REAL ROOTS OF NONLINEAR SYSTEMS OF EQUATIONS THROUGH A METAHEURISTIC ALGORITHM

RAÍCES REALES DE SISTEMAS DE ECUACIONES NO LINEALES MEDIANTE UN ALGORITMO METAHEURÍSTICO

IVÁN AMAYA

B.Sc. Mechatronics Engineering, Ph.D. Engineering student, Universidad Industrial de Santander, iamaya2@gmail.com

JORGE CRUZ

Electronics Engineering student, Universidad Industrial de Santander, mrcrois@hotmail.com

RODRIGO CORREA

Ph.D. on Polymer Science and Engineering, Professor, Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, Universidad Industrial de Santander, crcorrea@uis.edu.co

Received for review March 22 th, 2011, accepted May 17th, 2011, final version May, 23 th, 2011

ABSTRACT: This article describes the use of a numeric strategy to perform a metaheuristic optimization for finding the real roots of a nonlinear equation system. A theorem that shows why it can be treated as an optimization problem is shown. Some two-, three-, and five-equation systems are used as examples of the strategy.

KEYWORDS: nonlinear equations, optimization methods, particle swarm optimization

RESUMEN: Este artículo describe una estrategia numérica de optimización metaheurística, utilizada para determinar las soluciones en el conjunto de los números reales de un sistema de ecuaciones no lineales. Se utiliza un teorema que demuestra la validez de conversión de un problema de solución de ecuaciones no lineales en otro de optimización. A título de demostración, se presentan algunos resultados para sistemas de dos, tres y cinco ecuaciones.

PALABRAS CLAVE: ecuaciones no lineales, métodos de optimización, optimización por enjambre de partículas

1. INTRODUCTION

A nonlinear equations system is defined, explicitly, by (1). It can also be defined in vectorial (or compact) notation by (2). Traditional numerical methods for the solution of these systems are based in algorithms that make use of matricial operations, whose elements belong to the derivatives of its functions. These direct methods pose several variations and approaches, but one of the most used, in engineering, is Newton-Raphson and its variations. Nevertheless, they take a long time to converge, and they only deliver one root (in its traditional form). Therefore, a tendency to migrate to more efficient methods is currently underway. One approach is to transform the solution of the nonlinear system into an optimization problem [1] whose mathematical foundation is included in this article. Most optimization methods, such as Newton's direct root method and its variants, also have an elevated computational cost, which is mainly due to the calculation and storage of the Jacobian (first order derivatives) and Hessian (second order derivatives) matrices that need to be evaluated at each candidate point for every iteration. A first approach proposed to reduce the memory requirements changed the matrices by an approximated one. Methods based on this change are known as quasi-Newton, and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is an example of them. It replaces the matrix with an expression that contains several matricial operations (additions, subtractions, products, divisions, and transposes), so the amount of memory required to obtain a new search direction is not vastly reduced [2]. Even so, all of these methods are easily trapped in local minima, or in saddle points, so a starting point near the solution is required, thus limiting its application in real life situations.

A new optimization strategy has appeared, known as *metaheuristic* algorithms. These make use of stochastic processes, and therefore they are not trapped by the presence of local minima. A method, whose popularity has been on the rise, is particle swarm optimization (PSO), developed in 1995 by Eberhart and Kennedy. It is based on imitating the behavior of animal flocks when looking for new food sources [3], and it is also a cooperative method, where all of its members (or *particles*) communicate better spots (or *optimums*) to the swarm [4]. During this research, PSO was used to find the real roots of nonlinear systems, due to its inherent complexity.

This article begins by presenting a mandatory justification, which allows for the transformation of a nonlinear system into an optimization problem. The validation process is performed by using, as an example, systems of two, three, and five equations.

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0. \\ f_2(x_1, x_2, \ldots, x_n) &= 0. \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0. \end{aligned} \tag{1}$$

$$\mathbb{F}(\mathbb{x}) = \mathbf{0} \tag{2}$$

2. FOUNDATIONS

Let equation (1) describe a system of n nonlinear equations with n unknowns, where f is a mapping of the n-dimensional $\mathbf{R}^n$ vector $\mathbb{x} = (x_1, x_2, \ldots, x_n)$ into the real axis $\mathbf{R}$. This is equivalent to considering a function $\mathbb{F}$ and mapping $\mathbf{R}^n$ into $\mathbf{R}^n$ by (3), where each function is a *coordinate function* of $\mathbb{F}$.

$$\mathbb{F}(\mathbb{x}) = (f_1(\mathbb{x}), f_2(\mathbb{x}), \ldots, f_n(\mathbb{x}))^T \tag{3}$$

Some required definitions are shown below. They are related to the continuity and differentiability of the functions of $\mathbf{R}^n$ in $\mathbf{R}^n$.

