Tracking the exploration and exploitation in stochastic population-based nature-inspired algorithms using recurrence plots

Daniel Angus[1] and Iztok Fister Jr.[2⋆]

[1] Digital Media Research Centre, Queensland University of Technology, Brisbane, Queensland 4059, Australia
[2] University of Maribor, Faculty of Electrical Engineering and Computer Science, Koroška cesta 46, Slovenia

Abstract. The success of every stochastic population-based nature-inspired algorithms is characterized through the dichotomy of exploration and exploitation. In general, exploration refers to the evaluation of points in previously untested areas of a search space, while exploitation refers to evaluation of points in close vicinity to previously visited points. How to balance both components properly during the evolutionary process is still considered as a topical problem in the evolutionary computation community. In this paper, we propose a recurrence plot visualization method for evaluating this process. Our analysis shows that recurrence plots are highly appropriate for revealing how particular algorithms balance exploration and exploitation.

Keywords: exploration, exploitation, nature-inspired algorithms, recurrence plot, optimization

1 Introduction

Stochastic population-based nature-inspired algorithms are a kind of search algorithms that are considered as a powerful tool for coping with optimization problems in continuous, as well as discrete, domains. Most of them are inspired by the biological principles of behavior of various animals living in nature, while some of them are even inspired by physical phenomena. Each stochastic population-based nature-inspired algorithm consists of a population of individuals that undergo variation operators during the evolution process and generate a new subsequent population. Despite the popularity of this subject, a lot of different algorithms have been developed in the past decades. Nevertheless, characteristic examples that fit under this umbrella are: Artificial Bee Colony (ABC) algorithm [7], Bat Algorithm (BA) [16], Differential Evolution (DE) [12], Firefly Algorithm (FA) [15], Genetic Algorithm (GA) [5], Particle Swarm Optimization (PSO) [8]. Each algorithm starts with a randomly generated initial population that updates over multiple generations/cycles using specific variation operators. For example,

⋆ Corresponding Author: iztok.fister1@um.si

a GA uses three variation operators: selection, crossover and mutation; while the BA variation operator is guided by the physical phenomenon of echolocation observed in micro-bats. All of these operators influence the diversity of a population, and balance the exploration and exploitation components [1] and, most importantly, determine the overall quality of returned solutions. For this reason, it is important to have deep knowledge of the manner in which parameters of a particular algorithm impact on its search performance, which can help us understand what its weaknesses and advantages are during the evolutionary process. Additionally, such insights can help us to decide which algorithm is good for particular problems, as well as how to approach solving particular problems. We propose the use of recurrence plots to visualize graphically the evolutionary path of various nature-inspired algorithms, and reveal performance over time in a more informative manner than by simply tracking unitary measures such as the single best solution found. To the authors‘ knowledge, there is only one study [13] that used recurrence plots for study phase transitions in swarm optimization algorithms.

The main contributions of this paper are summarized as follows:

- to verify that there is a possibility to track the whole path of a stochastic population-based nature-inspired algorithm during the evolutionary process using recurrence plots,
- to investigate whether there is a possibility to observe changes between the exploration phase as well as the exploitation phase on recurrence plots,
- to study if there is a possibility to decide which algorithm is good for a particular problem based on the visualization of recurrence plots.

The structure of this paper is as follows: In Sec. 2 the stochastic population-based nature-inspired algorithms that are used in our study are outlined, while Sec. 3 and 4 present the methodology. The results of experiments are presented in Sec. 5. Sec. 6 concludes the paper, with remarks for future work.

2 Stochastic population-based nature-inspired algorithms

The purpose of this section is to acquaint the reader with the population-based nature-inspired algorithms [3] that are being used in our experiments.

The Bat Algorithm is an example of Swarm Intelligence (SI) based algorithms [4]. BA is inspired by a physical phenomenon of micro-bats called echolocation. Differential Evolution is an evolutionary algorithm used widely in solving many combinatorial, continuous, as well as real-world problems. DE was proposed by Storn and Price in 1997 [12]. The Firefly Algorithm that was developed by Yang in 2008 is an SI-based algorithm inspired by the mating behavior of fireflies. The phenomenon of fireflies is regarding the flashing lights that attract mating partners on the one hand, while, on the other, it serves as protection mechanism. Particle Swarm Optimization is also a member of SI-algorithms that was first presented in 1995 [8]. The inspirations of PSO lie in the social foraging behavior of some animals, such as the flocking behavior of birds.

