PROVISIONAL TITLE

Jan Herlyn, Adrià Lisa

Abstract—In this project we continue the exploration of time series data with a focus on interpretability, required prior knowledge and limitations of time series models. We compare results obtained from fitting a SARIMAX model to a Lotka-Voltera function learned by a neural ODE model in the context of canadian lynx and snowshoe hare data from the 19th century. We then analyze what general statements we can make about the system dynamics based on the parameters of our models.

Keywords—*Machine Learning, Time Series, Neural ODE*

1. Introduction

1.1. Statistical Learning in Time Series

Analysis of time series has a rich scientific history dating back to the 17th century [8]. It treats the prediction of future values based on past observations. A wide range of methodologies exists ranging from simple **A**uto **R**egressive models that model the next value as a linear combination of past observations to complex **L**ong **S**hort-**T**erm **M**emory [1] networks that predict future observations using recurrent networks.

Each method comes with it's own drawbacks and advantages. In this report we explore a very simple linear model SAR(I)MAX and compare it to a very advanced but surprisingly interpretable model called Neural Ordinary Differential Equations.

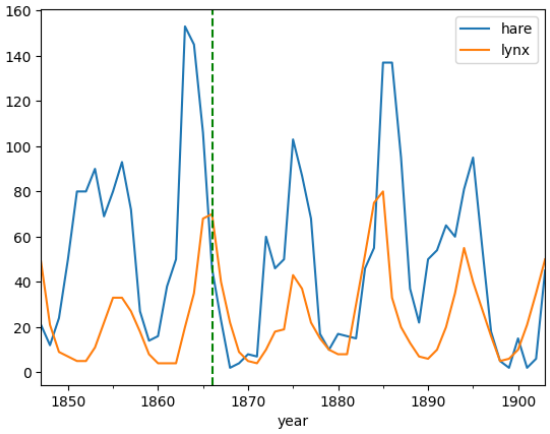

Figure 1. Yearly Snowshoe Hare and Canadian Lynx populations in 1000s of animals. The green dashed line represents the separation between train and test.

2. SAR(I)MAX

The SARIMAX model is a classical linear model for time series.

$$X_t - \alpha_1 X_{t-1} - \cdots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q},$$

AR(p) The autoregressive part of the SARIMAX model is perhaps the most intuitive. It states that the value of the model at time t is a linear function of the last p observations plus an error term:

$$X_t = \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

The AR model by itself, however, is limited in which types of data can be accurately described. It is required that the mean and variance of the time series stays constant over time or in other words that it is weak-sense stationary. Thus larger trends or seasonality can't be modeled correctly.

I(d) The autoregressive model can be extended to account for trends. This can be solved by performing differencing on the data before applying the auto regressive portion. Thus we create the modified series y' with

$$y'_t = y_t - y_{t-1}$$

Differencing can be extended to more than one order.

The data we explored, however, did not exhibit any trend characteristics so we did not add this step to our model.

MA(q) The moving average model, describes the current value of the time series as a linear combination of the **error terms** of the past observations.

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} = \mu + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

Ad hoc experiments showed that our models generally performed worse when adding a moving average parameter to our model.

Seasonality So far, the parts of the models we described did not take into account seasonality. In our case, we look at the year by year data, so there are no clear external factors that would indicate that the populations would be seasonal instead of cyclic. However, when looking at the data, we can see that there are peaks roughly every 9 years, so we also explored adding a cyclic component of every nine years.

Exogenous Variables In addition to the previous values of our time series, we can also add exogenous variables which impact the value of the current time series. In our case, we set the estimated lynx population as an exogenous variable for estimating the hare population.

