ordpy: A Python package for data analysis with permutation entropy and ordinal network methods

Cite as: Chaos 31, 063110 (2021); doi: 10.1063/5.0049901
Submitted: 10 March 2021 · Accepted: 18 May 2021 · Published Online: 7 June 2021

Arthur A. B. Pessa and Haroldo V. Ribeiro

AFFILIATIONS
Departamento de Física, Universidade Estadual de Maringá, Maringá, PR 87020-900, Brazil

*Author to whom correspondence should be addressed: arthur_pessa@hotmail.com

ABSTRACT
Since Bandt and Pompe’s seminal work, permutation entropy has been used in several applications and is now an essential tool for time series analysis. Beyond becoming a popular and successful technique, permutation entropy inspired a framework for mapping time series into symbolic sequences that triggered the development of many other tools, including an approach for creating networks from time series known as ordinal networks. Despite increasing popularity, the computational development of these methods is fragmented, and there were still no efforts focusing on creating a unified software package. Here, we present ordpy (http://github.com/arthurpessa/ordpy), a simple and open-source Python module that implements permutation entropy and several of the principal methods related to Bandt and Pompe’s framework to analyze time series and two-dimensional data. In particular, ordpy implements permutation entropy, Tsallis and Rényi permutation entropies, complexity–entropy plane, complexity–entropy curves, missing ordinal patterns, ordinal networks, and missing ordinal transitions for one-dimensional (time series) and two-dimensional (images) data as well as their multiscale generalizations. We review some theoretical aspects of these tools and illustrate the use of ordpy by replicating several literature results.

Published under an exclusive license by AIP Publishing.

Permutation entropy is a complexity measure and data analysis tool stemming from nonlinear time series analysis and information theory. Almost two decades since its conception, this method has gained prominence and has been extensively studied and used by researchers from several fields. The concept of ordinal patterns introduced with permutation entropy has also inspired a whole ecosystem of related techniques, including an approach to map time series into networks known as ordinal networks. However, this ecosystem of tools still lacks a more comprehensive numerical implementation, limiting the further spreading of ordinal methods, especially to fields with less tradition in developing computational tools. In this article, we present ordpy—an open-source Python package that implements several tools related to ordinal patterns for the analysis of time series and images. We present ordpy-functionality together with a review of all pertinent theoretical developments, replicate several literature results, and highlight possible developments of ordinal methods that can be explored with ordpy.

I. INTRODUCTION

Stemming from a combination of ideas from nonlinear time series analysis and information theory, permutation entropy was first introduced in 2002 by Bandt and Pompe as a simple, robust, and computationally efficient complexity measure for time series. This complexity measure is defined as the Shannon entropy of a probability distribution associated with ordinal patterns evaluated from partitions of a time series—a procedure known as the Bandt–Pompe symbolization approach. Permutation entropy and its underlying symbolization approach have become increasingly popular among researchers working with time series analysis, leading to successful applications in fields such as biomedical sciences, econophysics, physical sciences, and engineering. The uses of permutation entropy also span a large variety of goals such as monitoring the dynamical regime of a system, detecting anomalies in time series, characterizing time series data, and testing for serial independence and are further documented in review articles by Zanin et al., Riedl et al., Amigo et al., and Keller et al.
Permutation entropy’s success is not limited to its practical usage as this approach has inspired numerous time series analysis tools. Some of these related methods consider different quantifiers for the ordinal probability distribution. \cite{15, 16, 17} generalize the Bandt–Pompe symbolization algorithm to evaluate ordinal structures on multiple temporal scales, \cite{18, 19} include signal amplitude information, \cite{20} and account for equal values in time series. \cite{21}

Other works have generalized permutation entropy and its ordinal approach to two-dimensional data such as images. \cite{22, 23} The ordinal patterns undergoing permutation entropy have also been used for mapping time series and images into networks known as ordinal networks. \cite{24, 25, 26}

The original version of permutation entropy and its various generalizations represent an essential and appealing framework for data analysis, especially when considering the increasing availability of large data sets and the steady demand for reliable and computationally efficient methods for extracting meaningful information from these data sets. \cite{27, 28} However, most methods emerging from Bandt and Pompe’s seminal work lack freely available computational implementation, and the exceptions are limited to a single approach or very few approaches. Here, we help to fill this gap by presenting ordpy—a simple and open-source Python module implementing several of the principal methods related to Bandt and Pompe’s seminal work lack freely available computation entropy and its related ordinal methods. Our work alternates with a review of the pertinent theoretical developments of permutation entropy and its symbolic representations. This latter scheme was initially proposed in Bandt and Pompe. \cite{29, 30} As we shall see, the choices of \(d_t\) and \(\tau_t\) are important, and there is research exclusively devoted to determining optimal values for these parameters.

Next, for each partition \(w_j\), we evaluate the permutation \(\pi_{j} = \left( r_{j_{1}}, r_{j_{2}}, \ldots, r_{j_{d_t}} \right) \) of the index numbers \((0, 1, \ldots, d_t − 1)\) that sorts the elements of \(w_j\) in an ascending order, that is, the permutation of the index numbers defined by the inequality \( x_{pi_{j_k}} ≤ x_{pi_{j_{k+1}}} ≤ \cdots ≤ x_{pi_{j_{d_t}}} \). In case of equal values, we maintain the occurrence order of the partition elements, that is, if \( x_{pi_{j_k}} = x_{pi_{j_{k+1}}} \), then \( r_{j_{k+1}} < r_{j_{k}} \) for \( k = 1, \ldots, d_t − 1 \). As an illustration, suppose we have \( x = (5, 3, 2, 2, 7, 9) \) and set \( d_t = 4 \) and \( \tau_t = 1 \). The first partition is \( w_1 = (5, 3, 2, 2) \), and sorting its elements, we find \( 2 \leq 2 < 3 < 5 \) and \( x_{i2} < x_{i1} < x_{i4} < x_{i3} \). Thus, the permutation symbol (ordinal pattern) associated with \( w_1 \) is \( \pi_1 = (2, 3, 1, 0) \). Another possibility for dealing with ties among partition elements consists of adding a small noise perturbation without modifying all other ordering relations. This latter scheme was initially proposed in Bandt and Pompe’s seminal work, but it is much less used in the literature. In most cases, time series data have enough resolution for making these equalities negligible; however, this issue can become critical for low-resolution signals. \cite{31, 32}

After evaluating the permutation symbols associated with all data partitions, we obtain a symbolic sequence \( \{ \pi_j \}_{j=1, \ldots, m} \). The ordpy’s function ordinal_sequence returns this sequence as illustrated in the following code:

```python
>>> from ordpy import ordinal_sequence
>>> x = [5, 3, 2, 2, 7, 9]
>>> ordinal_sequence(x, dx=4, taux=1)
array([[2, 3, 1, 0],
       [1, 2, 0, 3],
       [0, 1, 2, 3]])
```

The last two examples illustrate the use of the parameter \( \text{tie\_precision} \) that defines the number of decimals considered for establishing the ordinal relations. This parameter is available in most ordpy’s functions and is particularly relevant when working with time series presenting equal values that could be mistaken.
FIG. 1. The symbolization process of Bandt and Pompe. (a) Illustration of the Bandt and Pompe method applied to a time series and the resulting ordinal sequences for embedding parameters $d_x = 3$ and $\tau_x = 1$ (left) and $d_x = 2$ and $\tau_x = 2$ (right). (b) Application of the two-dimensional version of the Bandt–Pompe method to a data array and the resulting ordinal sequences for embedding parameters $d_x = d_y = 2$ and $\tau_x = \tau_y = 1$ (left) and $d_x = d_y = 2$ and $\tau_x = \tau_y = 2$ (right). In both panels, the colored boxes indicate the data partitioning scheme for each set of embedding parameters, and the numbers above them represent the index numbers used for determining the ordinal patterns or permutation symbols.

by floating-point number representation. Figure 1(a) illustrates the application of the Bandt–Pompe approach for a simple time series and different values of $d_x$ and $\tau_x$.

The ordinal probability distribution $P = \{\rho_i(\Pi_i)\}_{i=1}^{n_x}$ is simply the relative frequency of all possible permutations within the symbolic sequence; that is,

$$\rho_i(\Pi_i) = \frac{\text{number of partitions of type } \Pi_i \text{ in } \{\pi_p\}}{n_x}. \quad (2)$$
where \( \Pi_i \) represents each one of the \( n_x = d^l \) different ordinal patterns. The following code shows how to obtain the ordinal distribution with \texttt{ordpy}'s function \texttt{ordinal_distribution}:

```python
>>> from ordpy import ordinal_distribution
>>> x = [5, 3, 2, 2, 7, 9]
>>> pis, rho = ordinal_distribution(x, dx=3)
```

The two arrays returned by \texttt{ordinal_distribution} are the ordinal patterns and their corresponding relative frequencies, respectively. By default, \texttt{ordinal_distribution} does not return non-occuring permutations [that is, those with \( \rho(\Pi_i) = 0 \)]; however, the parameter \texttt{return_missing} modifies this behavior as in

```python
>>> from ordpy import ordinal_distribution
>>> x = [5, 3, 2, 2, 7, 9]
>>> pis, rho = ordinal_distribution(x, dx=3, return_missing=True)
```

Missing permutation symbols are always the latest elements of the returned array.

Having the ordinal probability distribution \( P \), we can calculate its Shannon entropy and define the permutation entropy as

\[
S(P) = -\sum_{i=1}^{n_x} \rho(\Pi_i) \log \rho(\Pi_i),
\]

where \( \log(...) \) stands for the base-2 logarithm. Permutation entropy quantifies the randomness in the ordering dynamics of a time series such that \( S \approx \log n_x \) indicates a random behavior, while \( S \approx 0 \) implies a more regular dynamics. Because the maximum value of \( S \) is \( S_{\text{max}} = \log n_x \), we can further define the normalized permutation entropy as

\[
H(P) = \frac{S(P)}{\log n_x},
\]

where the values of \( H \) are restricted to the interval \([0,1]\). The \texttt{ordpy}'s function \texttt{permutation_entropy} calculates the values of \( S \) and \( H \) directly from a time series as illustrated in

```python
>>> from ordpy import permutation_entropy
>>> x = [5, 3, 2, 2, 7, 9]
>>> permutation_entropy(x)
```

The permutation_entropy function uses the base-2 logarithm function by default; however, the parameter \texttt{base} can modify this behavior.

