to iteration \( k \). As before, \( \Delta t \) is considered unitary. The other parameters are problem dependent; \( c_1 \) and \( c_2 \) are “trust” parameters indicating how much confidence the current particle has in itself (\( c_1 \) or cognitive parameter) and how much confidence it has in the swarm (\( c_2 \) or social parameter), and \( w \) is the inertia weight. This last parameter is very important in the exploration abilities of the swarm. It controls the space to explore: higher values help the global behavior while smaller values facilitate the local behavior.

The PSO algorithm consists in changing the position and in accelerating the particle with the velocity toward its pbest and gbest locations at each iteration.

The original process for implementing the global version of PSO can be outlined as follows:

1. Initialize a population of particles with random positions and velocities in the search space.
2. For each particle, evaluate the desired optimization fitness function.
3. Compare the particle fitness evaluation with the particle pbest. If current value is better than pbest, then set pbest value equal to the current value and the pbest location equal to the current location in the search space.
4. Compare fitness evaluation with the population’s overall previous best. If the current value is better than gbest, then reset gbest to the current particle’s array index and value.
5. Change the velocity and position of the particle according to equations (2) and (1), respectively.
6. Go to step 2 until stopping criterion is met, usually a sufficiently good fitness or a maximum number of iterations.

The figure 1 taken from Perez and Behdinan [3] shows graphically how the PSO position and velocity update is.

Figure 1: PSO position and velocity update
4.2. PSO models
- Global model
  The global or gbest model favors a fast convergence over robustness. In this method there is just one particle, the global best particle, which gives the “best solution” across all the particles of the swarm. The particle is like an attractor and tries to keep the movement of the rest of the particles towards it. However, the particles can converge prematurely.
- Local model
  The local or ibest model tries to avoid the premature convergence using more than one attractor. A subset of particles is assigned for each particle from which the local best particle is then selected.

5. Algorithm improvements
Different authors have introduced different enhancements. Some of them are summarized in the current work and a subset of these ones will be used for the implementation.

5.1. Problem parameters
Depending on the values given to the “trust” parameters, the algorithm gives more importance to the group, higher values to the social parameter $c_2$, or to the individual, higher values to the cognitive parameter $c_1$. Usual values could be between 0.5 to 2.5, depending on the authors and on the nature of the problems. The inertia weight $w$ is another parameter with high influence in the search. In Eq.(2) the first part of the second term corresponds to the previous velocity of the particle. The other two parts help the particle to move in other directions. Without the first part on the right of the Eq.(2) the movement of the particles would only be determined by the current position and their best in history. The original PSO equation of updating positions could be obtained by setting $w=1$. In Ref. [2] Shi and Eberhart have studied the influence of this factor in the convergence. Setting its value to $w \in [0.8,1.2]$ gives a faster convergence, but values bigger than 1.2 gives more problems. They propose a dynamic variation of the inertia weight as a function of the iteration number, as shown in Eq.(3)

$$w_{k+1} = w_{\text{max}} - \frac{w_{\text{max}} - w_{\text{min}}}{k_{\text{max}}} k$$

being $k$ the current iteration and $k_{\text{max}}$ the maximum number of iterations chosen by the user.

If after a determined number of iterations no improvement of the solution is obtained Fourie and Groenwold [4] proposed a dynamic decrease of the inertia weight value with a fraction multiplier $k_w$. This implementation is shown in Eq.(4). An initial value $w^0$, based on numerical experimentation, is chosen and then it is decreased with Eq.(4) if no improvement of the solution is found during $h$ consecutive steps.

$$\text{If } f(p_g)|_k > f(p_g)|_{k-h} \text{ then } w_{k+1} = k_w w_k$$

where $k_w$ is a constant between 0 and 1.

