Elisabeth Roesch, Christopher Rackauckas and Michael P. H. Stumpf*

# Collocation based training of neural ordinary differential equations

https://doi.org/10.1515/sagmb-2020-0025
Received April 6, 2020; accepted May 4, 2021; published online July 9, 2021

**Abstract:** The predictive power of machine learning models often exceeds that of mechanistic modeling approaches. However, the interpretability of purely data-driven models, without any mechanistic basis is often complicated, and predictive power by itself can be a poor metric by which we might want to judge different methods. In this work, we focus on the relatively new modeling techniques of neural ordinary differential equations. We discuss how they relate to machine learning and mechanistic models, with the potential to narrow the gulf between these two frameworks: they constitute a class of hybrid model that integrates ideas from data-driven and dynamical systems approaches. Training neural ODEs as representations of dynamical systems data has its own specific demands, and we here propose a collocation scheme as a fast and efficient training strategy. This alleviates the need for costly ODE solvers. We illustrate the advantages that collocation approaches offer, as well as their robustness to qualitative features of a dynamical system, and the quantity and quality of observational data. We focus on systems that exemplify some of the hallmarks of complex dynamical systems encountered in systems biology, and we map out how these methods can be used in the analysis of mathematical models of cellular and physiological processes.

**Keywords:** collocation; dynamical systems; neural ODE; systems biology.

## 1 Introduction

In physics, simple fundamental principles often suffice to derive equations depicting the behaviour of natural systems. Through Noether's theorem, for example, conservation of energy, momentum, etc. are linked to continuous symmetries, from which models can be formulated and tested. There is an implicit trade-off between the explanatory and the predictive power of modelling approaches (Baker et al. 2018). When there are sufficient data about the underlying system, purely data-driven modelling, for example, with deep learning models, is typically superior in predicting unseen behaviour of the system than mechanistic modelling approaches, such as ordinary differential equations (ODE),

$$\frac{\mathrm{d}Y(t)}{\mathrm{d}t} = f(Y) \tag{1}$$

or stochastic differential equations (SDE)

$$\mathrm{d}Y(t) = f(Y)\mathrm{d}t + g(Y)\mathrm{d}W_t. \tag{2}$$

***Corresponding author: Michael P. H. Stumpf**, School of BioSciences, Biosciences 4, The University of Melbourne, Royal Parade, Parkville, VIC 3052, Australia; and School of Mathematics and Statistics, University of Melbourne, 813 Swanston Street, Parkville, VIC 3010, Australia, E-mail: mstumpf@unimelb.edu.au
**Elisabeth Roesch**, Melbourne Integrative Genomics, University of Melbourne, 30 Royal Parade, Parkville, VIC 3052, Australia; and School of Mathematics and Statistics, University of Melbourne, 813 Swanston Street, Parkville, VIC 3010, Australia
**Christopher Rackauckas**, Department of Mathematics, Massachusetts Institute of Technology, 182 Memorial Dr, Cambridge, MA 02142, USA; Julia Computing, 240 Elm Street, 2nd Floor, Somerville, Massachusetts 02144, USA; and Pumas-AI, 14711 Kamputa Drive, Centerville, VA 20120, USA

Here $Y$ denotes the states described in the system, $f(Y)$, the deterministic dynamics, $g(Y)$ the stochastic dynamics, and $\mathrm{d}W_t$ is a Wiener Process increment (Gardiner 2009). Mechanistic models require us to spell out all assumptions (types of interactions, reaction rate parameters, etc.) before we can proceed with analysis. Especially if backed up by formal model selection and model checking (Kirk et al. 2013; Liepe et al. 2013, 2014), and combined with carefully designed, discriminatory experiments, mechanistic models are able to deliver insights into how natural or engineered systems work, and how we can control and guide their behaviour (Gupta and Khammash 2014; Lakatos and Stumpf 2017; Milias-Argeitis et al. 2011). This tension between the modelling schools, is sometimes overlooked by practitioners who are often invested in one of these fields. But for many domain experts, people who are more interested in the scientific problem at hand, the choice of modelling ansatz can and should matter a great deal, and therefore ought to be chosen with care.

Hybrid models, which here, following precedents in the literature (Baker et al. 2018), we define as modelling approaches that share and combine aspects of mechanistic modelling as well as data-driven modelling in order to arrive at better understanding of complicated systems, hold some promise. For example, known mechanistic relationships could be modelled using ODEs or SDEs, and less well characterised aspects of the system could be captured purely by data-driven approaches (Rackauckas et al. 2020). Defining and refining the interface where the two modelling domains meet still requires a lot of work. For mechanistic models, however, the advantages really come to the fore if the models are realistically parameterized. This does not necessarily require us to know all parameters with high accuracy, but we should have at least an assessment of confidence in each parameter (Kirk et al. 2015) (from which, for example, robustness of solutions can be assessed). Parameterizing models of even moderate size is problematic and a well known challenge in the systems biology, statistical inference for dynamical systems and inverse problems literature (Kersting et al. 2020).

This motivates an alternative perspective on modelling, which can also loosely be viewed as bridging the gap between mechanistic and data-driven modelling: focusing primarily on the qualitative features of dynamical systems (Jost 2005). For example, many systems may be qualitatively more constrained than quantitatively: bifurcations in dynamical are the canonical example of such qualitative features that profoundly shape system dynamics (Roesch and Stumpf 2019). An attractive alternative to making an explicit model might be to determine all qualitative features of a dynamical system (from data or observations) and then develop a data-driven model. One of the many uses of neural ODEs is precisely this. Neural ODEs have been introduced by Chen et al. (2019) in 2018. Since then the approach as been extended in various directions: Dupont et al. (2019) extended the approach to augmented neural ODEs; the use of neural ODEs for latent systems is discussed in more detail by Rubanova et al. (2019); the technique of neural ODEs has also been extended to other forms of differential equations, including SDEs independently by Liu et al. (2019) and Tzen and Raginsky (2019); neural jump SDEs have been formulated by Jia and Benson (2019); and neural PDEs have been introduced by Rackauckas et al. (2019). In this work, we focus on neural ODEs as introduced in the original publication by Chen et al. (2019) and discuss alternative ways of fitting neural ODEs to observed dynamics.

As the notion of a neural ODE is relatively new, their methodology will be outlined in the next section. After that we illustrate the use and some of the factors determining the computational efficiency of neural ODEs. The dearth of mathematical models for most biological systems is an issue for predictive biology, and, perhaps even more so, for synthetic biology (Leon et al. 2016; Scholes et al. 2019). We believe that neural ODEs have the potential to form a stepping stone between purely data-driven modelling and mechanistic modelling. The contribution that this paper makes is to map out some of the practicalities that are encountered when training neural ODE models on data from dynamical systems; with these results in hand there is clearly reason to be optimistic about the scope of applying neural ODEs in systems and synthetic biology.