The embedding dimension \( d \) defines the number of possible permutations \( n_x = d^l \), and following Bandt and Pompe's recommendation,

\[
\text{it is common to choose the values of } d \in \{3, 4, 5, 6, 7\} \text{ to satisfy the condition } d^l \leq N_x \text{ to obtain a reliable estimate of the ordinal probability distribution. Another less common choice is to use a value of } d \text{ such that } 5d^l \leq N_x.
\]

More recently, however, Cuesta-Frau et al.\(^{[43]}\) have shown that these requirements on \( d \) can be considerably loosened in several situations related to classification tasks. The embedding delay \( \tau \) defines a time scale for the system under analysis and is often set as 1; however, different values of \( \tau \) may inform about delayed feedback mechanisms and time-correlation structures. We present a more detailed discussion about the choices of \( d \) and \( \tau \) in Appendix A.

The permutation entropy framework was extended to two-dimensional data by Ribeiro et al.\(^{[39]}\) and Zunino and Ribeiro.\(^{[40]}\) To present this generalization, let us consider an arbitrary two-dimensional data array \( \{y_{t+1}, \ldots, y_T\} \), whose elements may represent pixels of an image. We further define the embedding dimensions \( d_x \) and \( d_y \) along the horizontal and vertical directions (respectively) and the corresponding embedding delays \( \tau_x \) and \( \tau_y \). Similarly to the one-dimensional case, we slice the data array in partitions of size \( d_x \times d_y \) defined by

\[
w^d = \left( y^d_{1:\tau_x-1} y^d_{\tau_x} \ldots y^d_{\tau_x+1:\tau_x-1} \right),
\]

where the indexes \( p = 1, \ldots, n_x \) and \( q = 1, \ldots, n_y \) with \( n_x = N_T - (d_x - 1)\tau_x \) and \( n_y = N_T - (d_y - 1)\tau_y \), cover all \( n_xn_y \) data partitions. To associate a permutation symbol with each two-dimensional partition, we flatten the partitions \( w^d_p \) line by line; that is,

\[
w^d_p = \left( y^p_{\tau_x+1}, \ldots, y^p_{\tau_x+d_y-1} \right), \ldots, \left( y^p_{\tau_x+d_y-1}, \ldots, y^p_{\tau_x+d_y-1+d_x-1} \right).
\]

As this procedure does not depend on a particular partition, we can simplify the notation by representing \( w^d_p \) as

\[
w^d_p = \left( y_{\tau_x+1}, \ldots, y_{\tau_x+d_y-1} \right), \ldots, \left( y_{\tau_x+d_y-1}, \ldots, y_{\tau_x+d_y-1+d_x-1} \right).
\]

where \( y_{\tau_x+1} = y^p_{\tau_x+1}, \ldots, y_{\tau_x+d_y-1} = y^p_{\tau_x+d_y-1} \) and so on. Then, we evaluate the permutation symbol associated with each data partition as in the one-dimensional case to define the symbolic array \( \{\Pi^d\}_{p=1}^{n_xn_y} \) related to the data set \( \{y_{t+1}, \ldots, y_T\} \). Figure 10(b) illustrates the Bandt–Pompe approach...
for two-dimensional data. From this array, we calculate the relative frequency for all \( n_\pi = (d_d,d_d)! \) possible permutations \( \Pi_n \)

\[
\rho_i(\Pi_i) = \frac{\text{number of partitions of type } \pi_i}{n_\pi},
\]

where \( i = 1, \ldots, n_\pi \), and therefore, the ordering probability distribution is \( P = [\rho_i(\Pi_i)]_{i=1}^{n_\pi} \). It is worth noticing that the ordering procedure defining the permutation symbols is no longer unique as in the one-dimensional case. For instance, we would find a different symbolic array by flattening the partitions \( w_j \) column by column. However, different ordering procedures do not modify the set of elements comprising the ordinal probability distribution (only their order is changed).

As in the one-dimensional case, the two-dimensional permutation entropy is simply the Shannon entropy of the ordinal distribution \( P = [\rho_i(\Pi_i)]_{i=1}^{n_\pi} \); therefore, we can calculate the two-dimensional permutation entropy and its normalized version using Eqs. (3) and (4), respectively. Only the total number of possible ordinal patterns \( n_\pi = (d_d,d_d)! \) in the two-dimensional case is modified.

Similarly to the one-dimensional case, the values of \( d_d \) and \( d_d \) are usually constrained by the condition \((d_d,d_d)! \ll N,N \) in order to obtain a reliable estimate of the ordinal distribution \( P \). Naturally, this two-dimensional formulation recovers the one-dimensional case \((N = 1 \text{ for time series data})\) by setting \( d_d = d_d = 1 \). In ordy, the functions \texttt{ordinal_sequence}, \texttt{ordinal_distribution}, and \texttt{permutation_entropy} automatically implement this two-dimensional generalization when the input data are a two-dimensional array as in

\[
\texttt{... from ordy import ordinal_sequence, \ldots ordinal_distribution, permutation_entropy} \\
\texttt{... ordinal_sequence(y, dx=2, dy=2)} \\
\texttt{... ordinal_distribution(y, dx=2, dy=2)} \\
\texttt{... permutation_entropy(y, dx=2, dy=2)} \\
\texttt{0.21810429198553155}
\]

In addition to permutation entropy, the complexity–entropy plane proposed by Rosso et al. \cite{rosso} is another popular time series analysis tool directly related to Bandt and Pompe’s symbolization approach. This method was initially introduced for distinguishing between chaotic and stochastic time series but has been successfully used as an effective discriminating tool in several other contexts.\textsuperscript{60–62} The complexity–entropy plane combines the normalized permutation entropy \( H \) [Eq. (4)] with an intensive statistical complexity measure \( C \) (also calculated using the ordinal distribution) to build a two-dimensional representation space with the values of \( C \) vs \( H \). The statistical complexity \( C \) used by Rosso et al. is inspired by the work of López-Ruiz et al. \cite{lopezruiz} and is defined by the product of the normalized permutation and a normalized version of the Jensen–Shannon divergence\textsuperscript{63} between the ordinal distribution \( P = [\rho_i(\Pi_i)]_{i=1}^{n_\pi} \) and the uniform distribution \( U = [1/n_\pi]_{i=1}^{n_\pi} \)

\[
C(P) = \frac{D(P, U)H(P)}{D_{\text{max}}},
\]

where

\[
D(P, U) = S((P + U)/2) - \frac{1}{2}S(P) - \frac{1}{2}S(U)
\]

is the Jensen–Shannon divergence and

\[
D_{\text{max}} = \frac{1}{2} \left( \frac{n_\pi}{n_\pi - 1} \log(n_\pi + 1) - 2 \log(2n_\pi) + \log(n_\pi + 1) \right)
\]

is a normalization constant. This latter constant expresses the maximum possible value of \( D(P, U) \) occurring for \( P = [\delta_{ij}]_{i=1}^{n_\pi} e^{\pi/4} \), where \( \delta_{ij} = 0 \) if \( i=j \) is the Kronecker delta function.

Differently from permutation entropy, the statistical complexity \( C \) is zero in both extremes of order (when only one permutation symbol occurs) and disorder (when all permutations are equally likely to happen). The value of \( C \) quantifies structural complexity and provides additional information that is not carried by the value of \( H \). Furthermore, \( C \) is a nontrivial function of \( H \) in the sense that for a given value of \( H \), there exists a range of possible values for \( C \).\textsuperscript{60–62} This happens because \( H \) and \( D \) are expressed by different sums of \( \rho_i(\Pi_i) \), and there is thus no reason for assuming a univocal relationship between \( H \) and \( C \).

To better illustrate this feature, let us assume (for simplicity) we replace the Jensen–Shannon divergence by the Euclidean distance between \( P \) and \( U \) (as in the seminal work of López-Ruiz et al.\cite{lopezruiz}); that is, \( D(P, U) = \sum_{i=1}^{n_\pi} (\rho_i(\Pi_i) - 1/n_\pi)^2 \). In this case, the statistical complexity is

\[
C(P) \propto -\left( \sum_{i=1}^{n_\pi} (\rho_i(\Pi_i)) \log \rho_i(\Pi_i) \right) \left( \sum_{i=1}^{n_\pi} (\rho_i(\Pi_i) - 1/n_\pi)^2 \right),
\]

and we can readily observe that different ordinal distributions \( P = [\rho_i(\Pi_i)]_{i=1}^{n_\pi} \) may lead to the same value of \( H \) but different values of \( C \) (or vice versa). Let us further consider a particular ordinal distribution with three possible permutation symbols [this would be equivalent to having \( d_d = 3 \) or \( (d_d,d_d)! = 3 \), if possible], that is, \( P = [a,b,1-(a+b)] \), where \( a > 0 \) and \( b > 0 \) are real numbers such that \( (a+b) \leq 1 \) (to ensure the normalization of \( P \)). For this case, we have \( S = -a \log a - b \log b - [1-(a+b)] \log[1-(a+b)] \) and \( D = (a-1/3)^2 + (b-1/3)^2 + [(1-(a+b)-1/3)^2] \). Thus, for instance, if \( a = 0.79 \) and \( b = 0.18 \) or \( a = 0.80 \) and \( b = 0.16 \), we find the same value of \( H = S/\log 3 \approx 0.55 \) but different values for \( D \) (0.32 in the first case and 0.33 in the second) and, consequently, for \( C \).
Furthermore, the complexity–entropy plane was generalized for two-dimensional data (notice that the only changes are related to the process of estimating the ordinal distribution), and the complexity_entropy function also accepts two-dimensional arrays as input as shown in

```python
>>> from ordpy import complexity_entropy
>>> complexity_entropy([[1, 2, 1], [8, 3, 4],...
... [6, 7, 5]], dx=2, dy=2)
(0.3271564379782973, 0.2701200547320647)
```

III. APPLICATIONS OF BANDT AND POMPE’S FRAMEWORK WITH ordpy

This section presents more engaging applications of ordpy’s functions by replicating literature results. We start by determining the ordinal probability distributions of two different time series of stochastic and chaotic nature, namely, a random walk with Gaussian steps and the logistic map at fully developed chaos (see Appendix B for definitions). We choose these two examples because their ordinal distributions are exactly known for some combinations of the embedding parameters (notice that the only changes are defined in Bandt and Pompe’s work). More specifically, for \(d_x = 3\) and \(\tau_x = 1\), the probability distributions associated with the permutation symbols \((0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)\) are \(P_{\text{walk}} = \{1/4, 1/8, 1/8, 1/4\}\) and \(P_{\text{logistic}} = \{1/3, 1/15, 2/15, 3/15, 4/15, 0\}\) for the random walk and the logistic map, respectively.

