PARAMETER ESTIMATION OF CHAOTIC SYSTEMS USING PYTHON LIBRARY

Lappeenranta–Lahti University of Technology LUT

Degree Programme in Computational Engineering, Bachelor's thesis

2024

Max Wesamaa

Examiners: Academy research fellow Andreas Rupp

professor Heikki Haario

Abstract

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Degree Programme in Computational Engineering

Max Wesamaa

Parameter estimation of chaotic systems using Python library

Bachelor's thesis

2024

22 pages, 9 figures

Examiners: Academy research fellow Andreas Rupp and professor Heikki Haario

Keywords: parameter estimation, Lorenz system, Chaos theory

With empirical cumulative distribution function (eCDF) we are able to estimate the parameters of chaotic systems. The aim of this thesis is to estimate the parameters in a chaotic system, more specifically, the Lorenz system. This is done using the eCDF method coupled with the *ecdf_estimator* Python library.

The data is generated using Euler method. When creating this data, we change the values of one parameter at a time. This parameter is the one we are estimating. The Python library works well with the data and all the required tests and tasks using the library were performed without any issues.

We were able to estimate all three parameters to the desired values. The results demonstrate that even though chaotic, it is possible to estimate the parameters in the Lorenz system. They also show that the Python library, which originally was made for estimating the parameters of cellular automata, can be used for estimating the parameters in a chaotic system.

Tiivistelmä

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUTin insinööritieteiden tiedekunta

Laskennallisen tekniikan koulutusohjelma

Max Wesamaa

Parametrien estimointi kaoottisissa järjestelmissä Python-kirjaston avulla

Kandidaatintyö

2024

22 sivua, 9 kuvaa

Tarkastajat: Akatemiatutkija Andreas Rupp ja professori Heikki Haario

Avainsanat: parametriestimointi, Lorenzin yhtälöt, Kaaosteoria

Empiirisen kertymäfunktion (eCDF) avulla voimme arvioida kaoottisten järjestelmien parametreja. Tämän opinnäytetyön tavoitteena on arvioida parametrien arvoja kaoottisessa järjestelmässä, joka tällä kertaa osoittautui Lorenzin yhtälöiksi. Tämä tehdään eCDF-menetelmää hyödyntämällä, sekä *ecdf_estimator* kirjaston avulla.

Data luodaan Eulerin menetelmällä. Datan luomisen yhteydessä muutamme yhden parametrin arvoa kerrallaan. Tämä parametri on se, jota arvioidaan. Python-kirjasto toimi moitteettomasti datan kanssa, ja kaikki tarvittavat testit sekä tehtävät suoritettiin ongelmitta kirjaston avulla.

Onnistuimme arvioimaan kaikki kolme parametria haluttuihin arvoihin. Tulokset osoittavat, että vaikka järjestelmä on kaoottinen, on mahdollista arvioida Lorenzin yhtälöiden parametreja. Tulokset osoittavat myös, että Python-kirjastoa, joka alun perin luotiin soluautomaattien parametrien arviointiin, voidaan käyttää kaoottisten järjestelmien parametrien arviointiin.

Acknowledgements

I would like to thank my supervisors Heikki Haario and Andreas Rupp. Andreas was able to give me great insights on his Python library and I am forever grateful for the opportunity he gave me to learn about the Lorenz system. I am also grateful for Heikki's contributions to the parameter estimations and understanding the Lorenz system. This thesis was a very interesting topic and I'm glad I chose it. A big thank you to both of my supervisors for giving me this opportunity and giving me the help I needed in this thesis.

Table of contents

List of Figures

1 Introduction

Chaotic systems are unpredictable nonlinear dynamic systems that have a sensitive dependence on initial conditions (Britannica, 2024). The way these systems behave can be depicted visually as movement on what is known as an attractor (Britannica, 2024). The attractors in chaotic systems are often called strange attractors. In this context chaotic means unpredictability. Strange attractors are attractors that are sensitive to even the slightest changes in initial conditions (Manning, 2017).

In this thesis, we will be using a Python library, *ecdf_estimator*, to estimate the parameters in the Lorenz system, the library can be found in (Rupp, 2022). The library was originally for identifying cellular automata parameters (Kazarnikov et al., 2023), but it is possible to use it for this thesis subject too.

Parameter estimation in chaotic systems can be done in different ways, one way, which is the one we will also be using, is by using empirical cumulative distribution functions (eCDFs). The core principle of this method is to analyze the statistics of eCDFs which are derived from the training data produced by a specific model, which in this case is the Lorenz system, with the goal of identifying the model's parameters.

Estimating parameters in chaotic systems using eCDF-vectors has been demonstrated in (Springer et al., 2019), and estimating the parameters in specifically the Lorenz system has also been proven to work with this method in (Haario, Kalachev & Hakkarainen, 2015). Both (Springer et al., 2019), and (Haario, Kalachev & Hakkarainen, 2015) use Markov chain Monte Carlo (MCMC) sampling to get the parameter posteriors after the parameter estimation. MCMC sampling could be an exciting opportunity for future research on this thesis' topic.

