Applications of the Circle-Inspired Optimization Algorithm (CIOA): comparison between MatLab and Python versions

Otávio A. P. de Souza[1], Letícia F. F. Miguel[2]

[1]*Postgraduate Program in Civil Engineering (PPGEC), Federal University of Rio Grande do Sul (UFRGS)*
Av. Osvaldo Aranha, 99, 90035-190, Porto Alegre/Rio Grande do Sul, Brazil
otavio.peter@hotmail.com
[2]*Dept. of Mechanical Engineering (DEMEC), Postgraduate Program in Mechanical Engineering (PROMEC), Postgraduate Program in Civil Engineering (PPGEC), Federal University of Rio Grande do Sul (UFGRS)*
Av. Sarmento Leite, 425, 90050, Porto Alegre/Rio Grande do Sul, Brazil
letffm@ufrgs.br

Abstract. This paper presents new applications of the Circle-Inspired Optimization Algorithm (CIOA), a modern and efficient optimization algorithm developed by the authors, in optimization problems implemented in MatLab and Python, with the aim of making a comparison between the version of the algorithm in each computational language. Thus, different types of optimization problems were implemented and solved multiple times in both versions of the CIOA (MatLab and Python), evaluating the best solution, the average among all solutions, the standard deviation, and computational time. The results demonstrate the efficiency of CIOA in both computational languages. In terms of precision and robustness, it was not possible to determine which language performed better, as the differences obtained were small, probably due to the random characteristics of the algorithm. However, in relation to computational time, the MatLab version presented better performance. This advantage can be explained due to the libraries used and the authors' experience in each programming language.

Keywords: Circle-Inspired Optimization Algorithm, MatLab, Python.

1 Introduction

Metaheuristic algorithms are efficient tools for solving challenging optimization problems. Over the last few decades, many metaheuristic algorithms have been developed, the most famous of which can be cited: Particle Swarm Optimization (PSO), developed by Kennedy and Eberhart [1]; Differential Evolution (DE), developed by Storn and Price [2]; Harmony Search (HS), created by Geem et al. [3]; Firefly Algorithm, implemented by Yang [4]; Search Group Algorithm (SGA), developed by Gonçalves et al. [5]; Whale Optimization Algorithm (WOA), created by Mirjalili and Lewis [6]; Butterfly Optimization Algorithm (BOA), developed by Arora and Singh [7].

In this context, the Circle-Inspired Optimization Algorithm (CIOA) was developed by the authors (Souza [8] and Souza and Miguel [9]) with the promise of being a very efficient metaheuristic algorithm to solve engineering optimization problems. Some applications of the CIOA can be cited, such as the work of Mahato et al. [10] and Miguel e Souza [11], in engineering studies. Uses of the algorithm in other areas of research are also found, such as the work of Salim and Sarath [12], who developed an adaptation inspired by CIOA to use it in cancer studies.

CIOA was initially developed and published solely in the MatLab computational language, as were most of the algorithms mentioned above. However, the increasing use of other computational languages, particularly Python in the engineering field, motivated the implementation of CIOA in this language as well. Currently, two versions of CIOA are available: one in MatLab and one in Python.

Therefore, the main objective of this paper is to apply the CIOA to different optimization problems and, as a

novel contribution, to compare the efficiency of the versions implemented in MatLab and Python. Thus, this paper is organized as follows: the first section provides a brief introduction to the work; the second section consists of a summary of the CIOA formulation; the third section describes the optimization problems to which CIOA was applied, with the aim of comparing the versions in different computer languages; in the fourth section, results are presented and discussed; and the fifth section highlights the conclusions.

2 Formulation of the Circle-Inspired Optimization Algorithm (CIOA)

The Circle-Inspired Optimization Algorithm (CIOA) is an optimization algorithm developed by the authors (Souza [8] and Souza and Miguel [9]), in which each search agent moves through circular arcs with a radius r. The better the value of the objective function obtained by a given search agent, the smaller the radius r used in the movement of this agent in the next iteration. The formulation of the CIOA is presented below.

