# Multiple shooting for training neural differential equations on time series

Evren Mert Turan and Johannes Jäschke

***Abstract*— Neural differential equations have recently emerged as a flexible data-driven/hybrid approach to model time-series data. This work experimentally demonstrates that if the data contains oscillations, then standard fitting of a neural differential equation may result in a "flattened out" trajectory that fails to describe the data. We then introduce the multiple shooting method and present successful demonstrations of this method for the fitting of a neural differential equation to two datasets (synthetic and experimental) that the standard approach fails to fit. Constraints introduced by multiple shooting can be satisfied using a penalty or augmented Lagrangian method.**

***Index Terms*— Multiple shooting, neural differential equations, neural networks, time series.**

## I. Introduction

MECHANISTIC or first principle modelling of systems described by differential equations requires specification of the functional form, following which parameter estimation can be undertaken given data. Neural differential equations (DEs) are a data-driven approach to developing dynamic models from time series data. Neural DEs give continuous dynamics, allow for irregular/incomplete time series, and can be more efficient than neural network approaches through the use modern of ODE solvers [1], [2]. In comparison to mechanistic modelling, neural DEs reduce the need to decide on functional form, while still allowing domain knowledge to be included in the model [1], [2].

Despite the application of neural DEs to complex problems [2], there are still challenges in their use. The standard approach to fitting the neural DE is to iteratively calculate a trajectory by integrating the system to the final time and updating the parameters based on this trajectory. In the dynamic optimization literature, this is known as single shooting as a single trajectory is calculated that depends entirely on the parameters and initial point [3].

Fitting a neural DE via single shooting to a system or time series with oscillatory behaviour or with a long time span can be difficult. The optimization of neural ODE, with randomly initialised weights, may result in a "flattened out" or low frequency trajectory that does not describe higher frequency responses, as shown in Figure 1. Indeed, researchers have demonstrated that neural networks have a spectral bias: low-frequency components of functions are learnt faster during training via gradient descent [4], [5].

The contribution of this work is to propose and numerically demonstrate the use of the multiple shooting method to fit neural differential equations, with constraints satisfied by an Augmented Lagrangian method. In multiple shooting the idea is to form several successive time intervals from the original time span and apply single shooting to each interval. This allows for an initially discontinuous trajectory to be formed early in the optimisation. As the optimization proceeds, the trajectory becomes continuous through the enforcement of shooting constraints. Multiple shooting is widely used for optimisation of ill-conditioned and unstable systems, see [3]. We demonstrate this method on a synthetic and experimental data set, that the neural DE otherwise fails to fit.

## II. Background

### A. Neural differential equations

Neural ODEs were introduced by [1] to be a differential equation specified by a neural network, i.e.:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = NN(x, u, t, \theta) \tag{1}$$

where $x$ are the states, $\theta$ are the neural network parameters, $u$ is an exogenous input, and $t$ is time. As the neural ODE is restricted by construction to be the solution of a differential equation, it is not a universal approximator [6]. Nevertheless, a wide range of systems in science and engineering are described by differential equations and neural ODEs allow one to fit a model to these systems, without specifying a function form for the differential equation.

Later authors demonstrated the use of neural networks in other types of differential equations, with the potential incorporation of a known functional form (neural differential equations) [2], e.g. a first order differential equation of the form:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x, u, NN(x, u, t, \theta), t) \tag{2}$$

This formulation allows first principle knowledge, such as conservation laws or relationships between quantities to be specified, while using the neural network to model

This work was supported by the Norwegian Research Council through the AutoPRO project.

E.M. Turan and J. Jäschke are with the Department of Chemical Engineering, Norwegian University of Science and Technology (NTNU), Trondheim, Norway. evren.m.turan@ntnu.no, johannes.jaschke@ntnu.no

unknown relationships [2]. Note that equation 1 is a special form of equation 2.