1.1 Lorenz system

The Lorenz system was studied by a meteorologist named Edward Lorenz, and hence, it was named after him (Manning, 2017). This was the first system where chaos was observed (Manning, 2017). The Lorenz equations are as follows

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y-x) \\ \frac{dy}{dt} &= x(\rho-z)-y \\ \frac{dz}{dt} &= xy-\beta z \end{aligned} \tag{1}$$

,where σ>0 is the Prandtl number, ρ>0 is the Rayleigh number, and β doesn't have a name but it is proportional to the dimensions of the fluid layer (Manning, 2017). The equations for the variables are as follows:

$$\sigma = \frac{v}{a} \quad = \frac{\text{viscous diffusion rate}}{\text{thermal diffusion rate}} \tag{2}$$

$$\rho = Gr_x Pr_x \tag{3}$$

, where in equation (2) v, measured in m^2/s, is the momentum diffusivity, and a, also measured in m^2/s, is the thermal diffusivity (Manning, 2017). In equation (3) Gr_x is the Grashof number, and Pr_x is the Prandtl number which is described above (Manning, 2017).

To keep things simple in future references, the Prandtl number will be σ, Rayleigh number will be ρ, and beta will be β (Manning, 2017).

The following chapter is based on information found in (Manning, 2017). The Lorenz system has various properties, the first of which is being nonlinear. This can be seen in the second equation as xz, and in the third equation as xy. Nonlinearity gives the system a nature in which changes to the initial conditions aren't proportional to the outputs of the system. It's also symmetrical on the x and y axis, meaning that if a point (x,y,z) has a solution in the system, the point (-x,y,z) also has a solution in the system. Another property of the system is volume contraction. This means the system will always remain confined within a limited set of points indefinetely.

The Lorenz system is a strange attractor, meaning is has a fractal structure. As stated before, the Lorenz system is chaotic, and has no noticeable pattern to its behaviour. This means that there aren't any points that are visited more than once within the same trajectory, otherwise you could make a prediction on the pattern. It has a very unique butterfly shape.

1.2 Goals

The goal for this thesis is to create our own data with which we are able to estimate the values for the three parameters in the Lorenz system. This will all be done using the Python library made by my supervisor Andreas Rupp.

1.3 Structure

The thesis structure: Section 2 introduces the research methods which has subsections about creating the data, the estimation approach, and the *ecdf_estimator* Python-library. Section

3 discusses evaluating the training data. In Section 4 we estimate the three parameters. To finish, we have the conclusion in Section 5.

2 Research methods

The Python code for creating and simulating the Lorenz attractors is based on (Matplotlib, 2024). The code for estimating the parameters is based on an example implementation of the Python library found in (Rupp, 2022). Creating of the data will be decsribed in more detail in Section 2.1, and an introduction to the Python library is found in Section 2.2.

2.1 Data

The data in this project is created in Python using two libraries, matplotlib, and numpy. The program calculates an amount of points given by the user in a Lorenz system according to the users given values for σ, ρ, and β. While doing this, the program changes the initial values for x, y, and z with in interval of ± 0.1. This is repeated for a number of epochs (also given by the user, and it has to be an equal value) to obtain many Lorenz systems that vary from each other. If the user wishes to plot the systems, the program can also do this.

As stated, the user is able to choose the amount of points and epochs. In this thesis, we will be calculating 111 points in 50 epochs.

The amount of epochs has to be an equal value because the next step is to divide them into pairs and each pairs' Euclidean distance is calculated for each point. Before the distances are computed, we drop the first 51 points from each epoch. This is done because sometimes the attractors may have relatively small distances with each other's points during the start. Using the Euclidean distances, we can compute the eCDF-vectors of the distances, which are then used to obtain the statistical multinormal distribution by calculating the mean and covariance of the eCDF-vectors. When we have the data in the form of a statistical multinormal distribution, we can use the ecdf-estimator Python-library to estimate the parameters. A more detailed explanation of the commands used to do all of this is given in section 3.

