

# Augmented Neural Ordinary Differential Neural Network Model using Data-Oriented Multiple Shooting

June Mari Berge Trøan<sup>a</sup>, Vinicius Viena Santana<sup>a</sup>, Brendan Allison<sup>b</sup>, Fredy Vides<sup>c,d</sup> and Idelfonso B.R. Nogueira<sup>a</sup>

<sup>a</sup>Department of Chemical Engineering, Norwegian University of Science and Technology, Trondheim, Norway

<sup>b</sup>Department of Integrative Biology, University of Texas at Austin, Austin, Texas, USA

<sup>c</sup>Department of Statistics and Research, National Commission of Banks and Insurance Companies of Honduras, Tegucigalpa, Honduras

<sup>d</sup>Department of Applied Mathematics, National Autonomous University of Honduras, Tegucigalpa, Honduras

## ARTICLE INFO

### Keywords:

Neural Networks  
Augmented Neural Ordinary Differential Equations  
Continuous time series  
Multiple Shooting  
Growing Window

## ABSTRACT

As we navigate an age defined by data and technological progress, deriving meaningful insight from data has become increasingly important. This pursuit has enhanced the importance of time series forecasting, and several fields have found interest in the advancements of powerful data-driven tools for uncovering patterns and predicting future trends within time series data. Current state-of-the-art methods for forecasting continuous time series use Neural Differential Equations to develop dynamic models from time series data, as they have continuous dynamics and allow for irregular time series. However, these methods often suffer from failing to fit the neural differential equation to oscillatory time series data resulting in a flattened-out trajectory. This paper proposes a novel model, the Augmented Neural Ordinary Differential Equations using Data-Oriented Multiple Shooting, to alleviate the averaged trajectory. To test the performance of the proposed model, it is compared to an Augmented Neural Ordinary Differential Neural Network Model using the Expanding Window method for training, and a Neural Ordinary Differential Neural Network Model using Multiple Shooting. For the benchmarking of these models, all models are trained and tested on synthetically made continuous time series synthesised using Lotka-Volterra dynamics. When comparing the performance of the three models on all chosen metrics, it is clear that the proposed model succeeds in modeling the oscillatory behaviour to a similar precision to the Neural Ordinary Differential Equation Neural Network Model, while the Augmented Neural Network Model using the Growing Window method for training fails to model the time series. Due to uneven testing conditions, the significance of the model's ability to model dynamic, nonlinear systems remains unknown. However, the proposed model does exhibit an example of the effectiveness of employing the multiple shooting method for the training of neural networks. In turn, this paper fosters interest in the further development of models using multiple shooting for the successful modeling of dynamic, nonlinear systems containing noise and trends, such as financial time series.


## 1. Introduction

In an era defined by data and technology advancements, extracting meaningful insight from data has become prevalent. In business, finance, and everyday life there exists a desire to estimate what will happen in the future, as a basis for taking action or making a decision (Brown, 2004). This pursuit has given rise to the interdisciplinary field of time series forecasting, where powerful data-oriented tools for uncovering patterns and predicting future trends within time series data are being developed.

Time series refers to a set of observations made sequentially through time and many real-world phenomena can be recorded as time series (Chatfield, 2000). Examples of time series include sales of a particular product in successive months, the measurement of chemical compound concentration over time, or the performance data chart of a financial index. When predicting time series, the approach can be likened to addressing a regression task. This involves generating output variables that correspond to the same quantity as the input variables but are measured at different

points in time (Perzyk and Rodeziewicz, 2012). In finance, the goal of forecasting the time series is rooted in a desire to predict the future behaviour of financial parameters, such as daily stock prices, inflation or cash flow. Financial time series data are inherently complex and difficult to predict due to investors being influenced by volatile and unpredictable market conditions. Therefore, financial market trends tend to be non-linear, non-stationary and contain high levels of noise (Tsang, Deng and Xie, 2018). While traditional statistical prediction methods, such as linear regression and chaos theory have been popular for many years, they often fail to capture the complex dynamics of these systems (Billah, Waheed and Hanifa, 2016). Therefore, better models are needed to accurately model financial time series by identifying patterns within the data that may be obscured by noise and distortion (Wasserman, 1989).

With the widespread increased interest in machine learning, methods for using machine learning in time series forecasting have become a popular field of interest (Bao, Yue and Rao, 2017). Machine learning can leverage existing data and time series analysis methods while applying machine learning models to address challenges related to the successful description of the complex patterns of dynamic systems.

 jmtroan@stud.ntnu.no (J.M.B. Trøan)  
ORCID(s): 0009-0009-6768-7728 (J.M.B. Trøan)

Several methods are being applied to model the behaviour of time series data, with some of the most used methods being regression-based models. Within the realm of time series analysis, classical regression, also known as linear regression, assumes that a dependent time series is influenced by a set of independent series (Shumway and Stoffer, 2017). For time series regression, this assumption is excessively rigid due to the noisiness of the data; Additionally, the models are often insufficient for explaining all the dynamics of interest that influence the behaviour of time series (Shumway and Stoffer, 2017).

To alleviate the mentioned problems related to rigidity, the autoregressive (AR) model has been proposed as a better fit for time series forecasting. The AR model is based on the idea that the current value of the time series can be explained as a linear combination of past values. This often works well for systems with linear behaviour. However, for nonlinear systems other methods should be preferred over the traditional AR model due to its success on nonlinear systems being limited.

