EasyChair Preprint

№ 15198

Modelling Dynamical Systems: Learning ODEs with No Internal ODE Resolution

Johanne Cohen, Emmanuel Goutierre, Hayg Guler, Fotios Kapotos, Sida-Bastien Li, Michèle Sébag and Bowen Zhu

October 6, 2024

Modelling dynamical systems: Learning ODEs with no internal ODE resolution

Johanne Cohen[1] ✉, Emmanuel Goutierre[1], Hayg Guler[2], Fatios Kapotos[3], Sida-Bastien Li[3] Michèle Sébag[1], and Bowen Zhu[3]

[1] LISN, CNRS, Université Paris-Saclay, France
firstname.lastname@universite-paris-saclay.fr
[2] IJCLAB, CNRS, Université Paris-Saclay, France
hayg.guler@ijclab.in2p3.fr
[3] CentraleSupelec, Université Paris-Saclay, France
firstname.lastname@student-cs.fr

Abstract. The quest for accurate modelling and simulation of dynamical systems is the Holy Grail of computational physics and numerical engineering. In deep learning, main approaches proposed in the literature include prediction by time series and modelling by Ordinary Differential Equations (ODEs). The usual methods for learning optimal parameters then consist of formulating the question as a reachability problem and then optimizing some suitable cost function for this reachability problem. However, these two approaches fail to model specific complex dynamical systems. The presented work considers the case of modelling and predicting the behaviour of beams in particle accelerators. The difficulty lies in the associated dynamic, which is highly versatile and possibly discontinuous. In order to extend the scope of dynamical system modelling to meet the particle accelerator modelling challenge, we present a new approach that can cover this context called implicit neural ODE (INode). It uses the modelling of discontinuous behaviour through integral operators; these operators are used to pre-process the data to get a more classical regression problem. Finally, the global model of the dynamical system is formulated as the solution of an ODE, which contains the solution of the regression problem. INode thus enables the learning of a data-driven ODE while removing the computationally heavy ODE resolution from the training loop.
The formal analysis of the approach establishes its consistency and convergence properties under moderate assumptions.

1 Introduction

Machine learning techniques have profoundly altered the field of dynamic system modelling and numerical engineering, leveraging the data and/or prior knowledge to meet the prediction challenge. Depending on the available knowledge, the sought model can be specified to satisfy physical constraints and/or used to generate new data consistent with the physics of the problem.

Two main approaches have been presented in the literature in the context of deep learning. A first approach and a long-studied direction tackle the prediction of time series forecasts. A second approach aims at identifying the ordinary differential equation (ODE) underlying the observed data [5,30,19,22].[4]

From an abstract point of view, these (deep learning) methods consist of formulating the question as a reachability problem and then optimizing some suitable cost function on this reachability problem. Such parameters are obtained using some descent gradient method. This leads to particular methods, such as the adjoint method. These methods fit on methods trying to optimize the reachability of some desired goal from some given initial data, in some incremental manner, guided by the local optimization of some cost function: we provide a concise review in Section 3.1 and how this approach relates to some reachability question.

Our motivating application is modelling particle accelerators, where particles are generated and submitted to diverse electromagnetic fields to deliver a beam with a prescribed behaviour. Efficient simulators have been designed to predict the beam behavior with the desired level of accuracy. Their limitation is that they are too computationally heavy to support, e.g., the number of experiments required to calibrate the accelerator and achieve the prescribed behavior with the desired accuracy. Several approaches, centered on learning surrogate simulators and inspired from time series forecasting, have been presented in the literature [23,2,34,9,33,29].

However, it turns out from our experiments that the above existing approaches cannot model the highly versatile behaviour of a particle accelerator in the general case. The behaviour of the beam is controlled by the several dozen electromagnetic fields involved in the accelerator chambers, referred to as *particle accelerator control settings* in what follows, and control settings when there is no confusion to be feared. Depending on the control settings in particular, a significant fraction of the particles may be ejected and the behaviour of the beam becomes discontinuous.

Consequently, the existing methodology must be adapted to cover these highly versatile and possibly discontinuous behaviours. One of our key ideas is not to work directly over the inputs but on their transformation by a smooth, deterministic operator. This aims to reduce the versatility of the data involved. An example of a considered operator is the integral of the inputs. Notice that this is inspired by ideas coming from analogue computations [31,4,3], where, for example, integration was considered as an operation far more stable than derivative. This is especially true when the observable trajectory u experiences a discontinuity at some time t^*, implying that the derivative is infinite at this point, unlike the integral.

[4] A third emerging direction involves generative modelling, where the generated examples are filtered using the prior knowledge to support the identification of the sought system. This approach is outside the scope of the paper and will not be considered in the remainder.

Indeed, a new methodology, enabling the modelling of dynamic systems with highly diverse and possibly discontinuous behaviour, is presented in this article and referred to *Implicit Neural ODE* (INode). INode works in four stages.

1. Firstly, the observed behavioral data of the dynamical system is transformed using integral operators; a new, continuous-by-design trajectory is computed, called *implicit trajectory*.
   Notice that we use the vocabulary sequential data and trajectory interchangeably in the following.
2. Secondly, the implicit trajectory is used to define a classical regression problem aimed to predict the observed trajectory from the implicit trajectory.
3. Thirdly, the implicit trajectory is by design solution of an ODE involving the original trajectory. We then consider the implicit ODE where the original trajectory is replaced by its approximation, learned in Step 2. The implicit ODE is solved and yields an approximate implicit trajectory.
4. Lastly, the original trajectory is estimated from the approximate implicit trajectory using the model learned in Step 2.

The main contributions of the approach are as follows. Firstly, INode aims to characterize parametrized ODEs (one ODE for each control settings), while prominent NODE approaches aim to characterize *the* ODE best fitting the data. Secondly, the theoretical analysis of the approach shows that, under moderate assumptions, i) the solution of the implicit ODE is unique (stage 3. above); ii) the original trajectory can be approximated with arbitrary precision in the large sample limit (universal approximation property). Thirdly, and importantly, the ODE resolution is contained in Stage 3., thus outside of the learning loop (Stage 2. above), with a significant gain in computational complexity.