[3] sorted alphabetically

3 Recurrence quantification analysis and recurrence plots

The recurrence plotting plot technique was initially invented as a technique to display and identify patterns from time series data, specifically data from high-dimensional dynamical systems [3]. The recurrence plot is a 2D plot where the horizontal and vertical axes represent time series data, and individual elements of the plot indicate times where the phase space trajectory of the system visits the same region of phase space.

While visual inspection of recurrence plots is useful for revealing the structure and dynamics of dynamical systems, Recurrence Quantification Analysis (RQA) extends this technique by specifying a set of metrics designed to capture specific features of recurrence plots [10, 14]. In the 25 years following the original work of Eckmann et al. (1987), recurrence analysis has been applied across diverse areas including financial analysis, neural recordings, engineering, earth science and chemistry [9].

4 Methodology

In order to analyze recurrence plots for tracking the exploration and exploitation of population-based nature-inspired algorithms, we conducted a series of experiments. All experiments are based on the optimization of continuous benchmark functions [6] that are presented in Table 1.

f	Function name	Definition
$f1$	Sphere	$f(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$
$f2$	Ackley	$f(\mathbf{x}) = -a\ \exp\left(-b\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D} cos(c\ x_i)\right) + a + \exp(1)$
$f3$	Griewank	$f(\mathbf{x}) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$
$f4$	Rastrigin	$f(\mathbf{x}) = 10D + \sum_{i=1}^{D} \left(x_i^2 - 10\cos(2\pi x_i)\right)$

Table 1: Benchmark functions used in our experiments.

To generate a recurrence plot, a similarity measurement is required to compare any two points of the time series being plotted on the recurrence plot. In the case here, a single time point is the state of the population of the EC algorithm for a single generation, and, therefore, the similarity measurement is designed to measure the difference between two populations of solutions. The similarity measurement is based on the Euclidean distance between points in the population. The Euclidean distance metric is a relatively straightforward metric to calculate; given two solutions, p and q, and problem dimensionality of d, the Euclidean distance is calculated as:

$$\text{Euclidean distance}(p, q) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2} \tag{1}$$

To obtain a population similarity score, Alg. 1 is applied. According to the Alg. 1, the algorithm returns a pairwise similarity score between any two solutions. Algorithm 1 sums the similarity of every solution between two populations. This sum is then divided by the population size. In line with this, Algorithm 2 presents how final time series are being generated. The algorithm iterates through all iterations, and calculates a similarity score for every two generations, i.e. the current generation and one next generation are taken into account. Finally, all points in the time series are normalized in order to get a similarity value between [0,1].

Algorithm 1 Population similarity score for populations $P1$ and $P2$

```
Score = 0
for i = 1 to NP_P1 do
    for j = 1 to NP_P2 do
        Score+ = Score(P1_i, P2_j)
    end for
end for
Score/ = NP
```

Algorithm 2 Building time series

```
TimeSeries = ∅;
for i = 1 to MAX_ITER do
    Point = Calculate_population_similarity()
    TimeSeries.append(Point)
end for
TimeSeries = normalize(TimeSeries(0, 1))
```

4.1 RQA analysis and generating a recurrence plot

Each population-based nature-inspired algorithm was run for 500 iterations, generating 20 new solutions per iteration. During the evolutionary cycle, we stored all solutions of each iteration. All included algorithms were run on 25 independent runs. Let us mention that the dimension of the problem was set to 30 for all algorithms on every benchmark function. Tables 2, 3, 4, 5 present mean and std. values for each algorithm over the 25 runs. RQA was calculated using PyRQA software [11], while laminarity, divergence, trapping time and determinism measures were taken into account. For generating a recurrence plot, we chose 1 single run randomly from the pool of 25 runs. Recurrence plots were generated using the pyunicorn package [2].

