VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Wetenschappen en bio-ingenieurswetenschappen: Fysica en Sterrenkunde

STATISTICAL ANALYSIS OF DIFFERENT MODELS FOR IDENTIFYING MICROBIAL INTERACTIONS FROM TIME-SERIES

Jari Beysen

June 2022

Promotor: prof. dr. de Buyl
sciences and bioengineering sciences

Acknowledgements

I would like to thank my promotor, prof. dr.de Buyl, for guiding me through the process of writing a bachelor thesis. Her many insights came in handy when I needed them the most. I would also like to thank my partner and my parents for bearing through my endless rambling sessions about Physics and Computer science. Especially my partner for proofreading my work and correcting my grammatical errors.

Contents

Chapter 1

Introduction

Over the last decade, many studies have attempted to create models which accurately infer the dynamics of microbial time series, from novel inference methods and classic machine learning strategies, to far more exotic models. This current trend is due to the fact that studies have shown correlation between human health and the bacterial make up of ones stomach flora. For example, a recent study by Wanyin et al [1] found a correlation between the diversity of intestinal bacteria and the severity of SARS-CoV-2 symptoms. Modern metagenomic sequencing methods have as well improved tremendously to the point that it is possible to track abundances of microbes over time. Its hypothesized that the evolution of these abundances over time can be described by the Lotka-Volterra equation, also known as the prey-predator equation.

The concept of 'Physics Informed Deep Learning' gained momentum in the realm of Computer science, as well as Physics research recently. This concept has been shown to be robust to noisy and small datasets. Research in this field is still in its early stages, and has not yet been applied to microbial time series.

In this bachelor thesis a meta-analysis between popular regression methods applied to the Stochastic Generalized Lotka-Volterra equation (sgLV) will be done, as well as an attempt at implementing a Physics Informed Neural Network (PINN) data-driven discovery model to this field of research. A novel idea for improving robustness to extremely small datasets will also be explored.

Chapter 2

Generalized Lotka-Volterra equation (gLV)

The time series of microbial abundances are often described by the Lotka-Volterra equation (2.1). This equation represents the nature of interspecies interaction.

$$\frac{dx_i(t)}{dt} = x_i(t)(\sum_j M_{ij}x_j(t) + \vec{r_i}) \tag{2.1}$$

The parameters $\mathbf{M}$ and $\vec{r}$ are the interaction matrix and the growth rate. The rate of change of a species's abundance (x_i) is dependant on other species's abundances. Typically this equation is used for prey-predator interactions, which is not truly the case for bacterial interactions. Bacteria produce and consume nutrients in their living environment, for example: species A might produce nutrients which are beneficial for species B but detrimental for species C. This is not a mutually consistent relation. Species B might produce nutrients which are detrimental for species A.

The gLV equation is able to produce a variety of dynamics, from chaos and limit cycles to point attraction. Fixed points of a gLV equation can be calculated with equation 2.2.

$$\bar{\vec{x}} = -\mathbf{M}^{-1}\vec{r} \tag{2.2}$$

Commonly, for regression, the growth rate is expressed in term of a time series's converged abundances.

$$\vec{r} = -\mathbf{M}\bar{\vec{x}} \tag{2.3}$$

This transforms equation 2.1 into:

$$\frac{dx_i(t)}{dt} = x_i(t)(\sum_j M_{ij}(x_j(t) - \bar{x_j})) \tag{2.4}$$

Stochastic generalized Lotka-Volterra equation (sgLV)

Experimental microbial time series display stochastic behaviour [2]. Noise described by brownian motion (σdW(t)) was added to the Lotka-Voltarra equation, which results in equation 2.5.

$$\frac{dx_i(t)}{dt} = x_i(t)[\sum_j \mathbf{M}_{ij}(x_j(t) + \vec{r_i} + \sigma dw(t)] \quad \Big| \quad dw(t)\sqrt{dt} \sim \mathcal{N}(0,1) \tag{2.5}$$

In silico time series

There are a multitude of ways to solve systems of differential equations. Python has many packages, like sdeint [3], which can handle such problems. For transparency's sake, data was generated by discretizing the sgLV equation.

$$\Delta x_i(t, \Delta t) = x_i(t)\big[\sum_j M_{ij}(x_j(t) + \vec{r_i}\big]\Delta t + x_i(t)\sigma\sqrt{\Delta t} \tag{2.6}$$

Data can be generated via equation 2.7 by recursively adding equation 2.6, beginning at a randomized vector of starting abundances (x_0). This method is sometimes called the Euler–Maruyama method.

$$x_i(t + \Delta t) = x_i(t) + \Delta x_i(t, \Delta t) \quad | \quad x_i(t = 0) = x_0 \tag{2.7}$$

Dataset generation

A large dataset of randomized in silico time series is needed to do a statistical analysis for different regression methods. The next two short sections describe the specific methods for generating randomized interaction matrices and the corresponding time series.

Interaction matrix and growth rate

The matrices $\mathbf{M}$ were initialized with random uniform diagonal elements $\sim \mathcal{U}(-1, 0)$. K random gaussian distributed off-diagonal elements[1] $\sim \mathcal{N}(0.1, 0.2)$ were iteratively added if the resulting time series converged. Growth rates were sampled from a lognormal distribution $\sim ln(\mathcal{N}(0, 1))$.

[1] K varies between 0 and $N^2 - 1$

Time series

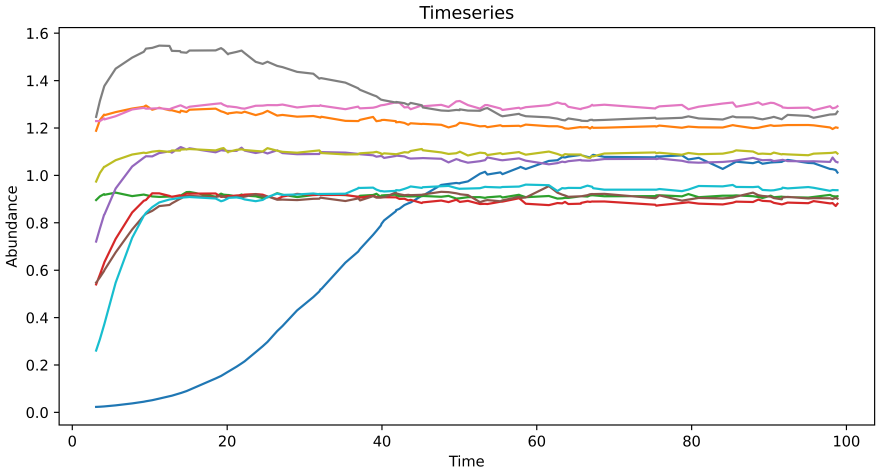

Figure 2.1: Example of a time series generated by the numerical approximation of the sgLV Equation2.7

Equation 2.7 with t $\in [0, 100]$, a step size of 10^{-4}, and brownian noise standard deviation of 0.01 was used to create time series. 100 Random timepoints were kept to create a dataset of discrete time series.

Chapter 3

Regression methods

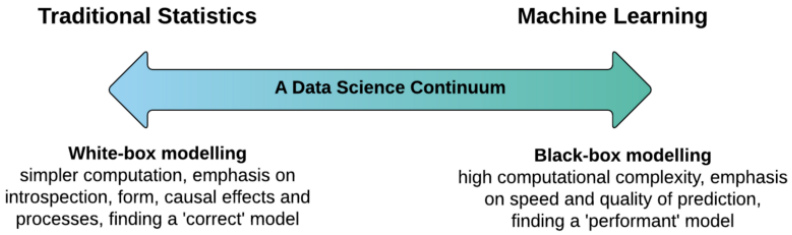

Figure 3.1: Image by Loyola-González[4]

3.1 White box

White box models as described by Loyola-González [4, p. 6], are models which are interpretable and explainable. The actions these models make can be mathematically or logistically explained. Two different models, LIMITS 3.1.1 and Ridge regression 3.1.2, will be explored.