Considering N_{ag} the number of search agents and L_b and U_b the lower and upper limits, respectively, of each variable of design, a vector of radii $\vec{r}$ is generated, where the value of each element r_j of this vector is calculated according to Equations (1), where c_r is a constant.

$$r_j = \frac{c_r j^2}{N_{ag}} \qquad \text{and} \qquad c_r = \frac{\sqrt{U_b - L_b}}{N_{ag}} \tag{1}$$

After initialization, the first solution generated by CIOA is random. In subsequent iterations, the solutions obtained are classified, so that the search agent that produced the j^{th} best solution in an iteration k will have its coordinates updated in iteration $k+1$ through Equations (2) and (3), in which the indices 2_i and 2_{i-1} of the coordinates (or design variables) refer to even and odd numbers, respectively. $rand$ variables refer to random numbers drawn from a uniform distribution between zero and one.

$$x_{2i}(k+1) = x_{2i}(k) - rand_1 r_j \sin(k\theta) + rand_2 r_j \sin((k+1)\theta) \tag{2}$$

$$x_{2i-1}(k+1) = x_{2i-1}(k) - rand_3 r_j \cos(k\theta) + rand_4 r_j \cos((k+1)\theta) \tag{3}$$

To schematize the movement of search agents in CIOA, it is assumed that in an iteration a search agent moves from point 1 to point 2, in a movement governed by angle θ and radius $r_{1,2}$ with center at $O_{1,2}$. In the subsequent iteration, the search agent moves from point 2 to point 3, the movement is governed by the same angle θ but now by the radius $r_{2,3}$ with center at $O_{2,3}$. This procedure is presented in Figure 1 for two different situations: in the first (Figure 1.a), the search agent improves the classification of its solution, thus, its new movement will be made with a smaller radius ($r_{2,3} < r_{1,2}$); in the second (Figure 1.b), the classification of the agent's solution worsens, so its subsequent movement will be governed by a larger radius ($r_{2,3} > r_{1,2}$).

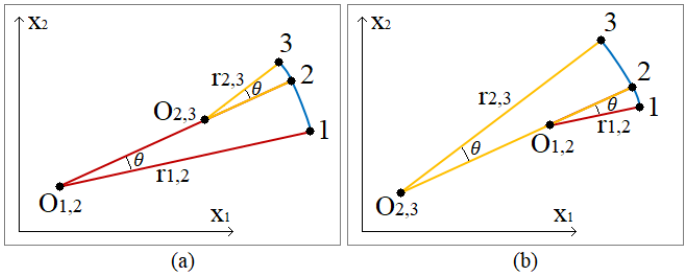

Figure 1. Changing the radius and updating the circle center.

Whenever a search agent causes the variable x_i to assume a coordinate outside the variable boundary, its value will be updated to the coordinate generated by the best-classified search agent in this iteration. If the variable boundaries are exceeded by the best agent, the values will be updated to L_b or U_b. Whenever the agents complete a "circle", that is, when $k\theta$ exceeds a multiple of 360º, the radius vector will be updated according to the Equation $\vec{r}_{new} = \vec{r}.\vec{r}_{up}$, where $\vec{r}_{new}$ is the new radius vector, and r_{up} is an update coefficient, usually set at $r_{up} = 0.99$.

To introduce an exclusively local search step into the algorithm, a parameter called $Glob_{It}$ is defined, which can assume values in the range $(0, 1]$, to be defined by the user. This way, the exclusively local search will start

in iteration k when the proportion between k and the total number of iterations assumes values greater than $Glob_{It}$. At this point, all search agents of the CIOA are restarted, starting from the coordinates that generated the best solution until the current iteration. Furthermore, an update of the design variable boundaries is carried out, according to Equations (4), so that the new lower and upper bound values, respectively L_{b1_i} and U_{b1_i}, for each design variable x_i, data will be given as a function of the old limits (L_b and U_b) and the variable x_{ibest}, which is the variable of dimension i that produced the best solution to date.

$$L_{b1_i} = x_{i_{best}} - \frac{U_b - L_b}{10000} \qquad \text{and} \qquad U_{b1_i} = x_{i_{best}} + \frac{U_b - L_b}{10000} \tag{4}$$

Once initialized, this step of the algorithm is governed by the same equations presented previously (Equations (3)), now restricting the search space delimited by L_{b1i} and U_{b1i}. In specific cases where $L_{b1i} < L_b$ or $U_{b1i} > U_b$, whenever after updating a design variable, its value is in the ranges $L_{b1i} < x_i < L_b$ or $U_{b1i} > x_i > U_b$, the value of this variable will be updated to, respectively, $x_i = L_b$ or $x_i = U_b$.

