IMPROVED PARTICLE SWARM OPTIMIZATION FOR NON-LINEAR PROGRAMMING PROBLEM WITH BARRIER METHOD

Raju Prajapati
1University Of Engineering & Management, Kolkata, W.B. India

Om Prakash Dubey2, Randhir Kumar3
2,3Bengal College Of Engineering & Technology, Durgapur, W.B. India

email: raju.prajapati20102011@gmail.com

Article History: Submitted on 10th September, Revised on 07th November, Published on 30th November 2017

Abstract: The Non-Linear Programming Problems (NLPP) are computationally hard to solve as compared to the Linear Programming Problems (LPP). To solve NLPP, the available methods are Lagrangian Multipliers, Sub gradient method, Karush-Kuhn-Tucker conditions, Penalty and Barrier method etc. In this paper, we are applying Barrier method to convert the NLPP with equality constraint to an NLPP without constraint. We use the improved version of famous Particle Swarm Optimization (PSO) method to obtain the solution of NLPP without constraint. SCILAB programming language is used to evaluate the solution on sample problems. The results of sample problems are compared on Improved PSO and general PSO.

Keywords: Non-Linear Programming Problem, Barrier Method, Particle Swarm Optimization, Equality and Inequality Constraints.

INTRODUCTION

A large number of algorithms are available to solve optimization problems, inspired by evolutionary process in nature. e.g. Genetic Algorithm, Particle Swarm Optimization, Ant Colony Optimization, Bee Colony Optimization, Memetic Algorithm, etc. In general, these algorithms require computational software for processing and working over a particular optimization problem. PSO is a meta-heuristics algorithm, which was developed by Kennedy and Eberhart in 1995 [1].

The process of PSO is easily understood by the following example: Consider a flock of birds or fish are searching for food on a particular specific area. If a bird/fish can smell the position of food at a particular point, it is assumed to be closest with the food. It can inform the other bird/fish for that, and finally the search space get smaller. Ultimately, the search space get smaller and smaller and food could be found. The PSO works on the same process of nature.

Following this, we may generate several random solutions within a given space/interval. If a solution is very close to the solution, we consider the fitness value of that particular solution as very high. We declare it a ‘global best’ solution. It is denoted by ‘gbest’. We fix it, and move to find a better solution within the search space. For a particular iteration we have one ‘pbest’ solution also. It is giving the best of a particle’s position. If this is better than ‘gbest’, we update the gbest by pbest and move to the next iteration. Ultimately the solution could be found by a large number of iteration or fixing the error amount with the final solution.

It is observed that, the PSO algorithm generally applied to function optimization, machine study, neural network training, the signal processing, system control, image classification etc. [2, 3, 4, 5].

In this paper, we are working over multivariate non-linear optimization problem, particularly the problems with equality constraints. These problems needed to be converted to NLPP with equality constraint, using barrier method, and then improved PSO is used to obtain the solution of the problem. Here, SCILAB software is used to demonstrate the approximate improvement on computation due to improved PSO, as compared to the case of general PSO. The algorithm has also been demonstrated with sample NLPP with equality constraints.

The basic Particle Swarm Optimization consists of ‘n’ particles/solutions, which are initialized randomly over the given searched space.

PARTICLE SWARM OPTIMIZATION ALGORITHM

The basic Particle Swarm Optimization consists of ‘n’ particles/solutions, which are initialized randomly over the given searched space. The goal is to find the position, where the maximum or minimum fitness of particle(s) exist(s) i.e. the co-ordinates of a point, where maximum or minimum value of objective function exists. The position has been initialized randomly in the given searched space. These particles should converge towards the optimal solution by giving some “drift” along optimal solution. This ‘drift’ is termed as the ‘velocity’ of the particle. Therefore each particle, which has been initialized randomly in the given search space, move with some velocity towards the optimal solution.
Initially the position has been initialized randomly, while velocity is taken as zero (we can also assume random velocity initially). Two more terms are there, the ‘pbest’ and ‘gbest’.

- **pbest**: each particle keeps record of its own best position in the history of all iterations, which is called pbest.
- **gbest**: It is the best position achieved till now during all iteration by any of the particle out of ‘n’ particles.

**Use of Gbest And Pbest**

Each particle moves with some velocity and this velocity is updated by previous velocity, a fraction of gbest and a fraction of pbest.