Regardless of the formulation, the problem of training the parameters of a neural DE is the same as estimating the parameters of a differential equation. If one had access to the states and time derivatives ($x$, $\frac{\mathrm{d}x}{\mathrm{d}t}$) then the parameter estimation would be a fitting problem, i.e. one would not have to integrate the system. However, typically only noisy measurements of some of the states are available. There are two main approaches to estimate the parameters of a differential equation. The first is to use a two stage method where first a flexible smooth function (typically spline bases) is fit to the data to provide estimates ($\hat{x}$, $\frac{\mathrm{d}\hat{x}}{\mathrm{d}t}$) which are then used to estimate the model parameters [7]. This technique requires that all states are measured, and that the estimate $\frac{\mathrm{d}\hat{x}}{\mathrm{d}t}$ is accurate. This later requirement becomes increasingly difficult to satisfy with increasing noise and sparsity of sampling. The alternative approach is to integrate the ODE and define the cost function $C$ using the measured and predicted states, typically the sum of squared errors (SSE) is used. This is the approach most often used for neural DEs and was taken from the optimal control literature [1].

### B. Single shooting with neural differential equations

Despite the potential of neural DEs, a significant issue is the existence of local minima during the training procedure. For example, consider the example of fitting a neural ODE to the spiral differential equation [1], [2], [8]:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = Ax^3 \tag{3}$$

$$= \begin{pmatrix} -0.1 & 2.0 \\ -2.0 & -0.1 \end{pmatrix} x^3 \tag{4}$$

The system is solved with $x_0 = [2., 0.]$, and $t \epsilon [0., 6.0]$ using a Runge–Kutta method [9]. Synthetic data points are recorded at 0.1 intervals and normally distributed noise ($\mathcal{N}(0.0, 0.2)$) is introduced.

We consider the task of fitting a neural ODE with a neural network with $x^3$ as input, i.e. $\frac{\mathrm{d}x}{\mathrm{d}t} = f_{NN}(x^3, \theta)$, using the sum of squared error as the cost function. This means that the neural network has the task of approximating $A$ in equation 3. The neural network has one hidden layer of 16 nodes, tanh is used as the activation function in the input and hidden layer, and initial weights are set via Glorot initialization [10].

An issue with single shooting is that the optimiser must simultaneously select parameters to improve the fit at all points along the trajectory. This can result in the network getting stuck during training by fitting a flattened curve through the middle of the data as shown in Figure 1. Fitting is performed by a Nesterov momentum version of the Adam algorithm (Nadam) [11], [12], with an initial learning rate of 0.001.

When changing the learning rate, different behaviour can be observed during the training. For example, the initial oscillation of the system ($0 \leq t \leq 1.5$) can be fit followed by straight or warped lines "within" the oscillations of the data, similar to Figure 1. It should be noted that the curve can be fit by careful adjustments of the learning rate (scheduling) during the optimization. However, in general this will require manual interaction.

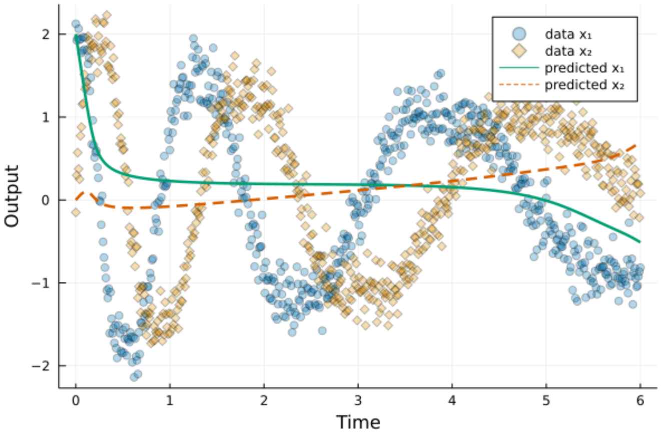

Fig. 1. Plot of neural ODE fitted to data from equation 3 via single shooting with Nadam [12].

### C. Multiple shooting

Multiple shooting is an alternative method of fitting, wherein the time span $[t_0, t_f]$ is partitioned into $N_s$ intervals by forming a grid of $N_s + 1$ points, $t_0 = \tau_0 < \tau_1 < ... < \tau_{N_s} = t_f$ [13]. The values of the state $x$ at the grid points are introduced as additional variables (shooting variables) e.g. in Figure 2, interval 2 is formed by $\tau_1 = 2.0$ to $\tau_2 = 4.0$ and the initial and final values in this interval are denoted as $x_0^{(2)}$ and $x_f^{(2)}$.

