ANAMNESIC NEURAL DIFFERENTIAL EQUATIONS WITH ORTHOGONAL POLYNOMIALS PROJECTIONS

Edward De Brouwer
ESAT-STADIUS
KU Leuven
Leuven, Belgium
edward.debrouwer@kuleuven.be

Rahul G. Krishnan
Department of Computer Science
University of Toronto
Toronto, Canada
rahulgk@cs.toronto.edu

ABSTRACT

Neural ordinary differential equations (Neural ODEs) are an effective framework for learning dynamical systems from irregularly sampled time series data. These models provide a continuous-time latent representation of the underlying dynamical system where new observations at arbitrary time points can be used to update the latent representation of the dynamical system. Existing parameterizations for the dynamics functions of Neural ODEs limit the ability of the model to retain global information about the time series; specifically, a piece-wise integration of the latent process between observations can result in a loss of memory on the dynamic patterns of previously observed data points. We propose PolyODE, a Neural ODE that models the latent continuous-time process as a projection onto a basis of orthogonal polynomials. This formulation enforces long-range memory and preserves a global representation of the underlying dynamical system. Our construction is backed by favourable theoretical guarantees and in a series of experiments, we demonstrate that it outperforms previous works in the reconstruction of *past and future data*, and in downstream prediction tasks. Our code is available at https://github.com/edebrouwer/polyode.

1 INTRODUCTION

Time series are ubiquitous in many fields of science and as such, represent an important but challenging data modality for machine learning. Indeed, their temporal nature, along with the potentially high dimensionality makes them arduous to manipulate as mathematical objects. A long-standing line of research has thus focused on efforts in learning informative time series representations, such as simple vectors, that are capable of capturing local and global structure in such data (Franceschi et al., 2019; Gu et al., 2020). Such architectures include recurrent neural networks (Malhotra et al., 2017), temporal transformers (Zhou et al., 2021) and neural ordinary differential equations (neural ODEs) (Chen et al., 2018).

In particular, neural ODEs have emerged as a popular choice for time series modelling due to their sequential nature and their ability to handle irregularly sampled time-series data. By positing an underlying continuous time dynamic process, neural ODEs sequentially process irregularly sampled time series via piece-wise numerical integration of the dynamics between observations. The flexibility of this model family arises from the use of neural networks to parameterize the temporal derivative, and different choices of parameterizations lead to different properties. For instance, bounding the output of the neural networks can enforce Lipschitz constants over the temporal process (Onken et al., 2021).

The problem this work tackles is that the piece-wise integration of the latent process between observations can fail to retain a global representation of the time series. Specifically, each change to the hidden state of the dynamical system from a new observation can result in a loss of memory about prior dynamical states the model was originally in. This pathology limits the utility of neural ODEs when there is a necessity to retain information about the recent and distant past; *i.e.* current neural

ODE formulations are *amnesic*. We illustrate this effect in Figure 1, where we see that backward integration of a learned neural ODE (that is competent at forecasting) quickly diverges, indicating the state only retains sufficient local information about the future dynamics.

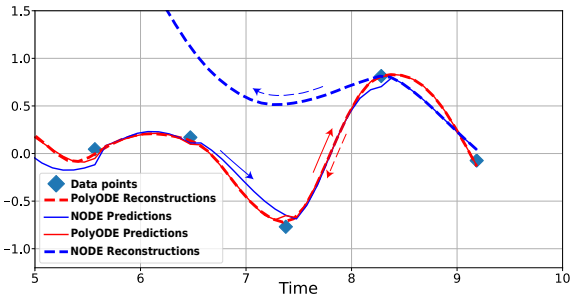

Figure 1: **PolyODE:** Illustration of the ability of PolyODE to reconstruct past trajectories. The solid lines show the forecasting trajectories conditioned on past observations for NODE (blue) and PolyODE (red). The dotted line represents the backward reconstruction for the past trajectories conditioned on the latent process at the last observation. We observe that PolyODE is able to accurately reconstruct the past trajectories while NODE quickly diverges. PolyODE is also more accurate in terms of forecasting.

One strategy that has been explored in the past to address this pathology is to regularize the model to encourage it to capture long-range patterns by reconstructing the time series from the last observation, using an auto-encoder architecture (Rubanova et al., 2019). This class of approaches results in higher complexity and does not provide any guarantees on the retention of the history of a time series. In contrast, our work proposes an alternative parameterization of the dynamics function that, *by design*, captures long-range memory within a neural ODE. Inspired by the recent successes of the HiPPO framework (Gu et al., 2020), we achieve this by enforcing that the dynamics of the hidden process follow the dynamics of the projection of the observed temporal process onto a basis of orthogonal polynomials. The resulting model, *PolyODE*, is a new neural ODE architecture that encodes long-range past information in the latent process and is thus *anamnesic*. As depicted in Figure 1, the resulting time series embeddings are able to reconstruct the past time series with significantly better accuracy.