RQA measures attempt to capture moments where a dynamical system under analysis is persisting in a single point in state space, drifting from or between different states, or randomly moving about a state space. RQA analysis is therefore of strong interest here due to its ability to capture aspects of convergent and non-convergent algorithmic behaviour. Laminarity is a measure of intermittent behaviour which will form vertical lines on a recurrence plot. Divergence is the inverse of the maximal diagonal line length which if low would indicate that an algorithm has converged or is moving along a cyclic trajectory through state space. Trapping time is the average length of vertical lines, which indicates the amount of time a system spends in a particular state, for an optimisation algorithm a high trapping time would indicate exploitation behaviour. Determinism is a percentage measure of how many recurrence points form diagonal lines. For determinism to be low a plot will contain mostly random noise (single recurrence dots), rather than longer diagonal lines which would indicate convergent behaviour, therefore low determinism indicates more exploration.

5 Discussion

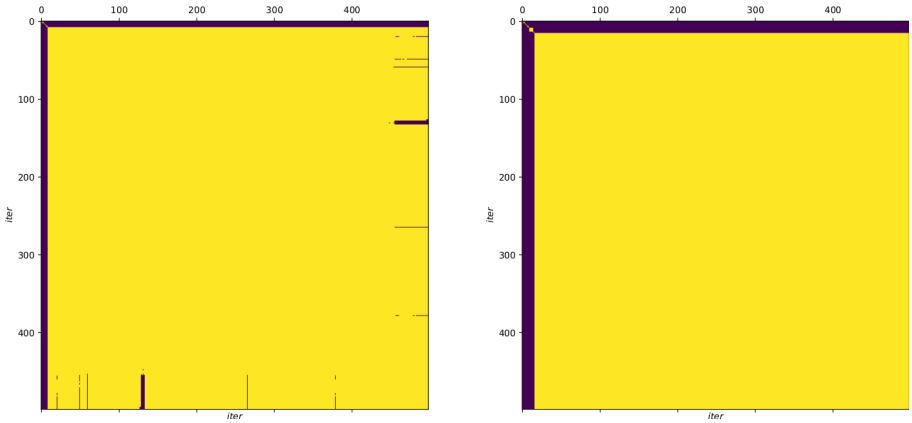
For the algorithms tested we observe both qualitative and quantitatively different results from the recurrence analysis [4]. The BA algorithm produced some of the most interesting results, given that the algorithm seemed to converge within only a handful (10-30 iterations) leading to very large values for the RQA measures and almost complete visual recurrence. At least in the configuration of the algorithm we used this would suggest that the BA algorithm is incredibly quick to converge and that care should be taken in ensuring that population diversity is maintained when in use.

Table 2: RQA of BA.

Function	Measure	*Laminarity*	*Divergence*	*Trappingtime*	*Determinism*
$f1$	Mean	0.9999	0.0026	366.1704	1.0000
	Std.	0.0001	0.0009	143.7695	0.0000
$f2$	Mean	0.9999	0.0021	474.5477	1.0000
	Std.	0.0000	0.0000	20.2955	0.0000
$f3$	Mean	0.9999	0.0023	450.7964	1.0000
	Std.	0.0001	0.0007	92.2660	0.0001
$f4$	Mean	0.9999	0.0021	468.0336	0.9998
	Std.	0.0004	0.0000	24.5921	0.0009

The Firefly algorithm had the second highest values for Determinism, indicating that it too behaved in a highly exploitative fashion. The visual plots for this algorithm does reveal that this exploitation behaviour occurs mostly towards the end of the algorithm run, and that the algorithm seems to move its population slowly through state space, highlighted also by the low divergence scores