To model nonlinear systems several different strategies are being employed and to classify some of these different strategies we chose to divide them into two categories: discrete models and continuous models. Discrete models use time series data from systems collected at regular intervals, or time series data coming from discrete systems (Romanov, Voronina, Guskov, Moshkina and Yarushkina, 2020). Examples of discrete time series are monthly sales figures or hourly stock prices. The trivial advantage of using discrete-time modeling can be linked to the increased mathematical simplicity of the discrete-time models, often leading to low computation costs. However, the physical interpretation of the estimated model parameters may change considerably between discrete-time differential equations and continuous-time differential equation representations, resulting in discrete-time methods potentially performing inferior to continuous-time methods for dynamic systems containing noise and trends (Yao, Wang and Zhang, 2019). Furthermore, continuous-time models may also be desired, although only discrete-time data are available, for example, due to their flexibility in handling irregular spacing (Elliott, Granger and Timmermann, 2006). Therefore, with the development of technology with the ability to better record time series data at a higher frequency and the drastic increase in computational capacity, there has been an increased focus on developing accurate continuous models.

Financial institutions are undoubtedly interested in the use of machine learning for financial time series prediction due to its broad implementation possibilities and substantial impact (Sezer, Gudelek and Ozbayoglu, 2020). As financial time series data often is recorded at high frequency it can be regarded as continuous data. As the data often contains large amounts of noise and trends, continuous time series models are of interest to researchers seeking to develop robust and accurate models for financial time series forecasting.

Recently, deep learning models have gained interest for application in financial time series forecasting, with results

that significantly outperform traditional machine learning models (Sezer et al., 2020). In 2017 Chong, Han and Park (2017) reported that there were no known methods that applied deep learning to extract information from the stock return time series. Additionally, they recalled no known methods that can extract abstract features from financial time series data or that are capable of identifying hidden nonlinear relationships without relying on econometric assumptions and human expertise, in turn, highlighting deep learning as a particularly attractive alternative to existing models and approaches in financial time series forecasting.

In 2018 Chen, Rubanova, Bettencourt and Duvenaud (2018) proposed a new family of deep neural network models, named neural ordinary differential equations (NODEs) which parameterise the derivative of the hidden states using a neural network. Since many principles in engineering and economics are defined by differential equations, NODEs allow for fitting a model to such systems without specifying a function for the differential equation (Turan and Jäschke, 2021). Additionally, these continuous models allow explicit control of the trade-off between computational speed and accuracy, which proposes a faster and more accurate modeling technique for modeling continuous time series (Chen et al., 2018). Consequently, research on using NODEs has become a popular topic of research.

Despite its suitable structure to model time series, Dupont, Doucet and Teh (2019) brought light to several drawbacks of using NODEs, such as the structure failing to represent several simple classes of functions with a reasonable computational cost. Subsequently, they proposed Augmented Neural ODEs (ANODEs), which are ordinary differential equations specified by a neural network with augmented states. Using ANODEs allows for improved stability and better generalisation than NODEs, and has been shown to successfully model more complex systems than NODEs at a lower computational cost.

In 2019 Rackauckas, Innes, Y., Bettencourt and Dixit (2019) published the documentation of a new Julia library for neural differential equations - *DiffEqFlux.jl*. The library includes among many different libraries, methods for performing training on NODEs using multiple shooting and allows for modeling of ANODEs in Julia (Julia Computing, 2023). However, the multiple shooting method assumes full observability of the underlying dynamics and lack of noise, which for financial time series forecasting is insufficient (Julia Computing, 2019). The library also lacks a method for utilising multiple shooting to train ANODEs. Hence, it appears advantageous to explore a method for modeling dynamic systems in Julia using ANODEs, which employ multiple shooting for the pre-training of a forecasting model.

In 2021 Turan and Jäschke (2021) demonstrated that if the time-series data contains oscillations, the standard fitting of NODEs may result in a flattened-out trajectory that fails to describe the data and proposed a multiple shooting method for NODEs that could handle experimental data containing

noise. Their model performed well on real-life data containing noise; Therefore, models using multiple shooting seem opportune in the context of financial time series forecasting.

Adapting the multiple shooting method for training ANODEs might parallel the strategy outlined in Turan and Jäschke (2021). Nonetheless, the complexity of estimating augmented state values escalates with their increasing number. An innovative approach might derive from existing research on generative latent NODEs, as seen in Chen et al. (2018), where the estimation of latent trajectory initial conditions is conditioned on observations via a recognition neural network.

In light of the promising results from the current literature, this paper proposes an ANODE Neural Network Model using Data-Oriented Multiple Shooting to improve the forecasting of nonlinear time series containing noise, trend, and seasonality, which traditional NODE Neural Networks fail to model. Building upon existing literature Chen et al. (2018), the proposed model features a data-oriented multiple-shooting method for estimating the initial conditions of latent trajectories in ANODEs. In this approach, the initial values for each segment are conditioned on the corresponding observations, enabled by a trainable recognition neural network. These explicitly defined initial parameters differ from arbitrary initial parameters, as these parameters are influenced by the dynamics learned by the recognition neural network. The resulting conditioned initial conditions are hypothesised to result in more accurate modelling of nonlinear systems at a faster convergence (Yu and O'Brien, 1992). Additionally, using better estimators of the initial conditions has been found to both improve the convergence and generalization of neural networks in previous works of Thimm and Fiesler (1995) and Atiya and Ji (1997). Therefore, we promote the data-oriented approach to determine initial conditions as an efficient and precise way of improving the modeling of dynamic, nonlinear time series by ANODEs.