On each interval an initial value problem can be solved, giving a potentially discontinuous trajectory $\hat{x}$ as shown in Figure 2. This trajectory is used to calculate the cost (and gradient) as in single shooting. The trajectory becomes meaningful, when the gap between intervals (the shooting gap, see Figure 2) introduced by the new state variables is zero, i.e. at the end of the training procedure, the following constraints need to be satisfied:

$$x_f^{(i)} - x_0^{(i+1)} = 0 \qquad i = 1, 2, \ldots, N_s \tag{5}$$

The use of multiple shooting offers two advantages for training neural DEs: (1) the time series data can be used to provide an initial guess for the unknown states at the shooting points - thus, the influence of poor initial parametrisation is reduced [13], (2) the $N_s$ initial value problems are independent and hence their solving is parallelisable. Point (1) can aid in the initial fitting of a neural DE - while the network weights are small (i.e. the neural network is close to linear) the optimiser can improve the fit by adjusting the shooting variables, thereby giving a discontinuous trajectory that describes the data. The disadvantage of multiple shooting is that the optimisation problem has $N_s$ constraints that need to be satisfied, e.g. by the methods outlined in the following section.

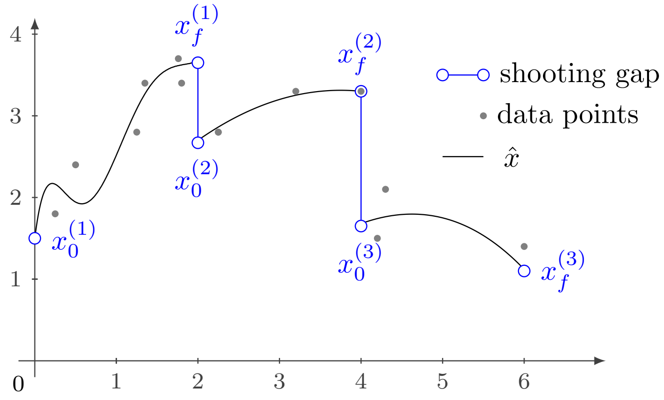

Fig. 2. Schematic of multiple shooting.

### D. Penalty and Augmented Lagrangian methods

Consider the constrained optimization problem:

$$\min C(z) \tag{6}$$

$$\text{s.t. } h(z) = 0 \tag{7}$$

where $C$ is the cost function and $h$ is a vector function of equality constraints, in our case the shooting constraints (Equation 5), and $z$ are the optimisation variables. Supervised learning of neural networks is typically performed by optimising an unconstrained problem, that is related to the constrained optimisation problem. A common approach is to define a proxy cost function, $\phi$, which has the constraints as penalties terms:

$$\phi = C(z) + \rho Q(h(z)) \tag{8}$$

where $\rho$ is a hyper-parameter, and $Q$ is a penalty function. The most common choices of $Q$ are a quadratic penalty function ($Q(h(z)) = h(z))^2$), or the $l_1$, $l_2$ (not squared) and $l_\infty$ norms

The later three norms give an exact penalty function, which means that under standard assumptions [14] a single minimization with some $\rho^*$ can yield the same solution as the constrained problem. Note that a too large $\rho$ can result in numerical issues, while a too small $\rho$ may result in constraint violation [14].

An alternative approach is to use the method of multipliers or the augmented Lagrangian method which defines the objective function as:

$$\phi = C(z) + \sum h_i(z)^T v_i + \rho \sum h(z)_i^T h(z)_i \tag{9}$$

where $v$ is an approximation of the Lagrange multipliers, that is updated, along with $\rho$, as part of the optimization algorithm. Algorithm 1 outlines a potential Augmented Lagrangian algorithm.

**Algorithm 1** Augmented Lagrangian

```
1:  Initialize the constrained optimisation problem
2:  Set: $v \leftarrow 0$, $\rho \leftarrow 0$, $k \leftarrow 1$
3:  repeat
4:      $\theta^k, x^k = \arg\min \phi$                ▷ Unconstrained
5:      if $h(x^k, \theta^k) = 0$ then
6:          Converged ← True
7:      else
8:          Update $v$ and $\rho$          ▷ Algorithm dependent
9:          $k \leftarrow k + 1$
10:     end if
11: until Converged
```

Augmented Lagrange algorithms can use any unconstrained optimiser to solve the unconstrained optimization problem (line 4). Globally convergent augmented Lagrange algorithms have been implemented [15], [16].