The CIOA pseudocode is presented in Figure 2. For better functioning of the algorithm, a non-multiple of 360º should be defined for θ and a value in the range between 0.75 and 0.95 for the $Glob_{It}$ parameter. More information about the formulation and implementation of the algorithm, as well as validation and efficiency tests where CIOA is compared with other famous algorithms, can be seen in Souza [8] and in Souza and Miguel [9].

Begin
  Define θ and $Glob_{It}$
  Initialize a radii vector $\vec{r}$
  Assign random values to design variables and evaluate the objective function for each search agent
  while1 ($k \leq Glob_{It} \times$ Maximum number of iterations)
    Classify search agents according to the quality of the solution obtained and update your positions
    Verify if any design variable exceeds the imposed boundaries
    if (k is a multiple of the value rounded down to $360/\theta$)
      Update vector $\vec{r}$
    end if
  end while1
  Reset all search agents with the position that generated the best solution so far
  Update design variable bounds
  while2 ($k \leq$ Maximum number of iterations)
    Repeat the procedures described in **while1** using the updated design variable bounds
  end while2
  Results visualization
End

Figure 2. Pseudocode of the CIOA.

3 Algorithm Configuration and Optimization Problems

This section presents the optimization problems that were solved using CIOA. These problems are separated into three different categories: Benchmark Functions, Real-World Problems, and Truss Optimization Problems. The CIOA parameters were configured as $\theta = 17°$ and $Glob_{It} = 0.85$. All problems were implemented and solved on the same computing platform: Windows 10, 8th generation Core i5 processor, and 8 GB of memory. MATLAB R2015a (8.5.0) and Jupyter Notebook 6.1.4 (Anaconda 3) were used. The parameters compared in each problem are the best solution, mean of solutions, standard deviation, and computational time.

3.1 Benchmark functions

The first set of optimization problems solved by CIOA consists of 20 benchmark functions known in the literature. Details of each function can be found in Mirjalili and Lewis [6]. In this paper, functions f_1 through f_7 are unimodal while the functions f_8 through f_{11} are multimodal. These functions correspond to functions with the same numbering (i.e., f_1 to f_{11}) in Mirjalili and Lewis [6]. Functions f_{12} through f_{20} in this paper are multimodal with fixed dimensions and correspond to the functions f_{15} through f_{23} in Mirjalili and Lewis [6]. For both versions

of CIOA, each function was solved 51 times, using 800 iterations and 250 search agents, totaling 200,000 evaluations of the objective function. The results are presented in the results chapter.

3.2 Real-world optimization problems

The second set of problems solved by CIOA consists of ten real-world problems that are part of a package of optimization problems used in the 'CEC2020 Real-World One-Objective Constrained Optimization Competition' and available in Kumar et al. [13]. The ten optimization problems analyzed in this paper, referred R_1 to R_{10}, are: Weight minimization of a speed reducer, Optimal design of industrial refrigeration system, Tension/compression spring design (case 1), Pressure vessel design, Welded beam design, Multiple disk clutch brake design problem, Planetary gear train design optimization problem, Step-cone pulley problem, Gear train design problem and Himmelblau's function. Each version of CIOA solved each real-world problem 51 times, using 400 iterations and 250 search agents, totaling 100,000 objective function evaluations, except for problem R_2 which, due to the greater number of design variables, required 800 iterations and 250 search agents, totaling 200,000 evaluations.

3.3 Truss structural optimization problems

In this subsection, four truss structural optimization problems solved by CIOA are presented in Table 1. More details about the implementation of these problems can be found in Souza [8], Souza and Miguel [9], and Miguel and Fadel Miguel [14, 15]. In Table 1, n refers to the number of design variables and g to the number of inequality constraints. The numbers of research agents (N_{ag}), iterations (N_{it}) and objective function evaluations (N_{ev}) used in each problem are also presented. Each problem was run 20 times on both versions of CIOA.