To assess the performance of the proposed model it is compared to two models: an ANODE Neural Network Model using a state-of-the-art training method, namely the Growing Window, and a NODE Neural Network Model using Multiple Shooting, as proposed in the Julia library *DiffEqFlux.jl* (Julia Computing, 2019). All models are trained on synthetic time series data created from Lotka-Volterra dynamics. Results suggest that the ANODE-based Neural Network Model using Multiple Shooting performs predictions of similar accuracy to the NODE Neural Network Model using Multiple Shooting, while the ANODE Neural Network Model using Growing Window fails to fit the dynamic system.

Subsequently, this paper demonstrates the effectiveness of employing the multiple shooting method for the training of nonlinear, dynamic systems; Consequently, establishing interest in the further examination of models using multiple shooting for the successful modeling of dynamic, nonlinear systems containing noise, season and trend, such as financial time series.

## 2. Methodology

### 2.1. Neural Ordinary Differential Equations

In 2018 Chen et al. (2018) proposed a new family of deep neural network models, named neural ordinary differential equations (NODEs) which are specified by a neural network in the following way,

$$\frac{d\mathbf{h}(\mathbf{t})}{dt} = f(\mathbf{h}(\mathbf{t}), \mathbf{t}, \theta) \quad (1)$$

where  $\mathbf{h}$  are the states,  $t$  is time, and  $\theta$  is the neural network parameters (Chen et al., 2018). These models can be trained efficiently with back-propagation and are capable of handling irregularly sampled data, which recurrent neural networks fail to fit (Chen et al., 2018). Subsequently, NODEs can handle a great variety of functions, but at a high computational cost (Dupont et al., 2019). As the training of a NODE progresses and the system becomes increasingly more complex, the number of steps required to solve the ODE increases, which is the major disadvantage of the method (Dupont et al., 2019). In later years other disadvantages have also been brought to light, such as the lack of ability to adapt to incoming data points and explain the underlying dynamics of sparse data (Norcliffe, Bodnar, Day, Moss and Liò, 2021).

### 2.2. Augmented Neural Ordinary Differential Equations

To mitigate the high computational cost, slow learning and lack of representation for some classes of functions in NODEs Dupont et al. (2019) introduced Augmented Neural Ordinary Differential Equations (ANODEs). ANODEs are ordinary differential equations specified by a neural network with augmented states,

$$\frac{d}{dt} \begin{bmatrix} \mathbf{h}(\mathbf{t}) \\ \mathbf{a}(\mathbf{t}) \end{bmatrix} = \mathbf{f} \left( \begin{bmatrix} \mathbf{h}(\mathbf{t}) \\ \mathbf{a}(\mathbf{t}) \end{bmatrix}, \mathbf{t} \right) \quad \begin{bmatrix} \mathbf{h}(\mathbf{0}) \\ \mathbf{a}(\mathbf{0}) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix} \quad (2)$$

where  $\mathbf{h}$  are the states,  $\mathbf{a}$  are the augmented states, and  $\mathbf{t}$  is time. These models are capable of modeling more complex functions with lower losses, have reduced computational cost, and, improved stability and generalisation (Dupont et al., 2019). Even though ANODEs are faster than NODEs, they are not faster than Residual Nets, which can be considered a discretisation of the ODE; They also change the dimension of the input space, which is not desirable for all applications (Dupont et al., 2019).

### 2.3. Single Shooting

Single shooting is a direct optimal control method where continuous control signals are discredited and system dynamics are obtained through simulation (Bayat and J.T., 2023). Deep neural network training can be regarded as optimal control problems where the control variable,  $u$ , corresponds to the parameters of the neural network,  $\theta$  (Vialard, Kwitt, Wei and Niethammer, 2018). Under this

lens, single shooting can be used to solve the training task of the neural network (Evens, Latafat, Themelis, Suykens and Patrinos, 2021). This modelling approach proposes an elegant framework for system identification and control design (Perrusquía and Yu, 2021).

However, single shooting may lead to ill-conditioning, partially due to the optimiser having to simultaneously select parameters to improve the fit at all points along the trajectory; This can result in the network getting stuck while fitting the model to curves in the training data, resulting in a flattened out trajectory (Turan and Jäschke, 2021).

## 2.4. Multiple Shooting

Multiple shooting is an alternative method for direct optimal control, which partitions the dynamic, continuous system into trajectories which are solved as initial value problems using single shooting. To ensure a continuous solution, the method needs the following constraint to be satisfied,

$$x_f^i - x_0^{i+1} = 0 \quad i = 1, 2, \dots, N \quad (3)$$

where  $x_f^i$  is the final data point in trajectory  $i$ ,  $x_0^{i+1}$  is the first element of the trajectory  $i + 1$ , i.e., the last element of each trajectory needs to be equal to the first element of the next trajectory, and  $N$  is the total number of trajectories (Turan and Jäschke, 2021). Subsequently, the problem of flattening the training trajectory is mitigated by utilising the multiple shooting method.

## 2.5. ANODE Neural Network Model using Data-Oriented Multiple Shooting

The proposed Augmented Neural ODE Neural Network Model uses Data-Oriented Multiple Shooting (ANODE-MS). Inspired by the results of Chen et al. (2018) and Turan and Jäschke (2021), this model is built on the philosophy of incorporating conditioning in estimating latent trajectory initial conditions through a recognition neural network. By conditioning the initial conditions on the observations, faster convergence and more stable training are expected. The recognition neural network is coupled with the application of multiple shooting to ensure appropriate training of complex, nonlinear systems. Specifically, the multiple shooting method is employed in the pre-training of the neural networks of the proposed method.