The update in velocity could be given by the following equation,

\[ v_{id}^{k+1} = v_{id}^k + c_1 r_1^k (p_{best_{id}}^k - x_{id}^k) + c_2 r_2^k (g_{best_{id}}^k - x_{id}^k) \]  

...(1)

The position of each particle could be updated as,

\[ x_{id}^{k+1} = x_{id}^k + v_{id}^{k+1} \]  

...(2)

Where, \( v_{id}^k \) denotes the velocity of \( i \) th particle at the \( k \) th iteration. \( c_1 \) and \( c_2 \) are two constants, that we have to fix at the start of iteration. \( r_1^k \) and \( r_2^k \) are 2 random numbers taken from [0,1], \( x_{id}^k \) is the position of \( i \) th particle at \( k \) th iteration. \( p_{best_{id}}^k \) is the particle i’s best position till \( k \) th iteration, and \( g_{best_{id}}^k \) is the global best position of any particle till \( k \) th iteration.

The velocity of particle at any iteration depends on 2 factors: i) the pbest and ii) the gbest. If \( r_1^k \) is smaller and \( r_2^k \) is higher then the impact of gbest is higher, and correspondingly there is more chance of getting global optimal solution. In the same way, if \( r_1^k \) is higher, then we can get a local optimal solution more frequently.

At the end of each iteration, the position of each particle is updated by adding up the velocity of particle to the previous position.

**ADVANTAGES AND DISADVANTAGES**

Advantages of PSO algorithm:

- Since PSO algorithm is an evolutionary process (a meta-heuristic) inspired by nature, the algorithm could be equally applicable to all optimization problems.
- Calculation part is very easy and straightforward. Therefore, vast number of problem could be solved.
- Unlike Genetic Algorithm (GA), the PSO skips the steps of mutation/overlapping. The speed of searching is very fast, as each particle is taking information from best particle (gbest).

Disadvantages of PSO algorithm:

- Depends on random number \( r_1^k \) and \( r_2^k \). If \( r_1^k \) and \( r_2^k \) both are very low, a very small change could be found in the next iteration (i.e. \( k + 1 \)th iteration).
- Could not guarantee a global optimal solution in the given searched space [15].

**IMPROVED PSO ALGORITHM**

Despite of exponentially large number of paper on PSO, some improvement have been made on PSO algorithm. The main improvement includes: (i) Inertia weight, (ii) optimal selection of particles, (iii) convergence factor, (iv) proper random number generator, and (v) blending the PSO with other algorithms.
Inertia weight: An inertia weight ‘w’ is a constant agent, which is to be multiplied with the velocity of the previous iteration in the equation for next iteration velocity [5]. The equation for velocity update after applying inertia weight is given by,

$$v_{id}^{k+1} = wv_{id}^k + c_1r_1^k (pbest_{id}^k - x_{id}^k) + c_2r_2^k (gbest_{id}^k - x_{id}^k)$$  \(\text{(3)}\)

Inertia weight gives the impact of last speed to the present speed. In general w is taken to be 1, if it is not applied. But if we take inertia weight to be between 0.9 to 1.2, the convergence is very fast [6]. The inertia weight affects the searching ability of the swarm in the given space. Larger the inertia weight, the particles will have a tendency to explore the whole space, and similarly, lower inertia weight gives higher local search ability.

Optimal Selection of Particles: If we select the particle at the initial stage, very close to the optimal solution, then very low number of iteration is sufficient to solve the problem, and then we have a computational advantage. If the robust problem, or the problem where computation is complex, we can choose arbitrary large number of particles at the initial stage. The compound PSO put forward by Angeline in 1999 [7], is based on the basic selection mechanism, giving the computational advantage. In that method, we use the better particle obtained in one iteration for the next iteration.

Constriction Factor: It was given by Eberhert & Shi in 2000 [8]. If the velocity at the next iteration is influenced by a constriction factor then,

$$v_{id}^{k+1} = \chi (v_{id}^k + c_1r_1^k (pbest_{id}^k - x_{id}^k) + c_2r_2^k (gbest_{id}^k - x_{id}^k))$$  \(\text{(4)}\)

Where, \(\chi = \frac{2}{2 - \phi - \sqrt{\phi^2 - 4\phi}}\), \(\phi = c_1 + c_2 > 4\), then the convergence speed is much faster. In general \(\phi\) is taken to be 4.1. Then the constant \(\chi\) is equal to 0.729. When this method is compared with the inertia weight type of improved method, the convergence was found much higher. When a proper values of \(w\). \(c_1\) and \(c_2\) is used, the constriction factor gives same improvement as the weight factor is. The particle swarm optimization with constriction factor may be assumed as a special case of PSO with applied weight factor.