3.1.1 LIMITS

LIMITS [5] (Learning Interactions from MIcrobial Time-Series) is a method based on linear regression. A first important step in defining the LIMITS algorithm is approximating the discretized sgLV equation 2.6 by linear equations. This was done by combining it with equation 2.4 and approximately solving it, which results

in:

$$x_i(t+\Delta t) = ln(n_i(t))x_i(t)exp(\Delta t(\sum_j M_{ij}(x_j(t)-\bar{x_j})) \quad \Big| \quad n_i \sim \mathcal{N}(1,\sigma^2) \quad (3.1)$$

This equation can be reformulated into a linear equation.

$$Y_i(t) = n_i(t) + \mathbf{M_i} \cdot \vec{X}(t) \quad \Big| \quad Y_i(t) = \frac{ln(x_i(t+\Delta t)) - ln(x_i(t))}{\Delta t} \& \quad X_i(t) = x_i(t) - \bar{x_i} \quad (3.2)$$

Now the foundations are laid down for a linear regression and the LIMITS algorithm.

$$\vec{Y_i} = \mathbf{M_i} \cdot \vec{X} + e_i \quad (3.3)$$

$$\hat{\mathbf{M}}_\mathbf{i} = \vec{X}^{+1} \cdot \vec{Y_i} \quad (3.4)$$

We can define the error on this regression as:

$$\hat{e_i}^2 = (\vec{Y_i} - \hat{\mathbf{M}}_\mathbf{i} \cdot \vec{X})^2 \quad (3.5)$$

The LIMITS algorithm can be described by the following steps:

1. Initialize a base regression case by 'turning off'[2] every dimension of $\vec{X}$ except index i, and calculate the error of the regression. this represents a base case where there are no interspecies interactions bewteen $\vec{X}$ and $\vec{Y}$.
2. Create regression instances by 'turning on' every 'off' dimension of $\vec{X}$ individually.
3. Keep the regression with the lowest error and check if it crosses a pre-specified threshold.
4. If this regression does cross this threshold, add the dimension to the base case and repeat starting at 2.
5. Else, stop the algorithm and use the dimension-reduced $\vec{X}$ and do a final regression.
6. Repeat for every Y_i to create a matrix.

[1]The pseudo-inverse of a matrix. This is commonly used approximation to the inverse when a matrix is singular.

[2]replacing a dimension by a list of 0's

Additionally, Fischer et al [5] applied a bagging procedure to LIMITS. This was done by splitting the training set into n equal parts and applying LIMITS on these sets. Afterwards, the median interaction matrix was calculated from the inferred interaction matrices. In section 5.1 the impact of bagging will be analyzed.

An observant reader might notice that structurally LIMITS reduces to a Greedy[3] Search without backtracking in a directed graph where every node is a dimension reduced $\vec{X}$. Greedy search will not always produce a global minimum. As well, due to the threshold, certain combinations might be isolated and will not be checked by a greedy search instance. A way to check if this is to look at the solutions different algorithms produce, which are not greedy by nature. To achieve absolute certainty about the optimality of the solution, it would be best to search the entire hypothesis space.

A first thing to ask if this is a feasible thing to do. Lets first define a function that returns the size of this hypothesis space depending on the amount of species, which is represented by the dimension of $\vec{X}$:

$$S(N) = \sum_{n=0}^{N} \frac{(N-1)!}{(N-1-n)!} = \Gamma(1, N) \tag{3.6}$$

This equation was formed by looking at the recursive nature of the tree representing every path the algorithm could take when there is no threshold. $\Gamma(a, b)$ is the incomplete gamma function. Figure 3.2 indicates S(N)$\sim O(10^{N-2})$ since it displays approximately linear behaviour (for dimension ranges used in this bachelor thesis) when viewed on a logarithmic scale. Greedy search would be $O(N)$. Do keep in mind that this would be the worst case scenario where no threshold is present. This reveals that searching the entire hypothesis space for larger dimension is computationally ill-advised, so some limitations have to be set.

One solution would be to artificially limit the amount of nodes checked by the algorithm. One could do this by limiting the amount of nodes checked or by limiting the search to a percentage of the hypothesis space size. For further testing, the algorithm was limited to checking 100 nodes.

[3] A search with a locally optimal heuristic.

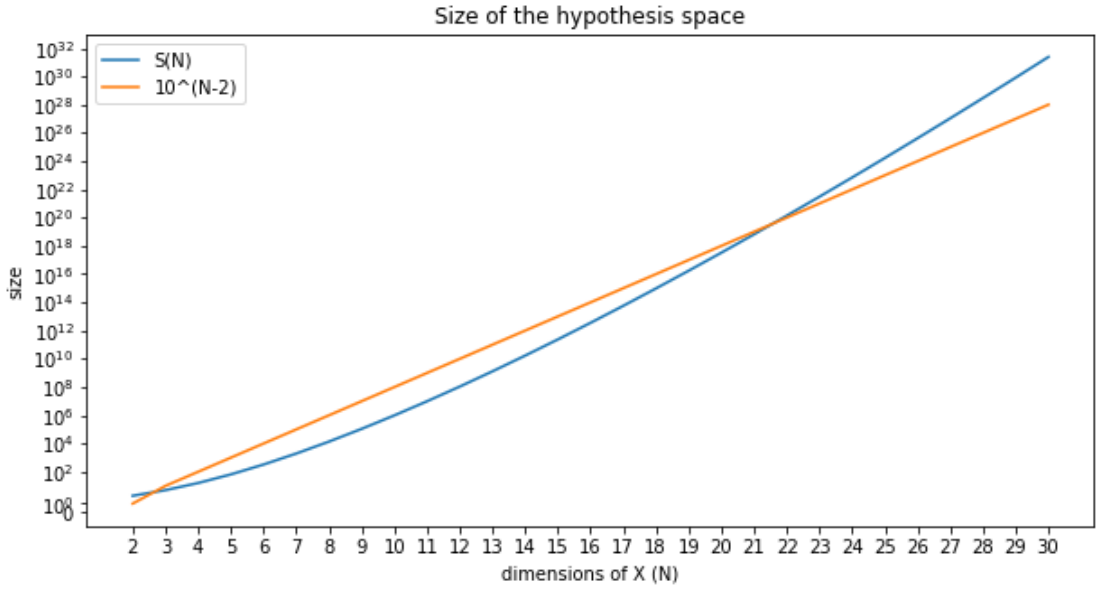

Figure 3.2: Plot of S(N) with logarithmic scale

Dijkstra's algorithm

Dijsktra's algorithm[6] is a very basic algorithm which ensures the shortest path in a graph is found. It in essence reduces to expanding the node with the shortest additive vertex[4]. This algorithm is not that useful in the case of LIMITS because limiting the search length will not allow the algorithm to explore beyond a certain depth. A slight alteration can be made to Dijkstra in the case of LIMITS. Path depth is correlated with a smaller error since more active dimensions of $\vec{X}$ will result in a smaller error, thus expand the node with the smallest additive inverse error. Lastly, to find the optimal solution, return the state with the lowest error in the validated list[5].