Putting it together The full SARIMA model can be described as

$$\begin{aligned} Y_t - \delta_0 - \delta_1 t - \ldots - \delta_k t^k - X_t \beta & \\ &= \varepsilon_t (1-L)^d (1-L^s)^D \Phi(L) \Phi_s(L) \varepsilon_t \\ &= \Theta(L) \Theta_s(L) \eta_t \end{aligned}$$

With L being the lag operator, $\eta_t \sim WN(0, \sigma^2)$ being the white noise process, and $G(L)$ the lag polynomials corresponding to the autoregressive (Φ), seasonal autoregressive (Φ_s), moving average (Θ), and seasonal moving average components (Θ_s).

3. The Lokta-Volterra differential equations

These differential equations 1 are a widely known theoretical model for the population changes over time of a predator species, $x(t)$, and a prey species, $y(t)$.

$$\begin{aligned} \dot{x} &= \alpha x - \beta xy \\ \dot{y} &= \gamma xy - \delta y \end{aligned} \tag{1}$$

There are two linear terms, whose parameters α, δ represent the growth rate of the prey and the decay rate of the predator. The non-linear terms represent the interaction between the two species. The rate at which the prey is being hunted is represented by β. The rate at which the predators are able to reproduce due to the abundance or lack of preys to hunt on is γ.

Any solution this model leads to an oscillatory trend, where the two populations alternate between growth and decay eras. Therefore, for this model, the extinction state is unreachable [3].

3.1. A note on numerical differential equation solving

Differential equations need to be solved numerically. There is a very rich mathematical literature on this topic [14], with very sophisticated methods widely available at many software libraries. Consider the following ordinary differential equation (ODE) in general form:

$$\frac{d\boldsymbol{u}}{dt} = f(\boldsymbol{u}, t) \tag{2}$$

In general, these methods are given some initial value $\boldsymbol{u}_0$ and provide a discretization of the solution $\{\hat{u}_0 = \boldsymbol{u}_0, \hat{u}_1, ..., \hat{u}_n, \}$ over a finite time-span $\{t_0, ..., t_n\}$. The majority of methods are actually extensions or twists from the most basic one: Euler. Eulers's method computes the discretization as follows:

$$\hat{u}_{i+1} = \hat{u}_i + (t_{i+1} - t_i) \cdot f(\hat{u}_i, t_i) \tag{3}$$

When attempting to solve certain ODE problems numerically, inappropriate step-sizes $\{t_{i+1} - t_i\}$ or high-frequency oscillations in the solutions can cause the methods to become numerically unstable. This phenomena is called *stiffness*, and it has been the cause of many headaches in our work.

4. Neural ODEs

There is a particular deep Neural Network (NN) architecture known as ResNETs [2] for which every layer or block h_b is computed as follows:

$$\boldsymbol{h}_{b+1} = \boldsymbol{h}_b + \mathcal{F}(\boldsymbol{h_b}, \boldsymbol{\theta}) \tag{4}$$

Where F is some functional, trainable with respect to parameters θ. The outstanding similarities between the equations of this learning model (4) and Euler's method for differential equations (3) led [4] to invent a new learning model that would combine both worlds. This new model will later on be called "Neural Ordinary Differential Equations" (NeuralODE) [6]. We recommend reading the survey by Patrick Kidger on NeuralODEs [13] for more information on this topic.

Essentially, a NeuralODE is a ResNET defined implicitly by a trainable differential equation.

$$\frac{d\boldsymbol{h}}{dt} = f(\boldsymbol{h}, t, \theta) \tag{5}$$

For which the inputs are assigned to the initial-time state, $\boldsymbol{h}(0) = h_0$, and the outputs are the final-time state, $\boldsymbol{h}(1)$.

Note that the implicit definition of this model contemplates a continuum of layers $h(t) \forall t \in [0, 1]$, but 5 will always be solved explicitly by a numerical method that will be yielding a discretization $\{t_0 = 0, t_1, ..., t_{n-1}, t_n = 1\}$.

Therefore, for our computers, this will be somewhat equivalent to an n-depth ResNET. But n will only depend on the ODE solver, so it can be arbitrarily large without increasing the number of parameters θ in 5.