The pre-training is carried out by grouping the time series data into  $m$  trajectories of size  $N$ . For the grouping of the trajectories Equation 4 is used.

$$m = \left\lfloor \frac{d}{N} \right\rfloor \quad (4)$$

where  $m$  is the number of trajectories,  $d$  is the number of data points in the time series and  $N$  is the desired size of the trajectories. Note that depending on the  $N$  and  $d$ , not all data points are guaranteed to be included in the partitioned trajectories.

Following the grouping of the trajectories, each trajectory is passed to the recognition neural network to condition the initial parameters from the observed data to find the initial conditions  $[h_0^j, a_0^j]$  of each trajectory. Here, the first initial condition is the initial value of the observed values in the given time series and the second initial condition accounts for the augmented state.

The conditioned initial conditions are then passed to the ODE solver which solves each trajectory as an initial value problem and calculates a prediction,  $\hat{y}_{t_{i,j}}$ . To ensure a continuous solution, the ODE solution is penalised, such that the shooting gaps,  $s_{j-1}$ , are compliant with the constraint presented in Equation 3. The shooting gap  $s_{j+1}$  is presented in Equation 5 and measures the distance between the last element of the previous predicted trajectory and the first element of the next predicted trajectory.

$$s_{j+1} = \hat{y}_{t_{1,j+1}} - \hat{y}_{t_{N,j}} \quad (5)$$

The objective function  $J$ , presented in Equation 6, then evaluates the prediction and continuity loss.

$$J = \sum_{m=1}^j \sum_{N}^{i=1} (y_{t_{i,j}}^{obs} - \hat{y}_{t_{i,j}})^2 + \sum_{m=1}^j (\hat{y}_{t_{1,j+1}} - \hat{y}_{t_{N,j}})^2 \quad (6)$$

where the first term accounts for the prediction loss and the second term accounts for the continuity loss.

## 2.6. Benchmarking

To test the performance of the proposed model, it is benchmarked on synthetically made time series data with behaviour coming from Lotka-Volterra dynamics and compared with two models: an ANODE Neural Network Model using Growing Window as a method for pre-training the model, and an NODE Neural Network Model using the Multiple Shooting method. Figure 1 attempts to visualise the different models and provides an overview of the differences and similarities of the three models.

### 2.6.1. Lotka-Volterra Dynamics

The Lotka-Volterra is a dynamic model used in ecology to model the dynamics of a two-species ecosystem. The model consists of differential equations describing the population of the prey,  $u_1$ , and the predator,  $u_2$ ,

$$\dot{u}_1 = \alpha \cdot u_1 - \beta \cdot u_1 \cdot u_2 \quad (7)$$

$$\dot{u}_2 = \gamma \cdot u_1 \cdot u_2 - \delta \cdot u_2 \quad (8)$$

where  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $\delta$  are positive parameters (Norcliffe et al., 2021). The Lotka-Volterra exhibits an oscillatory behaviour and the proposed model is tested on time series samples that have been created using these dynamics to examine its ability to model dynamic, nonlinear systems.



### 2.6.2. *ANODE Neural Network Model using Growing Window*

To evaluate the performance of the proposed model an ANODE Neural Network Model which uses the Growing Window (ANODE-GW) for the training of the model is employed. Instead of the model using the grouping function defined in the proposed model, a single shooting function is defined for performing single shooting on a subset, called a window, of the data. This subset is then increased iteratively with a factor of  $N$  for each iteration  $i$  until all time series data is a part of the window.

### 2.6.3. *NODE Neural Network Model using Multiple Shooting*

To assess the performance of an ANODE Neural Network Model compared to a NODE Neural Network Model for modeling dynamic systems a model using a NODE Neural Network Model using Multiple Shooting (NODE-MS) is built. The method is built based on code found in the *DiffEqFlux.jl* documentation in Julia, which uses a pre-built method for Multiple Shooting (Julia Computing, 2021) (Julia Computing, 2023).

