sciendo

DOI: 10.2478/awutm-2024-0007

Annals of West University of Timisoara
Mathematics and Computer Science
60 (2024), 98–109

Solving Equations Systems Using the Penguins Search Optimization

Liviu Octavian Mafteiu-Scai, Andrei Mursa, and Roxana-Teodora Mafteiu-Scai

Abstract The Penguin Search Optimization (PeSOA) is an optimization method based on collaborative hunting strategy of penguins. In this work, a solving linear equation systems method based on PeSOA is proposed. A dual population for PeSOA part was also tested. Experimental results obtained with our proposed method compared with those obtained with classical math methods, showed positive perspectives regarding possible extensions of the proposed method for systems of equations of larger dimensions. Conclusions as well as future research directions are also included.

AMS Subject Classification (2020) 60-08; 65C20
Keywords Nature inspired algorithms; equations systems; penguin search optimization

1 Introduction

1.1 Methods for solving equations systems

Even if solving equation systems is a very old and discussed problem in Algebra, this issue continues to be of great interest in math, computer science and engineering, the proof being the large number of scientific papers about numerical methods that have appeared lately. Nowadays, mathematicians consider two main types of methods for solving equation systems: direct methods and iterative methods. Mainly, direct methods consist of converting the system of equations into an equivalent triangular system that is easier to solve.

Direct methods make it possible to determine the exact solution of the system only in the ideal case when there are no rounding errors due to the finite representation of

numbers in computer memory. Major problems arise when systems with more than 100 equations are solved with direct methods, when these methods are unusable due to the accumulation of rounding errors that seriously alter the solution. From the complexity point of view, in general, the number of arithmetic operations performed is $O(n^3)$, which is not very encouraging for a software developer. The most known methods of this type are: Cramer, Gaussian elimination, Gauss-Jordan, LU factorization (decomposition) etc.

The iterative methods determine the solution on the basis of an iterative process, starting from an initial approximation of the solution, in the conditions in which the equations system is well conditioned. In practice, the iterative process is stopped after a preset number of steps (unknown a priori) or after certain conditions have been met. By iterative methods, the solution of a system of linear equations is obtained by generating a sequence that tends to its exact solution. A major advantage of iterative methods is that in practice rounding errors and truncation errors may be insignificant. We note that even if the solution is affected by truncation errors that are not encountered in direct methods, the iterative solution may have better accuracy than the solution obtained through direct methods [4]. The most known methods of this type are: Gauss-Seidel, conjugate gradient, Quasi-Newton methods, Broyden, Chebyshev, etc [4] . In conclusion, we can say that the right choice of a classical numerical method is not simple, and more, improper selection can lead to wrong solutions.

Artificial Intelligence and its results in recent decades have led to a new stage in solving equation systems. Besides the classical numerical methods for solving equations systems, in the last decades new methods that use artificial intelligence techniques to determine the solutions were proposed, many more research papers being evidence of this reality. Methods like Monte Carlo Methods (MCM), Artificial Neural Networks (ANN), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) algorithm, Imperialist Competitive Algorithm (ICA) , Harmony Search (HS) optimization, Memetic Algorithm (MA), Brainstorming Optimization (BSO), Cuckoo Search algorithm (CS), Greedy Randomized Adaptive Search Procedure (GRASP), Glowworm Swarm Optimization (GSO), Quantum Computing and many more were used to solve equation systems [5].

The hybridization of these new methods with classical methods has also led to promising results. In the last years, a lot of hybrid methods have been proposed for solving equations systems. These methods combine metaheuristics with performant iterative methods of system solving, the results being methods with good spatial and temporal complexity and accurate solutions [5]. More than one hundred published papers on this subject show researchers' concern in solving equation systems using artificial intelligence techniques. An increase in this interest can be seen especially in the last decade. Also, in the last decade one can observe a dominance of metaheuristic methods and an increase in the interest in hybrid methods [5]. Regarding the convergence of methods that use techniques from artificial intelligence, there are only a few published papers in which, for the particular cases of the proposed methods, the convergence is mathematically demonstrated [5].