Definition 1: Let f be a function of $A \subset \mathbf{R}^n$ in $\mathbf{R}$. It is said that f has a limit L in $\mathbb{x}_0$,

$$\lim_{\mathbb{x} \to \mathbb{x}_0} f(\mathbb{x}) = L \tag{4}$$

if, given any number $\varepsilon > 0$ there exists a number $\delta > 0$ with the property of

$$|f(\mathbb{x}) - L| < \varepsilon \tag{5}$$

always that $\mathbb{x} \in A$ and

$$0 < \|\mathbb{x} - \mathbb{x}_0\| < \delta. \tag{6}$$

Definition 2: Let f be a function of $A \subset \mathbf{R}^n$ in $\mathbf{R}$; f is continuous in $\mathbb{x}_0 \in A$ if $\lim_{\mathbb{x} \to \mathbb{x}_0} f(\mathbb{x})$ exists and

$$\lim_{\mathbb{x} \to \mathbb{x}_0} f(\mathbb{x}) = f(\mathbb{x}_0). \tag{7}$$

Moreover, f is continuous in A if f is continuous for every point of A.

DEFINITION 3: LET $\mathbb{F}$ BE A FUNCTION OF $A \subset \mathbf{R}^n$ IN $\mathbf{R}^n$ AS FOLLOWS:

$$\mathbb{F}(\mathbb{x}) = (f_1(\mathbb{x}), f_2(\mathbb{x}), \ldots, f_n(\mathbb{x}))^T \tag{8}$$

where, f_i is mapped from $\mathbf{R}^n$ into $\mathbf{R}$ for every *i*. It is then defined that

$$\lim_{\mathbb{x} \to \mathbb{x}_0} \mathbb{F}(\mathbb{x}) = \mathbb{L} = (L_1, L_2, \ldots, L_n)^T \tag{9}$$

if and only if $\lim_{\mathbb{x} \to \mathbb{x}_0} f_i(\mathbb{x}) = L_i$ for every $i = 1, 2, \ldots, n$.

There exist several numerical strategies that strive to solve this type of system, such as the fixed point and multidimensional Newton-Raphson approaches [5]. The first one rarely succeeds, while the other one (more famous in engineering) converges quickly only if the starting point is near the solution, thus reducing its application in real life situations and for bigger problems. Besides, it requires a lot of computer resources and normally finds real roots. However, if initialized with a complex point, it will return a complex solution. The elevated computer requirements are related to the need for calculating, evaluating, and storing the Jacobian matrix and the $n \times n$ system in all iterations. An improvement to such a disadvantage lies in the quasi-Newton algorithms, such as BFGS, where an approximation matrix is calculated instead of the Jacobian. Even if it lowers the computing requirements, its convergence speed is also reduced to the so called *superlinear convergence*. Further improvements include a redefinition of the inverse matrix, proposed by Sherman and Morrison [6]. There are also some other options, such as *homotopy* [7]. Recent literature

describes some of the most important methods for solving nonlinear systems [1], [8-11]

2.1 An optimization problem

A good approach, whose author cannot be tracked in time, is to transform a problem of solving a system of nonlinear equations into an optimization one [1,5,12-14]. Perhaps one of the most famous examples is the steepest descent method (including its variations), that has been vastly used (even though it only has linear convergence) to obtain the starting points for multidimensional Newton-Raphson [5]. Its main weakness, is the need to calculate the gradient, which can become a very time consuming task and, in some cases, an almost impossible one by analytic means. If done through commercial software, the computational requirements escalate because of the use of symbolic math. The theorem that allows the change into an optimization problem is presented below. It is mentioned for real roots, but its generalization into the complex realm is evident [15].

Theorem 1: Real roots

Let X be a subset of $\mathbf{R}^n$ and consider the system (1), where, for each i, f_i is a function whose domain contains X, and whose range is within real numbers. Let $f: X \mapsto \mathbf{R}$ be defined by (10). Note that f needs to be properly defined.

$$f(x) = \sum_{i=1}^{m} (f_i(X))^2 \qquad (10)$$

$$X = (x_1, x_2, \ldots, x_n)$$

besides:

Proposition 1. Suppose that (1) has solution in X and let $a = (a_1, a_2, \ldots, a_n) \in X$. Therefore, a satisfies (1) if, and only if, a minimizes f.

Proof. If a satisfies (1), then $f_i(a) = 0$ for each $i = 1,2,\ldots,n$. Therefore, $f(a) = 0$ and since $f(x) \geq 0$ for every $x \in X$, then a is a minimum for f.