Reversing Dijkstra's algorithm

Instead of searching for the shortest path, one could look for the longest possible path. This seems contradictory, certainly because finding the longest path without revisiting notes is an NP-complete problem. An incomplete search can be done by reversing Dijkstra's algorithm: expanding the node with the biggest additive inverse error. This will behave like a Greedy search but with backtracking[6].

[4]Sum of the values of the vertices connecting a node to the starting state.
[5]States which passed the threshold.
[6]Going backwards in a tree.

Adjusted LIMITS algorithm

1. Initialize a base regression case by 'turning off' every dimension of $\vec{X}$ except index i, and calculate the error of the regression.
2. Initialize an empty queue, validated list and completed list. These will be filled with lists of nodes and their parents[7]. This is done to prevent cycles.
3. Expand the node with the largest additive inverse error in the queue and add its children to the queue if this node is not in the completed list and passes the threshold. A child in this case takes the parent's state and turns 'on' 1 more dimension.
4. If the threshold was passed add it to the validated list
5. Repeat from 2. until the check limit is reached.
6. Return the state with the lowest error
7. Repeat for every Y_i to create a matrix.

3.1.2 Ridge regression

Another frequently used method in machine learning, for situations where there are highly correlated independent coefficients, is Ridge regression. It is simply put a linear regression with an L^2-penalty on the regressed coefficients [7, p. 9]. The Ridge estimator can be described by 3.8.

$$\hat{\mathbf{M}} = argmin_{\hat{\mathbf{M}}}\left[||\vec{Y} - \hat{\mathbf{M}}\vec{X}||^2 - \lambda||\hat{\mathbf{M}}||^2\right] \quad \Big| \quad \lambda = penalty \tag{3.7}$$

This equation can be simplied to:

$$\hat{\mathbf{M}} = \vec{Y}\vec{X}^T(\vec{X}\vec{X}^T + diag(\lambda))^{-1} \tag{3.8}$$

To find the ideal value of λ, K-fold cross validation(N datapoints/10) was used wherein every iteration the remaining partition was used as a test set. A λ_i was found by minimizing equation 3.8 equation with respect to λ which turns into equation 3.9, this was applied to this test set. Afterwards, the λ_i which resulted in the lowest error was used as the final λ for training.

$$\lambda_i = argmin_{\lambda_i}\left[\vec{Y_{test}}\vec{X_{test}}^T(\vec{X_{test}}\vec{X_{test}}^T + diag(\lambda_i))^{-1}\right] \tag{3.9}$$

[7]The nodes it was expanded by.

3.2 Black box

Black box models, as described by Loyola-González, are models with a low amount of interpretability [4, p. 2]. However, these models frequently are able to learn far more complex problems to a better degree than the average white box model.

3.2.1 Neural Networks and Deep Learning

Neural networks are a network of nodes which are interconnected and sectioned into layers [8]. These nodes add the output of the previous layers' nodes multiplied by trainable weights and passed through a (commonly non-linear) activation function. Training data is passed through these layers, which is referred to as a **Feed-Forward** network. Training these weights is done by defining a **Loss Function** and minimizing this function with respect to the weights in the network, also called **Back Propagation**.

The terms 'deep learning' and 'artificial neural networks' are frequently used interchangeably. They only really differ by the amount of human intervention. Lets look at the classic example where we are trying learn to differentiate between cat and dog pictures. An artificial neural network would have some sort of data augmentation pipeline, like edge detection followed by Principal Component Analysis (PCA, a form of dimension reduction) before feeding into the network. A deep learning model would feed straight into the network without data augmentation being done beforehand, the model is totally hands-off.

Physics informed deep learning

Physics Informed Neural Networks (PINN's) are identical to classic neural networks in terms of architecture (Maziar , Paris and George[9],2017,p.1). They only differ by one key factor, their loss functions penalize deviation from the physical laws their specific systems abide by. These physical laws are represented by differential equations (DE's or PDE's). This small addition to the loss function makes PINN's viable for situations where less training data is available. This framework however does not yet provide a way to do regression.

Before moving forward, it is necessary to talk about the software which was used to build the physics informed neural network. The large majority of data scientists use Python for deep learning. There are two packages which are most frequently used, Tensorflow and Pytorch. Pytorch is generally aimed more towards research, and Tensorflow is aimed more towards real world implementations. They are however fully interchangeable. Tensorflow has more prebuilt features, and can be

programmed in the backend as well as the in the frontend. Due to prior experience and the previous statement it was used from here on out.

To demonstrate how regression can be done, take for example a simple system such as a mass falling towards a celestial body. Here we could be interested in the gravitational constant. First, create a simple shallow neural network that tries to find the path this object took through space and time. In addition to the means squared error, the mean of the squared residue of Newton's law is added in the loss function:

$$L = \frac{1}{N}\sum_{i=0}^{N}(h_i - \hat{h_i})^2 + \frac{1}{N}\sum_{i=0}^{N}(\frac{d}{dt}\hat{h_i} - v + g \cdot t)^2 \tag{3.10}$$

The unorthodox concept that allows the neural network to infer the value of g and v is the ability of TensorFlow to add trainable variables to a model. Now not only will the model reduce the loss function with respect to the network weights, but also with respect to this trainable variable. Let g in this example be the gravitational constant for the Moon, 1.62 $\frac{m}{s^2}$ and the mass' initial height and speed of 1 m and 3 $\frac{m}{s^2}$. The height of a thrown mass was measured over a period of 3.5 seconds and a step size of 0.1 second. Normal noise with a standard deviation of 0.1 was added. The model was set up with: 5 layers, 10 neurons per layer with a sigmoid activation function and the Nadam[8] optimizer with a learning rate of $4 \cdot 10^{-5}$. The model found a value of 1.6155126 $\frac{m}{s^2}$ for g and 2.9860628 $\frac{m}{s}$ for v after training for 200 000 iterations.
In the case of microbiome time series, the Lotka-Volterra non-linear ODE should describe the interactions between microbes.

Stochastic physics informed deep learning

Physics informed neural networks are not 'informed' enough to handle and predict noise in the sgLV equation. Stochastic Physics Informed Neural Networks (SPINN's) [10] modify the concept of PINN's in a way that it can handle stochastic differential equations (SDE's). This would require that multiple time series for one interaction matrix with the same starting abundances are generated. This is not easily feasible for experimental data. Since the ultimate goal of these models is to be performed on experimental data, this was not implemented.