*Remark 1:* At the time of writing, there is unpublished related work, similar in spirit, in an example in a neural differential equation package [17]. In the example a penalty approach is used, and the shooting intervals are restricted to start and end on selected data points. Thus, the approach is unsuitable for real, noisy systems because the data points are noise contaminated and there is no reason why the fitted solution should go through these data points.

## III. Multiple shooting with Neural DEs

In the following sections we demonstrate the approach on two problems. This work is coded in Julia [18], using the following packages: DifferentialEquations.jl [19], [20], DiffEqFlux [2], [17], Flux.jl [21], ForwardDiff.jl [22], NLopt.jl [23] and Hyperopt.jl [24].

### A. Spiral differential equation - synthetic example

The spiral differential equation introduced in the preceding section (Eq. 3) is used as a synthetic example to demonstrate the proposed procedure. The sum of squared errors (SSE) is again used for the cost function, however to allow for good out-of-sample prediction we introduce a regularisation term, $R(\theta)$, that penalises the complexity of the neural DE, i.e. $C(\theta) = \text{SSE}(\theta) + \rho_R R(\theta)$, where $\rho_R$ is a regularisation constant.

We use the sum of the spectral norm of the weights in each layer as the regularisation term [25], with a regularisation constant of 1.0. Furthermore, we remove the bias nodes from the network as we wish to map zeros-to-zeros, as an application of prior knowledge. The additional variables introduced by multiple shooting are initially set to $x(0)$.

Training is performed with 20 intervals using an Augmented Lagrangian method [15], [16], with LBFGS [26] used as the inner optimiser. Figure 3 shows that in comparison to single shooting (Figure 1), multiple shooting is able to give a neural ODE that fits the data. Moreover, the trained neural ODE shows good

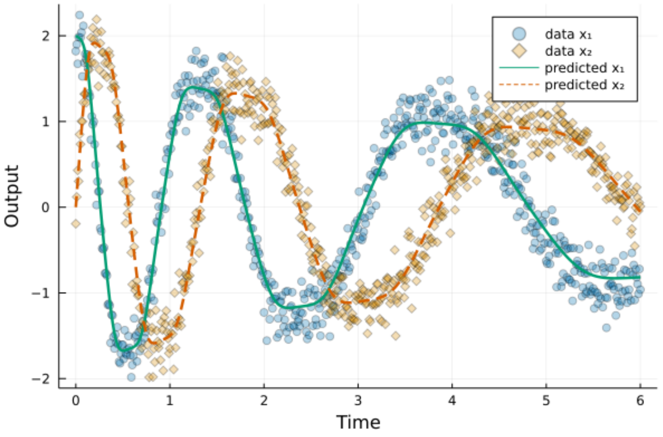

Fig. 3. Plot of neural ODE fitted to data from equation 3 via multiple shooting with 20 intervals. Figure was made using the fitted parameters in a single IVP, i.e. without intervals.

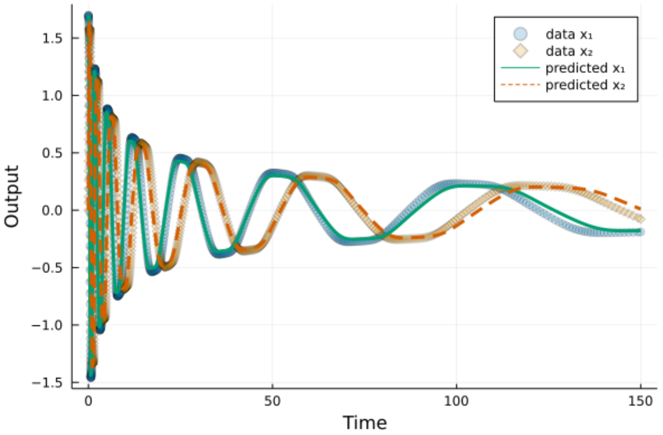

Fig. 4. Plot of neural ODE fitted to data from equation 3 on $t\epsilon[\mathbf{0.}, \mathbf{6.0}]$, on a much larger span. Neural ODE fitted via multiple shooting with 20 intervals.

generalisation to a much longer time scale $t\epsilon[0., 250.0]$, as shown in Figure 4, despite only being trained on data up to $t = 6.0$. This is partially because the removal of the bias nodes forces the neural DE to have a time derivative of 0.0 when both states are 0.0.