Now, if a minimizes f but does not satisfy (1), then $f(a)$ must be a positive number since $f(x) \geq 0$ for every $x \in X$. Given that the system has a solution in X there exists an $x^* \in X$ that makes $f(x^*) = 0$ $f(x^*) = 0$ and $x^* \neq a$. Therefore, $f(x^*) < f(a)$ which violates a being the minimum for f. Note that the general condition on the consistency of the system is vital, since it is always possible to construct f for a given system and, if a minimizes it, it does not imply that a solution exists. Therefore, finding the roots for a system of nonlinear equations over a given set X can be transformed into an optimization problem (minimization for this case) of the function f over the set X. An algorithm containing this is as follows:

Algorithm 1

Input: The nonlinear equations system (1) and the set X

Step 1: Build f

Step 2: Minimize f over X.

Step 3: Let $a \in X$ be a minimum for f. If $f(a) = 0$ then a satisfies (1). Otherwise, it does not have solution in X.

Based on this theorem, a PSO algorithm was used to generate a real root of the system with a given precision of $1.0\mathbf{x}10^{-8}$, instead of using it to generate the starting point for Newton's direct root method. Some simulations with systems of two, three, and five nonlinear equations are presented in Section 3. It is important to remark that there is also evidence of other methods for solving nonlinear systems, but due to space restrictions, comparative results are not shown [16].

2.2 PSO

As mentioned, PSO appeared in 1995, thanks to Eberhart and Kennedy, who studied the social behavior of some animal groups which were looking for new sources of food. Unlike other evolutionary approaches (e.g., genetic algorithms), PSO is cooperative, sharing information with neighboring particles. Neighborhoods may have different topologies, so this is a key point for branches and variations. In its traditional form, the neighborhood is composed of all the particles

in the swarm, so every better point found will be communicated to them. Another key point is related to the way its basic equations (position and speed) are updated, traditionally given by (11) and (12), where i, j represent pointers for each position and time step, respectively; X is a particle's position, V its speed, w an inertia factor to limit the effect of its previous speed, C_1, C_2 are the self and swarm trust factors, R_1, R_2 are random numbers (uniformly distributed) between zero and one, P_{Best_i} is the best position each particle has found and G_{Best} is the best position of all the swarm.

$$X_i^{j+1} = X_i^j + V_i^{j+1} \tag{11}$$

$$V_i^{j+1} = wV_i^j + C_1R_1\left(P_{Best_i} - X_i^j\right) + C_2R_2\left(G_{Best} - X_i^j\right) \tag{12}$$

One way to implement this algorithm is:

1. Assign a random initial position and zero speed for each particle.
2. Evaluate f and find P_{Best_i}, G_{Best}.
3. Update the position and speed for each particle with (11) and (12).
4. Evaluate the objective function.
5. Compare, for each particle, the evaluated value and P_{Best_i}. If it is lower, then update P_{Best_i}.
6. Select the best particle and compare it to G_{Best}. If lower, then update G_{Best}.
7. Compare G_{Best} with convergence criteria. If it does not comply, return to 3.

3. Experiments and Results

3.1 Two-equation systems

3.1.1 System (13)

Figure 1 shows the results obtained after executing the PSO algorithm multiple times, starting at ($x_1 = -5$, $x_2 = 5$), ($x_1 = -100$, $x_2 = 5$) and ($x_1 = -5$, $x_2 = 100$), respectively. It can be seen that, by increasing the number of particles, the real root found is not critically affected, so, from now on, only total values will be shown for each starting point. This system has two real roots, located at ($x_1 = -2.73$, $x_2 = 11.36$) and ($x_1 = 1.54$, $x_2 = 10.95$), that are obtained even if starting at the same point. This behavior is due to the heuristic characteristic of PSO and allows it to avoid getting stuck at a solution.

$$f_1 = -x_1(x_1 + 1) + 2x_2 - 18$$
$$f_2 = (x_1 - 1) + (x_2 - 6)^2 - 25 \tag{13}$$

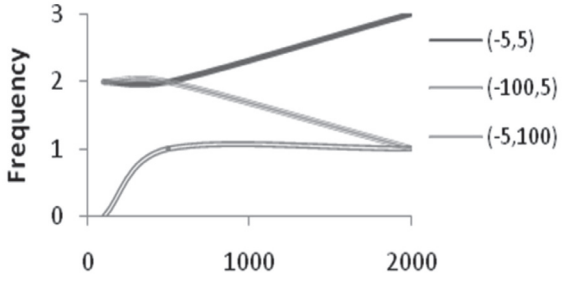

Figure 1. Roots distribution for each swarm size and starting point, system (13)

It is interesting to show that there seems to be a predilection for finding the closest roots, as can be deduced from checking against Figure 2, where in 2/3 of the cases, the solution given by PSO was the closest one. Figure 3 shows how the number of iterations varies with the size of the swarm. As the number of particles rises, it requires fewer cycles to converge. However, this reduction is lower for each increase, so there must be a minimum number of iterations, under which convergence is impossible to achieve (for a given margin of error).