1.2 Penguin Search Optimization Algorithm (PeSOA)

The PeSOA is inspired from penguins hunting strategy where each penguin represents a solution of the optimization problem [2]. A penguin colony is organized in groups and each group searches for food in an assigned hole. Each penguin dives at a random depth and begins searching in random directions. After a number of dives it returns to the surface and shares the amount of food and location. Each group shares the location of the food and at the next iteration penguins from groups with little food follow penguins who caught a lot of fish. The penguins hunt in groups and must synchronize their dives. The PeSOA method agrees to optimize their objective function, such as maximizing the amount of energy extracted from the energy invested. The relation (1.1) [2] is the base of the PeSOA, where a new candidate solution is computing using the best local solution, the last solution and a random number used for distribution.

$$D_{\text{new}} = D_{\text{LastLast}} + \text{rand}()^* \left| X_{\text{LocalBest}} - X_{\text{LocalLast}} \right| \tag{1.1}$$

where D_{new} is the new solution, D_{LastLast} is the last best solution corresponding to this penguin, rand() is an uniformly distributed random number, $X_{\text{LocalBest}}$ the best local solution and $X_{\text{LocalLast}}$ the local last solution.

The following rules explain the basic algorithm [2]:

- Rule 1: A penguin population is made up of several groups.
- Rule 2: Each group is composed of a variable number of penguins that can vary depending on food availability in a specific location.
- Rule 3: They hunt in group and move randomly until they find food when oxygen reserves are not depleted.
- Rule 4: They can perform simultaneous dives to a depth identical.
- Rule 5: Each group of penguins starts searching in a specific position (hole i) and random levels (levels j1, j2, ..., jn).
- Rule 6: Each penguin looks for food in random way and individually in its group, and after rough number of dives, penguins back on the ice to share with its affiliates, the location (represented by the level or depth of the dive) where he found food and plenty of it (represented by the amount of eaten fisch). This rule ensures intra-group communication.
- Rule 7: At one level, one can have from 0 to N penguins (penguin or any group) according to the abundance of food.
- Rule 8: If the number of fish in a hole is not enough (or none) for the group, part of the group (or the whole group) migrates to another hole. (This rule ensures inter-group communication)
- Rule 9: The group who ate the most fish delivers the location of rich food represented by the hole and the level.

A detailed description of the basic algorithm as well as the auxiliary ones in pseudo-code can be seen in the paper [2]:

***The basic Algorithm of PeSOA* [2]:**

1: Generate K regions in the solution space
2: Generate penguins x(ij), (j = 1, 2, ..., N/K) for each group i within the designated region;
3: while stopping criterion is not reached do
4: Initialise the oxygen reserve for each penguin;
5: for each group i do
6: for each penguin j in this group do
7: improve the penguin position x(ij)
8: endfor
9: update the food abundance degree for this group ;
10: endfor
11: update the global best solution;
12: update membership function values for each group b;
13: redistribute penguins to groups according to the membership function;
14: abandon the group if it has no members;
15: endwhile
16: end

The algorithm proposed in [2] was validated with different benchmarks test functions like Rastrigin function, sphere function, Rosenbrock function, Schwefel function, and Michalewicz function. Additionally, some PeSOA applications are listed. An application of PeSOA is given in paper [8] where the method is used to solve the traveling salesman problem (TSP). To improve the search technique of PeSOA, for the same problem, a hybridization of PeSOA with the Harmony Search algorithm (HS) is done in work [7]. In paper [1] the PeSOA is used to consolidate multiple virtual machines in exceedingly cloud-based distributed hosting surroundings. Paper [9] proposes a PeSOA method for solving one of the most difficult NP-hard combinatorial optimization problems in discrete case, the Quadratic assignment problem. A hybridization of GA, ACO and PeSOA methods for solving the vehicle routing problem (VRP) is described in [1]. A modified version of the PeSOA by adding gaussian exploration for use in water resource field is proposed in paper [6]. In paper [10] the PeSOA is hybridized with three different classifiers, Naive Bayes(NB), k-Nearest Neighbors (kNN) and Support Vector Machines (SVMs) to find the best subset features in case of Biomedical Data Classification. PeSOA is used for finding the optimal location and sizing of Distributed Generation and Distribution STATic COMpensator (DSTATCOM) with the aim of reducing the total power loss along with voltage profile improvement of Radial Distribution System in work [11]. In paper [3], the authors proposed an Algorithm for Association Rules Mining (ARM) that uses PeSOA. As a final remark, from 2016 more than one hundred papers about real problems that can be solved by PeSOA were published.