Contributions (1) We propose a novel dynamics function for a neural ODE resulting in PolyODE, a model that learns a global representation of high-dimensional time series and is capable of long-term forecasting and reconstruction by design. PolyODE is the first investigation of the potential of the HiPPO operator for neural ODEs architectures.

(2) Methodologically, we highlight the practical challenges in learning PolyODE and show how adaptive solvers for ODEs can overcome them. Theoretically, we provide bounds characterizing the quality of reconstruction of time series when using PolyODE.

(3) Empirically, we study the performance of our approach by assessing the ability of the learnt embeddings to reconstruct the past of the time series and by studying their utility as inputs for downstream predictive tasks. We show that our model provides better time series representations, relative to several existing neural ODEs architectures, based on the ability of the representations to accurately make predictions on several downstream tasks based on chaotic time series and irregularly sampled data from patients in intensive care unit.

2 Related work

Time series modelling in machine learning: There is vast literature on the use of machine learning for time series modelling and we highlight some of the ideas that have been explored to adapt diverse kinds of models for irregular time series data. Although not naturally well suited to learning representations of such data, there have been modifications proposed to discrete-time models such as recurrent neural networks (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) to handle such data. Models such as mTANs (Shukla and Marlin, 2021) leverage an attention-based approach to interpolate sequences to create discrete-time data from irregularly sampled data. Another strategy has been architectural modifications to the recurrence equations e.g. CT-GRU (Mozer et al., 2017), GRU-D (Che et al., 2018) and Unitary RNNs (Arjovsky et al., 2016). Much more closely aligned to our work, and a natural fit for irregularly sampled data is research that uses differential equations to model continuous-time processes (Chen et al., 2018). By parameterizing the derivative of a time series using neural networks and integrating the dynamics over unobserved time points, this class of models is well suited to handle irregularly sampled data. This includes models such as ODE-RNN (Rubanova et al., 2019), ODE-LSTM (Lechner and Hasani, 2020) and Neural CDE (Kidger

et al., 2020). ODE-based approaches require the use of differential equation solvers during training and inference, which can come at the cost of runtime (Shukla and Marlin, 2021). PolyODEs lie in this family of models; specifically, this work proposes a new parameterization of the dynamics function and a practical method for learning that enables this model family to accurately forecast the future and reconstruct the past greatly enhancing the scope and utility of the learned embeddings.

Orthogonal polynomials: PolyODEs are inspired by a rich line of work in orthogonal decomposition of time series data. Orthogonal polynomials have been a mainstay in the toolkit for engineering (Heuberger et al., 2003) and uncertainty quantification (Li et al., 2011). In the context of machine learning, the limitations of RNNs to retain long-term memory have been studied empirically and theoretically (Zhao et al., 2020). Indeed, the GRU (Chung et al., 2014) and LSTM (Graves et al., 2007) architectures were created in part to improve the long-term memory of such models. Recent approaches for discrete-time models have used orthogonal polynomials and their ability to represent temporal processes in a memory-efficient manner. The Legendre Memory Unit (Voelker et al., 2019) and Fourier Recurrent Unit can be seen as a projection of data onto Legendre polynomials and Fourier basis respectively.

Our method builds upon and is inspired by the HiPPO framework which defines an operator to compute the coefficients of the projections on a basis of orthogonal polynomials. HiPPO-RNN and S4 are the most prominent examples of architectures building upon that framework (Gu et al., 2020; 2021). These models rely on a linear interpolation of the data in between observations, which can lead to a decrease of performance when the sampling rate of the input process is low. Furthermore, HiPPO-RNN and S4 perform the orthogonal polynomial projection of a non-invertible representation of the input data, which therefore doesn't enforce reconstruction in the observation space by design. Their design choices are motivated toward the goal of efficient mechanisms for capturing long term dependency for a target task (such as trajectory classification). In contrast, this work aims at exploring the abilities of the HiPPO operator for representation learning of irregular time series, when the downstream task is not known in advance.

Despite attempts to improve the *computational* performance of learning from long-term sequences (Morrill et al., 2021), to our knowledge, PolyODE is the first work that investigates the advantages of the HiPPO operator in the context of memory retention for continuous time architectures.