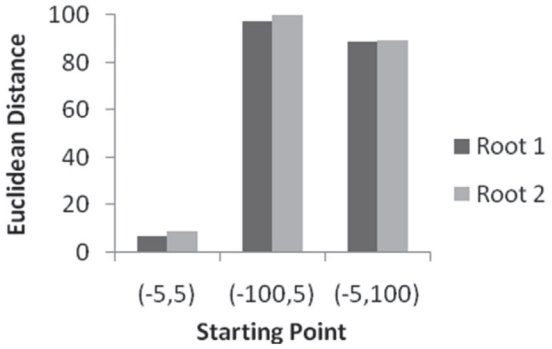

Figure 2. Euclidean distance from each starting point to real roots, system (13)

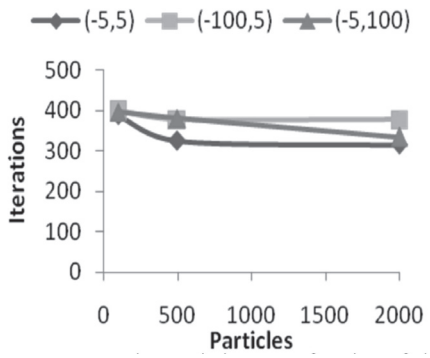

Figure 3. Iteration variation as a function of the swarm size, system (13)

3.1.2 System (14)

Figure 4 shows the results obtained after implementing PSO for system (14), whose roots are located at ($x_1 = 0.5$, $x_2 = 0.86$) and ($x_1 = 0.5$, $x_2 = -0.86$). Once again, beginning at the same point leads to a different solution. Nevertheless, it is also good to remark that beginning at different points can lead to the same real root.

$$f_1 = 3x_1^2 - x_2^2$$
$$f_2 = 3x_1x_2^2 - x_1^3 - 1 \quad (14)$$

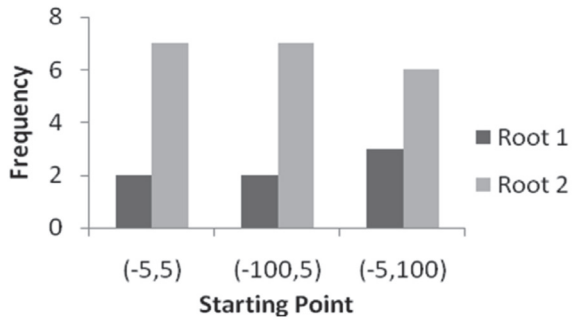

Figure 4. Real roots distribution for system (14)

Figure 5 shows the way computation time behaves as the swarm size increases. Unlike iterations, this variable directly increases with population. This is noteworthy, since one would expect that by having a lower number of iterations (Figure 3), convergence will be achieved faster. However, this could be explained as follows: by having more particles, the computational cost is increased, thus making it so that each iteration takes longer to complete. Figure 5 also shows that only for big swarm sizes (2000 particles), the starting point considerably affects the time required to converge.

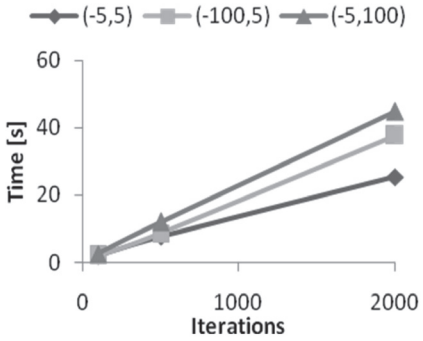

Figure 5. Computation time as a function of the swarm size and the starting point, system (14)

3.1.3 System (15)

Figure 6 shows the results after computing system (15), whose roots are located at ($x_1 = 1.26$, $x_2 = 6.6$), ($x_1 = 4.56$, $x_2 = -0.2$), ($x_1 = 0.47$, $x_2 = 1.52$), and ($x_1 = 4.47$, $x_2 = 2.41$). When compared to Figure 7, it can be seen that there seems to be a predilection to converge to closer roots. However, this does not mean that it only finds that root, but that it will appear more frequently. Figure 8 once again shows the variation in the number of iterations as the swarm gets bigger. In the same manner as shown previously, the bigger the swarm gets, the less iterations will be required to converge (up to a limit point for a given precision). Moreover, computation time will increase due to the excess of operations.