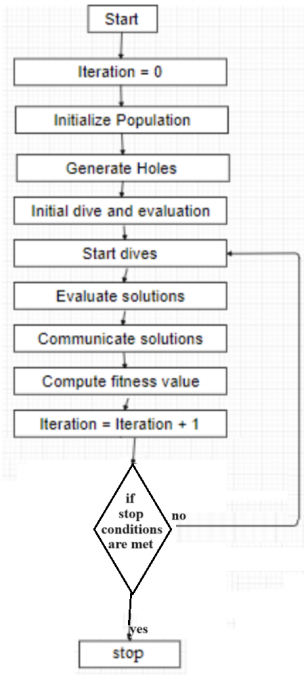

Figure 1: Diagram of Simple-PeSOA

2 The proposed method

The problem to find a solution for an equations system is in fact a multi-objective optimization problem (MOP). The continuous MOP considered in our problem can be mathematically defined as:

$$\text{minimize } F(x) = (f_1(x), f_2(x), \ldots, f_m(x))^T \tag{2.1}$$

with $\Omega \subset \mathbb{R}^n$ the decision space or variable space, $x = (x_1, x_2, \ldots x_n) \in \Omega$ a candidate solution, the function $F : \Omega \to \mathbb{R}^m$ consisting of m real-valued objective functions (in our problem, m is the number of equations) and $\mathbb{R}^m$ the objective space.

The basic idea of the proposed algorithm for finding solutions of equations systems is to generate random solutions and improve them until termination conditions are satisfied.

Here we have three termination conditions: an exact solution is found (the value of fitness function is equal to zero), an approximate solution (noted good enough in the diagram) is found (a certain precision is satisfied) or a preset number of iterations is reached.

A first variant of the proposed algorithm follows the main structure of the PeSOA proposed in [2] and is represented in Figure 1.

The computational complexity for the population generation process is $O\left(p^2 + n\right)$ where p represents the number of groups and n represents the number of penguins. The complexity for the iterative search process is $O(n \cdot i)$ where i is the maximum number of iterations (as a condition to stop the search). Thus, we have $O\left(p^2 + n + ni\right)$. Because in practical application, $p << n$, we have $O\left(ni\right)$ as a worst case.

In our proposed method for solving linear equations systems, an individual is rep-

resented by a vector in which the elements represent the values/solutions of unknowns. Even if the most components of this vector are good or good enough, only one or a few bad components can negatively affect the global fitness of the individual, the global solution of the linear equations system. It is known that diversity is very important in evolutionary multi-objective optimization. In general, in evolutionary algorithms based on fitness of individuals, individuals are selected for the next generation based on their fitness values. Individuals with lower fitness are weeded out. The fitness value of an individual is given by the sum of all sub-fitness of all genes. In our opinion, this is very risky, because these individuals may have the most favorable genes but a poor minority of genes may negatively influence the global fitness of the individual. For solving this possible problem we propose to keep a little part of individuals with bad fitness and recombine them. It is possible for these bad individuals to lead in a new generation to individuals with good fitness. This strategy is the idea for a new variant of the PeSOA, called Dual-Penguin Search Optimization Algorithm (DPeSOA). In the case of DPeSOA we have two populations, the first formed by individuals with best fitness (main population) and the second formed by individuals with worst fitness (second population). This variant of the proposed algorithm begins with only one population but after the first iteration the individuals selection process forms two populations. The main population is used to obtain accuracy while the second is good for diversity and for individuals with latent genes. It should be noted that migration between the two populations is possible. The main disadvantage of working with two populations is obviously the increase in computational effort and the larger memory space required.

In the proposed algorithm for solving systems of equations inspired from penguins hunting strategy, the problem is viewed as an optimization problem.

Finding a solution for the system of equations $f(x) = 0$ where the vector $x = \{x_1, x_2, \ldots, x_n\}$ represents the unknowns, involves finding a solution such that every equation in the system is satisfied, i.e.:

$$\{f_1(x_1, x_2, \ldots, x_n) = 0, f_2(x_1, x_2, \ldots, x_n) = 0, \ldots, f_n(x_1, x_2, \ldots, x_n) = 0\}. \tag{2.2}$$

A solution is an assignment of values to the variables $x_1, x_2, \ldots, x_n$ such that each of the equations is satisfied. The solution set is the set of all possible solutions for the equations system and there are three possible situations: unique, infinitely many, or no solution for the system. In our approach the method is viewed as an optimization problem (finding a solution by minimizing a given objective function). This is called a fitness function and it should quantitatively measure how fit a given solution is in solving the problem. For each equation in part we used function $f_i^2(x)$ and to compute the fitness for entire system we used the formula given by relation (2.3)

$$F(x) = \sum_{i=1}^{n} f_i^2(x) \tag{2.3}$$

where x is the solution vector and n is the number of equations. This fitness function tells the algorithm how good a particular solution is. It is obvious that the condition must be met $F(x) \geq 0$ by definition. Thus, for x^* to be global minimum of $F(x)$, i.e. an