3 BACKGROUND

Orthogonal Polynomial Projections: Orthogonal polynomials are defined with respect to a measure μ as a sequence of polynomials $\{P_0(s), P_1(s), ...\}$ such that $deg(P_i) = i$ and

$$\langle P_n, P_m \rangle = \int P_n(s) P_m(s) d\mu(s) = \delta_{n=m} \alpha_n, \tag{1}$$

where α_n are normalizing scalars and δ is the Kronecker delta. For simplicity, we consider only absolutely continuous measures with respect to the Lebesgue measure, such that there exists a weight function $\omega(\cdot)$ such that $d\mu(s) = \omega(s)ds$. The measure μ determines the class of polynomials obtained from the conditions above (Eq. 1). Examples include Legendre, Hermite or Laguerre classes of orthogonal polynomials. The measure μ also defines an inner product $\langle \cdot, \cdot \rangle_\mu$ such that the orthogonal projection of a 1-dimensional continuous process $f(\cdot) : \mathbb{R} \to \mathbb{R}$ on the space of polynomials of degree N, $\mathcal{P}_N$, is given as

$$f_N(t) = \sum_{n=0}^{N} c_n P_n(t) \frac{1}{\alpha_n} \text{ with } c_n = \langle f, P_n \rangle_\mu = \int f(s) P_n(s) d\mu(s). \tag{2}$$

This projection minimizes the distance $\|f - p\|_\mu$ for all $p \in \mathcal{P}_N$ and is thus optimal with respect to the measure μ. One can thus encode a process f by storing its projection coefficients $\{c_0, ..., c_N\}$. We write the vector of coefficients up to degree N as $\mathbf{c}$ (the degree N is omitted) and $\mathbf{c}_i = c_i$. Intuitively, the measure assigns different weights at times of the process and thus allows for modulating the importance of different parts of the input signal for the reconstruction.

Continuous update of approximation coefficients: The projection of a process f onto a basis of orthogonal polynomials provides an optimal representation for reconstruction. However, there is often a need to update this representation continuously as new observations of the process f become available. Let $f_{<t}$ be the temporal process observed up until time t. We wish to compute

the coefficients of this process at different times t. We can define for this purpose a time-varying measure μ^t and corresponding weight function ω^t that can incorporate our requirements in terms of reconstruction abilities over time. For instance, if one cares about reconstruction of a process Δ temporal units in the past, one could use a time-varying weight function $\omega_t(s) = \mathbb{I}[s \in (t-\Delta, t)]$. This time-varying weight function induces a time-varying basis of orthogonal polynomials P_n^t for $n = 0, ..., N$. We can define the time-varying orthogonal projection and its coefficients $c_n(t)$ as

$$f_{<t} \approx f_{<t,N} = \sum_{n=0}^{N} c_n(t) P_n^t \frac{1}{\alpha_n^t} \text{ with } c_n(t) = \langle f_{<t}, P_n^t \rangle_{\mu^t} = \int f_{<t}(s) P_n^t(s) d\mu^t(s). \quad (3)$$

Dynamics of the projection coefficients: Computing the coefficients of the projection at each time step would be both computationally wasteful and would require storing the whole time series in memory, going against the principle of sequential updates to the model. Instead, we can leverage the fact that the coefficients evolve according to known linear dynamics over time. Remarkably, for a wide range of time-varying measures μ^t, Gu et al. (2020) show that the coefficients $\mathbf{c}_N(t)$ follow:

$$\begin{aligned} \frac{dc_n(t)}{dt} &= \frac{d}{dt} \int f_{<t}(s) P_n^t(s) d\mu^t(s), \quad \forall n \in \mathbb{N} \\ \frac{d\mathbf{c}(t)}{dt} &= A_\mu \mathbf{c}(t) + B_\mu f(t) \end{aligned} \quad (4)$$

where A_μ and B_μ are fixed matrices (for completeness, we provide a derivation of the relation for the translated Legendre measure in Appendix A). We use the translated Legendre measure in all our experiments. Using the dynamics of Eq. 4, it is possible to update the coefficients of the projection sequentially by only using the new incoming sample $f(t)$, while retaining the desired reconstruction abilities. Gu et al. (2020) use a discretization of the above dynamics to model discrete time-sequential data via a recurrent neural network architecture. Specifically, their architecture projects the hidden representation of an RNN onto a single time series that is projected onto an polynomial basis. Our approach differs in two ways. First, we work with a continuous time model. Second, we jointly model the evolution of d-dimensional time-varying process as a overparameterized hidden representation that uses orthogonal projections to serve as memory banks. The resulting model is a new neural ODE architecture as we detail below.