Table 1. Truss structural optimization problems.

	Name	n	g	N_{ag}	N_{it}	N_{ev}
S_1	Size optimization of a 25-bar space truss with stress and displacement constraints	8	124	250	800	200,000
S_2	Shape and size optimization of an 18-bar plane truss with stress and buckling constraints	12	54	250	1,600	400,000
S_3	Shape and size optimization of a 52-bar space truss with natural frequency constraints	13	2	250	2,400	600,000
S_4	Size optimization of a realistic transmission tower of 163-bar with stress, displacement, buckling, and fundamental natural frequency constraints	11	810	250	800	200,000

4 Results

This section presents the results obtained for the optimization problems described in Section 3.

4.1 Results for benchmark functions

Table 2 presents the results generated by each version of CIOA in optimizing the benchmark functions mentioned in Section 3.1. For each parameter evaluated, the version of CIOA that generated the best result is highlighted in bold. For the best solution, MatLab performed better in 9 functions, while CIOA in Python performed better in 11 functions. For the mean solution and standard deviation, MatLab was more efficient in 8 functions, while Python was more efficient in 12 functions. In problems ranging from f_{13} to f_{20}, the same result was apparently obtained by both versions in some of the parameters; however, this occurs only due to the rounding used to present the results. In these problems, some differences are present in the 7th or 8th decimal place.

The results of the mean solution are also compared with those results obtained by Mirjalili and Lewis [6], who used the Whale Optimization Algorithm (WOA0 in MatLab, to validate the analyzes in the present work. In 11 functions, both versions of the CIOA generated better results than those reported by Mirjalili and Lewis [6].

Table 2. Results for benchmark functions.

	Results obtained in this paper								[6]
	Best solution		Mean Solution		Std. deviation		Time (s)		Mean Sol.
	Mat	Py	Mat	Py	Mat	Pyt	Mat	Py	WOA
f_1	**2.77E-08**	3.27E-08	1.01E-07	**9.72E-08**	2.92E-08	**2.36E-08**	**3.45**	15.95	1.41E-30
f_2	**2.74E-05**	4.90E-05	**1.77E-04**	6.53E-03	**1.30E-04**	3.49E-03	**3.20**	10.23	1.06E-21
f_3	1.23E-05	**1.46E-07**	8.63E-05	**1.57E-06**	4.37E-05	**2.54E-06**	**7.71**	19.12	5.39E-07
f_4	**1.43E-04**	1.88E-04	**2.80E-04**	3.80E-04	**6.36E-05**	1.58E-04	**4.77**	13.61	7.26E-02
f_5	**5.48E-03**	5.83E-01	**2.84E+00**	3.97E+00	1.74E+00	**1.07E+00**	**3.32**	10.63	2.79E+01
f_6	6.67E-10	**6.64E-10**	4.29E-09	**2.76E-09**	3.31E-09	**1.22E-09**	**3.07**	10.12	3.11E+00
f_7	**4.47E-04**	9.21E-04	**1.26E-03**	4.69E-03	**4.25E-04**	1.41E-03	**4.97**	14.55	1.43E-03
f_8	**-3834.51**	-3775.28	-3371.78	**-3372.80**	**144.7817**	148.7372	**4.91**	13.93	-5080.76
f_9	**1.61E-06**	9.96E-01	**3.38E-01**	3.65E+00	**4.68E-01**	9.03E-01	**3.27**	10.58	0
f_{10}	1.62E-04	**1.32E-04**	**2.29E-04**	5.69E-04	**2.86E-05**	5.36E-04	**4.96**	16.26	7.40E+00
f_{11}	1.41E-07	**4.84E-08**	**6.29E-07**	3.70E-03	**6.63E-07**	3.43E-03	**3.42**	14.08	2.89E-04
f_{12}	3.07E-04	**3.07E-04**	**3.16E-04**	3.32E-04	**6.05E-06**	2.05E-05	**3.59**	7.51	5.72E-04
f_{13}	-1.03163	**-1.03163**	-1.03163	**-1.03163**	2.09E-12	**1.70E-12**	**2.74**	5.11	-1.03163
f_{14}	**0.39789**	0.39789	0.39789	**0.39789**	9.90E-13	**8.26E-13**	**2.79**	5.79	0.39791
f_{15}	**3.00000**	3.00000	3.00000	**3.00000**	6.30E-11	**4.89E-11**	**2.72**	6.61	3.00000
f_{16}	-3.86278	**-3.86278**	-3.86278	**-3.86278**	1.15E-10	**5.74E-11**	**4.15**	12.60	-3.85616
f_{17}	-3.32200	**-3.32200**	-3.32200	**-3.32200**	2.92E-09	**6.53E-10**	**4.75**	12.17	-2.98105
f_{18}	-10.1532	**-10.1532**	-10.1532	**-10.1532**	5.67E-09	**5.25E-09**	**4.67**	9.92	-7.04918
f_{19}	-10.4029	**-10.4029**	-10.4029	**-10.4029**	5.89E-09	**5.47E-09**	**5.07**	11.10	-8.18178
f_{20}	-10.5364	**-10.5364**	-10.5364	**-10.5364**	5.60E-09	**5.04E-09**	**5.84**	12.87	-9.34238