2 Related work

Neural ODEs Ordinary Differential Equations is a widespread tool to model the evolution of systems over time [12]. The theory of dynamical systems, Ordinary Differential Equations (ODEs), Partial Differential Equations (PDEs) and their applications to various contexts is a common topic in mathematics, physics, and, more generally, in every applied science. Their use and adequateness in the context of deep learning have been pointed out recently [27]. In particular, neural ODEs [5] have demonstrated their performance in various contexts, particularly with respect to traditional approaches in deep-learning problems. Neural ODEs are particularly suitable for contexts with temporal dependencies, providing extensions of tools such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks [13], and hence refining traditional modeling methods (see for an overview [8]). Research and applications involving Neural Ordinary Differential Equations (Neural ODEs) are typically explored within two main frameworks: Continuous-depth deep learning models [5] and Physics-Informed Neural Networks (PINNs) [25]. PINNs leverage insights from the continuous

nature of physics underlying the differential equations, thereby aiding learnability by incorporating pre-existing knowledge. By integrating physics-informed principles with machine learning, as exemplified in PINNs, the learning process is anchored in established physical laws that the neural networks must adhere to. This approach assumes prior knowledge about the observed data, ensuring a more robust and interpretable learning process. Achieving such promising results involves a suite of techniques, including adaptive sampling methods [7], specialized neural network architectures [28], and dedicated optimization tools [14]. Applying them to high-dimensional problems remains a challenge.

The work considered here is more closely related to the continuous-depth deep learning model approach and its variants [5,27]. Neural ODEs are often associated with problems related to learning time series [17,5,26]. In particular, an approach is to consider the learning of suitable latent variables using models such as *Latent ODEs*. Some continuous-time extensions of (classical discrete-time) transformers have been proposed. It has been demonstrated that many variants of transformers can be extended to deal with irregular time series modeling [6]. Here, we learn directly from some hyperparameters and from the initial data but we do not work on time series. Learning a Neural ODE is classically based on techniques such as the adjoint method, and this requires in practise some numerical methods. Methods to reduce the practical costs have been proposed: [16] uses higher-order derivatives of solution trajectories to improve the process. Some methods have been proposed in constitutive equations by hardwiring some constraints from physics to help the learning process in this specific context of mechanics [21].

The context of particle accelerators Building a surrogate particle accelerator is a difficult task because the associated learning process requires to deal with discontinuous inputs (trajectories) to be learned. This is a difficulty in ODE-based approaches, as differentiability implies continuity. On the other hand, differentiation is well-known from an engineering point of view as a very numerically unstable operation. Our approach is based on not working with functions and their derivatives but working with their integrals.

Concerning various approaches and practical models to deal with similar contexts, we can mention the following: investigations with discontinuous dynamics resulted in the formulation of Neural Jump Stochastic Differential Equations, [15] enabling the modelling of systems experiencing sudden shifts. This innovation expands the utility of Neural ODE frameworks to represent diverse real-world scenarios more accurately. In addressing practical challenges such as irregularly-sampled data, the articles [26,19,20] showcased the adaptability of Neural ODEs to real-world data collection scenarios, enhancing their utility in various applications.

The scope of Neural ODEs has been broadened by introducing a framework for modeling systems ruled by integro-differential equations [35]. This seems to be pivotal in areas where understanding cumulative effects over time is essential. This progress diversifies the Neural ODE toolkit, accommodating a broader spectrum of dynamic behaviours.

3 Problem Description

In supervised learning, we deal with a set of inputs $\mathcal{X} \subset \mathbb{R}^n$ and corresponding outputs $\mathcal{Y} \subset \mathbb{R}^m$, both subsets of Euclidean spaces. The aim is to accurately approximate a *ground truth* function, depicted as a mapping $F : \mathcal{X} \to \mathcal{Y}$.

Here, F is typically considered as the solution of a reachability problem: it is assumed that there is some underlying Ordinary Differential Equation (ODE) $\frac{\mathrm{d}\mathsf{u}}{\mathrm{d}t}(t, \mathbf{c}) = f(\mathsf{u}(t, \mathbf{c}), t, \mathbf{c})$ so that F is the function that maps initial condition $u(0, \mathbf{c}) = x$ to its value $\mathsf{u}(T_1, \mathbf{c})$ at some given fixed time T_1 (or possibly at a set of given times $(T_i)_i$, but for simplicity we consider here first the case of only one measurement $T = T_1$). In other words, F is the flow of some underlying ODE, or if one prefers, the set of points reached from an initial condition x at some given time T_1.

To achieve this, we want to approximate the underlying *ground truth* function f by a family of approximating functions f_θ parametrized by $\theta \in \Theta$. The set of parameters, Θ, typically resides within another subset of an Euclidean space.

In a generic view, this problem then comes back to tackle an optimization problem generally expressed as $\inf_{\theta \in \Theta} \mathbb{E}_{x \sim \mathcal{X}}[\ell(F_\theta(x), F(x))]$, wherein $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ serves as a *loss function* and the input distribution is represented by a probability measure on $\mathcal{X}$. In this work, the definition of the loss function is the regression loss $\ell(x, y) = ||x - y||^2$.

If one prefers, assume that F is some fixed function, T_1 is some fixed parameter. We search for the optimal θ minimizing the loss function, measuring the (square of the) distance between $F(x)$ and $F_\theta(x)$, where F is the flow associated to f, and F_θ is the flow associated to f_θ (both at time $T = T_1$). As usual, in the context of deep learning, the difficulty is that when we state that F and T_1 are fixed, we only know F on some input/output points, but we do not know explicitly fully function F. We search for the best θ, minimizing the loss function on the available data.

In the more general case of many time/parameters measurements (i.e; possibly a set of given times $(T_i)_i$), we consider an *observable trajectory* as a function $\mathsf{u} : [t_0, T] \times \mathbb{R}^p \to \mathbb{R}^n$. We may then have T distinct from a single point T_1, but all the $(T_i)_i$: namely, $\sup_i T_i < T$. We always assume a trajectory is L^∞ (integrable and bounded), representing the state trajectory of a system, where p denotes the dimension of the state space. In practice, this trajectory is influenced by a set of p control parameters $\mathbf{c} \in \mathbb{R}^m$ which dictate the system's dynamics. Our objective is to learn an approximation for u based on known initial conditions $\mathsf{u}(t_0, \mathbf{c})$ and the control parameters $\mathbf{c}$, with the available data.