4 METHODOLOGY

Problem Setup. We consider a collection of sequences of temporal observations $\mathbf{x} = \{(\mathbf{x}_i, \mathbf{m}_i, t_i) : i \in \{1, ..., T\}\}$ that consist of a set of time-stamped observations and masks ($\mathbf{x}_i \in \mathbb{R}^d, \mathbf{m}_i \in \mathbb{R}^d, t_i \in \mathbb{R}$). We write $\mathbf{x}_{i,j}$ and $\mathbf{m}_{i,j}$ for the value of the j^{th} dimension of $\mathbf{x}_i$ and $\mathbf{m}_i$ respectively. The mask $\mathbf{m}_i$ encodes the presence of each dimension at a specific time point. We set $\mathbf{m}_{i,j} = 1$ if $\mathbf{x}_{i,j}$ is observed and $\mathbf{m}_{i,j} = 0$ otherwise. The number of observations for each sequence $\mathbf{x}$, T, can vary across sequences. We define the set of sequences as $\mathcal{S}$ and the distance between two time series observed at the same times as $d(\mathbf{x}, \mathbf{x}') = \frac{1}{T}\sum_i^T \|\mathbf{x}_i - \mathbf{x}_i'\|_2$. Our goal is to be able to embed a sequence $\mathbf{x}$ into a vector $\mathbf{h} \in \mathbb{R}^{d_h}$ such that (1) $\mathbf{h}$ retains a maximal amount of information contained in $\mathbf{x}$ and (2) $\mathbf{h}$ is informative for downstream prediction tasks. We formalize both objectives below.

Definition (Reverse reconstruction)**.** *Given an embedding $\mathbf{h}_t$ of a time series $\mathbf{x}$ at time t, we define the reverse reconstruction $\hat{\mathbf{x}}_{<t}$ as the predicted values of the time series at times prior to t. We write the observed time series prior to t as $\mathbf{x}_{<t}$.*

Objective 1 (Long memory representation)**.** *Let $\mathbf{h}_t$ and $\mathbf{h}_t'$ be two embeddings of the same time series $\mathbf{x}$. Let $\hat{\mathbf{x}}_{<t}$ and $\hat{\mathbf{x}}'_{<t}$ be their reverse reconstruction. We say that $\mathbf{h}_t$ enjoys more memory than $\mathbf{h}_t'$ if $d(\hat{\mathbf{x}}_{<t}, \mathbf{x}_{<t}) < d(\hat{\mathbf{x}}'_{<t}, \mathbf{x}_{<t})$.*

Objective 2 (Downstream task performance)**.** *Let $\mathbf{y} \in \mathbb{R}^{d_y}$ be an auxiliary vector drawn from a unknown distribution depending on $\mathbf{x}$. Let $\hat{\mathbf{y}}(\mathbf{x})$ and $\hat{\mathbf{y}}(\mathbf{x})'$ be the predictions obtained from embeddings $\mathbf{h}_t$ and $\mathbf{h}_t'$. For a performance metric $\alpha : \mathcal{S} \times \mathbb{R}^{d_y} \to \mathbb{R}$, we say that $\mathbf{h}_t$ is more informative than $\mathbf{h}_t'$ if $\mathbb{E}_{\mathbf{x},\mathbf{y}}[\alpha(\hat{\mathbf{y}}(\mathbf{x}), \mathbf{y})] > \mathbb{E}_{\mathbf{x},\mathbf{y}}[\alpha(\hat{\mathbf{y}}(\mathbf{x})', \mathbf{y})]$.*

4.1 POLYODE: ANAMNESIC NEURAL ODES

We make the assumption that the observed time series $\mathbf{x}$ comes from an unknown but continuous temporal process $\mathbf{x}(t)$. Given $\mathbf{h}(t) \in \mathbb{R}^{d_h}$ and a read-out function $g : \mathbb{R}^{d_h} \to \mathbb{R}^d$ we posit the

following generative process for the data:

$$\mathbf{x}(t) = g(\mathbf{h}(t)), \qquad \frac{d\mathbf{h}(t)}{dt} = \phi(\mathbf{h}(t)) \tag{5}$$

where part of $\phi(\cdot)$ is parametrized via a neural network $\phi_\theta(\cdot)$.

The augmentation of the state space is a known technique to improve the expressivity of Neural ODEs (Dupont et al., 2019). Here, to ensure that the hidden representation in our model has the capacity to retain long-term memory, we augment the state space of our model by including the dynamics of coefficients of orthogonal polynomials as described in Equation 4.

Similarly as classical filtering architectures (e.g. Kalman filters and ODE-RNN (Rubanova et al., 2019)), PolyODE alternates between two regimes : an integration step (that takes place in between observations) and an update step (that takes place at the times of observations), described below.