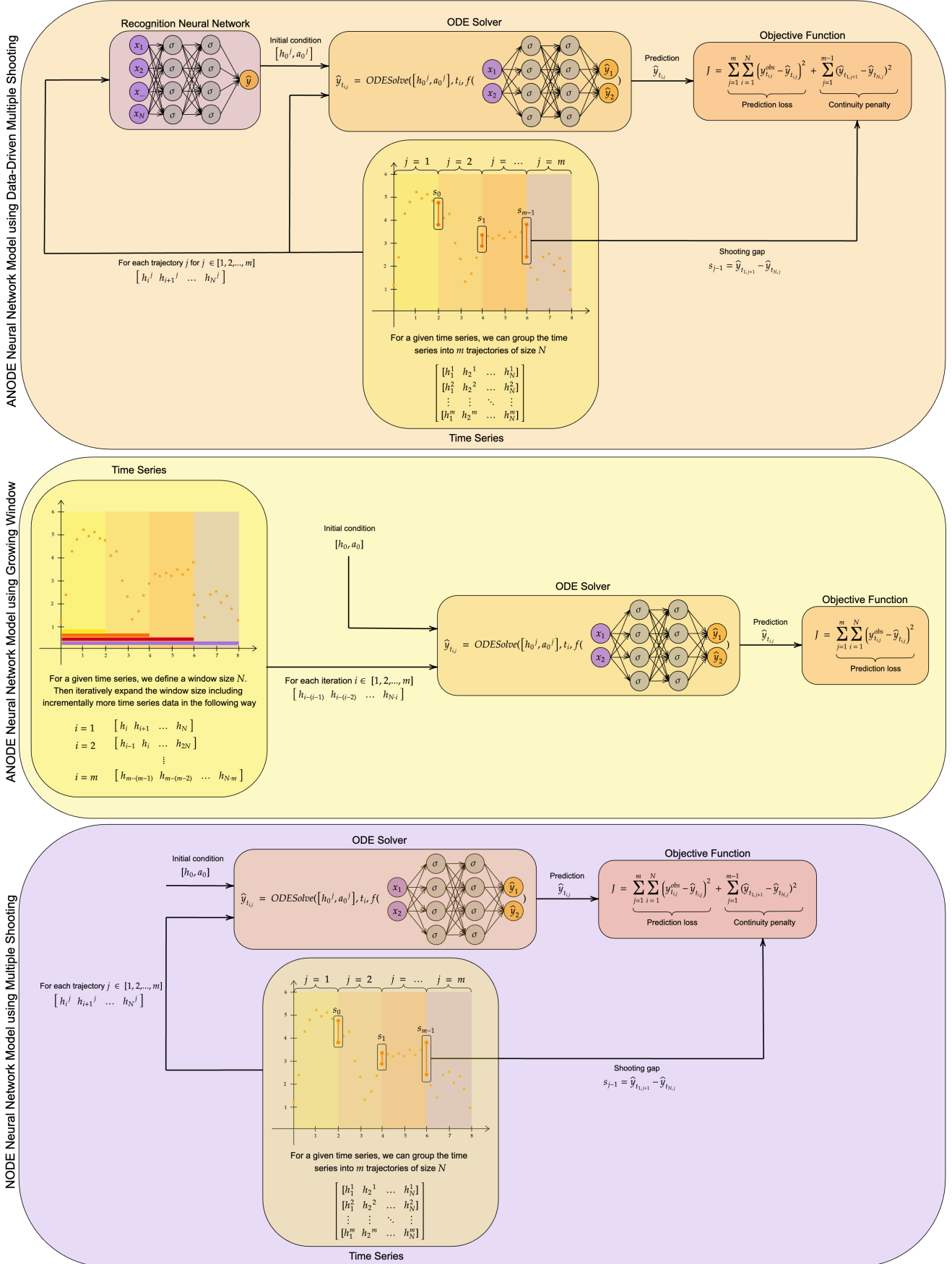
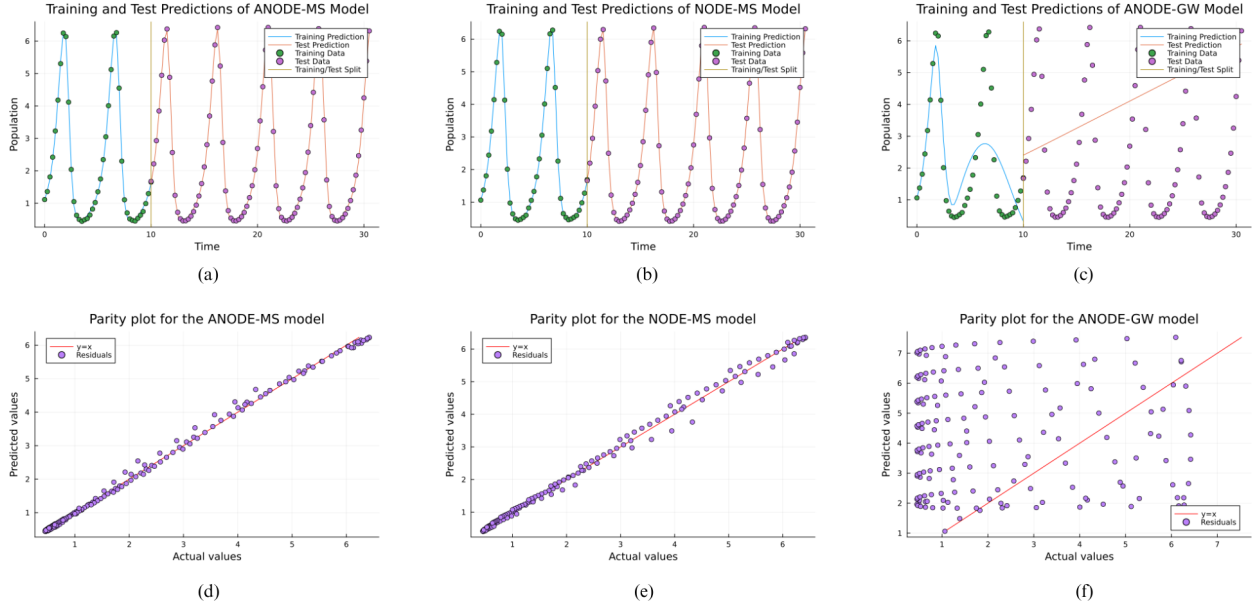


Figure 1: Diagram of the proposed model and benchmarking models.



**Figure 2:** Plot of training and test predictions of the three models and parity plots for each model.

### 3. Results and discussion

For the synthesis of the time series for the benchmarking of the model, the initial guess for the  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $\delta$  are set to be equal to  $[1.3, 0.9, 0.8, 1.8]$ . The time series are made over a time span ranging from  $(0, 10)$  with a time step of 0.25. This results in 82 data points divided on the two systems expressed by the differential equations in Equation 7 - 8. To create the data points the *DifferentialEquations.jl* library in Julia is used, specifically the *NeuralODE* function, which is initialised using the aforementioned parameters and an arbitrary value for  $u_0$ .