# 2 Methods

## 2.1 Neural ODEs

A neural ODE (Chen et al. 2019) is an ODE in which the derivative is specified by a neural network (Murphy 2012). Choosing the right network architecture is therefore a crucial challenge in the conceptual design of neural ODEs. Deep neural networks (Innes 2018; Innes et al. 2018; LeCun et al. 2015) with multiple layers have become the most popular approach in this area. Each layer of the neural network can be interpreted as a regression model and with this in mind we can start to reason about the way deep neural networks work.

We start by considering a single layer. Let $X$ be a real $m \times n$-matrix representing the input of a regression model, $\mathcal{M}$, and let $Y$ be a real $m \times o$-matrix representing the output of this model. For sake of simplicity, we consider input and output to be one-dimensional vectors of the same length, i.e. $m = 1$ and $n = o$. In the following, the observed input is referred to as $X$ and the observed output is referred to as $Y$, respectively. Together, the pair of $X$ and $Y$ relates to the training data. We learn the relationship of $X$ and $Y$ with the linear regression model $\mathcal{M}$, which predicts $Y^*$ for a given $X$,

$$\mathcal{M}_{W,B}(X) = Y^* = X \times W + B \tag{3}$$

where $W$ is an unknown weight $n \times o$-matrix, and $B$ is an unknown bias $m \times o$-matrix. In the training phase, we optimise the model's parameters $W$ and $B$ by minimising the distance $d$ between observation and prediction,

$$\hat{W}, \hat{B} = \min\{d(Y, Y^*)\} = \min\{d(Y, X \times W + B)\}, \tag{4}$$

where $Y$ denotes the observed output of the training data, and $Y^*$ the model's prediction for the training input $X$. For any new instance of $X$, which we denote by $\tilde{X}$, the prediction of the model $\mathcal{M}$ is then given by

$$\mathcal{M}_{\hat{W},\hat{B}}(\tilde{X}) = \tilde{X} \times W + B. \tag{5}$$

Here we introduced the example model $\mathcal{M}$ as a linear model. However, most deep learning models – including the ones in this work – are intended to capture non-linear relationships (LeCun et al. 2015). The main difference between a linear regression model, such as $M$, and a neural network layer $L$,

$$L(X) = Y = \sigma(X * W + B), \tag{6}$$

is the activation function $\sigma$ which performs a non-linear transformation on the data. In this work the hyperbolic tangent function is chosen as the activation function $\sigma$. Single layers, such as $L$ in Eq. (6), are the core elements of deep neural network structures. In order to move from a non-linear single layer model to a deep neural network NN we simply stack multiple single layers,

$$\mathrm{NN}(X) = Y = (L_1 \circ L_2 \circ \ldots \circ L_{l-1} \circ L_l)(X), \tag{7}$$

where $l$ is the total number of layers and the output $Y_i$ of layer $L_i$, is the input $X_{i+1}$ of layer $L_{i+1}$ for $i \in (2, 3, \ldots, l-1)$. The first layer is referred to as the input layer and an the $l$th layer is referred to as the output layer. The layers between input and output layer are the hidden layers. The activation functions as well as the dimensions of the weight and bias matrices may vary between the layers.

With a neural network as defined in Eq. (7), we can construct a neural ODE where the derivatives are modelled by NNs. As an example ODE we consider a first order ODE with $n$ states which are denoted by $U$. Note, that we use $U'$ rather than $\dot{U}$ to denote the derivative with respect to time for notational convenience later. The right-hand side of the ODE is given by the derivative with respect to time, $t$, denoted by the function $f$,

$$U' = \frac{\mathrm{d}U}{\mathrm{d}t} = f(U). \tag{8}$$

For example, in a conventional mechanistic modeling setting in systems biology, we might be able to define $f$ as a polynomial presenting rate equations following the laws of mass action kinetics (Schnoerr et al. 2017). In such settings the structure of $f$ is implicitly given, and unknown parameters may be inferred utilising observed data and Bayesian approaches such as approximate Bayesian computation (Tankhilevich et al. 2020; Toni et al. 2008). However, more often than not we do find ourselves in circumstances where we are not able to define $f$ that concise. This could be due to a variety of reasons; for example, the required assumptions for mass action kinetics are not met, or, the list of influencing components is incomplete. In these scenarios, neural ODEs can be of great value. In the neural ODE, instead of a polynomial as in the conventional case, a neural network, NN (Eq. (7)), is used to model $f$. We denote the modeled derivative by $\tilde{U}'$,

$$\tilde{U}' = \mathrm{NN}(U). \tag{9}$$

The interpretation of the output of the neural network, NN, in the neural ODE, relates to the predicted gradient, $\mathrm{d}U/\mathrm{d}t$, in the states $U$ over the time interval $\mathrm{d}t$. We can also retrieve the predicted ODE solution $\tilde{U}$ using any ODE solver of choice,

$$\tilde{U} = \mathrm{ODEsolver}(\mathrm{NN}, u_0, t). \tag{10}$$

The attributes $u_0$ and $t$ denote the initial condition and the time span of interest, and specify the prediction format. Note that whilst the output of the neural network is the gradient at a specific position in the state space, the solution of the neural ODE is a path through the state space. We refer to $\tilde{U}$ as the prediction of the neural ODE.

## 2.2 Training neural ODEs with ODE solvers

During training we fit the neural ODE model to some observed data $U$. The aim is to find model parameters that describe the observation $U$; these parameters are features of the neural ODE, and are not to be confused with the kinetic parameters of the dynamical system that we seek to capture. Prior to training, the neural network NN modeling the derivative $\tilde{U}'$ in the neural ODE (Eq. (9)) is initialised. For the parameters $p_{\mathrm{init}}$ we choose $(W_1, B_1, W_2, B_2, \ldots, W_l, B_l)$, where $l$ is the total number of layers in NN, $W_i$ are samples from the Glorot uniform distribution (Glorot and Bengio 2010), and $B_i$ is zero for $i \in (1, 2, \ldots, l)$. We optimise the parameters by minimising the distance between the observation, $U$, and the current prediction of the neural ODE, $\tilde{U}$. We employ Euclidean distance, defined as

$$d(A, B) = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2}, \tag{11}$$

where $m$ is the length of $A$ and $B$. In this setting $A$ relates to the observed data, $U$, and $B$ relates to the current prediction of the neural ODE, $\tilde{U}$. For clarity of notation, we assume the observation $U$ to be one dimensional, e.g. the system consists of one state only. We denote it by $U = (u_1, \ldots, u_m)$ for $m$ time points. Similarly, the prediction of the neural network is then referred to as $\tilde{U} = (\tilde{u}_1, \ldots, \tilde{u}_m)$ for the same $m$ time points. The loss function (Figure 1, loss 1) is then given by

$$\mathrm{loss}_1(p) = d(U, \tilde{U}) = \sqrt{\sum_{i=1}^{m} (u_i - \tilde{u}_i)^2} = \sqrt{\sum_{i=1}^{m} (u_i - \mathrm{ODEsolver}(NN(U))_i)^2}, \tag{12}$$

where $\mathrm{ODEsolver}(\mathrm{NN}(U))_i$ relates to the solution of the ODE at the $i$th training point. Our optimiser of choice is Gradient Descent (Innes 2018). For the training we use a fixed number of training epochs.