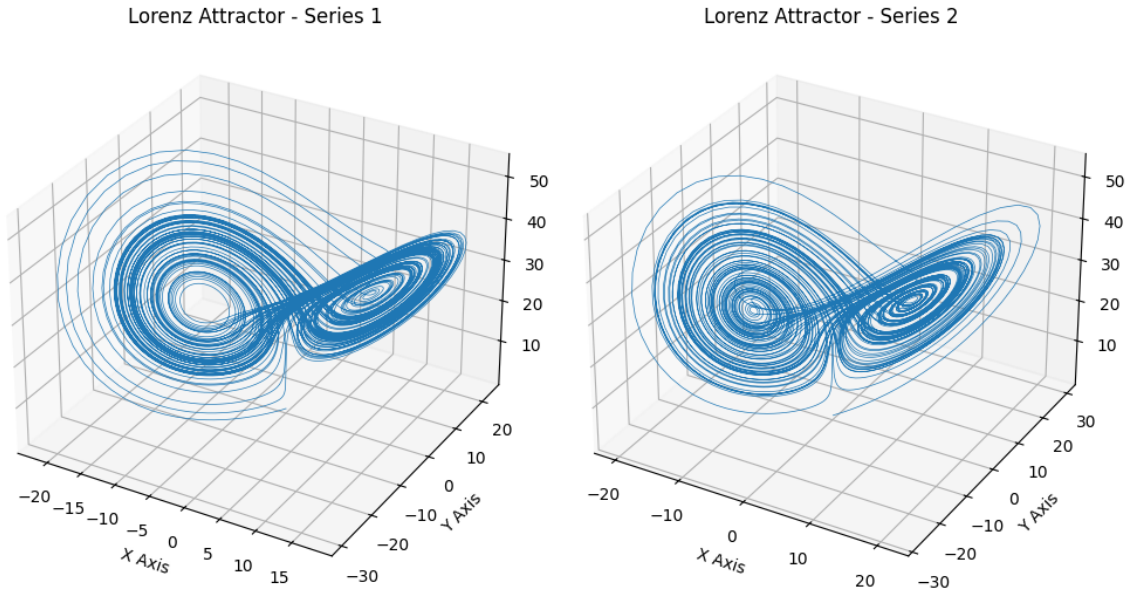

Figure 1: Two samples of Lorenz attractors with different initial coniditons, and values of σ=10, ρ=28, and β=2.667

From Figure 1 we can see that even the slightest change in the initial conditions have an effect on the system; these two initial points change the direction of the system and have significant differences when we approach the two attractors.

2.2 Python-library

First, we have to install the library. This is done by using the command `pip install ecdf_estimator`. As stated in the section above, we also need the libraries matplotlib and numpy, which are installed in the same way. After installing the packages, they need to be imported. This is done with the commands `import ecdf_estimator as ecdf`, `import numpy as np`, and `import matplotlib.pyplot as plt`.

The *ecdf_estimator* library was made by one of the supervisors of this thesis, Andreas Rupp. As stated, in (Rupp, 2022), an example code for parameter estimation can be found, which is what was also used as inspiration for the parameter estimation code in this thesis.

2.3 Estimation approach

The method we use to identify the parameters enables us to map a set of observed trajectories to a Gaussian distribution. With this we are able to define a statistical likelihood function, which is used as a cost function during the parameter estimation process.

The Lorenz system can be viewed as a dynamic model that depends on a set of parameters

$\boldsymbol{\theta} \in \mathbb{R}^p$. In this context, $\boldsymbol{\theta} = \theta$ is just a scalar parameter. The output of the "forward model" consists of a set of trajectories $\boldsymbol{x}(\boldsymbol{\theta})$, obtained by calculating the Lorenz system for various values of the parameter(s) $\boldsymbol{\theta}$ over a fixed time interval. Naturally, these trajectories change as the parameters do too. The trajectories also vary for fixed parameter values as strange attractors have a high sensitivity to initial conditions, which are changed when creating the data in this thesis. The goal is to distinguish this inherent variability from the systematic changes that arise due to the variation in the model parameters.

More specifically, our desire is to identify the parameter values that best fit a given set of trajectories. In order to accomplish this, we define a minimization problem using a stochastic cost function

$$f_{\boldsymbol{x}_{\text{data}}}(\boldsymbol{\theta}) \to \min, \tag{4}$$

and consider any solution to this problem as a set of parameters $\boldsymbol{\theta}$ that corresponds to the observed data $\boldsymbol{x}_{\text{data}}$. Let us go through the construction of $f_{\boldsymbol{x}_{\text{data}}}$ step-by-step.

To start, we need to specify what exactly is a solution to (4). Each set of parameter values result in a unique trajectory, but as stated before as well as demonstrated in Figure 1, because the Lorenz system is chaotic, even the slightest changes in the initial conditions cause the trajectory to change. The different parameter values are distinguished by the trajectories they produce. Given the data is three-dimensional, we need a metric to quantify the "distance" between two trajectories. Therefore, the set of trajectories is used to build a statistical likelihood function, which captures the variability within the data. This way the outcome is a range of acceptable solutions. The main idea is to define a "distance" measure d with which you are able to compare two trajectories $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$. Then the statistical properties of d are used to make a Gaussian likelihood based on the set of trajectories.

The set of the trajectories is divided into n number of subsets to statistically employ the training data. A function called eCDF is defined. This function takes two arguments, $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$, which carry N^+ and N^- patterns, respectively. The function is able to measure the correlation between $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$. Aside from the two arguments, the function depends on two parameters: a radius $R > 0$, and the "distance" d:

$$C(\boldsymbol{s}^+, \boldsymbol{s}^-; R, d) = \frac{1}{N^+ \times N^-} \sum_{i=1}^{N^+} \sum_{j=1}^{N^-} \#(d(\boldsymbol{s}_i^+, \boldsymbol{s}_j^-) < R). \tag{5}$$

The radius $R > 0$ is used to create a vector $\mathbf{y}$ which measures how similar two sets of trajectories are. After this we define a vector of bin values $(R_i)_{i=1}^m$ and set

$$\boldsymbol{y}(\boldsymbol{s}^+, \boldsymbol{s}^-) = \boldsymbol{y}(\boldsymbol{s}^+, \boldsymbol{s}^-; (R_i)_{i=1}^m, d) = C(\boldsymbol{s}^+, \boldsymbol{s}^-; R_i, d)_{i=1}^m.$$

This vector uses the bin values $(R_i)_{i=1}^m$ to show the eCDF of the set of values $d(\boldsymbol{s}^+,\boldsymbol{s}^-)$.