Chatterjee and Siarry [5] propose an adaptation of Eq.(3). They found the need of starting with a high value of the inertia weight to increase the global search and then decreasing linearly as in Eq.(3) to keep the local search in next iterations. They also present another possibility to improve the performance of the algorithm consisting of a nonlinear dynamic adaptation function that actualizes the velocity at each iteration. It is presented in Eq.(5). The actualization of the velocity is performed according to Eq.(5) in this article.

$$w_k = \left[ \frac{(k_{\text{max}}-k)^n}{k_{\text{max}}} \right] (w_{\text{max}} - w_{\text{min}}) + w_{\text{min}}$$

Usual values for maximum and minimum are $w_{\text{max}}=0.4$ and for $w_{\text{max}}=0.9$. Bergh [6] proposes a simple condition that guarantees the convergence:

$$\frac{1}{2} (c_1 + c_2) - 1 < w < 1$$

Numerical applications
In this point the efficiency of the PSO is proven with different test cases and validation examples. First, a common function has been tested with the optimization tool of the software BOSS Quattro and an evolutionary algorithm available in the BOSS Quattro code. Then it has been tested with the new algorithm, PSO, that we integrated in the BOSS Quattro software environment. After that, some simple examples of composite laminates have been performed in order to show that the performance of PSO for this kind of problem.

3
5.2. BOSS Quattro optimization tool
As explained in [7] BOSS Quattro is an open environment which controls simulation analyses based on parametric models. A parameterized model refers to the state of a system which is described using parameters. An analysis is a procedure applied on a model which yields results or responses. More generally, BOSS Quattro can be partly viewed as an application manager: it builds and runs chains of tasks involving different applications, and collects results automatically.

Among the different "engines" available in BOSS Quattro, this study is focusing on using the Optimization tool. The optimization engine supplies the designer with algorithms to optimize the value of a user-defined function when the system under study is subject to constraints.

The BOSS Quattro optimization task proposes to work with different algorithms:
- **SQP** (Sequential Quadratic Programming), second order algorithm.
- **CONLIN** (convex Linearization), first order algorithm based on convex linearization.
- **GCM** (Globally convergent Method), second order algorithm that uses the method of moving asymptotes.
- **Evolution JCO Bin2Dec**, evolution algorithm with binary coded individuals.
- **Evolution JCO Real**, evolution algorithm with real coded individuals.
- **Surrogate-based optimization**, zero order optimization tool designed for large models.

The algorithms considered for the simulations presented in this paper are the evolutionary ones, the zero-order algorithms allowing the user to solve complex functions without using sensitivities.

The purpose of the research is to introduce the PSO algorithm as a new option for this optimization tool and to compare its performance with respect to the existing ones being able to point out the scenarios where it is can be recommended for solving industrial applications.

5.3. Results
The PSO algorithm implemented for the simulations is based on the Eberhart and Shi work [2].

- **Analytic function**
At first the PSO implementation is validated on a simple test case base on test function Eq.(7). A study of the parameters of PSO, c₁ and c₂, has been carried out. Different population numbers have also been experimented to see which combination of parameters gives the best value and the lesser number of iterations.

\[ f(x_1, x_2) = x_1^2 + x_2^2 + x_1 x_2 - 3x_1 - 6x_2 + 1 \]  \hspace{1cm} (7)

The minimum of this function is in \( x^*=(0,3) \) and its value is \( f^*=-8 \).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Population size (s)</th>
<th>( c_1 )</th>
<th>( c_2 )</th>
<th>( f^* )</th>
<th>( x^* )</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO</td>
<td>10</td>
<td>1</td>
<td>0.5</td>
<td>-8</td>
<td>(0.3)</td>
<td>1463</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.5</td>
<td>2.5</td>
<td>28.82</td>
<td>(-3.222,10)</td>
<td>1681</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1</td>
<td>0.5</td>
<td>-8</td>
<td>(0.3)</td>
<td>3341</td>
</tr>
<tr>
<td>Evolutionary</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-8</td>
<td>(0.3)</td>
<td>251</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>-</td>
<td>-</td>
<td>-8</td>
<td>(0.3)</td>
<td>901</td>
</tr>
</tbody>
</table>

As it is shown in table 1, PSO requires more function evaluations than the genetic algorithm, probably due to that the version of PSO used in BOSS Quattro does not have any enhancements, and that the study of the stopping criterion must be also introduced. The fact that the initial populations chosen by BOSS Quattro are rather different form the one used for PSO should be taken into consideration and makes a rigorous comparison difficult.

Performing the calculations with a Matlab code and the population size of 10, \( c_1=1 \) and \( c_2=0.5 \), the number of evaluations turns around 800, much less than in BOSS Quattro, and with a population of \( s=20 \) the number of iterations until convergence increases until 1300.