To compensate for a very small amount of data and to prevent the model from

[8]Combination of RMSprop-and Momentum based gradient descent

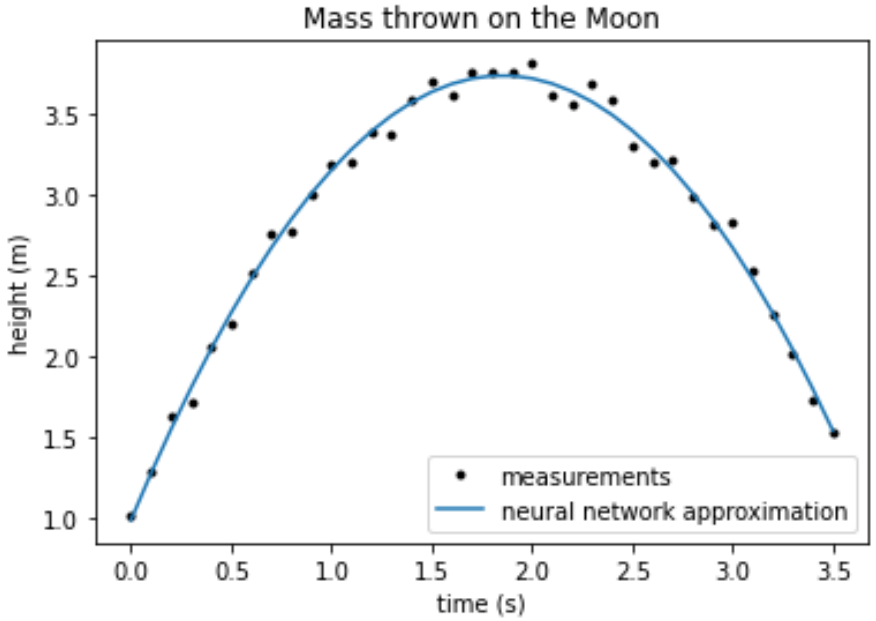

Figure 3.3: Plot of the measurements and the model's trained function

learning noise, term 3.11 was added to the loss function. This term penalizes the difference between the time series approximated by the neural network and the time series that can be generated from the predicted interaction matrix by using equation 2.5. Notice that this does not require training data. Generating a large amount of data every training cycle will slow the process down tremendously, so random short-length time periods are chosen every training cycle to generate data from. The standard deviation in the equation below should be equal to the standard deviation of the real time series. This could be approximated by using a moving average noise estimation method. For simplicity's sake the standard deviation was set to a static 0.02.

$$l_i = \frac{1}{tn}\sum_{i=0}^{tn}(\hat{x_i}(t_i) - x_{i,guess}(t_i))^2 \quad \Big| \quad x_{i,guess}(t+\Delta t)$$
$$= ln(n_i(t))x_{i,guess}exp(\Delta t\sum_j(\hat{M}_{ij}x_{i,guess}(t) - \bar{\hat{x}}_i)); \quad x_{i,guess}(t=0) = x_i(t=0)$$
$$\& \quad \Delta t = (t_{i+1}) - t_i \quad \& \quad n_i(t) \sim \mathcal{N}(0,\sigma) \tag{3.11}$$

Additionally a penalty on the starting abundances was included, the final loss

function takes the following form:

$$L = \frac{1}{N}\sum_{i=0}^{N}\Bigg[\frac{1}{tf}\sum_{i=0}^{tf}(x_i(t_i)-\hat{x}_i(t_i))^2 + \frac{1}{tf}\sum_{i=0}^{tf}s_i(t_i)^2 + \frac{1}{tn}\sum_{i=0}^{tn}(\hat{x}_i(t_i)-x_{i,guess}(t_i))^2 \\ +(x_i(t_0)-\hat{x}_i(t_0))^2\Bigg] \quad \Big| \quad s_i(t) = \frac{d}{dt}\hat{x}_i(t) - \hat{x}_i(t)\big[\sum_j \hat{M}_{ij}\hat{x}_j(t) + \hat{\vec{r}_i}\big] \tag{3.12}$$

By minimizing equation 3.12 we are able to approximate the interaction matrix $\hat{\mathbf{M}}$ as well as the growth rate $\hat{\vec{r}}$. We are not interested in the growth rate so this term will be ignored in further analysis.

Final deep learning model

Figure 3.4 illustrates the architecture and the training cycle of the neural network.

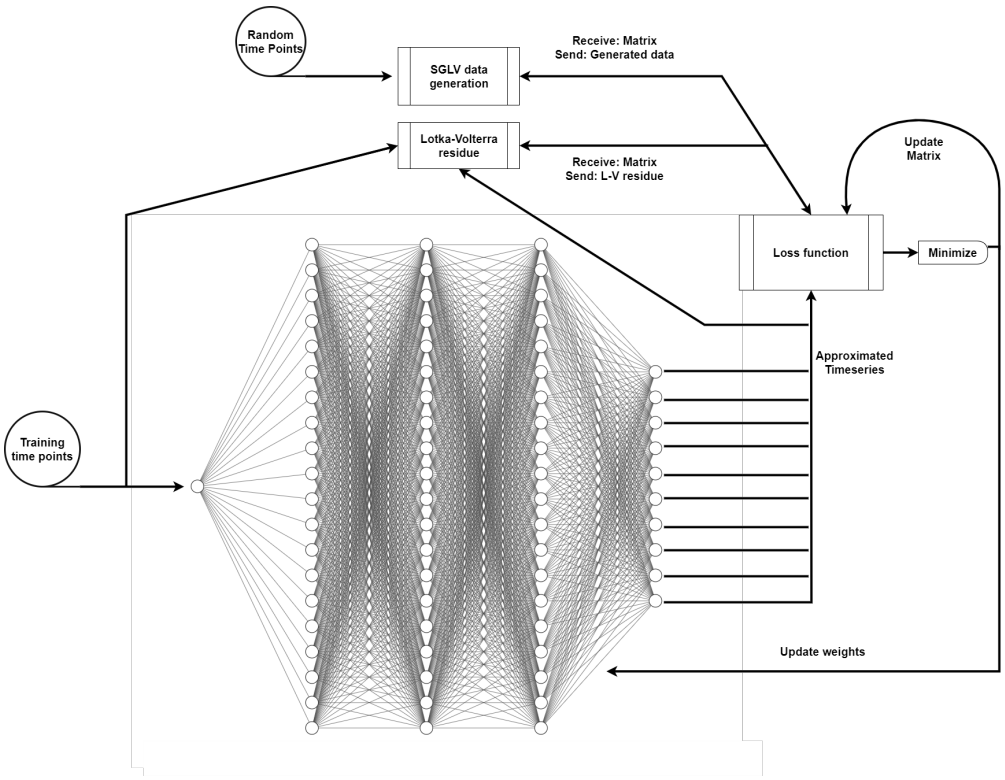

Figure 3.4: PINN architecture

The hyperparamters[9] as described by table 3.1, will be used to compare Physics Informed Neural Networks against other regression methods. These parameters are not the 'ideal' selection. The goal of this model is to perform adequately, fine-tuning these parameters was not done due to time constraints.

Layers	3
Neurons/layer	20
learning rate	5e-4
learning rate scheduler	lr*0.9*iter/epochs
optimizer	Nadam
epochs	20 000
activation function	sigmoid
kernel initializer	Glorot
bias in activation function	True
pre-processing	normalization
overfitting prevention	Dropout(0.05)

Table 3.1: Table of the PINN architecture

Some hyperparameters that were not discussed before or might not be well known will now be explained.

Learning rate is a parameter that regulates how much the neural network weights will change with respect to the optimizers output. A higher learning rate lets the model converge faster but it might not be able to learn details, a smaller learning rate might not be able to converge due to the fact that it cannot get out of local minima or get over local maxima. Generally, the more training data the model sees and the more neurons a network has the smaller this learning rate should be. Another trick that is frequently used in larger neural networks is a learning rate scheduler, this is a function that can change the learning rate throughout the learning process. Here a scheduler which linearly reduces the learning rate from 1e-3 to 1e-4 dependant on the iteration was used.

