Faculty of Science and Technology

Department of Physics and Technology

Complexity-Entropy Analysis of Chaotic and Intermittent Fluctuations in Physical Systems

Sivert Høgli Eilertsen

EOM-3901 Master's thesis in Energy, Climate and Environment 30 SP - June 2021

Abstract

Time series from chaotic and stochastic systems share properties which can make it hard to distinguish them from each other. The Complexity-Entropy analysis provides appropriate measures of entropy and complexity and presenting the calculated values in the representation space Complexity-Entropy plane have been shown to be able to distinguish between time series of stochastic and chaotic origin. It was found that these systems of stochastic and chaotic nature appear in different regions of the Complexity-Entropy plane. The representation space uses the measures of entropy and complexity as orthogonal coordinate axes such that they span a plane, the Complexity-Entropy plane.

Bandt and Pompe presented a new approach to defining a probability distribution from a time series by developing an algorithm to obtain the probability distribution naturally from any given time series. The algorithm is based on comparing neighboring values and the permutation of ordinal patterns of amplitude orderings. Using this probability distribution one can calculate the entropy measure and statistical complexity measure, being the permutation entropy and Jensen-Shannon complexity measure, respectively. The Complexity-Entropy analysis is applied to stochastic and chaotic processes with known locations in the Complexity-Entropy plane to confirm the already established result that these processes occupy different regions in the Complexity-Entropy plane. For continuous-time models, the effect of the discretization timestep through resampling of the time series is investigated. The result from this analysis shows that continuous-time models should be represented by curves in the Complexity-Entropy plane, which differs from what is implied in the literature where these models are represented by specific points. Using the fractional Brownian motion process, the effect of trends in the time series is investigated where it is found that detrending the time series using a running mean approach with a certain window size achieves Complexity-Entropy plane locations widely different from what is expected.

The Complexity-Entropy analysis is then applied to the well-known filtered Poisson process (FPP), which is a stochastic process of a superposition of uncorrelated pulses. The filtered Poisson process can model intermittent fluctuations in magnetically confined fusion plasmas and has been shown to agree favorably with experiment data from fusion experiments. Two versions of the filtered Poisson process are analyzed, one with constant duration times and the other for Pareto distributed duration times. The result from the analysis shows that that these models are represented by curves in the Complexity-Entropy and the largest factor which determines the shape of the curve seems to be the pulse shape. The effect of the distribution of duration times changes the shape of the curve very little compared to changes in the pulse shape. The Complexity-Entropy analysis is finally applied to the Bak-Tang-Wiesenfeld (BTW) model. The results from the different BTW models appeared close to, or on, the fractional Brownian motion line, which supports the notion from the literature that these models can be described by a fractional Brownian motion process.

Acknowledgement

First, I would like to thank my supervisor, Professor Odd Erik Garcia, for the enlightening discussions of the results and the guidance given for this thesis. I would also like to thank Garcia for the suggested corrections and additional points to consider in the thesis.

Likewise, I would like to thank Audun Theodorsen and Martin Rypdal for the discussions and suggestions on models to consider in the thesis, and models which due to time constraints was reserved for further research.

Next, I would like to thank Magdalena A. Korzeniowska and Juan M. Losada for the discussions, and for allowing me to use their simulation data for the filtered Poisson process with Pareto distributed duration times and the Bak-Tang-Wiesenfeld models, respectively. Additional thanks goes to Korzeniowska teaching my how to use the server, and the fusion research group at UiT for giving me access to that server. Without the help of the server, I would not have been able to complete this thesis in time.

Finally, I would like to thank my fellow students, my family and friends for their support throughout the entire thesis process.

Sivert Høgli Eilertsen
Tromsø, June 2021

Contents

1 Introduction

Using measures of entropy and statistical complexity has been demonstrated to be a computationally fast algorithm with the property of being able to distinguish between time series gathered from different physical systems [1, 2, 3]. Time series gathered from stochastic and chaotic systems share several properties making it difficult to distinguish between them [2]. These properties include:

1. Wide-band power spectrum
2. Delta-like autocorrelation function
3. Irregular behavior in the measured signal

However, given appropriate measures of entropy and complexity it is possible to distinguish between time series gathered from stochastic and chaotic systems [1, 2, 3]. By using the measures of entropy and statistical complexity as coordinate axis, the measures of entropy and complexity span a representation space called the Complexity-Entropy plane. A paper published by Rosso et al. [2] applied the Complexity-Entropy analysis to different known systems from stochastic and chaotic origins. The result from the analysis showed that time series from stochastic and chaotic systems occupy different regions in the Complexity-Entropy plane.

There are several known complexity measures such as entropies, Lyapunov exponents and fractal dimensions to name a few [1]. The problem presented by Bandt and Pompe [1] was that these definitions were not made for arbitrary time series. They then presented a different approach and developed an algorithm where the probability distribution comes naturally from the time series from the system under consideration. The probability distribution is based on comparing neighboring values, and the permutation of ordinal patterns from the time series. These probability distributions can then be used to calculate the entropy measure and a statistical complexity measure.

The entropy measure chosen from the analysis is in essence a measure of the randomness of a time series [1, 4]. A time series that is completely random will maximize the entropy measure, while a time series that is entirely predictable will minimize the entropy measure. This is discussed in more detail in section 3.2 and demonstrated in section 6.1. Whilst the complexity measure is a measure of structure in the time series [4]. The chosen complexity measure should reflect the physical structure from the probability distribution obtained using the Bandt-Pompe algorithm [4]. This is discussed in more detail in section 3.3.

The Complexity-Entropy analysis has been applied to plasma experiment data [3, 5, 6]. The analysis has been done on edge density fluctuations [3], temperature fluctuations [5], and magnetic fluctuations [6]. The analysis of all plasma fluctuations resulted in calculated levels of entropy and complexity in the chaotic region of the Complexity-Entropy plane. This confirms the already established result that fluctuations in magnetically confined plasmas are chaotic [3, 5]

An interesting result from the Complexity-Entropy analysis done by Weck et al. [6] on plasma wind tunnel experiment data (SSX) and on data from the solar wind plasma. The analysis highlighted a couple of differences between fluctuations in the solar wind plasma and experimental plasma. The Complexity-Entropy analysis showed that data from the plasma wind experiment had lower levels of entropy and higher levels of complexity compared to the solar wind plasma. This result suggest that compared to fluctuations in the solar wind, the fluctuations in the plasma wind experiment have fewer degrees of freedom. This also suggest that fully developed turbulence, which the solar wind is thought to be, will have high levels of entropy. If this is the case, then the fluctuations in plasma experiments, which is often referred to as turbulent, may not truly be turbulent or may in the least be weakly turbulent [6].

Taking a look at the title of the thesis, it might be beneficial to define some of the terms used. For a working definition for *chaotic*, or chaos, see section 6.3. The term *intermittent* needs to be defined to make sure one knows what is meant when the term is used. From the well-known central limit theorem, as the number of events increases in a given amount of time the probability distribution of the events will often converge towards a normal distribution. For an example where the probability distribution of the events converges towards a normal distribution, see section 7 where the probability distribution of the events of the filtered Poisson process converges towards a normal distribution when the intermittency parameter goes towards infinity. This has led to the definition of what intermittency mean; any variable is said to be intermittent if the probability distribution of that variable is such that extreme events, small or large, are more likely to happen than if the variable was normally distributed [7]. So, if the distribution of events deviates from a Gaussian distribution, one can say that the events are intermittent. A case where intermittency is used is for the stochastic model filtered Poisson process described in section 7. The model has proven itself to be in good agreement with experimental data, can is a good candidate to describe intermittent fluctuation in the Scrape-Off-Layer in magnetically confined plasmas [8]. In fact, under certain constraints on the distribution of the model parameters, the probability distribution of the burst events can be shown to follow a Gamma distribution [8].

The scope of the thesis is the following. To confirm the results presented in the literature, that systems of chaotic and stochastic nature does appear in different regions in the Complexity-Entropy plane. In addition, the effect of sampling frequency of continuous-time systems will be investigated. It will be shown that the sampling frequency will have a large influence on the location of the system in the Complexity-Entropy plane. One model that demonstrates this is the sine function. The sine function is analyzed in section 6.2 and in figure 8 shows the different location the model can take in the Complexity-Entropy plane depending on the sampling frequency used when sampling the function. In the literature the model is implied to be represented by a point in the Complexity-Entropy plane, but the analysis of this thesis shows that this model should rather be represented by a curve in the Complexity-Entropy plane. In addition to showing that the Complexity-Entropy analysis is a non-parametric tool that can distinguish between stochastic and chaotic models, it will also be shown to separate noise and trends in the signals.

The structure of the thesis is the following: A motivation of the Complexity-Entropy analysis is presented in section 2, giving examples of where similar analysis has been applied earlier and benefits of the Bandt-Pompe probability distribution algorithm compared to other methods. The Bandt-Pompe algortim to obtain the probability distribution, entropy measure and complexity measure is presented in the theoretical background section, section 3. Then the implementation of the code for the analysis is presented in section 4, here the design choices and decision are presented. An important element one needs to be aware of is the length requirement for the analysis to obtain reasonable results. This is emphasized with the result presented in section 5. Here, the length requirement is investigated and shows one needs to be careful to make sure the length of the time series are appropriate. Then, the Complexity-Entropy analysis is applied to models with known location on the Complexity-Entropy plane in section 6, therein linear model, white noise, sine function, Lorenz model, logistic map, fractional Brownian motion and fractional Gaussian noise. The Complexity-Entropy is applied to the filtered Poisson process with constant pulse duration times in section 7, then with Pareto distributed duration times in section 8. The final model the Complexity-Entropy analysis is applied to the Bak-Tang-Wiesenfeld model is section 9.

2 Motivation

Using the Bandt and Pompe algorithm one obtains a probability distribution inherently from any given time series. With the obtained probability distribution, a measure of entropy, in the paper by Band and Pompe denoted as permutation entropy [1], and statistical complexity can be calculated. Since Bandt and Pompe introduced the algorithm in 2002, the algorithm has found use in a wide variety of applications. Among them are economics [9] and analysis of effects on anesthetics on the brain [6, 10]. Measurements of complexity have been reported to distinguish healthy and sick subjects using data from the brain and heart [1]. It was found using complexity measures that healthy patients have heart cardiac dynamics which is more complex and more random compared to patients with congestive heart failure [11]. Permutation entropy have also been shown to distinguish between people with brain states associated with Alzheimer's disease, mild cognitive impaired patients and normal healthy elders. There, it was found that Alzheimer's disease patients exhibit lower levels of complexity. Order pattern and complexity analysis have been demonstrated to separate consciousness and unconsciousness [12]. It may even be possible to predict heart attacks [1] and epileptic seizures [1, 13] and absence seizures [14].

The Bandt-Pompe algorithm was used to analyze EEG brain time series from rats undergoing absence epileptic seizures [15]. There the algorithm was used to obtain the ordinal pattern probability distribution, which then was used to investigate "forbidden ordinal patterns". That is, ordinal patterns not accessed by the system because of some underlying deterministic structures. Using this technique, it was found that the EEG time series was more ordered, and therefore less random, under absence seizures compared to a normal EEG time series. As will be discussed in more detail in a later section, the entropy measure in some sense is a measure of randomness. With this in mind, the measured entropy level of the EEG time series will be lower for subjects undergoing absence seizures compared to the normal EEG time series. The result of the study showed that one could predict absence seizures with a 60.2% success rate. This result also supports the conjecture that tonic-clonic epileptic seizures both have entropy and complexity levels that differ from "normal" brain EEG time series [4].

There are several advantages using the Bandt-Pompe algorithm to obtain a probability distribution from a time series. To obtain the probability distribution, the time series are only assumed to be weakly stationary [1]. Compared to other methods to compute the entropy and complexity measures, the Bandt-Pompe algorithm is computationally fast and does not require careful preprocessing of the data [1]. Lyapunov exponents and other complexity measures works great for simulation data of low-dimensional dynamical systems, but some do not work when noise is present in the data. Therefore, when real-world data is used which always contain noise [2], careful preprocessing and noise elimination is required with fine-tuning of parameters, and the obtained results cannot be reproduced without carefully outlining the method used [1]. In contrast, the Bandt-Pompe algorithm still delivers meaningful results when calculating entropy and complexity measures when noise is present. The Bandt-Pompe algorithm for calculating the permutation entropy, and other complexity measures, is robust and is invariant to nonlinear monotonous transformations. This makes this complexity measure ideal when the data sets are huge and there is no time for preprocessing of the time series [1].

3 Theoretical Background

In this section the entropy and statistical complexity measures are presented. Both measures depend on a probability distribution associated with the system in question. The idea behind the Bandt-Pompe algorithm was that the probability distribution must come naturally from the time series from the system. With the probability distribution obtained from the system, the entropy and statistical complexity measures are calculated.

3.1 Probability Distribution

Consider a weakly stationary time series of length L, denoted as $\{x_{t=1,...,L}\}$. The time series is then partitioned into chunks of size d, where d is the so-called embedding dimension. For each partition of the time series the size ordering of the d consecutive values is denoted as a permutation π. The permutation of all L–$d+1$ chunks of the time series are investigated, and the occurrence of each permutation is counted. The number of occurrences of each permutation is then divided by the number of partitions of the time series, $L-d+1$, so that the relative frequency of occurrence of each permutation is obtained. The total number of unique permutations for a given embedding dimension is $d!$. Denoting each permutation with a π_i, the relative frequency of each permutation, $(\pi_1, \pi_2, \ldots, \pi_{d!})$, is what is denoted as Bandt-Pompe probability distribution. This is summarized with the following formula:

$$p(\pi) = \frac{\#[t|t \leq L-d+1, (x_{t+1}, ..., x_{t+d}) \text{ is of type } \pi]}{L-d+1} \tag{1}$$

Lets illustrate the algorithm with the following example.

Let the time series be the following sequence of numbers of length $L = 10$

$$x_t = [6, 2, 8, 10, 5, 4, 1, 3, 7, 9]$$

Also let the embedding dimension be $d = 3$.

Given the embedding dimension, the time series is then partitioned into chunks of size 3, also the total number of permutations the partitioned sequence can take is $d! = 3! = 6$.

Investigating the first partitioned chunk, $[6, 2, 8]$. The size ordering of this chunk is $x_1 < x_0 < x_2$, which means that the corresponding permutation is $\pi_3 = 102$. Next chunk is $[2, 8, 10]$. The entries in this sequence happens to be in increasing order, $x_0 < x_1 < x_2$, and the corresponding permutation $\pi_1 = 012$. The next chunk is $[8, 10, 5]$ which has the size ordering $x_2 < x_0 < x_1$ and thus corresponds to the permutation $\pi_5 = 201$. This is done to the remaining partitions of the time series. Counting the occurrences of all permutation and dividing by the total number of partitions, L–$d+1 = 10$–$3+1 = 8$, gives the relative frequency of the permutations.

The number of occurrence and the relative frequency of all possible permutations are summarized in the following table:

Table 1: List giving all the possible permutations, and the relative frequency of occurrence.

π	Number of occurrence	$p(\pi)$
$\pi_1 = 012$	3	3/8
$\pi_2 = 021$	0	0
$\pi_3 = 102$	1	1/8
$\pi_4 = 120$	1	1/8
$\pi_5 = 201$	1	1/8
$\pi_6 = 210$	2	2/8

To obtain the proper probability distribution for the permutations of the system the length of the time series must tend towards infinity [1]. However, it is assumed that the time series is long enough to get a decent estimation of the probability distribution. The embedding dimension cannot be chosen arbitrarily, and some thought must go into choosing the appropriate embedding dimension for the situation at hand [3].Also, there are practical limitations to the embedding dimension chosen. For the Bandt-Pompe algorithm to achieve accurate results the length of the time series must be significantly larger than the total number of permutations the system can access, that is $L \gg d!$ [1]. For embedding dimension $d = 6$ there are a total of $d! = 720$ possible permutations for amplitude orderings, while for embedding dimension $d = 10$ there are 3628800 possible permutations. The length of the time series is still beholden to the length requirement, $L \gg 3628800$. Meaning as the embedding dimension increases, the length requirement for the time series to obtain accurate results quickly becomes unreasonable. Consequently, Bandt and Pompe recommended that the embedding dimension to be limited to the range $d \in [3, 7]$ [1].

Papers give the common condition for the length of the time series in question to be $L > 5d!$ [6, 16]. As will be demonstrated in section 5, a time series of length $L = 5d!$ should be viewed as a minimum requirement for the length. Using a white noise process it will be demonstrated that a time series of length $L < 5d!$ comes nowhere close to the theoretical point it should have in the Complexity-Entropy plane.

3.2 Permutation Entropy

Using the probability distribution obtained from the Bandt-Pompe algorithm, the entropy measure of the time series can be calculated. The entropy measure of choice is Shannon's logarithmic information entropy [1, 2, 3, 5, 4, 17]. Shannon's information entropy is given as:

$$S[P] = -\sum_{i=1}^{d!} p(\pi_i) \log_2 p(\pi_i) \tag{2}$$

This sum runs over all $d!$ permutations for the given embedding dimension d.

Lets investigate the conditions required to maximize and minimize the entropy measure to gain some intuition. The entropy measure is minimized for a monotone time series. For a monotonically increasing or decreasing time series only one amplitude ordering permutation is accessed by the system with probability $p(\pi) = 1$. This makes the Shannon entropy:

$$S[P] = -\sum_{i=1}^{d!} p(\pi_i) \log_2 p(\pi_i) = -1 \cdot \log_2 1 = 0$$

The maximal value of the entropy measure is for a time series where all possible permutations occur with equal probability. This happens for a completely random time series from an independent, identically distributed sequence [1]. The probability that the system access one state is $p(\pi) = p = \frac{1}{d!}$.

The Shannon entropy is then:

$$S[P] = -\sum_{i=1}^{d!} p(\pi_i) \log_2 p(\pi_i) = -d!(p \log_2 p) = -d! \frac{1}{d!} \log_2 \frac{1}{d!} = -\log_2 \frac{1}{d!} = \log_2 d!$$

The permutation entropy is then defined as the Shannon information entropy normalized to its maximum value:

$$H[P] = \frac{S[P]}{\log_2 d!} \tag{3}$$

The discussion above makes it clear that $0 \leq H[P] \leq 1$.

From this, the entropy measure can be considered as a measure of the randomness of the time series [1, 4]. Ordered time series will have low entropy. As the randomness of the signal increases, so does the entropy until it is maximized for a completely random, uncorrelated time series. But the entropy measure only captures the randomness of the physical system the time series is generated from and this measure cannot determine if there are correlation structures present in the physical system [4]. To aid in capturing these structures a statistical complexity measure is used. The measure considered here is the Jensen-Shannon complexity measure.

3.3 Jensen-Shannon Complexity

The statistical complexity measure should capture the relation between the components of the physical system which influence the probability distribution from the system [4]. The desired behavior for the chosen statistical complexity measure should be small for high degree of order, and for high degree of randomness. For the extreme cases, being a perfectly ordered system and completely random system, the complexity measure should vanish as these types of systems do not possess any structure to speak of [4]. In between these extremes the complexity measure should reflect the possible degrees of physical structure from the underlying probability distribution [4].

In the paper by Martin et al. [4] a family of statistical complexity measures was presented, one of them was the Jensen-Shannon complexity measure. These statistical complexity measures were defined to, in a way, represent a "distance" between the probability distribution obtained from the system to the uniform probability distribution. In general, these complexity measures have the functional form [4]:

$$C[P] = Q[P]H[P] \tag{4}$$

Here $H[P]$ is a measure of entropy and $Q[P]$ represents the "distance" to, or "disequilibrium" from the uniform distribution. $Q[P]$ is given as:

$$Q[P] = Q_0 D[P, P_e] \tag{5}$$

Q_0 is a normalization factor such that $0 \leq Q[P] \leq 1$, and $D[P, P_e]$ is the "distance" from the probability distribution P to the uniform distribution P_e [4].

The disequilibrium $Q[P]$ will be different from zero only if there exist "privileged states", states which the system is more likely to be in. If all states are accessed by the system with equal probability, then there are no privileged states, and the system probability distribution is uniform. Therefore, in this case the disequilibrium measure is zero. Which in turns mean that the complexity measure is zero. For the opposite case where the system only access one state, the disequilibrium measure from the uniform distribution is maximized, meaning that $Q[P] = 1$, but the complexity measure should still be zero in this case. Notice that the complexity measure is the product of the disequilibrium measure $Q[P]$ and an entropy measure $H[P]$ (equation (4)). In the case where the system only access one state, the system is completely ordered meaning that the entropy measure is zero which results in the complexity measure also being zero. Thus, the complexity measure is compatible with its desired behavior described earlier and vanishes in the trivial cases.

For the following Complexity-Entropy analysis the Jensen-Shannon complexity measure is chosen. As mentioned in the paper by Martin et al. [4] and Lamberti et al. [17], the Jensen-Shannon complexity measure is an intensive quantity. Because of its intensive property, the Jensen-Shannon complexity measure is the best choice as a statistical complexity measure for systems of physical origins and hence is the reason the Jensen-Shannon complexity measure was chosen.

The Jensen-Shannon complexity measure is defined as

$$C_{JS}[P] = Q_J[P, P_e]H[P] \tag{6}$$

Here, P denotes the probability distribution obtained from the system in question using the Bandt-Pompe algorithm. P_e is the uniform distribution. $H[P]$ is the permutation entropy defined by equation (3). $Q_J[P, P_e]$ is the normalized Jensen-Shannon divergence, or disequilibrium measure, defined as

$$Q_J[P,P_e] = Q_0 \left[S\left[\frac{P+P_e}{2}\right] - \frac{S[P]}{2} - \frac{S[P_e]}{2} \right] \tag{7}$$

$P + P_e$ denotes the addition between the probability distribution obtained from the system and the uniform distribution. $S[\cdot]$ is the Shannon entropy measure defined by equation (2). The normalization constant Q_0 is given as:

$$Q_0 = -\frac{2}{\frac{d!+1}{d!}\log_2(d!+1) - 2\log_2(2d!) + \log_2(d!)} \tag{8}$$

The complexity measure is not a trivial function of entropy [4]. For a value of entropy in the range $H \in (0,1)$ the complexity measure can take a range of values between a minimum and a maximum value. Martin et al. [4] proved with the use of Lagrangian multipliers that one can define probability distributions that maximizes and minimizes the complexity measure in the range $H \in (0,1)$, essentially defining two curves that gives the upper and lower bound for the complexity measure. These probability distributions were nicely summarized in two tables in the paper by Zhu et al. [3], and are presented in table 2 nad 3 below. Notice that the probability distributions which defines the maximum and minimum complexity lines only depend on the embedding dimension d.

Table 2: The probability distributions that minimizes the complexity measure, taken from table 1 in [3].

Number of states p_i	p_i	range p_i
1	p_{min}	$\left[\frac{1}{d!}, 1\right]$
$d! - 1$	$\frac{1-p_{\text{min}}}{d!-1}$	$\left[0, \frac{1}{d!}\right]$

The probability distribution that defines the minimum complexity line starts of as a uniform distribution where all entries have the same value $p_{min} = \frac{1}{d!}$. Then the probability that the system will access one state increases until the state will be accessed with probability $p_{min} = 1$. The probability for the rest of the $d! - 1$ states decreases accordingly such that $\sum p_i = 1$. For small enough step-size between possible p_{min} values, a smooth curve for minimum complexity is achieved.

Table 3: The probability distributions that maximizes the complexity measure, taken from table 2 in [3]. The range for the number n is $0 \leq n \leq (d! - 1)$

Number of states p_i	p_i	range p_i
n	0	0
1	p_{max}	$\left[0, \frac{1}{d!-n}\right]$
$d! - n - 1$	$\frac{1-p_{\text{max}}}{d!-n-1}$	$\left[\frac{1}{d!-n-1}\right]$

The maximum complexity line is defined by $d! - 1$ different probability distributions. n denotes the number of states in a probability distribution that are zero and increases from 0 states to $d! - 1$ states. For each case of n, one state varies in probability of occurring from 0 to $\frac{1}{d!-n}$. The rest of the states change accordingly such that $\sum p_i = 1$. As is visible in figure 1, the maximum complexity line is not a smooth function [3].

Figure 1 shows how the maximum and minimum complexity line changes with the embedding dimension d. For lower values of d numerical effects are visible.

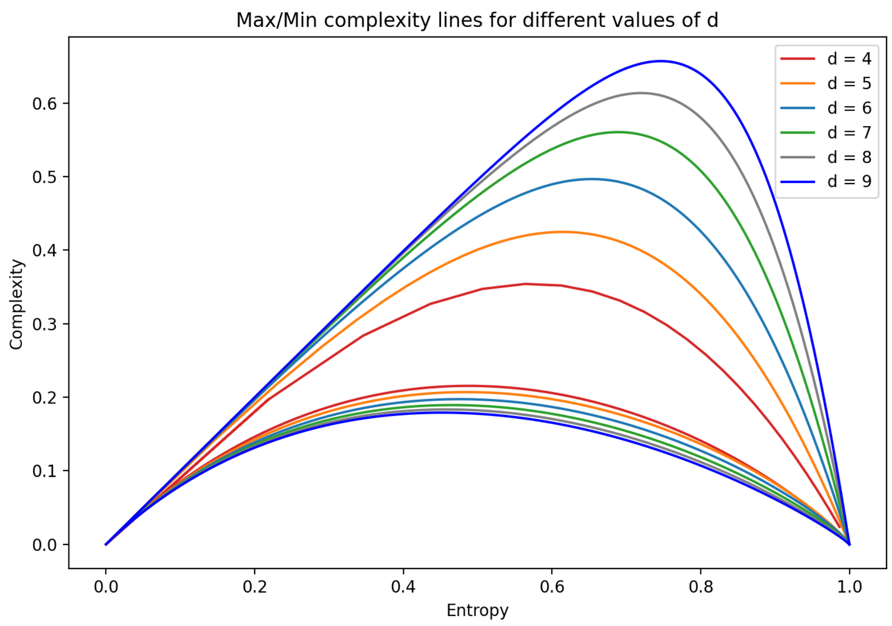


Figure 1: The maximum and minimum complexity lines for different embedding dimensions as function of entropy.

3.4 Embedding Delay and the Effects of Subsampling

Real-world data time series always contain noise [2] and the effect of noise in the time series is an increase in the entropy measure of the time series [5]. Removing the effect of the noise would therefore be beneficial to better capture the nature of the process behind the time series [5]. Let the time between two consecutive data points in the time series be denoted as Δt. The timescale of the amplitude orderings considered when calculating the Bandt-Pompe probability distribution is then $d\Delta t$. If this timescale is small compared to the timescale of the structures of interest, then the amplitude ordering will be dominated by high-frequency noise rather than the structures of the system [3]. To ensure that the amplitude ordering of the segments of the length d better reflect that of the physical structures in the time series, noise removal is required. There are several known and common techniques to remove the effect of noise from the time series. One technique is to digitally filter out the noise from the time series [5]. The Fourier amplitude of the signals are altered, and if the frequency range of the noise can be identified, the noise can effectively be removed from the time series. Another technique to remove noise is wavelet de-noising [5], where the wavelet coefficients of the signals are changed to reduce the contribution of noise in the time series. The downside to these noise removal techniques is that they greatly alter the location the time series takes in the Complexity-Entropy plane, both for stochastic and chaotic time series [5].

However, subsampling the time series by introducing an embedding delay τ is a technique that reduces the effect of the noise while not greatly altering the position of the time series in the Complexity-Entropy plane [5]. Here, not every consecutive data point in the time series is considered, but only every τ data points. From a time series of length L, the new time series is a subsampling of the original time series with a length L/τ. This new time series is then used with the Bandt-Pompe algorithm to obtain the amplitude ordering probability distribution from which entropy and complexity measures are calculated. Essentially, the time series is sampled on a longer timescale which reduces the effect of the high-frequency noise in the data. The new length of the time series must still satisfy the length condition $\frac{L}{\tau} \gg d!$. The embedding delay cannot be chosen arbitrarily, and good choices of embedding delays are restricted [3]. The best choice of embedding delay must both reduce the effect of noise and resolve the features of interest from the time series. If the embedding delay gets too large, then the time step between the data points in the new time series can be longer than the characteristic timescale of the features one seeks to resolve [3].

4 Implementation

The implementation of the Bandt-Pompe algorithm to calculate the ordinal pattern probability distribution and calculating the permutation entropy and Jensen-Shannon complexity was done object-oriented. All functions to calculate the probability distribution, permutation entropy and Jensen-Shannon complexity are contained in a class. In this class framework, class attributes can be defined which can be accessed by all functions contained in that class. Then, say the embedding dimension is not needed to be passed as an argument in every function where it is used. By defining it as a class variable, or class attribute, those variables can be viewed as inherent to the class and can therefore be accessed by all functions contained in the class when needed. Using this class framework, one can define a function to call all necessary functions like the function to calculate the probability distribution, then using the probability distribution call the functions to calculate the permutation entropy and Jensen-Shannon complexity. This is done in the class function `CH_plane` (See section 12, line 145). This function returns the calculated value for permutation entropy and Jensen-Shannon complexity. This effectively replaces three different function calls with one. One for the function which calculates the Bandt-Pompe ordinal pattern probability distribution named the `Permutation_frequency`, one to calculate the permutation entropy `Permutation_Entropy` and one to calculate the Jensen-Shannon complexity `Jensen_Shannon_Complexity`. The last two functions takes one argument, the Bandt-Pompe probability distribution. This property of using a function to call other functions and returning the appropriate values is not exclusive to object-oriented programming and the class-framework. However, if these instances, or objects, from these classes are correctly initialized each object from that class will have access to all functions encompassed by the class. Meaning, if some code is moved from one file to another and some functions are missed in the code transfer, the program will not function as intended. But with the class-framework one can easily transfer the class to the new file and all the functions will follow. The strengths and benefits of using object-oriented programming become more apparent as the programs becomes larger and more complex. Properties of object-oriented programming like inheritance and polymorphism, if used correctly, can make programs easier to understand and use. Lets demonstrate the use of polymorphism and inheritance, starting with polymorphism.

Assume that there exists a list containing objects, `objects`, which are instances of different classes that all contain a function to plot them to a screen, `plot()`. Further assuming there exists a screen to plot them to, one can loop through the `objects`-list and for each object call the plot function. This is demonstrated using some pseudo-code:

```
# list containing objects from different classes
objects = [obj1, obj2, obj3, ..., objn]

# looping through list, calling plot-function
for object in objects:
        object.plot()
```

This general call to the `plot`-function can be done with no knowledge of which class the objects are instances from, nor the intricacies of each plot-function and how they work for each class. As long there exists a `plot`-function in the classes, the appropriate plotting function will be called for each object. This interface, calling a `plot`-function for the objects in question even though the implementation of the plot-function may differ and independent of each other for all objects is the reason for the name polymorphism. One function call for several different implementations of the same function from different classes. A level of abstraction has been created in the code as there is no need to know how each `plot`-function works as long they exists.

Then there is inheritance. If there are two or more classes which share functions, then one class can be defined as a parent-class while the other can be defined as a child-class and "inherit" the shared functions from the parent class. Then functions only need to be defined once. Lets demonstrate this with some pseudo-code.

```
class Parent():
        def __init__():
                # Class variables
                Some_attributes

        def some_function():
                pass


class Child():
        def __init__():
                # Class variables
                Some_attributes

        def some_function():
                pass

        def some_other_function():
                pass
```

In this simplified example, there is two classes `Parent` and `Child`, which share a function called `some_function()`. Here the function is defined twice for the separate classes. Using inheritance defining one as the parent class, and one as the child class so that the child class can inherit the shared function. This is done formally by writing `class Child(Parent):` when defining the child class, indicating that the `Child` class will inherit from the `Parent` class. Then in the initializing function `__init__()` for the `Child` class the function `super().__init__()` is called, initializing the `Parent` class from the `Child` class and making the functions in the `Parent` class available in the `Child` class.

```
class parent():
        def __init__():
        # Class variables
        Some_attributes

    def some_function():
        pass

class Child(Parent):
    def __init__():
        # Class variables
        Some_attributes
        super().__init__()

    def some_other_function():
        pass
```

Now, the shared function `some_function()` is only defined in the `Parent` class, but is made available in the `Child` class. In this implementation, inheritance is used to make the functions to calculate the permutation entropy and Jensen-Shannon complexity available in two classes. Here, the relevant functions are only defined in the parent class `ComplexityEntropy` and inherited by the child class `MaxMin_complexity`.

A downside to object-oriented programming is that, compared to "regular" programming, object-oriented programming is slightly slower and uses more memory. As all instances of objects from a class associates with itself a pointer to where the class and all functions are defined in memory, and therefore takes up more memory. But for a modern computer this is not detrimental to the performance of the program. The number of time series considered is limited and should not pose a memory error. The way this program is implemented, using inheritance and polymorphism, makes this program easy to expand on in the future. It is also of personal opinion that object-oriented programming provides a familiar and easy-to-use framework with a working interface where only one function call is needed to obtain the desired measures of permutation entropy and Jensen-Shannon complexity for each time series considered.

Lets now present the classes which represents the source code for the analysis. The `ComplexityEntropy` class and the `MaxMin_complexity` class. The code is given in appendix A 12

4.1 Complexity-Entropy Class

This class contains the relevant functions to calculate the Bandt-Pompe probability distribution of amplitude ordering, and permutation entropy, and Jensen-Shannon complexity based on the calculated probability distribution. Each instance of this class is initialized with the time series and the embedding dimension as arguments with the embedding delay as an optional argument which can be changed when wanted. If not, it is set to a default value of $\tau = 1$ meaning no delay.