$$f_1 = 4x_1^2 - 20x_1 + \frac{1}{4}x_2^2 + 8$$
$$f_2 = \frac{1}{2}x_1x_2^2 - 2x_1 - 5x_2 + 8 \quad (15)$$

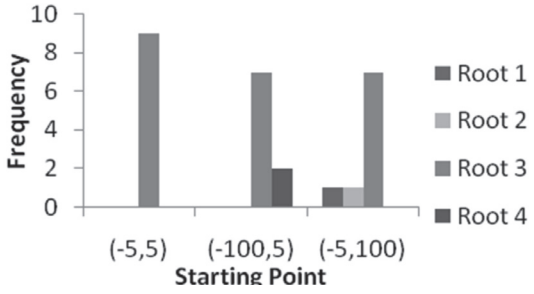

Figure 6. Real roots distribution for system (15)

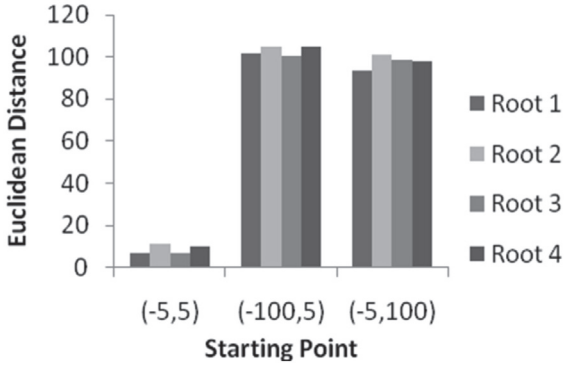

Figure 7. Euclidean distance from each starting point to real roots, system (15)

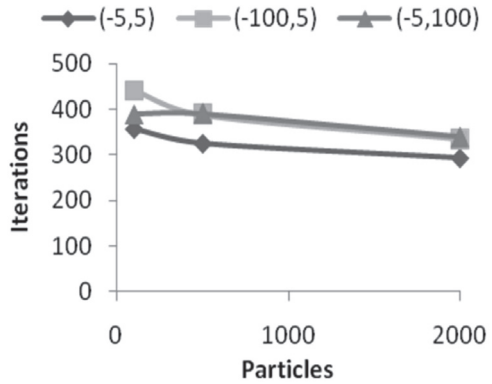

Figure 8. Iteration variation as a function of the swarm size, system (15)

3.1.4 System (16)

Figure 9 presents the results achieved for system (16), whose roots are located at ($x_1 = -1.87$, $x_2 = 3.12$), ($x_1 = 0.62$, $x_2 = 2.18$), ($x_1 = 2.11$, $x_2 = -1.33$), and ($x_1 = -4.86$ $x_2 = -3.96$). It is shown that the starting point does not affect the algorithm's convergence, even though Figure 10 confirms the predilection for closer real roots. Figure 11 shows the behavior of the approximation error as a function of the swarm size. It is easily seen that dependence exists on the number of particles and on the starting point. However, in average terms, an excess of particles will lead to a higher error. Therefore, it is of the utmost importance to choose an appropriate swarm size.

$$f_1 = x_1^2 - x_2^2 + 2x_2$$
$$f_2 = 2x_1 + x_2^2 - 6 \qquad (16)$$

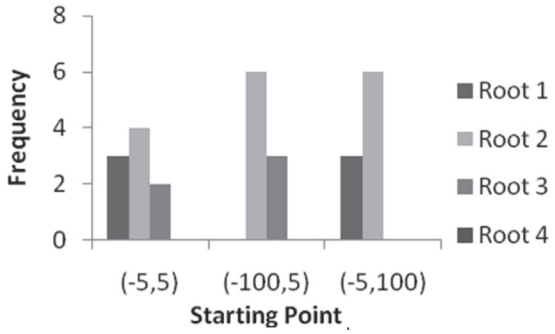

Figure 9. Real roots distribution for system (16)

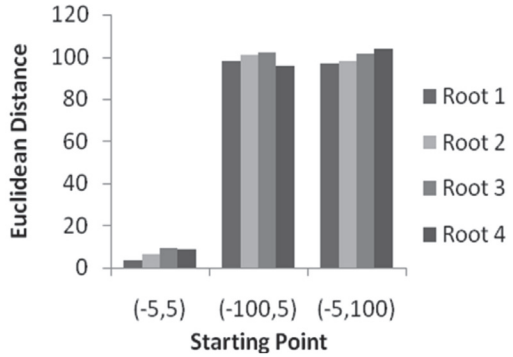

Figure 10. Euclidean distance from each starting point to real roots, system (16)