To numerically estimate these two ordinal distributions, we generate a time series from a Gaussian random walk process and another time series from iterations of the fully chaotic logistic map. In both cases, we have simulated one realization of each process with \(10^6\) observations and used the ordinal_distribution function. Figure 2 shows that the exact ordinal distributions are in excellent agreement with simulated results obtained with ordpy. It is intriguing to observe that the ordinal pattern \((2, 1, 0)\) ("descending permutation") does not occur in the logistic series (it has probability zero). This fact is best understood as a feature directly associated with the intrinsic determinism of the logistic map dynamics. As we shall discuss in Sec. IV, investigations about such "missing ordinal patterns" are also useful for characterizing time series dynamics.

To better illustrate the use of the permutation_entropy function, we partially reproduce Bandt and Pompe’s analysis of the logistic map (Fig. 2 of Ref. 4). We generate time series consisting of \(10^6\) iterations of the logistic map for each value of parameter \(r \in \{3.5, 3.5001, 3.5002, \ldots, 4.0\}\) (see Appendix B for definitions). Next, we calculate the permutation entropy \(S\) for each of these 5001 time series using permutation_entropy with embedding parameters \(d_x = 6\) and \(\tau_x = 1\). We further divide the permutation entropy by 5 to obtain the permutation entropy per symbol of order 6; that is, \(h_6 = 5/5\) as defined in Bandt and Pompe’s work.

Figure 3(a) depicts the well-known bifurcation diagram for the logistic map, while Fig. 3(b) shows the values of \(h_6\) as a function of the parameter \(r\). We note that the permutation entropy per symbol has an overall increasing trend with the parameter \(r\), marked by abrupt drops in intervals of \(r\) related to periodic behaviors. As noticed by Bandt and Pompe, the behavior of the permutation entropy is similar to the one observed for the Lyapunov exponent.

In another example with permutation_entropy, we replicate a numerical experiment of Cao et al. (see their Fig. 1) that searches for dynamical changes in the transient logistic map time series (see Appendix B for definitions). This problem illustrates the role of the embedding delay \(\tau_x\). As in the original article, we iterate the transient logistic map starting with the initial condition \(x_0 = 0.65\) and incrementing the logistic parameter \(r\) from 2.8 to 4 in
steps of size $10^{-5}$. This process generates a time series with 120 001 observations as shown in Fig. 3(c). Using this time series, we calculate the normalized permutation entropy within a sliding window with 1024 observations for the embedding dimension $d_x = 5$ and two values for the embedding delay ($\tau_x = 1$ and $\tau_x = 2$).

As Cao et al., we denote the permutation entropy values by $H_r(t)$, where $r(t)$ represents the logistic parameter at the end of the sliding window. Figure 3(d) shows the values of $H_r(t)$, where abrupt changes are clearly associated with dynamical changes observed in the time series [Fig. 3(c)]. Despite the overall similarities, we note that the embedding delay $\tau_x = 2$ identifies these dynamical changes better than the case with $\tau_x = 1$; for instance, the transition from period-8 to period-16 (at $r \approx 3.56$) is missed when $\tau_x = 1$ but captured when $\tau_x = 2$.\textsuperscript{20}

As we have mentioned, a generalization of permutation entropy to two-dimensional data was first proposed by Ribeiro et al.\textsuperscript{35} To illustrate the use of the permutation entropy function with two-dimensional data, we replicate a numerical experiment related to Ising surfaces (see Appendix C for definitions) present in that work (Fig. 8 of Ref. 35). These surfaces represent the accumulated sum of spin variables of the canonical two-dimensional Ising model in a Monte Carlo simulation. Figure 3(e) shows four examples of these surfaces (square lattices of size 250 × 250) obtained after $10^6$ Monte Carlo steps for different reduced temperatures $T_r$. We notice non-trivial patterns emerging when the reduced temperature is equal to the critical temperature ($T_r = 1$) of phase transition for the Ising model.\textsuperscript{35} Following the original article, we generate Ising surfaces (size 250 × 250) for reduced temperatures $T_r \in [0.5, 0.6, \ldots, 3.0]$ and calculate their normalized permutation entropy with $d_x = 5$ and $d_y = 2$, and $d_x = 2$ and $d_y = 3$, both for $\tau_x = \tau_y = 1$. In agreement with Ribeiro et al.,\textsuperscript{35} Fig. 3(f) shows that the permutation entropy precisely identifies the phase transition of the Ising model (the sudden decrease around the critical temperature) and that these Ising surfaces are symmetric under reversal of the embedding dimensions.

The complexity_entropy function simultaneously calculates the permutation entropy and the statistical complexity from time series and image data. To illustrate its usage, we partially
reproduce the results of Rosso et al.\(^4\) (Fig. 1 in that work) on distinguishing chaotic from stochastic time series. By following their article, we iterate four discrete maps to generate chaotic series. Specifically, we obtain chaotic time series from a skew tent map (parameter \(w = 0.1847\)), a Hénon map (\(x\)-component, parameters \(a = 1.4\) and \(b = 0.3\)), a logistic map (\(r = 4\)), and a Schuster map (parameter \(z \in (3/2, 2, 5/2)\))—see Appendix B for definitions. We further generate stochastic series from three stochastic processes: noises with \(1/f^4\) power spectrum (for \(k \in [0.00, 0.25, \ldots, 3.00]\)), fractional Brownian motion (Hurst exponent \(h \in [0.1, 0.2, \ldots, 0.9]\)), and fractional Gaussian noise (also \(h \in [0.1, 0.2, \ldots, 0.9]\))—see Appendix C for definitions. For each of these maps and stochastic processes, we generate ten time series with \(2^{15}\) observations and random initial conditions. Next, we use \(\text{complexity}_{\text{entropy}}\) with embedding parameters \(d_x = 6\) and \(r_x = 1\) to calculate their statistical complexity and permutation entropy (average values over ten time series realizations).

As in the original work of Rosso et al., Fig. 4(a) shows that chaotic series usually have high complexity and low entropy values. Stochastic time series, in turn, display high entropy and intermediary complexity values. It is also interesting to note that stochastic time series approach the lower-right corner of the complexity–entropy plane (\(H \to 1\) and \(C \to 0\)) as the serial auto-correlation decreases.\(^7\) These results also illustrate that some stochastic and chaotic series have very similar entropy values but different statistical complexity (for instance, fractional Brownian motion with \(h = 0.9\) and Schuster map with \(z = 3/2\)), confirming that the statistical complexity extracts additional information from the ordinal distribution. In this figure, we have also included two solid lines delimiting the accessible region of the complexity–entropy plane.\(^8\)

In \texttt{ordpy}, the functions \texttt{maximum\_complexity\_entropy} and \texttt{minimum\_complexity\_entropy} generate these curves, as shown in the following code snippet:

```python
>>> from ordpy import ...
... maximum\_complexity\_entropy,
... minimum\_complexity\_entropy
>>> maximum\_complexity\_entropy(dx=4)
array([[-0.21810429, 0.19670592],
      [0.34568712, 0.28362016],
      ...
      [0.98660828, 0.02388382]])
>>> minimum\_complexity\_entropy(dx=4)
array([[-0.00000000e+00, -0.00000000e+00],
      [2.67076969e-02, 2.55212327e-02],
      ...
      [1.00000000e+00, -3.66606083e-16]])
```

The \texttt{complexity\_entropy} function also works with two-dimensional data, and to illustrate its usage, we follow Sigaki et al.\(^9\) and use the complexity–entropy plane to investigate patterns in art paintings. Due to the large-scale data analyzed by Sigaki et al. and to keep our examples self-contained, we do not reproduce their original results but simply use their ideas to illustrate how complexity and entropy extract useful information from images. To do so, we handpick three paintings from https://www.wikiart.org/ (in the original article, the authors studied 137 364 images obtained from the same webpage). These are a Color Field Painting artwork (Blue, 1953 by Ad Reinhardt, image size 768 × 435\(^\text{10}\)), a Brazilian Modernist artwork (Abaporu, 1928 by Tarsila do Amaral, image size 1200 × 1026\(^\text{11}\)), and an American Abstract Expressionist painting

![FIG. 4. Complexity–entropy plane for one- and two-dimensional data. (a) Average values of the statistical complexity \(C\) vs the normalized permutation entropy \(H\) (over ten realizations) evaluated from time series of chaotic maps and stochastic processes. The embedding parameters are \(d_x = 6\) and \(r_x = 1\). The solid lines represent the maximum and minimal possible values of complexity for a given entropy (for \(d_x = 6\) and \(r_x = 1\)). (b) Localization of three art paintings in the complexity–entropy plane with embedding parameters \(d_x = d_y = 2\) and \(r_x = r_y = 1\). All data and code necessary to reproduce this figure are available in a Jupyter notebook at ordpy’s webpage (http://github.com/arthurpessa/ordpy).](Image 457x317 to 496x363)
IV. MISSING ORDINAL PATTERNS

As we have commented, the logistic map at fully developed chaos does not exhibit the “descending permutation” \((2, 1, 0)\) for \(d = 3\) [see Fig. 2(b)]. This feature is not a particularity of the logistic map. Indeed, these missing ordinal patterns (also called forbidden patterns) occur in different systems, and simple statistics associated with them have proven to be useful and reliable indicators of a system’s dynamics.\(^{65-67}\) The works of Amigó et al.\(^{67}\) are seminal in this regard, and by following their classification, we can divide these forbidden ordinal patterns into two categories: true or false.\(^{67}\) True forbidden patterns [such as the \((2, 1, 0)\) in the logistic map] are a fingerprint of determinism in a time series dynamics and represent an intrinsic feature of the underlying dynamical process;\(^{67}\) that is, these patterns are not an artifact related to the finite length of empirical observations. In turn, false forbidden patterns are related to the finite length of time series\(^{67}\) and can emerge even from completely random processes.