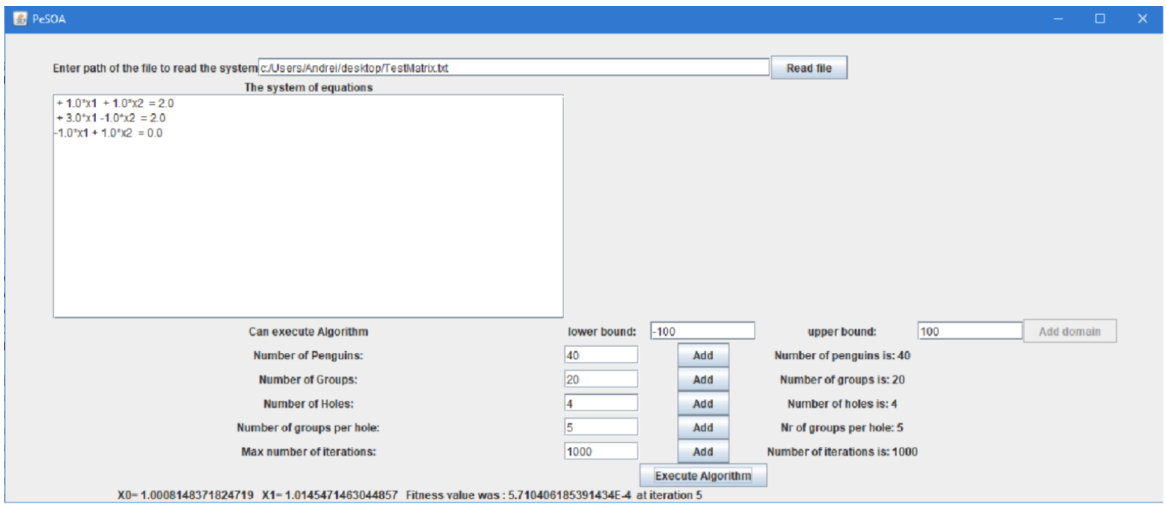

Figure 2: The main user interface

exact solution of equations system, then the following must hold true: $f_1(x^*) = f_2(x^*) = \ldots = f_n(x^*) = 0$.

As will be seen in next section, the proposed method is able to find solutions even in cases when the number of unknowns is smaller than the number of equations or in cases when the number of unknowns is greater than the number of equations in the system, i.e. over-determinate and under-determinate systems.

In the main user interface of our application (Figure 2) the user can select the system of equations, the parameters for the initialization of the program and the working parameters. First, the user has to select the folder containing details about the equations system. The system is displayed in the text area for inspection. Afterwards, a domain can be added for each unknown in the system. The user can also select the number of penguins in the population, the number of penguin groups, the number of holes and how many groups should be assigned per group as well as the maximum number of iterations. After all data was selected the user can press the button Execute Algorithm to start the program. The result is displayed at the bottom and includes the value for each unknown, the fitness value and the iteration at which the solution was found.

The main idea of the proposed algorithm to solve linear equations systems is to generate random solutions for a given equations system and improve them until a certain precision is satisfied or until a number of iterations is reached. The algorithm as can be seen in Figure 1, follows the main structure of the PeSOA proposed in [2]. The most important implementation steps are briefly described in the following:

- population initialization is performed at the beginning of the program, a method that creates the initial penguins population and assign penguins to groups, as can be seen in Figure 2;

- assigning each group to a hole is the next phase. Because the search space for equations systems can be big, it is divided into smaller intervals. Each penguin searches at an assigned depth and hole, and is searching for a part of the solution;

- after the dives each group generates a solution, the fitness value is calculated and it is passed to the other groups. As every group communicates its own solution, new solutions are generated based on different hole location and depth and the fitness value is computed. The solution with the best fitness value is communicated to all groups and at the next iteration penguins have a probability to follow penguins that found a better solution;

- the hole represents where it performs the search, the depth represents the search space and the depth level is used to identify which part of the solution the penguin returns;