- **Laminates**
For the laminates, the objective function is the strain energy of the ply which needs to be minimized. The variables are the ply orientations, \( \theta \), considering them between the bounds of \(-45^\circ\) and \(180^\circ\). In order to simplify the test case,
the only values that orientations can take will be: 0°, 90° and +/-45°. The reference axis and orientations for all the simulations are shown in figure 2.

![Coordinate system](image)

Figure 2. Coordinate system

1) Laminate with 1 ply
The material of the laminate considered in this article is graphite/epoxy. Table 2 shows the material data taken from Ref. [9].

Table 2. Data of graphite/epoxy

<table>
<thead>
<tr>
<th>E₁(GPa)</th>
<th>E₂(GPa)</th>
<th>G₁₂(GPa)</th>
<th>ν (Poisson ratio)</th>
<th>Ply thickness(mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>181</td>
<td>10.3</td>
<td>7.17</td>
<td>0.28</td>
<td>1</td>
</tr>
</tbody>
</table>

According to the theory of laminates [8], the objective function to minimize is the strain energy Eq. (8).

\[
u = \frac{1}{2} \varepsilon^T A \varepsilon
\] (8)

Being \( \varepsilon \) the strain, and \( A \) the extensional stiffness matrix. The strain energy can be written as well as

\[
C = 2u
\] (9)

where \( C \) is the compliance, which is the inverse of the stiffness matrix. Minimizing the strain energy or the compliance is therefore equivalent to maximizing the stiffness of the laminate.

The extensional stiffness of the laminate, matrix \( A \), is defined as follows:

\[
A_{ij} = \sum_{k=1}^{n} \left[ Q_{ij} \right]_k (h_k - h_{k-1})
\] (10)

The load case will be \( N_x = 1000 \text{ N/m} \), Figure 3a, and the results obtained are shown in table 3.

![Load case](image)

Figure 3. 3a) Load case for laminate with one ply; 3b) Load case for balanced laminate
10th World Congress on Structural and Multidisciplinary Optimization
May 19-24, 2013, Orlando, Florida, USA

Table 3. Results for load case for one ply laminate

<table>
<thead>
<tr>
<th>Software</th>
<th>Algorithm</th>
<th>$f_{optimum}$</th>
<th>Orientation of the ply (°)</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>PSO</td>
<td>0.0026</td>
<td>[0]</td>
<td>50</td>
</tr>
<tr>
<td>BOSS Quattro</td>
<td>PSO</td>
<td>0.002557</td>
<td>[0]</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>0.0029</td>
<td>[0]</td>
<td>121</td>
</tr>
</tbody>
</table>

In this case the two algorithms give the same results but with a change of number of iterations. In this case the PSO gives a faster convergence with a smaller number of function evaluations both in MATLAB or Boss Quattro implementations. In PSO, as in the evolutionary algorithms, the optimum orientation of the fibers is obviously 0°. This orientation yields the minimum strain energy and the minimum compliance so the maximum stiffness. As the load is applied along the $x$ axis, $N_x=1000$ N/m, $N_y=N_{xy}=0$, the maximum Young modulus being along the longitudinal in this case of a graphite/epoxy ($E_1>>E_2$) The material is used in the best way, i.e. optimized if the load is aligned with the fibers, that is fibers are oriented to 0° with respect to the $x$-axis.

2) Balanced laminate with 2 plies
Because of the balancing of the laminate, only one design variable is required. The material and properties are the same as showed in Table 2, and the load case is shown in Figure 3b. The considered load includes both longitudinal and transversal components, $N_x=N_y=1000$ N/m. Here the objective function is the strain energy, which is minimized as in the previous case. The results are shown in Table 4.

Table 4. Results for symmetric laminate

<table>
<thead>
<tr>
<th>Software</th>
<th>Algorithm</th>
<th>$f_{optimum}$</th>
<th>Orientation of the ply (°)</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>PSO</td>
<td>0.0047</td>
<td>[+/-45]</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>0.00469</td>
<td>[+/-45]</td>
<td>101</td>
</tr>
</tbody>
</table>

In this application, PSO algorithm works really well and gives the same results as the evolutionary algorithms. As the loads have the same magnitude and are perpendicular, the best orientation is obvious 45°. If the load in the $x$ axis, $N_x$, is bigger than $N_y$, the orientation would be closer to 0°, in opposition, if the $N_y$ load were bigger than $N_x$, the orientation would be closer to 90°. This shows that the longitudinal Young Modulus, $E_1$, attached to the main axis of the lamina, is bigger than $E_2$, transversal Young Modulus, so it will define the orientation of the plies. As defined in the Handbook [8], for high axial loads most of the plies must be oriented parallel to loading direction.