This distinction is not straightforward when dealing with empirical data, but a typical analysis in this context consists of investigating the number of missing patterns \((n)\) as a function of the time series length \((N)\). The behavior of this curve is useful for discriminating time series. In "ordpy," the `missing_patterns` function identifies missing ordinal patterns and estimates their relative frequency as in

```python
>>> from ordpy import missing_patterns
>>> missing_patterns({4, 7, 9, 10, 6, 11, 3, 5, ... 6, 2, 3, 1}, dx=3)
{array([[0, 2, 1],
        [2, 1, 0]]),
  0.3333333333333333}
```

To better illustrate the use of this function, we investigate missing ordinal patterns in time series obtained from the logistic map at fully developed chaos \((r = 4)\) and Gaussian random walks. In both cases, we use the embedding dimensions \(d_1 = 5\) and \(d_2 = 6\) (with \(\tau_1 = 1\) and series lengths \(N_1 \in \{80, 150, 240, \ldots, 6000\}\). Figure 5(a) shows the results. We observe that the number of missing ordinal patterns approaches zero as the time series length of random walks increases. Conversely, the number of missing permutations related to the logistic map displays an initial decay with the time series length, but it rapidly saturates in considerably large numbers, indicating that these missing patterns are intrinsically associated with the underlying determinism of the process.\(^{67}\)

In another application with the `missing_patterns` function, we replicate a result of Amigó et al.\(^{67}\) (Fig. 4 in their work) to further show that the number of missing patterns is a good indicator of determinism in time series.\(^{67}\) By following the original work, we generate time series from the logistic map at fully developed chaos (6000 iterations) and add to them uniformly distributed noise in the interval \([-\xi, \xi]\), where \(\xi\) is the noise amplitude. Next, we estimate the average number of missing patterns (over ten time series replicas) for each noise level \(\xi \in \{0, 0.001, 0.002, \ldots, 0.5\}\) and embedding dimensions \(d_1 = 5\) and \(d_2 = 6\) (with \(\tau_1 = 1\)). Figure 5(b) shows the number of missing ordinal patterns as a function of noise amplitude \(\xi\) for both embedding dimensions. We observe that the number of missing patterns related to these deterministic series contaminated with noise approaches zero as noise amplitude grows. However, significantly higher noise levels are necessary to remove all signs of determinism expressed by the lack of permutation patterns when \(d = 6\).\(^{67}\)

V. TSALLIS AND RÉNYI ENTROPY-BASED QUANTIFIERS OF THE ORDINAL DISTRIBUTION

In addition to Shannon’s entropy and the statistical complexity, researchers have proposed to use other quantifiers of the ordinal probability distribution.\(^{15-19, 22}\) As we have explicitly verified for the statistical complexity, these different quantifiers are supposed to extract additional information from a time series’ dynamics that is not captured by permutation entropy and statistical complexity. In this context, a productive approach is to consider parametric generalizations of Shannon’s entropy, such as those proposed by Tsallis\(^{22}\) and Rényi.\(^{19}\) The work of Zunino et al.\(^{18}\) was the first to consider the Tsallis entropy in place of Shannon’s entropy to define the Tsallis permutation entropy as

\[
S_\beta(P) = \frac{1}{\beta - 1} \sum_{i=1}^{n_\pi} (\rho_i(\Pi_i) - \rho_i(\Pi))^\beta, \quad (11)
\]

where \(\beta\) is a real parameter \((\beta \rightarrow 1)\) recovers the usual Shannon entropy and so the permutation entropy. Tsallis’ entropy is also maximized by the uniform distribution such that \(S_{\beta}^{\text{max}} = \frac{1}{1 - (n_\pi)^{\beta}}\).

Thus, the normalized Tsallis permutation entropy is

\[
H_\beta(P) = (\beta - 1) \frac{S_\beta(P)}{1 - (n_\pi)^{\beta}}. \quad (12)
\]
Similarly, Liang et al.\textsuperscript{23} have proposed the Rényi permutation entropy
\begin{equation}
S_\alpha(P) = \frac{1}{1-\alpha} \ln \left( \sum_{\pi=1}^{n_r} \rho(\pi)^\alpha \right),
\end{equation}
where $\alpha > 0$ is a real parameter. Rényi’s entropy converges to Shannon’s entropy when $\alpha \to 1$ and is maximized by the uniform distribution ($S_\infty = \ln n_r$, as the usual Shannon entropy). Thus, the normalized Rényi permutation entropy is
\begin{equation}
H_\alpha(P) = \frac{S_\alpha(P)}{\ln n_r}.
\end{equation}

In both cases, the generalized entropic form is monoparametric and has a term where the ordinal probabilities appear raised to the power of the entropic parameter [that is, $\rho(\pi)^\beta$ and $\rho(\pi)^\gamma$]. These parameters assign different weights to the underlying ordinal probabilities, allowing us to access different dynamical scales and produce a family of quantifiers for the ordinal distribution. In \texttt{ordpy}, the \texttt{tsallis_entropy} and \texttt{renyi_entropy} functions implement these two quantifiers as in

```python
>>> from ordpy import tsallis_entropy,
... renyi_entropy
>>> tsallis_entropy([4, 7, 9, 10, 6, 11, 3],
... q=[1, 2], dx=2) #Here q plays the ...
... role of beta.
array([0.91829583, 0.88888889])
>>> renyi_entropy([4, 7, 9, 10, 6, 11, 3],
... alpha=[1, 2], dx=2)
array([0.91829583, 0.84799691])
```

In a similar direction, there are also the developments of complexity–entropy curves proposed by Ribeiro et al.\textsuperscript{23} and Jauregui et al.\textsuperscript{24} These works have further extended the complexity–entropy plane concept by considering the Tsallis and Rényi entropies combined with proper generalizations of statistical complexity.\textsuperscript{24} Thus, instead of having a single point in the complexity–entropy plane for a given time series, Ribeiro et al.\textsuperscript{23} and Jauregui et al.\textsuperscript{24} have created parametric curves by varying the entropic parameter ($\beta$ or $\alpha$) and simultaneously calculating the generalized entropy and the generalized statistical complexity.

To define the Tsallis complexity–entropy curves,\textsuperscript{23} we first extend the statistical complexity [Eq. (9)] using the Tsallis entropy; that is,
\begin{equation}
C_\beta(P) = \frac{D_\beta(P, U)H_\beta(P)}{P^{\max}},
\end{equation}
where
\begin{equation}
D_\beta(P, U) = \frac{1}{2} K_\beta \left( \left( P^{1/\beta} + U^{1/\beta} \right)^{\beta} \right) + \frac{1}{2} K_\beta \left( \left( \frac{P^{1/\beta}}{U^{1/\beta}} \right)^{\beta} \right)
\end{equation}
is the Jensen–Tsallis divergence\textsuperscript{25} written in terms of the corresponding Kullback–Leibler divergence\textsuperscript{26}
\begin{equation}
K_\beta(V|R) = \frac{1}{\beta - 1} \sum_{i} v_i^{1-\beta} \ln \left( \frac{v_i^{1-\beta}}{r_i^{1-\beta}} \right).
\end{equation}
where $V = \{v_i\}_{i=1,...,n_v}$ and $R = \{r_i\}_{i=1,...,n_r}$ are two arbitrary distributions. In Eq. (15),
\[
D^\text{max}_\beta = \frac{2^{-\beta}n_v - (1 + n_r)^{1-\beta} - n_v(1 + 1/n_v)^{1-\beta} - n_r + 1}{2^{1-\beta}n_v(1 - \beta)}
\]
is a normalization constant representing the maximum possible value of $D_\beta(P, U)$ that occurs for $P = \{h_i\}_{i=1,...,n_v}$ (as in the usual Jensen–Shannon divergence). By following Ribeiro et al.,\(^{23}\) we construct a parametric representation of the ordered pairs $[H_\alpha(P), C_\beta(P)]$ for $\beta > 0$, obtaining the Tsallis complexity–entropy curves.

Similarly, to define the Rényi complexity–entropy curves,\(^{26}\) we generalize the statistical complexity in Rényi’s formalism as
\[
C_\beta(P, U) = \frac{D_\beta(P, U)H_\alpha(P)}{D^\text{max}_\beta},
\]
where
\[
D_\beta(P, U) = \frac{1}{2}K_\alpha\left(\frac{P + U}{2}\right) + \frac{1}{2}K_\alpha\left(\frac{U + P}{2}\right)
\]
is the Jensen–Rényi divergence\(^{24}\) written in terms of
\[
K_\alpha(V|R) = \frac{1}{\alpha - 1} \ln \left( \sum_{i=1}^n v_i^\alpha r_i^{1-\alpha} \right),
\]
the corresponding Kullback–Leibler divergence for Rényi’s entropy.\(^{24}\) The normalization constant
\[
D^\text{max}_\alpha = \frac{1}{2(\alpha - 1)} \ln \left[ \left( \frac{n_v + 1}{n_v} \right)^{1-\alpha} + \frac{n_v - 1}{n_v} \left( \frac{n_v + 1}{4n_v} \right)^{1-\alpha} \right]
\]
corresponds to the maximum possible value of $D_\alpha(P, U)$ occurring for $P = \{h_i\}_{i=1,...,n_v}$ (as in the usual Jensen–Shannon divergence). Again, we can construct a parametric representation of the ordered pairs $[H_\alpha(P), C_\beta(P)]$ for $\alpha > 0$, obtaining the Rényi complexity–entropy curves proposed by Jauregui et al.\(^{22}\)

In ordpay, the functions `tsallis_complexity_entropy` and `renyi_complexity_entropy` implement the Tsallis and Rényi complexity–entropy curves as shown in the following code snippet:

```python
>>> from ordpay import ...
... tsallis_complexity_entropy,
... renyi_complexity_entropy
>>> tsallis_complexity_entropy{
... [4, 7, 9, 10, 6, 11, 3], ...
... dx=2, q=(1, 2)) #Here q plays the ...
... role of beta.
array([[0.91829583, 0.06112817],
... [0.88868899, 0.07619048]])
>>> renyi_complexity_entropy{
... [4, 7, 9, 10, 6, 11, 3], ...
... dx=2, alpha=[1, 2])
array([[0.91829583, 0.06112817],
... [0.84796961, 0.08303895]])
```

To better illustrate the use of these `ordpay`’s functions, we replicate some numerical experiments involving the logistic map and random walks presented in the original works of Ribeiro et al.\(^{23}\) (Figs. 1 and 6 in that work) and Jauregui et al.\(^{22}\) (Figs. 1 and 3 in that work). We start by generating time series from the logistic map at fully developed chaos ($r = 4$, random initial condition) and a Gaussian random walk. For the logistic map series, we discard the first 10\(^4\) iterations to avoid transient effects and iterate other 10\(^4\) steps. The random walk series also has 10\(^4\) observations. By using these time series, we generate their corresponding Tsallis complexity–entropy curves for $d_x = 3$ and $\tau_x = 1$ by sampling 10\(^3\) log-spaced values of the entropic parameter $\beta$ between 0.01 and 100 for the logistic map and between 0.001 and 100 for the random walk.