Proper random number generation: If we use the general PSO with 2 random numbers \(r_1^k\) and \(r_2^k\), the 2 number, at the same time may give low or high values. If both are closer to 0, then there is no more improvement in the next iteration. Also, If the fraction \(r_1^k\) is very small and \(r_2^k\) is large, then we have more impact of global best particle over solution, and correspondingly, the global maxima or minima would be possible as a result. If the fraction \(r_1^k\) is large and \(r_2^k\) is least, there will be an impact of pbest on the solution, and there will be a possibility of obtaining local solution rather then global minima/maxima [9].

Therefore, for improvement, we came across a newer equation,

$$v_{id}^{k+1} = v_{id}^k + c_1r_1^k (pbest_{id}^k - x_{id}^k) + c_2(1-r_1^k)(gbest_{id}^k - x_{id}^k)$$  \(\text{(5)}\)

Here, only one random number \(r_1\) is used. Further, we have to generate only one random number \(r_1\) in each iteration. The above equation will take care of all the deficiency discussed above.

Blending of PSO with other algorithm: There are several evolutionary algorithms such as Genetic Algorithm, Memetic Algorithm, Ant Colony Optimization (ACO), Simulation Anneling and many more. The new algorithm is called hybrid algorithm. In paper [10], hybrid of Genetic Algorithm and PSO is discussed. In paper [11], Simulation Anneling is used along with PSO, to achieve better results. In paper [12], improved PSO and ACO have been compared.

BARRIER METHOD

Suppose we are given an NLP with constraint

$$\text{Min. } f(x),$$

Subject to, \(g(x) \leq 0; \text{ and } h(x) = 0;\)
Then we could convert the NLPP with constraint to an NLPP without constraint by imposing a barrier function on the objective part. This barrier could be prepared with the help of constraints \[19\]. Therefore, our NLPP becomes,

\[
\text{Min. } f(x) + \frac{r}{g(x)} + \frac{(h(x))^2}{\sqrt{r}}, \quad \text{where } r \text{ is a constant used to impose a barrier.}
\]

This \( r \) could ultimately be minimized to strengthen the barrier. In our computation, we gradually takes \( r = 1, 0.1, \) and ultimately stops at \( r = 0.01. \)

**USE OF BARRIER METHOD**

The method is basically used to convert the Non Linear Programming Problem with constraint to Non Linear Programming Problem without constraint, which makes the entire computation easier. We only have to take care of a constant called ‘barrier constant’, which is not a big deal throughout the computation. This barrier constant works for changing the convexity of entire function, higher convex region gives you a smaller region, in which the search space is minimized. Now, depending upon the software available to us, we can increase as much barrier as we can.

**METHODOLOGY**

In the present paper, we are going to apply the Improved PSO algorithm over Non-Linear Constrained Optimization Problem (NL COP). Although a large number of works has been done over this, but the PSO over NL COP is still a matter of interest \[13, 14, 17, 18\]. Some work over NL COP with equality constraints are done in \[16\], in which constrains are removed by mixed variable partioning. In our work, we are using a different approach.

There are various methods which could convert the NLP with constraints to NLP without constraints e.g. Penalty method, Barrier method, Lagrangian multiplier, sub gradient method, etc. Here, we use Barrier method \[19\] to convert to problem as NLP without constraint, and then we apply Improved PSO method to find the solution of the problem.

The PSO method used here consist of 3 updates in the original equation. Let’s consider the PSO velocity and position update formula below, from equation (4),

\[
v_{id}^{k+1} = \chi \{v_{id}^{k} + c_1 r_1^{k} (p_{best_{id}}^{k} - x_{id}^{k}) + c_2 r_2^{k} (g_{best_{id}}^{k} - x_{id}^{k})\}
\]

where, \( \chi \) is a weighing factor.

or,

\[
v_{id}^{k+1} = \chi v_{id}^{k} + \chi c_1 r_1^{k} (p_{best_{id}}^{k} - x_{id}^{k}) + \chi c_2 r_2^{k} (g_{best_{id}}^{k} - x_{id}^{k})
\]

or,

\[
v_{id}^{k+1} = \chi v_{id}^{k} + k_1 r_1^{k} (p_{best_{id}}^{k} - x_{id}^{k}) + k_2 r_2^{k} (g_{best_{id}}^{k} - x_{id}^{k}).
\]

The main update is in random number \( r_2 \), which is taken to be \( 1 - r_1 \). The benefits we are getting with this change in NLP are already discussed in previous section.