Activation functions are generally non-linear. These functions take the output from the previous layers' nodes and returns an output which will be passed to the next layer, generally between 0 and 1. The sigmoid activation function (3.13) is continuous and returns an output between 0 and 1. This is the second most frequently used activation function only surpassed by the Rectified Linear (ReLu) function. This is not a continuous function, which has an adverse effect on taking

[9]Parameters that are not learnt by the model and have to be manually set.

derivatives of the model's output with respect to the model's input.

$$S(x) = \frac{1}{1+e^{-x}} \quad \Big| \quad x = \sum_i w_i x_i \tag{3.13}$$

Initializing a neural network's weights can have a big impact on consistency and convergence time. Each type of activation function comes paired with an optimal method of initializing the model's weights. Glorot initialization (3.14) is dependant on the amount of inputs and outputs a neural network has.

$$w_{i,0} = \mathcal{N}(0, \frac{2}{N_{inputs} + N_{outputs}}) \tag{3.14}$$

Bias in an activation function is an extra learnable intercept (3.15) added to the input of the activation function.

$$x = \sum_i w_i x_i + e \tag{3.15}$$

The final hyperparameter that will be discussed is dropout. A problem practically every machine learning model has to deal with is overfitting. This is a situation where a model has perfectly learned the training data, but when presented with test data it underperforms. In neural neural networks there are many ways to prevent this. Here dropout was used. Simply put, every training cycle a percentage of the models nodes are removed from the network for one iteration, which forces the model to make use of all of its nodes and not abuse a small subset.

Chapter 4

An example

To show what sort of results every method produces, regressions were done on the example time series provided in section 2. Figure 4.1 provides multiple representations of every matrix that was inferred from the previously mentioned time series. The first column is a flattened plot of the inferred values of the matrix with respect to the real values. Column 2 shows the inferred matrices, and column 3 indicates whether a specific index was deemed as a detrimental interaction, beneficial interaction or non interacting.

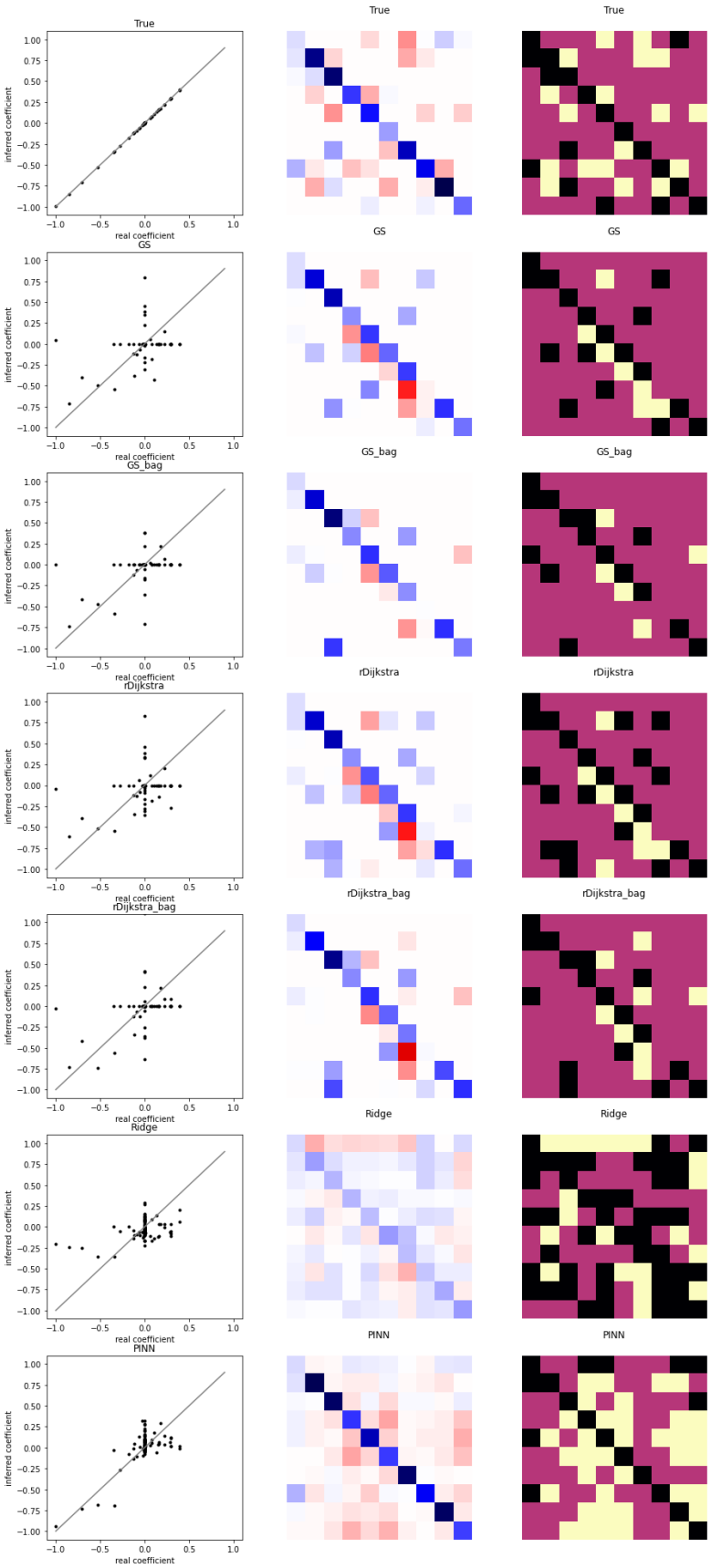

Figure 4.1: Different interaction matrix representations

Chapter 5

Statistical analysis

There are multiple ways to quantify how accurate an inferred interaction matrix is. One could look at the average euclidean distance, also known as Mean Squared Error (MSE), of the inferred matrix with respect to the real matrix. Another metric could be the amount of species that were correctly identified as interacting (positive or negative). And finally the MSE of the time series that can be generated from the inferred interaction matrix, with respect to the real time series. One could also look at how these quantities correlate with one another.

Comparing different methods interaction classification accuracy can be done by calculating their average specificity and sensitivity. Specificity is the amount of parameters correctly identified as non-interacting divided by the amount of parameters identified as non-interacting. Specificity is the amount of species correctly identified as interacting (sign included) divided by the amount of species identified as interacting.

Something both Stein et al [7] and Fischer et al [5] did not analyze in their respective papers is the significance of their results. For a single regression this can be done by taking the average value of the p-value of a multivariate t-test. The null-hypothesis can be formulated as: "The inferred interaction matrix is a diagonal matrix", in which case the alternative hypothesis would be: "The inferred interaction matrix is not a diagonal matrix". The null hypothesis would imply that the regression did not detect interspecies interaction. In their respective sections (5.1 LIMITS and 5.2 Ridge) the equations for the interaction matrix variance will be derived. This however can not be done for a simple deep learning model. There is no mathematical expression (as of June 2022) for the variance of a deep learning regression. One could 'brute force' the variance by running the model multiple times on a single time series. The initial weights of the network are randomized which will lead to different results every run. Although this is not a robust method

and it was not implemented due to time constraints. Another slight problem is the fact that training neural networks is a computationally demanding task. Ordinary personal computers are not suited for this task. Due to this problem the dataset for testing the physics informed deep learning implementation was limited to 100 samples. The dataset for comparing white box models was limited to 500 samples.