There are several results on Universal Approximation Theorems [7] [11] [12]. For this project it will suffice to claim that having a sufficiently large NN $f(\boldsymbol{h}, \theta)$ as the vector field in 5 will guarantee that our Neural ODE can approximate any function.

5. Methods

5.1. Data Analysis

We divide our dataset into a train subset, consisting on the first 20 years, and a test subset consisting on the remaining 37 years.

We do not conduct any shuffling of the data, as the temporality is very important. However, we did normalize the training data for the neural ODE by dividing every point w.r.t. the largest entry. To perform the testing we *de-normalized* the outputs of our model, and compared them to our original data.

We consider a *baseline* model that would just output the mean values of every species, $\mu_{hare} = 50.561$ and $\mu_{lynx} = 24.035$. The MSE for that model would be 1915.368, corresponding to an average error of 47.764.

5.2. Fitting SARIMAX to Hare and Lynx populations

The SARIMAX parameters can be obtained using the generalized least squares method. The challenge lies in deciding the hyper-parameters $(p, d, q) \times (P, D, Q)$ and in our case the seasonality s (s is normally assumed to be known, ie 1 year of observations).

To this end, there are some heuristics that can be employed like the autocorrelation and partial autocorrelation of the training data shown in figure 2

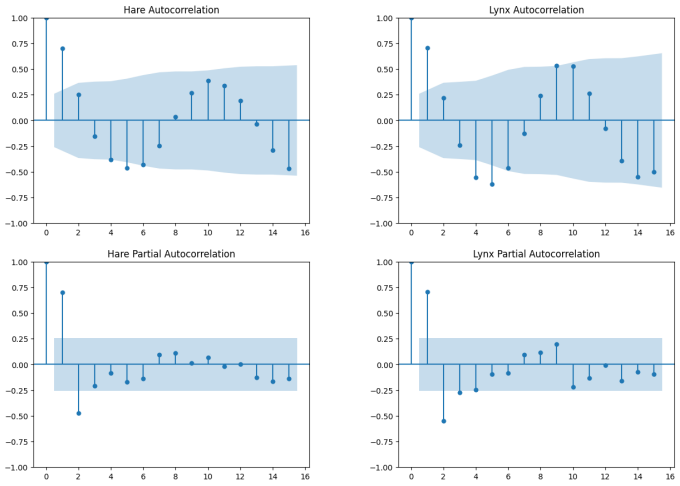

Figure 2. (Partial) Autocorrelations of Lynx and hares

Since we didn't observe a trend, we decided to set the integrated parameters d and D to 0. s was obtained by looking at the distance between the peaks and set to 9. The autocorrelations showed statistically significant correlations for both the hares and the lynxes up the to 5th lagged value. This gave some starting point of how many values to consider. For the seasonal component, we could at most consider 1 season in the past because of the low amount of training data thus P was set to at most 1.

To make the results between the SARIMAX and the Neural ODE comparable, we decided to predict the entire training data set instead of just the next year respectively.

5.2.1. Lynx

We then evaluated different combinations of values and found that for the lynx, predictions work remarkably well for a simple AR model of 2 parameters.

In all cases, adding a moving average parameter worsened the predictions.

Adding a seasonal component to our AR(2) model considerably improved the mean square error.

5.2.2. Hares

For the hares, predictions were considerably worse. Especially predictions with just the hare dataset by itself did not yield reasonable results. The best model we could find was an AR(1) model which was essentially flat, thus predicting the mean. These results were even worsened by adding a seasonal component as well as the lynx populations prediction as exogenous variable. If we compare the mean of the best model for our hare predictions it is only marginally better than the baseline, thus showing that given the small amount of training data for hares, SARIMAX wasn't the right fit to predict future hare populations.