The statistics of $\boldsymbol{y}(\boldsymbol{s}^+,\boldsymbol{s}^-)$, which in this case are the mean and variance, are quantified across different subsets $\boldsymbol{s}^+$, $\boldsymbol{s}^-$ from the entire training dataset $\boldsymbol{s}_{\text{data}}$. This is done using the following methods: We get a scalar distance value N^2 for each subset pair. From this we compute a single eCDF-vector. We repeat this for all distinct $n(n-1)/2$ pairs

$$\boldsymbol{y}^{k,l} = \boldsymbol{y}(\boldsymbol{s}^k_{\text{data}},\boldsymbol{s}^l_{\text{data}}) \in \mathbb{R}^m \quad 0 \leq k < l \leq n,$$

so we can estimate the mean $\boldsymbol{\mu}_0 \in \mathbb{R}^m$ and the covariance $\boldsymbol{\Sigma}_0 \in \mathbb{R}^{m,m}$ of all the pairs of distinct eCDF-vectors.

According to the classical Donsker theorem and its generalizations, as $N \to \infty$, finite dimensional approximations of the eCDF-vectors become Gaussian. When N is sufficiently large, the eCDF-vectors are multinormally distributed, meaning the mean and covariance are sufficient enough to describe the statistical variablity of the function $\mathbf{y}$ within the selected subsets of the training data. The normality is tested numerically: We test whether the eCDF-vectors are Gaussian using the χ^2 (chi-squared)-test . For this, we evaluate all the values of the negative log-likelihood function

$$\left(\boldsymbol{y}_{k,l} - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}_0^{-1} \left(\boldsymbol{y}_{k,l} - \boldsymbol{\mu}\right),$$

from which we get a histogram which is compared against the density function of the distribution χ^2_M with m degrees of freedom.

To evaluate the cost function for a new parameter value $\boldsymbol{\theta}$, we simulate the Lorenz system N times using $\boldsymbol{\theta}$ and collect the resulting trajectories as $\boldsymbol{x}(\boldsymbol{\theta})$. After this the eCDF-vector $\boldsymbol{y}(\boldsymbol{\sigma}) = \boldsymbol{y}(\boldsymbol{s}(\boldsymbol{\sigma}),\boldsymbol{s}^{\text{data}}_k)$ is computed for a randomly selected $k \in \{1,2,\ldots,n\}$ and the likelihood value is evaluated as

$$f_{\boldsymbol{s}_{\text{data}}}(\boldsymbol{\sigma}) = (\boldsymbol{y}(\boldsymbol{\sigma}) - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\sigma}) - \boldsymbol{\mu}_0).$$

This Section (2.3) is based on information from (Kazarnikov et al., 2023).

3 Evaluation of training data

To start, we have to create the training data. This is done by creating a function `lorenz_system()`, which calculates the points in a Lorenz attractor with the given parameters. Next we have to choose the amount of points we want to calculate from an attractor, and how many attractors we want to calculate. The user gets to choose the initial values for σ, ρ, and β, and in this case we will choose the values used in Figure 1, which are also the values Lorenz used. They are σ=10, ρ=28, and β=2.667. Now the `lorenz_system()`-function is called and looped by the chosen amount of points and attractors which creates our training data. The data is created using Euler method.

Next, the training data is divided into subsets and another function `distance()` is defined. This function calculates and returns the Euclidean distance between all subset pairs.

Now, we have to create the eCDF-vectors. The eCDF-vectors are going to be created twice, once with a large amount of bins and another time with a smaller amount of bins. To determine a suitable region for the bins, we use the command `min_val, max_val, distance_data = ecdf.estimate_radii_values(xyz_points[0:subset_sizes[0]], xyz_points[subset_sizes[0]:subset_sizes[0]+subset_sizes[1]], distance)`. The inputs for the command are the first two subsets of the training data and the diatnce function. The command returns the minimum and maximum values for the bin regions, and the distances between data points.

The range obtained from the minimum and maximum values is split into 50 bins with which we can calculate our first objective function using the command `aux_func = ecdf.standard(xyz_points, bins, distance, subset_sizes`. The command takes the training data, bins, `distance()`-function and `subset_sizes` as inputs. The command calculates all the distances between the subsets. The eCDF-vectors are created by cumulatively calculating the amount of these distances that fall into any of the 50 bins. The mean and covariance of the training data is also calculated using this command. The results are stored in the variable `aux_func` for further use.

Now we create the second objective function. This function is going to have less bins. This is done to make sure that there is no correlation between neighboring bin values. The amount of bins is chosen using the `choose_bins()`-function from the estimator. The same command as in the previous paragraph is used to calculate the function, only this time with the smaller amount of bins. The results are saved in different variable, named `func`.

Now that the results are saved in the two variables, we can plot the distribution of the

eCDF-vectors. This is done using the `plot_ecdf_vectors()`, and `plot_mean_vector()`-functions from the *ecdf_estimator* library.