The weighting factor for \( 1^\text{st} \) term \( \chi \) is taken to be zero in our computational experiment, and remain unchanged for \( 2^\text{nd} \) and \( 3^\text{rd} \) term (\( k_1 \) and \( k_2 \) remain unchanged), which gives a velocity in each iteration, which is independent of the velocity in previous iteration. The only thing it’s using is the effect of \( p_{best} \) and \( g_{best} \) obtained in the past. The velocity update formula is, therefore,

\[
v_{i}^{k+1} = k_1 r_1^{k} (p_{best_{i}}^{k} - x_{i}^{k}) + k_2 (1 - r_1^{k}) (g_{best_{i}}^{k} - x_{i}^{k})
\]

\[\ldots (6)\]

In our experiment, we have taken the constant \( k_1 \) and \( k_2 \) as 2 each. \( \chi = 0 \) for \( 1^\text{st} \) term, and \( r_2^{k} = 1 - r_1^{k}, \) (where \( k_1 = \chi c_1 \) and \( k_2 = \chi c_2 \)).

The position update of PSO is given by, from equation (2),

\[
x_{i}^{k+1} = x_{i}^{k} + v_{i}^{k+1}
\]
The main theme of this paper is to apply the equation (2) and (6) to NLPP with equality constraint, using barrier method. The entire computational work has been done in SCILAB software.

**COMPUTATIONAL RESULTS**

We have implemented our improved PSO over five test functions with equality constraints. We are intentionally taking smaller problems, as they could also be worked out manually. We can also test other type of functions in our developed SCILAB program. The sample test functions are as follows:

**Test function 1:** Minimize \( x + y \),

Subjected to \( x^2 + y^2 = 1 \).

The solution is -1.414, at \( x = -0.71 \), and \( y = -0.71 \) (Evaluated manually).

**Test function 2:** Minimize \( x^2 / 9 + y^2 / 4 \)

Subject to \( x + y = 1 \).

The solution is 0.077 at \( x = 0.7 \) and \( y = 0.3 \) ( Evaluated manually).

**Test function 3:** Minimize \( x^2 + y^2 \)

Subject to \( (x - 2)^2 + (y - 2)^2 = 1 \).

The solution is 3.3282 at \( x = 1.29 \), \( y = 1.29 \) (Evaluated manually).

**Test function 4:** Minimize \( y - x^2 \)

Subject to \( x = y^2 \).

The solution is -18 at \( x = 4 \), and \( y = -2 \) (Evaluated manually).

**Test function 5:** Minimize \( y - x^2 \)

Subject to \( x^2 + y^2 = 1 \).

The solution is -1.25 at \( x = -0.86 \) and \( y = -0.512 \) (Evaluated manually).

The barrier function used for sample test functions are as follows:

For Test function 1; Barrier function, \( f = x + y + \frac{(x^2 + y^2 - 1)^2}{.01} \).

For Test function 2; Barrier function, \( f = \frac{x^2}{9} + \frac{y^2}{4} + \frac{(x + y - 1)^2}{.01} \).

For Test function 3; Barrier function, \( f = x^2 + y^2 + \frac{(x-2)^2 + (y-2)^2 - 1)^2}{.01} \).

For Test function 4; Barrier function, \( f = y - x^2 + \frac{(x-y)^2}{.01} \).

For Test function 5; Barrier function, \( f = y - x^2 + \frac{(x^2 + y^2 - 1)^2}{.01} \).

The results evaluated manually for each test functions are very close to the results obtained by running the program in SCILAB.
Four tables are prepared to analyze the results, having variation in \( r \) (for barrier function) and \( r_1, r_2 \) (as random number), i.e. there are two cases of barrier function \( r = .1 \) and \( r = .01 \), for checking the difference. Also The difference is to be checked, when we change the random number from \( r_2 \) to \( 1 - r_1 \). We take, Minimum \( x = -4 \), Maximum \( x = 4 \), and Minimum \( y = -4 \), Maximum \( y = 4 \) in each of the following cases.