Regarding computational time, CIOA in MatLab demonstrated the best performance in all functions, requiring approximately 37% of the time needed for Python to solve the same function. The use of certain libraries, such as numpy (in Python) and the authors' greater experience in programming in MatLab, could justify this difference. It is also important to highlight that the Python version of CIOA is an adaptation of the initial version, in MatLab. Therefore, certain performance-enhancing features available in Python may not have been utilized to keep the code as similar as possible to the original MatLab implementation.

4.2 Results for real-world optimization problems

The results for the real-world optimization problems can be seen in Table 3, where they are further compared with those obtained by Sallam et al. [16] who used the Multi-Operator Differential Evolution (MODE) algorithm.

Table 3. Results for real-world problems.

	Results obtained in this paper								[16]
	Best solution		Mean solution		Std. deviation		Time (s)		Best
	MatLab	Python	MatLab	Python	MatLab	Python	MatLab	Python	MODE
R_1	**2994.62**	2994.69	2994.94	**2994.90**	1.48E-01	**1.22E-01**	**2.29**	8.47	2994.4
R_2	**0.07165**	0.13171	**0.16705**	0.24925	**4.44E-02**	6.93E-02	**6.54**	24.62	0.0322
R_3	**0.01267**	0,01267	0.01271	**0,01270**	**3.19E-05**	3,26E-05	**1.46**	3.29	0.0127
R_4	6059.71	**6047.37**	6094.32	**6080.33**	7.50E+01	**1.59E+01**	**1.71**	3.86	6059.7
R_5	**1.67091**	1.67334	**1.68085**	1.68453	**4.72E-03**	5.78E-03	**1.78**	6.49	1.6702
R_6	0.23524	**0.23524**	0.23524	**0.23524**	6.36E-07	**5.09E-07**	**1.96**	7.02	0.2352
R_7	**0.52325**	**0.52325**	**0.52878**	0.52953	2.27E-03	**1.93E-03**	**3.46**	18.37	0.5258
R_8	16.1118	**16.1064**	16.2354	**16.2203**	5.81E-02	**5.14E-02**	**2.13**	11.99	16.070
R_9	**0**	**0**	**0**	**0**	**0.00E+00**	**0.00E+00**	**1.41**	4.43	0
R_{10}	**-30664.2**	-30662.3	**-30654.6**	-30645.5	**5.24E+00**	7.14E+00	**1.64**	6.12	-30666

Regarding the best solution, the MatLab version of the CIOA performed better in 5 problems, while the

Python version performed better in 3. In two problems, both versions achieved the same result. For the mean solution and standard deviation, the MatLab and Python presented the best performance in 4 and 5 problems, respectively, while in 1 problem they demonstrated the same performance. In terms of computational time, the MatLab showed the best performance in all problems, requiring approximately 29% of the time used by Python. Finally, the results obtained in this work are very similar to those reported by Sallam et al. [16].

4.3 Results for truss structural optimization problems

Table 4 presents the results obtained by CIOA in structural optimization problems for trusses. For the best and the mean solutions, MatLab performed better in 3 problems, while Python performed better in only 1. Conversely, for the standard deviation, MatLab was better in 1 problem and Python in 3. Regarding computational time, CIOA in MatLab demonstrated the best performance in all analyses, requiring approximately 29% of the time used by Python. The results for the best solution are compared with those obtained by Miguel and Fadel Miguel [15] using the Firefly Algorithm (FA). It is noted that CIOA achieved better results in two problems.