4.1.1 Probability Distribution

Function: `Permutation_frequency()`

The function that calculates the Bandt-Pompe probability distribution of ordinal patterns is `Permutation_frequency`. It is known from theory that the total number of possible permutations for a given embedding dimension d is $d!$. To keep a consistent length to the probability distributions the number of possible permutations is used to pad the probability distribution obtained from the system such that every probability distribution has the same length, $d!$ elements long. If necessary, the Python library *itertools* has a function, `permutation()`, which returns a list of the possible permutation for a range of numbers. Passing the range of numbers from 0 to $d-1$ gives all permutations of amplitude orderings for a given embedding dimension, $d!$. The list of possible permutations is not needed only the total number of permutations, the math library from Python is used to calculate the length requirement of $d!$ elements from the probability distributions. If the embedding delay differ from its default value, the embedding delay is then applied to the time series. The amplitude ordering probability distribution is obtained using the Bandt-Pompe algorithm described in section 3.1. The time series is partitioned into segments of lengths d elements. The permutation of each partition is obtained using the `argsort` function from the NumPy library from Python. The `argsort` function returns the indexes to sort the partition such that its elements appear in increasing order, which is the Bandt-Pompe permutation. The permutation of that partition is then stored in a list. This is done for all $L–d+1$ partitions of the time series. With the use of the `unique` function from the NumPy library, the list of all permutations from the time series can be looped through and all unique entries in that list can be identified and the number of times they appear is counted. The `unique` function returns an array of all unique entries in the permutation list and an array of the number of times they appear, where the same index of the two arrays corresponds to the same entry. Using the array of number of times the unique entries appear, each array datapoint can be divided by the total number of partitions, $L–d+1$ which gives the relative frequency of each unique permutation, or what is here called the probability distribution of permutation. The ordinal pattern probability distribution is obtained and returned by the function.

4.1.2 Permutation Entropy, Shannon Entropy

Function:
`Permutation_Entropy(Probability_distribution)`,
`Shannon_Entropy(Probability_distribution)`

Implementing the Shannon entropy, and consequently the permutation entropy, is straightforward. A function is created implementing equation (2) applied to the calculated probability distribution from the function described previously, where the quantity $p \log_2 p$ is calculated for each entry p in the probability distribution and is then summed up. Finally multiplying with -1 obtains the Shannon entropy. This value is the returned by the function. For the permutation entropy the same applies, only that the retuned value is normalized to the maximum value of the entropy measure $\log_2 d!$ following equation (3). These entropy measures are defined as two separate functions as both variants are used calculating the Jensen-Shannon complexity measure.

4.1.3 Jensen-Shannon Complexity

Function: `Jensen_Shannon_Complexity(Probability_distribution)`

The Jensen-Shannon complexity is the multiplication of the Jensen-Shannon divergence and the permutation entropy following equation (6). A function is created which implements this equation. Again, the Jensen-Shannon divergence is the distance between the ordinal pattern probability distribution and the uniform distribution. Calculating the Jensen-Shannon divergence follows equation (7). The normalization constant Q_0 which follows equation (8) is split up into three parts as to make the expression easier to follow. By construction, the length of the Bandt-Pompe ordinal pattern probability distribution is $d!$ elements long. This guarantees that all probability distributions have a consistent length. And by definition the uniform distribution has $d!$ elements each with value $\frac{1}{d!}$. By making these probability distributions the same length they are compatible and can easily be added together as the list arithmetic adds these lists together elementwise. The Jensen-Shannon divergence is then calculated using the Shannon entropy function following equation (7). The Jensen-Shannon complexity is then the multiplication of the normalization constant, the Jensen-Shannon divergence, and the permutation entropy of the ordinal pattern probability distribution following equation (6) and is returned by the function.

4.1.4 CH-plane

Function: `CH_plane()`

The function `Ch_plane` is the final function in the class. Here, the probability distribution is calculated with a function call to the `Permutation_frequency`-function. The probability distribution is then used as an argument to calculate the permutation entropy and Jensen-Shannon complexity with calls to the functions `Permutation_Entropy` and `Jensen_Shannon_complexity` respectively. This function returns the calculated value of permutation entropy and Jensen-Shannon complexity of the time series considered.

4.2 MaxMin-Complexity Class

The functions contained in this class `Minimum()` and `Maximum()` simply implements the probability distributions defined in table 2 and 3 taken from the paper by Zhu et al. [3]. Lists are defined to contain the Complexity-Entropy plane coordinates as class attributes. The initializing function `__init__()` for the class is passed with the embedding dimension, d, as an argument. With the embedding dimension there is an optional argument, the number of steps p_{min} and p_{max} uses to cover their ranges. Default value set to 500 steps. The functions to calculate the permutation entropy and Jensen-Shannon complexity is used and is inherited from the parent class. These functions are made available to this class using `super().__init__()`. In this case the probability distributions are synthetically made so there is no time series, so the parent class is initialized without passing a time series as argument. Instead the time series is set to `None`, a "null" value. The embedding dimension is passed as normal. The function call to initialize the parent class is:

```
super().__init__(time_series = None, d = self.d)
```

Where d is the embedding dimension, and `self.d` indicates that the parent class is initialized with the same embedding dimension as the class attribute for this class. In the `Minimum()`-function, a list of possible p_{min} values is made, evenly spaced in the range defined by table 2 with the number of steps set by the initializing argument. Looping through the list of p_{min} values, where each entry defines the value of one element in the probability distribution. The rest of the $d! - 1$ entries in the probability distribution are equal in size and defined such that $\sum p = 1$ holds. The probability distribution is then passed to the functions for permutation entropy and Jensen-Shannon complexity from the parent class and the retuned values are stored in the appropriate lists.

For the `Maximum()` function the same happens, but here there exists another encompassing loop which fills the probability distributions with entries of zero value following table 3. The p_{max} values have a different range, and the rest of the entries in the probability distribution are still defined such that they have the same value and $\sum p = 1$ holds.

5 Time Series Length and Convergence

In this section, the effect of different time series lengths on the location of the time series in the Complexity-Entropy plane is discussed. To illustrate this effect, a white noise process was simulated with different time series lengths and for each time series, the Complexity-Entropy analysis was applied.

The general length requirement that the time series must satisfy is $L \gg d!$ [1] as mentioned in section 3.1. The importance of this length requirement is to make sure the results obtained in the Complexity-Entropy analysis are accurate. Some papers also give a common condition on the length of the time series where $L > 5d!$ [6, 16]. Using this time series length condition as a basis, the white noise process was simulated to generate time series with lengths equal to different multiples of $5d!$. The white noise process time series was generated to have lengths of $0.1, 1, 2, 5$ times the length condition $5d!$. The Complexity-Entropy analysis of the different realizations gives the following result:

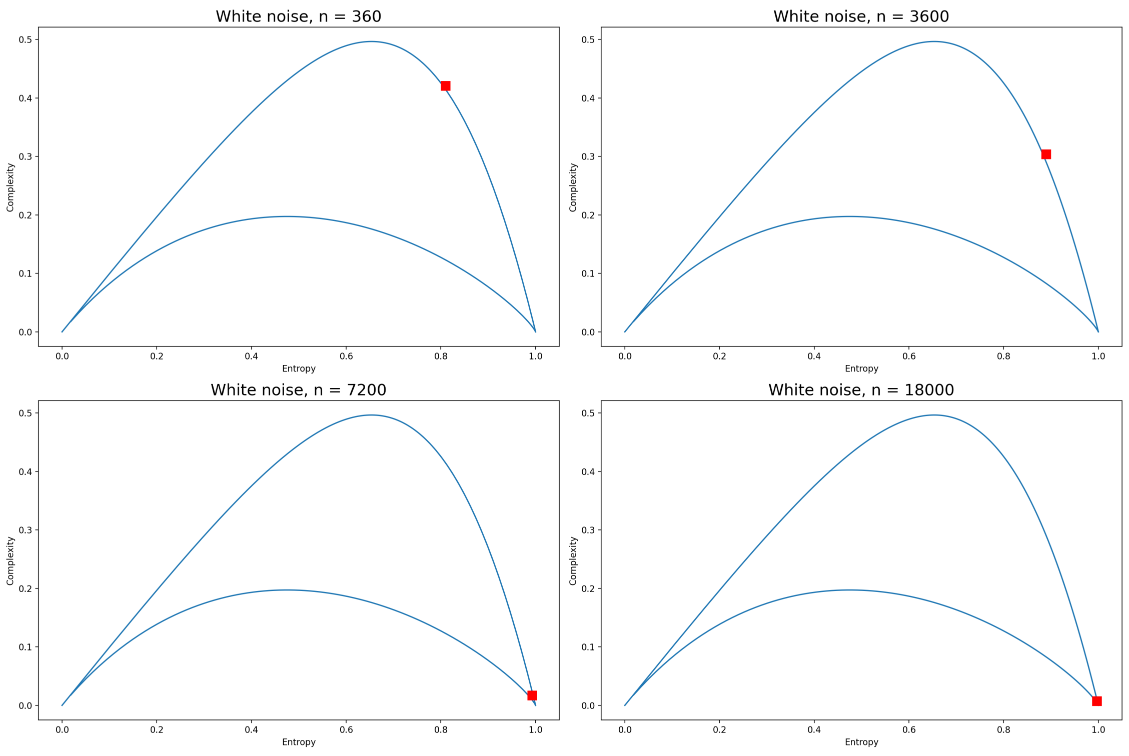


Figure 2: Complexity-Entropy analysis for white noise process with time series lengths: $L = 0.1 \cdot 5d!$ (top left), $L = 1 \cdot 5d!$ (top right), $L = 2 \cdot 5d!$ (bottom left), $L = 5 \cdot 5d!$ (bottom right)

What is immediately obvious is that the length of the time series used in the Complexity-Entropy analysis has a great effect on the location of the time series in the Complexity-Entropy plane. The top plots in figure 2 had the shortest time series and both ended up in the completely wrong location in the Complexity-Entropy plane, see discussion in the theory section 3.3 and the proper analysis of the white noise process in section 6.1. For the bottom plots in figure 2, the calculated locations of the time series are close to the theoretical point in the Complexity-Entropy plane, in the bottom right corner. But the time series with length $L = 2 \cdot 5d!$ deviates slightly from the theoretical point with levels of entropy slightly lower than 1 and complexity levels slightly higher than 0. For the time series of length $L = 18000$ the Complexity-Entropy plane location of the white noise process is at the theoretically expected point.

However, the white noise process is special in this case. The signal from a white noise process is completely random, and each datapoint is uncorrelated uncorrelated to its neighbors. Since the signal is completely random, the amplitude ordering of the time series must be completely random too. Because of the uncorrelated nature of the datapoints, one needs to generate a sufficiently long time series to capture every amplitude ordering with the appropriate frequency as every amplitude ordering in a completely random time series should appear with the same frequency. The ideal case, theory wise, is to generate an infinitely long time series for each process. This is ,however, not possible. At some point, one has to cut the simulation short, but care must be taken to make sure the time series is not cut too short to get an amplitude ordering probability distribution that is true to the process.

6 Processes with Known Location in the Complexity-Entropy Plane

In this section, time series form models that have known locations in the Complexity-Entropy plane as they have been studied in other publications. More specifically the linear model and white noise process is studied as their locations in the Complexity-Entropy plane are readily available from the theoretical background section 3.2 and 3.3. These two models represent the trivial cases for the entropy and complexity measures. The Complexity-Entropy analysis was then applied to a periodic function, being the sine function. The chaotic models chosen was the logistic map and the Lorenz model. Then the stochastic model the fractional Brownian motion and fractional Gaussian noise processes was studied. The Complexity-Entropy analysis of the fractional Brownian motion is used in the literature as a guide and can be viewed as one of the highest complexity stochastic processes and is used to separate the chaotic region and the stochastic region of the Complexity-Entropy plane [5]. An additional step in the analysis was done for the continuous-time models. The reason for the additional step was the weird results from the Complexity-Entropy analysis done in previous work in the Project Paper. The location of the Lorenz model with model parameters where the model is chaotic was not in the chaotic region of the Complexity-Entropy plane, this was also the case for the initial filtered Poisson process Complexity-Entropy analysis from the same work. Initial analysis to explain these odd locations in the Complexity-Entropy plane indicated that the location of these continuous-time models was tied to the discretization timestep of the simulations of these models. This relationship with the location in the Complexity-Entropy plane was carried on into this thesis. To simulate a rougher discretization timestep, a long time series of these continuous-time models (sine function and Lorenz model) was made with a fine discretization timestep, before the time series was then resampled with an increasing lag in the resampling algorithm. Each resampled time series underwent the Complexity-Entropy analysis and plotted in the same figure, the calculated Complexity-Entropy plane locations of these resampled time series produced a curve in the Complexity-Entropy plane. The discretization timestep of the resampled time series would be changed following:

$$\Delta t \qquad \overrightarrow{\text{Resampling}} \qquad \Delta t' = lag \cdot \Delta t \tag{9}$$

Then if the original discretization timestep was 0.001 and the resampled time series was sampling the original time series with a lag of 500 datapoints, the simulated discretization timestep of the resampled time series would then be $\Delta t' = lag \cdot \Delta t = 500 \cdot 0.001 = 0.5$. After te resampling of the time series, one could plot the entropy and complexity measures as functions of lag. One would then see how the measures of entropy and complexity evolves with the lag in the resampling. The inspiration for these plots was taken from the work done by Zunino et al. [18]. What will be shown is that the location of the continuous-time models in the Complexity-Entropy plane changes very much with the discretization timestep of the simulation, and rather than what is implied from previous publications, span a curve in the Complexity-Entropy plane and not in a specific point.

6.1 Trivial Cases

These cases represent the trivial cases for the Complexity-Entropy analysis. Being from a system which is predictable with certainty, or a completely random system. Their location in the Complexity-Entropy plane has been discussed in section 3.3, and can be used to test the Complexity-Entropy algorithm to see if it is compatible with the theory behind the complexity measure.

6.1.1 Linear Model

This model follows the general linear equation $f(x) = ax + b$. The parameters of the model were chosen to be the following: $a = 0.01$, $b = 1$, making the equation for the model to be

$$f(x) = 0.01x + 1 \tag{10}$$

It is trivial to see that a time series simulated from the model equation the datapoints will have the following relation $x_t < x_{t+1}, \forall t > 0$. This means that only one amplitude ordering permutation will be accessed by the system. As discussed in section 3.2 and section 3.3, the entropy measure for this model will be zero, and likewise for the complexity measure, and thus occupy the lower left corner of the Complexity-Entropy plane.

The model was simulated following equation (10) with 10000 datapoints where the x values was every whole number in the range $x \in [0, 10000]$.

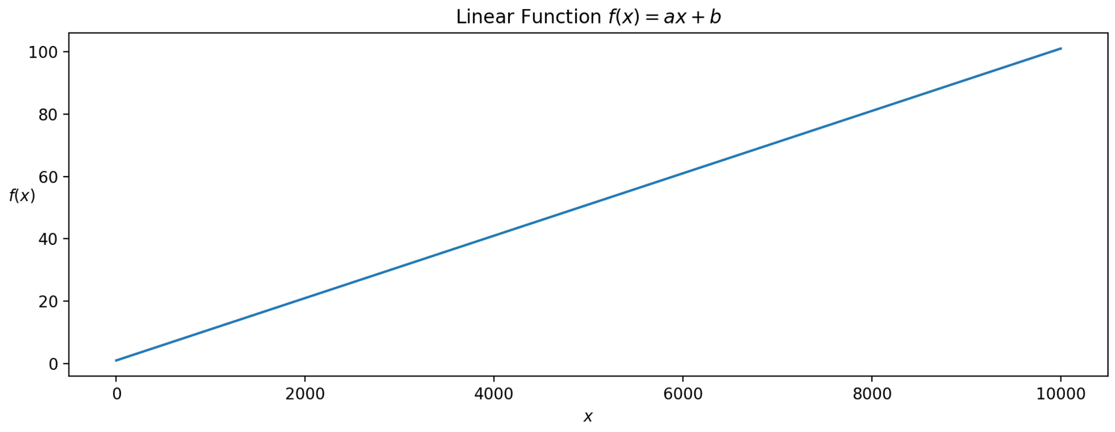


Figure 3: Time series plot of the linear model.

The Complexity-Entropy analysis for the linear model and the white noise process are combined in the same plot, see figure 5.

6.1.2 White Noise

This model represents the other trivial case described in the theory section. The datapoints in this model was selected from a normally distributed variable with zero mean and unit standard deviation, $W \sim \mathcal{N}(0, 1)$. The time series then contain an uncorrelated sequence from an independent, identically distributed variable where all possible amplitude permutation will appear with equal probability [1]. Since the sequence is uncorrelated, the only issue with the simulation is that one needs to make sure that the length of the simulation time series is long enough to capture every amplitude permutation at roughly the same frequency. This ties into the length requirement for the time series $L \gg d!$ as discussed in section 5, where the effect of this is discussed.

The white noise process was simulated with 20000 datapoints using the Random module from Python. Figure 4 shows a plot of the simulation time series where the first 1000 datapoints were plotted.

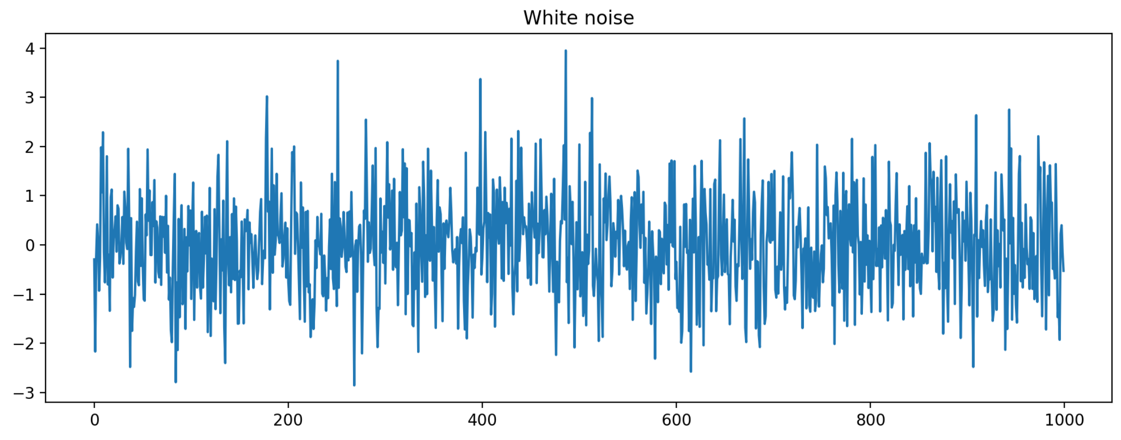


Figure 4: Time series plot of the white noise process. Only the first 1000 datapoints are plotted.

Since this a completely random process one expects from theory that this process will occupy the lower right section of the Complexity-Entropy plane, since the entropy measure will be maximized and therefore the complexity measure must be zero by design. This leads to the model's theoretical point at $H = 1$ and $C = 0$.

The following plot shows the Complexity-Entropy plane location for both the linear model and the white noise process.

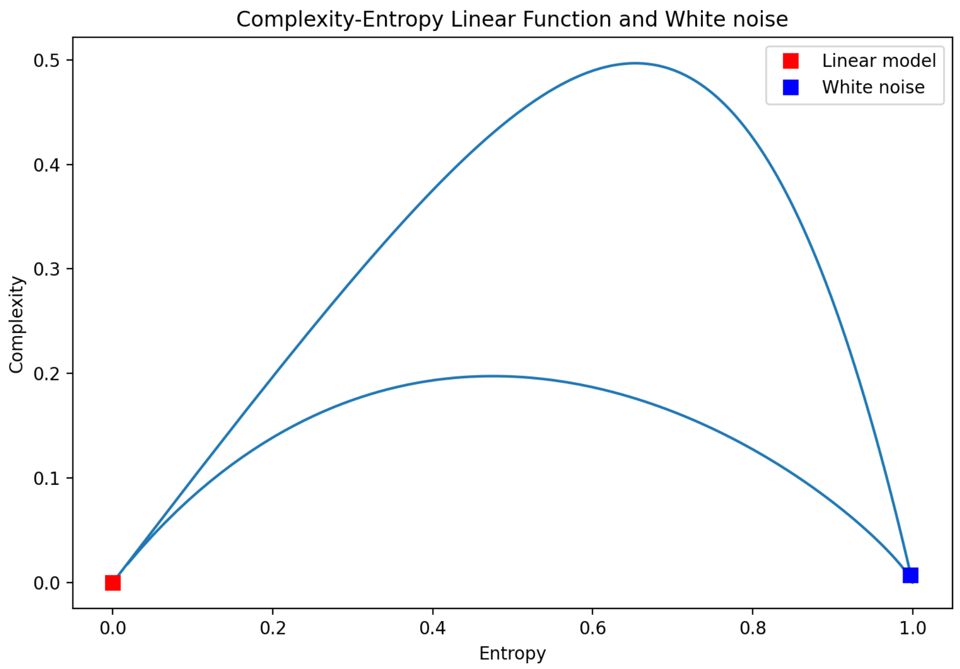


Figure 5: Complexity-Entropy plot of the linear model and the white noise process. Both models appear in their theoretical locations.

The Complexity-Entropy plane location of the linear model and the white noise process are both at their prescribed theoretical location.

6.2 Sine Function

Next the Complexity-Entropy analysis was applied to a periodic function, the sine function. This function contains local regions where the function is monotonic with a periodic switching between a region with a monotonic increasing and decreasing signal. This behavior of the signal indicates that the measured value of entropy will be low and in the neighborhood of the linear model, but the change from one monotonic region to another opposite monotonic region will make the entropy measure a little higher than a linear model. The reason for the low value of entropy is because the system, with its monotonic regions and periodic change between them, will only access a few amplitude orderings and the resulting entropy measure will then be low. A signal produced from the sine function is very structured meaning that the complexity measure should be maximized for the calculated entropy value.

The initial simulation of the sine function was done for a normal sine function $\sin(\omega t)$ where $\omega = \pi$ with a time difference between consecutive datapoints set to $\Delta t = 0.001$. The time variable, t, ranged from 0 to 10 making the initial length of the time series 10000 datapoints.

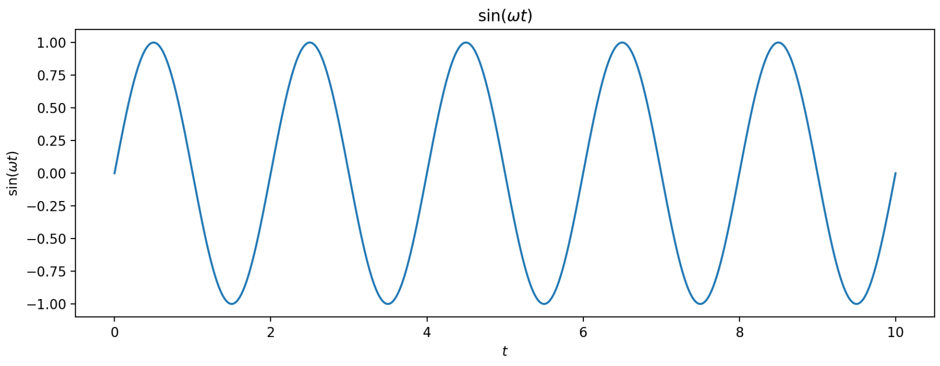


Figure 6: Time series plot of the initial simulation of the sine function.

The time series from the simulation produced the following position in the Complexity-Entropy plane:

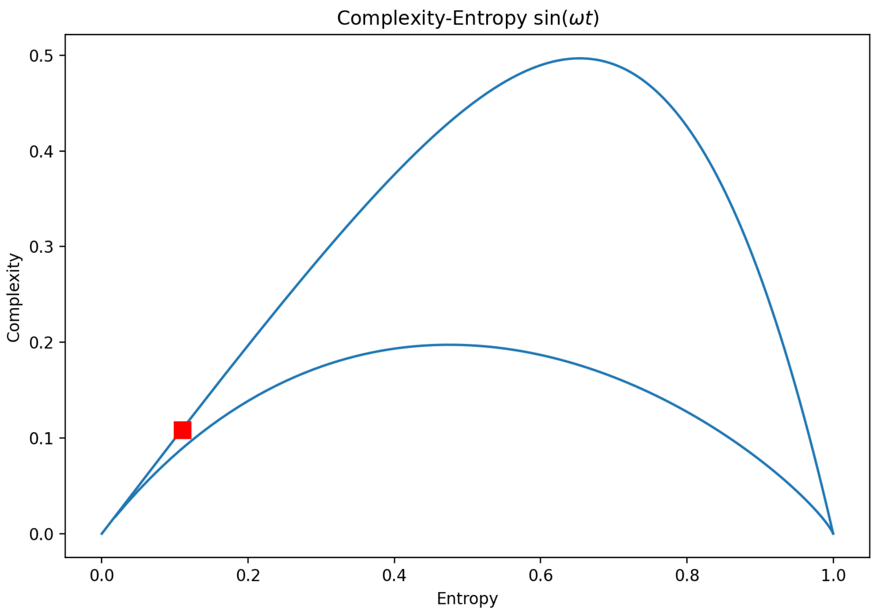


Figure 7: Complexity-Entropy plot of the sine function.

With these continuous-time models and systems, one can tweak the timestep between the considered datapoints. For this, a very long simulation of the sine-function was used. With the original signal as a base, the signal was then resampled with an increasing amount of lag, considering every 2nd datapoint, then every 3rd datapoint and so on. The original length of the sine-function simulation was 10 million datapoints. The discretization timestep in the original time series was kept the same as the initial simulation with $\Delta t = 0.001$. Then the original time series was resampled with a coarser discretization constant by introducing a lag between the considered datapoints creating a new, smaller time series from the original time series. The range of the considered lag when resampling the original time series was from 1 to 2000. For a lag of 1, every datapoint from the original time series was considered and remained unchanged. For a lag of 2000, only every 2000 datapoints from the original time series was considered. The length of the new time series is then 1/2000 of the length of the original time series. Because of this, the length of the original time series was then a requirement, and not an arbitrarily chosen value. To reach the length requirement of the time series to obtain reasonable results from the Complexity-Entropy analysis, and with the resampling scheme chosen the time series had to be this long. The original time series and all resampled time series of the original underwent the Complexity-Entropy analysis and plotted on the Complexity-Entropy plane and produced the plot in figure 8:

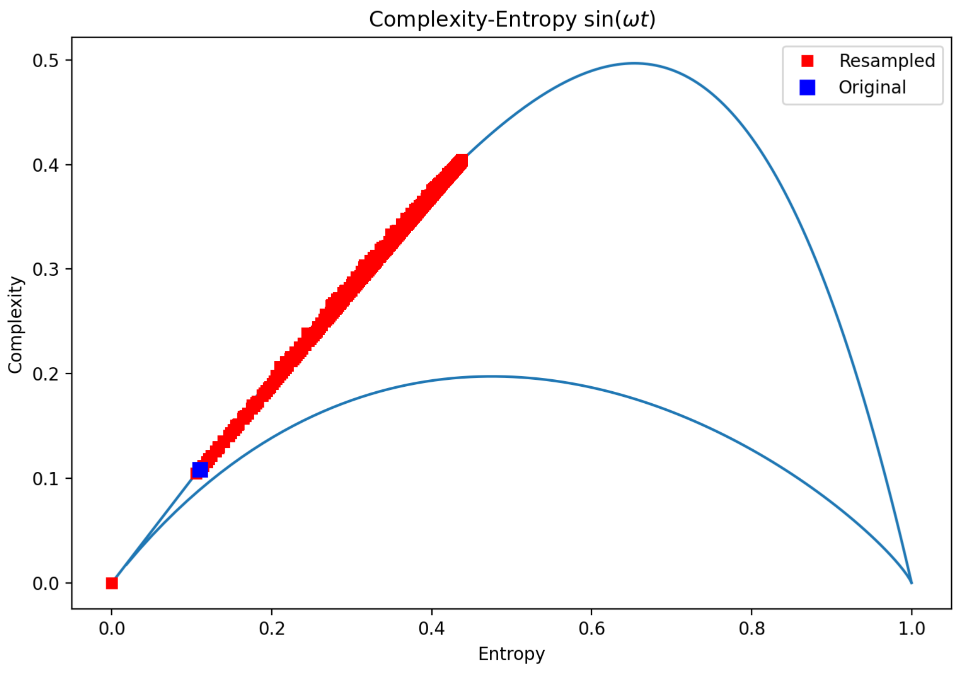


Figure 8: Complexity-Entropy plot for the resampled sine function for 2000 different resampling lags.

The first result that stands out is the point at $H = 0$ and $C = 0$. This point is for the resampled signal with a lag of 2000 datapoints. Looking at the argument passed, ωt, to the sine function, we can see the reason for this. The frequency of the sine-function was $\omega = \pi$, while the time sampled was $t = 0.001n = \frac{1}{1000}n$ in the original time series. With the introduced lag, the new sampling time was $2000t = 2000\frac{1}{1000}n = 2n$. The argument of the sine-function was always a multiple of 2π, meaning that the same point was always sampled, and the resampled time series was a constant time series. The way the Bandt-Pompe algorithm was implemented when the time series is constant and all entries are the same, the index at which they appear will be their amplitude permutation. This means that a partition with the entries [2, 2, 2, 2] will have the amplitude permutation [0, 1, 2, 3]. This results in the constant time series being represented in the same way as a linear time series, which explains the position of the time series in the Complexity-Entropy plane.

To get a better sense as to how the resampling affects the Complexity-Entropy plane location for the model, only the original time series and a select few resampling lags were considered. The original, unaltered time series was included in the Complexity-Entropy analysis as a benchmark. The signal was resampled for the lags $[100, 150, 300, 400, 221, 2000]$ and the results plotted in the Complexity-Entropy plane. Also, to get a better understanding on how the lags alter the time series, the first 3000 datapoints of the original time series was plotted and the sampled datapoints for the different resampling lags was also marked in figure 9.

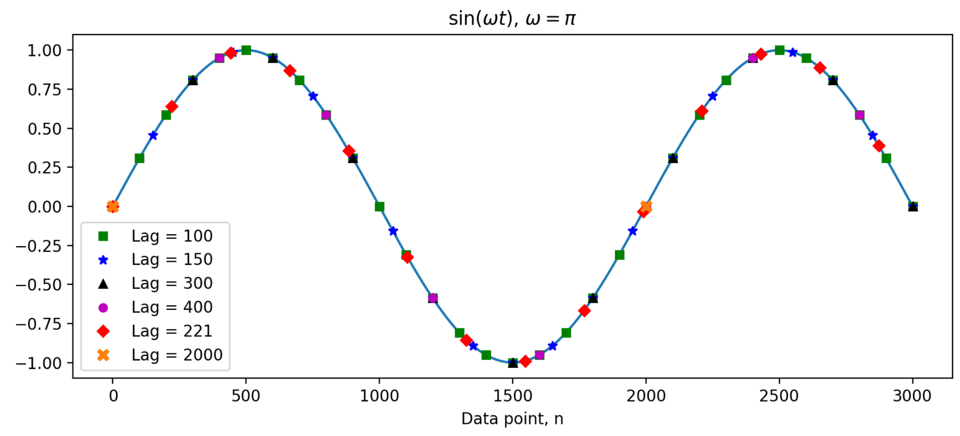


Figure 9: Time series plot of the original time series with the datapoints considered at different resampling lags marked.

The original and the resampled time series produced the Complexity-Entropy plane locations presented in figure 10.

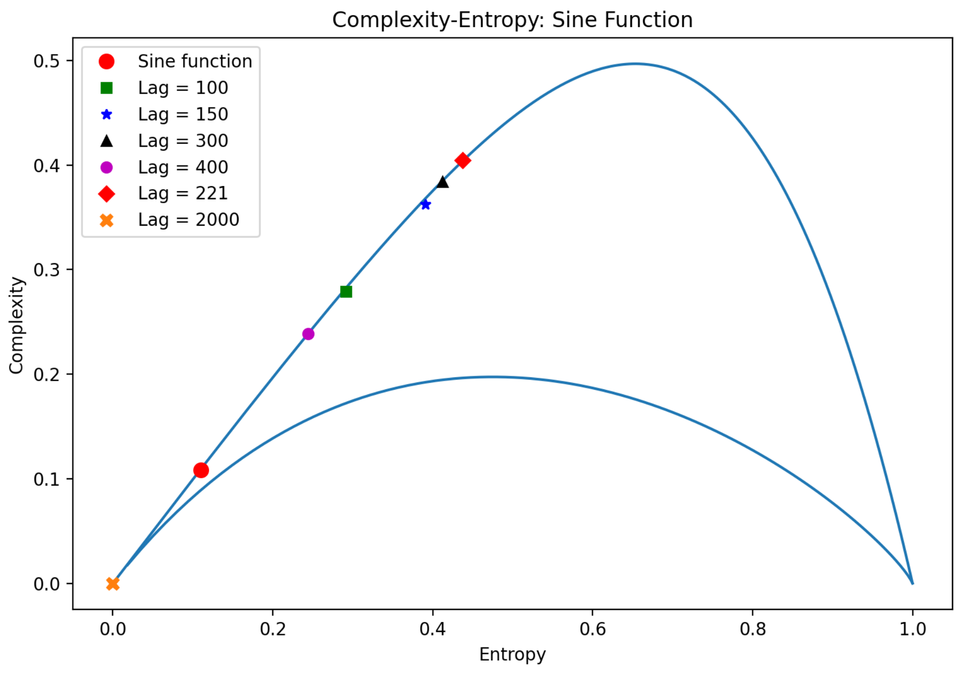


Figure 10: Complexity-Entropy plot of the sine function and resampled signals as different resampling lags. The original time series is marked as a red circle.

One can see the complexity and entropy measures increase for increasing lag for the lags of 100, 150 and 300 datapoints. Then the complexity and entropy measure drop for lag of 400 datapoints. The resampled signal with lag of 221 datapoints represents one of the highest achieved levels of complexity and entropy for the sine function. Only one other lag achieves higher entropy and complexity levels. The calculated levels of entropy and complexity for the two different time series only differ after the 4th decimal point.