We structure the hidden state as $\mathbf{h}(t) = [\mathbf{h}_0(t), \mathbf{h}_1(t), \dots, \mathbf{h}_d(t)]$ where $\mathbf{h}_0(t) \in \mathbb{R}^d$ has the same dimension as the input process $\mathbf{x}$, $\mathbf{h}_i(t) \in \mathbb{R}^N, \forall i \in 1, \dots, d$, has the same dimension as the vector of projection coefficients $\mathbf{c}^i(t)$ and $[\cdot, \cdot]$ is the concatenation operator. We define the readout function $g_i(\cdot) : \mathbb{R}^{(N+1)d} \to \mathbb{R}$ such that $g_i(\mathbf{h}(t)) = \mathbf{h}_0(t)_i$. That is, g_i is fixed and returns the i^{th} value of the input vector. This leads to the following system of ODEs that characterize the evolution of $\mathbf{h}(t)$:

Integration Step.

$$\begin{cases} \frac{d\mathbf{c}^1(t)}{dt} & = A_\mu \mathbf{c}^1(t) + B_\mu g_1(\mathbf{h}(t)) \\ & \vdots \\ \frac{d\mathbf{c}^d(t)}{dt} & = A_\mu \mathbf{c}^d(t) + B_\mu g_d(\mathbf{h}(t)) \\ \frac{d\mathbf{h}(t)}{dt} & = \phi_\theta(\mathbf{h}(\mathbf{t})) \end{cases} \tag{6}$$

This parametrization allows learning arbitrarily complex dynamics for the temporal process $\mathbf{x}$. We define a sub-system of equations of projection coefficients update for each dimension of the input temporal process $\mathbf{x}(t) \in \mathbb{R}^d$. This sub-system is equivalent to Equation 4, where we have substituted the input process by the prediction from the hidden process $\mathbf{h}(t)$ through a mapping $g_i(\cdot)$. The hidden process $\mathbf{h}_0(t)$ acts similarly as in a classical Neural ODEs and the processes $\mathbf{c}(t)$ captures long-range information about the observed time series. During the integration step, we integrate both the hidden process $\mathbf{h}(t)$ and the coefficients $\mathbf{c}(t)$ forward in time, using the system of Equation 6. At each time step, we can provide an estimate of the time series $\hat{\mathbf{x}}(t)$ conditioned on the hidden process $\mathbf{h}(t)$, with $\hat{\mathbf{x}}(t) = g(\mathbf{h}(t))$.

The coefficients $\mathbf{c}(t)$ are influenced by the values of $\mathbf{h}(t)$ through $\mathbf{h}_0(t)$ only. The process $\mathbf{h}_0(t)$ provides the signal that will be memorized by projecting onto the orthogonal polynomial basis. The $\mathbf{c}(t)$ serve as memory banks and do not influence the dynamics of $\mathbf{h}(t)$ during the integration step.

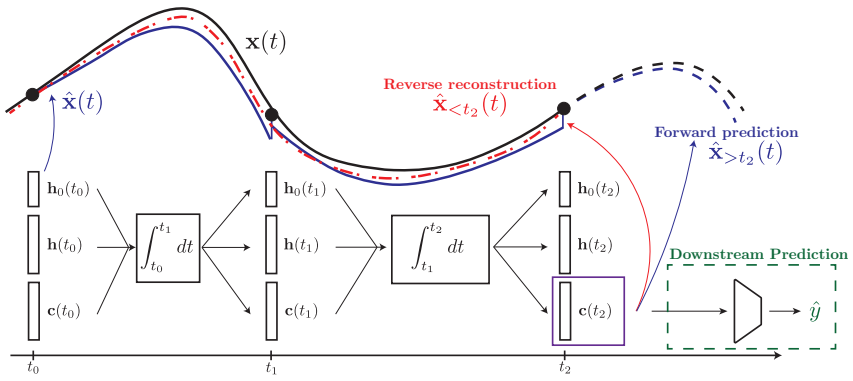

Figure 2: **PolyODE time series embedding process**. The model processes the time series sequentially by alternating between integration steps (between observations) and update steps when observations are collected. Informative embeddings should allow for (1) reconstructing the past of the time series (reverse reconstruction - in red), (2) forecasting the future of the sequence (forward prediction - in blue) and (3) being informative for downstream predictions (in green).

The system of equations in Eq. 6 characterises the dynamics in between observations. When a new observation becomes available, we update the system as follows.