For this project the trajectory size  $N$  is set to five, resulting in the synthetic data being partitioned into 10 trajectories. For the recognition neural network, the number of hidden layers is selected to be two, with two input and output nodes respectively. The hidden layers are defined to have 30 nodes. For the neural network in the ODE solver, the neural network has two hidden layers with 30 nodes in each hidden layer. The number of inputs is set to be equal to the size of the trajectories, i.e. the number of inputs was five. As the system is used for predictions on one-dimensional time series, the number of outputs of the neural network is set to be singular. For both the recognition neural network and the neural network utilised in the ODE solver, the activation function is set to be the hyperbolic tangent.

For the pre-training of the proposed model, the selected ODE solver was the *Tsit5* for non-stiff problems and *Rosenbrock23* for stiff problems, formulated in the controller *AutoTsit5(Rosenbrock23)*. The controller is specified with the *autodiff* set to *false*, signifying that the solver is using numerical differentiation instead of automatic differentiation

to compute the Jacobian. For the full training of the system, the selected solver is the *Vern7* for non-stiff problems and *KenCarp4* for stiff problems. For the *KenCarp4* solver, the *autodiff* is set to *true* to enable the use of automatic differentiation to compute the Jacobian. The absolute and relative tolerances were both set to be equal to  $1 \times 10^{-6}$  for both the pre-training and the full training of the system.

For the sensitivity analysis, the selected method for the pre-training of the proposed model was the *ReverseDiffAdjoint*, which uses reverse-mode automatic differentiation. For the full training, *InterpolatingAdjoint* is selected, performing an adjoint sensitivity analysis. Specifically, the *autojvec* was set to equal *ReverseDiffVJP* for the full training, which signifies that the method computes the Jacobian-vector product using reverse-mode automatic differentiation. For more efficient handling of sparse Jacobians, the *ReverseDiffVJP* is used including the *true* argument. i.e. *ReverseDiffVJP(true)* is used.

The pre-training of the proposed model is performed using the *Adaptive Movement Estimation* optimiser (*ADAM*), which computes adaptive learning rates for different parameters, making it suitable for noisy or sparse gradients. After the initial training of the model is performed, the pre-trained model is fitted to the time series data by performing single-shooting optimisation on the entire data set using the *Broyden-Fletcher-Goldfarb-Shanno* (*BFGS*) optimiser algorithm with an initial stepnorm of 0.01. This quasi-Newton method approximates Newton's method for optimisation, finding the minimum of the objective function.

To make the three models comparable the size of the trajectories is set to five for both the models using multiple shooting, ANODE-MS and the NODE-MS. For the method

**Table 1**

Total loss of the three compared Neural Network Models

Neural Network Model	Total Loss
ANODE-MS	0.5143
NODE-MS	1.0320
ANODE-GW	310.82

using the growing window, ANODE-GW, the size of the window is set to be equal to the size of the trajectories, i.e. the size of the window is equal to five. For the ANODE-MS and the NODE-MS models, the solutions are obtained throughout 5000 iterations, while for the method using the growing window, ANODE-GW, the number of iterations is set to 5000 for each window. For the ODE solvers, the methods for performing the sensitivity analysis, and the optimisers, are selected to be as close to those of the proposed models. For the ANODE-GW model, the selected ODE solver is Tsit5 for non-stiff problems and Rosenbrock23 for stiff problems. The chosen sensitivity analysis method is the InterpolatingAdjoint with the autojavec set to ReverseDiffVJP(true). The chosen optimiser is ADAM. For the NODE-MS model, the chosen ODE solvers are the Tsit5 for non-stiff problems and RosenBrock23 for stiff problems, with the autodiff set to *false*. The chosen sensitivity analysis method is the InterpolatingAdjoint with the autojavec defined to be equal to ReverseDiffVJP(true)). Lastly, the NODE-MS is optimised with the ADAM and BFGS algorithms, with an initial stepnorm of 0.01 for the BFGS algorithm.

The proposed model is trained and tested on the synthetic time series data and compared to the performance of the ANODE-GW and the NODE-MS. The training and test results of all model predictions are presented in the first row of Figure 2 with plots (a), (b), and (c). Plots showcase the models' ability to model the oscillatory behaviour of the synthetic time series data. Additionally, the total loss of each model is used for the model evaluation. The calculated total loss of each model is presented in Table 1.

By examining Figure 2, it is possible to see that the proposed model and the NODE from *DiffEqFlux.jl* are shown to successfully model the dynamic behaviour of the synthetic time series data. Both models are shown to succeed in making predictions that fit the real data well, and this is also reflected in the total loss of the models. When comparing the proposed method and the NODE-MS, the proposed model is measured to have a slight advantage when predicting the behaviour of the time series data. Still, as the study has not explored the performance of the models with different hyperparameters the models are assumed to have similar performance on the synthetic time series data employed. Nevertheless, the initial hypothesis is that the proposed model shows potential for using neural network models for time series prediction of dynamic, nonlinear systems with noise and trends, such as financial time series, where the NODE is expected to be inadequate.

However, when analysing the different optimisers used in the two models using multiple shooting, the advantage of the ANODE-MS model might be explained by the BFGS optimiser finding an optimal solution faster for the ANODE-MS model, than for the NODE-MS model for the set parameters of our experiments. This results in a possible stop to the iterations of the optimisation process and by not guaranteeing that the optimisation of the models is performed with the same amount of iterations, the advantage of the ANODE-MS model is not guaranteed. Therefore, to examine the performance of the ANODE-MS model compared to the NODE-MS model, it is necessary to force the BFGS optimiser to run for the same amount of iterations for the optimisation of both models respectively.

Another study of interest to validate the performance of the proposed model against the NODE-MS model is to test the models' performance on different parameters. For this paper, only one set of parameters has been tested. Therefore, if the trends of the results obtained when the two models are run with the same number of iterations are the same as found in this work, it would be interesting to explore further the effect different parameters have on the chosen performance metrics.