5.1 LIMITS

To derive a formula for the confidence of the LIMITS algorithm, a formula for the variance of the interaction coefficients in the interaction matrix is needed. In the case of LIMITS it is the same formula as a linear regression. As explained by [11] We start by defining the residual sum of squares (RSS).

$$\vec{\epsilon}_{RSS} = \sum_{i=0}^{n} (\vec{Y}(t_i) - \mathbf{M}_i \vec{X}(t_i))^2 \tag{5.1}$$

Now a conditional expectation value of the RSS can be defined.

$$\vec{\epsilon}_{cRSS} = E\{\vec{\epsilon}_{RSS} | \vec{X}\} \tag{5.2}$$

From this, a definition of the residual error variance can be derived.

$$Var\{\vec{\epsilon}_{RSS} | \vec{X}\} = \frac{(\vec{\epsilon}_{RSS} - \vec{\epsilon}_{cRSS})(\vec{\epsilon}_{RSS} - \vec{\epsilon}_{cRSS})^T}{n-k} \quad \Big| \quad k = \#\text{ species} \tag{5.3}$$

An important assumption of linear regression is the independence of the residual error with respect to $\vec{X}$. Now the previous equation reduces to:

$$Var\{\vec{\epsilon}_{RSS}\} = \frac{\vec{\epsilon}_{RSS}(\vec{\epsilon}_{RSS})^T}{n-k} \tag{5.4}$$

Furthermore we will assume that the RSS of the fitted model ($\vec{\epsilon}$) is an unbiased estimator.

$$\vec{\epsilon}(\vec{\epsilon})^T = (\vec{Y_i} - \hat{\mathbf{M}}_i \vec{X})(\vec{Y_i} - \hat{\mathbf{M}} \vec{X})^T \tag{5.5}$$

The inferred $\hat{\mathbf{M}}_i$ can be found by taking the partial derivative of the above equation with respect to $\hat{\mathbf{M}}_i$. This reduces to the formula:

$$\hat{\mathbf{M}}_i = (\vec{X}^T \vec{X})^{-1} \vec{X}^T \vec{Y_i} \tag{5.6}$$

To calculate the variance a last step is needed:

$$(\vec{X}^T \vec{X})^{-1} \vec{X}^T \vec{Y_i} = \mathbf{M}_i + (\vec{X}^T \vec{X})^{-1} \vec{X}^T \vec{\epsilon} \tag{5.7}$$

Now the final regression variance:

$$\begin{aligned} Var\{\hat{\mathbf{M}}_i\} &= E\{((\vec{X}^T\vec{X})^{-1}\vec{X}^T\vec{\epsilon})((\vec{X}^T\vec{X})^{-1}\vec{X}^T\vec{\epsilon})^T\} \\ &= \hat{\sigma}^2(\vec{X}^T\vec{X})^{-1} \quad \Big| \quad \hat{\sigma}^2 = \frac{\vec{\epsilon}(\vec{\epsilon})^T}{n-k} \end{aligned} \tag{5.8}$$

With the definition for $\hat{\mathbf{M}}$ and its variance we can apply a multivariate t-test.

$$\vec{Z} = \frac{\hat{\mathbf{M}}}{\sqrt{\hat{\sigma}^2}} \tag{5.9}$$

Here we compare the interaction matrix with a zero-filled matrix. If you assume that the noise present is normally distributed, then every Z_i should be t-distributed. The p-values can now be calculated by computing the cdf of the t-distribution for every value in $\vec{Z}$. These p-values represent the probability of finding a sample equal or larger than the absolute value of their respective Z_i's. If the average of these p-values is larger than 0.05 we will accept the null-hypothesis and reject the alternative hypothesis. The degrees of freedom are the amount of time points subtracted by the amount of species.

$$p_i = 2\int_{|Z_i|}^{\infty} f(u_i, df)\, du_i \quad \Big| \quad f(u) = \text{t-distribution} \quad \& \quad df = n-k \tag{5.10}$$

The final p-value is the mean of $\vec{p}_{od}$, where 'od' refers to the off-diagonal, non-zero indices.

For further testing we would like to know how LIMITS behaves with respect to its threshold. We would also like to know what the optimal threshold is for the specific dataset for every version of LIMITS.

Matrix error and time series error

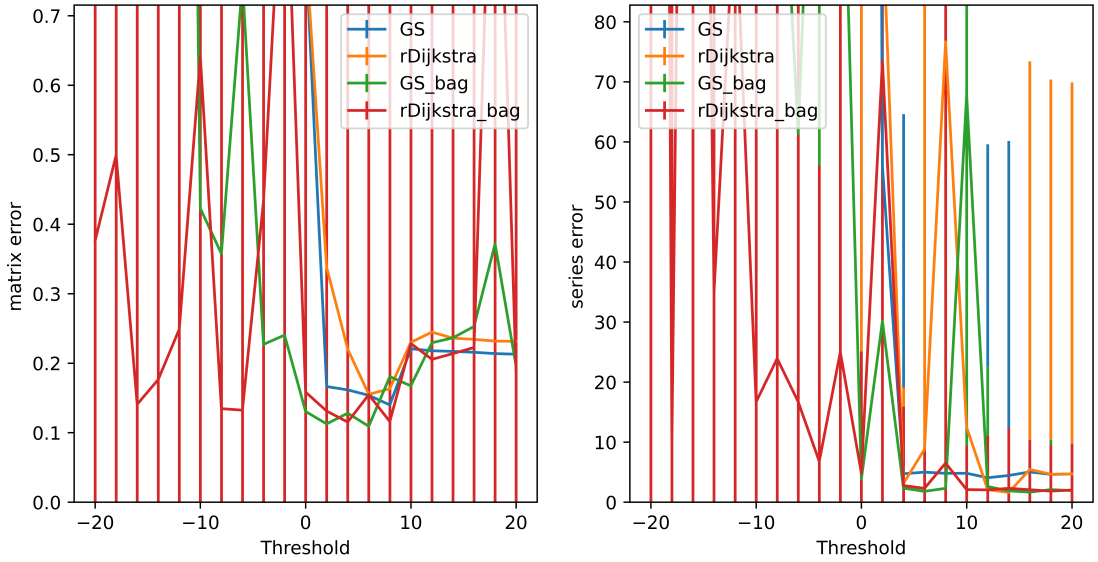

Figure 5.1: Plot of the matrix and series error with respect to the threshold.

Figure 5.1 reveals the inconsistent nature of LIMITS by displaying the relation between the threshold and the mean squared error of the interaction matrix and the time series that can be generated from it. The MSE variance at almost every threshold for every model can be an order larger than the MSE mean. The lower the threshold the higher the mean and variance. This might indicate that LIMITS's regression quality is largely dependant on the amount of perceived interacting species. Very high thresholds where LIMITS produces mostly diagonal matrices shows better behaviour for every method. One could find it strange that these MSE's grow smaller when the threshold increases. An explanation could be the behaviour of the uncertainty with respect to the threshold (figure 5.3 shows the average p-value of the inferred coefficients, which is positively correlated to the uncertainty of the inferred coefficient).

In the following equation: a is the inferred coefficient, σ^2 is the variance of this inferred coefficient (equation 5.8), and c is the real value of the coefficient. Here we assumed that the inferred coefficient was normally distributed.

$$E\{(x-c)^2\} = \frac{1}{\hat{\sigma}\sqrt{2\pi}} \int_{-\infty}^{\infty} (x-c)^2 e^{-1/2(\frac{x-a}{\hat{\sigma}})^2} dx = (a-c)^2 + \hat{\sigma}^2 \tag{5.11}$$

Higher inferred coefficient variance will on average lead to a larger error, as can be seen in equation 5.11. This could imply that the assumption in section 3.1.1 was wrong. We assumed that more 'turned on' dimensions would lead to a lower error, however we can see in figure 5.1, which can be explained by equation 5.11 (keep in mind that the variance positively correlates with the threshold), that this assumption is false. This might explain why the alternative LIMITS algorithm does not seem to outperform the standard procedure, and converge prematurely when the threshold gets smaller in figure 5.2. A side-note is that the squared error defines a metric, which cannot be smaller than 0. Do not get confused and assume that high standard deviation in figure 5.1 implies that the error can be smaller than 0.