- the fitness value, the new solution that is found, the last solution and the best solution that is found are communicated to the whole group. Each penguin in the group will dive numerous times and at a specified depth and return a solution to the group. The penguins move in a random way but they will also try to follow the penguins that have a smaller fitness value, hence they have found a better solution;

- in our implementation there is a class called PenguinGroup, that contains the list of penguins for this group, the group number and the number of unknowns in the system. It is used to calculate the max depth level, the fitness value for the group, a possible solution based on the findings of the penguins and the best solution found by this group or another. Each penguin group will retrieve solutions found by penguins and will compute the fitness value to determine the best local solution. After a dive session the best local solution is communicated to the other groups, this way intercommunication between groups is achieved;

- the class PeSOAProcess contains the method execute() which is called from the main UI when the Execute Algorithm button is pressed. It contains the population, a best global solution which is returned as output after ending conditions are met, a list of all holes and the initial search spaces for each unknown in the system, these are later divided into smaller intervals and assigned to each hole, the fitness value and a list that hold all the unique solutions communicated by the penguin groups;

- before running the program, the user will create a new depth level for each unknown in the system. After all depth levels are created the user can run the algorithm to find the solution. To do this, in UI implementation there is an option Add domain. Also, the user can set the running parameters of the program;

- at the beginning of the computation process, a method called firstTimeDive(), is called once to initialize the first solution for each penguin, and it chooses a random number inside the search space. After this, a method called dive() is called every iteration and represents the way penguins move through the search space. The new solution is generated by the following formula:

$$X_{\text{new}} = X_{\text{last}} + o + r^* \left(X_{\text{best}} - X_{\text{last}}\right) \tag{2.4}$$

where X_{new} is the new solution, X_{last} is the last solution found, o is a random number in $(0, 1)$ that represents the oxygen reserve of the penguin, r is a random

number in $(0, 1)$, $r \neq 0$, and X_{best} is the best solution found at the corresponding depth.

- the penguins usually would follow the best solution but there is a chance they will search for a random solution in the search space.

3 Experimental results

3.1 First set of experiments

This set of experiments aimed at testing the first variant of the proposed algorithm, i.e. only one population of penguins was used. Thus, systems of equations, of small dimensions, including under-determinate and over-determinate ones, were solved using this variant of the algorithm, while others methods fail, as can be seen in Figure 3, where some experimental results are presented. For each system of equations, the solutions obtained in the case of four different runs of the proposed simple method are presented, in order to highlight the convergence of the algorithm, even if the number of iterations differs, sometimes by one or even two orders of magnitude. The average values for each individual system are highlighted with italic fonts. The working parameters values in algorithm implementation were and can be set by the application user, as can be seen in Figure 2:

- number of penguins: 20 x number of systems unknowns;
- number of penguins groups: 20;
- number of holes: 4;
- number of penguins group per hole: 5;
- searching interval: [-100, 100];
- precision: 0.005;
- max. number of iterations: 2000

3.2 Second set of experiments

The second set of experiments aimed at testing the second variant of the proposed algorithm DPeSOA, i.e. the variant that works with two populations of penguins. In this set of experiments we wanted to see if DPeSOA leads to better convergence or solutions. First, we generated linear equation systems with sizes between 2 and 5. After that, only the systems that admit a unique solution were selected. The set of experimental data includes 10 systems of equations for each size. The working parameters values were the same as in the first set of experiments. Regarding to the second variant of the proposed algorithm DPeSOA that uses a dual population, the following were observed:

The equations system	Different solutions found by proposed algorithm – version with only one population -			Fitness values	Iterations	Result obtained with other methods								
						Cramer's rule			Gauss-Jordan elimination			Conjugate gradient		
	x	y	z			x	y	z	x	y	z	x	y	z
$\begin{cases} 3x - y = 7 \\ 2x + 3y = 1 \end{cases}$	2.011885	-0.997051	-	0.002133	86	2	-1	-	2	-1	-	2	-1	-
	2.017346	-1.014635	-	0.004530	207									
	1.985057	-1.000035	-	0.002905	32									
	2.001226	-1.004760	-	2.111549	112									
average	***2.003878***	***-1.004120***		***0.530279***	***109***									
$\begin{cases} x + y + z = 6 \\ x - y - z = -4 \\ -x - y + z = 0 \end{cases}$	0.816547	1.995072	2.952009	0.092619	796	1	2	3	1	2	3	Fail {1, 3, 2} -verify only first and second equation		
	1.030761	1.966309	3.022293	0.007719	896									
	0.960966	2.024918	2.939549	0.007719	2000									
	1.032014	1.945629	2.982230	0.012479	2000									
average	***0.960072***	***1.982982***	***2.974020***	***0.030134***	***1423***									
$\begin{cases} 2x + y + 3z = 13 \\ x + 5y + z = 14 \\ 3x + y + 4z = 17 \end{cases}$	1.192135	1.979484	2.865513	0.003917	1331	1	2	3	1	2	3	1	2	3
	1.363166	1.994439	2.731543	0.011735	2000									
	1.551023	1.987907	2.597279	0.022594	2000									
	1.089056	1.985640	2.925325	0.009034	2000									
average	***1.298845***	***1.986867***	***2.779915***	***0.01182***	***1832***									
$\begin{cases} 2x + 4y + z = 5 \\ 4x + 4y + 3z = 9 \\ 4x + 8y + z = 9 \end{cases}$	1.416290	0.273266	1.142172	0.004643	73	1	0.5	1	1	0.5	1	Fail {-6.5, 2.5, 8} -verify only first equation		
	1.412155	0.271928	1.141395	0.004270	49									
	1.430361	0.264251	1.113503	0.003605	48									
	1.522498	0.249860	0.969456	0.003597	70									
average	***1.445326***	***0.264826***	***1.091631***	***0.004028***	***60***									
$\begin{cases} 5x + 4y - z = 8 \\ x - y + z = 4 \end{cases}$	1.901122	0.221585	2.322081	0.004885	28	Fail -null determinant			x=8/3 -(1/3)z y=4/3 =(2/3)z z - free			Fail {1, 1, 1} verify only first equation		
	1.767549	0.435803	2.643907	0.004555	449									
	1.956825	0.065937	2.076643	0.001881	40									
Underdeterminate	1.857533	0.265927	2.410700	0.001881	103									
average	***1.870757***	***0.247313***	***2.363332***	***0.003300***	***155***									
$\begin{cases} x + y = 2 \\ 3x - y = 2 \\ -x + y = 0 \end{cases}$	0.991741	0.990635	-	5.493348	7	Fail -null determinant			1	1	-	Fail {2, 0} -verify only first equation		
	0.985988	0.973608	-	0.002030	7									
	0.995041	1.020287	-	0.002108	14									
Overdeterminate	0.976277	0.973367	-	0.004527	5									
average	***0.987261***	***0.989474***		***1.375503***	***8***									

Figure 3: Experimental resuls with one population

- the solutions found by both variants were about the same;
- in general, the number of necessary iterations has decreased in the case of DPeSOA, sometimes up to half, except when the maximum number of iterations (2000 in our implementation) has been reached, as can be seen in Figure 4;
- even if the execution time increases due to the management and processing of two populations, the overall execution time decreases due to the significant reduction of the number of iterations needed to find the solution. As an average, the reduction in execution time is about 15%, a fact that is not necessarily an advantage in the case of small systems;
- using two different populations, the algorithm can explore the objective space more thoroughly and can have more stable performance;
- experimentally it was observed that the best ratio between the main and the secondary population seems to be 2/1. A comparation of PeSOA and DPeSOA variants from number of iterations needed can be seen in Figure 4. We mention again that the solutions founded by both variants are the same or very close (differences from 3th decimal).