In contrast to the ANODE-MS and the NODE-MS models, the ANODE-GW model is shown to fail to learn the behaviour of the data during the training process. This results in the model making predictions which do not fit the oscillatory synthetic data, which is illustrated in the parity plot and the plot of the predictions. Naturally, this results in a high total loss. Hence, this study demonstrates the effectiveness of employing the multiple-shooting approach in the pre-training of neural networks, since both models implement the multiple-shooting method in different ways. Therefore, for further work, it could be interesting to examine further different strategies for implementing multiple shooting for the pre-training of neural networks. By developing novel modeling strategies for employing multiple shooting for pre-training, we hypothesise that there is a potential to discover an efficient model for modeling dynamic, nonlinear systems containing noise, season and trends, such as financial time series.

## 4. Conclusion

In conclusion, this paper proposes an ANODE Neural Network Model using Data-Oriented Multiple Shooting for training, which succeeds in capturing the nonlinear behaviour of time series data. To examine the true potential of the model further testing of the model on time series data containing noise and trends is needed. As the models used for the benchmarking of the proposed models are compared on uneven grounds, it is hard to come to a definite conclusion about the significance of the model's ability to model dynamic, nonlinear solutions. Therefore, further testing ensuring equal conditions is needed to be able to assess the usefulness of the proposed model.

Even though the models' usefulness for the modeling of nonlinear, dynamic systems is still uncertain, this paper



exhibits another example of the effectiveness of employing the multiple shooting method in the training of neural networks. The method for using multiple shooting in the proposed model differs from the method of utilising multiple shooting in the *DiffEqFlux.jl*-library in the definition of the objective function for the ODE problem. Therefore, it could be interesting to explore additional methods for using multiple shooting for the training of neural networks to establish which implementation is most effective and for what problems using multiple shooting holds the most promise in the successful modeling of dynamic, nonlinear systems containing noise and trends, such as financial time series.

## 5. Acknowledgements

### 5.1. Foundation of the proposed model

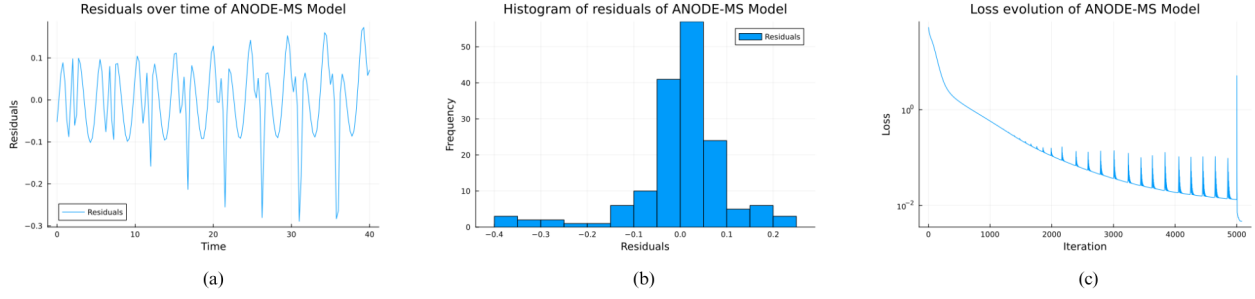
For the development of the proposed model, a draft of the model was received from Brendan Allison from the University of Texas at Austin and used as the foundation of the proposed model. After receiving the code it was debugged, modified and developed further to become the proposed model as presented in this paper.

### 5.2. Inspiration of the ANODE-GW model

The implementation of the growing window method for the training of the neural networks was inspired by public code from Sebastian Callh on using Neural ODEs for weather forecasting (Callh, 2021).

### 5.3. Use of Artificial Intelligence tools

During the development of the proposed model and the preparation of this paper, GitHub Copilot and Chat GPT 3.5 were used to perform simple tasks of autocompleting in the development of the code for the proposed model and to improve language and readability (GitHub, 2023)(OpenAI, 2023). After using these tools, the code for the proposed model and this paper have been reviewed and edited as needed for the work to be considered my own.



**Figure 3:** Plot of residuals over time, histogram and residuals and loss evolution for the proposed model.

## A. Plots

In Figure 3 additional plots used in the evaluation of the goodness of fit of the proposed model are presented; In (a) the residuals over time are presented, in (b) the histogram of residuals is presented, and in (c) the loss evolution for the proposed model is shown.

## B. Libraries

In the proposed method, the following external Julia libraries were used:

- DifferentialEquations.jl (Rackauckas, 2016-2020a)
- SciMLSensitivity.jl (Rackauckas, 2016-2020b)
- Optimization.jl (Julia Computing, 2022c)
- OptimizationOptimisers.jl (Julia Computing, 2022a)
- OptimizationOptimJL.jl (Julia Computing, 2022b)
- ComponentArrays.jl (Diegelman, 2020)
- Lux.jl (Pal, 2022)
- Zygote.jl (Innes, 2018-2019)

## C. Access to the code

The code developed for this project is made available through the link provided in the references (Trøan, J.M.B., 2023).