### B. Cascading tanks - real data

*1) System description:* The cascading tank system considered here is part of a non-linear identification benchmark problem, fully described in [27]. The system consists of two tanks, arranged as per Figure 5. Water is fed to tank one (the pump voltage is the exogenous input signal, $u$), and then flows into tank two before leaving the system. Water can also overflow over the edge of the tanks, and a portion of the overflow from tank one may enter tank two. Only the water level of the second tank (output signal, $y$) is recorded.

The benchmark problem consists of a training and testing dataset. The input, $u(t)$, is a multisine signal consisting of 1024 measurements (at 4 seconds intervals), and the output signal, $y(t)$, is the measurement from the water level sensor at these same time points, giving the data shown in Figure 6. The second half of training set is used for validation. The initial height of the tanks are unknown, but are the same for both data sets, i.e. $y(0)$ is estimated from the data. Our goal is to fit a neural ODE to this dataset. The SSE is used as the cost function. As the input signal ($u$) is discrete, we use a constant piecewise interpolation of the data for evaluation in continuous time. In comparison, using a cubic spline has influence on the results. Inputs before the data period are assumed to be constant, i.e. $u(t) = u(0), \ \ \forall t < 0.0$.

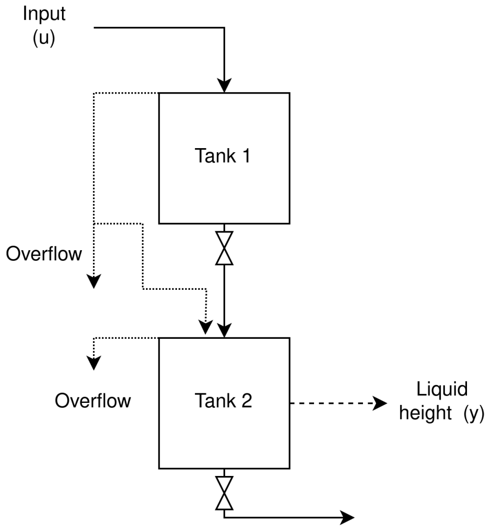

Fig. 5. Diagram of cascading tank system

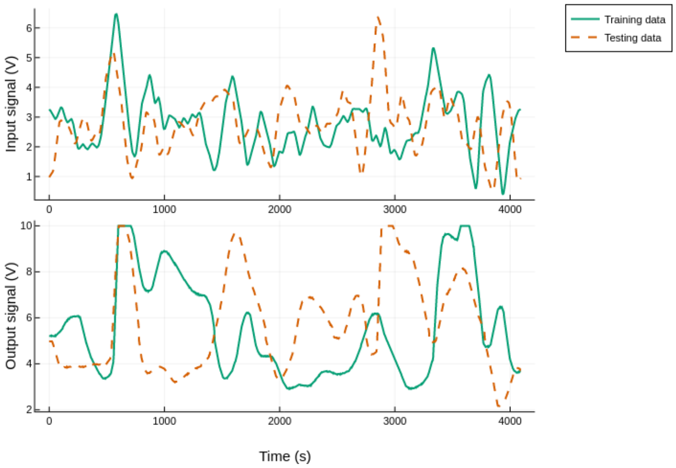

Fig. 6. Input and output signals for the cascading tanks system [27]

*2) Neural network:* A neural network with no bias units, and one hidden layer with 64 units, tanh as the activation function in the input and hidden layer, and initial weights set via Glorot initialization [10]. For regularisation, an $l_2$ penalty is applied on the network weights, with the penalty constant, $\rho_{l_2}$, as a hyper parameter.

Using only the input, $u(t)$, and output, $y(t)$, signals as features for the neural network gives a poor fit to the data. As such we provide three additional inputs:

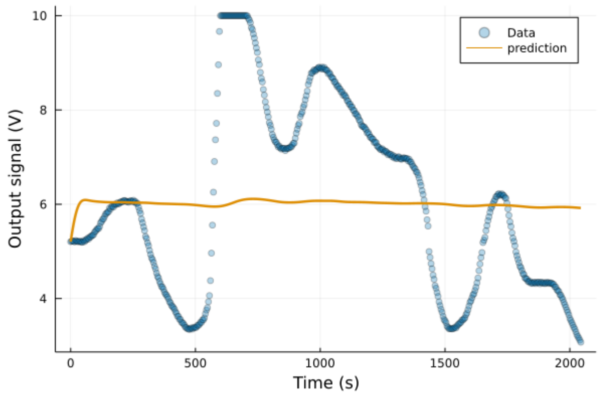

Fig. 7. Neural ODE for cascading tanks system fitted via single shooting, with Adam [11] (2000 iterations, with learning rate set to 0.001).