3.1 Background in our context

This can be related to Recurrent Neural Networks (RNNs). In RNNs, the system's evolution from one time step to the next is described by the equation:

$$\mathsf{u}(t_{i+1}, \mathbf{c}) = \mathsf{u}(t_i, \mathbf{c}) + f_\theta(\mathsf{u}(t_i, \mathbf{c})) \tag{1}$$

Here, f is a function parameterized by θ, representing the system's behaviour dynamics. θ corresponds to the weights.

The concept of NODE [5] can be interpreted as a continuous generalization of the classical recurrent neural network by adding additional layers and progressively decreasing the step size. NODE's novelty lies in considering the function u as the solution of an ODE:

$$\frac{\mathrm{d}\mathsf{u}}{\mathrm{d}t}(t,\mathbf{c}) = f(\mathsf{u}(t,\mathbf{c}),t,\mathbf{c}) \tag{2}$$

A neural network f_θ is employed to approximate the unknown f function. The estimate function $\widehat{\mathsf{u}}$ is the solution to the initial value problem defined by f_θ, and $\mathsf{u}(t_0,\mathbf{c})$.

$$\frac{\mathrm{d}\widehat{\mathsf{u}}}{\mathrm{d}t}(t,\mathbf{c}) = f_\theta(\widehat{\mathsf{u}}(t,\mathbf{c}),t,\mathbf{c}) \tag{3}$$

The state estimate at any time can thus be predicted by solving the ODE:

$$\widehat{\mathsf{u}}(t,\mathbf{c}) = \mathrm{ODESolve}(f_\theta,\mathsf{u}(t_0,\mathbf{c}),t_0,t) \tag{4}$$

$\mathrm{ODESolve}(f_\theta,\mathsf{u}(t_0,\mathbf{c}),t_0,t)$ serves as a black-box that solves Eq. (3), starting with the value $\mathsf{u}(t_0,\mathbf{c})$ at time t_0. In summary, the NODE algorithm can be formulated in our context as follows:

1. Initialize a neural network f_θ (weights are randomly initialized)
2. For a control parameter $\mathbf{c}$, and its associated trajectory $\mathsf{u}(t,\mathbf{c})$
   (a) Compute $\widehat{\mathsf{u}}(t,\mathbf{c})$ such that $\frac{\mathrm{d}\widehat{\mathsf{u}}}{\mathrm{d}t}(t,\mathbf{c}) = f_\theta(\widehat{\mathsf{u}}(t,\mathbf{c}),t,\mathbf{c})$ using an ODE solver.
   (b) Compute the loss $L(\widehat{\mathsf{u}}(T,\mathbf{c})) = \ell(\widehat{\mathsf{u}}(T,\mathbf{c}),\mathsf{u}(T,\mathbf{c})) = ||\widehat{\mathsf{u}}(T,\mathbf{c}) - \mathsf{u}(T,\mathbf{c})||^2$.
   (c) Compute the gradient of the loss $\ell(\widehat{\mathsf{u}}(T,\mathbf{c}),\mathsf{u}(T,\mathbf{c}))$ using the adjoint method (see [5] or below).
   (d) Update the network weights based on the gradient.

To calculate the gradient of the loss function $L(\widehat{\mathsf{u}})$, the adjoint method is classicalyutilized [5]: The adjoint state $\mathsf{a}(t)$ is defined as the gradient of L with respect to the state u, i.e., $\mathsf{a}(t,\mathbf{c}) = \nabla_{\mathsf{u}(t,\mathbf{c})}L$. The key insight of the adjoint method is that the dynamics of $\mathsf{a}(t,\mathbf{c})$ is given by another ODE:

$$-\frac{d\mathbf{a}(t,\mathbf{c})}{dt} = \mathsf{a}(t,\mathbf{c})^T \frac{\partial f_\theta(\mathsf{u}(t,\mathbf{c}),t,\mathbf{c})}{\partial \mathsf{u}(t,\mathbf{c})}.$$

This ODE is solved backwards in time, from t_1 to t_0, with the initial condition $\mathbf{a}(t_1) = \nabla_{\mathbf{u}(t_1)}L$, obtained from the derivative of the loss with respect to the output of the ODE solver. The gradient $\nabla_\theta L$ is then computed by integrating

$$\mathsf{a}(t)^T \frac{\partial f_\theta(\mathsf{u}(t,\mathbf{c}),t,\mathbf{c})}{\partial \theta}$$

over time from t_0 to t_1. Despite the efficient memory consumption of such a method, the full differential equation must be solved at each training step, which

becomes particularly challenging when dealing with complex underlying neural networks or extended sequences.

This method provides an efficient way to find (possibly locally) optimal parameters θ, solving the associated reachability problem corresponding to the considered supervised learning problem.

4 Our approach

We propose a new model, INode, inspired by NODE, aimed at learning the observable trajectory u. Our approach involves incorporating additional information, referred to as *integrated trajectory* v, required to be a solution of a differential equation (controlled by u). In other words, v is designed to fulfill the differential equation:

$$\frac{\mathrm{d}\mathsf{v}}{\mathrm{d}t}(t, \mathbf{c}) = g(\mathsf{u}(t, \mathbf{c}), \mathsf{v}(t, \mathbf{c}), t) \tag{5}$$

where function g called *driving function*, defined as $g : \mathbb{R}^n \times \mathbb{R}^m \times [t_0, T] \to \mathbb{R}^m$ if it satisfied the three properties:

1. **Lipschitz Continuity in State Space**: for every $\mathbf{x_v} \in \mathbb{R}^m$ and $t \in [t_0, T]$, the function $g(\cdot, \mathbf{x_v}, t)$ is k_u-Lipschitz. This means there exists a constant $k_\mathsf{u} \in \mathbb{R}_+$ such that for all $(\mathbf{x_{u,1}}, \mathbf{x_{u,2}}) \in \mathbb{R}^n \times \mathbb{R}^n$, $\|g(\mathbf{x_{u,2}}, \mathbf{x_v}, t) - g(\mathbf{x_{u,1}}, \mathbf{x_v}, t)\| \leq k_\mathsf{u} \|\mathbf{x_{u,2}} - \mathbf{x_{u,1}}\|$.
2. **Lipschitz Continuity in Integral State Space**: for every $\mathbf{x_u} \in \mathbb{R}^n$ and $t \in [t_0, T]$, the function $g(\mathbf{x_u}, \cdot, t)$ is k_v-Lipschitz. Thus, for all $(\mathbf{x_{v,1}}, \mathbf{x_{v,2}}) \in \mathbb{R}^m \times \mathbb{R}^m$, $\|g(\mathsf{u}, \mathbf{x_{v,2}}, t) - g(\mathsf{u}, \mathbf{x_{v,1}}, t)\| \leq k_\mathsf{v} \|\mathbf{x_{v,2}} - \mathbf{x_{v,1}}\|$.
3. **Continuity over the Time**: the function g is continuous with respect to time, meaning that for any $\forall (\mathbf{x_u}, \mathbf{x_v}) \in \mathbb{R}^n \times \mathbb{R}^m$ the mapping $t \to g(\mathbf{x_u}, \mathbf{x_v}, t)$ is continuous over the interval $[t_0, T]$.

Note that for a given integrated trajectory v, multiple potential suitable functions g may exist, but we only require to know one. Inspired by analogue computational methods [31], our focus will primarily be on the integral operator, rendering the integrated trajectory as $\mathsf{v}(t, \mathbf{c}) = \int_{t=t_0}^{t} \mathsf{u}(s, \mathbf{c})\, \mathrm{d}s$. In this case, v corresponds to the integral.

Theorem 1 (Driving Function Existence). *Let $\mathsf{u} : [t_0, T] \times \mathbb{R}^p \to \mathbb{R}^n$ be a observable trajectory, let g be a driving function.*

For any initial condition $\mathsf{v}_0(\mathbf{c})$, there exists a unique absolutely continuous function $\mathsf{v} : [t_0, T] \times \mathbb{R}^p$, such that for all $\mathbf{c} \in \mathbb{R}^p$, for all $t \in [t_0, T]$, $\mathsf{v}(t, \mathbf{c}) = \mathsf{v}_0(\mathbf{c}) + \displaystyle\int_{t_0}^{t} g(\mathsf{u}(\tau, \mathbf{c}), \mathsf{v}(\tau, \mathbf{c}), \tau)\, \mathrm{d}\tau$.

In other words, there exists a unique function denoted as $\mathcal{F}$ and referred to as the *integral operator*, which maps observable trajectory u to integrated trajectory v: $\mathsf{v} = \mathcal{F}(\mathsf{u})$.

Assuming initial knowledge of the integrated trajectory's value $\mathsf{v}(t, \mathbf{c}) = \mathcal{F}(\mathsf{u})(t, \mathbf{c})$, we then proceed to train a neural network f_θ to learn the observable trajectory $\mathsf{u}(t, \mathbf{c})$ from integrated trajectory $\mathsf{v}(t, \mathbf{c})$:

$$\mathsf{u}(t, \mathbf{c}) = f_\theta(\mathsf{v}(t, \mathbf{c}), t, \mathbf{c}). \tag{6}$$

This neural network f_θ can be seen as defining a new ODE g_θ as follows:

$$\begin{aligned} \frac{\mathrm{d}\mathsf{v}}{\mathrm{d}t}(t, \mathbf{c}) &= g(f_\theta(\mathsf{v}(t, \mathbf{c}), t), \mathsf{v}(t), t, \mathbf{c}) \\ &= g_\theta(\mathsf{v}(t, \mathbf{c}), t, \mathbf{c}) \end{aligned} \tag{7}$$

This ODE will be learnt in two ways. Either we solve the regression problem defined by Eq. (6). Or we use the adjoint method to solve Eq. (7). Once we have learned g_θ, we might not have direct access to integrated trajectory v. However, at any time step, integrated trajectory v can be estimated by solving the ODE:

$$\widehat{\mathsf{v}}(t, \mathbf{c}) = \mathrm{ODESolve}(g_\theta, \mathsf{v}(t_0), t_0, t, \mathbf{c}) \tag{8}$$

Under the same assumptions as [5], g_θ fulfills the conditions of the Picard-Lindelöf theorem since most neural networks architecture are Lipschitz continuous functions, ensuring the unicity of the solution. For a formal proof and statement, refer to Theorem 2.

Theorem 2. *Let g be a driving function and f be a smooth estimator. For any initial condition $\mathsf{v}_0(\mathbf{c})$, there exists a unique function v such that for all $t \in [t_0, T]$, for all $\mathbf{c} \in \mathbb{R}^p$,*

$$\frac{\partial \mathsf{v}}{\partial t}(t, \mathbf{c}) = g(f(\mathsf{v}(t, \mathbf{c}), t, \mathbf{c}), \mathsf{v}(t, \mathbf{c}), t) \;\; \textit{and} \;\; \mathsf{v}(t_0, \mathbf{c}) = \mathsf{v}_0(\mathbf{c})$$

In other words, there exists a unique integrated trajectory that is the solution to this new ODE defined in Eq. (7). Then, we can estimate observable trajectory u by applying the following equation:

$$\widehat{\mathsf{u}}(t, \mathbf{c}) = f_\theta(\mathsf{v}(t), t, \mathbf{c}) \tag{9}$$

A key point of our approach is that we have access to additional information $\mathsf{v} = \mathcal{F}(\mathsf{u})$ during the training. This additional information is calculated from the datasets in a preprocessing step (Step 1 below).

In summary, the INode algorithm can be described as follows:

1. **Step 1: Operator preprocessing:** Compute $\mathsf{v} = \mathcal{F}(\mathsf{u})$ for training data.
2. **Step 2: Learning the ODE**
   (a) Initialize a neural network f_θ and consequently g_θ (Refer to Eq. (7) for understanding the relationship between the two networks)
   (b) For a control parameter $\mathbf{c}$, and its associated trajectory $\mathsf{u}(t, \mathbf{c})$
      i. Compute $\widehat{\mathsf{v}}(t, \mathbf{c})$ such that $\frac{\mathrm{d}\widehat{\mathsf{v}}}{\mathrm{d}t}(t, \mathbf{c}) = g_\theta(\widehat{\mathsf{v}}(t, \mathbf{c}), t, \mathbf{c})$ using an ODE solver.

ii. Compute the loss $L(\widehat{\mathsf{v}}(t,\mathbf{c})) = \ell(\widehat{\mathsf{v}}(t,\mathbf{c}), \mathsf{v}(t,\mathbf{c}))$.
iii. Compute the gradient of the loss $\ell(\widehat{\mathsf{v}}(t,\mathbf{c}), \mathsf{v}(t,\mathbf{c}))$ using the adjoint method.
iv. Update the network weights based on the gradient.