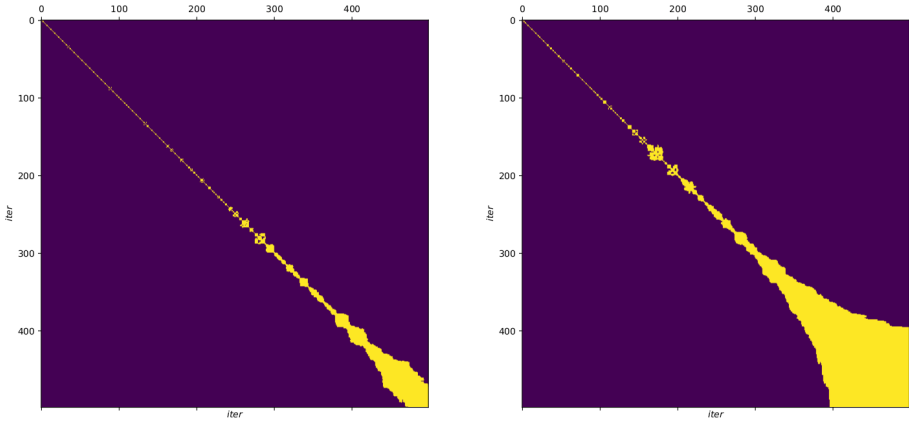
[4] only selected figures are presented in this paper



(a) Ackley (b) Griewank

Fig. 1: Recurrence plots of BA on selected benchmark functions

combined with high laminarity. Of all of the algorithms tested, FA tends to be the one algorithm that tends to transition the smoothest between exploratory and exploitative behaviours.



(a) Ackley (b) Sphere

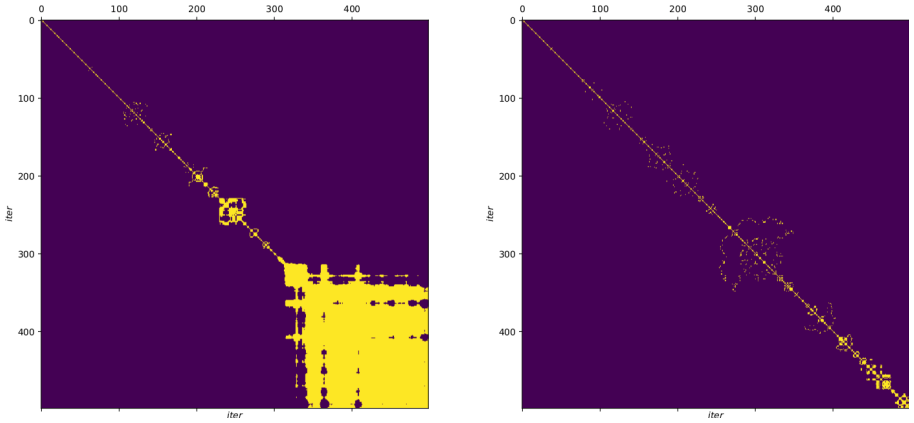
Fig. 2: Recurrence plots of FA on selected benchmark functions

For the DE algorithm there was a notable difference in behaviour on F3 (Griewank), which can be seen both visually and quantitatively. On F3, DE seemed somewhat non-convergent, seen through the lower scores for laminarity, trapping time and determinism, and higher scores for divergence. the Griewank

Table 3: RQA of FA.

Function	Measure	*Laminarity*	*Divergence*	*Trappingtime*	*Determinism*
$f1$	Mean	0.9927	0.0030	52.3893	0.9984
	Std.	0.0003	0.0001	1.4192	0.0004
$f2$	Mean	0.9566	0.0042	20.1938	0.9943
	Std.	0.0012	0.0002	0.6655	0.0021
$f3$	Mean	0.8857	0.0057	11.3448	0.9865
	Std.	0.0053	0.0004	0.6663	0.0059
$f4$	Mean	0.9926	0.0029	51.6059	0.9982
	Std.	0.0003	0.0001	1.0994	0.0003

function quite notably contains a vast number of closely placed local minima, which could explain the algorithms lack of convergence.