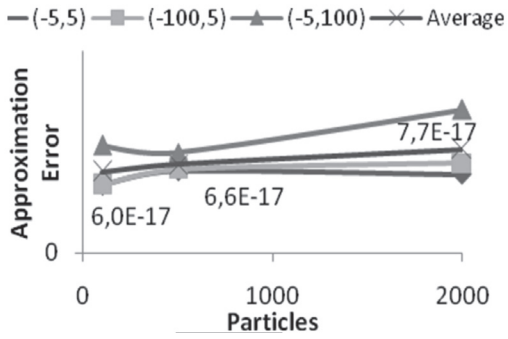

Figure 11. Square error variation as a function of the swarm size, system (16)

3.1.5 System (17)

Figure 12 plots the results after implementing system (17), whose roots are located at ($x_1 = -1$, $x_2 = 3.5$) and ($x_1 = 2.55$, $x_2 = 3.98$). Once again, unlike direct search methods, the starting point is not a restriction for the real root found as a solution. Even so, the predilection for closer roots is maintained.

$$f_1 = x_1(1 - x_1) + 4x_2 - 12$$
$$f_2 = (x_1 - 2)^2 + (2x_2 - 3)^2 - 25 \quad (17)$$

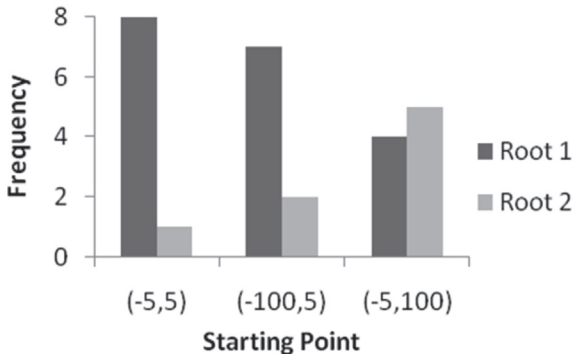

Figure 12. Real roots distribution for system (17)

Figure 13 allows corroboration of the behavior from Figure 5: the bigger the swarm, the lower the number of required iterations. However, computation time will be increased.

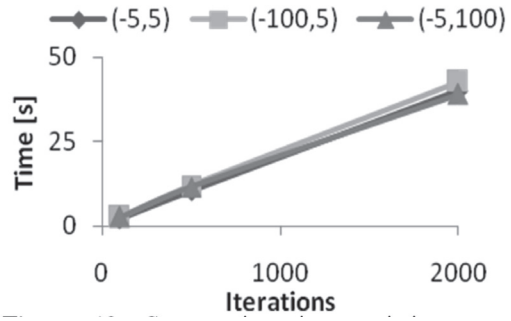

Figure 13. Computation time variation as a function of the swarm size, system (17)

3.2 Three-equation systems

Striving to analyze the combined behavior of systems (18)-(21), some plots are presented with relevant data. Figure 14 shows the variation on the computation time as a function of the number of iterations. It can be seen that it is normal to expect that a smaller system (i.e., one that performs more iterations) takes less time to converge. The relation between swarm size and number of iterations can be checked in Figure 15.

$$f_1 = x_1^2 + x_2 - 37$$
$$f_2 = x_1 - x_2^2 - 5 \quad (18)$$
$$f_3 = x_1 + x_2 + x_3 - 3$$

$$f_1 = x_1^2 + 2x_2 - x_2 - 2x_3$$
$$f_2 = x_1^2 - 8x_2^2 + 10x_3 - x_2^2 - 5 \quad (19)$$
$$f_3 = \frac{x_1^2}{7x_2x_3} - 1$$

$$f_1 = 10x_1 - 2x_2^2 + x_2 - 2x_3 - 5$$
$$f_2 = 8x_2^2 + 4x_3^2 - 9 \quad (20)$$
$$f_3 = 8x_2x_3 + 4$$

$$f_1 = 15x_1 + x_2^2 - 4x_3 - 13$$
$$f_2 = x_1^2 + 10x_2 - x_3 - 11 \quad (21)$$
$$f_3 = x_2^2 - 25x_3 + 22$$

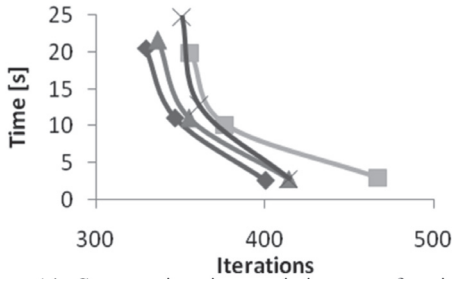

Figure 14. Computation time variation as a function of the swarm size, systems: (18): diamond), (19): square), (20): triangle), and (21): cross)