In each training epoch, we calculate the loss value with the defined loss function for one set of parameters. By fixing the number of training epochs (compared to for example setting a cutoff loss value) we are able to control the number of tested parameters and the number of loss evaluations. The latter is especially important in regards to the overall performance. As the choice of the ODE solver and it's precision (which is related to $m$) is flexible, the loss function evaluation can get computationally expansive; this holds especially for large $m$. In successful training, we observe the general trend of a decreasing loss value over time. However, due to local optima, there are also areas in parameter space that may temporarily increase the loss.

## 2.3 Training neural ODEs with the collocation method

Training neural ODEs using ODE solvers has been shown to perform well in previous studies (Che et al. 2018; Rackauckas et al. 2020). The precision, especially for thoroughly trained models, tends to be high. However, the time performance of this training strategy does suffer, especially for long time series as even the fastest numerical solvers for ODEs are prohibitively slow given the number of training runs required. Furthermore, noise in the observations can easily disturb the learning behaviour.

Here we use an alternative approach of calibrating neural ODEs against the behaviour of dynamical systems. Using *collocation methods* (Liang and Wu 2008), we construct a new loss function for neural ODEs, which we show to be more efficient and robust. This new strategy involves estimating the solution and derivative of the observation prior to the training. In this new loss function we compare the output of the neural network for a solution directly with the estimated derivative of the observations. In this way we are able to avoid using ODE solvers in each loss evaluation. And furthermore, the estimator of the derivative allows us to smooth out noise.

The collocation method used in this work originates in the field of inverse problems of differential equations (Liang and Wu 2008) and has been implemented in the Julia package DiffEqParamEstim.jl. In inverse problems, the focus is on the estimation of parameters from observations of the state of the system. We denote the output of the true data generating process, $F$, by $U$, with

$$U = (u_{t_1}, \ldots, u_{t_m}), \quad u_{t_i}^T = (u_{t_i}^{s_1}, \ldots, u_{t_i}^{s_n}). \tag{13}$$

This describes a dynamical system where $m$ is the number of time points $t$, and $n$ is the number of states, $s$, in the system. Here we assume that our observations, $V$, differ from the actual state, $U$, by some noise. We thus have for $V$,

$$V = (v_{t_1}, \ldots, v_{t_m}), \quad v_{t_i}^T = (v_{t_i}^{s_1}, \ldots, v_{t_i}^{s_n}). \tag{14}$$

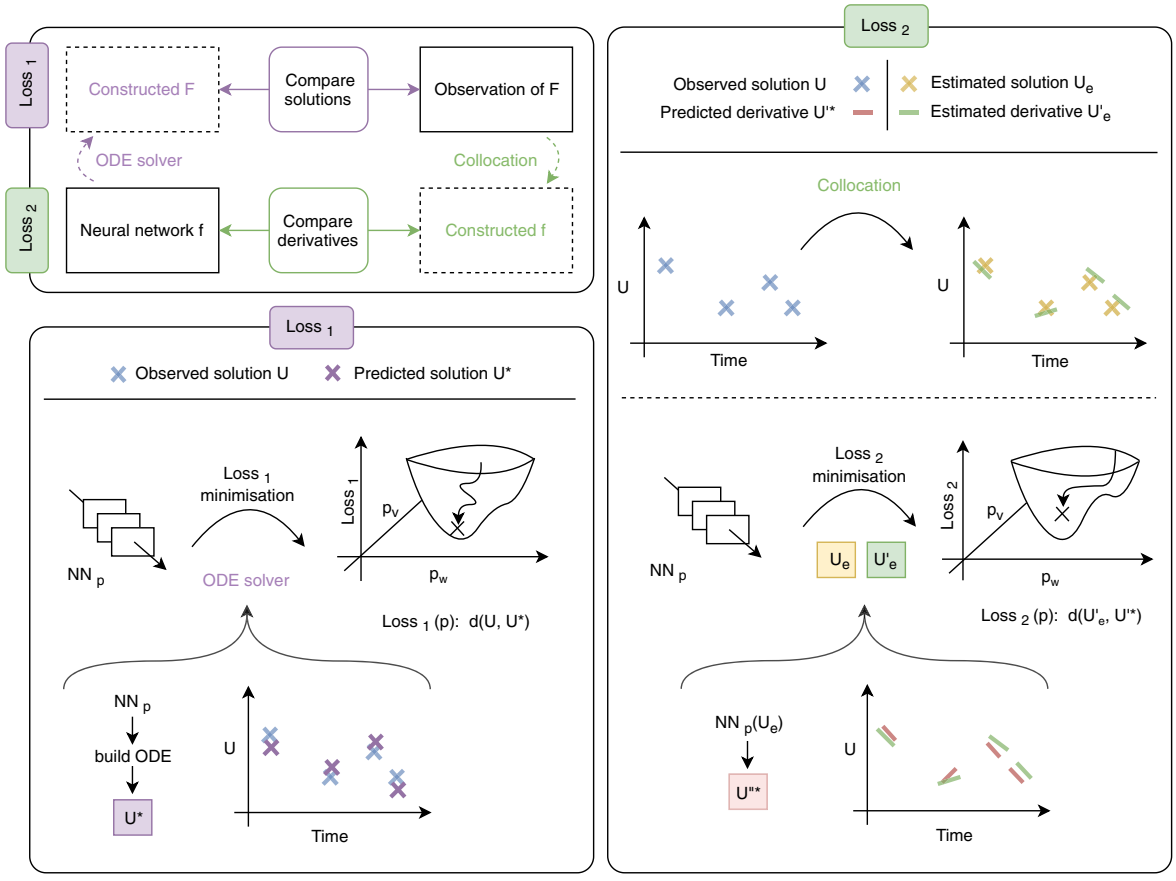


**Figure 1:** Two different training strategies for neural ODEs (loss 1 and loss 2) are described and compared. The most important difference between the two loss functions is that in the first loss function we compare solutions – we refer to them as $F$ – whereas in the second loss function we compare derivatives – these we refer to as $f$. This first training strategy (loss 1) relies on the distance between some observed data (blue) and the constructed ODE solution (purple) based on the neural network modeling the derivative. We optimise the parameters of the neural network by minimising this distance. In the second training strategy (loss 2), we estimate the solution (yellow) and the derivative (green) of the observed data (purple) using a collocation method. Subsequently, we compare the estimated derivative with the predicted derivative (red) of the neural network evaluated at the estimated solution. Again, we use an optimiser to find the parameters of the neural network that minimise the described distance.