From the included plots it was difficult to obtain a clear relation between the obtained complexity and entropy measure from the signal and the resampling lag. This behavior seemed odd since the highest achieved levels of entropy and complexity happened for a lag of 221 datapoints (and for 1110 datapoint lag) and the measures of entropy and complexity drops for a lag of 400 datapoints. Looking closer at figure 9 with lag of 221 datapoints, the 10th sampled datapoint comes close to that of lag 2000, where the sampled points always will have the same value. This indicates that it will take a long time for the same points to be sampled again and repeating the same amplitude ordering permutations, resulting in the calculated levels of entropy and complexity being high.

Notice that for lag of 300 the entropy and complexity measures are high, and compatible with the maximum achieved values. Then, for lag of 400 the entropy and complexity levels drop significantly. This indicates that when the lag reaches levels so that the argument of the sine function becomes a nice fraction of 2π the entropy and complexity levels drop. With a lag of 400 datapoints the argument of the sine-function can be calculated,

$$\text{lag} \cdot \omega t = 400 \cdot \frac{1}{1000} n\pi = 0.4\pi = \frac{1}{5} n \cdot 2\pi$$

meaning it takes 5 points before the sampled datapoints repeats themselves. So, the number of achieved amplitude orderings will also be limited.

To get a clearer sense on how the Complexity-Entropy plane location changes with the resampling lag the entropy and complexity measure changes as a function of lag, where the lag in resampling is normalized to the period of the sine function, T:

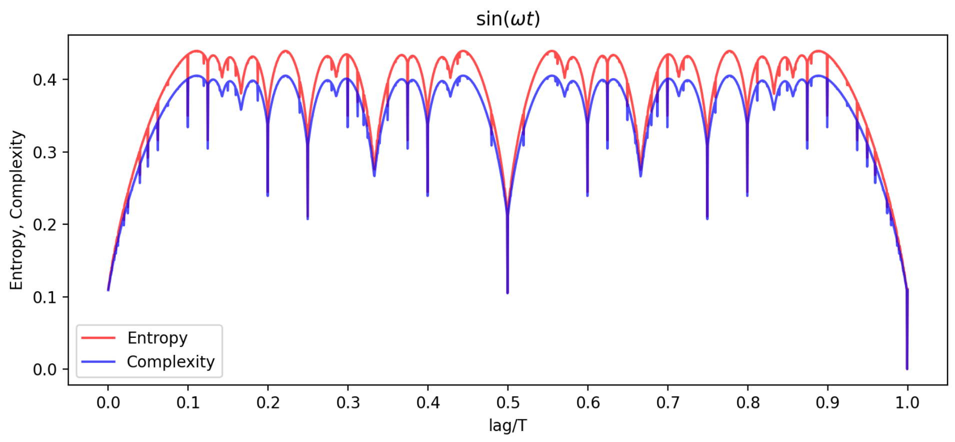


Figure 11: Complexity and Entropy plotted as functions of lag for different resampling lags. The lags ranges from 1 to 2000 normalized to the period of the sine function

Here, one explicitly sees how the entropy and complexity measure changes with lag. The entropy and complexity measures drop for lag values with low order fractions of 2π like for lag/T = 0.25, 0.5, and 0.75 to name a few. One interesting feature of this figure was that it is symmetric around the 0.5 lag/T mark. This plot also captures the non-trivial rise and fall of the complexity and entropy measures with increasing lag which was touched on earlier. Moreover, it captures the noticeable drop in entropy and complexity at a lag of 2000 datapoints. It reaches levels of entropy and complexity much lower than for every other lag and it is expected that the figure will be symmetric around the 2000 datapoint lag mark as well, since the sampled datapoints will have the same condition as the ones sampled in the beginning just one period apart.

6.3 Lorenz Model

The Lorenz model is a set of ordinary differential equations which Edward Lorenz derived from simplifying the set of differential equations derived by Saltzman which studied finite amplitude convection [19]. This convection model which Lorenz derived is given by the equations:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y-x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z \end{aligned} \tag{11}$$

The model parameters are σ the Prandtl number, ρ the Rayleigh number (denoted as r in the paper by Lorenz), and the unnamed parameter β.

For $x = y = z = 0$ the system of equations possesses a steady-state solution, representing a state with no convection [19]. It can be shown that for $\rho < 1$ every trajectory produced by the equations will approach the origin as time tend towards infinity [20](chapter 9). This led to the criterion of onset convection at $\rho = 1$ [19]. Critical values for $\rho > 1$ is found when the complex conjugate roots of the characteristic equation of the matrix defined from equation 11 are purely imaginary, and given by the relation:

$$\rho_H = \frac{\sigma(\sigma + \beta + 3)}{\sigma - \beta - 1} \tag{12}$$

and represents the value of ρ for the instability of steady convection, meaning that for $\rho > \rho_H$ the convection is unstable.

In the paper by Lorenz [19] the chosen parameter values followed Saltzman, where $\sigma = 10$ and $\beta = \frac{8}{3}$. These parameters gave the critical value for the Rayleigh number $\rho_H = \frac{470}{19} \approx 24.74$. Lorenz then chose the slightly supercritical value of $\rho = 28$.

After numerically integrating the system of equations (11) for a long period of time and plotting a projection of the 3D space on the 2D plane of y and z, Lorenz commented the following:

> *The trajectory apparently leaves one spiral only after exceeding some critical distance from the center. Moreover, the extent to which this distance is exceeded appear to determine the point at which the next spiral is entered; this in turns seems to determine the number of circuits to be executed before changing spirals again. It therefore seems that some single feature of a given circuit should predict the same feature of the following circuit.* (Edward Lorenz, 1963 [19])

The "single feature" was the local maximum of $z(t)$ where the idea was that the maximum z_n should predict the next local maximum point z_{n+1}. This idea lead to the function $z_n = f(z_n)$ which is now known as the Lorenz map. Using this, Lorenz was able to provide sufficient evidence that the solutions with these model parameters were in fact not stable limit cycles with an incredible long period not captured by the computer simulations [19] [20](chapter 9). By limit cycle meaning a closed trajectory. Since, "playing the devil's advocate" and using the result Lorenz showed that all trajectories for the system will become confined to a subspace having zero volume [19]. But using the Lorenz map, Lorenz was able to show that if any limit cycles exists they are unstable, and in fact all trajectories are unstable [19]. Therefore, any periodic trajectories one may encounter are unstable, and every other trajectory then represents deterministic nonperiodic flow. The chosen parameters of the model will yield what's called a strange attractor, an attractor that are sensitive to the initial conditions [20]. Any small deviation in the initial condition between two instances of the model will quickly lead to two very different states for the model at the same point in time.

This result when applied to the atmosphere indicates that the long-term prediction of the conditions of the atmosphere in the future is impossible, and the inevitable inaccuracy in the weather observations makes it impossible to obtain precise long-range forecasts of the weather. Using the working definition of chaos in the book by Strogatz where chaos is defined as "*Chaos is aperiodic long-term behavior in a deterministic system that exhibits sensitive dependence on initial conditions.*" [20], the Lorenz system of equations under certain conditions on the model parameters is chaotic.

Versions of the Lorenz map has been used in other works, and the Complexity-Entropy analysis has been applied to Lorenz maps, see the works of Rosso et al. [2]. There, the same technique to define the Lorenz map was used on the Rössler's oscillator. Using the Lorenz map of the Lorenz model is beyond the scope of this thesis. Here, the Complexity-Entropy analysis applied to the Lorenz model itself is of interest.

The analysis of the Lorenz model was done using numerical integration of the model equations 11 with the same model parameters as Edward Lorenz, $\sigma = 10$ and $\beta = \frac{8}{3}$ and $\rho = 28$ and with the same initial conditions $\mathbf{x}_0 = (0, 1, 0)$. The initial simulation of the Lorenz model with an integration time step set to $\Delta t = 0.001$ and a defined time array from 0 to 50 such that the length of the time series for the position in all coordinates was 50000 elements long. The simulation produces the following plot of the position in 3D. The code for the simulation was sourced from the Lorenz system article from Wikipedia[1]

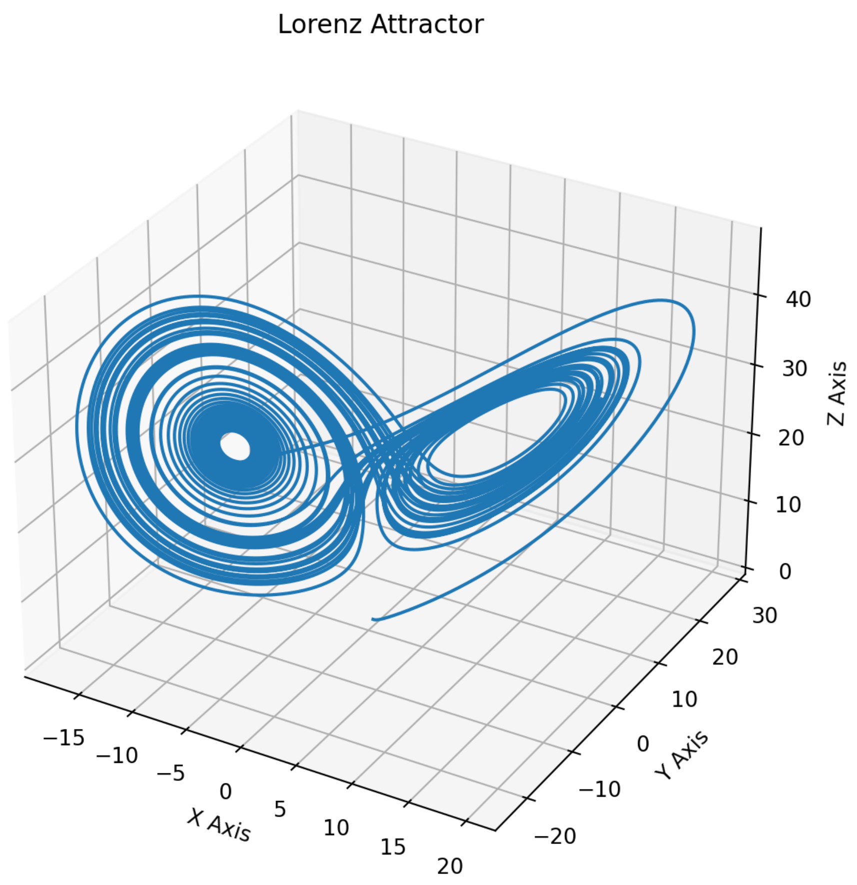


Figure 12: 3D plot of the Lorenz model for the model parameters $\sigma = 10$, $\beta = 8/3$, $\rho = 28$

[1]https://en.wikipedia.org/wiki/Lorenz_system

The Complexity-Entropy analysis was then done on each coordinate time series separately. The Complexity-Entropy analysis produced the plot of the model in the Complexity-Entropy plane presented in figure 13:

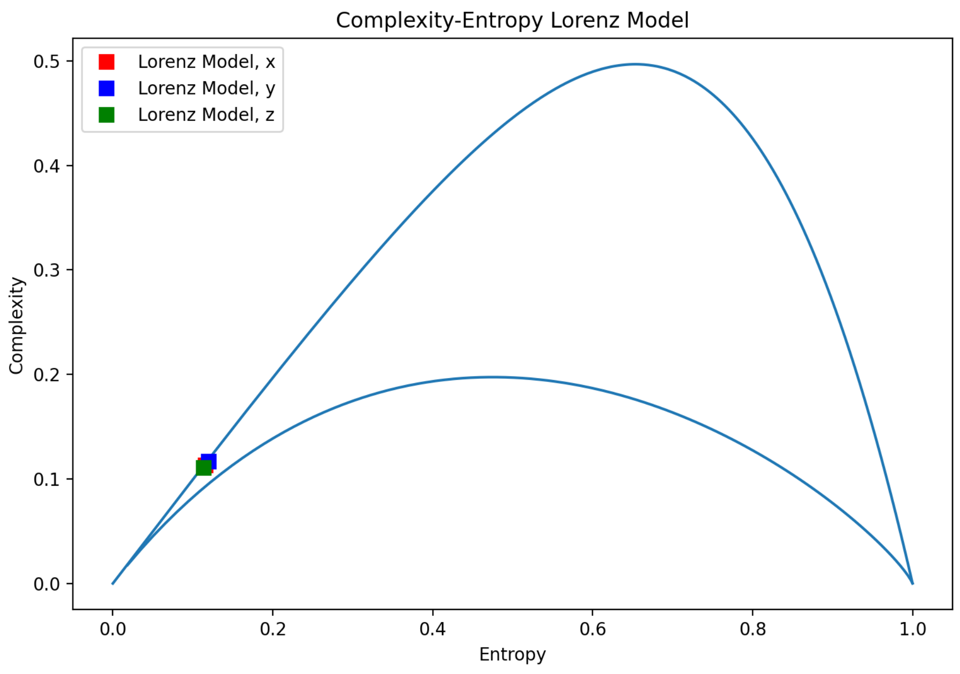


Figure 13: Complexity-Entropy plot of the time series of the system variables x, y, and z of the Lorenz model.

From theory, for the current model parameters for this model should result in chaotic dynamics. However, the calculated position in the Complexity-Entropy plane results in the model appearing in the region close to the non-chaotic, highly ordered sine-function. The simulation does not reflect the chaotic nature of the Lorenz model for the given model parameters. The fine time step for the numerical integration of the mode equation preserves the shape of the trajectory, but this preservation of the shape of the trajectory leads to a slow change in the amplitude orderings for the given embedding dimension d. This slow change in the amplitude orderings is what gives the low value of entropy and complexity for the Lorenz model. To investigate the role of the time step in the solution of the model equations, the same method that was done for the sine-function is applied to the Lorenz model.

To get an overview on how the model changes location in the Complexity-Entropy plane the Lorenz model was simulated with a small timestep and let the simulation run for a sufficiently long time so that the time series was of appropriate length for the highest lag between the datapoints. The model parameters, the initial conditions and the timestep of integration was kept the same as the previous simulation. The length of the time series for the 3D space coordinates was 25 million datapoints, the time series was then resampled with an increasing lag between considered datapoints. The range of resampling lag ran from 1 to 5000, where again, a lag of 1 does not alter the time series. A lag of 5000 means that the resampled time series consists of every 5000 datapoints form the original time series. The Complexity-Entropy analysis was done to each resampled time series and plotted in the same figure to see how the resampling of the time series changes as the lag between datapoints increases.

The resampling of the time series from the 3D axis from the model simulation with the given range of lags produced the Complexity-Entropy plane locations presented in figure 14.

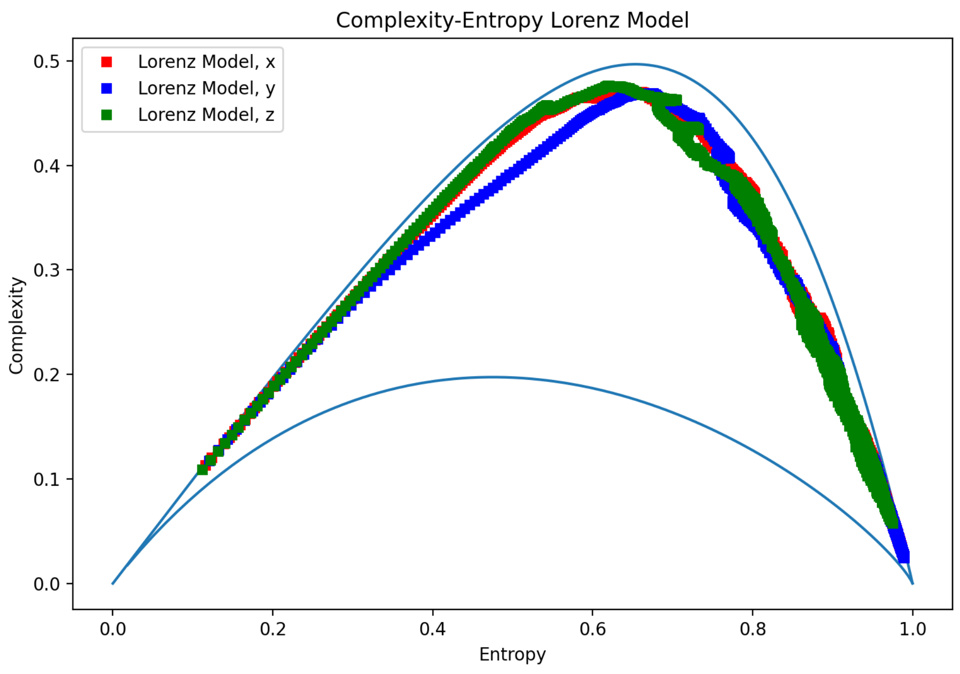


Figure 14: Complexity-Entropy plot from the resampled x, y and z variables of the Lorenz model for different resampling lags. The lags ranges from 1 to 5000 datapoints

From figure 14 the Lorenz model's complexity and entropy measures increase as the lag increases. This seems to be the general behavior, but this behavior is more apparent for lower lags and levels of entropy below $H \sim 0.65$. After which it gets harder to tell, as the entropy and complexity measures changes more erratically with increasing lag. An interesting observation is that the highest levels of entropy achieved by resampling of the time series is comparable to the entropy and complexity measures of the white noise process. This indicates that for high levels of lag in resampling, the behavior of the time series is more like a random number generator and that the time between the resampled datapoints is above the correlation time for the model. Another feature one can deduce from the figure is that the resampled time series from the Lorenz model almost covers the entire range of complexity and entropy. From the unaltered, original time series from the Lorenz model simulations, the resampled time series covers the entire range of entropy close to the lower, right-hand corner of the Complexity-Entropy plane where uncorrelated, random processes belongs.

One other point to mention is the deviation from the maximum complexity line as the entropy increases with increasing lag. From the start the calculated values of complexity and entropy follows the maximum complexity line closely. However, as the entropy level increases the model starts to deviate from the maximum complexity line. The Complexity-Entropy plane locations for the model returns to the maximum complexity line for the highest levels of entropy where the difference between the minimum and maximum complexity line is small.

To show the relation between the lag and calculated entropy and complexity measures more explicitly, the Complexity-Entropy analysis for selected lags is shown. The resampling lag under consideration is $1, 10, 50, 100, 200, 300, 400, 500$.

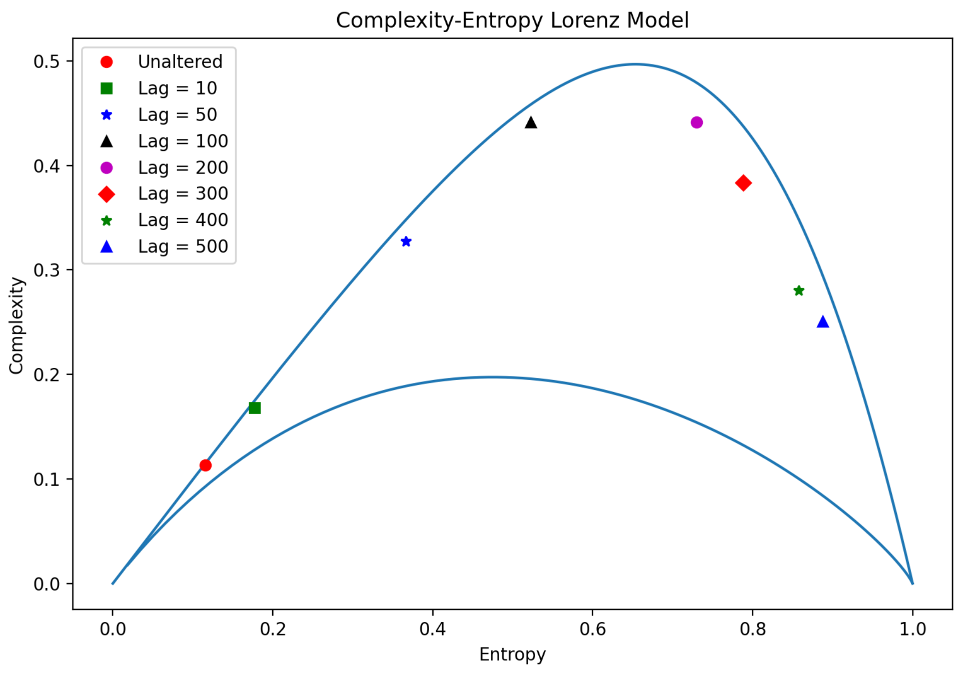


Figure 15: Complexity-Entropy plot of the resampled time series of the Lorenz model of the x variable for selected resampling lags.

Figure 15 supports the notion that the entropy and complexity measures increase as the lag increases. However, the increase of the entropy and complexity measures is not completely one-to-one with increasing lag. This can be seen from figure 14 as the calculated levels on entropy and complexity fluctuates as the lag increases. To show this relation between the entropy and complexity measures and lag more clearly, the entropy and complexity measures is plotted as functions of lag, similar to what was done to the sine-function.

Plotting the entropy and complexity measures as functions of lag produces the following figures, the complexity and entropy measures for the different variables is plotted separately.

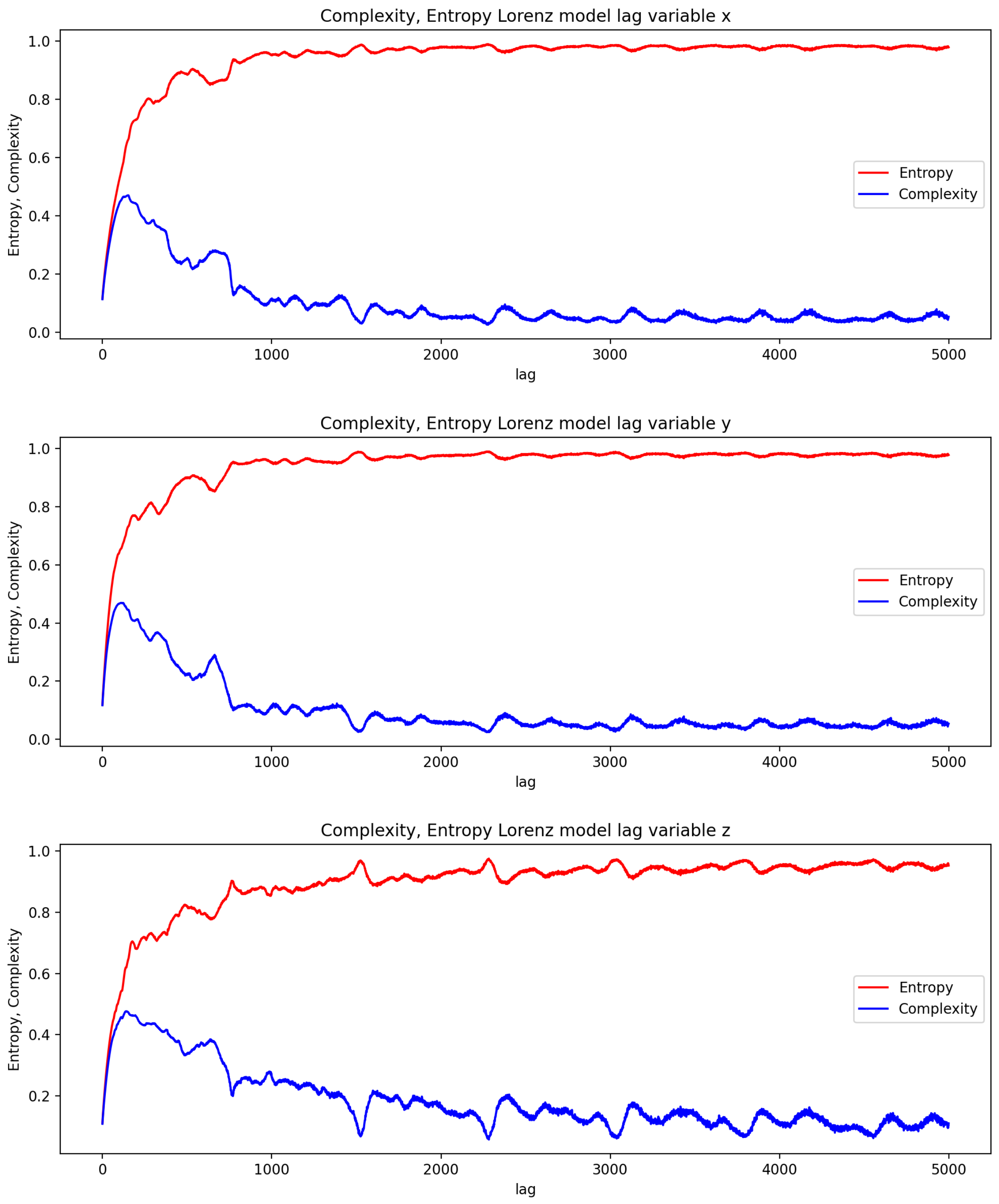


Figure 16: Complexity and Entropy plotted as functions of lag. Separate plots for the x, y and z variables.

Following these figures, one can in general say that the entropy measure increases with lag. The complexity measures in the beginning increases with lag, but as the entropy continues to increase the complexity measure starts to drop in accordance with the behavior of the complexity measure described in the theoretical background, see section 3.3. In the entire range of lags the entropy and complexity measures oscillate. In the beginning, the complexity and entropy measure changes relatively smoothly with increasing lag, but quickly starts to fluctuate as the lag continues to increase. For the complexity measure to comply with the prescribed behavior in the theory section, section 3.3, the complexity measure starts to drop. For levels of lag higher than 1500 the mean value of entropy and complexity seems to remain constant for all directions, while the erratic fluctuations remain.

6.4 Logistic Map

Now, the famous chaotic system the logistic map is considered. This system was made popular by the paper by Robert May published in 1976 [21]. The logistic map is a discrete-time analog to the logistic equation for population growth [20](chapter 10) and given as.

$$x_{n+1} = rx_n(1 - x_n) \tag{13}$$

Here, x_n is a dimensionless measure of a population in generation n and $r > 1$ is the intrinsic growth rate. [20](chapter 10)

This difference equation though requires that the model parameter r is contained in the interval $1 \leq r \leq 4$ so that x_n remains in the interval $0 \leq x_n \leq 1$. If $r < 1$ then all iterations tend towards $x_n \to 0$ as $n \to \infty$ [21] [20](chapter 10).

For a given initial value x_0 and value of the parameter r one can iterate equation (13) and see what happens. As mentioned, for the growth rate $r < 1$ the population goes extinct. For the range $1 < r < 3$ the population grows and will eventually reach a nonzero steady state solution [4] [20](chapter 10). For values $r > 3$ the population will still grow, but the population will oscillate between one high population state and one low population state [20](chapter 10). The logistic map undergoes period doubling, and when the value of x_n repeats every two iterations the oscillation is called a period-2-cycle. As the parameter r continues to increase, the logistic map will eventually oscillate between four different values and obtain a period-4-cycle. This continues, the logistic map undergoes more and more period doubling and reaches a limit at $r = 3.569946\ldots$ beyond which the logistic map has infinite periods and then exhibit chaotic behavior [20](chapter 10). Because of its simplicity and chaotic behavior, Robert May plead that simple models like the logistic map should be studied in universities so that students could develop the mathematical intuition that exotic behavior can be observed from simple nonlinear systems, like the logistic map [21]. A high-resolution image of the bifurcation diagram of the logistic map displaying the period-cycle of for a range of growth rate parameter is given courtesy of Wikipedia[2].

[2] https://en.wikipedia.org/wiki/Logistic_map

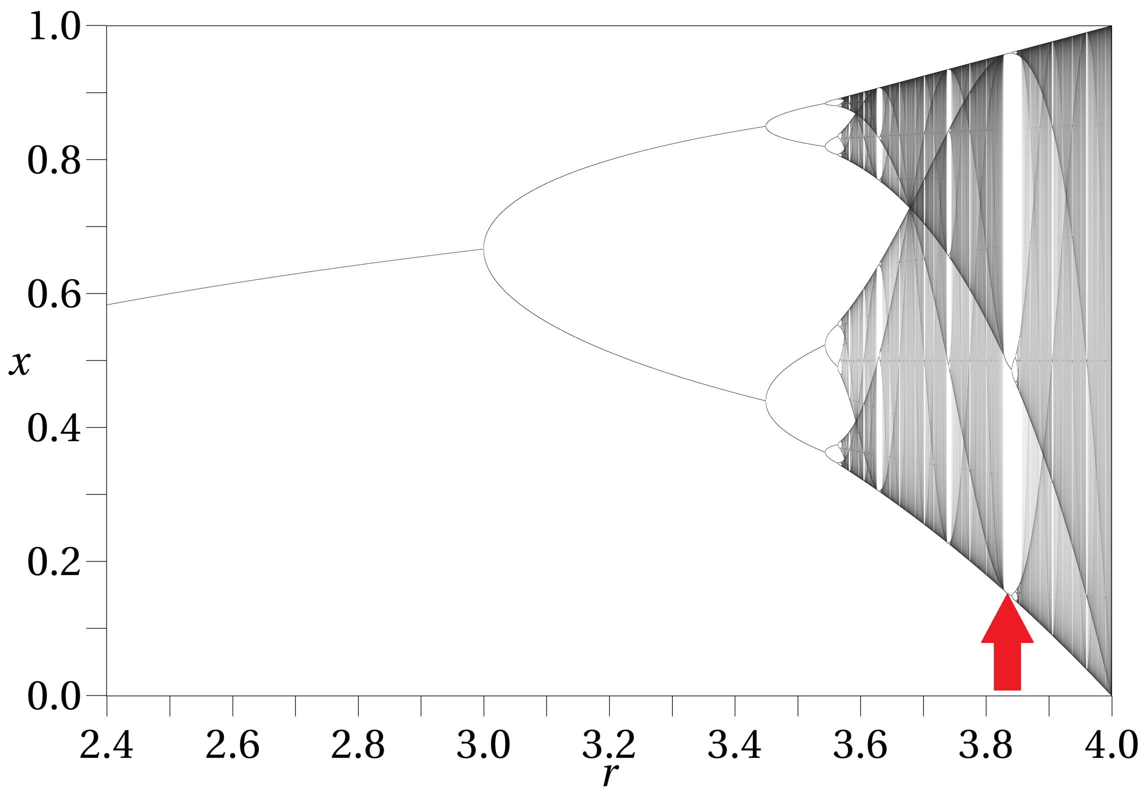


Figure 17: High resolution bifurcation diagram of the Logistic Map, red arrow marks the 3-period-window in the chaotic region of the growth parameter. Bifurcation diagram of the Logistic map courtesy of Wikipedia.

To gain some intuition on how the logistic map behaves for different values of the growth parameter r, the logistic map was simulated and their time series plotted for two values of r. One in the period-2-cycle region for $r = 3.2$ and one in the chaotic region $r = 3.7$. The first 100 values of the Logistic map for the chosen parameter values were plotted.

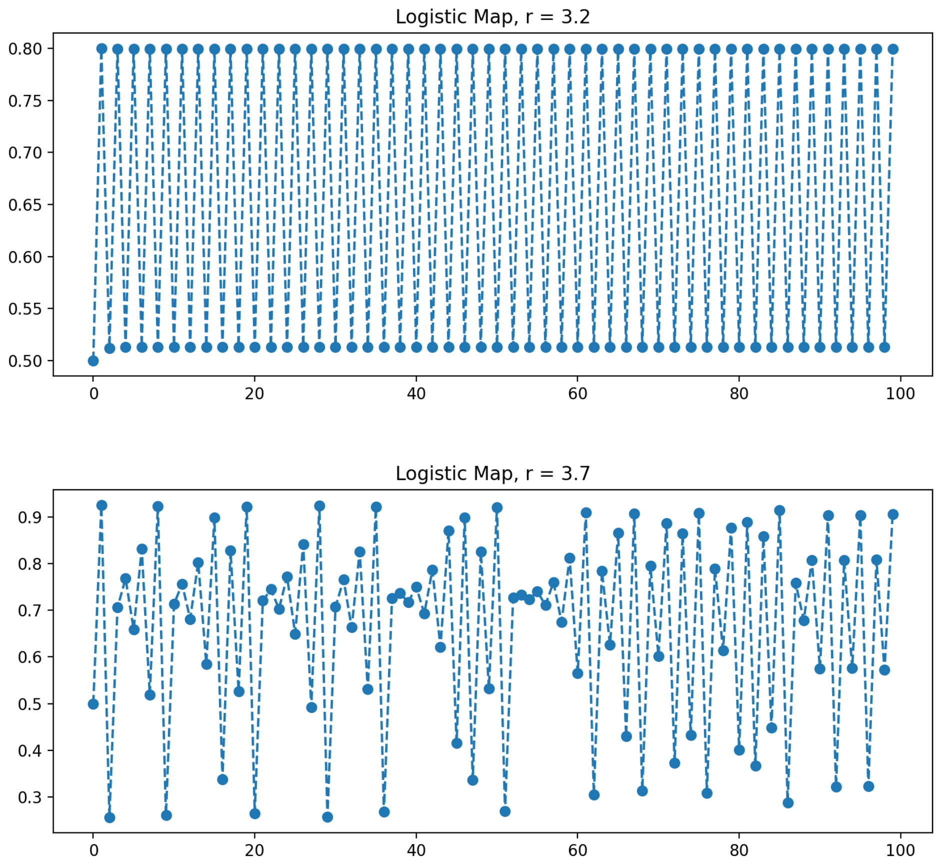


Figure 18: Time series plot from two different Logistic map simulations for growth parameter $r = 3.2$ (top) and $r = 3.7$ (bottom).

Inspecting the top plot in figure 18 one can clearly see the 2 point period of the logistic map characteristic to the growth parameter value. The Logistic map oscillates between 2 values and the oscillation pattern keeps repeating. For the plot from the chaotic value of the growth parameter, the bottom plot, there are no such oscillation between a fixed set of values. Rather there are no set values the model oscillates between. The aperiodic, nonrepeating chaotic nature of the logistic map is clearly visible for this parameter value, $r = 0.37$.