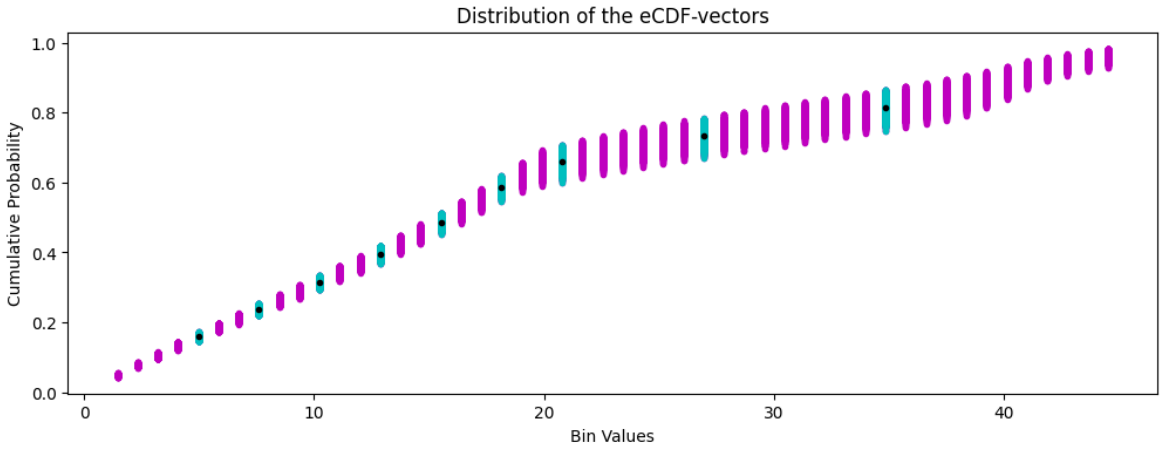

Figure 2: Figure of the distribution of the eCDF-vectors using this thesis' training data

After this, we need to make sure the eCDF-vectors are Gaussian. This is done by performing the χ^2-test. The estimator has a ready `plot_chi2_test()`-function that is used for this. To check how well all the vectors fit the objective function's model, the function computes negative log-likelihood values for all vectors. The values are normalized, after which a histogram of them is plotted. A probability density function of the χ^2 distribution is plotted in the same figure as the histogram with degrees of freedom that is equal to the number of eCDF-vectors. If the histogram fits the density function, we can confirm the normality of the eCDF-vectors and continue with the parameter estimation knowing the results should be reliable.

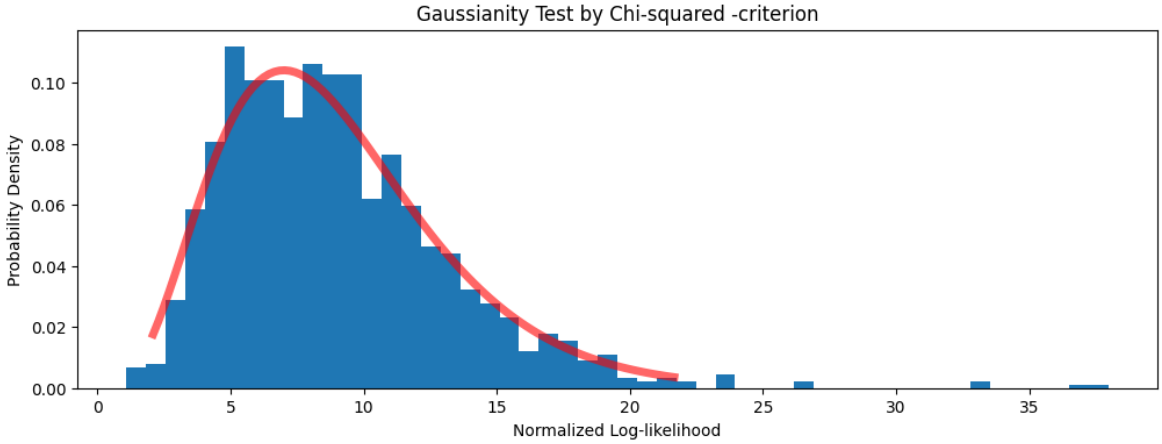

Figure 3: Figure of the χ^2-test using this thesis' data

Figure 3 shows us that the histogram does in fact fit the density function. Now that we can

trust the reliablility of the results, we can start the actual parameter estimation. In this thesis, we will be estimating all three parameters σ, ρ, and β. To make the estimation process simpler, we will only be estimating one parameter at a time, starting with σ, then ρ, and finally β.

4 Results

For the estimation, we have calculated 111 (in python the indexing starts from 0, so the input value for 111 points is 110) points with a delta time value of 0.01. This has been repeated for 50 epochs. From every epoch, we have ignored the first 51 points, meaning for the evaluation we only use the last 60 points. This way we only use the chaotic points of the system, because the first few points in every epoch do not differ from each other as much as the last ones do. The data for the estimations was calculated by changing the value of the desired parameter by ± 2 from the value that Lorenz used. This was done in a `linspace` with a sample size of 50. For every one of the 50 parameter values, the same amount of data was calculated as in the training data.

4.1 Estimation of the σ parameter

After evaluating the quality of the data, we can now calculate the results. First, we will be estimating the σ parameter. For this parameter, Lorenz used the value 10. As explained in the previous paragraph, we will be calculating the data using parameter values of 8-12. The first means of estimation is calculating negative log-likelihood values.