For each time point $t_i$ we define the two local neighbourhoods $N_{1,t_i}$ and $N_{2,t_i}$

$$N_{1,t_i} = \begin{pmatrix} 1 & t_i - t_1 \\ 1 & t_i - t_2 \\ \dots & \dots \\ 1 & t_i - t_m \end{pmatrix}, \quad N_{2,t_i} = \begin{pmatrix} 1 & t_i - t_1 & (t_i - t_1)^2 \\ 1 & t_i - t_2 & (t_i - t_2)^2 \\ \dots & \dots & \dots \\ 1 & t_i - t_m & (t_i - t_m)^2 \end{pmatrix}, \tag{15}$$

and a diagonal kernel matrix $H_{t_i}$

$$H_{t_i} = \begin{pmatrix} H_{t_i}^{t_1} & \dots & \dots & 0 \\ \dots & H_{t_i}^{t_2} & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & H_{t_i}^{t_m} \end{pmatrix}, \tag{16}$$

and $H_{t_i}^{t_j} = \frac{K\left(\frac{(t_i - t_j)}{h}\right)}{h}$, with $h = m^{-\frac{1}{5}} \cdot m^{-\frac{3}{35}} \cdot \log(m)^{-\frac{1}{16}}$, and $K$ is the Epanechnikov kernel. The estimated solution of the ODE at time point $t_i$ and its derivative at time point $t_i$ are then given by

$$\hat{u}_i = e_1^T * inv(N_{1,t_i}^T * H * N_{1,t_i}) * N_{1,t_i}^T * H * V^T \tag{17}$$

and

$$\hat{u}_i' = e_2^T * inv(N_{2,t_i}^T * H * N_{2,t_i}) N_{2,t_i}^T * H * V^T, \tag{18}$$

where $e_1 = (1,0)^T$ and $e_2 = (0,1,0)^T$. We denote the summarised estimates by $\hat{U} = (\hat{u}_1, \dots, \hat{u}_m)$ and $\hat{U}' = (\hat{u_1}', \dots, \hat{u_m}')$. In order to evaluate the fit of a parameter combination we compare the estimated derivative $\hat{U}'$ and the resulting prediction of the neural network evaluated at the estimated solution of the ODE $\hat{U}$. We denote the latter as $\tilde{\hat{U}}'$, where

$$\tilde{\hat{U}}' = (\tilde{\hat{u}}_1', \dots, \tilde{\hat{u}}_m'), \quad \tilde{\hat{u}}_i' = NN_p(\hat{u}_i), \tag{19}$$

and $p$ is the currently tested parameter combination. We define this second loss function (Figure 1, loss 2) for the parameter $p$ as

$$\text{loss}_2(p) = d(\tilde{\hat{U}}', \hat{U}') = \sqrt{\sum_{i=1}^{m} (\tilde{\hat{u}}_i' - \hat{u}_i')^2} = \sqrt{\sum_{i=1}^{m} (NN_p(\hat{u}_i) - \hat{u}_i')^2}, \tag{20}$$

where $d$ is again the Euclidean distance (Eq. (11)).

In this loss function we compare derivatives. By doing so we avoid having to use an explicit ODE solver to produce the predicted time series based on the current neural network in each loss evaluation. This is of advantage, because – even when using efficient, state of the art ODE solvers – the computation of the ODE solution becomes quickly too expensive to be practical when performed in a highly repetitive manner. In the standard neural ODE model we use the ODE solver in each loss evaluation, therefore each training epochs requires an ODE solver run. Using the collocation method avoids these computations.

# 3 Results

Here we outline how neural ODEs are trained. We sketch how different factors affect this training, and illustrate the use and usefulness of neural ODEs in the context of representative exemplar dynamical systems. Julia code implementing the suggested training approach as well as all examples of this article can be found at https://github.com/ElisabethRoesch/neural_ODE_fitting. Reference implementations of the approach can be also found in the DiffEqFlux.jl library (Rackauckas et al. 2020).

## 3.1 Damped oscillator

We train neural ODEs with suitable training data obtained by solving the dynamical systems (or suitable experimental data in applications to real-world systems). Solving differential equation systems can become computationally limiting and we can often employ alternatives, such as the collocation method, the use of which we illustrate here. First, we present the training results for a damped oscillatory system (Figure 2). Damped oscillations are frequently encountered in technical and biological systems (Silk et al. 2011), and a generic example (following (Chen et al. 2019; Rackauckas et al. 2019)) of such a system is given by,

$$\frac{dX}{dt} = -0.1X^3 - 2Y^3, \tag{21}$$

$$\frac{dY}{dt} = 2X^3 - 0.1Y^3, \tag{22}$$

where we have chosen parameters that give the desired behaviour, but which are to all intents and purposes generic. We generate our observation by solving the system (with the initial conditions of $X = 1.5$ and $Y = 0.0$) at 100 time points using the *Tsitouras 5/4 Runge-Kutta* method (Rackauckas and Nie 2017).

In order to learn the underlying dynamics of the system, we model the derivative using a deep neural network with three layers. The first layer, is a cubic function, while the second and third layers are dense layers with 100 nodes each. As an activation function we choose the hyperbolic tangent. We initialise the model's parameter $W$ using the Glorot uniform distribution (Glorot and Bengio 2010), and zeros bias $B$. For the optimiser we chose Gradient Descent with a learning rate of 0.001, and we train for 800 epochs (we have tested other combinations of solvers and optimisers and obtain the same results).

Training of the neural ODE with the collocation method is vastly quicker than training with an Euclidean norm loss function, simply because we do not have to solve the ODE in each training epoch (Figure 2). The accuracy of training, in turn, depends primarily on the accuracy of the collocation method (Figure 5).