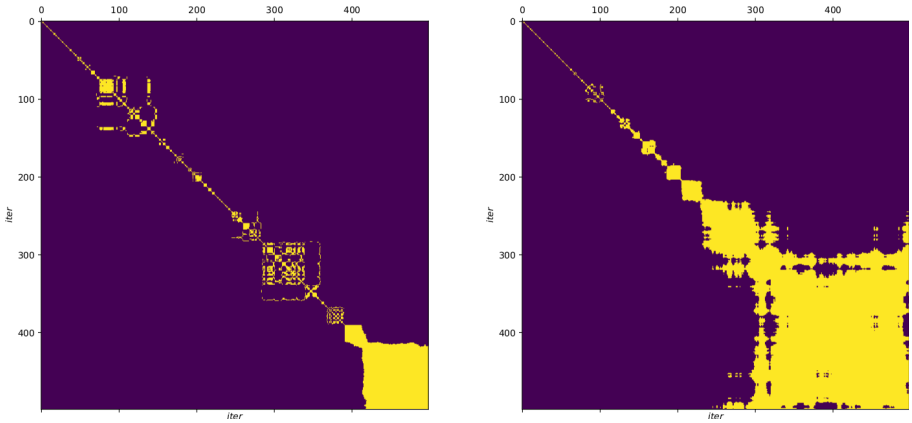


(a) Ackley (b) Griewank

Fig. 3: Recurrence plots of DE 1/2

Table 4: RQA of DE.

Function	Measure	*Laminarity*	*Divergence*	*Trappingtime*	*Determinism*
$f1$	Mean	0.9927	0.0028	39.8657	0.9916
	Std.	0.0048	0.0004	20.5044	0.0078
$f2$	Mean	0.9859	0.0052	72.6573	0.9947
	Std.	0.0140	0.0024	25.9202	0.0079
$f3$	Mean	0.6567	0.0453	4.6795	0.7223
	Std.	0.1937	0.0461	2.4264	0.1807
$f4$	Mean	0.8422	0.0349	20.4438	0.8379
	Std.	0.1005	0.0311	34.5553	0.1179



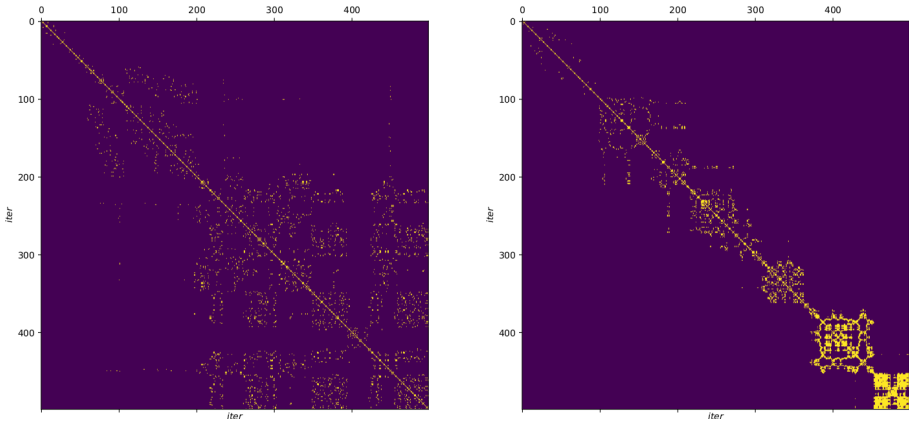
(a) Rastrigin (b) Sphere

Fig. 4: Recurrence plots of DE 2/2

PSO had some of the most exploratory behaviour of all of the algorithms, with the highest divergence values, and lowest trapping time, determinism and laminarity. The recurrence plots for PSO reveal more information though, as one can quite clearly see how this algorithm seems to persist in distinct areas of the state space for tens of iterations. In the case of Rastrigin's function, it is clear that from iteration 200 the algorithm moves back and forth between areas of the state space creating what almost looks like a chess board pattern. Contrasted to FA, the results of PSO look more random and chaotic, rather than smoothly transitioning from one state to the next. In the case of the Sphere function, the PSO algorithm exhibits what could be considered a punctuated equilibrium effect moving from one point of state space, persisting for a time, then moving to another completely different section of state space.

Table 5: RQA of PSO.