For the Complexity-Entropy analysis the logistic map was simulated with the growth parameter r covering the range $3.5 \leq r \leq 4$ with 1000 increments. Each simulation with a given growth rate parameter a 2^{15} datapoints long time series was generated. The time series generated from the model underwent the Complexity-Entropy analysis and plotted in the same figure. The Complexity-Entropy analysis produced the following plot:

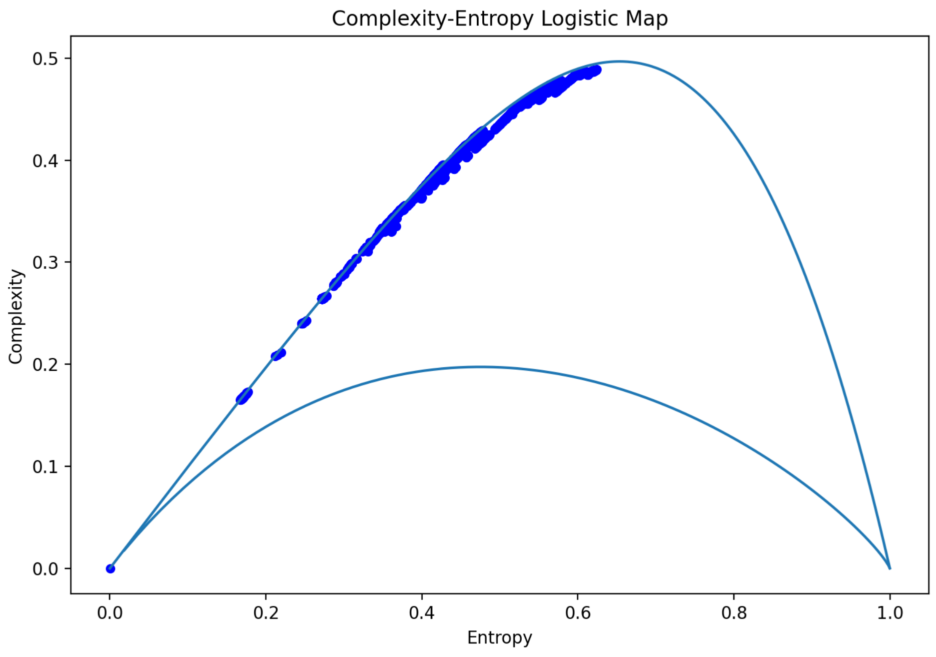


Figure 19: Complexity-Entropy plot for Logistic map realizations with growth parameter in the range $3.5 \leq r \leq 4$

To get a better sense on where the time series produced from value of the growth parameter in the chaotic region of its range, a tiny region of the parameter range in the chaotic region is investigated separately. The Complexity-Entropy analysis of these time series produced the following figure:

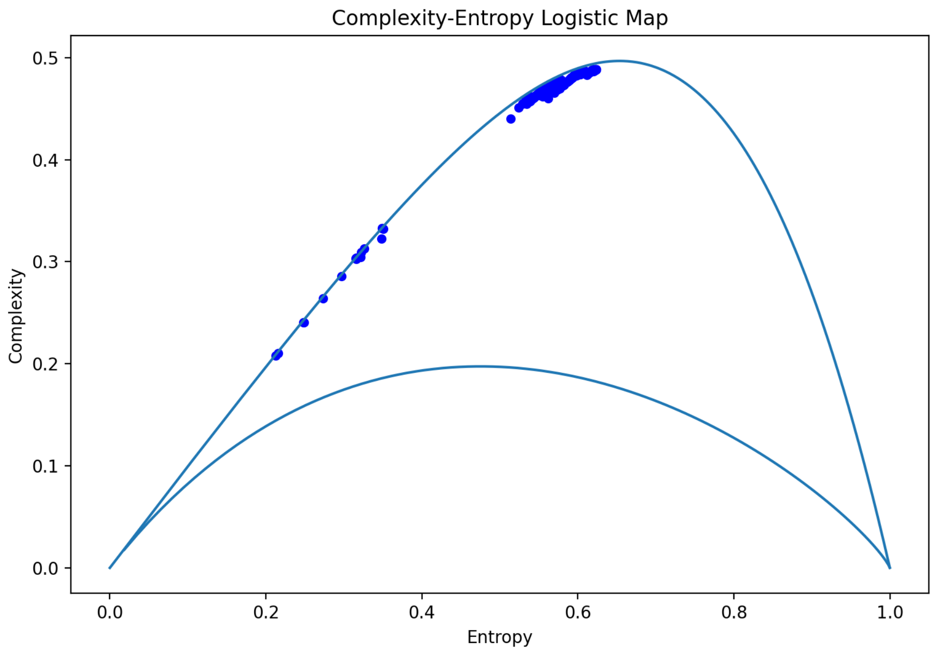


Figure 20: Complexity-Entropy plot of a small window in the chaotic region of the growth parameter. Growth parameter ranged from 3.9 to 4.0 with 500 steps.

From figure 20 one can see that the chaotic range of the growth parameter of the logistic map is in the upper middle part of the Complexity-Entropy plane. Using the logistic map bifurcation diagram, figure 17, looking closely at the region chosen from $r = 3.9$ to $r = 4.0$ there exists thin windows that are more stable and has a finite number of period cycles. Thus, entries with lower entropy and complexity than the chaotic entries in the upper middle of the plot appears.

One more interesting feature of the logistic map is intermittent chaos. Consider the period-3-window marked with a red arrow in figure 17 as an example. As the growth parameter approaches values just on the inside of the period-3-window, the period is visible in the time series simulations of the logistic map. However, the time series also includes bursts of intermittent chaos [20](chapter 10). This behavior of intermittent chaos also appears in other models, like the Lorenz model [20](chapter 10).

6.5 Fractional Brownian Motion and Fractional Gaussian Noise

Fractional Brownian motion are a family of Gaussian random functions defined as a moving average of $dB(t)$, where $B(t)$ represent ordinary Brownian motion [22]. Fractional Brownian motion are the only family of such processes which are [2, 22]:

- Gaussian
- Self-similar
- Its increments are stationary

Fractional Brownian motion are a continuous time Gaussian process $[B^{\mathcal{H}}(t), t > 0]$ which starts of at zero ($B^{\mathcal{H}}(0) = 0$), has zero mean $E[B^{\mathcal{H}}(t) = 0]$, and its covariance function is given as [2]

$$E[B^{\mathcal{H}}(t)B^{\mathcal{H}}(s)] = (t^{2\mathcal{H}} + s^{2\mathcal{H}} - |t-s|^{2\mathcal{H}})/2 \tag{14}$$

Here, $E[\cdot]$ denotes the expectation value, the exponent $\mathcal{H}$ is the Hurst exponent and has valid range $0 < \mathcal{H} < 1$. For $\mathcal{H} = \frac{1}{2}$ the fractional Brownian motion corresponds to classical Brownian motion where each consecutive increment of the process are just as likely to have the same sign as the opposite sign, i.e. the increments are uncorrelated [2]. For $\mathcal{H} < \frac{1}{2}$ the consecutive increments of the process are more likely to have the opposite sign, the increments are here anti-persistent and thus negatively correlated. For $\mathcal{H} > \frac{1}{2}$ the consecutive increments are more likely to have the same sign, the increments here are persistent and thus positively correlated.

The increments of the fractional Brownian motion:

$$W^{\mathcal{H}}(t) = B^{\mathcal{H}}(t+1) - B^{\mathcal{H}}(t) \tag{15}$$

are a fractional Gaussian noise process [2]. The autocorrelation function for the fractional Gaussian noise process is given as:

$$\rho(k) = E[W^{\mathcal{H}}(t)W^{\mathcal{H}}(t+k)] = \frac{1}{2}[(k+1)^{2\mathcal{H}} - 2k^{2\mathcal{H}} + |k-1|^{2\mathcal{H}}] \tag{16}$$

For $\mathcal{H} = \frac{1}{2}$ one can see that for non-zero lag, $k \neq 0$, all correlation vanish and therefore the fractional Gaussian noise represents "normal" white noise.

The fractional Brownian motion, and fractional Gaussian noise processes were all simulated using a package *FBM* from Python. The Fractional Brownian motion process is not stationary, and thus do not comply with the requirement to be applicable to the Complexity-Entropy analysis, where the requirement is that the time series under analysis must be weakly stationary, see section 3.1. However, because of their Gaussian nature, and stationary increments, the Complexity-Entropy analysis can still be applied to time series from these processes [2, 23]. For a more in-depth explanation as to why, see the paper by Bandt and Shiha [23].

To get a better sense on how the fractional Brownian motion and fractional Gaussian noise processes depend on the Hurst exponent, two simulations was done for both fractional Brownian motion and fractional Gaussian noise at two different values for the Hurst exponent. The time series of these simulations is presented in figure 21.

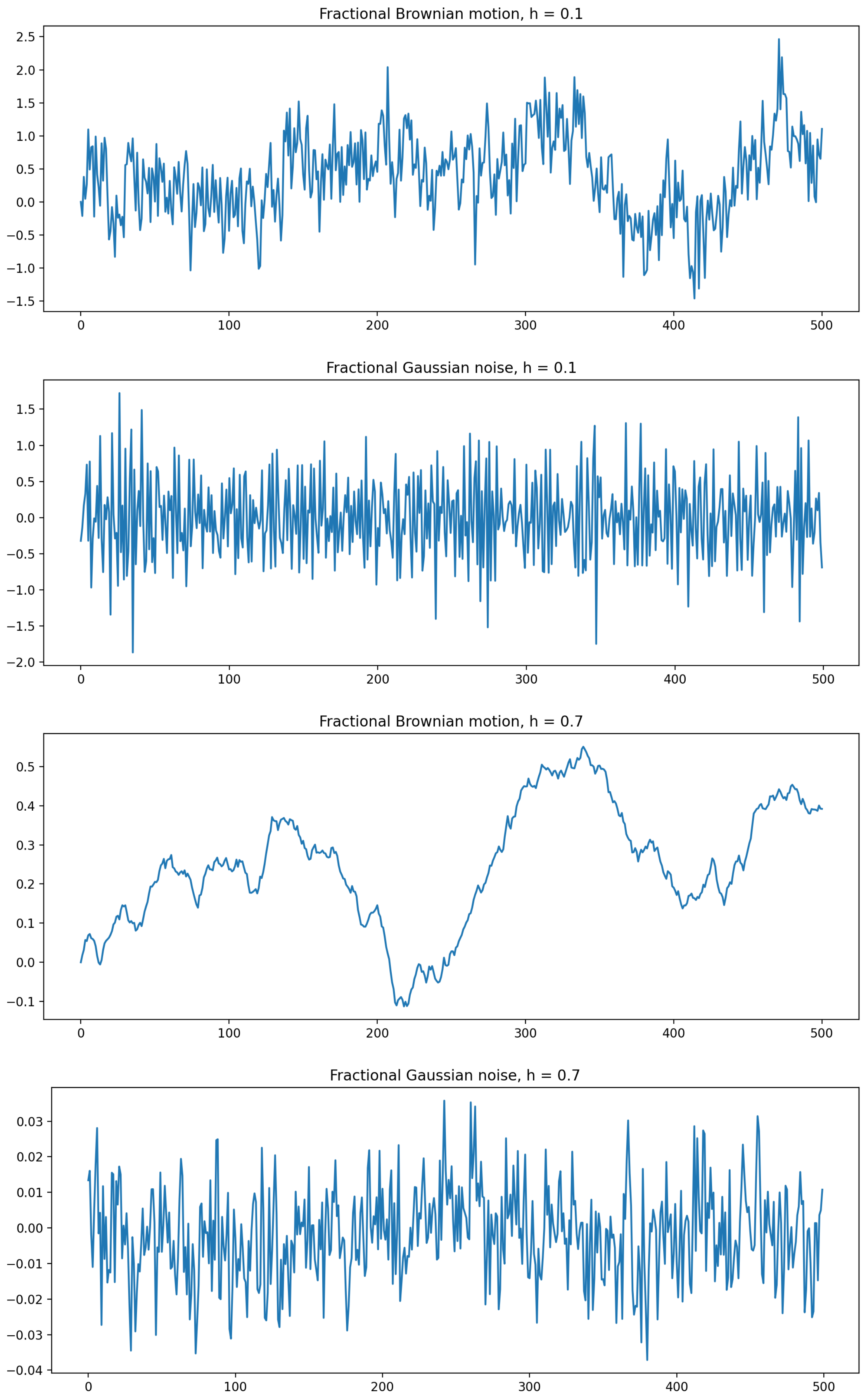


Figure 21: The time series plot of both fractional Brownian motion and fractional Gaussian noise process. The top two plots for $\mathcal{H} = 0.1$. The bottom two plots for $\mathcal{H} = 0.7$

Looking at the plots in figure 21 one can really see the anti-persistent and persistent nature of the increments for the Hurst exponents $\mathcal{H} = 0.1$ and $\mathcal{H} = 0.7$ in the fractional Gaussian noise plots. This is also reflected in the cumulative sum of the fractional Gaussian noise into the fractional Brownian motion. The fractional Brownian motion of the low Hurst exponent case is not much different to a noise process because of the anti-persistent increments. The high Hurst exponent case where the increments are persistent results in significant trends in the time series.

For the Complexity-Entropy analysis the processes were simulated by empirically determined values for the Hurst exponent given by $\mathcal{H} = [0.001, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.84, 0.88, 0.90, 0.905, 0.91, 0.915, 0.92]$. From testing, it showed that selecting these values was a good compromise between enough values to make the final plot look smooth, and the computational cost of adding more values of the Hurst exponent. The time series from each simulation was $2^{19} = 524288$ datapoints long. The simulation of all Hurst exponents was done 100 times. For all simulations, the entropy and complexity measure were calculated and the average of all entropy and complexity values of all 100 simulations for each Hurst exponent was plotted in figure 22.

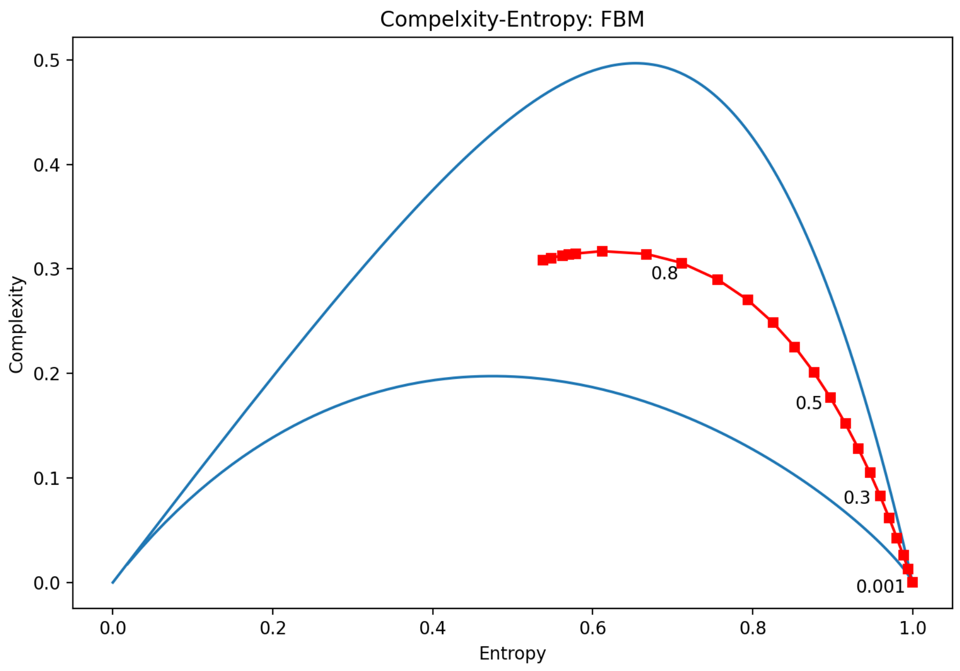


Figure 22: Complexity-Entropy plot of fractional Brownian motion (FBM) process for different Hurst exponents with the location for some selected Hurst exponents marked on the figure.

The equivalent plot for fractional Gaussian noise, but only the Complexity-Entropy plane location for $\mathcal{H} = [0.001, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]$ is plotted in figure 23. The rest of the Hurst exponents are more or less the same. The only difference is as the Hurst exponent increases beyond $\mathcal{H} = 0.5$, the complexity and entropy measures back up a line close to the Complexity-Entropy line drawn for $\mathcal{H} \in [0.001, 0.5]$ but slightly closer to the fractional Brownian motion line.

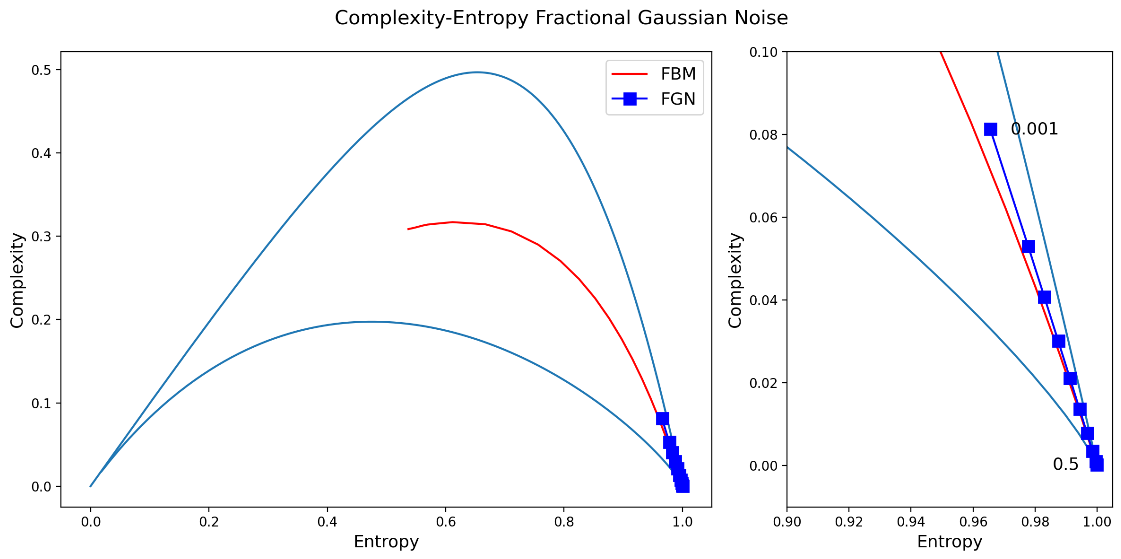


Figure 23: Complexity-Entropy plot of fractional Gaussian noise (FGN) process for different Hurst exponents.

Considering figure 22 one can see that the simulations of the fractional Brownian motion for the different Hurst exponents obtains complexity and entropy measures that lie neatly on a line. The lowest Hurst exponent for the fractional Brownian motion lie in the lower right corner and is similar to the white noise process, as the Hurst exponent increases in size the entropy measure is lowered. As the entropy measure is decreasing, the complexity measure of the fractional Brownian motion increasingly deviates from the maximum complexity line. The fractional Gaussian noise process also span a nice curve in the Complexity-Entropy plane. However, the highest complexity, lowest entropy measure happens for the lowest Hurst exponent. As the Hurst exponent increases, the complexity measure drops and the entropy measure increases until Hurst exponent $\mathcal{H} = 0.5$ where the entropy measure is maximized and the complexity measure vanishes. For Hurst exponent $\mathcal{H} = 0.5$ the fracional Gaussian noise represents white noise which one can see in figure 23. The fractional Brownian motion plot, figure 22, resembles more that which is presented in the literature, see [2, 3, 5, 6].

A thing to note here is that the simulations done to produce figure 22 was not fully consistent. The length requirement for the time series becomes large when the Hurst exponent comes closer to 1, like for $\mathcal{H} = 0.92$ which meant that the simulation length of the time series of necessity had to be larger than 2^{17} datapoints. So, to make the simulations more consistent, the length of the time series was set to 2^{19} datapoints long. Even this made little difference to the consistency between each simulation as there was still big differences in calculated entropy and complexity values from one simulation to the next. This led to the averaging of the entropy and complexity measures of each simulation, for each Hurst exponent the fractional Brownian motion process was simulated 100 times and the average of the entropy and complexity values were calculated. This resulted in consistency between each simulation at the cost of extremely long computation times. Only small differences in the entropy and complexity values for the very largest values of Hurst exponents are noticeable for different simulations.

One more thing worth investigating is the effect of the trend in the time series, and how much it effects the location of the process in the Complexity-Entropy plane. As seen earlier, a process dominated by trends like for the linear model in section 6.1 ended up in the lower left corner of the Complexity-Entropy plane.

To achieve this, one removes a running mean from the time series with different window lengths centered around the datapoint in question. This can be represented with equation (17) for some window length n for each datapoint x_t for $\forall t \in [0, T]$ in the time series $X_t = \{x_t\}_0^T$ changes its value according to the following equation:

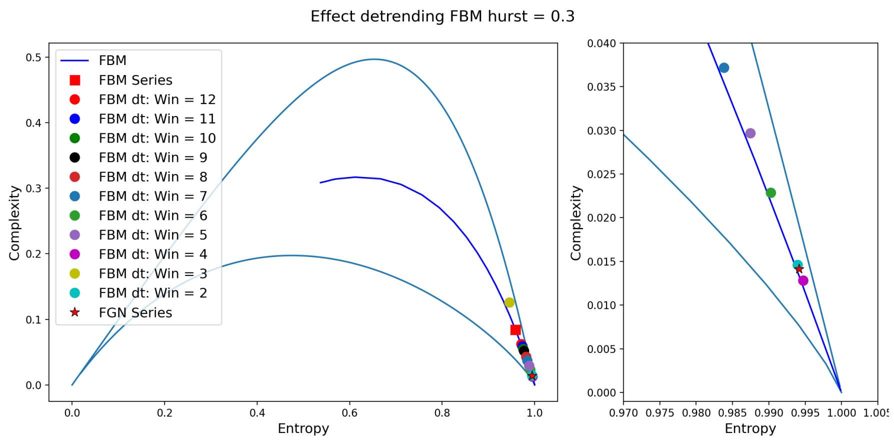
$$x_t = x_t - \frac{1}{n}(x_{n-2} + x_{n-3} + \ldots + x_{t-1} + x_t + x_{t+1} + \ldots + x_{t+(n-3)} + x_{t+(n-2)}) \tag{17}$$

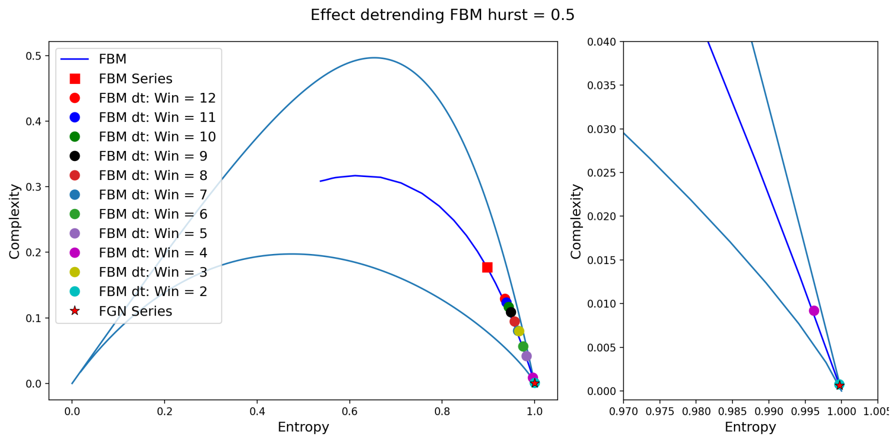
It is worth noting that for window of even length the considered datapoints in the window are:

$$\begin{aligned} n =&2 \qquad x_{t-1}, x_t \\ n =&4 \qquad x_{t-2}, x_{t-1}, x_t, x_{t+1} \\ n =&6 \qquad x_{t-3}, x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2} \\ &\ldots \end{aligned}$$

always adding to the window on the left side. For odd number window lengths, the procedure is straight forward. The window is perfectly centered around the datapoint x_t.

The window lengths under consideration was $[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$. Here the length of the fractional Brownian motion simulation was set to 200000 datapoints and for the selected Hurst exponents $\mathcal{H} = [0.3, 0.4, 0.5, 0.6, 0.7]$ the time series were detrended by subtracting the running mean with each of the window sizes. The Complexity-Entropy analysis of each of the time series from the fractional Brownian motion simulations, and the detrended time series using all window sizes was done. As a guide, the Complexity-Entropy analysis of the fractional Gaussian noise process for the corresponding Hurst exponent was also included. To get a better view of what happens in the lower right corner, a zoomed in version of the figure was also included and is presented in the following figure.





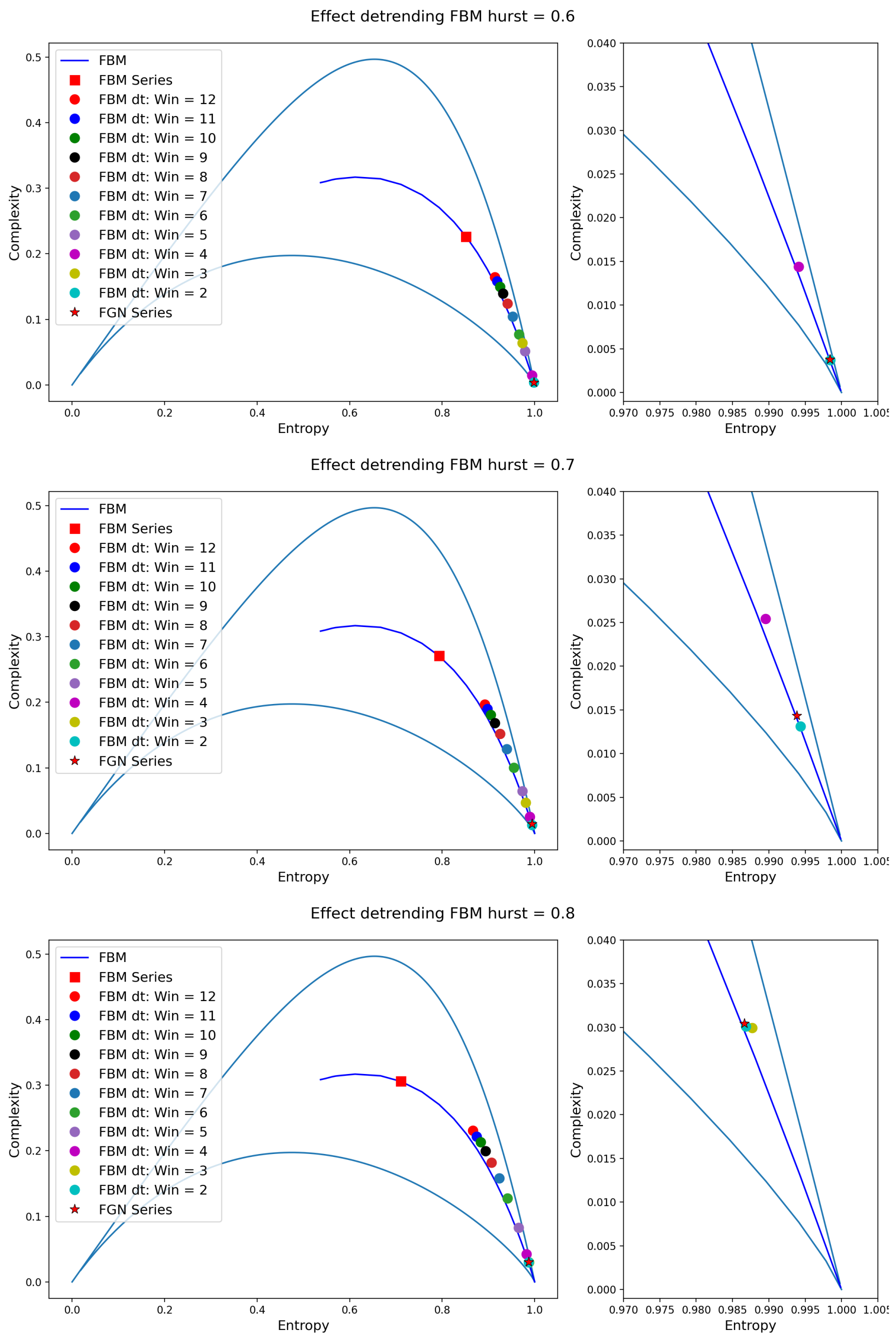


Figure 24: Effect of detrending fractional Brownian motion (FBM) processes by subtracting a running mean with different window sizes. The original time series of the FBM process is marked with a red square. The corresponding fractional Gaussian noise process is marked with a red star. Detrended FBM process time series is marked with circles.

From figure 24 it is clear that the detrending increases the entropy of the time series, and the smaller the window size when subtracting the running mean, the more the entropy increases. One sees that for a window size of 2 the detrended fractional Brownian motion process occupies a place very close to the fractional Gaussian noise process in the Complexity-Entropy plane, which makes sense. For a window size that small when a running mean is subtracted from the time series it comes close to detrending using a first difference approach

$$\Delta x_t = x_t - x_{t-1} \tag{18}$$

where the previous element in the time series is subtracted from the current. When subtracting the running mean, however, the average of the current and previous element is subtracted from the current element. When looking at the equation for the fractional Gaussian noise process, equation (15), the fractional Gaussian noise process is just the increments in a fractional Brownian motion process. Or said in a way that will make sense later, the fractional Brownian motion process is nothing but the cumulative sum of its increments defined by the fractional Gaussian noise process of the same Hurst exponent. When detrending the fractional Brownian motion process using the running mean approach with a window size of 2, which in some sense is approximately the same as the first difference approach, the detrended process comes close to that of the fractional Gaussian noise process in the Complexity-Entropy plane. Detrending with a first-difference method one obtains the increments of the process. For the fractional Brownian motion process this means that one obtains the fractional Gaussian noise process. It therefore makes perfect sense that the detrended fractional Brownian motion process with window 2 and the fractional Gaussian noise process of the same Hurst exponent are close in the Complexity-Entropy plane for all Hurst exponents.

One detrending window size that stick out is for the window size of 3. In the plot for Hurst exponent $\mathcal{H} = 0.3$ it reaches lower levels of entropy and higher levels of complexity than every other window size, even higher than the original fractional Brownian motion time series. For the other Hurst exponents, the window size of 3 still obtained levels of entropy and complexity which differ for the position one thinks it should have based on the relative position of the other detrending window sizes. But for higher Hurst exponents the position of the detrended time series in the Complexity-Entropy plane comes closer to the relative position based on the positions of the other window sizes, see plot for Hurst exponent $\mathcal{H} = 0.7$. Why this is the case that detrending for window size 3 leads to very different locations in the Complexity-Entropy plane than expected remains unknown for the time being.

The positions of the detrended time series are in close proximity of the fractional Brownian motion line and they move close to the line for decreasing window size. The detrending of the time series leads to an increase in entropy levels and a subsequent decrease in complexity levels. This observation holds for all Hurst exponents and window sizes, when ignoring window size of 3 which will be commented on later. The general trends is that for decreasing window size, the calculated entropy level for the detrended time series increases. The increase in entropy and decrease of complexity happens until for a window size of 2 where the entropy and complexity levels approach that of the corresponding fractional Gaussian noise process, which has no trend.

Now for the special case for window size 3. For the selected Hurst exponents, the detrending using a window if size 3 does not lie where it should be in the Complexity-Entropy plane. One would think, looking at the other detrending window sizes that detrending with window 3 would result in a location in the Complexity-Entropy plane in-between the windows of sizes 2 and 4. However, this is not the case. For increasing Hurst exponents, the Complexity-Entropy plane location of the detrended time series approaches its perceived location, where the closest to its expected location is for $\mathcal{H} = 0.8$. This indicates that there is something weird happening to the fractional Brownian motion process when its time series is detrended using a window of size 3.

This weird behavior of the detrending using a window of size 3 had to be investigated further. To get a better understanding of the detrending process the running mean of each of the time series was calculated using the same window sizes used for the detrending of the time series in figure 24 and these averaged time series was then using in the Complexity-Entropy analysis. This was done to investigate if the averaged time series also showed the same weird behavior as the detrended time series. The Complexity-Entropy analysis of the averaged time series form the fractional Brownian motion processes with the same hurst exponents and window sizes as in figure 24. This analysis produced the following result.