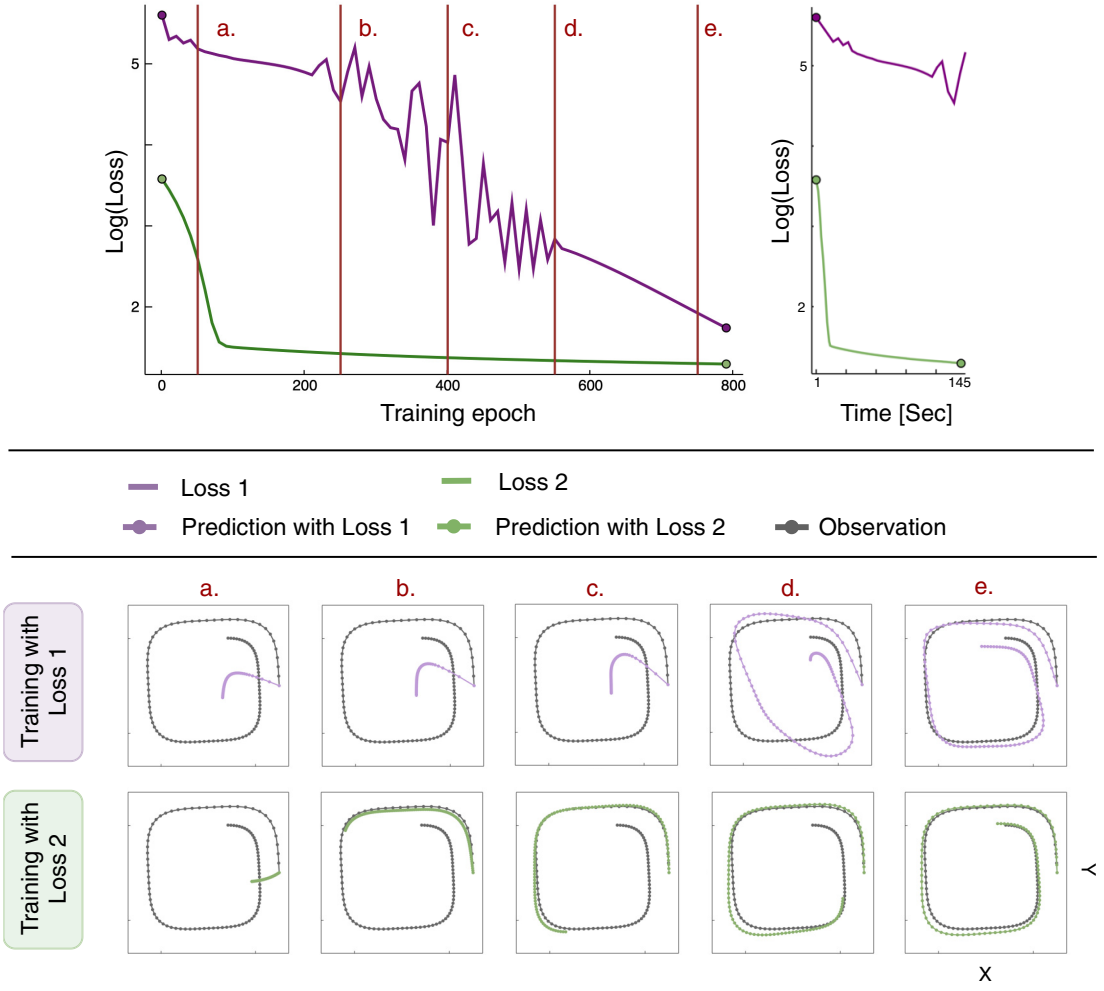


**Figure 2:** We train the neural ODE model with the two training strategies on data of the damped oscillator: Loss 1 (purple) represents the standard procedure using the Euclidean distance between the observation and the prediction of the neural ODE model. Loss 2 (green) represents the loss where we compare the predicted derivative of the neural network in the neural ODE directly with the estimated derivative of the observation using the collocation method. We show how the loss value decreases over the training (800 epochs) as well as the loss evolution over time (145 s). As the collocation approach results in significant shorter training time, we only show the loss evolution until the collocation approach terminates. For the selected training epochs (a) to (e) (red), the current predictions (model trained with loss 1: Purple, model trained with loss 2: Green) are shown alongside the observation (grey), respectively.

We also find that collocation loss decreases more quickly (and continuously) than the Euclidean distance does (the learning rate of the optimiser is identical in both instances). This simply reflects that collocation directly represents the right-hand-side of Eq. (8) (or Eq. (9)). So in many practical applications, this suggests, that we may prefer the collocation methods because of its speed and convenience.

## 3.2 Van der Pol oscillator

As a second example, we choose the Van der Pol (VdP) oscillator. The VdP oscillator has been an important example system in the dynamical systems literature, and has served as a starting point (Heinonen et al. 2018)

for e.g. the analysis of neuronal action potentials (Estakhroueieh et al. 2014). It carries the hallmarks of more complex models of oscillatory systems, and it is defined by the two differential equations

$$\frac{\mathrm{d}X}{\mathrm{d}t} = Y, \tag{23}$$

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = \mu \cdot (1 - X^2) \cdot Y - X, \tag{24}$$

and here we consider, for concreteness, the system for $\mu = 1$ as this results in moderate gradients (for high values of $\mu$ gradients are no longer well captured by the collocation method). We simulate the observed data by solving the ODE at 200 time points using again the *Tsitouras 5/4 Runge-Kutta* method (Rackauckas and Nie 2017). As initial conditions we choose $X = 2.0$ and $Y = 0.0$. This builds the base of the training data. Additionally, for the VdP example we modify the training data, because we want to stress test our model and training strategy; we induce the data with four levels of technical (observational) noise. This corresponds to a state-space model (Durbin and Koopman 2012), where the data are perturbed by additive Gaussian noise. The levels chosen here reflect different noise intensities and are: no noise ($\sigma = 0$), low noise ($\sigma = 0.1$), moderate noise ($\sigma = 0.2$), and high noise ($\sigma = 0.5$), where $\sigma$ is the parameter of a zero centred Normal distribution. For each case, the training set covers one period and the testing set includes five periods (Figure 3).

To model the gradient in our VdP neural ODE model, we use a neural network with four layers, where input and output layer consist of 100 parameters each, and the two hidden layers of 2500 parameters, respectively. The activation function, initialisation and optimiser, are chosen as in our first example, the damped oscillator (described in the previous Section 3.1). Using neural ODEs trained with the collocation method, we are able to capture the core structure of the VdP oscillator – the limit cycle – for all noise levels. In the training period the prediction aligns sufficiently well with the ODE solution for all noise levels (final loss values on training data for no/low/medium/high noise: 0.78, 2.51, 5.35, 12.97). For medium and high noise, however, the extrapolation in the testing area, does not capture the true ODE solution. However, the general, or qualitative structure of the solution is still recognisable.