Figures 6(a) and 6(b) show the empirical complexity–entropy curves in comparison with their exact shape (dashed lines). These theoretical curves can be determined for these time series because the ordinal distributions of the logistic map ($P_{\text{logistic}} = \{1/3, 1/15, 2/15, 3/15, 4/15, 0\}$) and random walks ($P_{\text{walk}} = \{1/4, 1/8, 1/8, 1/8, 1/8, 1/4\}$) are exactly known for $d_x = 3$.\(^{23}\) We observe that theoretical and empirical results are in excellent agreement. As discussed by Ribeiro et al.\(^{23}\) random series tend to form closed complexity–entropy curves [Fig. 6(a)], while chaotic time series are usually represented by open complexity–entropy curves [Fig. 6(b)]. These features emerge as a direct consequence of the existence or not of missing ordinal patterns captured by the limiting behavior of $H_\alpha$ as $\beta \to 0$ and $\beta \to \infty$.\(^{22}\)

By following a similar approach, we also estimate the Rényi complexity–entropy curves for the two previous time series for $d_x = 4$ and $\tau_x = 1$. Figures 6(c) and 6(d) show these Rényi complexity–entropy curves. Differently from the Tsallis case, Rényi complexity–entropy curves are always open,\(^{23}\) and the usage of these curves for distinguishing chaotic from stochastic series relies on a more subtle characteristic. Indeed, Jauregui et al.\(^{22}\) have found that the initial curvature of Rényi complexity–entropy curves ($dC_\beta/dH_\alpha$ for small $\alpha$) can be used as an indicator of determinism in time series. Specifically, they found that positive curvatures are associated with time series of stochastic nature, while negative ones are related to chaotic phenomena. This pattern also occurs in the results of Figs. 6(c) and 6(d).

VI. ORDINAL NETWORKS

Among the more recent developments related to the Bandt–Pompe framework, we have the so-called ordinal networks. First proposed by Small\(^{37}\) for investigating nonlinear dynamical systems and later generalized with his collaboration in a series of works,\(^{38}\) ordinal networks belong to a more general class of methods designed to map time series into networks, collectively known as time series networks.\(^{38}\) Beyond counting ordinal patterns, this approach considers first-order transitions among ordinal symbols within a symbolic sequence. In this network representation, the different ordinal patterns occurring in a data set are mapped into nodes of a complex network. The edges between nodes indicate that the associated permutation symbols are adjacent to each other in a symbolic sequence. Furthermore, edges can be directed according to the temporal succession of ordinal symbols and weighted by the relative frequencies in which the corresponding successions occur in a symbolic sequence.\(^{38}\)
FIG. 6. Tsallis and Rényi complexity–entropy curves. Tsallis complexity–entropy curves for time series obtained from (a) the logistic map at fully developed chaos and (b) a Gaussian random walk, both with embedding parameters $d_x = 3$ and $\tau_x = 1$. The solid lines represent the empirical results, and the dashed lines indicate the exact form of these complexity–entropy curves. Panels (c) and (d) show the Rényi complexity–entropy curves obtained from the same two time series with embedding parameters $d_x = 4$ and $\tau_x = 1$. In all panels, star markers indicate the beginning of the curves ($\beta \approx 0$ or $\alpha \approx 0$), while circle markers indicate the end of the curves (largest values of $\beta$ and $\alpha$). Data and code necessary to reproduce these results are available in a Jupyter notebook at ordpy’s webpage (http://github.com/arthurpessa/ordpy).

After applying the Bandt-Pompe method with embedding parameters $d_x$ and $\tau_x$ to a time series $\{x_t\}_{t=1}^{N_x}$ and obtaining the symbolic sequence $\{\pi_p\}_{p=1}^{N_x}$, we can define the elements of the weighted adjacency matrix of the corresponding ordinal network as:

$$\rho_{ij} = \frac{\text{total of transitions } \Pi_i \rightarrow \Pi_j \text{ in } \{\pi_p\}_{p=1}^{N_x}}{n_i - 1}, \quad (21)$$

where $i, j = 1, 2, \ldots, n_x$ (with $n_x = d_x!$), $\Pi_i$ and $\Pi_j$ represent all possible ordinal patterns, and the denominator $n_i - 1$ is the total number of ordinal transitions. In ordpy, the `ordinal_network` function returns the nodes, edges, and edge weights of an ordinal network mapped from a time series as in:

```python
>>> from ordpy import ordinal_network
>>> ordinal_network([4,7,9,10,6,11,8,3,7],
                   dx=2, normalized=False)
(array(['0|1', '1|0'], dtype='<U3'),
 array([['0|1', '0|1'],
        ['0|1', '1|0'],
        ['1|0', '0|1'],
        ['1|0', '1|0']], dtype='<U3'),
 array([2, 2, 2, 1]))
```
It is worth noting that the original algorithm of Small\(^\text{22}\) for mapping time series into ordinal networks uses a different approach for creating the symbolic sequence. Instead of defining overlapping partitions [Eq. (1)], Small\(^\text{22}\) evaluates the ordinal patterns in non-overlapping partitions of size \(d_t\) (the embedding delay is also not present in his original formulation). Furthermore, edges are undirected and unweighted in this initial formulation. This implementation is not as popular as the one directly following the Bandt–Pompe symbolization method\(^\text{40}–\text{41}\) but is also available in ordpy through the overlapping parameter in the \texttt{ordinal_network} function, as shown in

```python
>>> from ordpy import ordinal_network
>>> ordinal_network([(4,7,9,10,6,11,8,3,7)], ... dx=2, dy=2, normalized=False, ... overlapping=False)
(array([['0|1', '1|0'], ['0|1', '1|0']], dtype='<U3'),
dtype= '<U3'), array([2, 1]))
```

Ordinal networks have also been recently generalized by Pessa and Ribeiro\(^{\text{37}}\) to account for two-dimensional data sets such as images. In this case, we apply the two-dimensional version of Bandt and Pompe\('s symbolization approach\(^{\text{30}, \text{32}}\) [see Eqs. (5)–(7)] to a data array \(y_i^{m_{x},m_{y}}\) for given embedding dimensions \((d_x\) and \(d_y)\) and embedding delays \((\tau_x\) and \(\tau_y)\), obtaining the corresponding two-dimensional ordinal sequence \(\pi_i^{p_1,...,p_n}\). Similarly to the one-dimensional case, each permutation symbol \(\Pi_{i}\) \([i=1,...,n_x]\) with \(n_x=(d_x,d_y)!\) is associated with a node in the ordinal network, and directed edges connect permutation symbols that are vertically \((\pi_i^{p_1,...,p_n} \rightarrow \pi_{i+1}^{p_1,...,p_n})\) for \(q=1,...,n_y-1\) or horizontally \((\pi_i^{p_1,...,p_n} \rightarrow \pi_{i}^{p_1,...,p_{n-1}})\) for \(p=1,...,n_x-1\) adjacents in the symbolic sequence. The directed link between a pair of permutation symbols \((\Pi_i\) and \(\Pi_{i+1})\) is weighted by the total number of occurrences of this particular transition in the symbolic sequence. Thus, the weighted adjacency matrix representing the ordinal network mapped from two-dimensional data is\(^{\text{41}}\)

\[
\rho_{ij} = \frac{\text{total of transitions } \Pi_i \rightarrow \Pi_j \text{ in } \pi_i^{p_1,...,p_n}}{2n_x n_y - n_x n_y}, \tag{22}
\]

where \(i,j=1,...,n_x\) [with \(n_x=(d_x,d_y)!\)] and the denominator represents the total number of horizontal and vertical transitions. The \texttt{ordinal_network} function also handles two-dimensional data as in

```python
>>> from ordpy import ordinal_network
>>> ordinal_network([[1,2,1], [8,3,4], [6,7,5]], ... dx=2, dy=2, normalized=False)
(array([['0|1|3|2', '1|0|2|3', '1|2|3|0'],['0|1|3|2', '1|2|3|0'], ...
array([[1|2|3|0], ['0|1|3|2']], dtype='<U7'),
dtype= '<U7'),
array([1, 1, 1]))
```

Pessa and Ribeiro\(^{\text{37}}\) have also proposed to create ordinal networks by considering only horizontal (horizontal ordinal networks) or only vertical (vertical ordinal networks) transitions among the permutation symbols. They have shown that comparing properties of these two networks is useful for exploring visual symmetries in images. In ordpy, this possibility is available through the \texttt{connections} parameter in the \texttt{ordinal_network} function as in

```python
>>> from ordpy import ordinal_network
>>> ordinal_network([[1,2,1], [8,3,4], [6,7,5]], ... dx=2, dy=2, normalized=False, ... connections='horizontal')
(array([['0|1|3|2', '1|0|2|3', '1|2|3|0'],['0|1|3|2', ...
array([[1|2|3|0], ['0|1|3|2']], dtype='<U7'),
dtype= '<U7'),
array([1, 1, 1]))
```