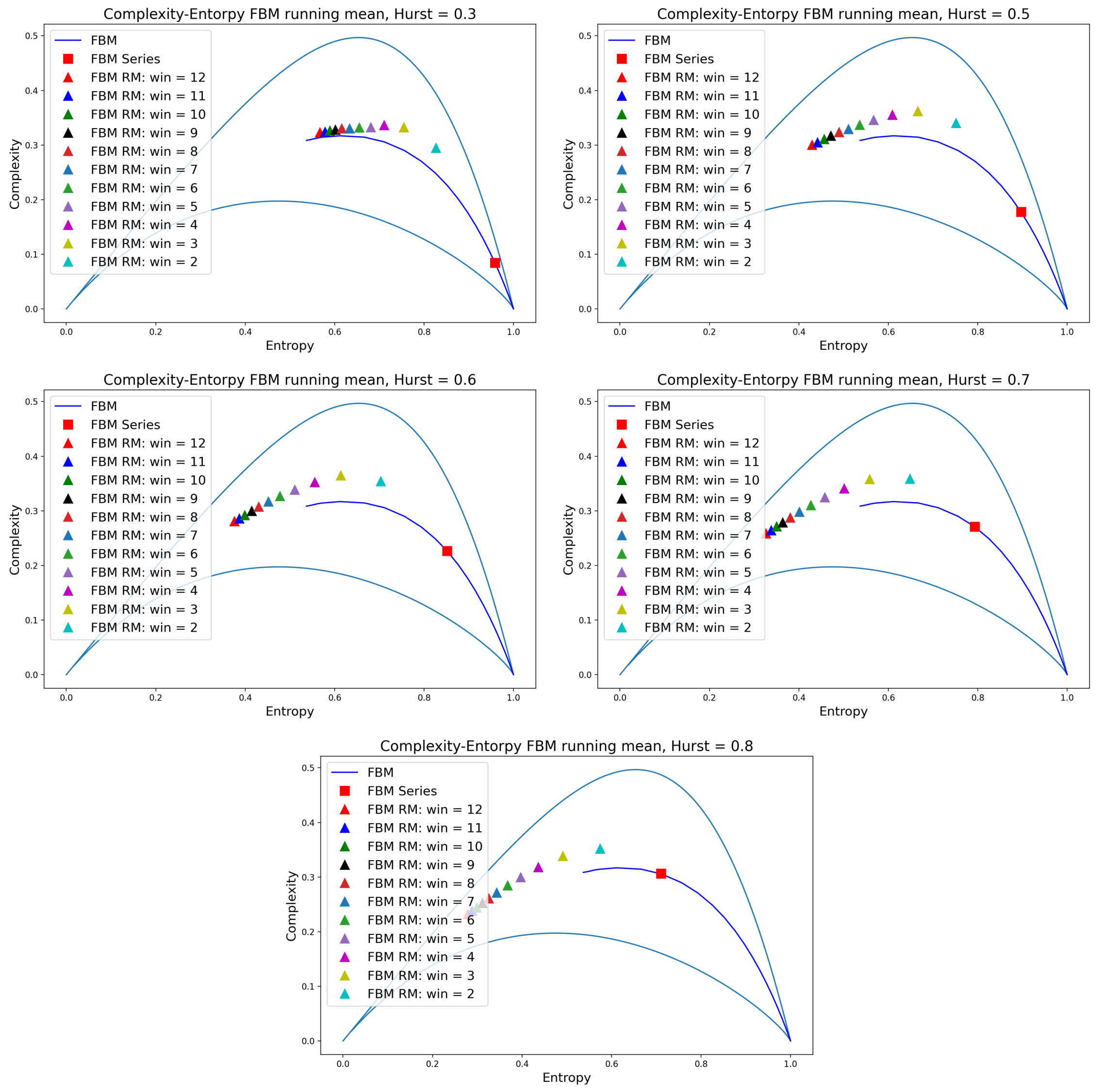


Figure 25: Complexity-Entropy analysis of the running averaged time series used in detrending the original time series of the fractional Brownian motion for different Hurst exponents

Looking at these results in figure 25 there is no indication of odd behavior of any window size, let alone window of size 3. The trends set of the location of the time series as the window size decreases is followed by all windows, except for some odd locations for the smallest window sizes. This gets less noticeable as the Hurst exponent increases. But that behavior is still mostly visible for window size 2, yet the location in the Complexity-Entropy plane of the detrended process for this window, presented in figure 24, does not follow that of window 3 and appear where it is expected to be. Thus, this was a dead end and provided no solid explanation for the odd behavior for the detrending window of size 3.

Another possible explanation for the odd behavior for window 3 was to investigate the algorithms to generate the fractional Brownian motion time series used in the package "FBM". The special case for Hurst exponent $\mathcal{H} = \frac{1}{2}$ was selected. For this Hurst exponent, the fractional Brownian motion is just classical Brownian motion. The increments of this process, the fractional Gaussian noise process, represents white noise. A different implementation of the running mean was also used to check if the running mean function originally used included in the Pandas package in python had some inaccuracies. Although the latter is not likely to be the reason for the behavior, as the *Pandas* package is a widely used package for time series analysis. If the running mean function included in the *Pandas* package had some inaccuracies in it, those would likely be discovered and delt with already.

Using the *FBM* package, a fractional Brownian motion process was simulated for $\mathcal{H} = \frac{1}{2}$. Using the same package, a fractional Gaussian noise process was also simulated with the same Hurst exponent. The cumulative sum of the elements in the time series produces was calculated, giving the corresponding fractional Brownian motion process. Independently, a white noise process was also simulated. This process would theoretically represent a fractional Gaussian noise process with Hurst exponent $\mathcal{H} = \frac{1}{2}$. The cumulative sum of this time series was calculated, and the resulting time series would then represent a fractional Brownian motion process with $\mathcal{H} = \frac{1}{2}$. This gives three different, but equivalent, fractional Brownian motion processes which then could be detrended and plotted in the Complexity-Entropy plane.

Each of these time series was detrended using both methods, using the running mean from Pandas and the running mean function from UiT Cosmo's uit-scripts repository on GitHub. To see if it somehow made a difference which functions are used for detrending the time series, the simulation of the normal fractional Brownian motion and the detrended version from both implementations was plotted in figure 26.

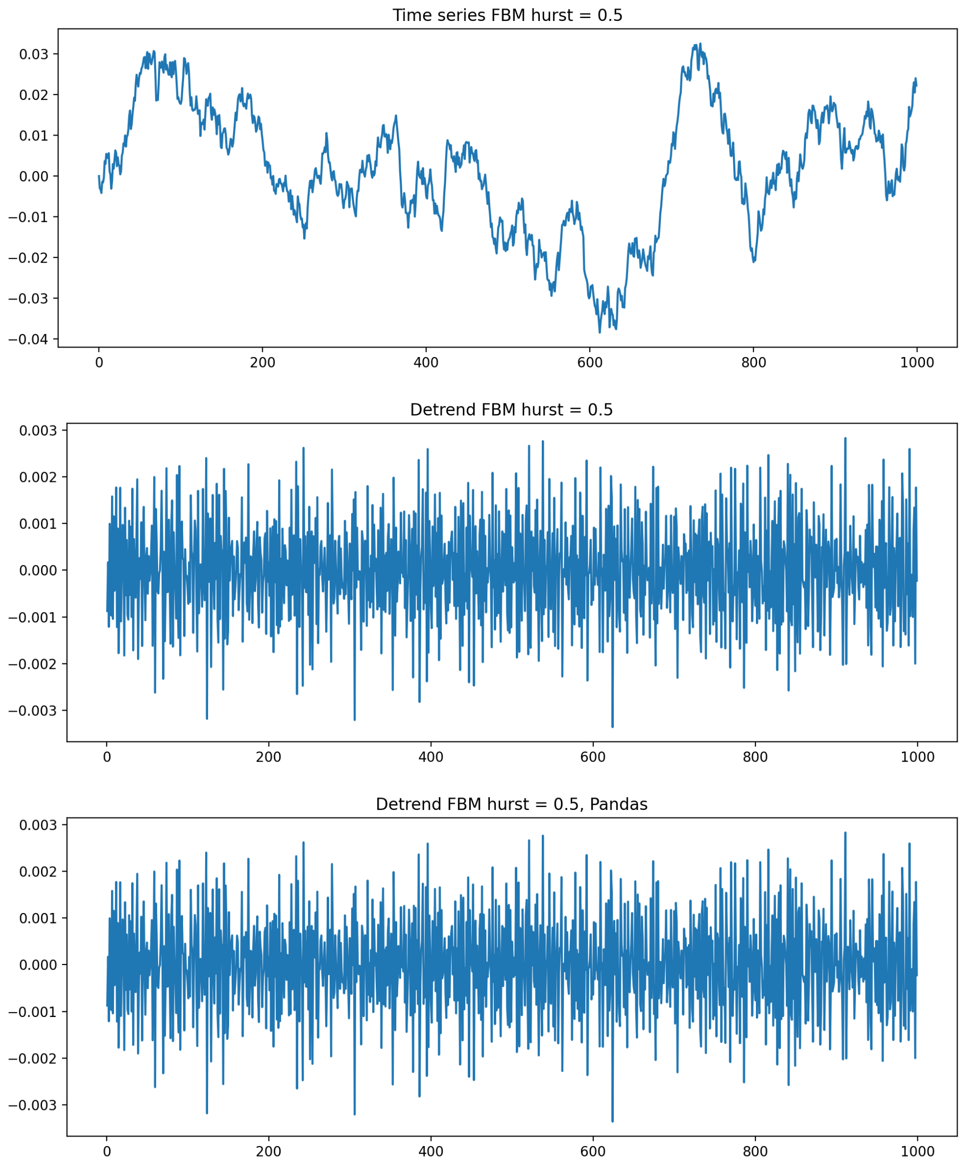


Figure 26: Time series plot of fractional Brownian motion for $\mathcal{H} = \frac{1}{2}$ and detrended time series plot using uit-scripts' running mean (middle) and Pandas' running mean (bottom) functions.

From figure 26 it is clear that there is no difference between the two methods for detrending the time series as the two detrended time series are identical. For further proof that there is no difference between them, the Complexity-Entropy analysis was applied to the three equivalent fractional Brownian motion simulations described earlier, and both methods for detrending was applied to each time series. The Complexity-Entropy analysis produced the following plots in figure 27.

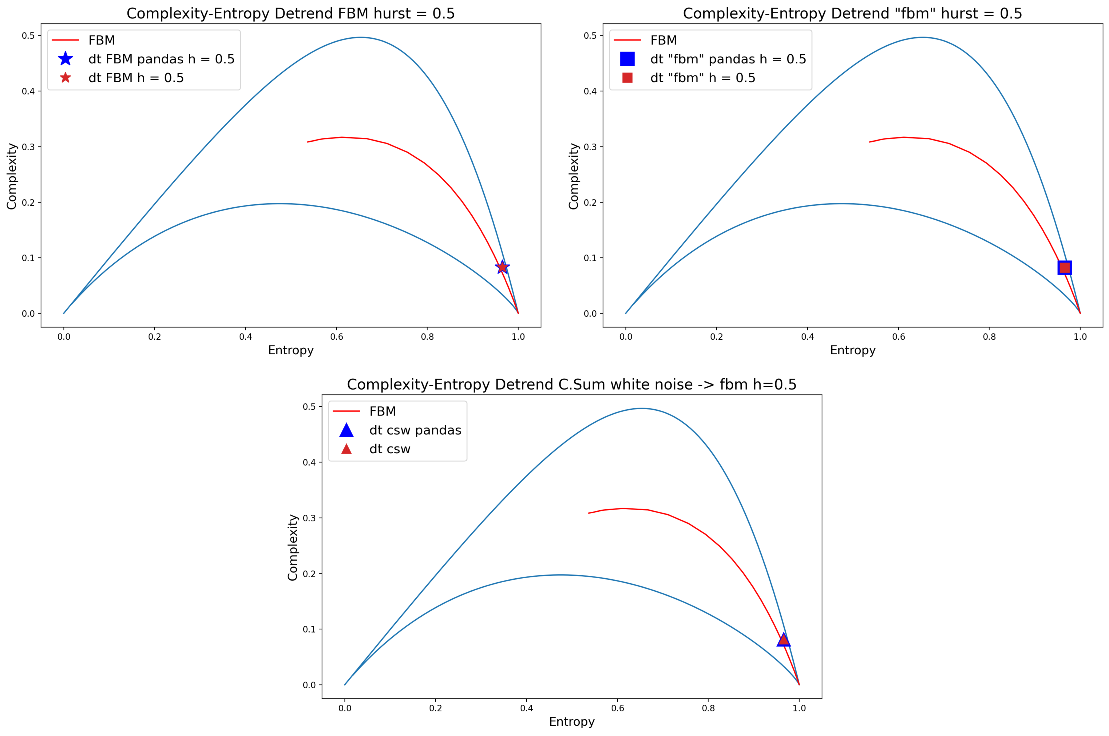


Figure 27: Complexity-Entropy plot of the equivalent realizations of fractional Brownian motion process for $\mathcal{H} = \frac{1}{2}$, detrended using uit-scripts and Pandas running mean function

Comparing the results from figure 27 with the one produced earlier, figure 24, one finds that this analysis resulted in the same Complexity-Entropy plane location as before. This indicates that the problem does not lie in the algorithms producing the fractional Brownian motion either. Also, since the running mean functions in the Pandas package and the uit-scripts repository produced identical detrended time series means that the problem does not lie there. Therefore, the initial investigation of the weird behavior of the detrended fractional Brownian motion process was unfruitful. The investigation revealed no explanation for the odd location of the detrended fractional Brownian motion process using the particular running mean window of size 3. For the time being, the reasons and mechanisms behind these results remains unknown.

Returning to figure 24 the main point to take away is that the trends in the time series does affect the Complexity-Entropy plane location of the model. By calculating the moving average of the time series with an increasing window size one removes the fluctuations of the time series and one is left with the trends in the time series. This can be seen in figure 25 as the largest window size, window of size 12, gave the lowest measures of entropy. Detrending the original time series by subtracting a moving average then removed the trends in the time series which lead to an increase in entropy measure, and a subsequent drop in the complexity measure in compliance with the behavior set for the complexity measure. This is also deducible without applying the detrending to the time series. As the lower Hurst exponents fractional Brownian motion simulations has less noticeable trends in the time series, see figure 21, but they also achieve the highest entropy measures in the Complexity-Entropy plane, see figure 22.

7 The Filtered Poisson Process (FPP)

As mentioned in section 1, the fluctuations in magnetically confined plasmas has been shown to be chaotic, and papers has shown the chaotic nature in the fluctuations of the temperature from electron heat transport [5] and edge density from particle transport [3]. Investigations has shown that the turbulence in the scrape-off-layer (SOL) of tokamaks fluctuates intermittently with blob-like structures which leads to non-diffusive transport of particles in the plasma [3]. These blob-like structures move radially outward, meaning there is cross-field transport of particles and heat which leads to an increase in plasma-wall interactions [8]. Single point measurements have been done and showed that the blob movements lead to large amplitude bursts with asymmetric pulse shape with a fast rise and slow decay [8]. Based on numerical simulations and experimental measurements it has been suggested that the plasma fluctuations can be represented by a random sequence of bursts [8]. A stochastic model for the plasma fluctuations has been developed as a superposition of uncorrelated pulses, and has been shown to favorably agree with experimental measurements [24]. This stochastic model is given by the equation:

$$\Phi_K(t) = \sum_{k=1}^{K(T)} A_k \phi\left(\frac{t-t_k}{\tau_k}\right) \tag{19}$$

The sum runs over a random sequence of $K(T)$ pulses in the given time interval $t \in [0,T]$. Each pulse is denoted as k and are characterized by an amplitude A_k, a pulse arrival time t_k, and a pulse duration time τ_k. The pulse shape, ϕ, in the stochastic model equation (19) is assumed constant for all pulses. The arrival times of the pulses are assumed to follow a uniform probability distribution on the time interval of the process $t \in [0,T]$. This makes the probability density function for the arrival times of the pulses $P_t(t_k) = \frac{1}{T}$. In general the pulse duration is randomly distributed with a probability density function denoted as $P_\tau(\tau)$. The average pulse duration is then defined to be [24]:

$$\tau_d = \langle \tau \rangle = \int_0^\infty d\tau \, \tau P_\tau(\tau) \tag{20}$$

The number of pulses $K(T)$ is also a random variable and assumed to follow a Poisson distribution [24]. The conditional probability that exactly K pulses arrives in a time interval of length T is then given as [24]:

$$P_K(K|T) = \frac{1}{K!}\left(\frac{T}{\tau_w}\right)^K \exp\left(-\frac{T}{\tau_w}\right) \tag{21}$$

Here, τ_w is the average waiting time for the pulses. For a Poisson distribution, it follows that the waiting time between two consecutive events, or pulses in this case, are exponentially distributed. The average number of pulses in the time interval $t \in [0,T]$ is given by [24]

$$\langle K \rangle = \sum_{K=0}^{\infty} K P_K(K|T) = \frac{T}{\tau_w} \tag{22}$$

Here, the angular brackets denote the average of the random variable over all its arguments.

By averaging over all random variables, the lowest order moments of the model can be calculated in a rather straight forward manner. The lowest order moment, the mean value, of the model has been calculated and is given as:

$$\langle \Phi \rangle = \frac{\tau_d}{\tau_w} I_1 \langle A \rangle \tag{23}$$

Here, τ_w is the average waiting time for the pulses and is defined as $\tau_w = \frac{T}{\langle K \rangle}$. I_1 is the integral of the pulse shape ϕ to the first power. The pulse shape is normalized so that [24]

$$\int_{-\infty}^{\infty} |\phi(\theta)| \, d\theta = 1 \tag{24}$$

Meaning for constant pulse duration, and with the assumption that the pulse shape is the same for all pulses, each pulse contributes equally to the mean value of $\Phi_K(t)$. The integral of the pulse shape can be generalized to be of any power, meaning:

$$I_n = \int_{-\infty}^{\infty} [\phi(\theta)]^n \, d\theta \tag{25}$$

Integrals of the pulse shape of higher order $n > 1$ appears for higher order moments of the stochastic model, like variance etc.
From the equation of the mean value of the stochastic process, a fraction appears that is of great interest. This fraction is defined to be the intermittency parameter γ [24]

$$\gamma = \frac{\tau_d}{\tau_w} \tag{26}$$

This parameter determines the amount of pulse overlap in the process. For low values of γ, the pulses will in general appear separately leading to low levels of pulse overlap and a strongly intermittent signal. While for high values of γ, individual pulses may be hard to detect as there is high levels of pulse overlap and the process may appear like random noise. In fact, it has been demonstrated that for high value of γ the random variable Φ_K will approach a normal distribution as $\gamma \to \infty$ [8, 24, 25]. This will happen independent of the pulse shape and amplitude distribution and is consistent with the central limit theorem [8, 25]. The effect of the intermittency parameter will be demonstrated by simulating the stochastic model for different value of γ.

The moments of the stochastic process [8, 25], auto-correlation function [24, 25], frequency power spectrum [24], level crossing [25, 26] and excess time statistics [26] have been discussed with various model configurations in other papers and are well-known. The results of which agree favorably with experimental data and findings.

To observe the effect of different intermittency parameters, the model was simulated for different values of γ. The chosen values were $\gamma = [0.1, 1, 10, 100]$. After removing the transient period of the signal, where the signal builds up and eventually oscillate around the mean value of the signal, the first 5000 datapoints of the signal was plotted. When simulating the model, it is for the moment assumed that the duration time of the pulses are the same. See section 8 for a discussion on when the probability distribution of the duration time of the pulses follows a power law. The amplitudes follow an exponential distribution. The number of pulses in the realization of the model, K, is a simulation parameter and therefore known. Another simulation parameter is the discretization timestep, Δt. The discretization timestep in the simulations are normalized to the duration time of the pulses. For this simulation the value $\Delta t = 0.01$ was used, and the pulse shape was chosen to be a one-sided exponential pulse. For the following, the time series from the model simulation was rescaled to have zero mean and unit standard deviation.

$$\Phi \quad \to \quad \tilde{\Phi} = \frac{\Phi - \langle\Phi\rangle}{\Phi_{\text{rms}}} \tag{27}$$

Here, $\langle\Phi\rangle$ is the mean value of the signal, and Φ_{rms} is the root mean square value of the signal.

The simulation of the model was done using the “uit-scripts” code which is available on GitHub[3].

To have some control over the length of the time series, and to make sure the time series lengths were somewhat consistent a factor $n_K = 1000$ was chosen and the total number of pulses was then defined to be $K = n_K\gamma$. Then the length of the time series could then be estimated to be $\frac{n_k}{\Delta t} \sim 100000$ datapoints long before removing the transient period in the signal. Removing the transient period of the signal, the lengths of the time series was then ≈ 98000 datapoints long.

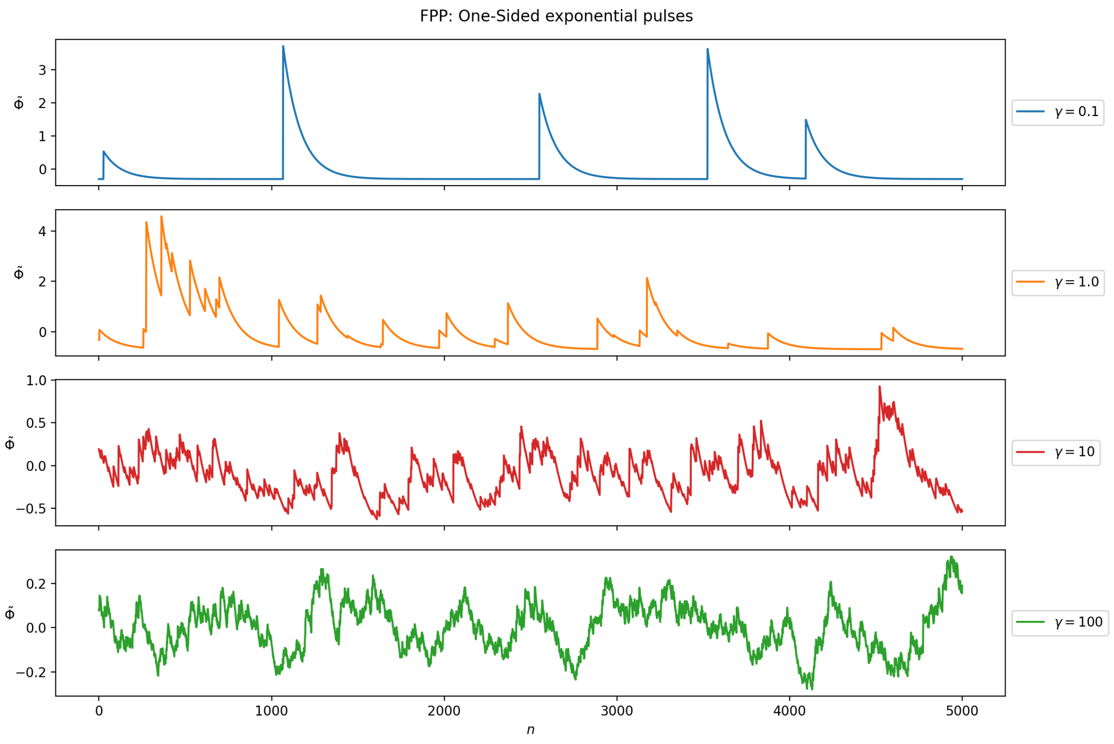


Figure 28: Time series plot of the first 5000 datapoints of realizations of the Filtered Poisson Process for different values of γ with a one-sided exponential pulse shape.

As discussed previously, for low values of γ the pulses are mostly separate, and one can see the high levels of intermittency in the time series. For higher values of γ the higher degree of pulse overlap is apparent and capturing the contribution for single pulses gets difficult. The convergence to the normal distribution as γ increases is also visible in figure 28 as the time series behaves more and more like white noise as γ increases. Considering the evolution of the time series as γ increases, and comparing the bottom plot in figure 28 to the plot of white noise, figure 4, it is not hard to see that they get increasingly similar as γ continues to increase.

The Complexity-Entropy analysis was then applied to the time series from the realizations of the Filtered Poisson Process plotted in figure 28. These time series produced the following Complexity-Entropy plane for the different values of gamma.

[3]https://github.com/audunth/uit_scripts. Note: This code base will eventually migrate to UiT-Cosmo repository

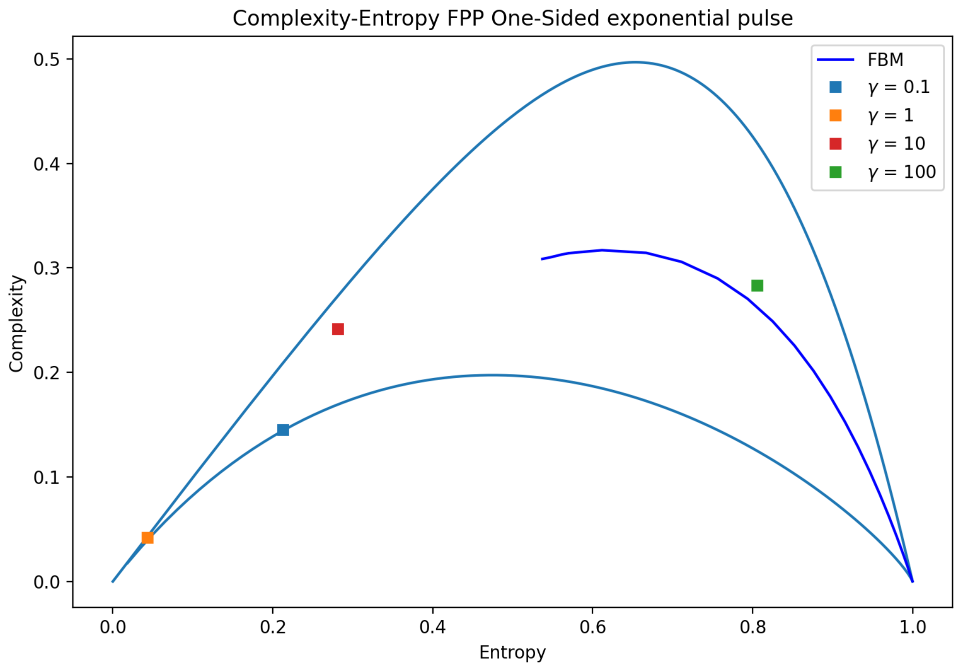


Figure 29: Complexity-Entropy plot of realizations of the Filtered Poisson Process for different values of γ with one-sided exponential pulses

Interestingly, there is quite the spread in the Complexity-Entropy locations for the different realizations depending on the intermittency parameter γ. For the lowest value $\gamma = 0.1$ it is calculated to have minimum complexity, but still being a little random from the very intermittent burst of pulses generated from the stochastic process. The next value, $\gamma = 1$ is stranger still. Even though it is less intermittent than for the previous value of γ, its entropy level is lower meaning even less random. It is for the moment believed that for the lowest value of γ there are vast distances between the pulses. Meaning that for the most part the signal is flat, and as discussed before in section 6.2 the signal is treated as a linear time series. Applying what was learned in section 6.1 the entropy is therefore minimized in these sections of the time series, and thus also the complexity. However, the effect on the entropy by the random burst of pulses makes the signal random to the point it departs from the lower left corner of the Complexity-Entropy plane. The signal for the most part still consists of unstructured, flat parts and therefore the time series for $\gamma = 0.1$ ended up where it is close to the minimum complexity line. This reasoning seems to not hold for $\gamma = 1$. Here it seems the bursts happen often enough for the signal to possess some form of structure for the signal to have some noticeable complexity as the time series seems to appear in between the maximum and minimum complexity line, although this is hard to tell. However, the bursts do not happen often, so the signal have low entropy measure and the random behavior of the stochastic process does not appear.

Again, as γ increases to $\gamma = 10$, the bursts happen more often, so the entropy measure of the time series increase compared to the lower values of γ. The bursts also give the signal some structure as there is an appreciable distance to the minimum complexity line. The same can be said for the case for $\gamma = 100$. The bursts happen more often still, making the entropy measure of the time series increase, and the bursts of pulses in the signal gives the time series some structure. Figure 29 also shows the behavior of the stochastic process that as the intermittency parameter, γ, increases in value the probability density function approaches that of a normal distribution. As γ increases the Complexity-Entropy plane location of the process moves in an arch towards the location for a time series selected from a normal distribution, like the white noise process, located in the bottom right corner of the Complexity-Entropy plane. See figure 5 in section 6.1.

It has been shown in literature for example [3] and [5] that by inspecting the power spectral density that the chaotic fluctuations in the scrape-off-layer of magnetically confined fusion plasma has exponential-shaped power spectral density. This, by Fourier-transform pairs, means that the corresponding pulse shape in time-domain is Lorentzian shaped. To simulate these chaotic fluctuations, the Filtered Poisson Process is then deployed with a Lorentzian shaped pulse function. The corresponding plot of figure 28 is made with Lorentzian shaped pulses instead. The simulation variables and procedures are kept the same, only change is the shape of the pulses. The discretization timestep is the same $\Delta t = 0.01$ and the number of pulses are calculated so that the length of the time series pre-removal of the transient period was 100000 datapoints long. The same values of γ was also used. The simulation of the Filtered Poisson process produced the following figure:

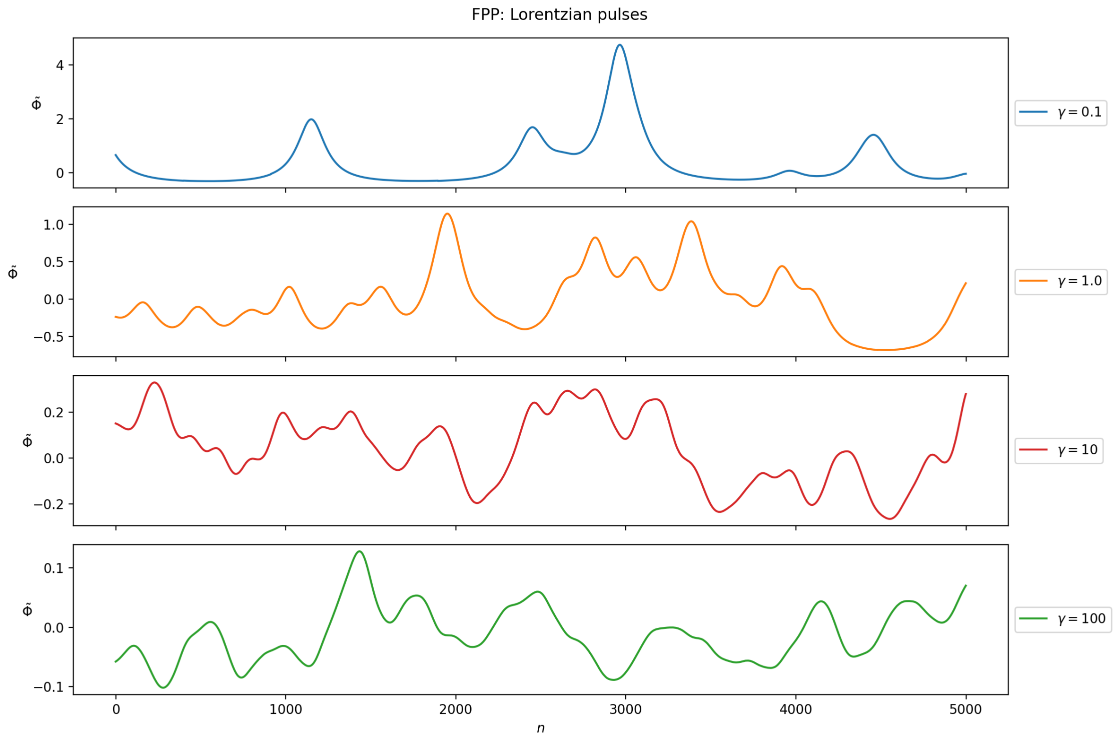


Figure 30: Time series plot of the first 5000 datapoints of the Filtered Poisson Process for different values of γ with Lorentzian shaped pulses.

The Complexity-Entropy analysis was then applied to the time series from the Filtered Poisson Process simulations plotted in figure 30. The time series produced the following Complexity-Entropy plane:

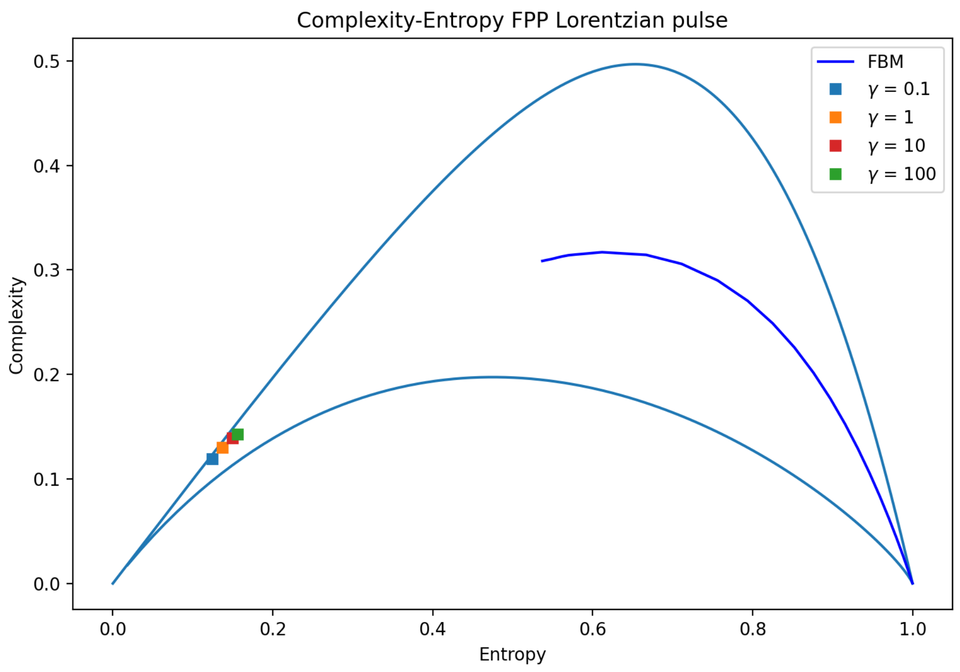


Figure 31: Complexity-Entropy plot of realization of the Filtered Poisson Process for different values of γ with Lorentzian shaped pulses.

Considering the large changes in the behavior of the one-sided exponential simulations in figure 28, the changes in the behavior in the time series between the different values in γ are more subtle in figure 30. One can detect more pulses in the time series as γ increases. But looking only at the two largest values of γ, the differences between them are hard to detect. Because of the nature of the pulse shape, the sharp increase in amplitude value as the pulse appears in the time series for the one-sided exponential case is replaced by a relative slow rise and a slower decay as time increases for the Lorentzian shaped pulses. The calculated locations of the realizations in the Complexity-Entropy plane comes as no surprise, as this was also the case for the Lorenz model, see section 6.3. This slow increase and decrease in the time series mean that a relative few number of amplitude permutations are repeated, leading to low values of entropy. The time series also contain structures, so the complexity measure compared to the maximum complexity line also makes sense. However, these results do not agree with literature. The fluctuations the filtered Poisson process are supposed to simulate are chaotic. Similar to the case for the Lorenz model, the calculated entropy and complexity measures are in the neighborhood of the non-chaotic, periodic sine function, which does not reflect the chaotic nature of the fluctuations are supposed to possess.

Because of the small discretization time step used in the simulation $\Delta t = 0.01$, since it is normalized to the duration time of the pulses, preserves the shape of the pulses. For the case with the Lorentzian shaped pulses where both the rise and decay is, relatively, slow gives the time series a smooth and ordered appearance. When the Complexity-Entropy analysis is then applied to those time series, where consecutive datapoints are compared, the smooth and ordered appearance masks the random behavior of the model and therefore hides the chaotic nature of the fluctuations. To remedy the situation the same analysis method used on the Sine-function in section 6.2 and for the Lorenz model in section 6.3 is used on the Filtered Poisson Process.