- $\sqrt{y(t)}$ - the flow out of a tank is proportional to the square root of the fluid height (Bernoulli's equation)
- $u(t - \tau_d)$ - the first tank acts as a time delay
- $\int_{t-\tau_i}^{t} u(t^*)dt^*$ - the output signal shows less rapid variation than the input as the first tank "smooths" the data (see Figure 6)

Thus, we are fitting the neural DE:

$$\begin{aligned}\frac{\mathrm{d}y}{\mathrm{d}t} =& NN\Big(u(t), y(t), \sqrt{y(t)}, u(t-\tau_d), \\ & \int_{t-\tau_i}^{t} u(t^*)dt^*\Big) \qquad NN : \mathbb{R}^5 \to \mathbb{R}\end{aligned} \tag{10}$$

*3) Fitting:* Fitting the neural DE via single shooting proceeds poorly as shown in Figure 7. In comparison, with multiple shooting the neural ODE is able to be fit to the data (Figure 8). The average square root error on the training, validation and test set is 0.42, 0.50, 0.62 respectively. Figure 9 shows that the neural DE is able to generalise well, although it has issues with the large peaks in the first 2000 seconds. The hyper parameters values are chosen via Bayesian optimization as $5.96 \times 10^{-2}$, 79.0s, and 164.0s, for $\rho_{l_2}$, $\tau_d$, and $\tau_i$ respectively. See [28] for an introduction to Bayesian optimization.

## IV. Discussion

The examples shown in the paper use the augmented Lagrangian method, however we found that using a penalty method is feasible. The penalty parameter strongly influences the fit of the neural ODE, with potential approaches to adjust this parameter discussed in [14] and [3].

Authors have recently showed that the interaction between neural DE and DE solver can lead to discrete dynamics, resulting in the neural DE depending on the numerical methods used in the fitting [29]. A specialised time stepping algorithm was recommended [29], however the use of "normal" adaptive time stepping, as implemented in DifferentialEquations.jl [19] showed no issues.

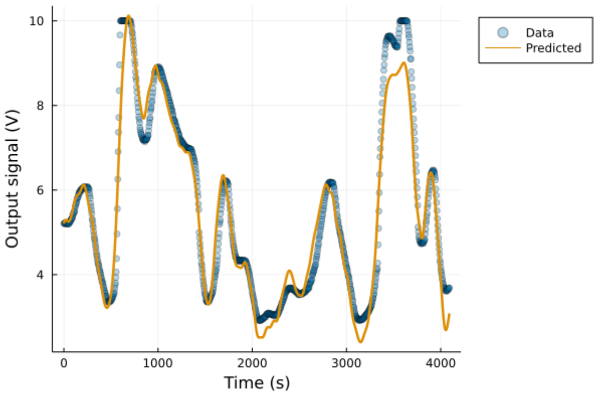

Fig. 8. Neural ODE for cascading tanks system fitted via multiple shooting, with an Augmented Lagrange method [16]. The first half of the data (0-2048 seconds) was used for training, and the later half for validation (2048-4096 seconds).

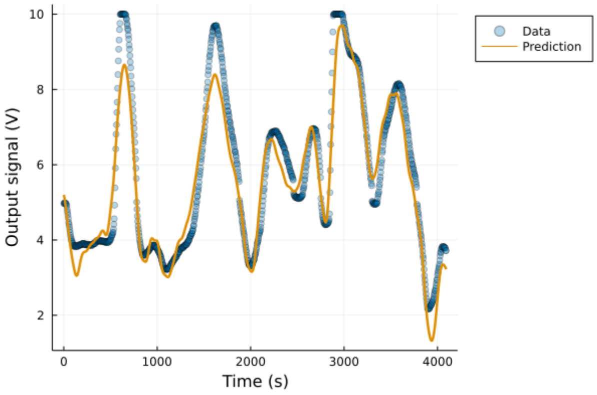

Fig. 9. Neural ODE for cascading tanks system fitted via multiple shooting, with an Augmented Lagrange method [16].