3. **Step 3: Evaluation**
   (a) Compute $\widehat{\mathsf{v}}$ by solving the ODE defined by g_θ (see Eq. (8))
   (b) Estimate u as $\widehat{\mathsf{u}} = f_\theta(\mathsf{v})$ (see Eq. (9))

The INODE algorithm acts as a universal approximator for any observable trajectory since this observable trajectory is integrable and essentially bounded. In other words, it allows us to approximate the discrepancy between u and $f_\theta(\mathsf{v})$ to an arbitrary degree, given the value of v (see Theorem 3): given the function v, there exists a neural network f_θ that acts as a universal approximator for $\mathsf{u}(t,\mathbf{c})$ (see. Step 2).

Following the notation from [18], we denote by $\mathcal{NN}^{\rho}_{m,n,k}$ the set of multilayer perceptrons that have m neurons in the input layer, n neurons in the output layer and an arbitrary number of hidden layer each containing at least k neurons each, using the activation function ρ.

Theorem 3 (Universal Approximation knowing the integrated trajectory). *Let u be a observable trajectory and g be a driving function. Let v be the integrated trajectory of u guaranteed by Theorem 1: $\mathsf{v} = \mathcal{F}_{g,\mathsf{v}_0}(\mathsf{u})$.*

For all $\varepsilon > 0$, for all $k > 0$, there exists a neural network

$$f_\theta \in \mathcal{NN}^{\mathrm{ReLU}}_{m+p+1,n,\max(m+p+2,n)}$$

such that f_θ is a smooth estimator with a uniform Lipschitz constant bounded by k, and for all $t, \mathbf{c} \in [t_0, T] \times \mathbb{R}^p$, $\|f_\theta(\mathsf{v}(t,\mathbf{c}), t, \mathbf{c}) - \mathsf{u}(t,\mathbf{c})\| < \varepsilon$.

Then, having such an approximation, we can infer another approximation for observable trajectory, this time without prior knowledge of the value of integrated trajectory (see Theorem 4).

Theorem 4 (Universal Approximation). *Let u be a observable trajectory and g be a driving function. Let v be the integrated trajectory of u corresponding to solution guaranteed by Theorem 1: $\mathsf{v} = \mathcal{F}_{g,\mathsf{v}_0}(\mathsf{u})$.*

For all $\varepsilon > 0$, for all constant $k > 0$, there exists $\varepsilon_1 > 0$ such that all neural networks $f_\theta \in \mathcal{NN}^{\mathrm{ReLU}}_{m+p+1,n,\max(m+p+2,n)}$ that satisfies the two following properties

1. *f_θ is a smooth estimator with a uniform Lipschitz constant bounded by k,*
2. *and for all $t, \mathbf{c} \in [t_0, T] \times \mathbb{R}^p$, $\|f_\theta(\mathsf{v}(t,\mathbf{c}), t, \mathbf{c}) - \mathsf{u}(t,\mathbf{c})\| < \varepsilon_1$.*

verify: $\|f_\theta(\mathbf{G}_{g,\mathsf{v}_0}(f_\theta)(t,\mathbf{c}), t, \mathbf{c}) - \mathsf{u}(t,\mathbf{c})\| < \varepsilon$ *where*
$\mathbf{G}_{g,\mathsf{v}_0}(f_\theta)(t,\mathbf{c}) = \int_{t_0}^{t} g(f_\theta(\mathsf{v}(\tau,\mathbf{c}), \tau, \mathbf{c}), \mathsf{v}(\tau,\mathbf{c}), \tau)\,\mathrm{d}\tau$.

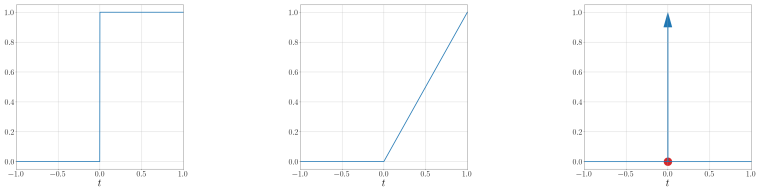
In other words, Theorem 4 says that if f_θ is a universal approximator for $\mathsf{u}(t, \mathbf{c})$, then the trajectory computed by Step 3 defines a universal approximator for $\mathsf{u}(t, \mathbf{c})$.

We have outlined our method using a generic operator between u and v, requiring only this operator to be differentiable. For example, we can outline three specific operators:

- the integral operator that we call INODE,
- the exponential smoothing (EXP-INODE), inspired by the exponential smoothing operator widely recognized in the literature,
- and a combination of the strategies of the first two approaches (Comb-INODE).

INODE*: Case when* v *corresponds to the integral of* u*:* $\mathsf{v}(t, \mathbf{c}) = \int_{t=t_0}^{t} \mathsf{u}(s, \mathbf{c})\, \mathrm{d}s$.

As previously mentioned, integration is known to be a numerically more stable operation than differentiation, particularly in the context of analog computations [31]. It is especially true when the signal u experiences a discontinuity at some time t^*, where the derivative would be infinite, unlike the integral. This concept can be illustrated with the Heaviside function H whose derivative, in the sense of distributions, is the Dirac delta distribution δ, whereas its integral is the ramp function R (also called ReLU function). When the observable trajectory u is the Heaviside function H, the integrated trajectory $\mathsf{v}(t, \mathbf{c})$ is continous (see Figure 1). Unlike NODE-based methods that must approximate the Dirac delta function, our method is designed to learn a mapping from the ramp function to the Heaviside function, which can be expressed in the following way: $H(t) = g(R(t), t) = \frac{R(t)}{t}$.



$\mathsf{u}(t, \mathbf{c}) = H(t)$ $\quad$ $\mathsf{v}(t, \mathbf{c}) = \int_{t=0}^{t} H(s)\, \mathrm{d}s$ $\quad$ $\frac{\mathrm{d}u}{\mathrm{d}t}(t, \mathbf{c}) = \delta(t)$

Fig. 1: Function u whose its integral is more stable than its derivative. The red point in the middle figure signifies the discontinuity.