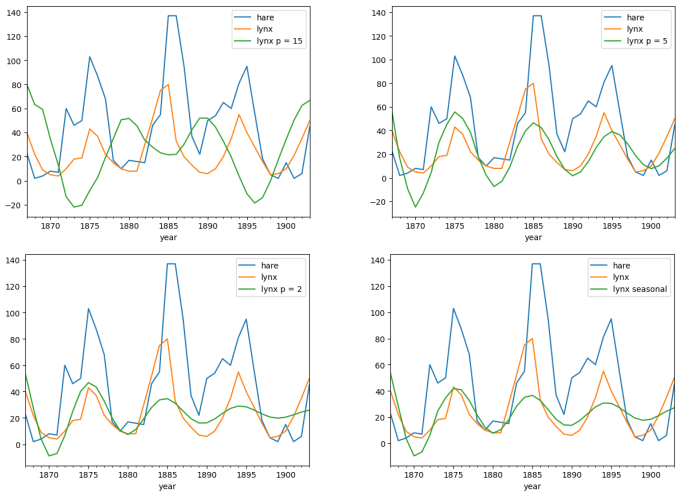

Figure 3. Lynx test data set and predictions

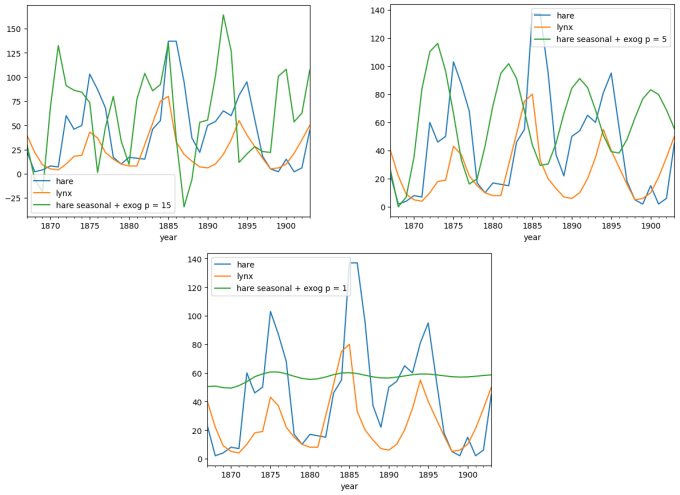

Figure 4. Hare seasonal models with lynx populations as exogenous variable

5.3. Learning a pure Lokta-Volterra model from data

As discussed in the previous assignment, we can compute some values for the parameters in 1 that would somehow fit the resulting trajectories into our dataset.

As the optimization requires to solve a differential equation twice for every epoch, the usage of a fast numerical ODE method is critical. Moreover, we also encountered problems with *stiffness*. In this project, we benchmarked many of the numerical ODE solvers for stiff problems that the Julia package `OrdinaryDiffEq.jl` [5] offers.

In the end, we decided for the composite solver `AutoTsit5(Rosenbrock23())`. By default it will run the flagship method `Tsit5()` (a.k.a. `ode45` in MATLAB), a Runge-Kutta method of order 5 that maches speed and precision, and it will automatically switch to the `Rosenbrock23()` method whenever high oscillations are detected.

The values we found for the parameters were $\alpha = 0.705, \beta = 2.139, \delta = 0.707, \gamma = 2.028$. The resulting trajectories can be seen in figure 5. The qualitative interpretation of this results is that the linear growth/decay rates of the two species are virtually the same ($\alpha \approx \delta$), meaning that hares grow as fast as lynx decay. From the fact that $\beta > \gamma$ follows that the interaction of the two species has a greater effect on the hares, causing it to decrease more rapidly when the lynx population begins to grow.