## V. Conclusion

Fitting a neural DE to a time series with oscillatory behaviour can be a challenging task. Multiple shooting can alleviate this difficulty, by providing the optimiser the flexibility to find an initially discontinuous trajectory that is close to the observed data [13]. This was demonstrated through fitting a synthetic and experimental dataset. An augmented Lagrangian method was used to fit the neural ODE due to the shooting interval constraints. In practice the penalty method can work well, if the penalty parameter is carefully chosen. Future work could investigate the effect on computational time due to the introduction of constraints and the possible parallelization, or the influence of the length of the shooting intervals.

## References

[1] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural Ordinary Differential Equations," *Advances in neural information processing systems*, 2018.

[2] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman, "Universal Differential Equations for Scientific Machine Learning," *arXiv preprint arXiv:2001.04385*, 2020.

[3] L. T. Biegler, *Nonlinear programming: concepts, algorithms, and applications to chemical processes.* Society for Industrial and Applied Mathematics, 2010.

[4] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the Spectral Bias of Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 5301–5310, PMLR, 2019.

[5] Z.-Q. J. Xu, Y. Zhang, and Y. Xiao, "Training behavior of deep neural network in frequency domain," in *International Conference on Neural Information Processing*, pp. 264–274, Springer, 2019.

[6] E. Dupont, A. Doucet, and Y. W. Teh, "Augmented Neural ODEs," *arXiv preprint arXiv:1904.01681*, 2019.

[7] J. M. Varah, "A Spline Least Squares Method for Numerical Parameter Estimation in Differential Equations," *SIAM Journal on Scientific and Statistical Computing*, vol. 3, no. 1, pp. 28–46, 1982.

[8] D. Onken and L. Ruthotto, "Discretize Optimize vs Optimize Discretize for Time Series Regression and Continuous Normalizing Flows," *arXiv preprint arXiv:2005.13420*, 2020.

[9] C. Tsitouras, "Runge–Kutta pairs of order 5 (4) satisfying only the first column simplifying assumption," *Computers & Mathematics with Applications*, vol. 62, no. 2, pp. 770–775, 2011.

[10] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[11] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[12] T. Dozat, "Incorporating nesterov momentum into ADAM," in *International Conference on Learning Representations*, 2016.

[13] H. G. Bock, "Numerical treatment of inverse problems in chemical reaction kinetics," in *Modelling of chemical reaction systems*, pp. 102–125, Springer, 1981.

[14] J. Nocedal and S. J. Wright, *Numerical Optimization.* Springer Science & Business Media, 2 ed., 2006.

[15] A. R. Conn, N. I. M. Gould, and P. Toint, "A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds," *SIAM Journal on Numerical Analysis*, vol. 28, no. 2, pp. 545–572, 1991.

[16] E. Birgin and J. Martínez, "Improving ultimate convergence of an augmented Lagrangian method," *Optimization Methods and Software*, vol. 23, no. 2, pp. 177–195, 2008.

[17] 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.

[18] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[19] 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.

[20] Y. Ma, V. Dixit, M. J. Innes, X. Guo, and C. Rackauckas, "A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–9, IEEE, 2021.

[21] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah, "Fashionable Modelling with Flux," *CoRR*, vol. abs/1811.0, 2018.

[22] J. Revels, M. Lubin, and T. Papamarkou, "Forward-Mode Automatic Differentiation in Julia," *arXiv preprint arXiv:1607.07892*, 2016.

[23] S. G. Johnson, "The NLopt nonlinear-optimization package," *http://github.com/stevengj/nlopt*, 2014.

[24] F. Bagge Carlson, "Hyperopt. jl: Hyperparameter optimization in Julia," *https://github.com/baggepinnen/Hyperopt.jl*, 2018.

[25] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree, "Regularisation of neural networks by enforcing Lipschitz continuity," *Machine Learning*, vol. 110, no. 2, pp. 393–416, 2021.

[26] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[27] M. Schoukens and J. Noël, "Three Benchmarks Addressing Open Challenges in Nonlinear System Identification," *IFAC-PapersOnLine*, vol. 50, pp. 446–451, jul 2017.

[28] P. I. Frazier, "A Tutorial on Bayesian Optimization," *arXiv preprint arXiv:1807.02811*, jul 2018.

[29] K. Ott, P. Katiyar, P. Hennig, and M. Tiemann, "When are Neural ODE Solutions Proper ODEs?," *arXiv preprint arXiv:2007.15386*, 2020.