Specificity and sensitivity

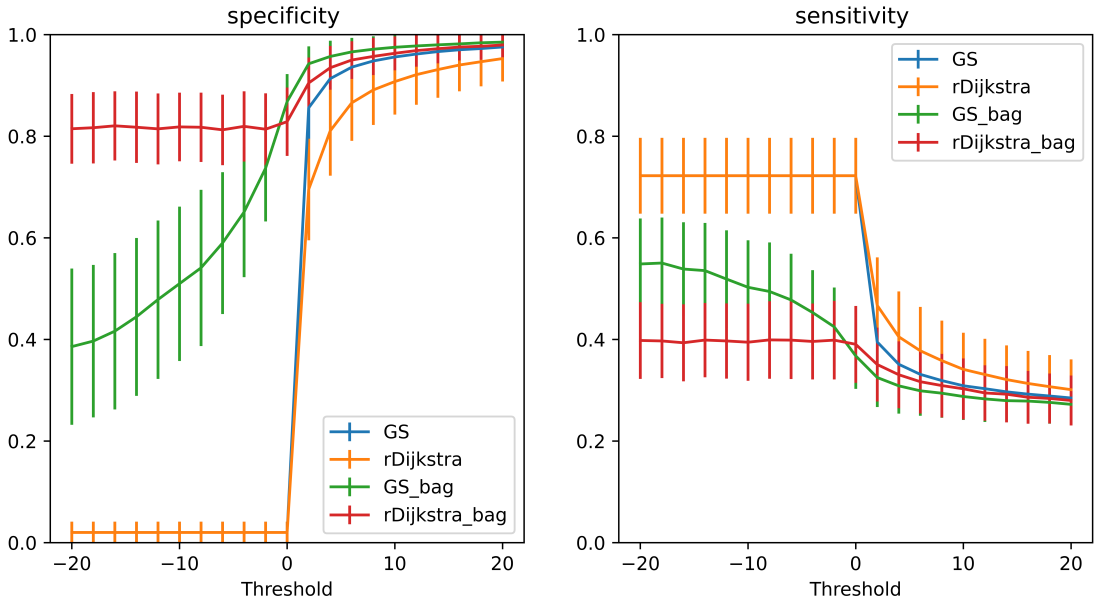

Figure 5.2: Plot of specificity and sensitivity with respect to the threshold.

To determine the threshold which will be used for the general analysis, the highest average sum of the specificity and sensitivity for every LIMITS model with respect

to threshold will be used. To be more exact than the step size used for figure 5.2, second order polynomial interpolation was used to determine a more precise value. Table 5.1 shows the ideal threshold for every LIMITS version.

LIMITS version	threshold
Greedy search	1.14
Greedy search with bagging	1.45
Reverse Dijkstra	-10.35
Reverse Dijkstra with bagging	-1.39

Table 5.1: Table of the ideal thresholds per LIMITS version

P-value

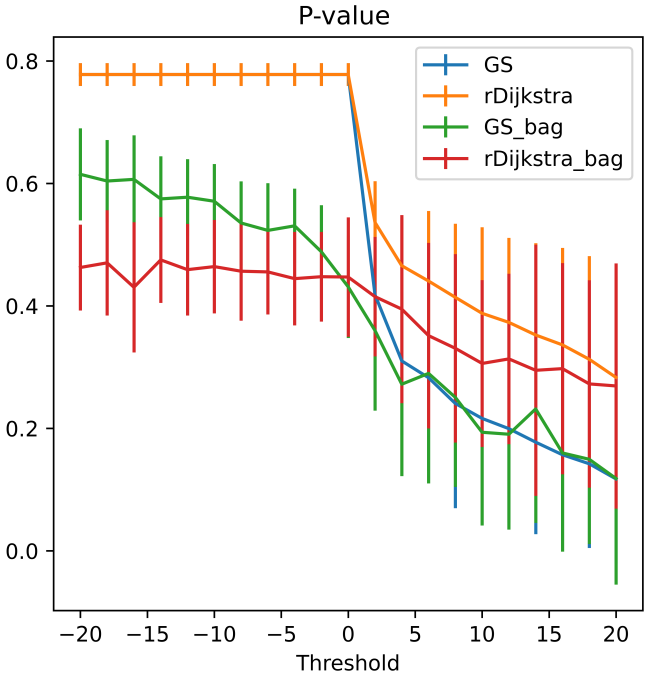

Figure 5.3: Plot of the p-value with respect to the threshold.

Lastly, figure 5.3 displays the behaviour of the p-value with respect to the threshold for every LIMITS version. As one can see not a single version for any threshold surpasses 0.05. This indicates that we could accept the null-hypothesis which was proposed in section 5. Now we could also test whether a diagonal linear regression is valid, which means that the null hypothesis would now be: "The interaction

matrix is a zero-matrix". The alternative hypothesis be: "The interaction matrix is not a zero-matrix". Testing this can be done by again using equation 5.10 where p is the mean of $\vec{p_d}$[1]. This results in a p-value of 0.029 ± 0.06. The fact that 0.05 is within 1 standard deviation away from the found p-value shows that we could potentially accept the null-hypothesis. A reason this might be the case is due to the growth rate approximation in equation 3.1. Since the time series include transient phases, there is a possibility that the steady-states are not equal to the mean of the time series. This is quite concerning since this approximation is also used in the Stein et al Ridge regression paper [7].

5.2 Ridge regression

The formula for the variance of a ridge regression is very similar to the variance of a linear regression. The derivation however is quite long and can be found at [12].

$$Var\{\hat{\mathbf{M}}_i\} = \hat{\sigma}^2(\vec{X}^T\vec{X} + diag(\lambda))^{-1}\vec{X}^T\vec{X}(\vec{X}^T\vec{X} + diag(\lambda))^{-1} \tag{5.12}$$

The p-values are again calculated by doing a t-test. The degrees of freedom however are different from a linear regression. They are approximated by:

$$p_i = 2\int_{|Z_i|}^{\infty} f(u_i, df)\, du_i \quad \Big| \quad f(u) = \text{t-distribution} \quad \& \quad df = \sum_i \frac{d_i}{d_i + \lambda} \tag{5.13}$$

Here d_i is the single value decomposition of $\vec{X}$. The final p-value is the mean of $\vec{p_{od}}$[2].

5.3 Physics Informed Neural Network (PINN)

As in section 3.2.1 was mentioned, an ODE solver was added in the loss function to generate additional data. In table 5.2 the matrix and time series error with- and without the data generator were tested on a dataset of 10 random samples. Generally adding a time series generator lowered the error on the inferred interaction matrix. The time series that can be generated from these matrices were more consistent with the real time series. As mentioned on page 17, the PINN architecture is not yet informed enough to give confidence intervals. Yang et al introduce a Bayesian physics informed deep learning framework in [13], which could potentially solve the problem of confidence intervals.

[1] diagonal indices

[2] The off-diagonal non-zero indices.

Metric	Without ts gen	with ts gen
Matrix error	0.16 ± 0.02	0.04 ± 0.017
Time series error	52.97±52.71	0.69±0.36

Table 5.2: Table of PINN strategies comparison

5.4 General analysis

In this section we will take a look at the results every regression method produces. The dataset for the PINN model was limited to the first 100 samples, whilst the dataset for the other methods contained the full 500 samples.