3) Laminate with 8 plies
In this simulation 8 plies are considered. However, the method allows the use of more plies. It is a symmetric laminate. Here, only 4 variables are necessary due to the fact that the symmetric constraint is involved in the extensional stiffness matrix $A$, in which the position of each ply is taken into consideration. The material properties are the same as in the previous cases, and the load is only longitudinal, $N_x=1000$N/m. The conditions of the simulations are shown in Figure 4.

Figure 4. Load case for laminate with 8 plies

Here the objective function is the strain energy, which is minimized as in the previous case. The results are shown in Table 5.
Table 5. Results for laminate with 8 plies

<table>
<thead>
<tr>
<th>Software</th>
<th>Algorithm</th>
<th>f optimum</th>
<th>Orientation of the ply (°)</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>PSO</td>
<td>0.000332</td>
<td>[(0)₃₈]</td>
<td>360</td>
</tr>
<tr>
<td>BOSS Quattro</td>
<td>PSO</td>
<td>0.0003197</td>
<td>[(0)₃₈]</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>0.0003328</td>
<td>[(0)₃₈]</td>
<td>225</td>
</tr>
</tbody>
</table>

As expected, the same results are obtained. As the load is in direction of the x axis, the optimal orientation of the plies is in the orientation of $E_1$, so the optimum orientation is 0°.

It is needed to remind the reader that here no laws of stacking sequence has been considered. Until here the PSO only deals with unconstrained problems. A future perspective would be to work with constrained ones. This would allow, for example, to consider the stacking laws of the laminates and so to obtain results complying with the practical applications of the laminates.

6. Updating of the particles

At the moment of the particle position and velocity update, different versions have been studied. In the implementation of the different examples of this paper, the velocity and location of a random particle are updated for new iteration arrives.

Moving just the particle with the best position at each iteration does not lead to any useful point due to it is always the same particle, so the algorithm does not make use of the exploration in the design space. Moving each time the particle with the worst position leads to better conclusions, giving better solution than the random particle method. Two other options have been investigated; updating half of the population with the best positions or with the worse, and moving the ones with the worse positions gave a better result due to the algorithm improves always those particles that are in worse spots.

Finally, the updating of all the population at each iteration gave good convergence but in a larger time due to higher number of function evaluations. This option can be useful for problems with more than one local minimum to avoid the convergence to them since the exploration of the search space will be bigger.

7. Augmented Lagrange for PSO

When the objective function to optimize is subject to some constraints, the algorithm presented in the previous section must be modified. In the case of the PSO, the best solution is to use the Augmented Lagrangian approach. Considering the following problem:

$$\begin{align*}
\text{Minimize } & f(x) \\
\text{Subject to } & h_j \geq 0 \quad j=1,...,m \\
& x_i^l \leq x_i \leq x_i^u \\
& i = 1,..., n
\end{align*} \tag{11}$$

The upper and lower bounds (side) constraints are considered separately because they are simple explicit functions. There is no need to introduce Lagrangian multipliers associated to these side constraints in the search of the optimal minimum.

In order to avoid the ill-conditioning of the unconstrained problems in the Augmented Lagrangian function method, a penalty term is added to the Lagrangian function Eq.(12):

$$L(x, \lambda) = f(x) - \sum_{j=1}^{m} \lambda_j h_j(x) \tag{12}$$

As Morris [10] explains, if there exists some $\lambda^*$ for which $x^*$ solves the unconstrained problem $\min L(x, \lambda^*)$, while satisfying the constraints $\{h_j(x) = 0; j = 1, \ldots, m\}$, then $x^*$ is a solution to the original problem. Therefore the problem

$$\begin{align*}
\text{Minimize } & \quad L(x, \lambda) \text{ given a suitable } \lambda \\
\text{Subject to } & \quad h_j(x) = 0 \quad j=1,\ldots,m \tag{13}
\end{align*}$$