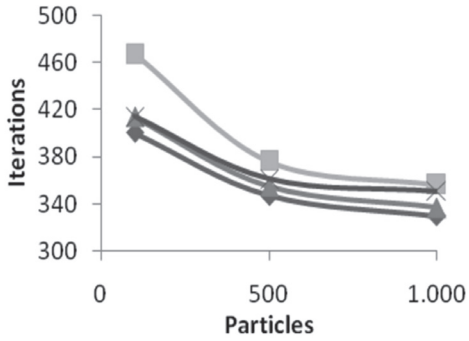

Figure 15. Iteration variation as a function of the swarm size, systems: (18): diamond), (19): square), (20): triangle), and (21): cross)

3.3 Five-equation systems

In an effort to check the behavior of PSO for more complex scenarios, systems (22) and (23) were implemented. Once again, Figure 16 shows that as the number of iterations goes up, the system is simplified and converges in less time.

$$
\begin{aligned}
f_1 &= 2.3x_1 + x_2^2 + 2x_4 + 0.01x_5 - 1.45 \\
f_2 &= -x_2 + 1.3x_5 + 9 \\
f_3 &= x_2x_3 - x_5^2 + 9 \\
f_4 &= x_1^3 - 2x_4x_3 + x_5^2 - 0.8 \\
f_5 &= -5x_3 - x_5 - 3x_5x_4x_3 + 3.6
\end{aligned} \tag{22}
$$

$$
\begin{aligned}
f_1 &= x_1 + 2x_2 - 0.3x_3x_4 - 10 \\
f_2 &= 0.1x_4x_5 + x_3 - 2.3x_1 \\
f_3 &= 7x_3 - 5.3x_1^2 + 3x_1 + 2.47 \\
f_4 &= x_5x_3x_2 - x_4^2 + 6.31 \\
f_5 &= x_1x_5 - x_3x_4 + 1.3x_2 - 4
\end{aligned} \tag{23}
$$

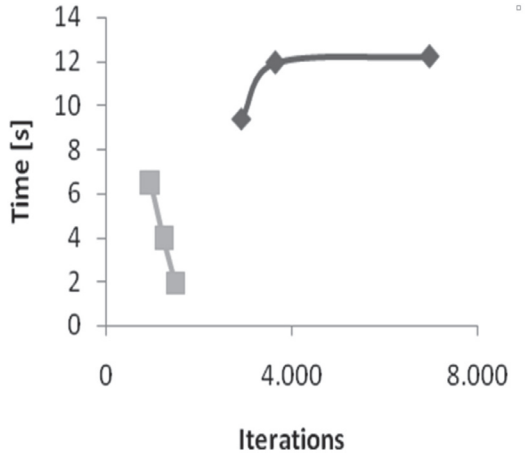

Figure 16. Computation time variation as a function of the swarm size. Systems: (22): diamond) and (23): square).

Figure 17 confirms the fact that smaller systems take more iterations to converge, but they are also simpler and therefore require less time to deliver a solution.

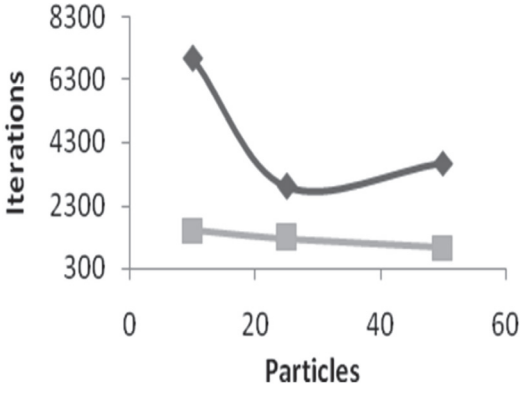

Figure 17. Iteration variation as a function of the swarm size. Systems: (22): diamond) and (23): square).

3.4 Performance Comparison

In an effort to compare the performance (in terms of speed) of PSO against commercial software, plots of average computation time for each type of system were performed. For problems of two and three equations, the average computation time of PSO was well above the one achieved with the commercial software. However, this changes for bigger systems, as can be seen in Figure 18, where it is easily deducted that for 10 particles, the required time is about half of that required by commercial solutions. Therefore, it appears that PSO is a good choice for bigger, more complex systems.

