

Entropy and Lorenz Model

November 2021

1 Entropy

1.1 Theory svd

We can find online the documentation of the function svd-entropy [1].

$$S_{svd} = S(\mathbf{x}(\mathbf{t}), order, delay) \quad (1)$$

We denote with \mathbf{x} the dataset that we want to study.

$$\begin{aligned} \mathbf{x}(\mathbf{t}_0) &= \mathbf{x}_0, \\ \mathbf{x}(\mathbf{t}) &= (x_1, x_2, x_3, \dots, x_N) \end{aligned} \quad (2)$$

This function creates a matrix \mathbf{Y} with the dataset.

$$\mathbf{y}(i) = (x_i, x_{i+delay}, \dots, x_{i+(order-1)delay}) \quad (3)$$

$$Y = \begin{pmatrix} \mathbf{y}(1) \\ \mathbf{y}(2) \\ \vdots \\ \mathbf{y}(N - (ord - 1)del) \end{pmatrix}$$

If we take as parameters $order = 3$ and $delay = 1$ we obtain the following matrix:

$$Y = \begin{pmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ \dots & \dots & \dots \\ \dots & x_{N-1} & x_N \end{pmatrix}$$

The next step is to write this matrix using the SVD decomposition.

$$Y = U\Sigma V^* \quad (4)$$

¹https://raphaelvallat.com/entropy/build/html/generated/entropy.svd_entropy.html

Where U is an unitary matrix, V is also an unitary matrix, and Σ is a diagonal matrix. This is linked with PCA. Now we denotes the eigenvalues of the matrix Σ with σ_i . Now we can compute the average eigenvalues:

$$\bar{\sigma}_k = \frac{\sigma_k}{\sum_j^M \sigma_j} \quad (5)$$

Where M is the number of eigenvalues.

After this we can compute the SVD Entropy:

$$S_{svd} = - \sum_k^M \bar{\sigma}_k \log_2(\bar{\sigma}_k) \quad (6)$$

1.2 Shannon Entropy

We analyze the Shannon Entropy function [2].

```

1 def shannon_entropy(time_series):
2     """Return the Shannon Entropy of the sample data.
3     Args:
4         time_series: Vector or string of the sample data
5     Returns:
6         The Shannon Entropy as float value
7     """
8
9     # Check if string
10    if not isinstance(time_series, str):
11        time_series = list(time_series)
12
13    # Create a frequency data
14    data_set = list(set(time_series))
15    freq_list = []
16    for entry in data_set:
17        counter = 0.
18        for i in time_series:
19            if i == entry:
20                counter += 1
21        freq_list.append(float(counter) / len(time_series))
22
23    # Shannon entropy
24    ent = 0.0
25    for freq in freq_list:
26        ent += freq * np.log2(freq)
27    ent = -ent
28    return ent

```

$$S_{sh} = S(\mathbf{x}) \quad (7)$$

Here we find the definition of Shannon Entropy:

$$S_{sh} = - \sum_i P(x_i) \log_2(P(x_i)) \quad (8)$$

Where $P(x_i)$ is the frequency of x_i in the dataset \mathbf{x} .

²<https://github.com/nikdon/pyEntropy/blob/master/pyentrp/entropy.py>

1.3 Approximate Entropy

```
1 def app_entropy(x, order=2, metric='chebyshev'):
2
3     phi = _app_samp_entropy(x, order=order, metric=metric,
4                             approximate=True) # it defines the vector phi
5     return np.subtract(phi[0], phi[1])
6
7 def _app_samp_entropy(x, order, metric='chebyshev', approximate=
8     True):
9     """Utility function for 'app_entropy' and 'sample_entropy'.
10
11     #technical stuff
12
13     _all_metrics = KDTree.valid_metrics
14     if metric not in _all_metrics:
15         raise ValueError('The given metric (%s) is not valid. The
16         valid '
17
18         'metric names are: %s' % (metric,
19         _all_metrics))
20
21     phi = np.zeros(2) #vector phi
22     r = 0.2 * np.std(x, ddof=0) #radius to define the concept of
23     neighbours (0.2* standard deviation with 0 DF of the
24     timeseries)
25
26     # compute phi(order, r)
27     _emb_data1 = _embed(x, order, 1) #creates a matrix with the
28     timeseries Nx2
29     if approximate: # TRUE
30         emb_data1 = _emb_data1
31     else:
32         emb_data1 = _emb_data1[:-1]
33     count1 = KDTree(emb_data1, metric=metric).query_radius(
34         emb_data1, r,
35
36         count_only=True
37
38         ).astype
39         (np.float64)
40
41         #
42         compute phi(order + 1, r)
43     emb_data2 = _embed(x, order + 1, 1) # same matrix but Nx3
44     count2 = KDTree(emb_data2, metric=metric).query_radius(emb_data2, r
45
46     ,
47
48         count_only=
49         True
50
51         ).astype(np.
52         float64)
53         # count the number of neighbours according
54         to Chebyshev norm for
55
56         # each point and put this number in a
57         vector.
58
59     if approximate: #true
60         phi[0] = np.mean(np.log(count1 / emb_data1.shape[0]))
```