is equivalent to the original problem Eq.(11) since $x^*$ is a local minimum and $\lambda^*$ is the associated Lagrangian multiplier vector. In order to form the final augmented Lagrangian function $L(x, \lambda)$, it is necessary to add a penalty function transformation for the equality constrained problem:
Given values of $\lambda$ and $r$, the multiplier method consists in applying the algorithm for unconstrained minimization to the Eq.(14), yielding a primal-dual point $x(\lambda, r)$ converging to $x^*$ as the iterations proceed. The multiplier method can easily be extended to deal with inequality constraints by converting them to equality constraints through the use of slack variables [11]. It turns out that these additional variables finally disappear from the augmented Lagrangian function, finally being:

$$L_A(x, \lambda, r_p) = f(x) - \sum_{j=1}^{m} \lambda_j h_j(x) + \frac{r_p}{2} \sum_{j=1}^{m} [h_j(x)]^2$$  \hspace{1cm} (14)$$

with

$$m_j(x, \lambda, r_p) = \begin{cases} h_j^2(x) - r_p \lambda_j h_j(x) & \text{if } h_j(x) \leq \frac{r_p}{2} A_j \\ -\left(\frac{r_p}{2} A_j\right)^2 & \text{if } h_j(x) > \frac{r_p}{2} A_j \end{cases}$$  \hspace{1cm} (16)$$

The update of the Lagrange multipliers is performed according to Eq.(17), with $v$ being a sub-problem:

$$\lambda_j^{v+1} = \max \left\{ 0, \lambda_j^v - \frac{2}{r_p^v} h_j(x^v) \right\}$$  \hspace{1cm} (17)$$

The update of the penalty factor is performed according to Eq.(18):

$$r_p^{v+1} = \begin{cases} 2 r_{p,j}^v & \text{if } h_j(x^v) > \varepsilon \\ \frac{1}{2} r_{p,j}^v & \text{if } h_j(x^v) \leq \varepsilon \\ r_{p,j}^v & \text{otherwise} \end{cases}$$  \hspace{1cm} (18)$$

As presented in Jansen and Perez [12], the update is done in such a way that it increases the penalty for infeasible movements, and that it is reduced if the constraint value is below a specific constraint violation tolerance $\varepsilon$.

Another criteria taken into consideration is to impose a lower bound for the penalty factors, which maintains an effective change in Lagrange multipliers update. The lower bound proposed is the following:

$$r_{p,j} \geq \frac{1}{2} \left[ \frac{1}{\varepsilon_{p,j}} \right]$$  \hspace{1cm} (19)$$

- **PSO and Lagrange**

The augmented Lagrange multiplier method described before can be combined with the PSO algorithm. As the correct Lagrange multipliers $\lambda^*$ and the required magnitude of the penalty factors $r_p$ are unknown, a sequence of unconstrained problems, defined by Eq.(15), must be solved sequentially. This direct approach requires a complete unconstrained minimization step with respect to $x$ before performing any update of the Lagrange multipliers and penalty factors. The augmented Lagrangian particle swarm optimization comprises the following steps:

1. Set $v=0$, $k=0$, $\lambda_0^v=0$, $r_{p,j}^0=r_{p0}$ and initialize particles with predefined or random positions. Evaluate corresponding function values according to Eq.(15) at each position.
2. Check termination criteria. If satisfied, the algorithm terminates with the solution $x^* = x^v = gbest$, $\lambda^* = \lambda^v$.
   If $v > v_{\text{max}}$, the algorithm stops with failure.
3. Solve unconstrained problem (17) according to original PSO algorithm with a limited number of iterations $k_{\text{max}}$.
4. Update $r_p$ and $\lambda$ according to (17) and (18) where $x^v$ is the solution of the $v$ -the subproblem solved in step 3. Set $v=v+1$ and $k=0$.
5. Go to step 2.

In order not to have only a stationary point at $x^*$ but also a minimum, a heuristic update scheme is applied for the
penalty factors \( r_p \). If the intermediate solution \( x^\nu \) of step 3 is not closer to the feasible region defined by the j-th constraint than the previous solution \( x^{\nu-1} \), the penalty factor \( r_p, j \) is increased. On the other hand, \( r_p, j \) is reduced if the j-th constraint is satisfied with respect to user defined tolerances.