Note that Step 3.(a) of INODE, which involves computing an integral, can be efficiently performed using simple numerical methods such as the Trapezoidal rule.

EXP-INODE*: Case when* v *corresponds to the Exponential Smoothing of* u*:*

$$\mathsf{v}\left(t, \mathbf{c}\right) = \mathsf{u}\left(t_0, \mathbf{c}\right) e^{-\lambda(t-t_0)} + \int_{t=t_0}^{t} \lambda e^{-\lambda(t-s)} \mathsf{u}\left(s, \mathbf{c}\right) \, \mathrm{d}s \tag{10}$$

Where λ is a positive constant. The exponential smoothing (or exponential moving average) results from averaging the past signal, but applying forgetting weight exponentially decreases with time. This operator is also chosen for its smoothing capability, acting as a low-pass filter inspired by signal processing theory.

Note that v fulfills an (ODE): $\frac{\mathrm{d}\mathsf{v}}{\mathrm{d}t}(t, \mathbf{c}) = \lambda\left(\mathsf{u}(t, \mathbf{c}) - \mathsf{v}(t, \mathbf{c})\right)$. This equation confirms that Step 3.(a) of Method INODE can be implemented.

Similar to the previous case, calculating $\mathsf{v}\left(t, \mathbf{c}\right)$ can be achieved by relying on the dataset because it involves a composition of an integral and an exponential function.

Comb-INODE*: Case when* $\mathsf{v}\left(t, \mathbf{c}\right)$ *corresponds to a combinations of the two previous cases:* Finally, the combination of these two approaches—integrating and exponential smoothing—suggests a hybrid strategy where both operators are employed to harness their respective advantages.

$$\mathsf{v}\left(t, \mathbf{c}\right) = \begin{bmatrix} \mathsf{v}_1\left(t, \mathbf{c}\right)) \\ \mathsf{v}_2\left(t, \mathbf{c}\right)) \end{bmatrix} = \begin{bmatrix} \int_{t_0}^{t} \mathsf{u}\left(s, \mathbf{c}\right)) \, ds \\ \mathsf{u}\left(t_0, \mathbf{c}\right) e^{-\lambda(t-t_0)} + \int_{t=t_0}^{t} \lambda e^{-\lambda(t-s)} \mathsf{u}\left(s\right) \, \mathrm{d}s \end{bmatrix} \tag{11}$$

Now, an integrated trajectory corresponds to $\mathsf{u} = f_\theta\left(\mathsf{v}\right) = f_\theta\left(\begin{bmatrix} \mathsf{v}_1 \\ \mathsf{v}_2 \end{bmatrix}\right)$. We then have a system of ODEs that we can also solve:

$$\frac{\mathrm{d}\mathsf{v}\left(t, \mathbf{c}\right)}{\mathrm{d}t} = \begin{bmatrix} \frac{\mathrm{d}\mathsf{v}_1(t, \mathbf{c})}{\mathrm{d}t} \\ \frac{\mathrm{d}\mathsf{v}_2(t, \mathbf{c})}{\mathrm{d}t} \end{bmatrix} = \begin{bmatrix} f_\theta\left(\mathsf{v}(t, \mathbf{c})\right) \\ \lambda(f_\theta\left(\mathsf{v}(t, \mathbf{c})\right) - \mathsf{v}_2(t, \mathbf{c})) \end{bmatrix}.$$

5 Experiments Results

We aim to create a surrogate for a part of the ThomX particle simulator [32]: This accelerator began operation in 2021 and is currently in the commissioning phase). We aim to focus on the linear accelerator section (corresponding to the part where the electron beam accelerates from zero velocity to the speed of light).

Dataset The overall Linac dataset [11] includes 4000 simulations, computed by Astra [10] and uniformly divided into a training set (80%), a validation set (10%) and a test set (10%). The overall dimension of the control settings ($\mathbf{c}$) is 36 [24]. It includes detailed simulation outputs of electron beams passing through a linear accelerator, capturing a series of parameters that describe beam dynamics.

Each trajectory i is represented by 4000 points, denoted as $(t_j, \mathsf{u}(t_j, \mathbf{c}_j))_{1 \leq j \leq 4000}$ where t_j are the time instances and $\mathsf{u}(t_j, \mathbf{c}_i)$ are the corresponding values of the trajectory under specific parameters $\mathbf{c}_i$. The time for each point t_j is uniformly drawn from the interval $[0, 9.393]$.

Experiments Set-up We evaluate our three variant approaches alongside the baselines (LSTM [13] and NODE [5]). Additionally, we explore two methodologies to resolve Eq. (7): One approach employs a regression technique, and the second refines the outcome of the first using the adjoint method. We will specify the use of the adjoint method by incorporating the term *"with finetuning"*.

All experiments, except LSTM, employ a Fully Connected Neural Network (FCNN). The NODE is trained using the loss function defined in the seminal paper [5]. The selection of hyperparameters, such as the optimizer, specifics of the ODE solver, sampling points, method of normalization, learning rate, and criteria for loss, is conducted through extensive testing using Optuna [1], a hyperparameter optimization framework.

Performance Measurement To compare the performance of the different models, we compare the predicted trajectory position with the trajectory from the dataset (referred to as ground truth). We will use the coefficient of determination R^2 to compare these two trajectories. The R^2 value is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where y_i are the ground truth values, $\hat{y}_i$ are the predicted values, and $\bar{y}$ is the mean of the actual values. The R^2 value indicates how well the predicted trajectory matches the ground truth, with a 1 indicating a perfect match.

Experiments Results This section aims to evaluate the capability of INode and its variations to accurately represent a LINAC's behaviour and compare their effectiveness with conventional time series forecasting models.