# References

- Atiya, A., Ji, C., 1997. How Initial Conditions Affect Generalization Performance in Large Networks. *IEEE Transactions on Neural Networks* 8, 448–451.
- Bao, W., Yue, J., Rao, Y., 2017. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLoS ONE* 12.
- Bayat, S., J.T., A., 2023. SS-MPC: A user-friendly software based on single shooting optimization to solve Model Predictive Control problems. *Software Impact* 17.
- Billah, M., Waheed, S., Hanifa, A., 2016. Stock market prediction using an improved training algorithm of neural networks. *2nd International Conference on Electrical, Computer & Telecommunication Engineering*.
- Brown, R.G., 2004. Smoothing, Forecasting and Prediction of Discrete Time Series. Dover Publications, Inc.
- Callh, S., 2021. Neural ODE weather forecasting. <https://github.com/SebastianCallh/neural-ode-weather-forecast.git>.
- Chatfield, C., 2000. Time-Series Forecasting. CRC Press.
- Chen, R., Rubanova, Y., Bettencourt, J., Duvenaud, D., 2018. Neural Ordinary Differential Equations. *Advances in Neural Information Processing Systems* 31.
- Chong, E., Han, C., Park, F., 2017. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications* 83, 187–205.
- Diegelman, J., 2020. ComponentArrays.jl. <https://jonniedie.github.io/ComponentArrays.jl/stable/>.
- Dupont, E., Doucet, A., Teh, Y., 2019. Augmented Neural ODEs. *Advances in Neural Information Processing Systems* 32.
- Elliott, G., Granger, C., Timmermann, A., 2006. Handbook of Economic Forecasting. North-Holland.
- Evens, B., Latafat, P., Themelis, A., Suykens, J., Patrinos, P., 2021. Neural Network Training as an Optimal Control Problem: - An Augmented Lagrangian Approach -. *IEEE Conference on Decision and Control*.
- GitHub, 2023. Copilot. <https://github.com/features/copilot>.
- Innes, M., 2018–2019. Zygote.jl. <https://fluxml.ai/Zygote.jl/stable/>.
- Julia Computing, 2019. DiffEqFlux.jl. <https://github.com/SciML/DiffEqFlux.jl.git>.
- Julia Computing, 2021. Multiple Shooting. [https://docs.sciml.ai/DiffEqFlux/dev/examples/multiple\\_shooting/](https://docs.sciml.ai/DiffEqFlux/dev/examples/multiple_shooting/).
- Julia Computing, 2022a. OptimizationOptimisers.jl. <https://github.com/SciML/Optimization.jl/blob/master/lib/OptimizationOptimisers/src/OptimizationOptimisers.jl>.
- Julia Computing, 2022b. OptimizationOptimJL.jl. <https://github.com/SciML/Optimization.jl/blob/master/lib/OptimizationOptimJL/src/OptimizationOptimJL.jl>.
- Julia Computing, 2022c. SciMLSensitivity.jl. <https://docs.sciml.ai/Optimization/stable/>.
- Julia Computing, 2023. Julia 1.9 Documentation. <https://docs.julialang.org/en/v1/>.
- Norcliffe, A., Bodnar, C., Day, B., Moss, J., Liò, P., 2021. Neural ODE Processes. *ICLR*.
- OpenAI, 2023. Chat GPT 3.5. <https://chat.openai.com/>.
- Pal, A., 2022. Lux.jl. <https://lux.csail.mit.edu/>.
- Perrusquía, A., Yu, W., 2021. Identification and optimal control of nonlinear systems using recurrent neural networks and reinforcement learning: An overview. *Neurocomputing* 438, 145–154.
- Perzyk, M., Rodeziewicz, A., 2012. Application of Time-series Analysis in Control of Chemical Composition of Gray Cast Iron. *Archives of Foundry Engineering* 12, 171–175.
- Rackauckas, C., 2016–2020a. DifferentialEquations.jl. <https://docs.sciml.ai/DiffEqDocs/stable/>.
- Rackauckas, C., 2016–2020b. SciMLSensitivity.jl. <https://docs.sciml.ai/SciMLSensitivity/stable/>.
- Rackauckas, C., Innes, M., Y., M., Bettencourt, J. White, L., Dixit, 2019. DiffEqFlux.jl - A Julia Library for Neural Differential Equations. <https://arxiv.org/abs/1902.02376>.
- Romanov, A., Voronina, V., Guskov, G., Moshkina, I., Yarushkina, N., 2020. Discrete and Fuzzy Models of Time Series in the Tasks of Forecasting and Diagnostics. *Axioms* 9.
- Sezer, O., Gudelek, M., Ozbayoglu, A., 2020. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing Journal* 90.
- Shumway, R., Stoffer, D., 2017. Time Series Analysis and Its Applications. Springer International Publishing.
- Thimm, G., Fiesler, E., 1995. Neural network initialization. From Natural to Artificial Neural Computation (IWANN).
- Trøan, J.M.B., 2023. ANODE-MS. <https://gitfront.io/r/junetroan/GjDL0DuPeEEv/ANODE-MS/>.
- Tsang, G., Deng, J., Xie, X., 2018. Recurrent Neural-Networks for Financial Time-Series Modelling. *Advances in Neural Information Processing Systems* 31.
- Turan, E., Jäschke, J., 2021. Multiple shooting for training neural differential equations on time series. *IEEE Control Systems Letters* 6, 1897–1902.
- Vialard, F.X., Kwitt, R., Wei, S., Niethammer, M., 2018. A shooting formulation of deep learning. *Advances in Neural Information Processing Systems* 33.
- Wasserman, P., 1989. Neural computing: theory and practise. Van Nostrand Reinhold.
- Yao, X., Wang, Z., Zhang, H., 2019. Prediction and identification of discrete-time dynamic nonlinear systems based on adaptive echo state network. *Neural Networks* 113, 11–19.
- Yu, L., O’Brien, J., 1992. On the Initial Condition in Parameter Estimation. *Journal of Physical Oceanography* 22.