Again, this method of resampling the time series required a long time series to be used as the original time series. The defined simulation constant n_K was defined to have the value $n_K = 250000$, which with an original discretization timestep set to $\Delta t = 0.01$ the estimated length of the time series was then $\frac{n_K}{\Delta t} \approx 25000000$ datapoints long. The resampling lag was set to be the range of values from 1 to 2000 datapoints. Making the approximate length of the final resampled time series to be $\sim$ 12500 datapoints long. The same values of γ as previous simulations was used again, $\gamma = [0.1, 1, 10, 100]$. First, the case for one-sided exponential pulses was considered. The Complexity-entropy analysis of the original and resampled Filtered Poisson Process simulation time series for the given values of γ produced the following Complexity-Entropy plots:

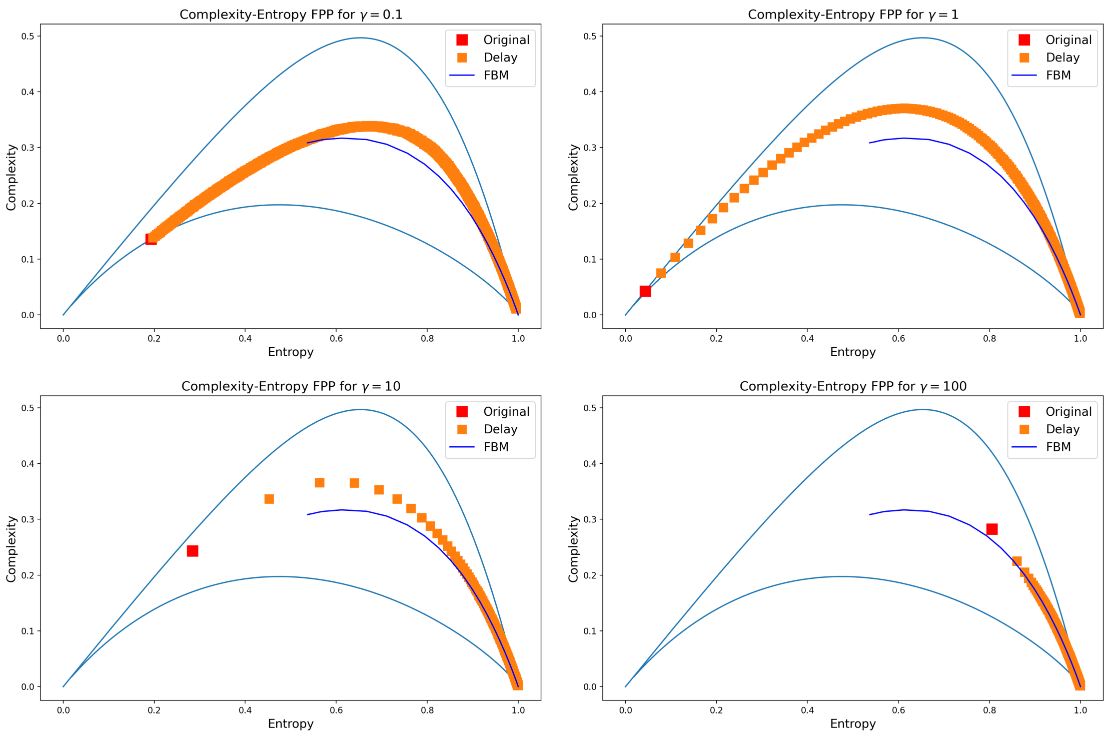


Figure 32: Complexity-Entropy plot of resampled time series for values of $\gamma = [0.1, 1, 10, 100]$ for filtered Poisson process with one-sided exponential pulse shape

From figure 32, it is clear that the resampling of the time series leads to in increase in the entropy measure for all values of γ. Unlike the equivalent plot for the Lorenz model, the complexity and entropy measures does not fluctuate as much as the lag increases.

Likewise for the Lorenz model and the Sine-function, the entropy and complexity measures were then plotted as functions of the resampling lag.

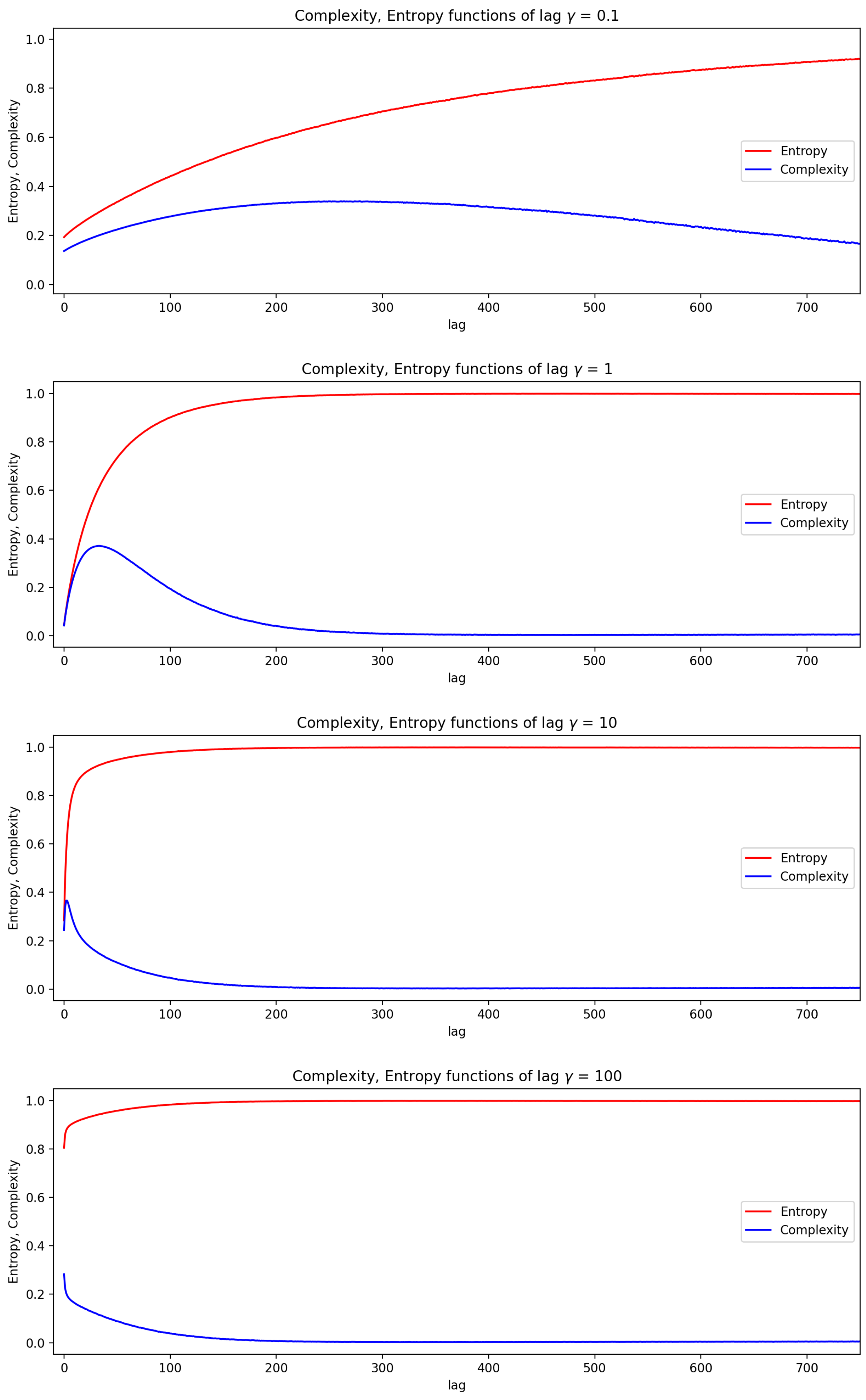


Figure 33: Complexity and entropy plotted as functions of lag for FPP with one-sided exponential pulses for values of $\gamma = [0.1, 1, 10, 100]$ (from top to bottom)

These plots confirm that was said earlier. The entropy and complexity measures increase evenly with increasing lag. The entropy measure increases from the entropy level of the original time series, rising evenly, and eventually converges to its maximum value of $H = 1$. In the beginning for the lower values of γ the complexity measure increases with the entropy, but eventually starts to drop according to the prescribed behavior of the complexity measure. The complexity measure then drops to zero as the entropy measure approaches one. For the highest values of γ the entropy measure of the original time series starts off so high that the complexity measure only drops as the lag increases.

The same analysis was done to the filtered Poisson process with Lorentzian shaped pulses. But as can be seen in figure 31 the location in the Complexity-Entropy plane as γ changes very little. So, the main contribution to the change in the Complexity-Entropy plane for Lorentzian shaped pulses will be the resampling lag introduced in the analysis. For $\gamma = [0.1, 1, 10, 100]$ the Complexity-Entropy analysis of the original time series and the resampled time series of the filtered Poisson process simulations gave the following Complexity-Entropy plots:

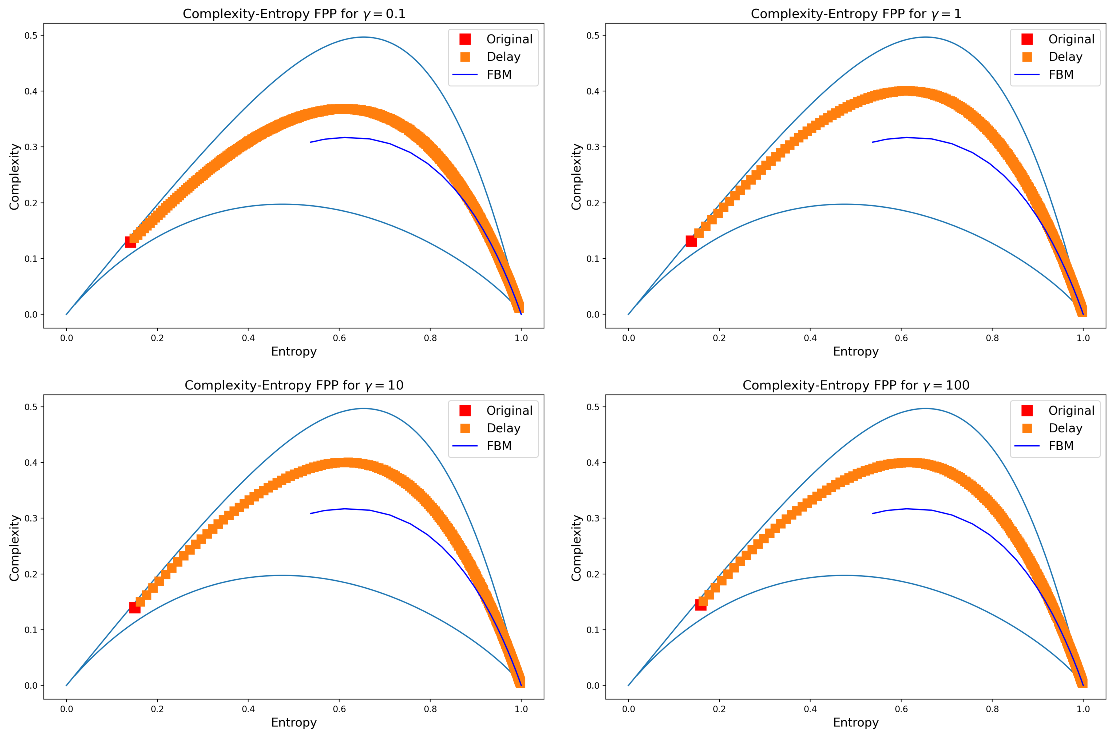


Figure 34: Complexity-Entropy plot of resampled time series for values of $\gamma = [0.1, 1, 10, 100]$ for filtered Poisson process with Lorentzian pulse shape

The complexity and entropy measure were then also plotted as functions of lag and produced the following figure:

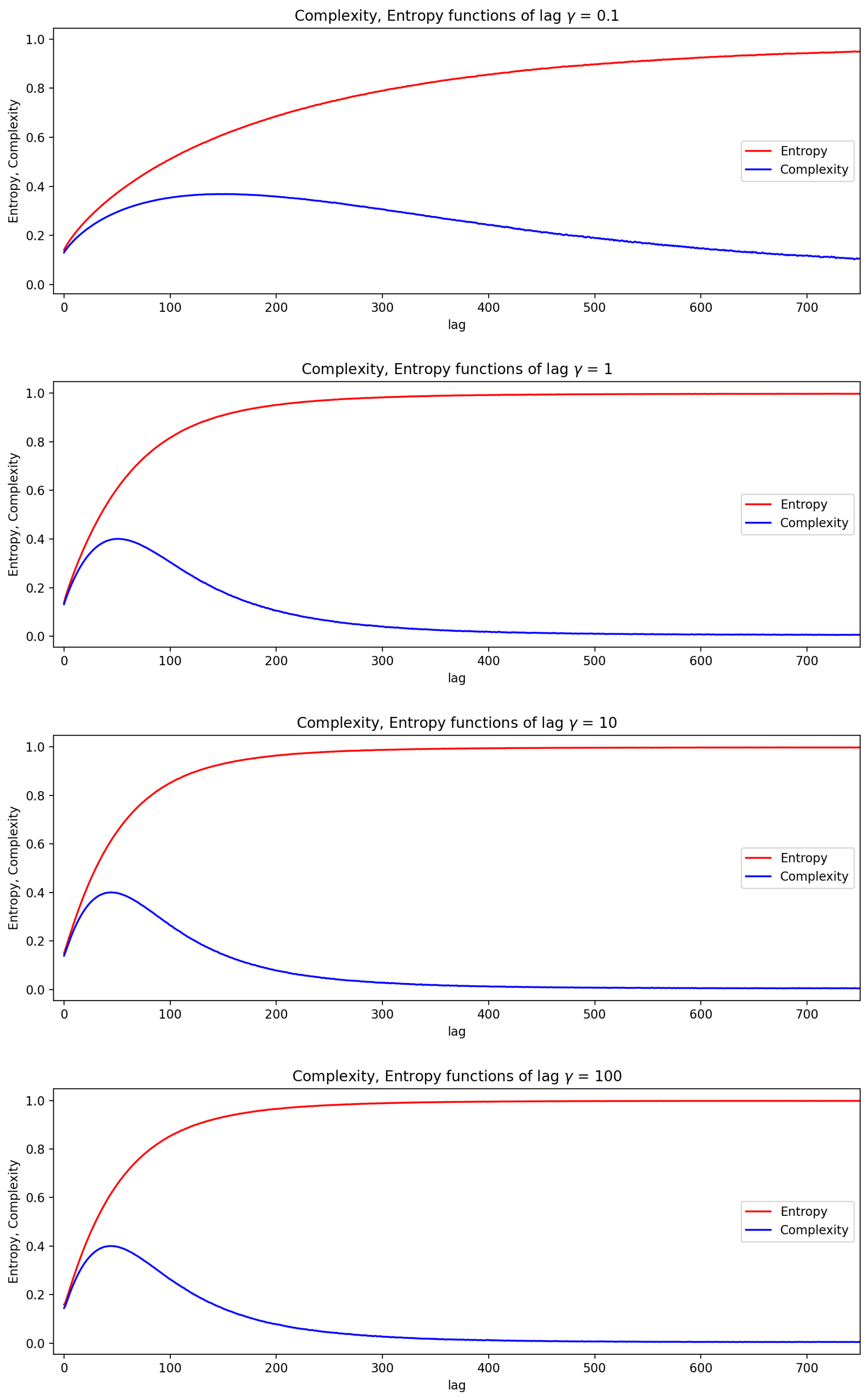
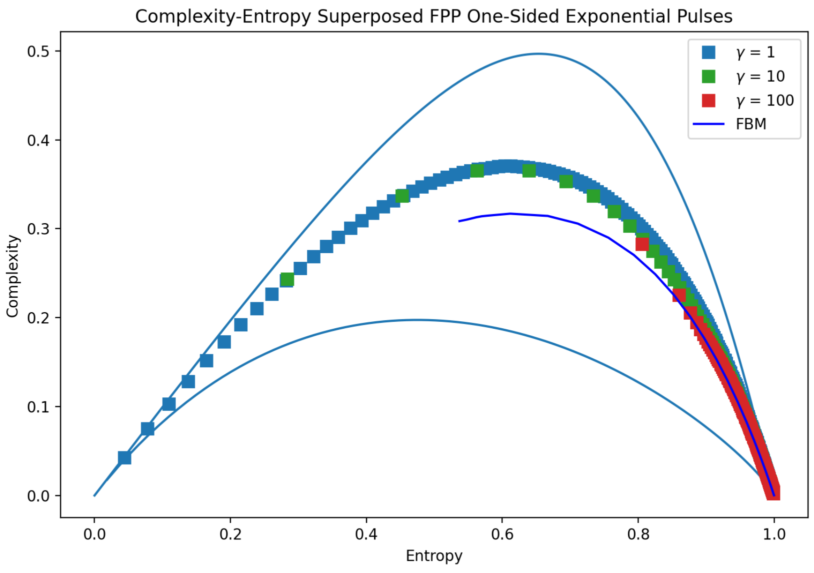


Figure 35: Complexity and entropy plotted as functions of lag for FPP with Lorentzian pulses for values of $\gamma = [0.1, 1, 10, 100]$ (from top to bottom)

The changes in complexity and entropy are more sudden for $\gamma = 1, 10$ and 100 compared to $\gamma = 0.1$. Figure 35 also show that $\gamma = 1, 10$ and 100 reaches levels of complexity around 0.4 which is higher than that reached for $\gamma = 0.1$. A similar case can also be seen in figure 33 with one-sided exponential shaped pulses. There, the complexity and entropy measures change more slowly and does not reach the same level of complexity compared to the higher γ cases. But again, for one-sided exponential pulses the starting location in the Complexity-Entropy plane changes a lot for γ. One interesting feature that one can see for both one-sided exponential pulses and Lorentzian pulses, when ignoring the low-γ case for $\gamma = 0.1$, the calculated complexity and entropy levels seem to follow a well-defined curve in the Complexity-Entropy plane. To see if this is the case, the different plots in figure 32 and 34 are superposed together in the same figure. This produces the following figures:



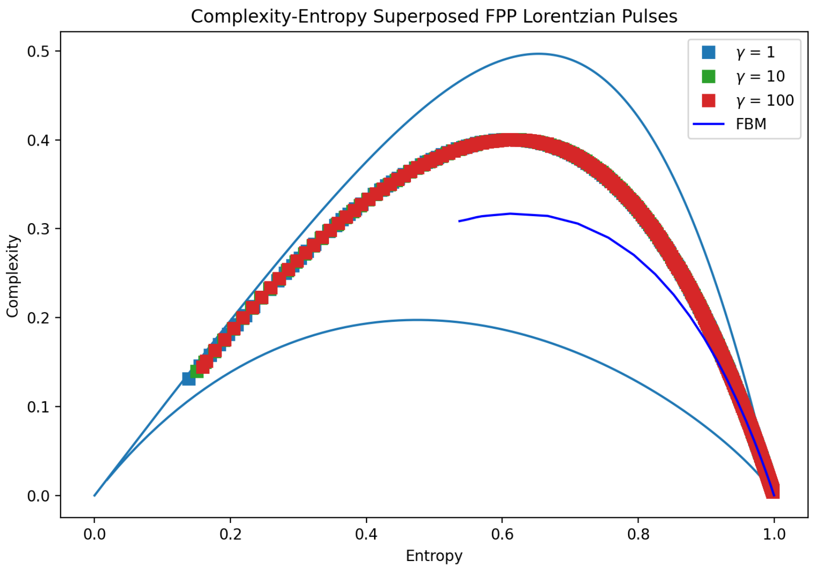


Figure 36: Superposed resampled Complexity-Entropy plane plots for one-sided exponential pulses and Lorentzian shaped pulses.

For the Lorentzian shaped pulses, the curves spanned by the calculated Complexity-Entropy entries overlap almost completely for the chosen values of γ. The only deviation is the lower entropy and complexity starting location for the different values of γ. The same cannot be said for realizations of the filtered Poisson process with one-sided exponential pulse shape. Here, the starting locations in the Complexity-Entropy plane is quite different as the value of γ increases and the curve they span seem to have some thickness, and therefore not quite well defined. It is still, however, quite possible that this is the case for Lorentzian shaped pulses as well, but the effect of the intermittency parameter for the Lorentzian shaped pulses is not as apparent as compared to the one-sided exponential pulses. One therefore does not cover the same range in starting locations as for the one-sided exponential pulse case. As the starting location with Lorentzian shaped pulses are very close together, it is therefore not unreasonable to assume the curves plotted as more lag in introduced in the resampling of the time series also lie close together. And as for a larger range of starting locations, like for the one-sided exponential pulse, any small differences can become more apparent and result in Complexity-Entropy curves which does not lie perfectly on top of each other. Nevertheless, the fact that the is not much difference between the curves plotted out for different values of γ for the one-sided exponential pulse shape case, and hardly any for the Lorentzian shaped pulses is very interesting. The implications of this can mean that, for at least constant pulse duration, each pulse shape may span a narrow band in the Complexity-Entropy plane where every realizations of the filtered Poisson process will lie.

8 FPP With Pareto Distributed Duration Times

In section 7 the duration time of the pulses were kept constant for the whole analysis. Now, the filtered Poisson process will be simulated with Pareto distributed duration times. A Pareto distribution is a power-law probability distribution. A Pareto distribution can be introduced into some of the random variables in the filtered Poisson process model (19), like the amplitude A_k, arrival times t_k or in this case the duration times τ_k.

Certain dynamical models when let to evolve with time can be found to organize themselves around a critical state reaching near-critical balance without fine-tuning from the outside world [27]. These dynamical models are then said to be self-organized critical systems [27]. Rooted in this concept of self-organized critical systems is the concept of self-similarity [27]. Self-similar systems behave the same independent of the scale the systems are observed at. Fractals and power-laws have this property of self-similarity in the scaling [27]. Processes which have this self-similarity property often have slower-than-exponential decay of the auto-correlation function which gives long-range correlation in the system [27]. Investigating the power spectrum of these systems gives inverse power-law dependence on the frequency $\sim 1/f^{\beta}$.

Introducing a power-law distribution in the random variables of the system, in this case for the duration times τ_k, resulted in certain parameter ranges ($1 < \alpha < 3$) a power-law dependence in the power spectral density function, the filtered Poisson process should then possess the self-similarity property and mimic a self-organized critical system [27].

There are two variants to the Pareto distribution. Where the possible duration time span from τ_{min} to $+\infty$ and denotes as standard Pareto distribution. The other one is denotes as a bounded Pareto distribution spanning the range from a minimum value τ_{min} to the maximum value τ_{max}. Because of limitations of the standard Pareto distribution [27], the bounded Pareto distribution is considered. The bounded Pareto probability density function is defined by the following equation [27]

$$P_{\tau}(\tau) = \frac{(\alpha-1)(\tau_{max}\tau_{min})^{\alpha}}{\tau_{max}^{\alpha}\tau_{min} - \tau_{max}\tau_{min}^{\alpha}}\frac{1}{\tau^{\alpha}}. \tag{28}$$

By introducing the parameter $d = \frac{\tau_{max}}{\tau_{min}}$ and normalize such that the average duration time of each pulse is $\tau_d = 1$ gives the following relation [27]:

$$P_{\tau}(\tau) = \frac{(\alpha-1)(\frac{(\alpha-1)(d^{2-\alpha}-1)}{(\alpha-2)(d^{1-\alpha}-1)})^{1-\alpha}}{1-d^{1-\alpha}}\frac{1}{\tau^{\alpha}}. \tag{29}$$

Where the normalized upper and lower bound are given as [27]

$$\hat{\tau}_{min} = \frac{(\alpha-2)(1-d^{1-\alpha})}{(\alpha-1)(1-d^{2-\alpha})}, \qquad \hat{\tau}_{max} = \frac{(\alpha-2)(-1+d^{1-\alpha})}{(\alpha-1)(-d^{-1}+d^{1-\alpha})}. \tag{30}$$

Outside this range $P_{\tau}(\tau) = 0$.

Equation (29) can be boiled down to a complicated function of α and d, $P_{\tau} = f(\alpha, d)\frac{1}{\tau^{\alpha}}$, this $\frac{1}{\tau^{\alpha}}$ is the inverse power-law feature in the probability density function for the duration time distribution. The following figure is a quick sketch showing the interesting parameters for a power spectral density plot demonstrating the power-law shape for some power spectral density function proportional to $1/\omega^{\beta}$, where $\beta = 3 - \alpha$. The "distance" marked with d should not be considered a distance. If the figure was plotted in a log-log plot, where the axis of the figure is scaled to the logarithm of the variable and d should then be considered how many decades larger the higher limit is to the lower limit.

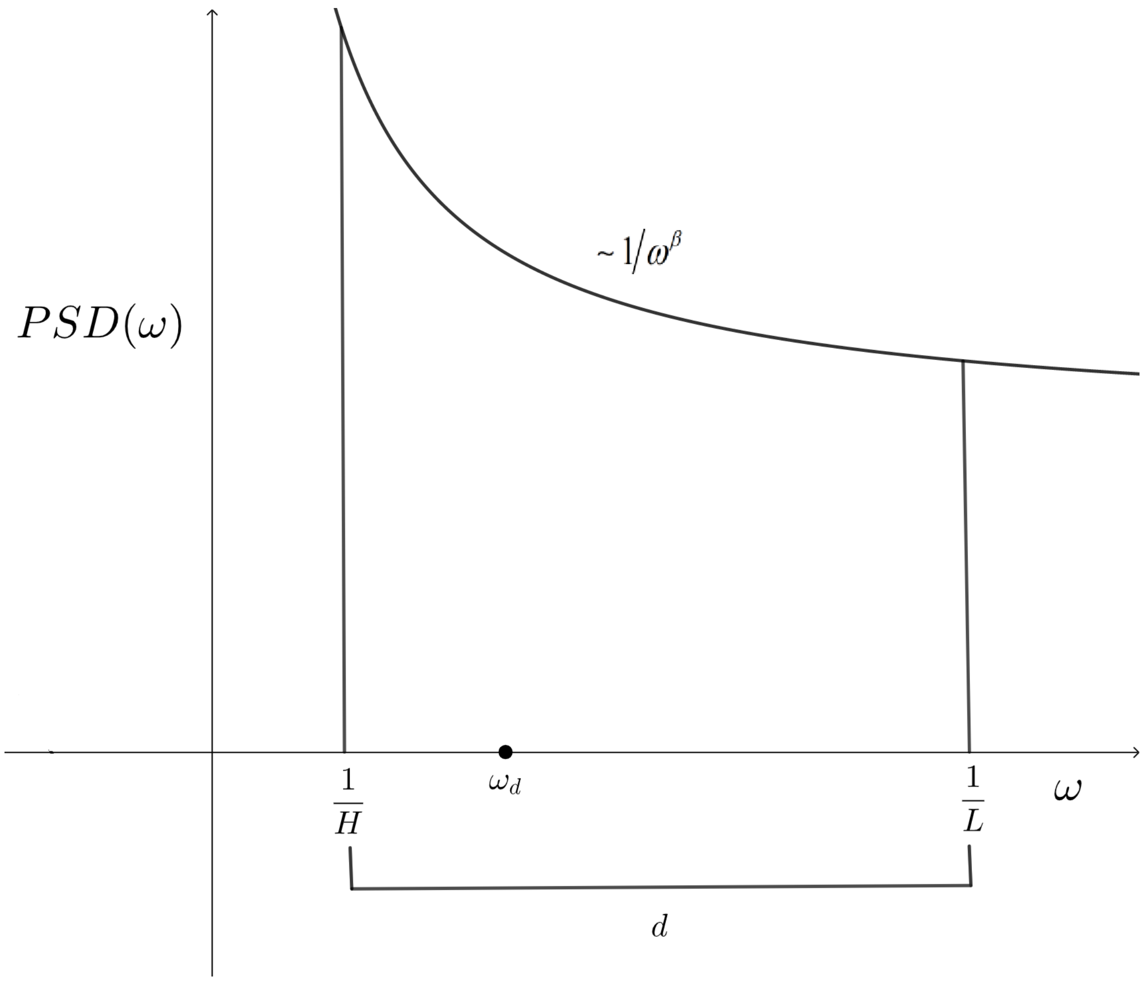


Figure 37: A sketch demonstrating the important parameters for the Pareto distribution.

During the simulation of the filtered Poisson process, the parameter d is fixed to the value $d = 10000$ meaning that the higher bound is 10000 times larger than the lower limit [27]. The duration time distribution was as mentioned normalized so that the mean duration time was $\tau_d = 1$. For a given value of α, the upper and lower limits were calculated accordingly following equations (30).

The filtered Poisson process was simulated for given values of $\gamma = [0.1, 0.3, 1.0, 3.0, 10.0, 30.0, 50.0, 100.0, 300.0]$ and for each value of γ used, the filtered Poisson process was simulated with Pareto distributed duration time with the given values of $\alpha = [1.00, 1.20, 1.40, 1.60, 1.80, 2.00, 2.20, 2.40, 2.60, 2.80, 3.00]$. The filtered Poisson process simulations with these model parameters was done for two pulse shapes, box-pulse and one-sided exponential pulse shape. The discretization timestep used during the simulation was not kept constant, it changes depending on the parameter α so that the the parameter d could be as large as possible to be get a good Pareto range and to keep the computation cost at a minimum. An example time series from the simulation is plotted in the following figure

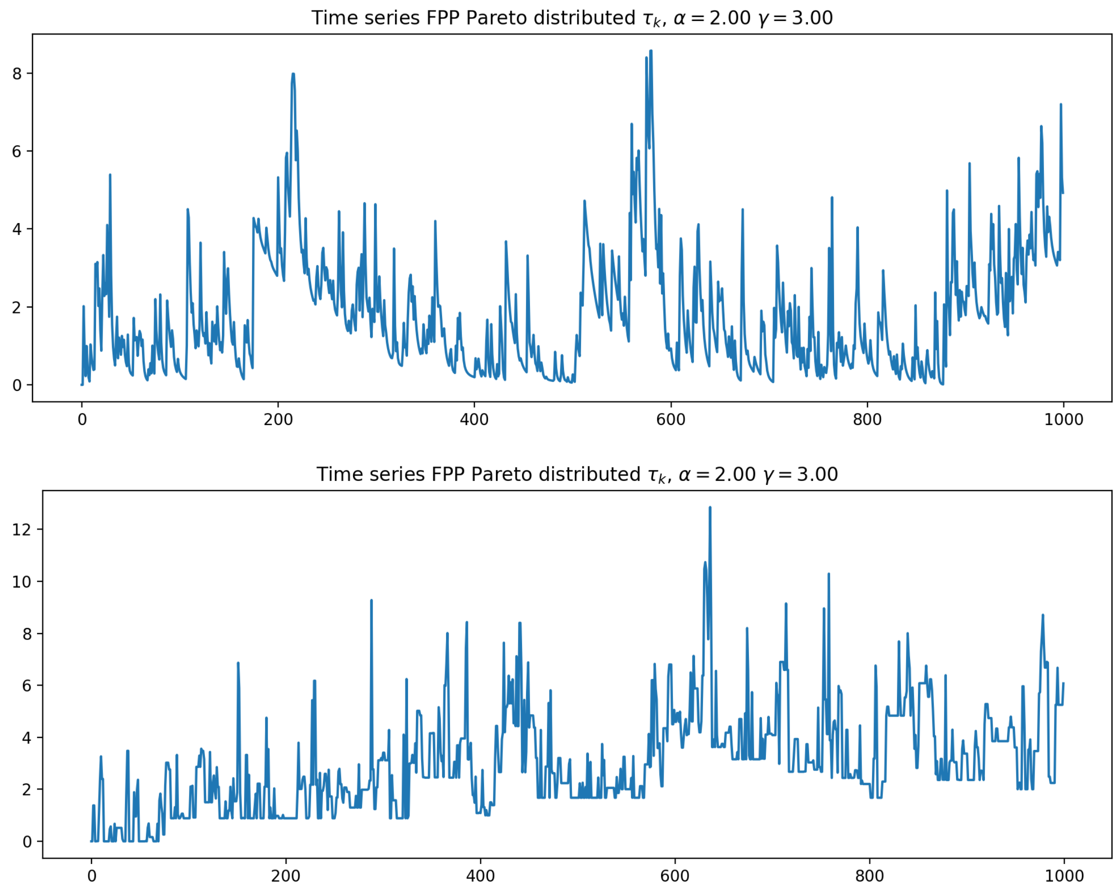


Figure 38: Time series plot of the filtered Poisson process simulation with Pareto distributed duration times for $\alpha = 2.0$ and $\gamma = 3.0$. One-sided exponential pulse (top), box pulse (bottom)

To have some structure in the Complexity-Entropy analysis, the simulation time series was grouped so that α was kept constant and the intermittency parameter γ was changing. This also made it so the simulation discretization timestep was constant as the discretization timestep varies with α. This was done to keep the number of parameters which is known to change the Complexity-Entropy plane location, based on the analysis already done in the previous section, of each simulation to a minimum. Here in this case, only γ.