The MSE of the resulting trajectories compared to values from the dataset was 501.106, with an average error of 19.699 and 24.782 (thousands of animals) for the hares and the lynx, respectively. The

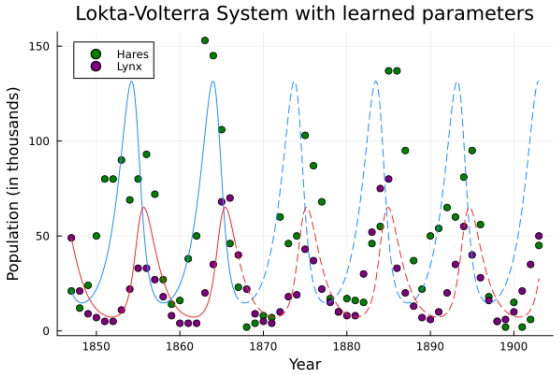

Figure 5. Projection of the Lokta-Volterra model. The paramaters obtained were directly optimized using the data from 1847 to 1866.

gap between the errors for the two species using this model was relatively small, compared to the others. This makes sense intuitively, as the differential equations 1 reflect a strong dependence between hares and lynx.

5.4. Fitting a NeuralODE

Our first NeuralODE model consisted on a pure NN vector field. We used the SCIML framework [9] from the `Julia` programming language, on which the architecture was expressed as follows:

```
Chain(Dense(2,5,rbf), Dense(5,5, tanh), Dense(5,2))
```

That is, a 2-deep NN with hidden with 5. The activation functions are *radial basis* (e^{-x}) and *hyperbolic tangent* ($tanh(x)$). The idea for this architecture came form looking at the examples of the SCIML documentation wiki.

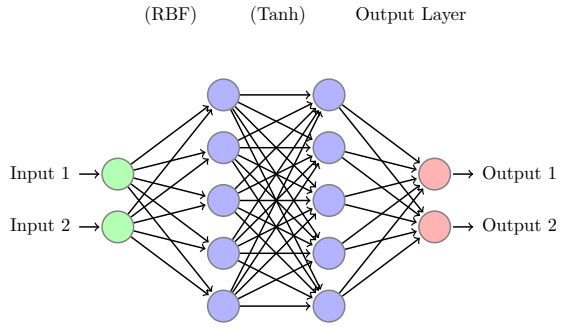

Figure 6. Architecture of the NN used for our NeuralODE.

The training process was very straightforward. We define our loss function as the mean square error with the data points. The optimization algorithm used was *ADAM*, with an initial learning rate of 0.01, and 500 training epochs. Once precompiled, the training phase only took a few seconds to run, and the convergence was quite fast as can bee seen in Figure 7.

Re-scaling the model and comparing it with the test values, the MSE was 495.734, with an average error of 24.589 and 19.325 (thousands) for the hares and the lynx, respectively. It can be seen in 8 that we are able to find good curves, but the orbital periods are harder to find, specially for the hare population.

Finally, a comment about the architecture. Decreasing the network size, both in terms of width and height, was possible, but impractical as it hindered the training process dramatically.

To give just one example, we considered the same NeuralODE model with the hidden dimension being 4 instead of 5 for both hidden layers, which reduces the number of parameters from 57 to 42. After several training phases with different learning rates, it took 2500 epochs to train a NeuralODE with a hidden width of 4 to the same point that the previous NeuralODE 6 reached in just 500 epochs.

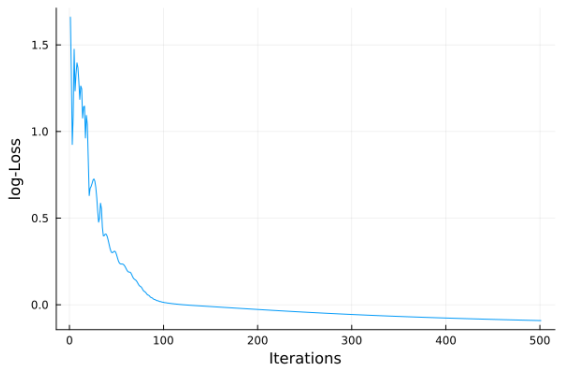

Figure 7. Evolution of the $\log_{10}(MSE)$ during the training phase.