Table 4. Results for truss structural optimization problems.

	Results obtained in this paper								[15]
	Best Solution		Mean Solution		Std deviation		Time (s)		Best Sol.
	MatLab	Python	MatLab	Python	MatLab	Python	Matlab	Python	FA
S_1	247.29	**247.26**	247.37	**247.32**	0.0615	**0.0424**	**87.38**	263.49	247.31
S_2	**2075.4**	2097.3	**2163.0**	2185.1	44.246	**38.360**	**251.37**	846.02	2058.8
S_3	**195.43**	196.15	**197.96**	198.79	1.7364	**1.4015**	**586.04**	3325.21	193.99
S_4	**17471.7**	17488.3	**17509.3**	17516.6	**23.809**	24.459	**2401.7**	6915.39	17557.5

4.4 Statistical Kruskal-Wallis Test

Finally, a statistical analysis using the Kruskal-Wallis Test is performed considering the results of the 51 independent simulations carried out for each benchmark function and real-world problem in each version of CIOA. The results obtained for the H statistic and p-value at a significance $\alpha = 0.05$ are presented in Table 5.

Table 5. Statistical analyzes

Type	f or R	H	p-value	Win	f or R	H	p-value	Win
	f_1	1.13	0.29	=	$\boldsymbol{f_{11}}$	**33.864**	**0**	**M**
	$\boldsymbol{f_2}$	**49.337**	**0**	**M**	$\boldsymbol{f_{12}}$	**11.719**	**0.0006**	**M**
	$\boldsymbol{f_3}$	**75.757**	**0**	**P**	f_{13}	2.359	0.1245	=
	$\boldsymbol{f_4}$	**20.621**	**0**	**M**	f_{14}	3.832	0.0503	=
Benchmark	$\boldsymbol{f_5}$	**7.977**	**0.0047**	**M**	$\boldsymbol{f_{15}}$	**5.331**	**0.0209**	**P**
Functions	$\boldsymbol{f_6}$	**5.254**	**0.0219**	**P**	$\boldsymbol{f_{16}}$	**8.417**	**0.0037**	**P**
	$\boldsymbol{f_7}$	**70.004**	**0**	**M**	$\boldsymbol{f_{17}}$	**16.777**	**0**	**P**
	f_8	0.015	0.9041	=	f_{18}	0.312	0.5763	=
	$\boldsymbol{f_9}$	**74.135**	**0**	**M**	f_{19}	0.484	0.4864	=
	$\boldsymbol{f_{10}}$	**7.920**	**0.0049**	**M**	f_{20}	0.223	0.6370	=
	R_1	2.755	0.0970	=	R_6	**14.835**	**0.0001**	**P**
Real-world	R_2	**31.268**	**0**	**M**	R_7	3.780	0.0519	=
Problems	R_3	0.598	0.4390	=	R_8	1.643	0.2000	=
	R_4	0.884	0.3470	=	R_9	0	1	=
	R_5	**10.043**	**0.0015**	**M**	R_{10}	**34.569**	**0**	**M**

Whenever p-value > 0.05, it means that the null hypothesis cannot be rejected. In this case, the results generated by the simulations of each version of the algorithm are obtained from the same distribution, making it impossible to determine which version is better. When the p-value < 0.05, the null hypothesis is rejected, indicating that the analyzed data are from different distributions. In these cases, highlighted in bold in Table 5, one version

of the algorithm is statistically better. Thus, in the "Win" column, "M" is used to indicate cases where MatLab performed better (11 functions or problems), and "P" is used for problems where Python performed better (6 functions or problems). It is important to highlight that the statistical analyzes considered values up to the 16th decimal place, therefore, many of the observed differences may be attributed to floating-point arithmetic.