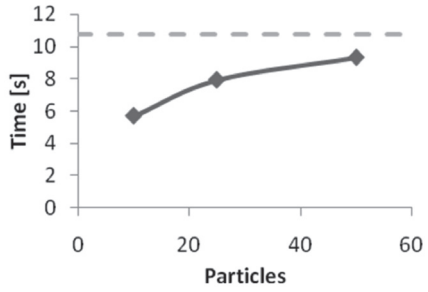

Figure 18. Average computation time for 5 x 5 systems. Diamond: PSO evolution. Dotted line: commercial software time

4. OBSERVATIONS AND CONCLUSIONS

After a careful review of the experimental data, it is natural to conclude that as the swarm gets bigger, the number of required iterations go down (see Figure 3,8,15). Interesting, though, is the fact that it has an adverse effect in the computation time (see Figure 5,13,14). This appears to be contradictory, since if there are less iterations it is expected that it will be quicker. Even though this is true, there is also the fact that by having a bigger swarm, the communication will be performed between more members, so it will adversely affect iteration time. Moreover, there will be more particles that require the calculation of P_{Best} and G_{Best}, so this will also increase the iteration time. In the end, convergence time will be higher than for a smaller swarm. Another interesting effect is the one that the swarm size has on the approximation error, which is directly proportional (see Figure 11). However, at the current time, it remains unknown whether it is due to some type of collision between particles or if it is due to the inertia weight (w), which could be stopping

particles before they are able to migrate to a better point. In spite of that, it is of the utmost importance to define a proper swarm size, so that a balance between iterations, computation time and approximation error, is found. If chosen properly, a solution faster than by commercial means can be achieved (Figure 18). It is remarkable that PSO appears to have a preference for finding roots which are closer to the starting point, but still finding the furthest ones.

Finally, and as future research, it would be interesting to study PSO with complex roots and interval mathematics. A first step towards this goal has been taken [17], and we hope to report more conclusive information in the near future.

ACKNOWLEDGMENTS

This work was supported by *Vicerrectoría de Investigación y Extensión* (*Universidad Industrial de Santander*), in the framework of project code 5551.

REFERENCES

[1] Grosan, C. and Abraham, A., A New Approach for Solving Nonlinear Equations Systems, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, Vol. 38, (3), pp. 698-714, May. 2008.

[2] Rao, S. S., Engineering Optimization: Theory and Practice, Fourth Ed. John Wiley & Sons, Inc., pp. 1-829, 2009

[3] Parsopoulos, K. E. and Vrahatis, M. N., Particle Swarm Optimization and Intelligence: Advances and Applications, First Ed. Information Science Reference, pp. 1-329, 2010.

[4] Clerc, M., Particle swarm optimization, First Ed. ISTE, ,. 243, P. 2006

[5] Burden, R. L. and Faires, J. D., Análisis Numérico, Sexta Ed. International Thomson Publishing, , pp. 1-802. 1998

[6] Dennis, J. E. J. and Moré, J. J., Quasi-Newton Methods, Motivation and Theory, SIAM Review, Vol. 19, (1), pp. 46-89, 1997.

[7] Allgower, E. L. and Georg, K., Numerical continuation methods: an introduction, Third Ed. Springer-Verlag New York, Inc., , pp. 1-388. 1990

[8] Effati, S. and Nazemi, A., A new method for solving a system of the nonlinear equations," Applied Mathematics and Computation, Vol. 168, no. 2, pp. 877-894, Sep. 2005.

[9] Bader, B. W., Tensor-Krylov Methods for Solving Large-Scale Systems of Nonlinear Equations, SIAM Journal on Numerical Analysis, Vol. 43, (3) , 1321 P. 2005.

[10] Salimbahrami, B. and Lohmann, B., Order reduction of large scale second-order systems using Krylov subspace methods, Linear Algebra and its Applications, Vol. 415, (2-3), pp. 385-405, Jun. 2006.

[11] Halton, J. H., Sequential Monte Carlo Techniques for Solving Non-Linear Systems, Monte Carlo Methods and Applications, Vol. 12, (2), pp. 113-141, Apr. 2006.

[12] Ortega, J. M. and Rheinboldt, W. C., Iterative Solution of Nonlinear Equations in Several Variables, First Ed. Academic Press, New York, , pp. 1-599, 1970.

[13] Nie, P., A null space method for solving system of equations, Applied Mathematics and Computation, Vol. 149, no. 1, pp. 215-226, Feb. 2004.

[14] Nie, P., An SQP approach with line search for a system of nonlinear equations, Mathematical and Computer Modelling, Vol. 43, (3-4), pp. 368-373, Feb. 2006.

[15] Gómez, L., Propuesta de demostración del teorema sobre la relación entre sistemas de ecuaciones y el problema de optimización (comunicación interna). pp. 1-2, 2010.

[16] Areiza, M., Un Método Numérico Cerrado para la Solución de Sistemas de Ecuaciones No Lineales en Dos Variables, Dyna, Vol. 69, (137), pp. 45-50, 2002.

[17] Vanegas, D., Barragán, K. S., and Correa, R., Comparación de las técnicas de optimización por análisis de intervalos y la de enjambre de partículas para funciones con restricciones," Ingeniería y Universidad (Universidad Javeriana - Accepted), 2011.