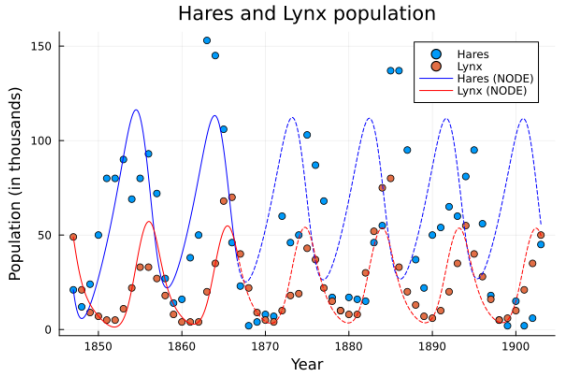

Figure 8. Trajectories of the NeuralODE model with hidden width 5.

5.5. Fitting a hybrid NeuralODE

The name and the idea of our last model comes from the paper by Rackauckas et.al [10]. It combines the Lokta-Volterra equations (1) and a NN (6) to model the vector fields, as eq. 6 describes.

$$\begin{aligned} \dot{x} &= \alpha x - \beta xy + NN_1(x, y, \theta) \\ \dot{y} &= \gamma xy - \delta y + NN_2(x, y, \theta) \end{aligned} \tag{6}$$

The goal was to learn the growth/decay rates $p = [\alpha, \beta, \gamma, \delta]$, and have the NN correct the noise and errors of the real data to obtain a model that would be both insightful and precise.

In the end, we realized that idea was an entelechy. The learned values for Lokta-Volterra parameters p made no sense, and the results we obtained were identical to 8, indicating that all the "learning" was occurring only at the NN.

In an attempt to reduce this effect, we introduced an additive term in the loss function that would penalize the NN (7). The measure was to take a vector-norm of all the NN parameters, $||\theta||_2^2$, and have all the activation functions be changed to tanh, as $e^{-0} > 0$ but $\tanh(0) = 0$. This would ensure that if $||\theta||_2^2 \to 0$ then $||NN(x, y, \theta)||_2^2 \to 0$.

$$\mathcal{L}(p, \theta) = \left\| \begin{pmatrix} \dot{x}(t_i, p, \theta) \\ \dot{y}(t_i, p, \theta) \end{pmatrix} - \begin{pmatrix} x(t_i) \\ y(t_i) \end{pmatrix} \right\|_2^2 + \xi ||\theta||_2^2 \tag{7}$$

With this setup, independently of the initial values for p or the NN complexity parameter $\xi > 0$, after the training, the resulting values for p were always the same: $p = [0.0, 0.0, 0.0, 0.0]$. Decreasing the network size also led to the same result.

6. Conclusion

Table 1 shows the cross-validation results we obtained with the proposed methods.

Unlike for the hare data, the linear SARIMAX method worked surprisingly well for the Lynx dataset. The results of the parameters, however, give little in the way of explaining the dynamics of the system and thus we deem the interpretability low for this method. While SARIMAX is a linear method, deciding the number of parameters for the autoregressive, integration and moving average components is also non trivial. One advantage of the SARIMAX model was that it was quite easy to decide how many known datapoints to include in the predictions. While out of the scope of our report, we did conduct some experiments showing that predicting year over year did lead to much better results.

Method	Hare MSE	Lynx MSE
Baseline	1029.574	365.649
Best AR(k)	908.585	222.846
Best SAR(k)	N/A	197.769
Best SARX(k)	1930.316	N/A
Pure LV	430.769	646.967
NeuralODE width 4	571.788	433.194
NeuralODE width 5	618.000	373.468
Hybrid NeuralODE	552.963	404.187

Table 1. Mean Squared Error of all the methods considered, measured in "thousands of animals".

The Pure Lokta-Volterra parameter estimation was a hard nonlinear optimization problem. The process had to be supervised, as for some values of the parameters the solver would crash down or get stuck on trajectories resembling a straight line.