Method	LSTM	NODE	INode	EXP-INode	Comb-INode
R^2	0.9588	0.9448	0.9911	0.9920	0.9944

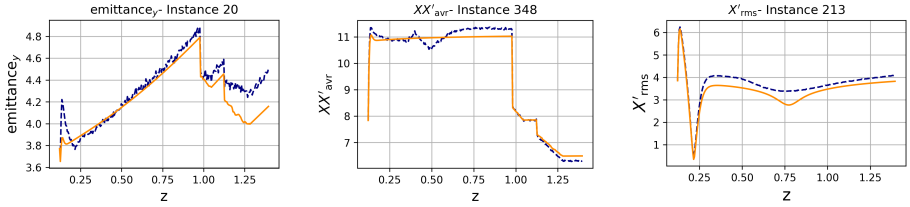
Table 1: Comparison of Global Metrics Across Different Methods

The results presented in Table 1 highlight the superior performance of the INode models over traditional time series prediction models such as LSTM and NODE.

We aim to visualize the predicted beam properties compared to the ground truth trajectory given by the Astra [10] beam tracking simulator. We focus on three significant parameters to characterize the beam:

1. *Emittance* measures the spread of a particle beam in phase space, representing the beam's quality and tendency to diverge. It describes the size and angular divergence of the beam, with lower emittance values indicating a higher quality beam that is more focused and less prone to spreading out.
2. *Average horizontal position-angle correlation in the beam transverse coordinates* (denoted by XX'_{avr}) is the covariance between the horizontal position of particles within the beam and their horizontal propagation angles.
3. *Horizontal angular spread in the beam transverse coordinates* (denoted by X'_{ms}) is a measure of the spread of particle angles in the horizontal plane, quantified as the root mean square (RMS) of these angles. It indicates how much the particles within the beam deviate from the average direction in the horizontal axis.

Figures 2, 3, 4, 5 and 6 offer a visual comparison of model performance across varying levels of sequence discontinuity in the dataset. Although the NODE model achieves satisfactory R^2 scores, its main strength lies in predicting the general scale of the sequences rather than handling discontinuities. In contrast, even the base INODE model accurately captures these discontinuities or exponential smoothing.

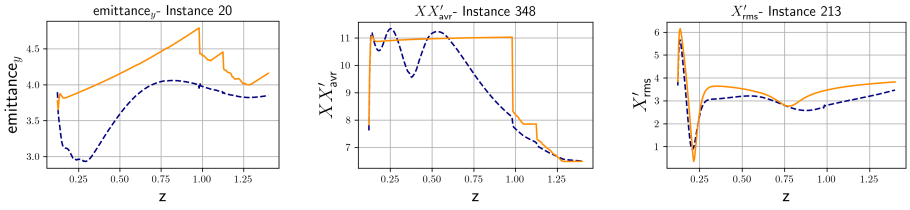


(a) Emittance over time (b) XX'_{avr} over time (c) X'_{ms} over time

Fig. 2: Performance of LSTM. The orange curve represents the ground truth trajectory, while the blue curve represents the trajectory predicted by the model.

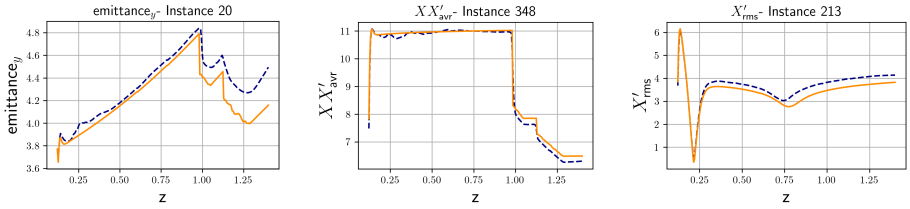
6 Conclusion

This article reviews the basic approach based on Neural Ordinary Differential equations for time series prediction and ODE modelling. It consists of searching from some underlying ordinary differential equation and searching for some (possibly local) optimal of a cost function over the associated reachability problem. This is done by using a gradient descent and requires the computation of the gradient of this cost function. The adjoint method is a classical method for evaluating this gradient, which solves another differential equation. This can be seen as a continuous time version of the backpropagation algorithm [5].



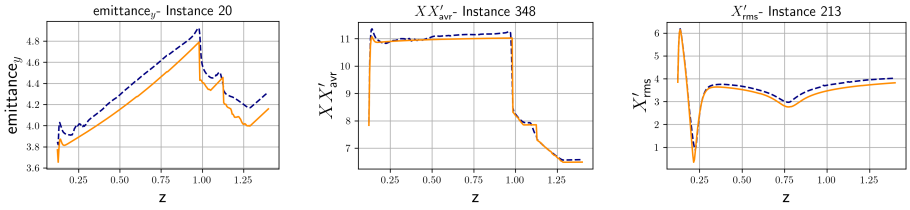
(a) Emittance over time (b) XX'_{avr} over time (c) X'_{ms} over time

Fig. 3: Performance of NODE. The orange curve represents the ground truth trajectory, while the blue curve represents the trajectory predicted by the model.



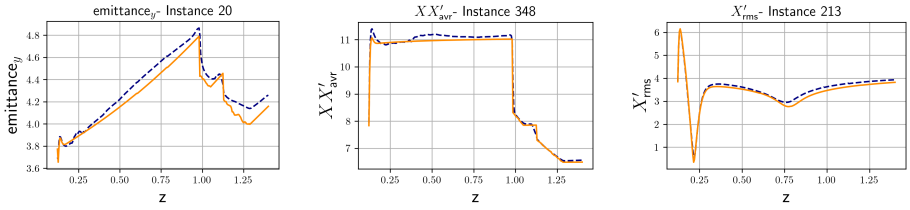
(a) Emittance over time (b) XX'_{avr} over time (c) X'_{ms} over time

Fig. 4: Performance of INODE. The orange curve represents the ground truth trajectory, while the blue curve represents the trajectory predicted by the model.



(a) Emittance over time (b) XX'_{avr} over time (c) X'_{ms} over time

Fig. 5: Performance of EXP-INODE. The orange curve represents the ground truth trajectory, while the blue curve represents the trajectory predicted by the model.



(a) Emittance over time (b) XX'_{avr} over time (c) X'_{ms} over time

Fig. 6: Performance of Comb-INode. The orange curve represents the ground truth trajectory, while the blue curve represents the trajectory predicted by the model.

This study introduces the Implicit Neural ODE (INode) framework, extending the approach in dynamical systems modelling, specifically tailored for applications involving discontinuous behaviours, such as those observed in particle accelerators. Unlike traditional methods, INode leverages integral operators to transform the input data into a more manageable form, bypassing the discontinuities' complexities. Our results demonstrate that INode not only addresses the challenges of modelling systems with abrupt behavioural changes.