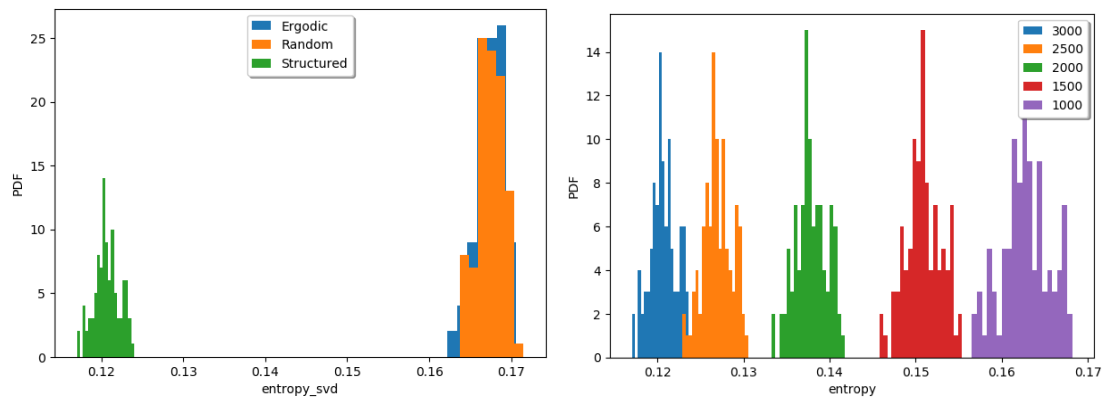
```

39     phi[1] = np.mean(np.log(count2 / emb_data2.shape[0]))
40     else:
41         phi[0] = np.mean((count1 - 1) / (emb_data1.shape[0] - 1))
42         phi[1] = np.mean((count2 - 1) / (emb_data2.shape[0] - 1))
43     return phi
44
45
46
47
48
49
50 def _embed(x, order=3, delay=1):
51
52     N = len(x)
53     if order * delay > N:
54         raise ValueError("Error: order * delay should be lower than
55                             x.size")
56     if delay < 1:
57         raise ValueError("Delay has to be at least 1.")
58     if order < 2:
59         raise ValueError("Order has to be at least 2.")
60     Y = np.zeros((order, N - (order - 1) * delay))
61     for i in range(order):
62         Y[i] = x[(i * delay):(i * delay + Y.shape[1])]
63     return Y.T
64
65 @jit('UniTuple(float64, 2)(float64[:], float64[:])', nopython=True)

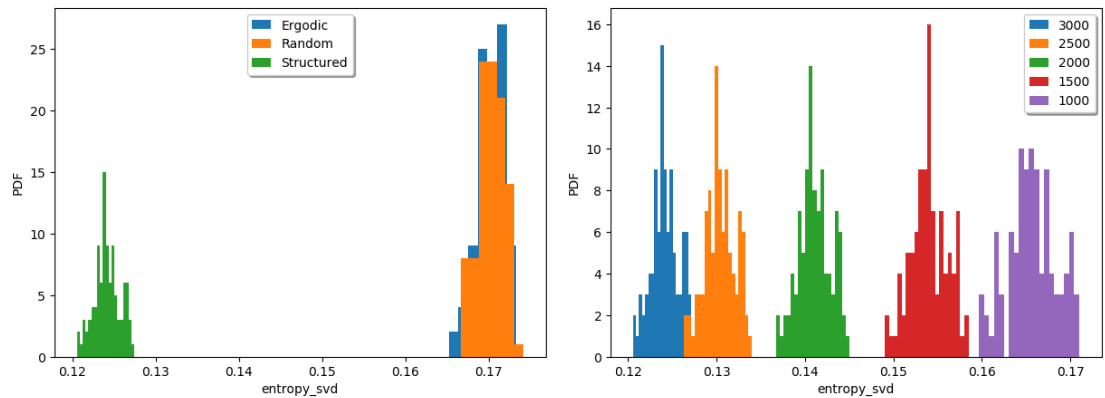
```