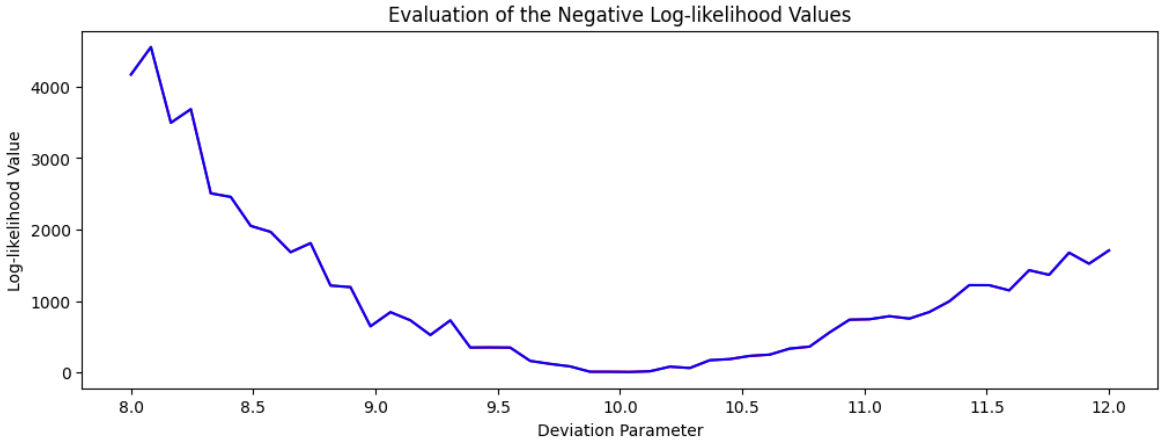

Figure 4: Visualization of the negative log-likelihood values for σ

As we can see from Figure 4, it clearly gets it's smallest values when the parameter value is around 10. This is exactly what we want to see, because with negative log-likelihood values, the parameter fits the data best with the smallest likelihood values. At first glance, there seems to be only slight differences in the negative log-likelihood values between the parameter values of 9-11. If we have a more in-depth look at the results, we can see that somewhere between the values 9.6 and 10.3, the blue dots are almost in a straight line and the value of the negative log-likelihood stays more or less the same.

The next, and last step of the evaluation is normalizing the log-likelihood values. For this, we multiply each value by -0.5 before exponentiating them. This turns the smallest negative log-likelihood values, which were the best results, into the largest numbers. Then with these values, we compute the sum of all the columns of the just calculated parameter. All values are divided by the sum to get the results.

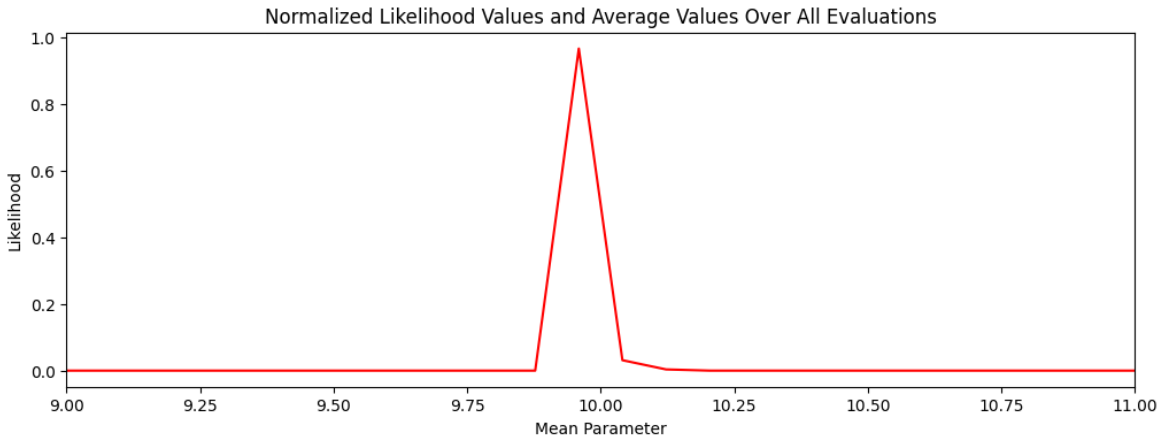

Figure 5: Visualization of the normalized likelihood values for σ

Figure 5 confirms that our conclusions from Figure 4 were on the right path. From the negative log-likelihood results, it's very hard to confirm where the actual best likelihood is at, because it looks like many values have the same likelihood. This is why the likelihoods are normalized. From this, we can clearly see that the best likelihood value is found at the parameter value of 10. And because all of the likelihood values add up to 1, we can confirm that these calculations are correct.

4.2 Estimation of the ρ parameter

Again, after making sure of the quality of the data with the χ^2 test (the same data is used for all parameters), we can start the estimation. For ρ, Lorenz used the value 28. This means the parameter values used for the data will be 26-30. The negative log-likelihood values for the estimation are:

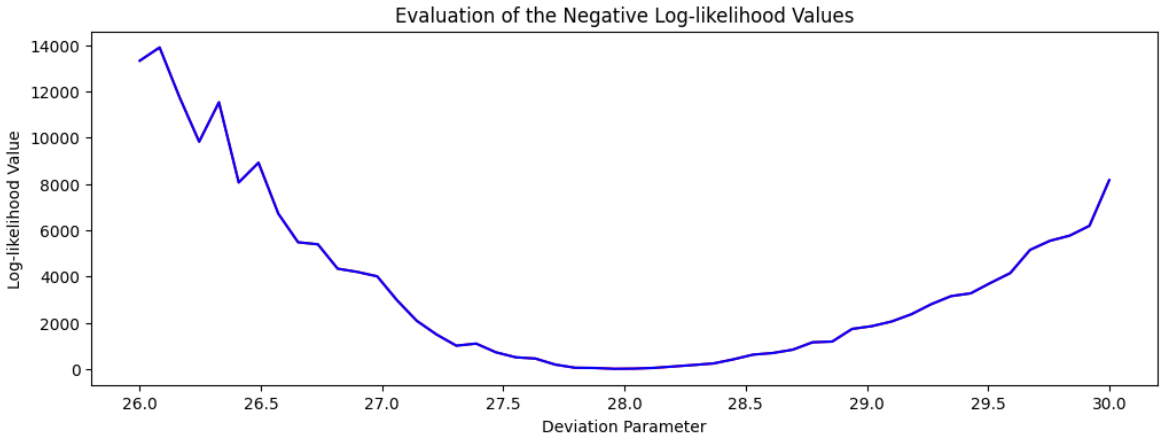

Figure 6: Visualization of the negative log-likelihood values for ρ

Figure 6 clearly shows us that the smallest values are found around the value 28. As with the negative log-likelihoods for σ, this too only has minor differences between the likelihoods for values 27.5-28.5, but if we take an even closer look, we can see that there is almost a straight line between 27.8 and 28.2. Again, the issue is we cant be sure if the best likelihood is exactly at 28. Let's normalize these likelihoods to get a better picture of the results

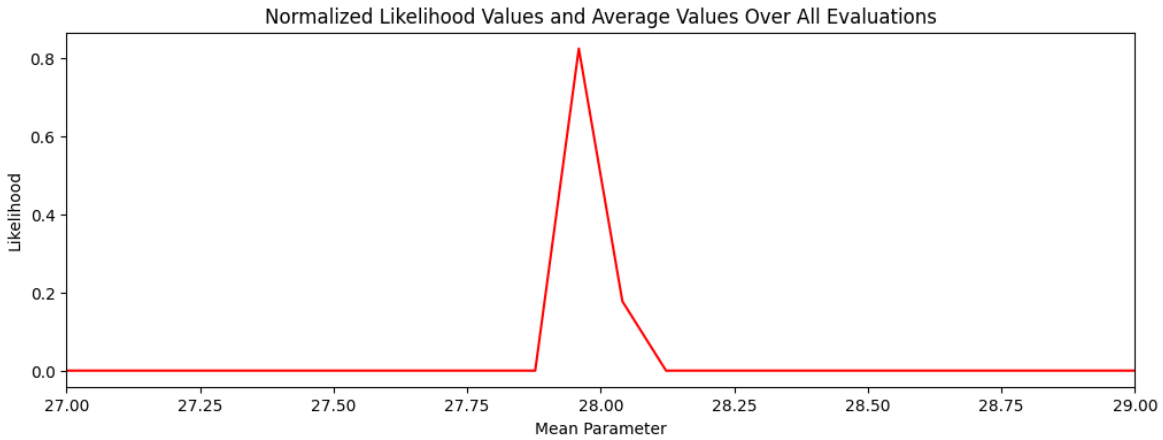

Figure 7: Visualization of the normalized likelihood values for ρ

Figure 7 confirms that the best value is in fact at a ρ value of 28. Again, the likelihoods add up to 1, so we can be sure these calculations are correct.

4.3 Estimation of the β parameter

As with the two parameters before, we start with the χ^2 test. After making sure the eCDF-vectors are Gaussian, we can estimate the parameter value. For the β parameter, Lorenz used

the value 2.67. Again, the parameter values used will be ±2, so 0.67-4.67. Let us start with the negative log-likelihood values:

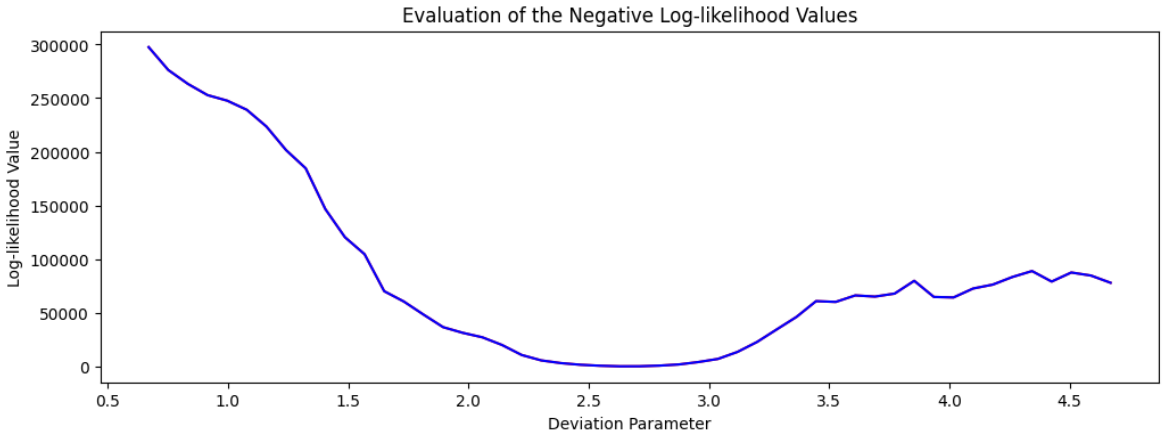

Figure 8: Visualization of the negative log-likelihood values for β

Figure 8 shows that the best value is somewhere between the values 2.3-3 for β. Let us normalize the likelihoods to see whether the best value is found at β value of 2.67.