### Table 1. Case 1: \( r = 0.1, r_1 \) and \( r_2 \) are distinct

<table>
<thead>
<tr>
<th>Test Function</th>
<th>No. of Particles</th>
<th>No. of Iterations</th>
<th>X value (Optimal)</th>
<th>Y value (Optimal)</th>
<th>Function Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>800</td>
<td>-0.7206045</td>
<td>-0.7179077</td>
<td>-1.4264974</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>800</td>
<td>0.6914522</td>
<td>0.3009966</td>
<td>0.0763428</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>800</td>
<td>1.2631644</td>
<td>1.263246</td>
<td>3.2648765</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>800</td>
<td>3.9997694</td>
<td>-1.9992603</td>
<td>-17.997341</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>800</td>
<td>-0.8940368</td>
<td>-0.5006396</td>
<td>-1.2749996</td>
</tr>
</tbody>
</table>

### Table 2. Case 2: \( r = 0.1, \) and \( r_2 = 1 - r_1 \)

<table>
<thead>
<tr>
<th>Test Function</th>
<th>No. of Particles</th>
<th>No. of Iterations</th>
<th>X value (Optimal)</th>
<th>Y value (Optimal)</th>
<th>Function Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>800</td>
<td>-0.7192910</td>
<td>-0.7192891</td>
<td>-1.4265001</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>800</td>
<td>0.6870229</td>
<td>0.3053435</td>
<td>0.0763359</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>800</td>
<td>1.2632084</td>
<td>1.2632084</td>
<td>3.2648765</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>800</td>
<td>3.9955221</td>
<td>-2.0019701</td>
<td>-17.966439</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>800</td>
<td>0.8943997</td>
<td>-0.5000587</td>
<td>-1.275</td>
</tr>
</tbody>
</table>

### Table 3. Case 3: \( r = 0.01, r_1 \) and \( r_2 \) are distinct

<table>
<thead>
<tr>
<th>Test Function</th>
<th>No. of Particles</th>
<th>No. of Iterations</th>
<th>X value (Optimal)</th>
<th>Y value (Optimal)</th>
<th>Function Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>800</td>
<td>-0.6743522</td>
<td>-0.7383912</td>
<td>-1.4119819</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>800</td>
<td>0.7119151</td>
<td>0.2880848</td>
<td>0.0770619</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>800</td>
<td>1.3028473</td>
<td>1.2767257</td>
<td>3.3358075</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>800</td>
<td>3.9992172</td>
<td>-2.0022126</td>
<td>-17.986822</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>800</td>
<td>-0.8944272</td>
<td>-0.5000000</td>
<td>-1.275</td>
</tr>
</tbody>
</table>

### Table 4. Case 4: \( r = 0.01, r_2 = 1 - r_1 \)

<table>
<thead>
<tr>
<th>Test Function</th>
<th>No. of Particles</th>
<th>No. of Iterations</th>
<th>X value (Optimal)</th>
<th>Y value (Optimal)</th>
<th>Function Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>800</td>
<td>-0.7083535</td>
<td>-0.7083535</td>
<td>-1.4154614</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>800</td>
<td>0.7298666</td>
<td>0.2679624</td>
<td>0.0776118</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>800</td>
<td>1.2784088</td>
<td>1.3008608</td>
<td>3.3355729</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>800</td>
<td>3.9981511</td>
<td>-1.995867</td>
<td>-17.95957</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>800</td>
<td>0.8636842</td>
<td>-0.5092981</td>
<td>-1.2524023</td>
</tr>
</tbody>
</table>
Following are the graphs of barrier functions for each test function (NLPP with constraints):

![Figure 1. Figure of sample test function 1](image1.png)

![Figure 2. For sample test function 2](image2.png)
CONCLUSION

It is observe that the result is sharper when we choose $r = 0.01$ and $r_2 = 1 - r_1$ as barrier factor and random numbers respectively, because the barrier is high, which creates larger convex region, and the $2^{nd}$ random number is depending on $1^{st}$ giving the computational advantage (i.e. almost sure drift in each iteration). Our method could easily be implemented over the famous NLPP such as rastrigin function, banana function, rosenbrook function, square function, etc. with any number of equality constraints.

REFERENCES