An intriguing feature of ordinal networks is the existence of intrinsic connectivity constraints\(^{\text{38}, \text{41}}\) inherited from Bandt and Pompe\('s symbolization method. These constraints are directly related to the fact that adjacent partitions share elements such that ordering relations in one partition are partially carried out to neighboring partitions. For one-dimensional data, these restrictions imply that all nodes in an ordinal network have in-degree and out-degree limited to numbers \(0\) and \(d_t\); consequently, the maximum number of edges is \(d_t \times (d_t)^{-1}\). Ordinal networks mapped from one-dimensional data can only have self-loops in nodes associated with solely ascending or solely descending ordinal patterns.\(^{\text{41}}\)

The horizontal and vertical transitions related to networks mapped from two-dimensional data impose similar but trickier connectivity constraints.\(^{\text{41}}\) In this case, the maximum number of outgoing connections emerging from horizontal and vertical transitions is \(C(d_x,d_y) \times d_t\) and \(C(d_x,d_x) \times d_t\), respectively.\(^{\text{41}}\) The sets of horizontal and vertical transitions are not disjoint, and their union defines all possible outgoing edges. Finding a general expression for the latter set operation is cumbersome because it depends on the ordinal pattern associated with the node under analysis. Thus, while limited, the maximum number of edges varies among the ordinal patterns and needs to be numerically obtained.\(^{\text{41}}\) Furthermore, differently from the one-dimensional case, ordinal networks mapped from two-dimensional data can display self-loops in several nodes.\(^{\text{41}}\)

A direct consequence of these intrinsic connectivity constraints is that ordinal networks mapped from completely random arrays (in one or two dimensions) are not random graphs.\(^{\text{40}, \text{41}}\) Even more counter-intuitive is the existence of different edge weights in random ordinal networks, albeit all permutations are equiprobable in random arrays.\(^{\text{40}, \text{41}}\) This non-trivial property results from the fact that, among all possible amplitude relations involved in an ordinal transition between a fixed permutation and all its possible neighboring permutations, some permutations appear more than once. For one-dimensional data, random ordinal networks only have two different edge weights: \(1/(d_t + 1)\) and \(2/(d_t + 1)\) (the denominator represents the sum of weights).\(^{\text{41}}\) A rule of thumb for determining the edges with double weight is to pick all transitions in which the index number equal to \(d_t - 1\) in the first permutation.\(^{\text{41}}\) For instance, the edge weight between permutations \((3,2,1,0)\) and \((2,1,0,3)\) has
double weight. Ordinal networks mapped from two-dimensional random data have more than two different edge weights, and there is no simple rule (at least up to now) for obtaining these weights. However, these values can be numerically calculated by explicitly considering each possible ordinal pattern.

In ordpy, the `random_ordinal_network` function generates the exact form of ordinal networks expected from the mapping of one- and two-dimensional random data with arbitrary embedding dimensions \((d_x, d_y)\). The following code illustrates the usage of `random_ordinal_network`:

```python
>>> from ordpy import random_ordinal_network
>>> random_ordinal_network(dx=2)
(array([['0|1', '0|1'], ['0|1', '1|0'], ['1|0', '0|1'], ['1|0', '1|0']], dtype='<U3'), array([0.0, 0.0, 0.0, 0.0]), array([0.125, 0.125, 0.125, 0.125]))
```

The three returned arrays represent nodes, edges, and edge weights of the random ordinal network, respectively. It is worth noticing that these connectivity constraints disappear when considering non-overlapping data partitions as in the initial proposal of Small. In this case, ordinal networks mapped from large enough random data sets are represented by complete graphs with self-loops and all-equal edge weights. The `random_ordinal_network` function returns these graphs by changing its argument `overlapping` to `False`:

```python
>>> from ordpy import random_ordinal_network
>>> random_ordinal_network(dx=2, overlapping=False)
(array([['0|1', '1|0'], ['0|1', '0|1'], ['1|0', '0|1'], ['1|0', '1|0']], dtype='<U3'), array([0.25, 0.25, 0.25, 0.25]), array([0.25, 0.25, 0.25, 0.25]))
```

Similarly, embedding delays larger than one modify how elements are shared among partitions and impose connectivity constraints to high-order transitions. The `random_ordinal_network` function is thus restricted to the case \(\tau_x = \tau_y = 1\) when considering overlapping partitions.

The primary purpose of mapping time series or images into ordinal networks is to use network measures to characterize data sets. In addition to the many network statistics derived from network science, the inherent probabilistic nature of nodes and edges in ordinal networks has motivated two entropy-related measures. The first one is a local measure defined at the node level known as the local node entropy:

\[
s_i = -\sum_{j\in\mathcal{O}_i} \rho_{ij} \log \rho_{ij},
\]

where the index \(i\) refers to a node related to a given permutation \(\Pi_i\), \(\rho_{ij}\) represents the renormalized probability of transitioning from node \(i\) to node \(j\) (permutations \(\Pi_i\) and \(\Pi_j\)), and \(\mathcal{O}_i\) is the outgoing neighborhood of node \(i\) (set of all edges leaving node \(i\)). This quantity measures the determinism of ordinal transitions at the node level such that \(s_i\) is maximum when all edges leaving \(i\) have the same weight, while \(s_i = 0\) if there is only one edge leaving node \(i\).

Using the local node entropy, we can further define the global node entropy:

\[
S_{GN} = \sum_{i=1}^{n} \rho_i s_i
\]

where \(\rho_i\) is the probability of finding the permutation \(\Pi_i\) [Eqs. (2) and (8)]. Thus, the value \(S_{GN}\) represents a weighted average of the local determinism over all nodes of an ordinal network (see also Unakafov and Keller\(^b\) for the definition of conditional entropy of ordinal patterns). In image classification tasks,\(^c\) global node entropy has proven to outperform different image quantifiers derived from gray-level co-occurrence matrices (GLCMs),\(^d\) a traditional technique for texture analysis.

Contrarily to permutation entropy\(^e\) and because of the intrinsic connectivity constraints of ordinal networks, the global node entropy is not maximized by random data.\(^f\) For one-dimensional data, the global node entropy calculated from a random ordinal network is

\[
S_{GN}^{\text{random}} = \log(d_x + 1) - (\log 4)/(d_x + 1).
\]

While there is no equivalent expression for two-dimensional data, it is possible to numerically calculate \(S_{GN}^{\text{random}}\) using random ordinal networks numerically generated.\(^g\) In both cases, the global node entropy can be normalized by the value of \(S_{GN}^{\text{random}}\); that is, \(H_{GN} = S_{GN}/S_{GN}^{\text{random}}\).

In ordpy, the `global_node_entropy` function evaluates \(S_{GN}\) directly from data arrays or using an ordinal network as returned by `ordinal_network()`. The following code shows simple usages of `global_node_entropy`:

```python
>>> from ordpy import global_node_entropy
>>> global_node_entropy(1,2,3,4,5,6,7,8,9, dx=2)
0.0
>>> global_node_entropy(1,2,3,4,5,6,7,8,9, dx=2)
0.0
>>> global_node_entropy(np.random.uniform(size=100000), dx=3)
1.4988332319747597
>>> global_node_entropy(1,2,3,4,5,6,7,8,9, dx=3)
1.5
```

VII. APPLICATIONS OF ORDINAL NETWORKS WITH ordpy

To better illustrate the use of ordpy in the context of ordinal networks, we review and replicate some literature results. Before starting, we remark that ordpy does not have functions for network analysis or graph visualization. The `ordinal_network`
function generates output data (nodes, edges, and weight lists) that can feed graph libraries such as graph_tool, networkx, and igraph. Here, we have used networkx and igraph.

We start by partially reproducing Small’s pioneering work in which “ordinal partition networks” first appeared (see Fig. 3 in that work). By following Small, we numerically solve the differential equations of the Rössler system (with parameters $a = 0.3, b = 2,$ and $c = 4$), see Appendix B for definitions) and sample the $x$-coordinate to obtain a time series with $10^5$ observations. Figure 7(a) illustrates the periodic behavior of this time series. We then create the ordinal network from this data set with embedding parameters $d_z = 16$ and $\tau_x = 1$. It is worth remembering that Small’s original algorithm uses non-overlapping partitions, and the edges of the resulting ordinal network are undirected and unweighted. The parameter overlapping in ordinal_network should be equal to False to properly use Small’s original algorithm. Figure 7(b) shows a visualization of this ordinal network, where the circular structure alludes to the periodicity of the original time series.

In another simple example with ordinal networks, we partially replicate Pessa and Ribeiro’s results on fractional Brownian motion (see Fig. 6 in their work). To do so, we generate a time series from this stochastic process with Hurst exponent $h = 0.8$ (see Appendix C for definitions) and $2^{16}$ observations, as illustrated in Fig. 7(c). Next, we map this time series into an ordinal network with embedding parameters $d_z = 3$ and $\tau_x = 1$ (this time using overlapping partitions as in the usual Bandt–Pompe approach). Figure 7(d) shows a visualization of the resulting ordinal network, where the persistent behavior imposed by the Hurst exponent $h = 0.8$ is captured by the quite intense autoloops associated with the ordinal patterns (0, 1, 2) and (2, 1, 0) (that is, the upward and downward trends of this time series). Pessa and Ribeiro have also shown that local properties of ordinal networks (for instance, average weighted shortest path) are quite effective for estimating the Hurst exponent of time series, having performance superior to widely used approaches such as detrended fluctuation analysis (DFA).