1.4 Plots Entropy

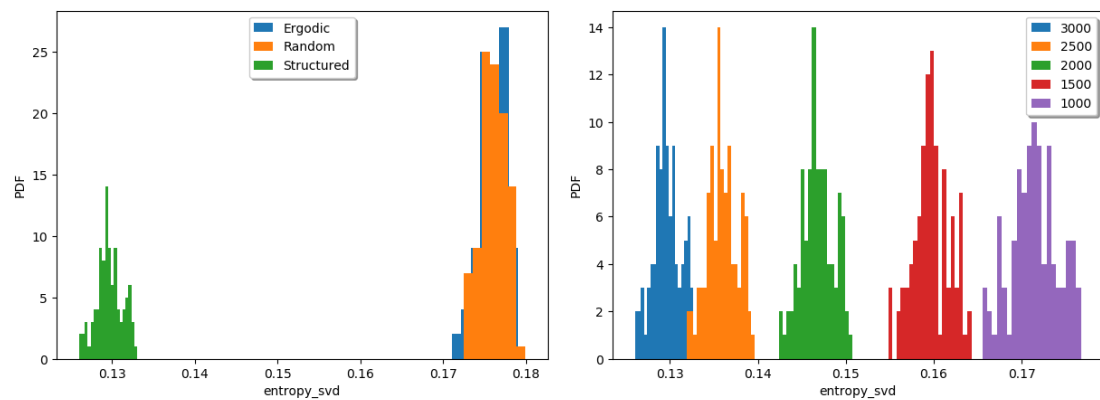
1.4.1 SVD with order 3



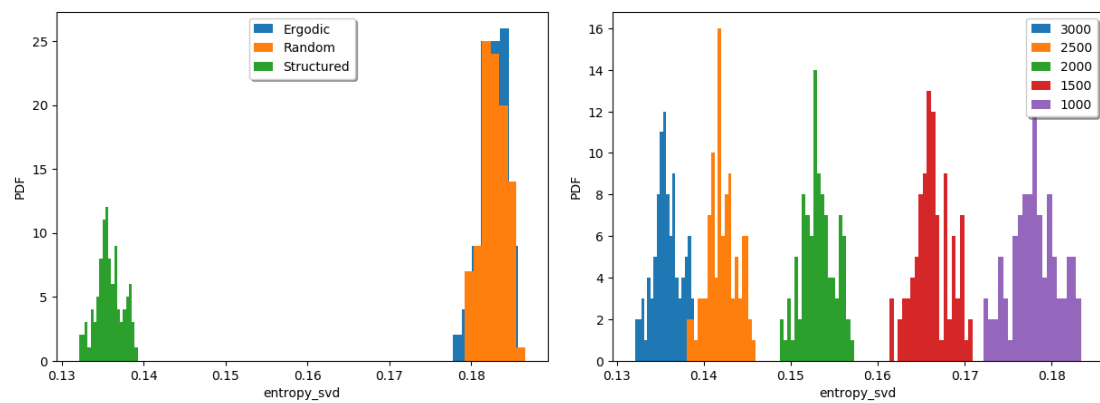
1.4.2 SVD with order 4



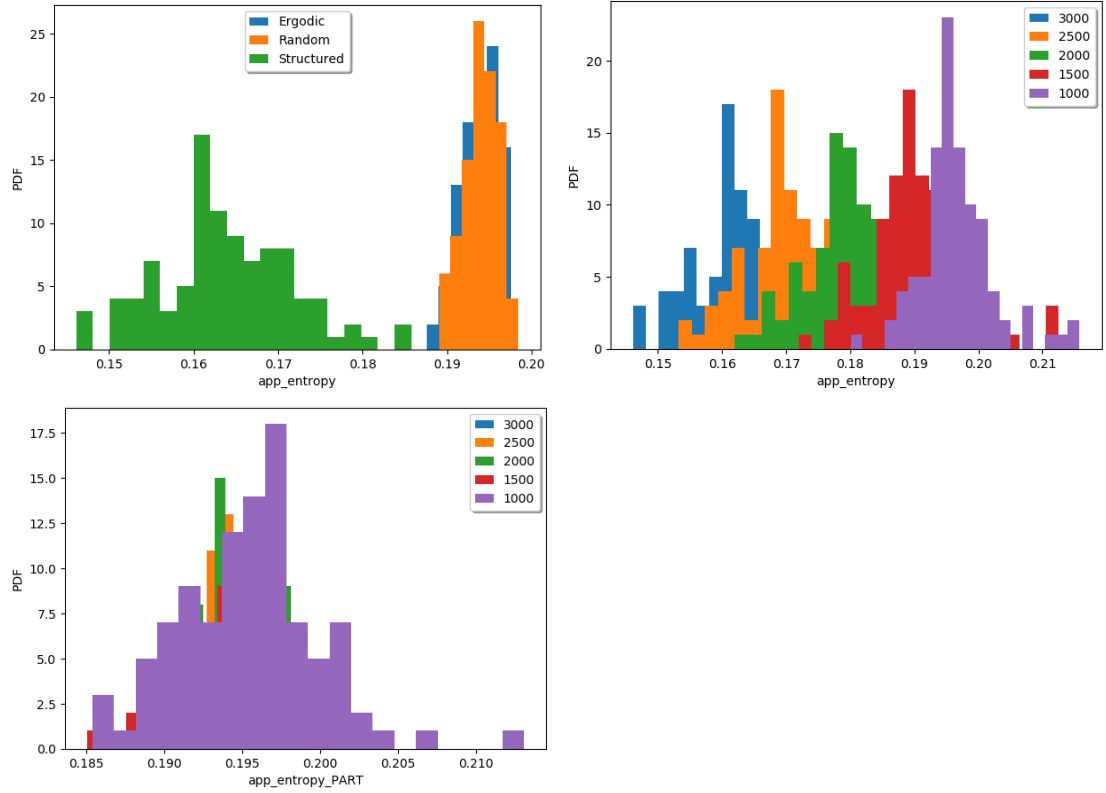