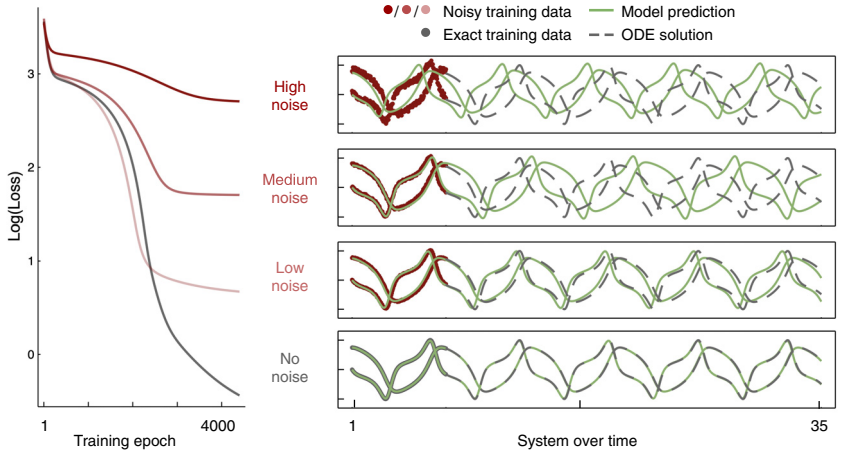


**Figure 3:** We illustrate the performance of the neural ODE model trained with the collocation method using the Van der Pol oscillator. More specifically, we focus on how training and testing are affected when noise is added to the solution. On the left side we show how the loss value evolves over the course of the training process. This is shown for four training sets with high, medium, low, and no noise (dark red, medium red, light red, and grey). We model the noise with a Gaussian distribution, $\mathcal{N}(0, \sigma)$. The levels of noise are: no noise ($\sigma = 0$), low noise ($\sigma = 0.1$), medium noise ($\sigma = 0.2$) and high noise ($\sigma = 0.5$). The loss values drops the slowest for high and medium noise, and the final loss value, e.g. the loss value of the trained model, is higher for training data with higher noise. On the right side of the figure, we show the model's prediction. This is visualised alongside the training data (red scatter), as well as the true ODE solution (grey). For no and low noise the predictive power is the highest. For medium and high noise, the shape of the periods are still clearly recognisable in the prediction of the neural ODE model, however prediction and reference data do not align.

In the test setting used here the training points of the models cover approximately one period of the oscillator (Simulation time span is from 0.0 to 7.0) for all noise levels. In reality, however, we often face the challenge of incomplete data. Therefore, we are interested in investigating whether a coverage of 100% is necessary in order to learn the dynamics of the underlying system – using neural ODEs trained with the collocation method (Figure 4). We simulate new training data of the VdP system for a second test for the neural ODE model trained with our collocation method. In this instance, we generate training data with low coverage (70, 85 and 90% of the period) and data with high coverage (100, 115 and 120% of the period).

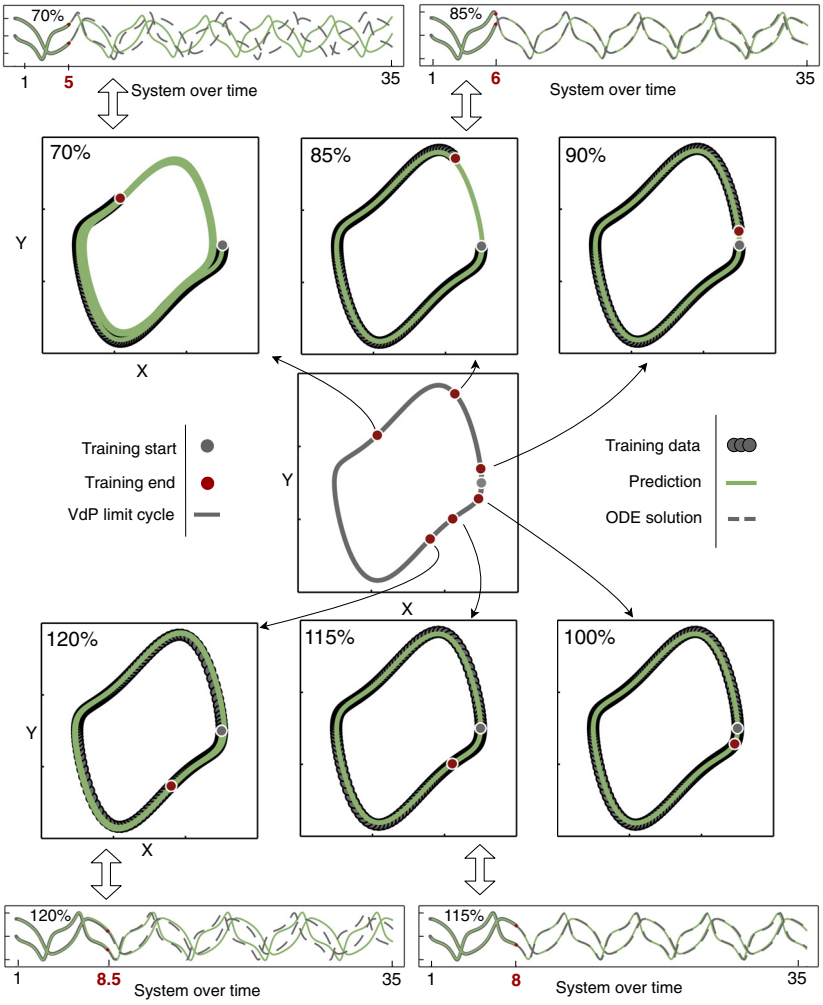


**Figure 4:** We demonstrate the predictive power of the neural ODE model on incomplete and over-complete training sets for the Van der Pol oscillator. In the centre we show the ODE solution of the Van der Pol oscillator (grey line) with the initial condition, e.g. Training start (grey dot) and various last training data points, e.g. Training end (red), in the state space (X,Y). The intention of this figure is to show how the prediction performance is affected, by the position of the last training data point. We compare six end points; they are at 5.0, 6.0, 6.5, 7.0, 8.0, and 8.5 and therefore cover approximately 70, 85, 90, 100, 115, and, 120% of one period of the oscillator. While we train on different levels of completeness, we test all models over five periods. In all cases the circular structure is learnt. Shown in the top left corner, the prediction of the model trained on 70% of the data (green line) does not align with the reference ODE solution (grey, dashed line). However, positioning the end point of the training data to 85% is enough to learn the dynamics of the system (top right). For 85% the endpoint is over the last curve of the limit cycle of the oscillator which may reason for this. Adding more points to training data, e.g. 100, 115, 120% does only cause no (top right, bottom right) or minor differences (bottom left).