We also consider ordinal networks mapped from two-dimensional data. We map a periodic ornament previously explored in Ref. 41 (see Fig. 2 in that reference). Figure 7(e) shows the ornament of size $250 \times 250$ (see Appendix C for more details), while Fig. 7(f) presents a visualization of the corresponding ordinal network with embedding parameters $d_z = d_y = 2$ and $\tau_x = \tau_y = 1$. We have made the edge thickness proportional to the edge weight [Eq. (22)] to highlight that a few edges concentrate most of the

**Fig. 7.** Ordinal networks mapped from one- and two-dimensional data. (a) Time series obtained from the $x$-coordinate of the Rössler system (with parameters $a = 0.3, b = 2,$ and $c = 4$). Here, we show only the latest 500 of all $10^5$ observations. This time series exhibits a periodic behavior after a short initial transient. (b) Visualization of the ordinal network mapped from the $x$-coordinate of the Rössler system with embedding parameters $d_z = 16$ and $\tau_x = 1$. This ordinal network uses Small’s original algorithm with non-overlapping data partitions and undirected and unweighted edges. (c) Time series obtained from a realization of the fractional Brownian motion with $h = 0.8$ (only the latest 500 of all $2^{16}$ observations are shown). (d) Ordinal network representation of the fractional Brownian motion time series with $d_z = 3$ and $\tau_x = 1$. Here, we have used overlapping data partitions and made the edge thickness proportional to the edge weight. (e) Example of a periodic ornament with size $250 \times 250$. (f) Ordinal network representation of the previous image with embedding parameters $d_z = d_y = 2$ and $\tau_x = \tau_y = 1$. In this visualization, the edge thickness is made proportional to the edge weight. Data and code necessary to reproduce these results are available in a Jupyter notebook at ordpy’s webpage (http://github.com/arthurpessa/ordpy).
transition probability of the network. Furthermore, we observe that this network has 12 nodes and 72 edges, that is, only a small fraction of all possible nodes (24) and edges (416) of ordinal networks with $d_x = d_y = 2$ and $\tau_x = \tau_y = 1$.

In addition to the previous more qualitative examples, we have also replicated some results related to the global node entropy of ordinal networks. For time series, we follow Pessa and Ribeiro\textsuperscript{40} (see Fig. 5 in their work) and generate a periodic sawtooth-like signal $x_t$ with $10^5$ observations and add to it uniform white noise in the interval $[-\xi, \xi]$, where $\xi$ represents the noise amplitude. We generate these noisy sawtooth-like time series for each $\xi \in \{0, 0.05, 0.1, \ldots, 2\}$ and determine the average values of the normalized permutation entropy ($H$) and the normalized global node entropy ($H_{GN}$) over ten time series replicas with $d_x = 4$ and $\tau_x = 1$.

Figure 8(b) shows the average values of $H$ and $H_{GN}$ as a function of the noise amplitude $\xi$. We note that both measures approach one with the increase of the noise amplitude. However, permutation entropy saturates for $\xi \approx 1$, while global node entropy requires significantly higher values of $\xi$. This result indicates that global node entropy is more robust to noise addition and has a higher discrimination power than permutation entropy.\textsuperscript{40}

![Figure 8](https://example.com/fig8.png)

**FIG. 8.** Global node entropy of one- and two-dimensional data. (a) The initial data points of a sawtooth-like time series defined as $x_t = \{0, 1/3, 1/6, 1, \ldots\}$. (b) Normalized permutation entropy ($H$) and normalized global node entropy ($H_{GN}$) as a function of the amplitude ($\xi$) of the uniform white noise added to the periodic sawtooth-like signals. The different curves represent average values of $H$ and $H_{GN}$ over ten realizations for $\xi \in \{0, 0.05, 0.1, \ldots, 2\}$. (c) Eight examples (out of 112) of the normalized Brodatz textures. These are grayscale images (256 gray levels) with size $640 \times 640$.\textsuperscript{91} (d) Differences between the global node entropy evaluated from the horizontal and vertical ordinal networks ($S_{Horizontal}^{GN} - S_{Vertical}^{GN}$) mapped from each Brodatz texture. We highlight eight textures [the same as shown in panel (c)] with the largest differences. Data and code necessary to reproduce these results are available in a Jupyter notebook at ordpy’s webpage (http://github.com/arthurpessa/ordpy).
To demonstrate the use of global_node_entropy with two-dimensional data, we calculate the global node entropy for a set of 112 8-bit images of natural textures known as the normalized Brodatz textures. Figure 8(c) shows examples of these images. By following Pessa and Ribeiro, we calculate the global node entropy from the horizontal ($S_{GN}^{Horizontal}$) and vertical ($S_{GN}^{Vertical}$) ordinal networks mapped from the Brodatz textures with $d_x = 2$ and $\tau = 1$. Figure 8(d) depicts the difference between these two entropy values (that is, $S_{GN}^{Horizontal} - S_{GN}^{Vertical}$) for each Brodatz texture. We have also highlighted eight textures with extreme values for this difference. Most of these images are characterized by stripes or line segments predominantly oriented in the vertical or horizontal directions, which, in turn, suggests that properties of vertical and horizontal ordinal networks can detect simple image symmetries.

In a final application with ordinal networks, we explore the concept of missing links or missing transitions among ordinal patterns. Similarly to the missing ordinal patterns described by Amigó et al., ordinal networks can display true and false forbidden transitions among ordinal patterns. In this case, true missing links are related to the intrinsic dynamics of the process under analysis while false missing links are associated with the finite size of empirical data sets. Because we know the exact form of random networks (here, these networks represent all possible connections), we can readily find all missing links of an empirical ordinal network. In ordpy, the missing_links function evaluates all missing ordinal transitions directly from a data set or the returned arrays of ordinal_network as in

```python
>>> from ordpy import missing_links
>>> missing_links([4,7,9,10,6,11,3], dx=2)
(array([['1|0', '1|0']], dtype='<U3'), 0.25)
```

To demonstrate the use of missing_links in a more engaging example, we replicate the results of Pessa and Ribeiro about missing links in ordinal networks mapped from Gaussian white noise time series (Fig. 4 in their work). We generate these time series with length $N_x$ varying logarithmically between 10 and $10^5$, and for each one, we estimate the average fraction of missing links over ten replicas for embedding dimensions $d_x \in \{3, 4, 5, 6\}$ and $\tau = 1$. Figure 9(a) shows these fractions of missing links as a function of the time series length. We observe that this quantity approaches zero as $N_x$ becomes sufficiently large. Furthermore, the smaller the embedding dimension, the faster the missing links vanish. This pattern is a fingerprint of false missing links. We have also carried out the same analysis with time series generated from logistic map iterations at fully developed chaos. Figure 9(b) shows the corresponding results.

Unlike white noise, the logistic map produces ordinal networks with missing links that persist even in considerably long time series. This behavior is typical of true missing links.

### VIII. CONCLUSIONS

We have introduced ordpy—an open-source Python module for data analysis that implements several ordinal methods associated with the Bandt–Pompe framework. Specifically, ordpy has functions implementing the following methods: permutation entropy, complexity–entropy plane, missing ordinal patterns, Tsallis and Rényi permutation entropies, complexity–entropy curves, ordinal

![Fraction of missing links vs. time series length](https://example.com/fig9.png)

**FIG. 9.** True and false missing links in ordinal networks. (a) Dependence of the fraction of missing links $f$ estimated from Gaussian white noise time series as a function of the time series length ($N_x$). (b) Dependence of the fraction of missing links $f$ estimated from fully chaotic logistic time series ($r = 4$) as a function of the time series length ($N_x$). In both panels, the different curves represent average values over ten realizations (for each series length) and embedding dimension $d_x \in \{3, 4, 5, 6\}$ with $\tau = 1$. We also use 194 values for $N_x$ logarithmically spaced in the interval $[10, 10^5]$ in both panels. Data and code necessary to reproduce these results are available in a Jupyter notebook at ordpy’s webpage (http://github.com/arthurpessa/ordpy).
networks, and missing ordinal transitions. All ordpy’s functions automatically deal with one-dimensional (time series) and two-dimensional (images) data. Furthermore, most of these functions are also ready for multiscale analysis via the embedding delay parameters. Along with the description of ordpy functionalities, we have also presented a literature review of several of the principal methods related to Bandt and Pompe’s framework. This review further includes a reproduction of several literature results with ordpy’s functions. Beyond the summarized description of ordpy’s functions presented here, we notice that a complete documentation is available at arthurpessa.github.io/ordpy. All data and code used in this work are also freely available at ordpy’s website (http://github.com/arthurpessa/ordpy).

We believe that ordpy will help to popularize ordinal methods even further, particularly in research fields with more limited tradition in scientific computing. In addition to a myriad of possible empirical applications, we also believe that ordpy can further promote the development of new methods related to Bandt and Pompe’s framework. We remark that some techniques available in ordpy have received little attention or have not even been formally proposed. These possible developments already implemented in ordpy include the use of complexity–entropy curves for two-dimensional data, multiscale complexity–entropy curves, ordinal networks with different embedding delays (particularly for two-dimensional data), analysis of missing patterns in two-dimensional data, and missing ordinal transitions. We also plan to implement more techniques based on Bandt and Pompe’s framework and include them in future versions of ordpy.

Finally, we hope our module helps making research methods more accessible and reproducible as well as other open-source software efforts such as the tisean (nonlinear time series analysis), pyunicorn (time series networks and recurrence analysis), and powerlaw (analysis of heavy-tailed distributions) packages.

ACKNOWLEDGMENTS

This research was supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq—Grant Nos. 407690/2018-2 and 303121/2018-1).

APPENDIX A: SELECTION OF EMBEDDING PARAMETERS

As we have commented in the main text, the embedding parameters \( (d_e, \tau_e) \) are important for several applications related to the Bandt-Pompe framework, and wrong choices can lead to misleading conclusions. At the same time, there is no unique fail-safe procedure for selecting optimal values for these parameters, and this choice often depends on the time-series nature and the research question under analysis. In the context of permutation entropy, Myers and Khasawneh\(^{11} \) suggest three main strategies: (i) follow experts’ suggestion, (ii) trial and error, and (iii) the use of nonlinear time series methods related to phase space reconstruction.

The first strategy consists of following good practices previously established in the literature, and good starting points are review articles on permutation entropy and related methods such as Refs. \(^{10–13} \). The work of Riedl et al.\(^{11} \) is particularly interesting for this strategy as the authors compile different choices of embedding parameters according to characteristics of time series and research field. Among other propositions, these authors suggest using \( \tau_e = 1 \) and the largest embedding dimension yielding a proper evaluation of the ordinal distribution when dealing with data more easily described by discrete models.

The second strategy refers to the computational origin of the Bandt-Pompe framework, and much in line with statistical learning methods,\(^{14,15} \) optimal parameter selection is often achieved by experimentation (trial and error), heuristics, and validation using null models. We believe that this strategy is fundamental in applications involving classification and regression tasks, where the optimal embedding parameters can be found by optimizing loss functions in cross-validation and train/test split strategies.\(^{16,17} \) For instance, Kulp et al.\(^{18} \) have suggested using ensemble of random series with the same length of the series under analysis and selecting the maximum embedding dimension for which the number of missing patterns is zero.