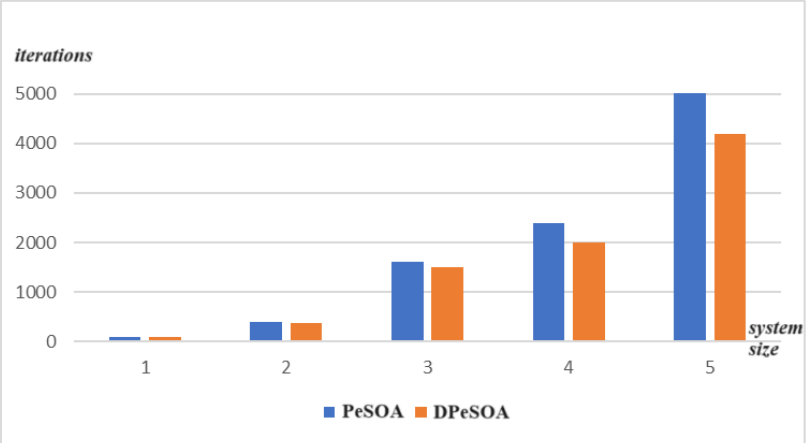

Figure 4: Experimental resuls with dual populations - average iterations needed

4 Conclusions and future work

In this paper, an approach inspired from nature for solving systems of equations was proposed. The problem was transformed into a problem of optimization, more exactly into a minimization problem from objective function point of view. The proposed algorithm is accurate and was able to find the solution of a given system of equations even in cases where traditional methods fail. If the maximum number of iterations was reached, a good enough approximation of the solution is given by the proposed method. The dual population approach seems to be better than the classical approach from the number of iterations and in cases of systems with multiple solutions, thus the study and improvement of this variant will be one of our concerns in the future. One of interests for future work is to reduce the time complexity and spatial complexity of the proposed algorithm and for these, the study of Pareto front and Gaussian joint distribution regarding their possible advantages in case of our algorithm, will be another future work. A parallelization of the algorithm will be taken into consideration, especially for DPeSOA, with the aim of solving large systems in reasonable times.

References

[1] M. Ammi, S. Chikhi, *Cooperative Parallel Metaheuristics based Penguin Optimization Search for Solving the Vehicle Routing Problem*, International Journal of Applied Metaheuristic Computing archive Volume **7 (1)** (2016), 1-18.

[2] Y. Gheraibia, A. Moussaoui, *Penguins Search Optimization Algorithm (PeSOA)*, in: M. Ali, T. Bosse, K. V. Hindriks, M. Hoogendoorn, C. M. Jonker, J. Treur (eds), Recent Trends in Applied Artificial Intelligence, IEA/AIE 2013, Lecture Notes in Computer Science vol. 7906, Springer, 2013.

[3] Y. Gheraibia, A. Moussaoui, Y. Djenouri, S. Kabir, P. Y. Yin, *Penguins Search Optimisation Algorithm for Association Rules Mining*, Journal of Computing and Information Technology **24 (2)** (2016), 165-179.

[4] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM, 1995.

[5] L. Mafteiu-Scai, E. Mafteiu, R. Mafteiu-Scai, *Solving Equations Systems Using Artificial Intelligence - a Survey.*, International Journal of New Computer Architectures and Their Applications **8.3** (2018), 102-119.

[6] A. Mansouri, B. Aminnejad, H. Ahmadi, *Introducing modified version of penguins search optimization algorithm (PeSOA) and its application in optimal operation of reservoir systems*, Water Supply Journal **18 (4)** (2018), 1484-1496.

[7] I. Mzili, M. Bouzidi, M. E. Riffi, *A novel hybrid penguins search optimization algorithm to solve travelling salesman problem*, 2015 Third World Conference on Complex Systems (WCCS), DOI: 10.1109/ICoCS.2015.7483237, IEEE 2016.

[8] I. Mzili, M. E. Riffi, *Discrete Penguins Search Optimization Algorithm To Solve The Traveling Salesman Problem*, Journal of Theoretical and Applied Information Technology **72 (3)** (2015), 331-336.

[9] I. Mzili, M. E. Riffi, F. Benzekri, *Penguins Search Optimization Algorithm to Solve Quadratic Assignment Problem*, March 2017 BDCA'17: Proceedings of the 2nd international Conference on Big Data, Cloud and Applications, ACM 2017.

[10] B. Noria, Z. Elberrichi, *Using Penguins Search Optimization Algorithm for Best Features Selection for Biomedical Data Classification*, International Journal of Organizational and Collective Intelligence **7 (4)** (2017), 51-62.

[11] J. Prabu, S. Muthuveerapan, *Optimum Placement and Sizing Determination of Distributed Generation and DSTATCOM Using Penguins Search Optimisation Algorithm*, International Journal of Innovative Research in Science, Engineering and Technology **5** (2016), 6675-6680.

Liviu Octavian Mafteiu-Scai
Computer Science Department West University of Timisoara, Timisoara, Romania
E-mail: liviu.mafteiu@e-uvt.ro

Andrei Mursa
BTC-Embedded Systems Timisoara, Romania
E-mail: andrei.mursa96@yahoo.com

Roxana-Teodora Mafteiu-Scai
Computer Science Department West University of Timisoara, Timisoara, Romania
E-mail: roxana.mafteiu97@e-uvt.ro

Received: 16.05.2024
Revised: 11.06.2024
Accepted: 12.08.2024