The Complexity-Entropy analysis was then applied to the simulated data. Keeping the Pareto parameter α constant and varying the filtered Poisson process parameter γ with both box-pulses and exponential pulses produced the following plots:

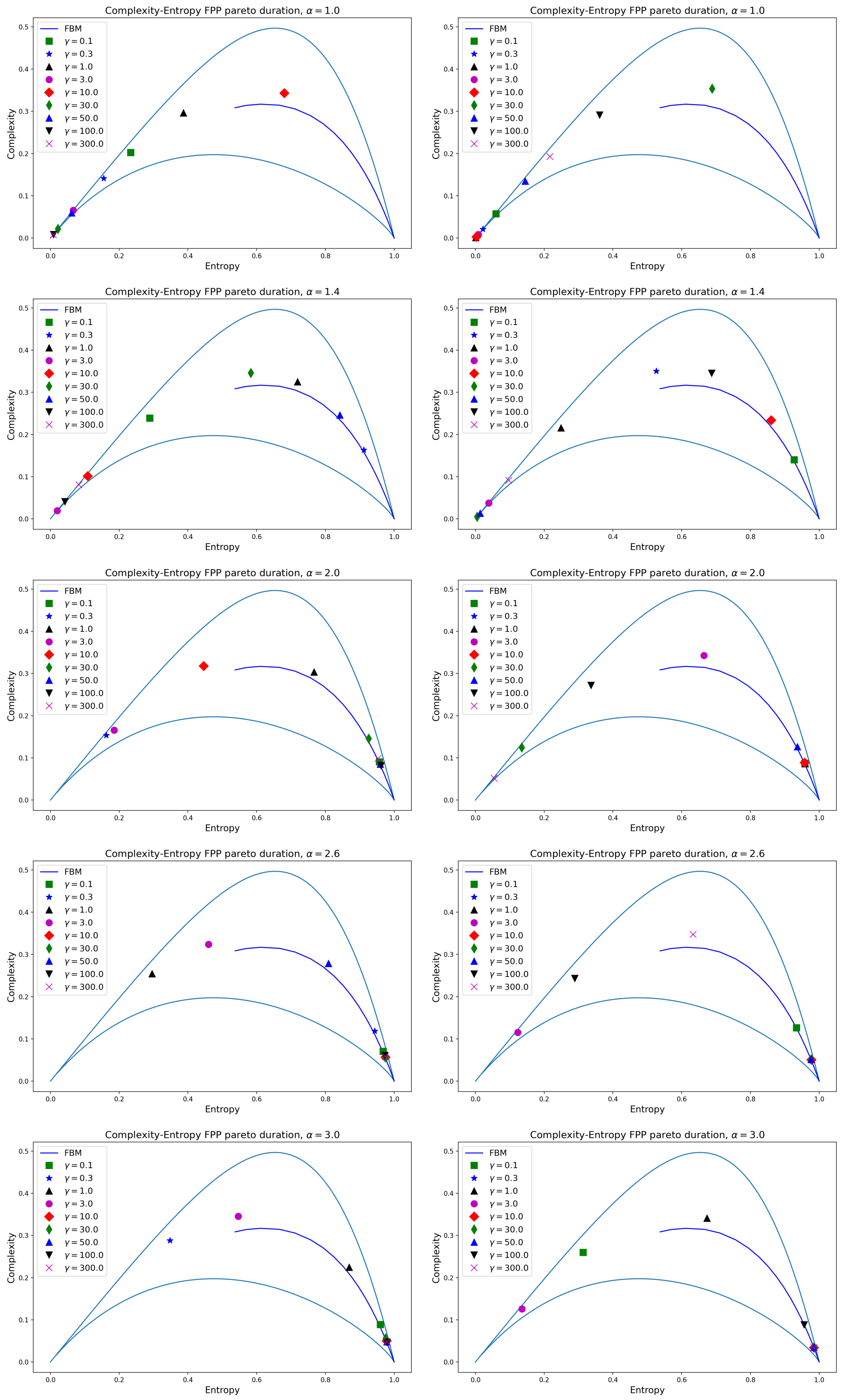
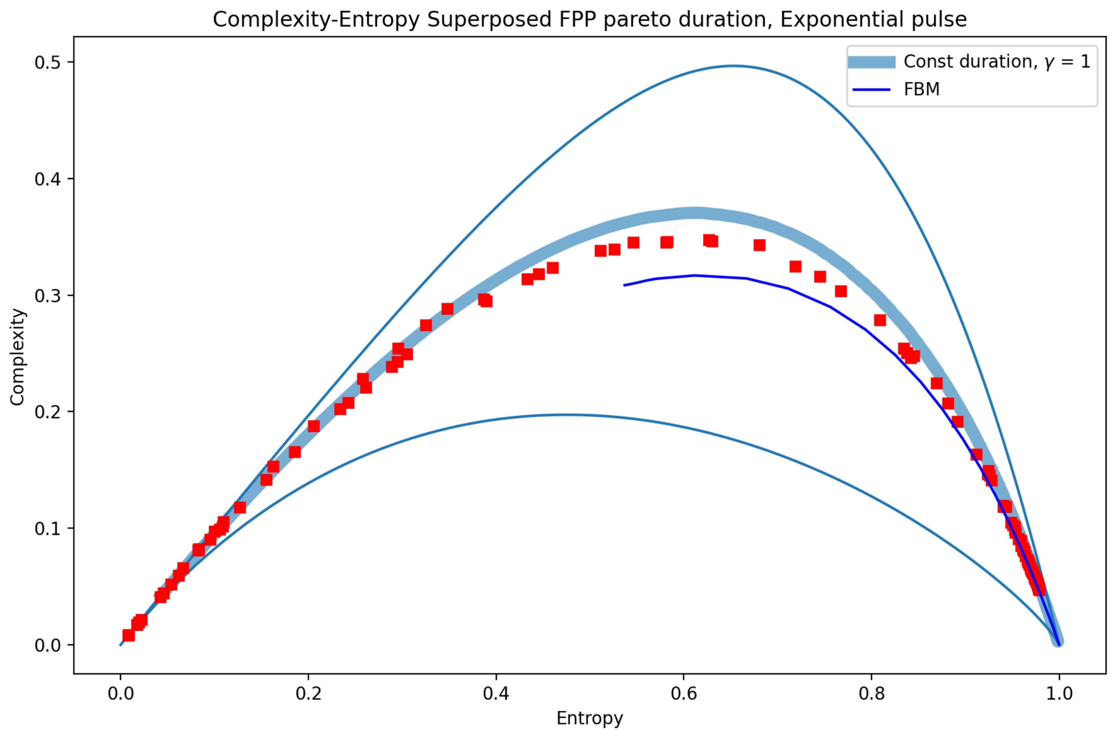


Figure 39: Complexity-Entropy analysis of filtered Poisson process with Pareto distributed duration times for selected values of α with varying γ. One-sided exponential pulses on the left side and box-pulses on the right.

From figure 39 one thing is clear: it gets hard to draw any concrete conclusion from the figures. In general it can be said that as α increases, the average entropy levels for the different γ's increases. As an increasing number of γ values ends up in the lower right corner of the Complexity-Entropy plane. This seems to be the case for both pulse shapes. One obvious effect of the Pareto distributed duration times is that effect of increasing γ as discussed in the previous section, is hardly noticeable. The introduction of Pareto distributed duration times for the pulses has made it harder to predict the location of the time series in the Complexity-Entropy plane using the intermittency parameter γ. As discussed in section 7, as γ increases the behavior of the filtered Poisson process should converge to that of a sampling from a normal probability distribution, as the probability density function of the filtered Poisson process converge to a normal distribution. This is not the case looking at figure 39. Looking at the case for $\alpha = 1.40$ for both pulse shapes, the highest value of $\gamma = 300$ appears in the lower left region of the Complexity-Entropy plane, characterized in the case with constant pulse duration with very low levels of γ.

However, it makes sense that when the duration time of the pulses can vary that there no longer is a clear relationship with the entropy measure and the intermittence parameter γ as for constant pulse duration. Now, the consecutive values considered in the time series when calculating the Bandt-Pompe ordinal pattern probability distribution now depend on both the intermittency parameter and the duration time of the particular pulse the datapoints are a part of. But that the pulse duration time would completely negate the effect of the intermittency parameter in some of the plots in figure 39 was rather unexpected.

Looking at the different locations in the Complexity-Entropy plane for increasing α in figure 39, the points seem to span a curve in the Complexity-Entropy plane. However, less obvious compared to the case with constant pulse duration, figure 32. To see the curve better, the individual plots in figure 39 for a given pulse shape are superposed together in the same figure. In addition, the line for constant pulse duration for $\gamma = 0.1$ in figure 32 was included as a guide as the figures for Pareto distributed duration times seem to lie close to that line, and it is the line which covers most of the possible range in the complexity and entropy measures.



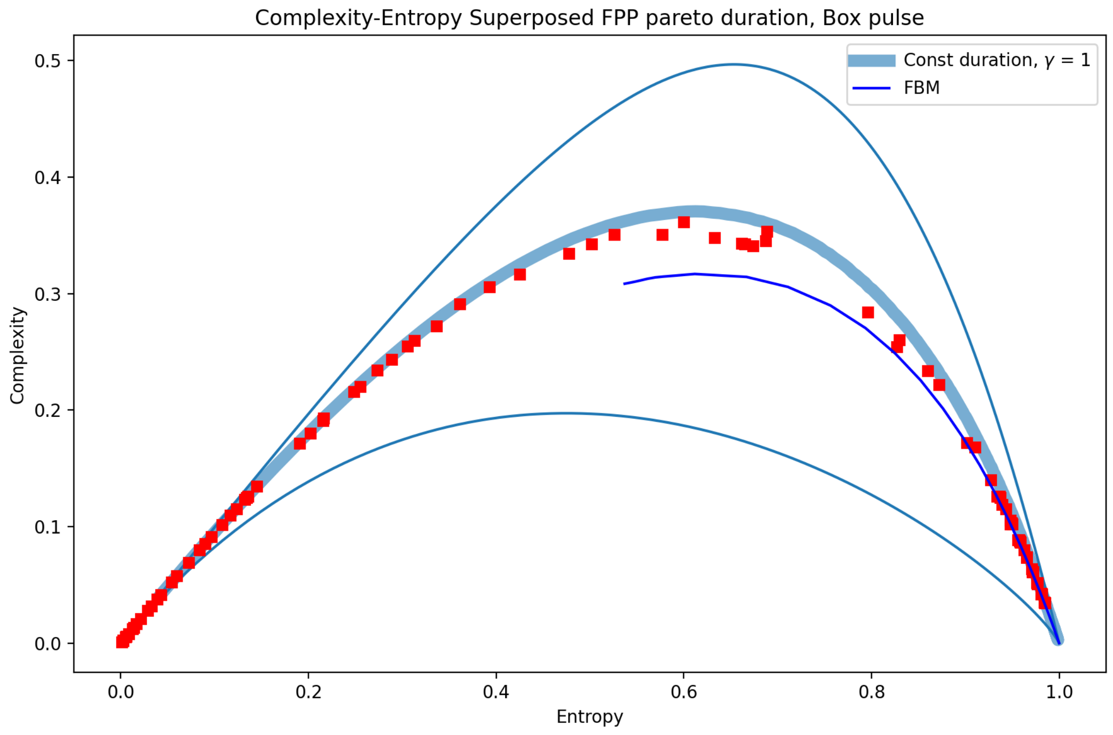


Figure 40: Superposed version of the Complexity-Entropy plots of figure 39 with exponential pulses on the top plot, and box pulses on the bottom plot

Inspecting the plots in figure 40 one can see that the Pareto distributed duration time Complexity-Entropy plane entries closely follow the drawn line from the Complexity-Entropy plane entries for constant pulse duration. The entries in the Complexity-Entropy plane achieves slightly lower complexity for the same levels of entropy compared to constant pulse duration. The pareto distributed duration time Complexity-Entropy entries do span a curve in the Complexity-Entropy plane; however, the curve is not as well defined as the constant pulse duration as the entries spanning the curve makes the curve have some thickness. Entries close to each other in entropy value has noticeably different values of complexity. The reason behind the thickness of this line may be caused by the points not having converged properly and fluctuate around their proper locations in the Complexity-Entropy plane.

Further research on the filtered Poisson process with Pareto distributed duration times would be to run these simulations for a longer time period to obtain longer time series to see of the locations of the model simulations have converged to their proper location in the Complexity-Entropy plane. In the previous section, the simulation time series length for constant pulse duration used in the Complexity-Entropy analysis were in the order of 10^6 datapoints long and for the resampling plots the starting length of the time series were in the order of 10^7. Or at the very least, run several simulations with the same length to see if the Complexity-Entropy plane locations change between simulations to determine the consistency of the simulations, and determine if the points in the Complexity-Entropy plane have converged to their proper locations.

9 Bak-Tang-Wiesenfeld (BTW) and SOC Models

In this section the Bak-Tang-Wiesenfeld (BTW) sandpile model is under study. The BTW sandpile model is a simple model which exhibit self-organized criticality (SOC) [28]. The self-organized criticality, as described in section 8, is a property certain system have where the system will organize itself to a near-critical state, without input from the outside world.

The BTW model's self-organized criticality can readily be explained form the dynamics from the system [28]. For a visual aid, the model can be thought of a grains of sand on a grid. An external drive will add grains of sand into the system and whence the number of grains of sand on each grid point piles up and reaches a critical level, the sandpile will collapse and distribute its grains of sand to its neighbors. If the collapsing grid point is at the edge of the system, the grains of sand will still be distributed to its neighbors but the grains of sand which moves out of the defined grid of the system, the grains of sand are lost. So, the system then has an external addition and a dissipation of grains of sand. Following the nomenclature of [28], when the system is in a low average grain density state, named phase A, only small events (toppling of critical states) will occur. When the system is in a high average grain density state where the system will continuously topple, is named phase B. When the system is in phase A, and the external drive add grain of sand slowly to the system, increasing the average density of grains of sand. When the system then reaches a critical grain density, the toppling of the critical grid points will remove grains from dissipation at the boundaries of the system and lower the average grain density of the system. Therefore, because of the dissipation of grains of sand the system will never reach phase B [28] and the system moves itself out of the critical state. The self-organized criticality will then occur when the timescale of the external drive with addition of grains of sand is larger than the timescale of the dissipation [28].

The classical BTW model can be described as follows; Assuming the 2D grid of the system has length L, the number of grip points will be L^2. At a given time t, each grid point in the system as associated with itself an integer number $z_t(x, y)$ which initially is randomly assigned. This integer z_t represents the number of grains of sand at grid point (x, y). Any grid point where $z_t(x, y) \geq 4$ is said to be unstable (or critical) and will collapse distributing its grains of sand to its immediate neighbors following:

$$\begin{aligned} z_{t+1}(x, y) &= z_t(x, y) - 4 \\ z_{t+1}(x \pm 1, y) &= z_t(x \pm 1, y) + 1 \\ z_{t+1}(x, y \pm 1) &= z_t(x, y \pm 1) + 1 \end{aligned} \tag{31}$$

At the edge of the system grid, the grains of sand will be distributed to "grid points" outside of the system grid. These grains of sand will then be considered lost and fall out of the grid. When the system grid has no critical grid points, grains of sand will be added to the system at a random position. This addition of grains will continue until there exists an unstable grid point, and the unstable grid point will collapse and give its grains to its neighbors. The toppling of the unstable grid point might cause the neighboring grid points to become unstable which will then collapse and so on creating an avalanche. Each avalanche will last a certain amount of time until the system reaches a state where all grid points are stable again. After the system is stable again the random addition of sand grains will start again. The time series that will be extracted from the sandpile model is the number of critical grid points at time t, $F(t)$ [28]. The time series of one of the BTW models is plotted in the following figure:

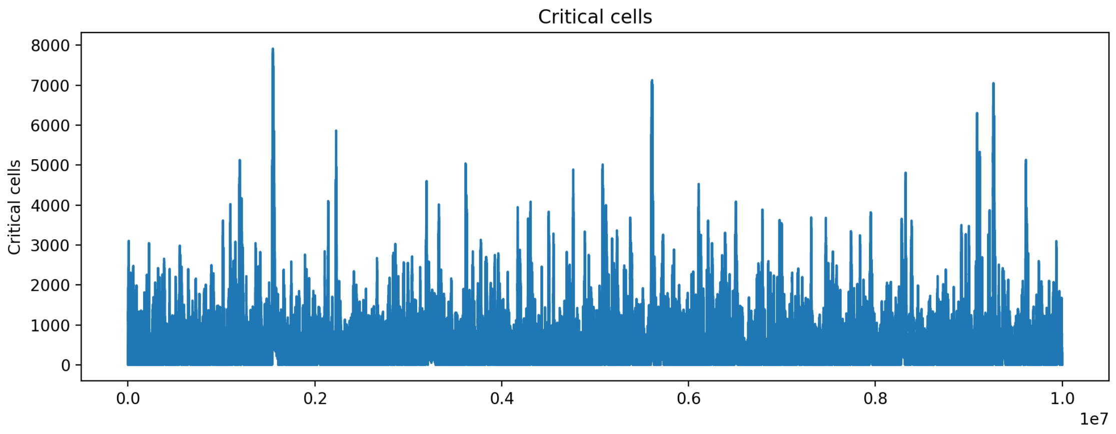


Figure 41: Time series plot of the number of critical cells in the sand pile model.

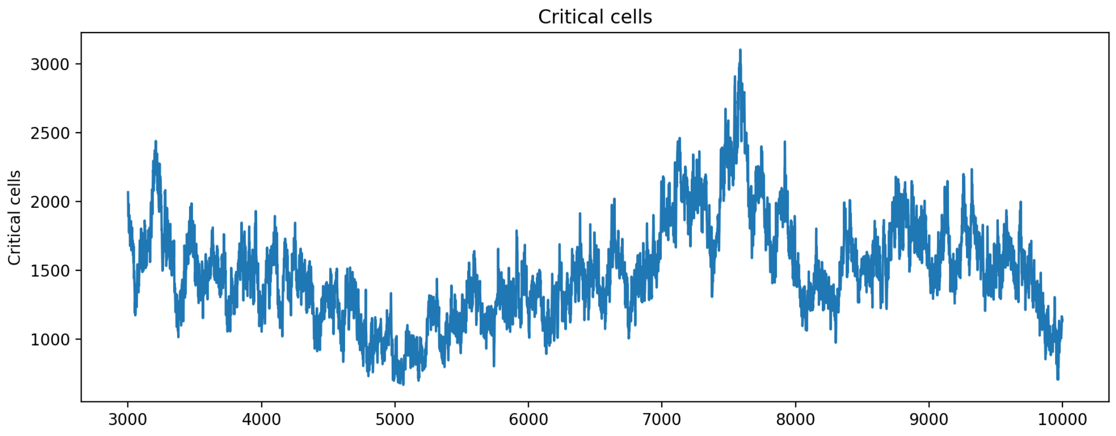


Figure 42: Time series plot of the number of critical cells in the sand pile model zoomed in. Datapoint from 3000 to 10 000 is plotted.

As mentioned in section 8, self-organized criticality are often associated with power-laws, like for example in the power spectral density. Examining the time series for the size and duration of the avalanche events one can show that the probability distribution of the duration of the events, τ, follow a power-law [28]:

$$P_\tau(\tau) \sim \tau^{-\alpha} \tag{32}$$

Also, the size of the avalanches can be written in terms of the duration of the avalanches, following

$$S(\tau) \sim \tau^{\gamma} \tag{33}$$

And the power spectral density follows a power-law [28]

$$\Omega_F(f) \sim f^{-\beta} \tag{34}$$

The Complexity-Entropy analysis was applied to six different simulations of the BTW model with different toppling rules and configurations. For example, the model named *BTW 1* is a 3D implementation of the BTW model. Where the critical state for each grid point is 6 grains. When the grid points reach this critical state the grid point collapses and sends four grains to each of its neighbors in the same plane as the critical state, while the other two grains are distributed to the grid point directly above and below the critical state in the 3D grid system. The model labeled *BTW 2* is for the normal 2D grid, but now when the grid points reach the critical state of four grains, only two grains will be distributed either to the horizontal neighbors or vertical neighbors chosen randomly. The model labeled *BTW 5* is the classical sandpile model where the grid is 2D and the toppling rules follows that described by equations (31)

There is a special case for these BTW models, labeled *BTW 6*, which is a forest fire model. Here the toppling rules follows that which is described in the Wikipedia article[4]. The mechanics of the model can be described as follows: A burning cell (or in the nomenclature above a critical grid point) will turn into an empty cell. The tree in a cell will burn if at least one of its neighbors is burning. There is a probability p_{fire} that a tree will start burning independent of the sate of its neighbors, meaning there is a probability that the tree will spontaneously catch fire. An empty cell has a probability p_{tree} that a tree will grow in the empty cell and fill it. So, the dynamics of the forest fire model is quite different than that of the normal sandpile models.

The Complexity-Entropy analysis of the six different BTW model produced the following Complexity-entropy plane plot.

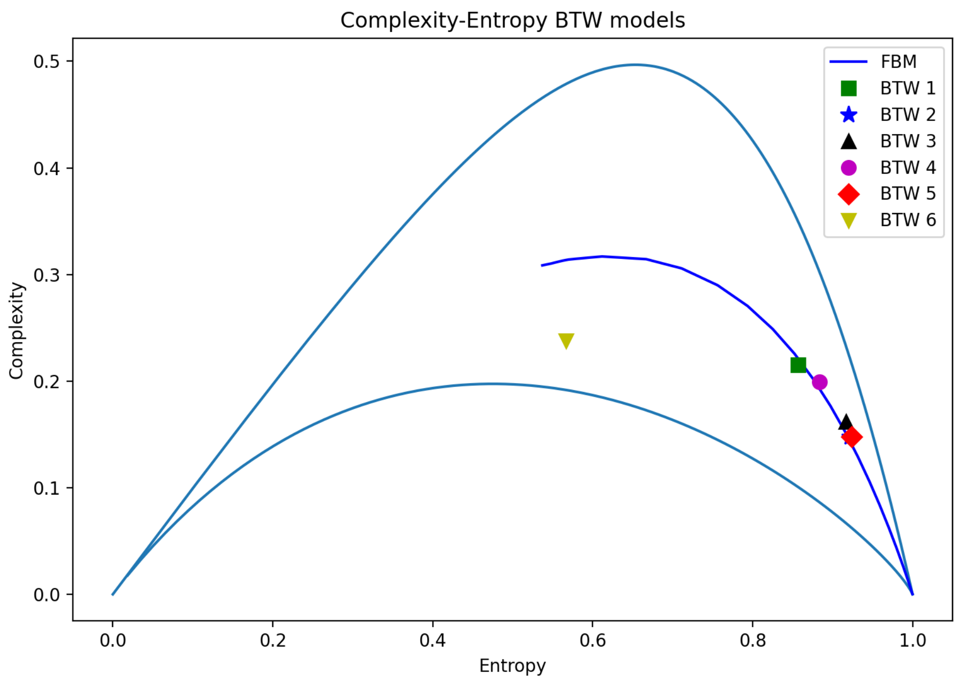


Figure 43: Complexity-Entropy analysis of BTW models with different toppling rules and configurations, most noticeable BTW 5 which is the classic BTW sandpile model, and BTW 6 which is the forest fire BTW model. BTW 2 is almost invisible behind the BTW 5 mark in the figure.

One very noticeable feature of figure 43 is that each model, except for the forest fire model (BTW 6), lie close to the fractional Brownian motion line. This result is consistent with that described in the literature [28, 29, 30]. The idea behind this is that the BTW models can be modelled by a stochastic differential equation (Using the same notation as [29]):

$$dX(t) = f(x)dt + \sigma\sqrt{X(t)}dW(t) \tag{35}$$

Which includes a drift term, $f(x)dt$, and a noise term, $\sigma\sqrt{X(t)}dW(t)$. Then they argue in [29] and [30] that when the definition of what is considered a toppling event, or avalanche event, to when the avalanching activity in the model is above a certain threshold X_c, the drift term can be ignored when one also assumes that the avalanches are small enough to not be affected by the boundaries of the system [29]. One is then left with the stochastic equation written in the form of a stochastic difference equation [29]:

$$\Delta x(k) = \sigma\sqrt{x(k)}w(k) \tag{36}$$

[4]https://en.wikipedia.org/wiki/Forest-fire_model

where $w(k)$ is a stationary, normalized, and uncorrelated Gaussian process. This can be rewritten in the form of a normalized toppling process [29, 30]

$$w(k) = \frac{\Delta x(k)}{\sigma \sqrt{x(k)}} \tag{37}$$

Running numerical simulations of the BTW models, one finds that the $w(k)$ can be described accurately as noise characterized by a Hurst exponent H [29], similar to a Gaussian noise process. This means $w(k) = W_H(k+1) - W_H(k)$, where W_H is a normalized fractional Brownian motion [29]. By the introducing variable rescaling one can then obtain the stochastic differential equation [29]

$$dX(t) = \sigma \sqrt{X(t)} dW_H(t) \tag{38}$$

By then ignoring the $\sqrt{X(t)}$ dependence on the right-hand side of equation (38) one obtain the equation for fractional Brownian motion with known scaling exponents $\alpha = 2 - H$ and $\beta = 2H + 1$ [28, 29].

In the work done by Losada [28], who did the simulations of the BTW models, the equivalent scaling exponents of α and β was calculated from the BTW time series data and plotted them in figure 10 in [28]. In addition to the estimated scaling exponents from the BTW model, the same scaling exponents for the fractional Brownian motion was calculated and plotted in the same figure. The result from that analysis was that there was good agreement between the scaling exponents from the fractional Brownian motion and those from the BTW model. Which suggests that the fractional Brownian motion might be a good candidate to describe the BTW sandpile models [28].

Returning to the result from the Complexity-Entropy analysis in figure 43. The result does indeed support the claim that fractional Brownian motion can be used to describe the BTW model and presents itself in excellent agreement with the literature [29, 30] and the work done by Losada [28]. The story is slightly different for the forest fire model BTW 6, that deviate away from the fractional Brownian motion unlike the rest of the BTW model simulations. However, in figure 10 in [28] the forest fire model places itself in good agreement with the scaling for the fractional Brownian motion. So, the scaling exponents for this model is in agreement with the scaling for the fractional Brownian motion, but the result from the Complexity-Entropy analysis is in a contradiction. The cause of this is believed to arise from the different toppling rules set for the forest fire model. The dynamics of the forest fire model is very different from the dynamics of the classical BTW model. For the forest fire model, any burning will cause their neighbor to start to burn with the assumption that the critical cell has neighboring trees. Also, each tree has a given probability to start burning regardless of condition of its neighbors. This will in turn lead to quite different dynamics which is believed to manifest itself with entropy and complexity measures that differs from a fractional Brownian motion. It is therefore believed that with some manipulation of the time series, like the resampling used earlier, or just being more selective in the section of the time series considered for the Complexity-Entropy analysis, and not using the whole time series, might rectify this divergence from the fractional Brownian motion line.

10 Discussion

In section 5 the length requirement of the time series was investigated. The investigation showed that the length of the time series is vital to get the correct results from the Complexity-Entropy analysis. The models which have the strictest length requirement to obtain reasonable results is for models which generate completely random, uncorrelated time series, like the white noise process. While in the other end of the spectrum a linear trend strictly speaking only require the time series to have the length d. This is easy to see, as only one amplitude ordering is accessed by the system all of the time. Therefore, to obtain the true ordinal amplitude permutation probability distribution one only requires one amplitude partition to achieve this. However, systems which can be modelled by a linear curve are of course entirely predictable. To summarize, if the time series one wishes to apply the Complexity-Entropy analysis on is from a system which is completely random, then the required length of the time series is of the order of $5 \cdot 5d!$. This requirement can be relaxed if one has a priori knowledge on the randomness of the system.

The first systems that was analyzed using the Complexity-Entropy analysis was the linear curve and the white noise process in section 6.1. The location of these models in the Complexity-Entropy plane was not surprising. Since only one amplitude permutation is accessed by the system with 100% certainty for a linear curve, the system is entirely predictable. The entropy measure for these systems will be minimized and because of the relation the complexity measure has to the entropy measure, the complexity measure will also be minimized, see section 3.3. Therefore, the location of the linear model in the lower left corner of the Complexity-Entropy plane and the location of the white noise process in the lower right corner of the Complexity-Entropy plane is entirely expected.

Next, the sine function was analyzed in section 6.2. The original location of the sine function was also expected. The sine function contains monotonic regions separated with a region where the signal changes sign of the first derivative and changes to another monotone region. In the regions where the signal is monotone, the entropy measure and complexity measure are minimized as the signal in these regions behaves as a linear curve. It is due to the regions where the signal changes sign of the first derivative that the entropy and complexity measure grow. But for the most part the signal consists of monotone regions, so the entropy measure will be low. The signal is also very structured, so that the calculated entropy and complexity measures are close to the maximum complexity line also makes sense as the complexity measure can be interpreted as a measure of correlation structures. When the discretization time step was synthetically changed by simulating the sine function with a long time series and then introducing resampling lag, the Complexity-Entropy plane location of the sine function spans a curve in a region of the Complexity-Entropy plane, see figure 8. However, using figure 8 to determine the relationship with the calculated entropy and complexity measure and resampling lag was rather difficult, so the entropy and complexity measure was plotted as function of resampling lag where the resampling lag was normalized to the period of the sine function, see figure 11. Here one explicitly sees the relationship of the entropy and complexity measure with the resampling lag. One here discovered that the relationship between the entropy and complexity measures and resampling lag makes symmetric shapes where the entropy and complexity measures oscillate together up and down. It is expected that the overall oscillation with the resampling lag has the same period as the sine function.

In section 6.3 the Lorenz model was analyzed using the Complexity-Entropy analysis. The result of the original analysis is given in figure 13 and when the time series are considered more carefully this result makes sense. The model was originally integrated with a small integration time step which gave smooth curves in the 3D plot of the model time series in figure 12. Since the shape of the oscillations in the time series are preserved, the time series for each variable behaves similar to a sine function. Accordingly, the location of the model time series for each variable in the Complexity-Entropy plane makes sense. Nevertheless, the location in the Complexity-Entropy plane does not reflect the chaotic region of the Lorenz model. The model is indeed chaotic for the model parameters chosen for the simulation. Therefore, the model should be in the upper-middle region of the Complexity-Entropy plane according to previous investigations, but instead the model appears in the region where the sine-function appear. The same resampling analysis technique used on the sine function was applied to the Lorenz model. Using this method, it was possible to determine that in general the entropy level of the model increases with increased lag, see figure 15. The increase in entropy level is not smooth and both the entropy and complexity levels of the model fluctuates with increasing lag, see figure 14. The estimates of the entropy and complexity levels starts off close to the maximum complexity line. But as the lag increases, the model location in the Complexity-Entropy plane starts to deviate from the maximum complexity line and does not return to the line before very high levels of entropy. This is inevitable as the available complexity levels as the entropy gets close to one is quite restricted. To get the explicit picture of how the complexity and entropy measure changes with lag, each measure was plotted as functions of lag for all variables and produced figure 16. Using this figure, it becomes clear that the entropy measure increases with lag and the complexity measure changes accordingly and both measures fluctuate as the lag increases. The Entropy measure for all variables is early maximized and so the complexity measure is minimized, and the model behaves similar to a white noise process. This is not unreasonable, as at that level of lag the distance in time between the datapoints considered is so large that one samples outside of the correlation time of the model. The process is then reduced to a sampling of random numbers and will then mimic the behavior of the white noise process. By using what is already established in the literature about the behavior of the Complexity-Entropy plane and from the locations of the logistic map in section 6.4 and results therein, the chaotic region of the Complexity-Entropy plane is the top-middle part. With figure 14 one can see that the Lorenz system eventually reaches the chaotic region of the Complexity-Entropy plane. Considering then figure 15 the Lorenz model reaches that part of the Complexity-Entropy plane for lags at and around $\sim 100 - 300$ datapoints. This suggests that to observe the chaotic behavior of the Lorenz model, the model needs to be simulated with a discretization timestep of $\Delta t' \sim lags \cdot \Delta t = [100, 300] \cdot 0.001 = [0.1, 0.3]$. However, this increase in discretization timestep does lead to less accurate solutions of the model equations. Plotting the time series then results in a rougher and more jagged plot, but the Complexity-Entropy analysis for the Lorenz model is then consistent with theory.

The next model investigated was the Logistic map. This simple discrete-time difference equation is known to be chaotic for certain ranges of the growth parameter r. Two time series was plotted to demonstrate the different behavior of the logistic map (figure 18), one time series plot was generated with a growth parameter in the non-chaotic regime, in fact it showed the period-2 oscillation of the logistic map. The other time series plot of the logistic map was in the chaotic regime. There, the chaotic behavior of the model is shown. The Complexity-Entropy analysis was applied to time series generated from a range of growth parameter r from 3.5 to 4 with 1000 increments. The 1000 different time series' Complexity-Entropy analysis results are given in figure 19. All entries of the logistic map lay close to the maximum complexity line which indicates that the time series contains correlation structures and that the logistic map time series are very structured. With figure 17 as a guide, most of the Complexity-Entropy locations from the logistic map should lie in the chaotic region of the Complexity-Entropy plane, in the upper-middle region. A more specialized plot for the chaotic regime of the logistic map was then produced from a tiny region which lay entirely in the chaotic regime of the growth parameter, from 3.9 to 4. Even here one encounters simulations from the logistic map which produce complexity and entropy levels which has much lower entropy from what is expected. Inspecting the bifurcation diagram of the Logistic map in figure 17 there are still bands of stability scattered throughout the growth parameter range in the chaotic regime of the parameter. Nevertheless, since the logistic map is a discrete model, one does not need to concern oneself with how fine the discretization time step is simulations will therefore produce time series which has entropy and complexity levels which already lies in the chaotic region of the Complexity-Entropy plane without manipulation of the simulation time series with, for example, resampling. The logistic map can therefore readily be used to show the chaotic region of the Complexity-Entropy plane, and often used in the literature as an example of a chaotic time series [2, 5, 4].

The fractional Brownian motion was simulated for a range of values for the Hurst exponent, from 0.001 to 0.92. The Complexity-Entropy analysis was applied to each of the time series generated using Hurst exponents in the set range and plotted in figure 22. A surprising element in this analysis was the length of the time series and the number of simulation results that had to be averaged in order to obtain the line plotted in figure 22. The extent one had to go to for the final Complexity-Entropy line of the fractional Brownian motion to resample those given in the literature was unexpected and this was not mentioned in the literature. Nevertheless, the result of the Complexity-Entropy analysis was a nice curve in the Complexity-Entropy plane which was similar to that presented in the literature.