Update Step. *At time $t = t_i$, after observing $\mathbf{x}_i$ and mask $\mathbf{m}_i$, we set*

$$\begin{cases} \mathbf{h}_j(t_i) := \mathbf{c}^j(t_i), \ \ \forall j \textit{ s.t. } \mathbf{m}_{i,j} = 1 \\ \mathbf{h}_0(t_i)_j := \mathbf{x}_{i,j}, \ \ \forall j \textit{ s.t. } \mathbf{m}_{i,j} = 1 \end{cases} \tag{7}$$

The update step serves the role of incorporating new observations in the hidden representation of the system. It proceeds by (1) reinitializing the hidden states of the system with the orthogonal polynomial projection coefficients $\mathbf{c}(t)$: $\mathbf{h}_j(t_i) := \mathbf{c}^j(t_i)$; and (2) resetting $\mathbf{h}_0(t)$ to the newly collected observation: $\mathbf{h}_0(t_i)_j := \mathbf{x}_{i,j}(t)$.

Remarks: Our model blends orthogonal polynomials with the flexibility offered in modelling the observations with NeuralODEs. The consequence of this is that while the coefficients serve as memory banks for each dimension of the time series, the Neural ODE over $\mathbf{h}_0(t)$ can be used to forecast from the model. That said, we acknowledge that a significant limitation of our current design is the need for the hidden dimension to track N coefficients for each time-series dimension. Given that many adjacent time series might be correlated, we anticipate that methods to reduce the space footprint of the coefficients within our model is fertile ground for future work.

4.2 Training

We train this architecture by minimizing the reconstruction error between the predictions and the observations: $\mathcal{L} = \sum_{i=1}^{T} \|\hat{\mathbf{x}}(t_i) - \mathbf{x}_i\|_2^2$. We first initialize the hidden processes $\mathbf{c}(0) = 0$ and $\mathbf{h}(0) = 0$ though they can be initialized with static information b, if available (*e.g.* $\mathbf{h}(0) = \psi_\theta(b)$). We subsequently alternate between integration steps between observations and update steps at observation times. The loss is updated at each observation time t_i. A pseudo-code description of the overall procedure is given in Algorithm 1.

Numerical integration. We integrate the system of differential equations of Equation 6 using differentiable numerical solvers as introduced in Chen et al. (2018). However, one of the technical challenges that arise with learning PolyODE is that the dynamical system in Equation 6 is relatively stiff and integrating this process with acceptable precision would lead to prohibitive computation times with explicit solvers. To deal with this instability we used an implicit solver such as Backward Euler or Adams-Moulton for the numerical integration (Sauer, 2011). A comparison of numerical integration schemes and an analysis of the stability of the ODE are available in Appendix I.

Algorithm 1: PolyODE Training

Data: $\mathbf{x}$, matrices A_μ, B_μ, number of dimensions d, number of observations T,
number of polynomial coefficients N
Result: Training loss $\mathcal{L}$ over a whole sequence $\mathbf{x}$
$t^* \leftarrow 0$
Initialize $\mathbf{h}_j(0) = \mathbf{c}_j(0) = \mathbf{0}_N, \forall j \in 1, ..., d$,
Loss $\mathcal{L} = 0$
for $i \leftarrow 1$ **to** T **do**
  Integrate $\mathbf{c}_{1,...,d}(t)$ and $\mathbf{h}_{0,..,d}(t)$ from $t = t^*$ until $t = t_i$
  $\hat{\mathbf{x}}_i \leftarrow \mathbf{h}_0(t^*)$
  Update $\mathbf{c}_{1,...,d}(t_i)$ and $\mathbf{h}_{0,...,d}(t_i)$ with $\mathbf{x}_i, \mathbf{m}_i$.
  $\mathcal{L} = \mathcal{L} + \|(\hat{\mathbf{x}}_i - \mathbf{x}_i) \odot \mathbf{m}_i\|_2^2$
  $t^* \leftarrow t_i$
end

Forecasting: From time t, we forecast the time series at an arbitrary time t^* as:

$$\hat{\mathbf{x}}_{>t}(t^*) = g(\mathbf{h}(t) + \int_t^{t^*} \phi_\theta(\mathbf{h}(s))ds), \tag{8}$$

where $\phi_\theta(\cdot)$ is the learned model that we use in the integration step and introduced in Eq. 5.

Reverse Reconstruction: Using Equation 3, we can compute the reverse reconstruction of the time series at any time t using the projection coefficients part of the hidden process:

$$\hat{\mathbf{x}}_{<t,j} = \sum_{n=0}^{N} c_n^j(t) \cdot P_n^t \cdot \frac{1}{\alpha_n^t}. \tag{9}$$

More details about this reconstruction process and its difference with respect to classical NODEs are available in Appendix E. The error between the prediction obtained during the integration step, $\hat{\mathbf{x}}(t)$, and the above reconstruction estimator is bounded above, as Result 4.1 shows.