The third strategy for selecting the embedding parameters refers to using methods derived or related to nonlinear time series analysis.\(^{19–23} \) Common techniques such as looking for the first zero of the autocorrelation function or the first minimum of the mutual information might be especially interesting when choosing \( \tau_e \).\(^{24,25} \) It is worth remembering that the concept of embedding parameters in the Bandt-Pompe approach is intimately related to the idea of embedding and phase-space reconstruction in the context of dynamical systems.\(^{26–28} \) Indeed, investigations based on ordinal methods in the context of chaotic dynamics are an instrumental part for the development of the Bandt-Pompe framework.\(^{29–31} \) In this context, a simple and interesting conceptualization on how the embedding dimension relates to the underlying phase space is presented by Groth.\(^{32} \)

In addition to the previous three main strategies, there are also attempts devoted to developing automatic procedures for selecting embedding parameters in the context of permutation entropy.\(^{33–37} \) Among these, we highlight the interesting comparison between expert recommendations and automatic approaches presented by Myers and Khasawneh.\(^{38} \)

Considering the more recent developments related to mapping time series into ordinal networks, several works on this topic have devoted efforts to the optimal selection of embedding parameters.\(^{39–42,43} \) In this context, topological properties or network metrics have become major criteria for properly selecting embedding parameters capable of capturing dynamical features of time series. To illustrate this approach, we have partially reproduced the results of Sakellariou et al.\(^{44} \) (see Fig. 8 in their work) about ordinal networks mapped from periodic logistic series with period \( 2^k \) for \( k \in \{2, 3, 4, 5\} \). Figure 10(a) shows a representation of these networks mapped with \( d_e = 17 \) and \( \tau_e = 1 \). We observe that this value of \( d_e \) is large enough to map the periodic behavior of these time series into a regular ring-like network structure with the number of nodes precisely equal to the time series period. As discussed by Sakellariou et al.\(^{44} \), the embedding dimension needs to be larger than \( 2^{k-1} \) so that the network topology explicitly represents the period \( 2^k \) of the time series. Figure 10(b) illustrates what happens with ordinal networks mapped from a period 8 time series (\( k = 3 \)) for different
FIG. 10. Ordinal networks mapped from periodic logistic series. (a) Ordinal networks mapped from periodic logistic series \((N_x = 100)\) with parameters \(r = 3.5\) (period 4), \(r = 3.56\) (period 8), \(r = 3.566\) (period 16), and \(r = 3.5695\) (period 32). All four networks are obtained with embedding parameters \(d_x = 17\) and \(\tau_x = 1\). (b) Ordinal networks mapped from a logistic series of period 8 \((r = 3.56)\) with embedding dimensions \(d_x \in \{2, 3, 4, 5\}\) and \(\tau_x = 1\). We notice that \(d_x\) must be larger than 4 so that the number of nodes is equal to the time series period. We further remark that this network structure is not modified when considering values \(d_x\) larger than 4 \([\text{as shown by the second network of panel (a)}]\). Data and code necessary to reproduce these results are available in a Jupyter notebook at ordpy’s webpage \([\text{http://github.com/arthurpessa/ordpy}]\).

values of \(d_x\), confirming that the network topology only explicitly accounts for period 8 behavior for \(d_x > 4\). Similar problems involving non-optimal choices for \(d_x\) emerge when using ordinal networks to estimate dynamical quantities such as the topological entropy. \(^{112}\)

APPENDIX B: DEFINITIONS OF DYNAMICAL SYSTEMS

In this appendix, we present a brief definition of the dynamical systems used in this article.

1. The logistic map is defined by the following difference equation:\(^{112}\)

\[
x_{t+1} = rx_t(1 - x_t),
\]

where \(r\) is a parameter. We have used \(r = 4\) in most applications of this article unless specified otherwise.

2. The transient logistic map is defined as \(^{51}\)

\[
x_{t+1} = r(t)x_t(1 - x_t),
\]

where the parameter \(r(t)\) changes at each iteration. The results of Figs. 3(e) and 3(f) were obtained with \(r(t) = 0.35\) and by incrementing this parameter in steps of size \(10^{-4}\) up to \(r(t) = 4\).

3. The skew tent map is defined as \(^{114}\)

\[
\begin{align*}
x_{t+1} &= \frac{x_t}{\omega} & \text{for } x \in [0, \omega], \\
(1 - x_t)/(1 - \omega) & \text{for } x \in [\omega, 1],
\end{align*}
\]

where \(\omega\) is a parameter. In the complexity–entropy plane application shown in Fig. 4(a), we have used \(\omega = 0.1847\).

4. The Hénon map is defined as \(^{115}\)

\[
\begin{align*}
x_{t+1} &= 1 - ax_t^2 + y_t, \\
y_{t+1} &= bx_t,
\end{align*}
\]

where \(a\) and \(|b| < 1\) are parameters. This map can be thought of as a two-dimensional extension of the logistic map. \(^{112}\) We have used \(a = 1.4\) and \(b = 0.3\) for the results related to the complexity–entropy plane.

5. The Schuster map is defined as \(^{115}\)

\[
x_{t+1} = (x_t + x_{t-1}) \mod 1,
\]

where \(z\) is a parameter, and the modulo operation returns the fractional (decimal) part of a number. We have used \(z \in \{2, 2.5, 3\}\), as shown in Fig. 4(a).
6. The Rössler system is a continuous time dynamical system defined as\cite{rossler1976}
\[
\frac{dx}{dt} = -y - z, \\
\frac{dy}{dt} = x + ay, \\
\frac{dz}{dt} = b + z(x - c),
\]
where \(a, b, \) and \(c\) are parameters. We have numerically solved this differential equation system using the SciPy Python module\cite{scipy} with parameters \(a = 0.3, b = 2,\) and \(c = 4.\)

**APPENDIX C: DEFINITIONS OF STOCHASTIC PROCESSES**

In this appendix, we briefly describe the stochastic processes used in the article.

1. An Ising surface\cite{ising1925,ising1928} is a square lattice in which the height at each lattice site represents the accumulated sum of spin variables of particles in a Monte Carlo simulation.\cite{mcmc} If we assume \(\sigma_i \in \{-1, 1\}\) represents the spin variable at site \(i\), we can write the Hamiltonian of this system as
\[
\mathcal{H} = -\sum_{\langle ij \rangle} \sigma_i \sigma_j,
\]
where the summation is over all pairs of first neighbors in a square lattice. The height \(S_i\) at site \(i\) of the corresponding Ising surface is then defined as
\[
S_i = \sum_j \sigma_j(t),
\]
where \(\sigma_j(t)\) is the spin value in step \(t\) of the Monte Carlo simulation. For each surface, we define the reduced temperature \(T_r\) as the ratio between the temperature \(T\) and the critical temperature \(T_c\) of the Ising system \([T_c = 2/\ln(1 + \sqrt{2})]\). Finally, we have used periodic boundary conditions in our numerical experiments.

2. A fractional Brownian motion is a continuous, self-similar, and non-stationary stochastic process introduced by Mandelbrot and Van Ness.\cite{Mandelbrot1968} The Hurst exponent \(h \in (0, 1)\) controls the roughness observed in samples of this process such that the smaller the values of \(h\), the rougher the time series. The case \(h = 1/2\) corresponds to ordinary Brownian motion (integrated Gaussian white noise). To generate samples (time series) of this stochastic process, we have used the Hosking method.\cite{hosking1981}

3. A fractional Gaussian noise is a stationary stochastic process that represents the increments of fractional Brownian motion. For this Gaussian process, the Hurst parameter \(h \in (0, 1)\) controls the range of auto-correlation of the time series. For \(h > 1/2\), the process presents long-range persistent memory. For \(h < 1/2\), samples present anti-persistent behavior. We have Gaussian white noise if \(h = 1/2\). To generate samples of a fractional Gaussian noise, we have also used the Hosking method.\cite{hosking1981} More detailed information about simulations of fractional Gaussian noise and fractional Brownian motion can be found in Ref. 123. The C source code used in this work is publicly available in Ref. 124.

4. A 1/f noise or a Flicker noise\cite{shannon1948,newsam2005} is a class of stochastic processes presenting a power-law power spectral density;\cite{newsam2005,newsam2014} that is, \(P \sim 1/f^\alpha\). The case \(\alpha = 0\) corresponds to white noise, while \(\alpha = 2\) corresponds to brown noise (random walk or integrated white noise). We have generated Gaussian distributed 1/f\(^\alpha\) noise for \(k \in \{0, 0.25, 0.50, \ldots, 3.0\}\) with the algorithm proposed by Timmer and König\cite{timmer2012} as implemented in Ref. 128.

5. The periodic ornament used in this work can be generated by first defining two square arrays
\[
X_{ij} = \frac{2\pi(j - 1)}{(n - 1)} \quad \text{and} \quad Y_{ij} = \frac{2\pi(i - 1)}{(n - 1)}
\]
and next by calculating
\[
Z_{ij} = \sin \left( \frac{\omega}{2\pi} X_{ij} \cos \theta - \frac{\omega}{2\pi} Y_{ij} \sin \theta \right),
\]
where \(i, j = 1, \ldots, n\), with \(n\) being the ornament size, \(\theta\) defining the stripes angle, and \(\omega > 0\) the stripe frequency. The ornament shown in Fig. 7(c) is obtained by setting \(n = 250\), \(\omega = 9\), and \(\theta = 135^\circ\). Previous works\cite{zunino2015,zunino2017} have also considered shuffled versions of this periodic ornament, where a parameter controls the fraction of elements \(Z_{ij}\) that are randomly shuffled. A function implementing this geometric ornament is available in orQpy’s notebook.

**DATA AVAILABILITY**

All data and code necessary to reproduce the results and figures of this work are available in GitHub at http://github.com/arthurpessa/orQpy, Ref. 95.

**REFERENCES**

Chaos, 31, 063110 (2021); doi: 10.1063/5.0049091
Published under an exclusive license by AIP Publishing.