Through rigorous theoretical analysis, we have established the consistency and convergence properties of the INode framework, validating its effectiveness under moderate assumptions. Moreover, our approach's ability to learn data-driven ODEs without direct interaction with the heavy computational process of ODE resolution, particularly when discontinuous are involved, marks a significant advancement over existing techniques, offering improved efficiency and robustness.

Acknowledgments. The authors would like to thank Olivier Bournez for his insightful and fruitful comments and constructive discussions on neural networks.

References

1. Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
2. S. Biedron, L. Brouwer, D. L. Bruhwiler, N. M. Cook, A. L. Edelen, D. Filippetto, C. K. Huang, A. Huebl, T. Katsouleas, N. Kuklev, R. Lehe, S. Lund, C. Messe, W. Mori, C. K. Ng, D. Perez, P. Piot, J. Qiang, R. Roussel, D. Sagan, A. Sahai, A. Scheinker, M. Thévenet, F. Tsung, J. L. Vay, D. Winklehner, and H. Zhang. Snowmass21 accelerator modeling community white paper, 2022.
3. Olivier Bournez and Manuel L. Campagnolo. *New Computational Paradigms. Changing Conceptions of What is Computable*, chapter A Survey on Continuous Time Computations, pages 383–423. Springer-Verlag, New York, 2008.

4. Olivier Bournez and Amaury Pouly. A survey on analog models of computation. In Vasco Brattka and Peter Hertling, editors, *Handbook of Computability and Complexity in Analysis*. Springer, 2021.
5. Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
6. Yuqi Chen, Kan Ren, Yansen Wang, Yuchen Fang, Weiwei Sun, and Dongsheng Li. Contiformer: Continuous-time transformer for irregular time series modeling. *Advances in Neural Information Processing Systems*, 36, 2024.
7. Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Mitigating propagation failures in physics-informed neural networks using retain-resample-release (r3) sampling. *arXiv preprint arXiv:2207.02338*, 2022.
8. Robert DiPietro and Gregory D Hager. Deep learning: Rnns and lstm. In *Handbook of medical image computing and computer assisted intervention*, pages 503–519. Elsevier, 2020.
9. Luca Fedeli, Axel Huebl, France Boillod-Cerneux, Thomas Clark, Kevin Gott, Conrad Hillairet, Stephan Jaure, Adrien Leblanc, Rémi Lehe, Andrew Myers, et al. Pushing the frontier in the design of laser-based electron accelerators with groundbreaking mesh-refined particle-in-cell simulations on exascale-class supercomputers. In *SC22: international conference for high performance computing, networking, storage and analysis*, pages 1–12. IEEE, 2022.
10. K. Flöttmann. ASTRA: A space charge tracking algorithm, manual, 2017. Technical report, 2017.
11. E. Goutierre. Linac dataset from Thomx. `https://zenodo.org/records/11084340`, 2024.
12. Morris W Hirsch, Stephen Smale, and Robert L Devaney. *Differential equations, dynamical systems, and an introduction to chaos*. Academic press, 2012.
13. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
14. Zheyuan Hu, Khemraj Shukla, George Em Karniadakis, and Kenji Kawaguchi. Tackling the curse of dimensionality with physics-informed neural networks. *arXiv preprint arXiv:2307.12306*, 2023.
15. Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
16. Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning differential equations that are easy to solve. *Advances in Neural Information Processing Systems*, 33:4370–4380, 2020.
17. Patrick Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.
18. Patrick Kidger and Terry Lyons. Universal Approximation with Deep Narrow Networks. In Jacob Abernethy and Shivani Agarwal, editors, *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 2306–2327. PMLR, 09–12 Jul 2020.
19. Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. In *Advances in Neural Information Processing Systems*, volume 33, pages 6696–6707. Curran Associates, Inc.
20. Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series.

21. Filippo Masi and Ioannis Stefanou. Evolution tann and the identification of internal variables and evolution equations in solid mechanics. *Journal of the Mechanics and Physics of Solids*, 174:105245, 2023.
22. Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. *Advances in Neural Information Processing Systems*, 33:3952–3963, 2020.
23. John A Miller, Mohammed Aldosari, Farah Saeed, Nasid Habib Barna, Subas Rana, I Budak Arpinar, and Ninghao Liu. A survey of deep learning and foundation models for time series forecasting. *arXiv preprint arXiv:2401.13912*, 2024.
24. H Purwar, E Goutierre, H Guler, M Rossetti Conti, S Chance, A Gonnin, H Monard, A Bacci, M Sebag, J Cohen, et al. Random error propagation on electron beam dynamics for a 50 mev s-band linac. *Journal of Physics Communications*, 7(2):025002, 2023.
25. Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017.
26. Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
27. Lars Ruthotto. Differential equations for continuous-time deep learning. *arXiv preprint arXiv:2401.03965*, 2024.
28. Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020.
29. Kunxiang Sun, Xiaolong Chen, Xiaoying Zhao, Xin Qi, Zhijun Wang, and Yuan He. Surrogate model of particle accelerators using encoder-decoder neural networks with physical regularization. *International Journal of Modern Physics A*, 38:2350145–2928, 2023.
30. Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit, 2019.
31. Bernd Ulmann. *Analog and hybrid computer programming*. Walter de Gruyter GmbH & Co KG, 2023.
32. Alessandro Variola, Jacques Haissinski, Alexandre Loulergue, Fabian Zomer, et al. Thomx technical design report. 2014.
33. J-L Vay, A Huebl, R Lehe, NM Cook, RJ England, U Niedermayer, P Piot, F Tsung, and D Winklehner. Modeling of advanced accelerator concepts. *Journal of Instrumentation*, 16(10):T10003, 2021.
34. Liling Xiao, Lixin Ge, Zenghai Li, and Cho-Kuen Ng. Advances in multiphysics modeling for parallel finite-element code suite ace3p. *IEEE Journal on Multiscale and Multiphysics Computational Techniques*, 4:298–306, 2019.
35. Emanuele Zappala, Antonio H. de O. Fonseca, Andrew H. Moberly, Michael J. Higley, Chadi Abdallah, Jessica A. Cardin, and David van Dijk. Neural integro-differential equations. 37(9):11104–11112, 2023.