5 Conclusions

This paper presented new applications of the Circle-Inspired Optimization Algorithm (CIOA) to complex optimization problems and presents, for the first time, a comparison between its MatLab and Python versions. The results show that CIOA performs efficiently in both computational languages. Performance differences are statistically significant only beyond a certain decimal precision, suggesting they are due to algorithmic randomness or floating-point arithmetic. MatLab consistently outperforms Python in terms of computational time, likely due to the libraries used and the authors' programming experience. As future work, it is suggested to implement CIOA in object-oriented Python in addition to adapting it to multi-objective optimization problems.

Acknowledgements. The authors acknowledge the financial support of CNPq.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] J. Kennedy and R. Eberhart. "Particle Swarm Optimization". Proceedings of ICNN'95 - *International Conference on Neural Networks*, pp.1942-1948.
[2] R. Storn and K. Price. "Differential Evolution – A simple and efficient heuristic for global optimization over continuous spaces". *Journal of Global Optimization*, vol. 11, n. 4, pp. 341-359, 1997.
[3] Z. W. Geem, J. H. Kim and G. V. Loganathan. "A new heuristic optimization: Harmony Search". *Simulation*, vol. 76, n. 2, pp. 60-68, 2001.
[4] X-S. Yang, "Firefly algorithms for multimodal optimization". In: O. Watanabe e T Zeugmann (eds.), *Fifth International Symposium on Stochastic Algorithms*, pp. 169-178.
[5] M. S. Gonçalves, R. H Lopez and L. F. Fadel Miguel. "Search group algorithm: A new metaheuristic method for the optimization of truss structures". *International Journal Computers & Structures*, vol. 153, pp. 165-184, 2015.
[6] S. Mirjalili and A. Lewis. "The whale optimization algorithm". *International Journal Advances of Engineering Software*, vol. 95, pp. 51-57, 2016.
[7] S. Arora and S. Singh. "Butterfly optimization algorithm: a novel approach for global optimization". *Soft Computing*, v. 23, n. 3, pp. 715-734, 2019.
[8] O. A. P. Souza. *Elaboração de um algoritmo de otimização aplicado à engenharia estrutural: Circle-Inspired Optimization Algorithm*. Masters dissertation, Federal University of Rio Grande do Sul, 2021.
[9] O. A. P. Souza and L. F. F. Miguel. "CIOA: Circle-Inspired Optimization Algorithm, an algorithm for engineering optimization". *SoftwareX*, v. 19, pp. 101192, 2022.
[10] D. Mahato, V. K. Aharwal and A. Sinha. "Electric Vehicle Charge Scheduling Based on Circle-Inspired Optimization Algorithm". In: G. Rajakumar, KL. Du, Á. Rocha (eds.), *Intelligent Communication Technologies and Virtual Mobile Networks*. (ICICV 2023), pp. 539-558.
[11] L. F. F. Miguel and O. A. P Souza. "Robust optimum design of MTMD for control of footbridges subjected to human-induced vibrations via the CIOA". *Structural Engineering and Mechanics*, vol. 86, n. 5, pp. 647-661, 2023.
[12] S. Salim and R. Sarath. "Breast cancer detection and classification using histopathological images based on optimization-enabled deep learning". *Biomedical Engineering: Applications, Basis and Communications*, vol. 36, n. 1, 2023.
[13] A. Kumar, G. Wu, M. Z. Ali, R. Mallipeddi, P. N. Suganthan and S. Das. "A test-suite of non-convex constrained optimization problems from the real-world and some baseline results". *Swarm and Evolutionary Computation*, vol. 56, pp. 100693, 2020.
[14] L. F. F. Miguel and L. F. F. Miguel. "Shape and size optimization of truss structures considering dynamic constraints trough modern metaheuristics algorithms". *Expert Systems with Applications*, vol. 39, pp. 9458-9467, 2012.
[15] L. F. F. Miguel and L. F. F. Miguel. "Assessment of modern metaheuristic algorithms – HS, ABC and FA – in shape and size optimisation of structures with different types of constraints". *Int. J. Metaheuristics*, vol. 2, n. 3, pp. 256-293, 2013.
[16] K. M. Sallam, S. M. Elsayed, R. K. Chakrabortty and M. J. Ryan. "Multi-Operator Differential Evolution Algorithm for Solving Real-World Constrained Optimization Problems." Proceedings of Congress on Evolutionary Computation, 2020, pp.1-8.