Result 4.1. *For a shifted rectangular weighting function with width Δ, $\omega^t(x) = \frac{1}{\Delta}\mathbb{I}_{[t-\Delta,t]}$ (which generate Legendre polynomials), the mean square error between the forward ($\hat{\mathbf{x}}$) and reverse prediction ($\hat{\mathbf{x}}_{<t}$) at each time t is bounded by:*

$$\|\hat{\mathbf{x}} - \hat{\mathbf{x}}_{<t}\|_{\mu^t}^2 \leq C_0 \frac{\Delta^2 L^2 (K+1)^2}{N(2N-1)} + C_1 \Delta L (K+1) S_K \xi\left(\frac{3}{2}, N\right) + C_2 S_K^2 \xi\left(\frac{3}{2}, N\right),$$

where K is the number of observations in the interval $[t-\Delta, t]$, L is the Lipschitz constant of the forward process, N is the degree of the polynomial approximation and $\xi(\cdot,\cdot)$ is the Hurwitz zeta function. $S_K = \sum_{i=1}^{K} |\hat{\mathbf{x}} - \mathbf{x_i}|$ is the sum of absolute errors between the forward process and observations incurred at the update steps. C_0, C_1 and C_2 are constants.

Expectedly, the bound goes to 0 as the degree of the approximation increases. The lower cumulative absolute error S_K also leads to a reduction of this bound. As the cumulative absolute error S_K and our loss function $\mathcal{L}$ share the same optimum, for fixed Δ, L, K and N, our training objective therefore implicitly enforces a minimization of the reconstruction error. This corresponds to optimizing Objective 1, where we set $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_{\mu^t}^2$. Our architecture thus jointly minimizes both global reconstruction and forecasting error. Notably, when $S_K = 0$, this result boils down to the well-known projection error for orthogonal polynomials projection of continuous processes (Canuto and Quarteroni, 1982). What is more, increasing the width of the weighting function (increasing Δ) predictably results in higher reconstruction error. However, this can be compensated by increasing the dimension of the polynomial basis accordingly. We also note a quadratic dependency on the Lipschitz constant of the temporal process, which can limit the reverse reconstruction abilities for high-frequency components. The full proof can be found in Appendix B.

5 EXPERIMENTS

We evaluate our approach on two objectives : (1) the ability of the learned embedding to encode global information about the time series, through the reverse reconstruction performance (or memorization) and (2) the ability of embedding to provide an informative input for a downstream task. We study our methods on the following datasets:

Synthetic Univariate. We validate our approach using a univariate synthetic time series. We simulate 1000 realizations from this process and sample it at irregularly spaced time points using a Poisson point process. For each generated irregularly sampled time series $\mathbf{x}$, we create a binary label $y = \mathbb{I}[x(5) > 0.5]$. Further details about datasets are to be found in Appendix G.

Chaotic Attractors. Chaotic dynamical systems exhibit a large dependence of the dynamics on the initial conditions. This means that a noisy or incomplete evaluation of the state space may not contain much information about the past of the time series. We consider two widely used chaotic dynamical systems: Lorenz63 and a 5-dimensional Lorenz96. We generate 1000 irregularly sampled time series from different initial conditions. We completely remove one dimension of the time series such that the state space is never fully observed. This forces the model to remember the past trajectories to create an accurate estimate of the state space at each time t.

MIMIC-IIII dataset. We use a pre-processed version of the MIMIC-III dataset (Johnson et al., 2016; Wang et al., 2020). This consists of the first 24 hours of follow-up for ICU patients. For each time series, the label y is the in-hospital mortality.

Baselines: We compare our approach against two sets of baselines: Neural ODEs architecture and variants of recurrent neural networks architectures designed for long-term memory. To ensure a fair comparison, we use the same dimensionality of the hidden state for all models.

Neural ODE baselines. We use a filtering implementation of Neural ODEs, *GRU-ODE-Bayes* (De Brouwer et al., 2019) and *ODE-RNN* (Rubanova et al., 2019), an auto-encoder relying on a Neural ODE for both the encoder and the decoder part. For theses baselines, we compute the reverse reconstruction by integrating the system of learnt ODEs backward in time. In case of ODE-RNN, we use the ODE of the decoder. Additionally, we compare against Neural RDE neural controlled differential equations for long time series (Neural RDE) (Morrill et al., 2021).