1.4.3 SVD with order 5



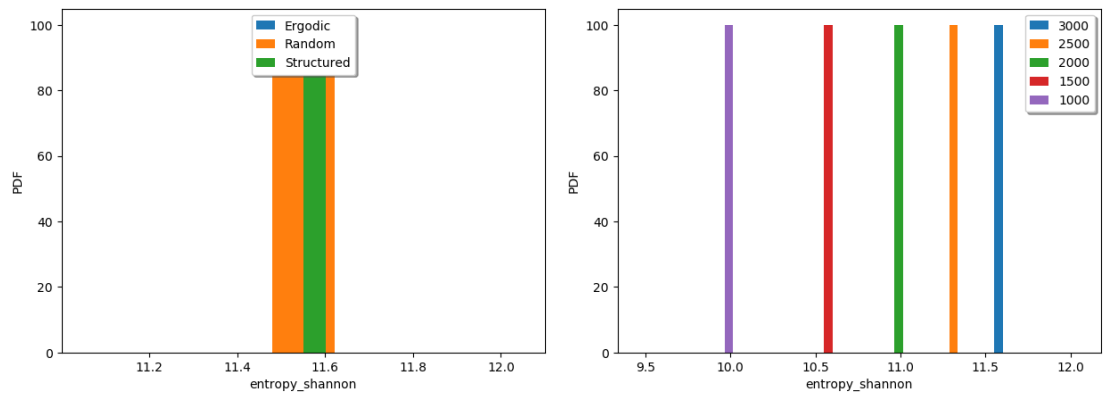
1.4.4 SVD with order 6



1.4.5 Approx Entropy



1.4.6 Shannon Entropy



Shannon Entropy value for each dataset= 11.550746785383529.