But once some reasonable values for $\alpha, \beta, \gamma, \delta$ are found, the differential equations 1 are the simplest non-linear model with a very high value in terms of interpretability. The downside would be that it is not fit for automatized production, and the accuracy is inferior.

Regarding the NeuralODE models, we deem all the results to be similar. There was no time to do an iterative testing on the models, but we observed that changes on the initial conditions (like changing the RNG seed) could lead to changes in these MSE results of the same order than the differences between these models. One invariant pattern for the results of 3 NeuralODE models is that they are more accurate on predicting the less noisy lynx data.

Still, the NeuralODE of reduced width has significantly harder to train, and the Hybrid NeuralODE had redundant parameters. For that reason, we consider the NeuralODE of width 5 as the victor.

The SAR(I)MAX models had the best accuracy when predicting the lynx data, which shows a clearer and less noisy pattern. The differential equations and NeuralODE models were able to reduce the gap between hare MSE and lynx MSE, as they incorporate the hypothesis of dependence between the two population, which proves to be very helpful for finding a pattern for the more chaotic hares data.

References

[1] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 1530-888X. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, *CoRR*, Dec. 2015. DOI: 10.48550/arxiv.1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[3] C. Lobry and T. Sari, “Migrations in the rosenzweig-macarthur model and the "atto-fox" problem”, *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, vol. Volume 20 - 2015 - Special... Nov. 2015. DOI: 10.46298/arima.1990.

[4] Y. Lu, A. Zhong, Q. Li, and B. Dong, "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations", *CoRR*, Oct. 2017. DOI: 10.48550/arxiv.1710.10121. [Online]. Available: http://arxiv.org/abs/1710.10121.

[5] C. Rackauckas and Q. Nie, "DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia", *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[6] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations", *CoRR*, Jun. 2018. [Online]. Available: https://arxiv.org/pdf/1806.07366.pdf.

[7] Q. Li, T. Lin, and Z. Shen, "Deep learning via dynamical systems: An approximation perspective", *Journal of the European Mathematical Society*, Dec. 2019. DOI: 10.48550/arxiv.1912.10382. [Online]. Available: https://arxiv.org/abs/1912.10382v2.

[8] A. Nielsen, *Practical Time Series Analysis: Prediction with Statistics and Machine Learning*. O'Reilly Media, 2019, ISBN: 9781492041603. [Online]. Available: https://books.google.es/books?id=xNOwDwAAQBAJ.

[9] C. Rackauckas, M. Innes, Y. Ma, J. Bettencourt, L. White, and V. Dixit, "Diffeqflux.jl-a julia library for neural differential equations", *arXiv preprint arXiv:1902.02376*, 2019.

[10] C. Rackauckas, Y. Ma, J. Martensen, *et al.*, *Universal differential equations for scientific machine learning*, 2020. DOI: 10.48550/ARXIV.2001.04385. [Online]. Available: https://arxiv.org/abs/2001.04385.

[11] T. Teshima, K. Tojo, M. Ikeda, I. Ishikawa, and K. Oono, "Universal approximation property of neural ordinary differential equations", *CoRR*, vol. abs/2012.02414, 2020. arXiv: 2012.02414. [Online]. Available: https://arxiv.org/abs/2012.02414.

[12] I. Ishikawa, T. Teshima, K. Tojo, K. Oono, M. Ikeda, and M. Sugiyama, "Universal approximation property of invertible neural networks", *CoRR*, vol. abs/2204.07415, 2022. DOI: 10.48550/arXiv.2204.07415. arXiv: 2204.07415. [Online]. Available: https://doi.org/10.48550/arXiv.2204.07415.

[13] P. Kidger, "On neural differential equations", Ph.D. dissertation, University of Oxford, 2022.

[14] K. V. F. V. M. van Gijzen, *Numerical Methods for Ordinary Differential Equations*. TU Delft Open, 2023, ISBN: 9789463666657.