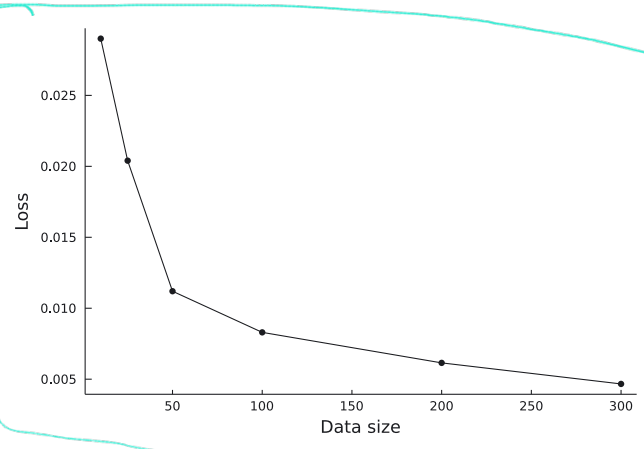


**Figure 5:** This figure shows how the collocation method reacts to sparsity in the training data. The normalized minimal loss value obtained during the training phase is plotted against the provided data size of the underlying training set. Neural ODEs are trained on data of the damped oscillator example defined in Eqs. (21) and (22). The selected data sizes are 10, 25, 50, 100, 200, and 300, respectively. We observe that with increasing data density, the minimal loss value is decreasing.

Our analysis suggests that in order to archive the highest predictive power it is necessary to include all key features defining the structure of the systems dynamics in the training data. In the given example, this includes the final (forth) curve of the limit cycle of the VdP oscillator. In the case of the lowest data coverage training data (70%), we are only able capture the core of the dynamic system – the limit cycle (final loss value for 70%: 0.86 on training data). However, the prediction for the testing periods does not align perfectly with the ODE solution. In all other cases the given information in the respective training data is sufficient to predict highly accurate, even with e.g. 15% of period missing (final loss value for 85%: 0.76 on training data).

Training of data-driven methods typically requires considerable amounts of data, and we would encourage their use only in such situations unless we have explicit ways to deal with sparsity in the training data (Žurauskienė et al. 2014). In Figure 5 we show that collocation training is nevertheless robust against data sparsity (in the regime where data-based modelling seems appropriate).

## 4 Conclusion

We have demonstrated the efficient and effective use of collocation methods in the training of neural ODEs. With a damped oscillator as an example, we have shown that training is faster and the (collocation) loss decreases more continuously over the course of the training, compared to conventional training with an Euclidean loss function as this requires explicit numerical solution of the ODE at each step. We know that in a real world scenario, the true signal will often not be accurately measurable. Therefore, we have also simulated training data in which the observation of the system deviates from the true ODE solution in a controlled manner. This has allowed us to map out the dependence of neural ODEs trained with the collocation method to a number of real-world data issues.

First, we have investigated how the performance of neural ODEs trained with the collocation method is affected by the addition of noise to the training data using the Van der Pol oscillator. Second, we have analysed how the modelling performance deteriorates as the size of the training data decreases. Noise in data will be a near ubiquituous problem in systems biology application; but neural ODEs trained with the collocation method show encouraging robustness to such noise. We note that in the envisaged applications we will always be in a data-rich regime; noise will be more widespread and its analysis more important, than the problem of data sparsity (which is known to affect interpolation methods quite generally). Our analysis suggests that the general structure (e.g. the structure of the limit cycle) of a dynamical system can still be learnt even in the presence of substantial noise. For moderately sparse data we are able to capture the system's structure as reflected by the gradient field. However, like all methods that seek to provide data-driven descriptions – such as Gaussian processes (Rasmussen and Williams 2006) or recurrent neural networks (Che et al. 2018) – the quality of the neural ODE as a description of the system rapidly deteriorates as data become too sparse. In the version used here, this deterioration is not captured to the extent it would be by Gaussian processes, where

the variance reflects the spacing of training points. On the other hand, neural ODEs capture the connected nature of the system's states better than the conventional single-output Gaussian processes; multi-output Gaussian processes (Álvarez et al. 2010; Žurauskienė et al. 2014) could offer a multi-variate perspective, but have only rarely been used. In combination with other non-parametric Bayesian methods, such as Dirichlet processes (Crook et al. 2019; Murphy 2012), it may thus be possible to generate flexible hybrid modelling frameworks.

We want to reiterate the importance of hybrid models – modeling approaches which bridge the two areas of mechanistic and data-driven modeling (Baker et al. 2018). In many cases mechanistic models are superior at explaining and understanding, while data-driven modelling tends to have higher predictive power. We encourage the use of hybrid models as they are have the potential to perform well in both areas. Neural ODEs (Chen et al. 2019) represent one methodology that falls into this category. Hybrid models such as neural ODEs could also help with improving modelling performance, as they provides the flexibility to incorporate partial prior knowledge. For neural ODEs, in particular, interpretability of system dynamics can also result, as by training the neural ODE, we are actually able to learn the whole vector field; this can provide mechanistic insights (Jost 2005; Tyson et al. 2003). From the perspective of mechanistic modeling, if there is insufficient knowledge to develop ODE or SDE models, but where there is a substantial amount of data, hybrid models such as a neural ODE model might be a good choice. In systems biology specifically, we see great potential of neural ODEs for systems such as a cellular signalling systems, ecological dynamical systems, or electrophysiological processes in the nervous system. With the collocation method as a new training strategy for neural ODEs, we are able to train the models faster, than is possible using the usual way of fitting ODE models to observed data. The main advantage of this speed, in our opinion, is that it allows us to explore more models, and larger models more comprehensively. And in situations where we lack mechanistic models this is a highly desirable thing (Babtie et al. 2014; Scholes et al. 2019).

**Acknowledgment:** We gratefully acknowledge discussions with members of the Theoretical Systems Biology group at University of Melbourne, Australia, and at Imperial College London, United Kingdom, as well as with the Julia community. The information, data, or work presented herein was funded in part by ARPA-E under award numbers DE-AR0001222 and DE-AR0001211, and NSF award number IIP-1938400. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
**Author contribution:** All the authors have accepted responsibility for the entire content of this submitted manuscript and approved submission.
**Research funding:** None declared.
**Conflict of interest statement:** The authors declare no conflicts of interest regarding this article.

## References

Álvarez, M., Luengo, D., Titsias, M., and Lawrence, N. (2010). Efficient multioutput Gaussian processes through variational inducing kernels. J. Mach. Learn. Res. 9: 25–32.
Babtie, A.C., Kirk, P., and Stumpf, M.P.H. (2014). Topological sensitivity analysis for systems biology. Proc. Natl. Acad. Sci. U.S.A. 111: 18507–18512.
Baker, R.E., Peña, J.-M., Jayamohan, J., and Jérusalem, A. (2018). Mechanistic models versus machine learning, a fight worth fighting for the biological community?. Biol. Lett. 14: 20170660.
Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. (2018). Recurrent neural networks for multivariate time series with missing values. Sci. Rep. 8.
Chen, R.T.Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2019). Neural ordinary differential equations. arXiv.
Crook, O.M., Gatto, L., and Kirk, P.D.W. (2019). Fast approximate inference for variable selection in dirichlet process mixtures, with an application to pan-cancer proteomics. Stat. Appl. Genet. Mol. Biol. 18: 20180065.
Dupont, E., Doucet, A., and Teh, Y.W. (2019). Augmented neural ODEs. arXiv.
Durbin, J. and Koopman, S.J. (2012). *Time series analysis by state space methods. Oxford statistical science series*, 2nd ed. Oxford University Press, Oxford.