Method	mean me	std me	mean se	std se	div rate	mean p-val	std p-val
LIMITS GS	1.67	7.19	16.9	70.30	0.03	0.48	0.13
LIMITS GS bag	1.96	7.87	57.96	205.24	0.16	0.66	0.09
LIMITS rD	3.13	7.19	21.32	87.47	0.1	0.54	0.10
LIMITS rD bag	0.51	7.19	299.93	1408.71	0.16	0.48	0.11
Ridge	0.14	0.04	123.08	633.140	0.03	0.008	0.02
PINN	0.04	0.01	2.11	4.05	0.03	x	x

Table 5.3: Error analysis of every method

Table 5.3 gives a full overview of the regressed interaction matrix error and corresponding time series error, as well as the p-value for every model. The model with the lowest interaction matrix error as well as the lowest time series error is the PINN model as described in section 3.2.1. Also notice that time series generated from the inferrred interaction matrix of the PINN model together with standard greedy search and ridge regression only diverged 3% of the time. The only white box method which passed the null-hypothesis is Ridge regression. Generally the proposed alterations to the LIMITS algorithm, as discussed in section 3.1.1, as well as bagging, performed worse. Looking at the results in table 5.4 shines a

Method	mean spec	std spec	mean sens	std sens
LIMITS GS	0.74	0.10	0.45	0.07
LIMITS GS bag	0.52	0.14	0.51	0.08
LIMITS rD	0.58	0.10	0.50	0.10
LIMITS rD bag	0.80	0.06	0.41	0.07
Ridge	0.14	0.16	0.55	0.16
PINN	0.09	0.04	0.73	0.07

Table 5.4: Specificity and sensitivity of every method

different light on the LIMITS algorithm. When taking the average value of the specificity and sensitivity, reverse Dijkstra with bagging slightly edges out the standard greedy search LIMITS algorithm. The worst performing method overall was Ridge regression. Notice however when looking at table 5.2, that the PINN model is able to achieve a high accuracy in determining the bacterial species that do interact whilst not having to trade-off on low errors.

Chapter 6

Discussion

The goal of this bachelor thesis was to analyze how statistically relevant popular regression techniques applied to the stochastic generalized Lotka-Volterra equation (sgLV) are. We were also interested in how well basic Physics Informed Neural Networks (PINN's), and PINN's boosted with a time series generator in the loss function could perform.

It was revealed that the linearization technique shown in section 3.1.1 might not be robust when the time series was generated using the Euler-Maruyama method. This might have impacted the quality of the Ridge regression and mainly the significance of the LIMITS algorithm. Another factor that might have impacted the results is the fact that the time series used included a transient phase [1], and could have an adverse effect on equation 2.4. This is due to the fact that the average of a specific bacterial abundance might be vastly different than the stable abundance, which would throw off the growth rate and might affect the confidence of the regression. This could be fixed by removing the growth rate approximation in equation 3.1 and adding an intercept to the linear regression, which would represent the growth rate. In itself this could potentially be tested in a future bachelor thesis on the subject of studying the dynamics of intestinal microbes. Since Fischer et al [5, p. 7-8, fig.6] did not take the significance of their method into account, they wrongly assumed their model was applicable to in vivo data.

We also demonstrated that PINN's results display very low errors in inferring interaction matrices and the respective time series that can be generated from these matrices. Adding a time series generator to the loss function improved the matrix error four fold whilst the time series error lowered a full order, and ratio of mean and variance was reduced by 50%. A crucial part that cannot be measured

[1] Unstable phase before equilibrium.

currently is the confidence level of the PINN's inferred interaction matrices. This could however be achieved by implementing the concept of Bayesian neural networks or stochastic physics informed neural networks. As of June 2022, no paper has been published on the subject of Bayesian or stochastic physics informed deep learning applied to the field of microbial dynamics. Time series analysis, in the case of deep learning, is usually done by implementing Recurrent Neural Networks (RNN's). The networks take time-dependency into account by connecting the output of the network with the input; it tries to learn the time series data by iterating over the time-points. This procedure is quite similar to the Euler–Maruyama method, as used in equation 2.7. These 'smarter' alternatives could be an interesting concept for a master thesis.

Lastly, something that was observed while testing the PINN model, without time series generation in the loss function, is that it was able to accurately recreate time series, but the inferred interaction matrix did not resemble the real one in any way. This might indicate that small time series with noise lack uniqueness. If that were to be the case there would be no accurate way to reconstruct the interaction matrix from a single time series. This would not imply that it's impossible to determine the interaction matrix from multiple time series from the same microbial community.

Bibliography

[1] W. Tao, G. Zhang, X. Wang *et al.*, "Analysis of the intestinal microbiota in COVID-19 patients and its correlation with the inflammatory factor IL-18," *Medicine in Microecology*, vol. 5, p. 100 023, Sep. 2020, ISSN: 25900978. DOI: 10.1016/j.medmic.2020.100023. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2590097820300203 (visited on 16/05/2022).

[2] X. Mao, S. Sabanis and E. Renshaw, "Asymptotic behaviour of the stochastic lotka–volterra model," *Journal of Mathematical Analysis and Applications*, vol. 287, no. 1, pp. 141–156, Nov. 2003, ISSN: 0022247X. DOI: 10.1016/S0022-247X(03)00539-0. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0022247X03005390 (visited on 16/05/2022).

[3] *Sdeint 0.2.4*. [Online]. Available: https://pypi.org/project/sdeint/.

[4] O. Loyola-Gonzalez, "Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view," *IEEE Access*, vol. 7, pp. 154 096–154 113, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2949286. [Online]. Available: https://ieeexplore.ieee.org/document/8882211/ (visited on 16/05/2022).

[5] C. K. Fisher and P. Mehta, "Identifying keystone species in the human gut microbiome from metagenomic timeseries using sparse linear regression," *PLoS ONE*, vol. 9, no. 7, B. A. White, Ed., e102451, 23rd Jul. 2014, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0102451. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0102451 (visited on 16/05/2022).

[6] E. W. Dijkstra, "A note on two problems in connexion with graphs. numerische math- ematik,," pp. 269–271, 1959.

[7] R. R. Stein, V. Bucci, N. C. Toussaint *et al.*, "Ecological modeling from time-series inference: Insight into dynamics and stability of intestinal microbiota," *PLoS Computational Biology*, vol. 9, no. 12, C. von Mering, Ed., e1003388, 12th Dec. 2013, ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1003388.

[Online]. Available: https://dx.plos.org/10.1371/journal.pcbi.1003388 (visited on 16/05/2022).

[8] E. Kavlakoglu. "AI vs. machine learning vs. deep learning vs. neural networks: What's the difference?" IBM article. (2020), [Online]. Available: https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks.

[9] M. Raissi, P. Perdikaris and G. E. Karniadakis, "Physics informed deep learning (part II): Data-driven discovery of nonlinear partial differential equations," 2017, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.1711.10566. [Online]. Available: https://arxiv.org/abs/1711.10566 (visited on 16/05/2022).

[10] J. O'Leary, J. A. Paulson and A. Mesbah, "Stochastic physics-informed neural networks (SPINN): A moment-matching framework for learning hidden physics within stochastic differential equations," 2021, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2109.01621. [Online]. Available: https://arxiv.org/abs/2109.01621 (visited on 16/05/2022).

[11] D. Sachin. "Time series analysis, regression and forecasting," A Deep Dive Into The Variance-Covariance Matrices Used In Linear Regression. (), [Online]. Available: https://timeseriesreasoning.com/contents/deep-dive-into-variance-covariance-matrices/.

[12] W. N. van Wieringen, "Lecture notes on ridge regression," p. 12, 2015, Publisher: arXiv Version Number: 7. DOI: 10.48550/ARXIV.1509.09169. [Online]. Available: https://arxiv.org/abs/1509.09169 (visited on 26/05/2022).

[13] L. Yang, X. Meng and G. E. Karniadakis, "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data," 2020, Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2003.06097. [Online]. Available: https://arxiv.org/abs/2003.06097 (visited on 27/05/2022).