Function	Measure	*Laminarity*	*Divergence*	*Trappingtime*	*Determinism*
$f1$	Mean	0.6419	0.0471	4.9304	0.6690
	Std.	0.1690	0.0446	1.9686	0.1788
$f2$	Mean	0.3901	0.1654	3.1237	0.4177
	Std.	0.2301	0.2094	1.0968	0.2347
$f3$	Mean	0.3309	0.1769	2.8711	0.3639
	Std.	0.1698	0.1992	0.8154	0.2046
$f4$	Mean	0.0377	0.8033	nan	0.0346
	Std.	0.0409	0.3003	nan	0.0647



(a) Rastrigin (b) Sphere

Fig. 5: Recurrence plots of PSO on selected benchmark functions

6 Conclusion

In this paper, we applied a well-known visualization technique, recurrence plotting, for tracking the exploration and exploitation of stochastic population-based nature-inspired algorithms. In addition, we also included the companion measures, Recurrence Quantification Analysis, which quantify distinct visual features of recurrence plots.

The resulting plots and RQA measures reveal much detail of the exploratory and exploitative behaviour of the algorithms under study. In the case of PSO, we could see much randomness, contained to a specific areas of search space, before moments where the algorithm jumped to a new area of search space to continue this behaviour anew. For FA, we noted a gradual shift from exploratory to exploitation as the algorithm progressed through its subsequent iterations. In DE disparity was seen in performance on different functions, indicating that recurrence plotting could help reveal disparity in performance within a single algorithmic class on different problems. And, in the case of BA we noted an almost instantaneous convergence behaviour, indicating an issue perhaps with the algorithms ability to trade off between exploration and exploitation over a single optimisation run.

These measures will benefit from more examination across more problem and algorithm classes, however through this modest study we have shown that these plots and measures can reveal much about the population dynamics of optimisation algorithms. The key difference between this and other unitary measures of algorithm performance, is that by taking full account of the population makeup, and change of this makeup over time, we can better ascertain an algorithms trajectory through state space. Future work could also examine the impact of different population similarity measurements on the resulting recurrence plots.

Acknowledgment

Iztok Fister Jr. acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057).

References

1. Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, 45(3):35, 2013.
2. Jonathan F Donges, Jobst Heitzig, Boyan Beronov, Marc Wiedermann, Jakob Runge, Qing Yi Feng, Liubov Tupikina, Veronika Stolbova, Reik V Donner, Norbert Marwan, et al. Unified functional network and nonlinear time series analysis for complex systems science: The pyunicorn package. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 25(11):113101, 2015.
3. J-P Eckmann, S Oliffson Kamphorst, and David Ruelle. Recurrence plots of dynamical systems. *EPL (Europhysics Letters)*, 4(9):973, 1987.
4. Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.
5. David E Goldberg. Genetic algorithms in search, optimization, and machine learning. 1989.
6. Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.
7. Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.
8. J Kennedy and R Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.
9. Norbert Marwan, M Carmen Romano, Marco Thiel, and Jürgen Kurths. Recurrence plots for the analysis of complex systems. *Physics reports*, 438(5-6):237–329, 2007.
10. Norbert Marwan, Niels Wessel, Udo Meyerfeldt, Alexander Schirdewan, and Jürgen Kurths. Recurrence-plot-based measures of complexity and their application to heart-rate-variability data. *Physical review E*, 66(2):026702, 2002.
11. Tobias Rawald, Mike Sips, and Norbert Marwan. Pyrqaconducting recurrence quantification analysis on very long time series efficiently. *Computers & Geosciences*, 104:101–108, 2017.
12. Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.
13. Tomáš Vantuch, Ivan Zelinka, Andrew Adamatzky, and Norbert Marwan. Phase transitions in swarm optimization algorithms. In *International Conference on Unconventional Computation and Natural Computation*, pages 204–216. Springer, 2018.
14. Charles L Webber Jr and Joseph P Zbilut. Dynamical assessment of physiological systems and states using recurrence plot strategies. *Journal of applied physiology*, 76(2):965–973, 1994.

15. Xin-She Yang. Firefly algorithm, stochastic test functions and design optimisation. *International Journal of Bio-Inspired Computation*, 2(2):78–84, 2010.
16. Xin-She Yang. A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pages 65–74. Springer, 2010.