For Hurst exponent below 0.5 the fractional Brownian motion is anti-persistent and the probability that consecutive datapoints have the opposite sign is greater than having the same sign. For Hurst exponent above 0.5 the fractional Brownian motion is persistent and the consecutive datapoints are more likely to have the same sign. The latter case is characterized cy clear trends in the simulation time series. The trends in the time series triggered the investigation of the effect of trends in the entropy and complexity measures. The time series was detrended using a running mean approach with increasing window sizes from 2 to 12 datapoints. An unexpected Complexity-Entropy plane entry from this detrending was observed for a window size of 3. In pursuit of an explanation for this odd behavior of the detrending window a detailed investigation was made. The first element of the detrending procedure was an investigation of the averaged time series which was subtracted from the original time series when detrending. Complexity-Entropy plane locations for the averaged time series gave no explanation for the odd location for the detrended time series with window size 3. Next up was the averaging algorithm itself. Originally, a function from the package *Pandas* from Python was used, another averaging algorithm was taken from the UiT GitHub repository. It turned out that these averaging algorithms produced identical time series. Next was an investigation of the algorithm for generating the fractional Brownian motion. The fractional Brownian motion was generated using another package from Python, *FBM*. The special case for Hurst exponent $\mathcal{H} = 0.5$ was used. This represents classical Brownian motion where the increments of this process, that is the fractional Gaussian noise, is represented by normal white noise. A simulation of the fractional Brownian motion, the cumulative sum of a fractional Gaussian noise, and the cumulative sum of a white noise process was done independently. No difference between could then be detected. Thus, the initial investigation could not explain the location for the detrended time series with window size 3.

An interesting observation is that the window size 3 is exactly half of the embedding dimension chosen for the Complexity-Entropy analysis and maybe this could be the reason for this odd behavior. Therefore, the Complexity-Entropy analysis for fractional Brownian motion using different embedding dimension $d = 5$ and $d = 7$ was done. Also, the detrending of the fractional Brownian motion with the same detrending windows as used in figure 24 was redone with these embedding dimensions. To see if this made a difference, the fractional Brownian motion with Hurst exponent $\mathcal{H} = 0.3$ was simulated and detrended with window of size 3. The results from the analysis is presented in figure 44.

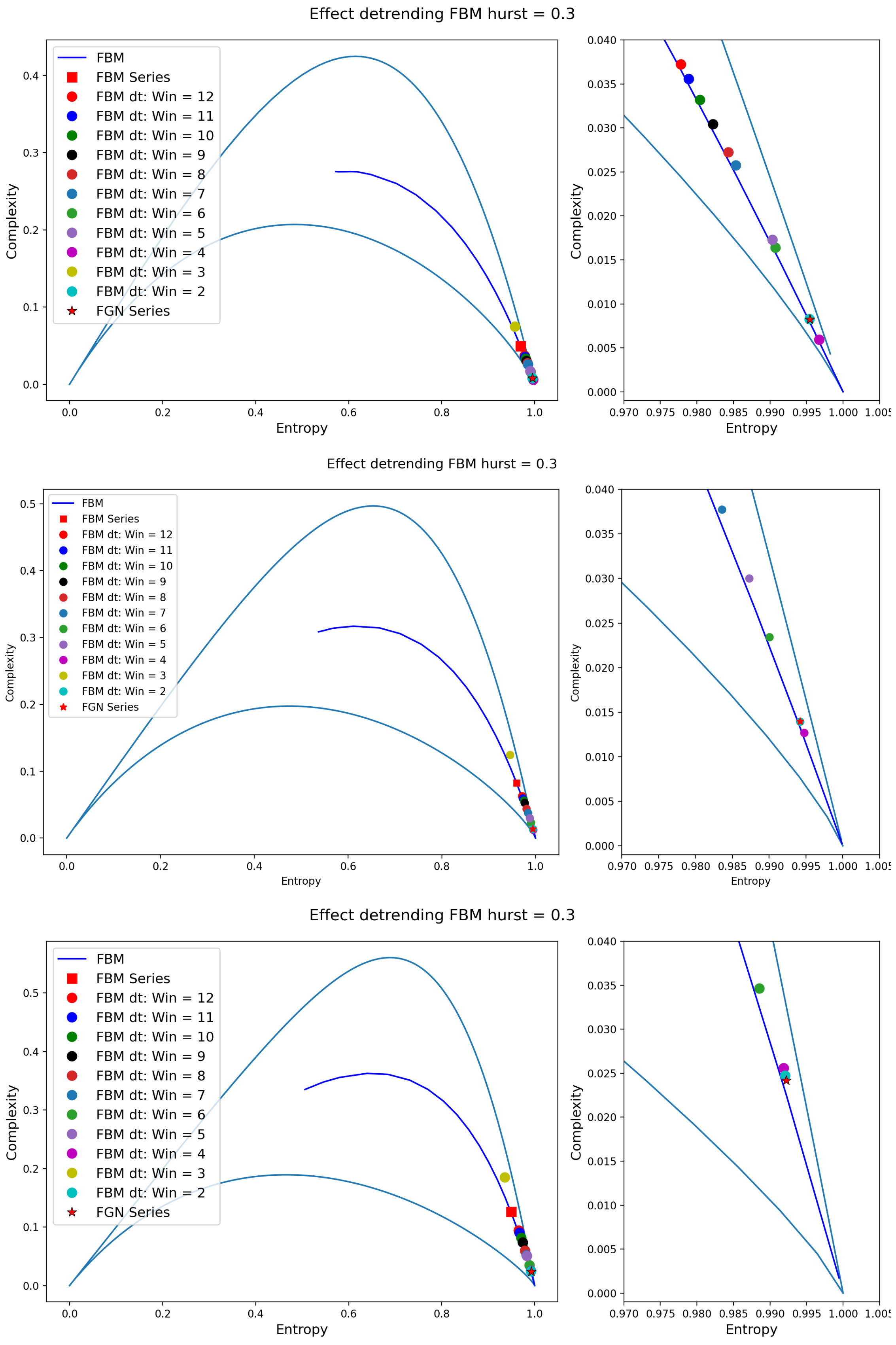


Figure 44: Complexity-Entropy analysis of detrended fractional Brownian motion time series for $\mathcal{H} = 0.3$, with embedding dimension $d = 5$ (top), $d = 6$ (middle), $d = 7$ (bottom).

This too provided no explanation for the location in the Complexity-Entropy plane for the detrended time series with window size 3. The investigation of the behavior of the detrending was unfruitful and the mechanism behind the location in the Complexity-Entropy plane remains unknown for the time being. Nevertheless, for all the other window sizes the detrending works fine. When the fractional Brownian motion process was detrended, the noise process was recovered which is reflected in the figures as the entropy measure of the processes increases.

From the analysis of the filtered Poisson process with constant pulse duration one found that, except for the very lowest intermittency parameters, increasing the intermittency parameter led to an increase in the entropy measure of the simulation time series, see figure 29. The oddities for the lowest intermittency parameter γ is thought to be caused by the large distances between each pulse that the time series is dominated by regions of nearly constant and low amplitudes, which cant be said to have structure, see figure 28. When a pulse does arrive, it created enough random behavior for the entropy measure to be different form zero but not enough for the time series to possess enough structures to move away from the minimum complexity line. Changing the pulse shape from a one-sided exponential pulse to a Lorentzian shaped pulse to simulate the fluctuations described in the literature, the same behavior of the intermittency parameter for the complexity and entropy measures was found, see figure 31. However, because of the fine discretization timestep used in the initial simulations, and the smooth rise and fall of the Lorentzian pulse shapes, figure 30, the difference between the entropy measures as γ increases was less noticeable than for the one-sided exponential pulse shape. The same analysis technique that was used on the sine function and the Lorenz model earlier was applied to the filtered Poisson process with both pulse shapes. What was found was that as the time series was resampled, the Complexity-Entropy plane entries for increasing lag spanned a curve in the Complexity-Entropy plane. The thickness of the curve was small for both cases, but for the Lorentzian shaped pulses, the curve spanned for the different intermittency parameters overlapped almost perfectly, see figure 36. The only difference was how low the entropy measure was for the starting time series. The interesting part here is that the pulse shapes alone may be the defining feature on which curve one ends up in the Complexity-Entropy plane. The discretization timestep and the intermittency parameter defines where on the curve one ends up. Looking at the Complexity-Entropy plane analysis results for the Pareto distributed duration times may indicate that the duration time distribution has little or no effect on the spanning curve of the filter Poisson process in the Complexity-Entropy plane.

The Pareto distributed duration times was introduced into the filtered Poisson process to obtain long-range correlations in the time series, a common property of self-similar systems rooted in self-organized criticality. Here, an important simulation parameter for the Pareto distribution was the scaling exponent α. The parameter α tells one how steep the line of the Pareto distribution is in a log-log plot. The data for the Complexity-Entropy analysis was sourced from the research group at UiT, but presented in a way to limit the parameters which are known to change the location of the model in the Complexity-Entropy plane, the discretization time step and the intermittency parameter. In this case, for each value of α the discretization timestep was held constant for the simulations with changing intermittency parameter. The Complexity-Entropy analysis did not provide any concrete trends and behaviors from where the different simulations ended up in the Complexity-Entropy plane. In general, one can say that for increasing α the mean entropy measure increases. As the parameter α increased in value and increasing number of model simulations ended up close to the location for white noise, in the lower right corner of the Complexity-Entropy plane. This holds true for both pulse shapes. To say anything more than this becomes quite difficult. The already established behavior for the intermittency parameter had changed completely. For some of the values of α the location of the simulation of the largest intermittency parameter achieved one of the lowest measured entropy levels, see simulations of $\alpha = 1.4$ in figure 39 for both pulse shapes. The addition of a probability distribution on the duration time of the pulses was expected to change the location of the simulations in the Complexity-Entropy plane. Now the measures of entropy and complexity was not only a function (in the case for the filtered Poisson process) of the intermittency parameter and the discretization timestep, but also on the Pareto distribution and its parameters. But to completely neglect the effect of the intermittency parameter found in the results was unexpected to say the least. Looking more closely at the Complexity-Entropy plane locations of the different simulations and the increasing α's, the entries seem to span a curve in the Complexity-Entropy plane as it did for the constant duration time simulations of the filtered Poisson process. All entries for all α's was superposed together and produced figure 40. From the figure, they do seem to span a curve. Looking more closely at the plot for one-sided exponential pulse shape, as that pulse shape was also used for the constant duration time case, the curve spanned for the Pareto distributed duration times achieved slightly lower complexity for the same levels of entropy compared to the constant duration time case. Which is also plotted in the same figure as the thick light-blue line. Further research will be required to verify the results, but the implication of this is quite clear. It does not seem to matter, to some extent, what the distribution of duration times is for the curves spanned by the filtered Poisson process. The thickness of the band set for the one-sided exponential pulse shape for the constant duration time and Pareto distributed duration time is not that thick. The only variable that seems to matter the most for the curves from the filtered Poisson process in the Complexity-Entropy plane is the pulse function, and not the distribution of duration times.

The BTW model was also used as a model that exhibit self-organized criticality. These sandpile models have relatively simple toppling rules but exhibit complex dynamics. The simulation data for these models was also repurposed to be used in the Complexity-Entropy analysis. The results confirm what is described in the literature, that these models can be described by the fractional Brownian motion process, see figure 43. The only exception to this happened for the forest fire model, *BTW 6*. But the deviation from the fractional Brownian motion line is likely due to the different dynamics of the forest fire model compared to the classical sandpile model.

To conclude the discussion, the results presented and discussed above shows that stochastic and chaotic models do indeed appear in different regions of the Complexity-Entropy plane. With the results from the different models one can identify each region of the Complexity-Entropy plane. The lower left corner of the Complexity-Entropy plane is dominated by linear trends. From the lower left corner up to about entropy level $H = 0.4$ and close to the maximum complexity line is dominated by periodic oscillations, like the sine function. The lower right corner of the Complexity-Entropy plane is occupied by noise processes, where the white noise process lies in the right corner with entropy value $H = 1$ and complexity value $C = 0$. The fractional Brownian motion defines a line in the Complexity-Entropy plane from $H \approx 0$ to about $H \approx 0.5$ and $C \approx 0$ to $C = 0.3$. Following figure 4 in the paper by Maggs et al. [5], the fractional Brownian motion line can be used as a guide to distinguish models of stochastic and chaotic nature. The idea is that the fractional Brownian motion line represents one of the stochastic models which obtains one of the highest measures of complexity for a given entropy level. Then, imagining a line slightly above the fractional Brownian motion line and completing the curve across the entire Complexity-Entropy plane. Any model which reaches complexity and entropy measures above this new imagined line can be considered chaotic, while every model with entropy and complexity levels below this line is stochastic. This is backed up by the result from the analysis if this thesis. The model which readily shows the chaotic region of the Complexity-Entropy plane is the logistic map, which showed that the chaotic region of the Complexity-Entropy plane is in the upper-middle part of the plane.

Continuous-time models should be represented by a curve in the Complexity-Entropy plane. Where on this curve the specific models appear will depend on the discretization timestep used when simulating the models and possibly on model parameters, such as the filtered Poisson process. In addition, the Complexity-Entropy plane has demonstrated the important property of being able to separate processes with trends and noise. The fact that this analysis tool has the ability to separate stochastic and chaotic processes and processes with trends and noise without adjustable analysis parameters is a powerful result, while still being computationally feasible to undertake. This result alone should be enough to consider the analysis tool in further research. Below a list is complied with suggestions for further research based on the results presented in this thesis:

- Continue simulation of FPP with Pareto distributed duration times to see if the model locations in the Complexity-Entropy plane presented in figure 39 have converged to their appropriate locations.

- Continue simulations of BTW models to see if they too have converged to their appropitate locations. In addition, methods to estimate the Hurst exponent of each BTW model and double-check if the location of the BTW model matches the fractional Brownian motion process with the equivalent Hurst exponent.

- Expand the models considered, for example models defined by stochastic differential equations like the Ornstein-Uhlenbeck processes, $dx_t = \theta(\mu - x_t)\, dt + \sigma\, dW_t$. Initial analysis showed interesting results that the Ornstein-Uhlenbeck process is equivalent to classical Brownian motion in the Complexity-Entropy plane, (occupies the same point in the plane), see figure 45.

- Carry the investigation of the detrended fractional Brownian motion process to try and present an explanation of the odd location of the detrended version of the fractional Brownian motion process with window size 3, figure 24.

- A simpler model to investigate the effect of trends in the signal. Instead of using the fractional Brownian motion process to investigate the trends, one could use a simple linear mode with added noise: $f(t) = \alpha t + \sigma W$. Then use this model to detrend with a running mean and investigate how the detrending affects the Complexity-Entropy plane location of the model. Here one could also investigate the effect of the signal-to-noise ratio on the complexity and entropy measure.

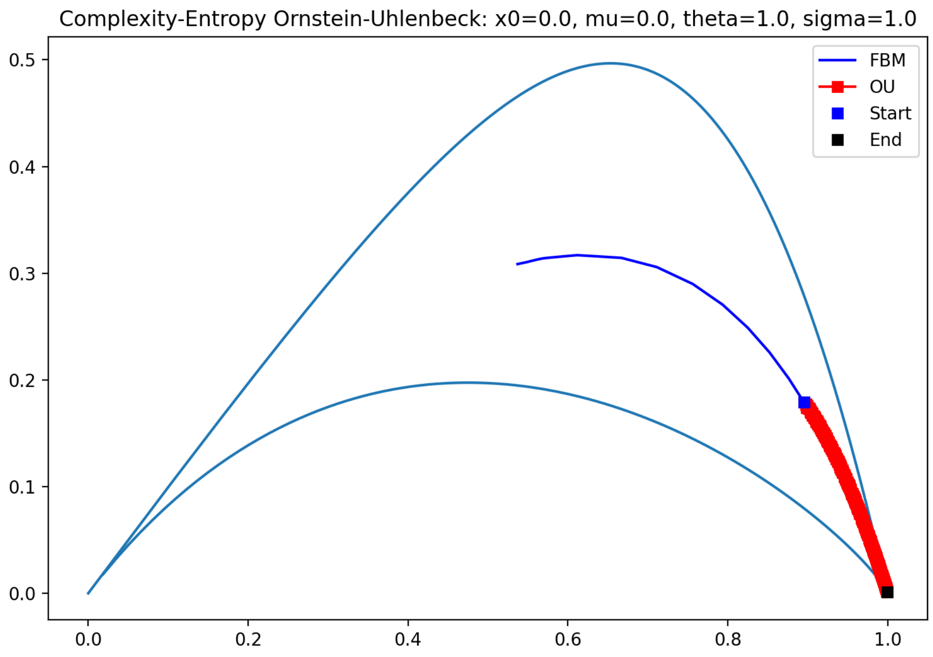


Figure 45: Complexity-Entropy analysis of an Ornstein-Uhlenbeck process with model parameters $\mu = 0$, $\theta = 1.0$, $\sigma = 1.0$.

11 Conclusion

The result of the analysis showed that the Complexity-Entropy plane has the ability to distinguish between stochastic and chaotic models based on their location in the Complexity-Entropy plane. The Complexity-Entropy analysis of continuous-time models should be represented by lines and curves in the Complexity-Entropy plane, rather than what is implied in the literature where these models are represented by specific points. By applying the Complexity-Entropy analysis on the fractional Brownian motion, and then comparing the location of the model after the process is detrended showed that the Complexity-Entropy analysis will distinguish between models with trends and models representing noise. Applying the Complexity-Entropy analysis on the filtered Poisson process showed that this model is represented by curves in the Complexity-Entropy plane. The dominant determining factor as to which curve the model lies on was showed to be the pulse shape, the pulse duration time distribution had little or no effect on the curve. The Complexity-Entropy analysis of the Bak-Tang-Wiesenfeld models produced confirming results to that described in the literature, that these models can be described by a fractional Brownian motion process.

The ability of the Complexity-Entropy analysis to distinguish between models from stochastic and chaotic nature, and also being able to separate models with trends and models represented by noise shows that the Complexity-Entropy analysis has excellent potential as a time series analysis tool. That the analysis tool also is non-parametric and computationally feasible suggests that the Complexity-Entropy analysis can and should be included in an initial test battery for time series analysis along with more common methods like power spectral density and so on.

12 Appendix A

Source code for the thesis

```python
import numpy as np
import itertools
import math
from tqdm import tqdm


class ComplexityEntropy():
    ''' Class containing appropriate functions to calculate the complexity and entopy values for a given time sereis '''

    def __init__(self, time_series, d, tau = 1):
        '''


        Parameters
        ----------
        time_series : LIST / ARRAY
            Time series
        d : INT
            Embedding dimension
        tau : INT, optional
            Embedding delay. The default is 1.

        Returns
        -------
        None. Initializing function

        '''
        self.time_series = np.array(time_series)
        self.d = d
        self.tau = tau


    def Permutation_frequency(self):
        '''
        Function that calculates the relative frequnecy for the different permutations
        of the time series for the given embedding dimension

        Returns
        -------
        relative_frequency/relative_frequency_pad : ARRAY
            Probability distribution for the possible ordinal patterns, Padded with
            zeros to have length d!

        '''
        Possible_permutaitons = math.factorial(self.d) #list(itertools.permutations(range(self.d)))
        perm_list = []

        # subsample the time series
        self.time_series = self.time_series[::self.tau]

        for i in range(len(self.time_series) - self.d + 1):
            # "permutation of emb_dimension sized segments of the time series"
            permutation = list(np.argsort(self.time_series[i : (self.d + i)]))
            perm_list.append(permutation)

        # Find the different permutations and calculates their number of appearance
        elements, frequency = np.unique(np.array(perm_list), return_counts = True, axis = 0)

        # Divides by the total number of permutations, gets relative frequency / "probalbility" of appearance
        relative_frequency = np.divide(frequency, (len(self.time_series) - self.tau * (self.d - 1)))

        # If the two arrays do not have the same shape, add zero padding to make their lengths equal
        if len(relative_frequency) != Possible_permutaitons:
                relative_frequency_pad = np.pad(relative_frequency, (0, int(Possible_permutaitons - len(relative_frequency))), mode = 'constant')
                return relative_frequency_pad

        else:
            return relative_frequency

    def Permutation_Entropy(self, Permutation_probability):
        '''
        Function to calculate the permutation entropy for a given probability distribution
        In this case, it retruns the normalized shannon entropy

        Parameters
        ----------
        Permutation_probability : ARRAY
            Array contaning the probalbility distribution of the ordinal patterns.

        Returns
        -------
        permutation_entropy : FLOAT
            Entropy value of the time series.

        '''
        permutation_entropy = 0.0

        # Calculate the max entropy, max = log(d!)
        max_entropy = np.log2(len(Permutation_probability))

        for p in Permutation_probability:
            if p != 0.0:
                permutation_entropy += p * np.log2(p)
        return - permutation_entropy/max_entropy

    def Shannon_Entropy(self, Permutation_probability):
        '''
        Regular Shannon entropy, not normalized

        Parameters
        ----------
        Permutation_probability : ARRAY
            Array contaning the probalbility distribution of the ordinal patterns.

        Returns
        -------
        shannon_entropy : FLOAT
            Shannon entropy value.

        '''
        shannon_entropy = 0.0
        for p in Permutation_probability:
            if p != 0.0:
                shannon_entropy += p * np.log2(p)
        return -shannon_entropy

    def Jensen_Shannon_Complexity(self, Permutation_probability):
        '''
        Function to calculate the Jensen-Shannon complexity value for the time sereis

        Parameters
        ----------
        Permutation_probability : ARRAY
            Array contaning the probalbility distribution of the ordinal patterns.

        Returns
        -------
        jensen_shannon_complexity : FLOAT
            Jensen-Shannon complexity value.

        '''
        P = Permutation_probability
        N = len(P)
        C1 = (N + 1)/N * np.log2(N + 1)
        C2 = 2 * np.log2(2*N)
        C3 = np.log2(N)
        PE = self.Permutation_Entropy(P)

        P_uniform = []
        for i in range(N):
            P_uniform.append(1/N)

        JS_div = self.Shannon_Entropy((P + P_uniform)*0.5) - 0.5 * self.Shannon_Entropy(P) - 0.5 * self.Shannon_Entropy(P_uniform)
        jensen_shannon_complexity = -2 * (1/(C1 - C2 + C3)) * JS_div * PE
        return jensen_shannon_complexity

    def CH_plane(self):
        '''
        Computes the permutation entropy and the Jensen-Shannon complexity for the time series
        with the functions defined in the class

        Returns
        -------
        permutation_entropy : FLOAT
            Permutation entropy value of the time series.
        jensen_shannon_complexity : FLOAT
            Jensen-Shannon complexity value of the time series.

        '''
        # Calling the function to generate the relative frequency for the ordinal patterns
        relative_frequency = self.Permutation_frequency()

        #Using relative frequency to calculate the entropy/complexity for the time series
        permutation_entropy = self.Permutation_Entropy(relative_frequency)
        jensen_shannon_complexity = self.Jensen_Shannon_Complexity(relative_frequency)

        return permutation_entropy, jensen_shannon_complexity


class MaxMin_complexity(ComplexityEntropy):
    '''
    Class containing the fuctions to calculate the maximum complexity and
    minimum complexity lines for the Complexity-Entropy plane with
    embedding dimension d
    '''

    def __init__(self, d, n_steps = 500):
        '''

        Parameters
        ----------
        d : INT
            Embedding dimension.

        Returns
        -------
        None.

        '''
        # definin class variables available to all functions/methods contained in this class
        self.d = d
        self.N = math.factorial(self.d)
        self.n_steps = n_steps
        self.d_step = (1 - 1/self.N) / (self.n_steps)

        # Initilalizing __init__()-function to parent (ComplexityEntropy class)
        # class to make the functions contained in that class available to this class
        super().__init__(time_series = None, d = self.d)

        # Lists to contain the x and y values for the minimum and maximum
        # complexity lines
        self.min_complexity_entropy_x = list()
        self.min_complexity_entropy_y = list()
        self.max_complexity_entropy_x = list()
        self.max_complexity_entropy_y = list()


    def Minimum(self):
        '''
        Function to calculate the minimum complexity line

        Returns
        -------
        min_complexity_entropy_x : LIST
            x-values for the minimum complexity line.

        min_complexity_entropy_y : LIST
            y-values for the minimum complexity line.

        '''
        p_min = list(np.arange(1/self.N, 1, self.d_step))
        for n in tqdm(range(len(p_min)), desc='Minimum', ncols=70):
            P_minimize = []
            if p_min[n] > 1:
                p_min[n] = 1
            P_minimize.append(p_min[n])
            for i in range(self.N - 1):
                p_rest = (1 - p_min[n]) / (self.N - 1)
                P_minimize.append(p_rest)

            # Convert from list structure to array structure
            P_minimize = np.array(P_minimize)

            # Adding the calculated x and y (entropy and complexity) values
            # to their approppriate lists
            self.min_complexity_entropy_x.append(self.Permutation_Entropy(P_minimize))
            self.min_complexity_entropy_y.append(self.Jensen_Shannon_Complexity(P_minimize))

        return self.min_complexity_entropy_x, self.min_complexity_entropy_y

    def Maximum(self):
        '''
        Function to calculate the maximum complexity line

        Returns
        -------
        max_complexity_entropy_x : FLOAT
            x-values for the maximum complexity line.

        max_complexity_entropy_y : FLOAT
            y-values for the maximum complexity line.

        '''

        for n in tqdm(range(self.N - 1), desc='Maximum', ncols=80):
            p_max = list(np.arange(0, 1 / (self.N - n), self.d_step))

            for m in range(len(p_max)):
                P_maximize = list()
                P_maximize.append(p_max[m])
                p_rest = (1 - p_max[m]) / (self.N - n - 1)
                for i in range(self.N - n - 1):
                    P_maximize.append(p_rest)

                if len(P_maximize) != self.N:
                    P_maximize = np.pad(P_maximize, (0, n), mode = 'constant')

                # Convert from list structure to array structure
                P_maximize = np.array(P_maximize)

                #Adding the calculated x and y (entropy and complexity) values
                # to their approppriate lists
                self.max_complexity_entropy_x.append(self.Permutation_Entropy(P_maximize))
                self.max_complexity_entropy_y.append(self.Jensen_Shannon_Complexity(P_maximize))

        return self.max_complexity_entropy_x, self.max_complexity_entropy_y
```

References

[1] Christoph Bandt and Bernd Pompe. Permutation entropy: A natural complexity measure for time series. *Phys. Rev. Lett.*, 88:174102, Apr 2002. doi:10.1103/PhysRevLett.88.174102.

[2] O. A. Rosso, H. A. Larrondo, M. T. Martin, A. Plastino, and M. A. Fuentes. Distinguishing noise from chaos. *Phys. Rev. Lett.*, 99:154102, Oct 2007. doi:10.1103/PhysRevLett.99.154102.

[3] Z. Zhu, A. E. White, T. A. Carter, S. G. Baek, and J. L. Terry. Chaotic edge density fluctuations in the alcator c-mod tokamak. *Physics of Plasmas*, 24(4):042301, 2017. doi:10.1063/1.4978784.

[4] M.T. Martin, A. Plastino, and O.A. Rosso. Generalized statistical complexity measures: Geometrical and analytical properties. *Physica A: Statistical Mechanics and its Applications*, 369(2):439 – 462, 2006. doi:https://doi.org/10.1016/j.physa.2005.11.053.

[5] J E Maggs and G J Morales. Permutation entropy analysis of temperature fluctuations from a basic electron heat transport experiment. *Plasma Physics and Controlled Fusion*, 55(8):085015, jun 2013. doi:10.1088/0741-3335/55/8/085015.

[6] P. J. Weck, D. A. Schaffner, M. R. Brown, and R. T. Wicks. Permutation entropy and statistical complexity analysis of turbulence in laboratory plasmas and the solar wind. *Phys. Rev. E*, 91:023101, Feb 2015. doi:10.1103/PhysRevE.91.023101.

[7] L Fattorini, Å Fredriksen, H L Pécseli, C Riccardi, and J K Trulsen. Turbulent transport in a toroidal magnetized plasma. *Plasma Physics and Controlled Fusion*, 54(8):085017, jul 2012. doi:10.1088/0741-3335/54/8/085017.

[8] O. E. Garcia. Stochastic modeling of intermittent scrape-off layer plasma fluctuations. *Phys. Rev. Lett.*, 108:265001, Jun 2012. doi:10.1103/PhysRevLett.108.265001.

[9] Massimiliano Zanin. Forbidden patterns in financial time series. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(1):013119, 2008. doi:10.1063/1.2841197.

[10] E Olofsen, J Sleigh, and Albert Dahan. Permutation entropy of the electroencephalogram: a measure of anaesthetic drug effect. *British journal of anaesthesia*, 101:810–21, 10 2008. doi:10.1093/bja/aen290.

[11] Sayan Mukherjee, Sanjay Kumar Palit, Santo Banerjee, M.R.K. Ariffin, Lamberto Rondoni, and D.K. Bhattacharya. Can complexity decrease in congestive heart failure? *Physica A: Statistical Mechanics and its Applications*, 439:93–102, 2015. doi:https://doi.org/10.1016/j.physa.2015.07.030.

[12] Denis Jordan, Gudrun Stockmanns, Eberhard Kochs, Stefanie Pilge, and Gerhard Schneider. Electroencephalographic order pattern analysis for the separation of consciousness and unconsciousness an analysis of approximate entropy, permutation entropy, recurrence rate, and phase coupling of order recurrence plots. *Anesthesiology*, 109:1014–22, 01 2009. doi:10.1097/ALN.0b013e31818d6c55.

[13] Yinhe Cao, Wen-wen Tung, J. B. Gao, V. A. Protopopescu, and L. M. Hively. Detecting dynamical changes in time series using the permutation entropy. *Phys. Rev. E*, 70:046217, Oct 2004. doi:10.1103/PhysRevE.70.046217.

[14] Xiaoli Li, Gaoxian Ouyang, and Douglas A. Richards. Predictability analysis of absence seizures with permutation entropy. *Epilepsy Research*, 77(1):70–74, 2007. doi:https://doi.org/10.1016/j.eplepsyres.2007.08.002.

[15] Gaoxiang Ouyang, Xiaoli Li, Chuangyin Dang, and Douglas A. Richards. Deterministic dynamics of neural activity during absence seizures in rats. *Phys. Rev. E*, 79:041146, Apr 2009. doi:10.1103/PhysRevE.79.041146.

[16] M. Riedl, A. Müller, and N. Wessel. Practical considerations of permutation entropy. *The European Physical Journal Special Topics*, 222(2):249–262, Jun 2013. doi:10.1140/epjst/e2013-01862-7.

[17] P.W Lamberti, M.T Martin, A Plastino, and O.A Rosso. Intensive entropic non-triviality measure. *Physica A: Statistical Mechanics and its Applications*, 334(1):119–131, 2004. doi: https://doi.org/10.1016/j.physa.2003.11.005.

[18] L. Zunino, M. C. Soriano, and O. A. Rosso. Distinguishing chaotic and stochastic dynamics from time series by using a multiscale symbolic approach. *Phys. Rev. E*, 86:046210, Oct 2012. doi:10.1103/PhysRevE.86.046210.

[19] Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20(2):130 – 141, 01 Mar. 1963. doi:10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

[20] Steven H. (Steven Henry) author Strogatz. *Nonlinear dynamics and chaos : with applications to physics, biology, chemistry, and engineering*. Second edition. Boulder, CO : Westview Press, a member of the Perseus Books Group, [2015], 2015. Includes bibliographical references and index. URL: https://search.library.wisc.edu/catalog/9910223127702121.

[21] Robert May. Simple mathematical models with very complicated dynamics. *Nature*, 26:457, 07 1976. doi:10.1038/261459a0.

[22] Benoit B. Mandelbrot and John W. Van Ness. Fractional brownian motions, fractional noises and applications. *SIAM Review*, 10(4):422–437, 1968. URL: http://www.jstor.org/stable/2027184.

[23] Christoph Bandt and Faten Shiha. Order patterns in time series. *Journal of Time Series Analysis*, 28:646–665, 09 2007. doi:10.1111/j.1467-9892.2007.00528.x.

[24] O. E. Garcia and A. Theodorsen. Auto-correlation function and frequency spectrum due to a super-position of uncorrelated exponential pulses. *Physics of Plasmas*, 24(3):032309, 2017. doi:10.1063/1.4978955.

[25] O. E. Garcia, R. Kube, A. Theodorsen, and H. L. Pécseli. Stochastic modelling of intermittent fluctuations in the scrape-off layer: Correlations, distributions, level crossings, and moment estimation. *Physics of Plasmas*, 23(5):052308, 2016. doi:10.1063/1.4951016.

[26] A. Theodorsen and O. E. Garcia. Level crossings, excess times, and transient plasma–wall interactions in fusion plasmas. *Physics of Plasmas*, 23(4):040702, 2016. doi:10.1063/1.4947235.

[27] Magdalena A. Korzeniowska. Intermittent fluctuations in pysical systems. Portfolio assignment in specialized curriculum course given at UiT The Artic University of Tromsø, Department of Physics and Technology, autumn 2020. Unpublished, 2020.

[28] Juan M. Losada. Sandpile models and soc. Portfolio assignment in specialized curriculum course given at UiT The Artic University of Tromsø, autumn 2020. Unpublished, 2020.

[29] M. Rypdal and K. Rypdal. Modeling temporal fluctuations in avalanching systems. *Phys. Rev. E*, 78:051127, Nov 2008. doi:10.1103/PhysRevE.78.051127.

[30] M Rypdal and K Rypdal. A stochastic theory for temporal fluctuations in self-organized critical systems. *New Journal of Physics*, 10(12):123010, dec 2008. doi:10.1088/1367-2630/10/12/123010.