- **Stopping criterion**

For the benchmark constrained problems, some stopping criteria have been added to the method. The number of iterations for the Lagrange functions, which is the number of Lagrange functions tested until convergence is reached, is fixed to a maximum of 50. The number of iterations for each Lagrange function is limited by a maximum number that can be between 100 and 5000, although with 1000 usually it is enough to obtain good results.

In order to be sure that the solution founded is stable a counter has been introduced to the algorithm. It gives as good a solution when the difference between \( n \) consecutive solutions is less than a tolerance.

### 7.1. Numerical applications with constrained problems

In this point some closed-form examples with different kinds of constraints has been tested with a Matlab code.

- **Case 1.** 1 equality constraint

  Problem to optimize:
  \[
  f_1(x_1, x_2) = x_1^2 + (x_2 - 1)^2 \\
  s.t. \quad g_1(x_1, x_2) = x_2 - x_1^2 = 0 \\
  -1 \leq x_i \leq 1, \quad i = 1, 2
  \]  
  (20)

- **Case 2.** 1 equality constraint and 1 inequality constraint

  Problem to optimize:
  \[
  f_2(x_1, x_2) = x_1^2 + x_2^2 \\
  s.t. \quad g_1(x_1, x_2) = x_1 - 3 = 0 \\
  h_1(x_1, x_2) = 2 - x_2 \leq 0 \\
  -10 \leq x_i \leq 10, \quad i = 1, 2
  \]  
  (21)

- **Case 3.** 2 inequalities

  Problem to optimize:
  \[
  f_3(x_1, x_2) = (x_1 - 10)^3 + (x_2 - 20)^3 \\
  s.t. \quad h_1(x_1, x_2) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \\
  h_2(x_1, x_2) = -(x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0 \\
  13 \leq x_1 \leq 100 \quad ; \quad 0 \leq x_2 \leq 10
  \]  
  (22)

### Table 6. Optimum value for the objective function \( f_{opt} \), optimum values of Lagrange multiplier \( \lambda_{opt} \), optimum positions \( (x_1, x_2)_{opt} \)

<table>
<thead>
<tr>
<th>Case</th>
<th>Iterations</th>
<th>( f_{opt} )</th>
<th>( \lambda_{opt} )</th>
<th>( x_{opt} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>0.75</td>
<td>[1]</td>
<td>[0.7, 0.5]</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>13</td>
<td>[-6.4]</td>
<td>[3.2]</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>-7951</td>
<td>[1.79, -423.29]</td>
<td>[13.6, 0]</td>
</tr>
</tbody>
</table>

These results presented to solve optimization problems with equality or inequality constraints show convincing results regarding to convergence properties and robustness. These results have been compared to the ones given in Sedlaczek and Eberhard [22]. This new approach is a good alternative reliable technique which can be used to solve constrained problems as the encountered while working with the design of composite materials.

### 8. Conclusion

In this paper the algorithm Particle Swarm Optimization has been presented as an alternative method to optimize the composite materials in order to optimize composite structures and laminates. One of the objectives of the research is to introduce the PSO algorithm as a new variant of algorithms in the BOSS Quattro optimization toolbox. Some improvements presented by different authors have been introduced and some tests have been performed including some of those improvements. For the tests a comparison between PSO and Evolutionary algorithms has been carried out. For the purpose the software BOSS Quattro has been used. First, an analytical function has been tested. Then some composite laminates have been optimized using the orientations of the plies as
design variables. In this case, the strain energy has been minimized. The numerical results obtained are consistent with the embedded simplified physics, but the stacking sequence laws must be introduced for future enhancements and industrial applications. The behavior of the PSO has been compared to other evolutionary algorithms giving convincing results. The common points between PSO and the Genetic Algorithms used is that they all use random initial population, that the use of a fitness value to evaluate the population and that the updating and search are done by random techniques. The differences of these two kinds of algorithms are that PSO does not use genetic operators (mutation or crossover), that the PSO update is done by the internal velocity and that the GA population moves like a group towards the optimal area while in PSO only the best particle gives out information to the others. The constrained problem has been also studied. For the purpose, the Augmented Lagrangian method with PSO has been presented, with equalities or inequalities constraints, giving convincing results. One of the future objectives will be the introduction of this method in BOSS Quattro to allow PSO to work with constrained problems. Other improvement that will be introduced, and that has been already studied and presented in this paper, will be the option of choosing which particles to update at each iteration depending of the necessities of the problem.

9. References