Estakhroueieh, M., Nikravesh, S., and Gharibzadeh, S. (2014). ECG generation based on action potential using modified van der pol equation. Annu. Res. Rev. Biol. 4: 4259–4272.

Gardiner, C. (2009). *Stochastic methods: a handbook for the natural and social sciences*. Springer, Berlin and Heidelberg.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. J. Mach. Learn. Res. 9: 249–256.

Gupta, A. and Khammash, M. (2014). Sensitivity analysis for stochastic chemical reaction networks with multiple time-scales. Electron. J. Probab. 19.

Heinonen, M., Yildiz, C., Mannerström, H., Intosalmi, J., and Lähdesmäki, H. (2018). Learning unknown ODE models with Gaussian processes. arXiv.

Innes, M. (2018). Flux: elegant machine learning with julia. J. Open Source Software 3: 602.

Innes, M., Saba, E., Fischer, K., Gandhi, D., Rudilosso, M.C., Joy, N.M., Karmali, T., Pal, A., and Shah, V. (2018). Fashionable modelling with flux. arXiv.

Jia, J. and Benson, A.R. (2019). Neural jump stochastic differential equations. arXiv.

Jost, J. (2005). *Dynamical systems examples of complex behaviour*. Springer, Berlin and Heidelberg.

Kersting, H., Krämer, N., Schiegg, M., Daniel, C., Tiemann, M., and Hennig, P. (2020). Differentiable likelihoods for fast inversion of 'likelihood-free' dynamical systems. arXiv.

Kirk, P., Thorne, T., and Stumpf, M.P. (2013). Model selection in systems and synthetic biology. Curr. Opin. Biotechnol. 24: 767–774.

Kirk, P.D.W., Babtie, A.C., and Stumpf, M.P.H. (2015). Systems biology (un)certainties. Science 350: 386–388.

Lakatos, E. and Stumpf, M.P.H. (2017). Control mechanisms for stochastic biochemical systems via computation of reachable sets. R. Soc. Open Sci. 4: 160790.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature 521: 436–444.

Leon, M., Woods, M.L., Fedorec, A.J.H., and Barnes, C.P. (2016). A computational method for the investigation of multistable systems and its application to genetic switches. BMC Syst. Biol. 10.

Liang, H. and Wu, H. (2008). Parameter estimation for differential equation models using a framework of measurement error in regression models. J. Am. Stat. Assoc. 103: 1570–1583.

Liepe, J., Filippi, S., Komorowski, M., and Stumpf, M.P.H. (2013). Maximizing the information content of experiments in systems biology. PLoS Comput. Biol. 9: e1002888.

Liepe, J., Kirk, P., Filippi, S., Toni, T., Barnes, C.P., and Stumpf, M.P.H. (2014). A framework for parameter estimation and model selection from experimental data in systems biology using approximate bayesian computation. Nat. Protoc. 9: 439–456.

Liu, X., Xiao, T., Si, S., Cao, Q., Kumar, S., and Hsieh, C.-J. (2019). Neural SDE: stabilizing neural ODE networks with stochastic noise. arXiv.

Milias-Argeitis, A., Summers, S., Stewart-Ornstein, J., Zuleta, I., Pincus, D., El-Samad, H., Khammash, M., and Lygeros, J. (2011). In silico feedback for in vivo regulation of a gene expression circuit. Nat. Biotechnol. 29: 1114–1116.

Murphy, K.P. (2012). *Machine learning: a probabilistic perspective*. MIT Press, Cambridge, Massachusetts and London, England.

Rackauckas, C. and Nie, Q. (2017). DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. J. Open Res. Software 5.

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., and Dixit, V. (2019). DiffEqFlux.jl–a julia library for neural differential equations. arXiv.

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., and Ramadhan, A. (2020). Universal differential equations for scientific machine learning. arXiv.

Rasmussen, C.E. and Williams, C.K.I. (2006). *Gaussian processes for machine learning*. MIT Press, Cambridge, Massachusetts and London, England.

Roesch, E. and Stumpf, M.P.H. (2019). Parameter inference in dynamical systems with co-dimension 1 bifurcations. R. Soc. Open Sci. 6: 190747.

Rubanova, Y., Chen, R.T.Q., and Duvenaud, D. (2019). Latent ODEs for irregularly-sampled time series. arXiv.

Schnoerr, D., Sanguinetti, G., and Grima, R. (2017). Approximation and inference methods for stochastic biochemical kinetics–a tutorial review. J. Phys. A: Math. Theor. 50: 093001.

Scholes, N.S., Schnoerr, D., Isalan, M., and Stumpf, M.P.H. (2019). A comprehensive network atlas reveals that turing patterns are common but not robust. Cell Syst. 9: 243–257.e4.

Silk, D., Barnes, C.P., Kirk, P.D.W., Kirk, P., Toni, T., Rose, A., Moon, S., Dallman, M.J., Stumpf, M.P.H., and Stumpf, M.P.H. (2011). Designing attractive models via automated identification of chaotic and oscillatory dynamical regimes. Nat. Commun. 2: 489.

Tankhilevich, E., Ish-Horowicz, J., Hameed, T., Roesch, E., Kleijn, I., Stumpf, M.P.H., and He, F. (2020). GpABC: a julia package for approximate bayesian computation with Gaussian process emulation. Bioinformatics 36: 3286–3287.

Toni, T., Welch, D., Strelkowa, N., Ipsen, A., and Stumpf, M.P. (2008). Approximate bayesian computation scheme for parameter inference and model selection in dynamical systems. J. R. Soc. Interface 6: 187–202.

Tyson, J.J., Chen, K.C., and Novák, B. (2003). Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell. Curr. Opin. Cell Biol. 15: 221–231.

Tzen, B. and Raginsky, M. (2019). Neural stochastic differential equations: deep latent Gaussian models in the diffusion limit. arXiv.

Žurauskienė, J., Kirk, P., Thorne, T., and Stumpf, M.P. (2014). Bayesian non-parametric approaches to reconstructing oscillatory systems and the nyquist limit. Phys. A 407: 33–42.