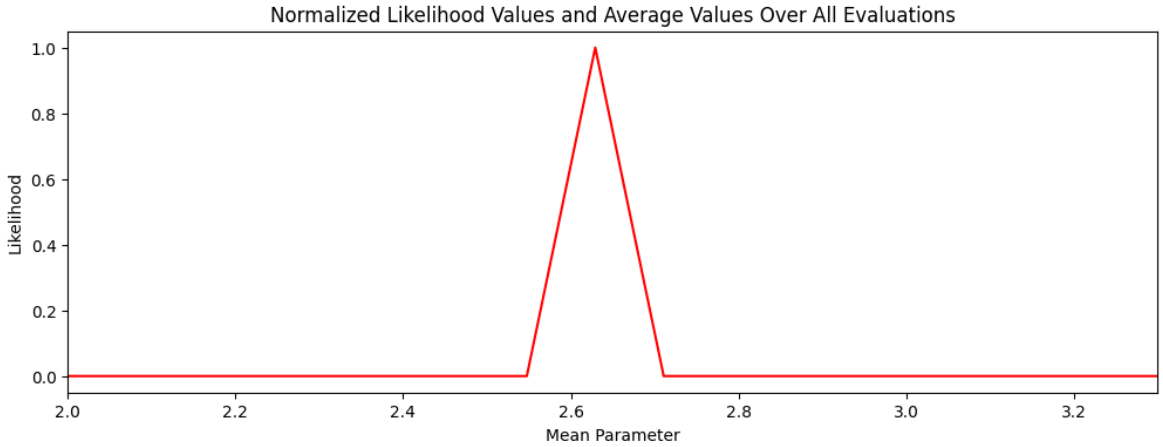

Figure 9: Visualization of the normalized likelihood values for β

Once more, the normalized likelihood values confirm that the best value is at the parameter value Lorenz used (around 2.67). The likelihood values also add up to 1 again, so the results are correct.

5 Conclusions

The objective of this thesis was to use the *ecdf_estimator* Python library to estimate the values of the three parameters in the Lorenz system. We started off by creating the data using Euler method with parameter values of $\sigma = 10$, $\rho = 28$, and $\beta = 2.67$. Following this, we calculated and plotted the eCDF-vecotrs, and performed a χ^2-test to validate that the eCDF-vectors are Gaussian. All these tasks were done with no complications.

After the χ^2-test the parameter estimation was started. The keep the estimation technique simple, we did not estimate all three parameters at once, but instead one at a time.

We started with the σ parameter, which we were hoping would get its best likelihood values at the parameter value of 10. We first calculated the negative log-likelihood values, which showed us that the best likelihood values were somewhere around 10, but due to the nature of negative log-likelihood values we can only make a rough estimation. After this we converted the likelihoods to normalized likelihood values, from which we could confirm that the best likelihood value was at 10. This means the estimation of σ was successful and we could move on to the next parameter.

The next parameter was ρ, which should have been getting its best likelihood values at 28. Again, we started off with the negative log-likelihood values, which showed us that the best likelihood value was somewhere close to the value 28. After normalizing the likelihood values and plotting them, we could confirm that the best likelihood value was at the parameter value of 28. The estimation of ρ was successful and now we could start estimating the last parameter.

The last parameter we estimated was β. β is supposed to get its best likelihood values at 2.67. The exact same procedures were used for this parameter too. The negative log-likelihood values confirmed that the best likelihood values were somewhere close to 2.67. After normalizing the values, we could again confirm that the best likelihood values were in fact at the parameter value we were expecting. Estimating the β parameter gave us the same successful outcome as the two previous parameters.

To conclude, we were able to create the training data and use the *ecdf_estimator* library to estimate the parameters in the Lorenz system without any issues, and the estimation results were as expected.

References

Britannica (2024). *Chaos Theory*. URL: https://www.britannica.com/science/chaos-theory.

Haario, H., Kalachev, L. & Hakkarainen, J. (2015). Generalized correlation integral vectors: A distance concept for chaotic dynamical systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 25(6), p. 063102. DOI: 10.1063/1.4921939. URL: http://dx.doi.org/10.1063/1.4921939.

Kazarnikov, A., Ray, N., Haario, H., Lappalainen, J. & Rupp, A. (2023). *Parameter estimation for cellular automata*. arXiv: 2301.13320 [nlin.CG]. URL: https://arxiv.org/abs/2301.13320.

Manning, J. (2017). Chaos: The Mathematics Behind the Butterfly Effect. Colby College Mathematics. URL: https://www.colby.edu/wp-content/uploads/2017/08/2017-Manning-Thesis.pdf.

Matplotlib (2024). *Lorenz attractor*. URL: https://matplotlib.org/stable/gallery/mplot3d/lorenz_attractor.html.

Rupp, A. (2022). *ecdf_estimator_examples*. URL: https://github.com/AndreasRupp/ecdf_estimator_examples/tree/main.

Springer, S., Haario, H., Shemyakin, V., Kalachev, L. & Shchepakin, D. (2019). Robust Parameter Estimation of Chaotic Systems. *Inverse Problems and Imaging* 13(5), pp. 955–978. DOI: 10.3934/ipi.2019053. URL: http://dx.doi.org/10.3934/ipi.2019053.