Long-term memory RNN baselines. We compare against *HiPPO-RNN* (Gu et al., 2020), a recurrent neural network architecture that uses orthogonal polynomial projections of the hidden process. We also use a variant of this approach where we directly use the HiPPO operator on the observed time series, rather than on the hidden process. We call this variant *HiPPO-obs*. We also compare against S4, an efficient state space model relying on the HiPPO matrix (Gu et al., 2021).

Long-range representation learning: For each dataset, we evaluate our method and the various baselines on different tasks. Implementation details are available in Appendix H.

Downstream Classification. We train the models on the available time series. After training, we extract time series embedding from each model and use them as input to a multi-layer perceptron trained to predict the time series label y. We report the area under the operator-characteristic curve evaluated on a left-out test set with 5 repetitions.

Time Series Reconstruction. Similarly as for the downstream classification, we extract the time series embeddings from models trained on the time series. We then compute the reverse reconstruction $\hat{\mathbf{x}}_{<t}$ and evaluate the MSE with respect to the true time series.

Forecasting. We compare the ability of all models to forecast the future of the time series. We compute the embedding of the time series observed until some time t_{cond} and predict over a horizon t_{horizon}. We then report the MSE between the prediction and true trajectories.

Table 1: Downstream task and reverse reconstruction results for synthetic and Lorenz datasets.

Model	Downstream Classification↑			Reconstruction↓		
	Synthetic	Lorenz63	Lorenz96	Synthetic	Lorenz63	Lorenz96
Irregular Rate λ	0.7	0.3	0.3	0.7	0.3	0.3
GRU-ODE	0.968 ± 0.004	0.825 ± 0.031	0.925 ± 0.004	0.057 ± 0.010	0.752 ± 0.057	0.346 ± 0.072
ODE-RNN	0.870 ± 0.032	0.813 ± 0.013	0.954 ± 0.012	0.080 ± 0.036	0.674 ± 0.049	0.214 ± 0.030
Neural-RDE	0.773 ± 0.111	0.604 ± 0.046	0.606 ± 0.112	0.167 ± 0.031	0.989 ± 0.074	1.747 ± 0.472
HiPPO-obs	0.758 ± 0.023	0.837 ± 0.034	0.949 ± 0.007	0.197 ± 0.010	0.511 ± 0.043	0.247 ± 0.005
HiPPO-RNN	0.742 ± 0.008	0.804 ± 0.023	0.944 ± 0.008	0.209 ± 0.018	0.784 ± 0.122	0.198 ± 0.014
S4	$\mathbf{0.994 \pm 0.003}$	0.911 ± 0.005	0.948 ± 0.016	0.032 ± 0.006	0.428 ± 0.040	0.171 ± 0.008
PolyODE	$\mathbf{0.994 \pm 0.003}$	$\mathbf{0.992 \pm 0.000}$	$\mathbf{0.984 \pm 0.002}$	$\mathbf{0.012 \pm 0.002}$	$\mathbf{0.034 \pm 0.008}$	$\mathbf{0.038 \pm 0.008}$

Results for these tasks are presented in Table 1 for Synthetic and Lorenz datasets and in Table 2 for MIMIC. We report additional results in Appendix C, with a larger array of irregular sampling rates. We observe that the reconstruction abilities of PolyODE clearly outperforms the other baselines, for all datasets under consideration. A similar trend is to be noted for the downstream classification for the synthetic and Lorenz datasets. For these datasets, accurate prediction of the label y requires a global representation of the time series, which results in better performance for our approach.

Table 2: Performance on MIMIC-III dataset.

Method	Classification ↑	Forecasting ↓	Reconstruction ↓
HiPPO-obs	0.793 ± 0.002	/	0.775 ± 0.000
HiPPO-RNN	0.764 ± 0.006	1.104 ± 0.009	0.969 ± 0.026
GRU-ODE	0.793 ± 0.005	1.413 ± 0.074	2025.6 ± 2365.1
ODE-RNN	$\mathbf{0.800 \pm 0.004}$	1.104 ± 0.026	6.343 ± 4.844
PolyODE	0.778 ± 0.005	$\mathbf{1.085 \pm 0.022}$	$\mathbf{0.187 \pm 0.005}$

For the MIMIC dataset, our approach compares favourably with the other methods for the downstream classification objective and outperforms other methods for trajectory forecasting. What is more, the reconstruction ability of PolyODE is significantly better than compared approaches. In Figure 3, we plot the reverse reconstructions of PolyODE for several vitals of a random patient over the first 24 hours in the ICU. This reconstruction is obtained by first sequentially processing the time series until $t = 24$ hours and subsequently using the hidden process to reconstruct the time series as in Equation 9. We observe that PolyODE can indeed capture the overall trend of the time series over the whole history.

Ablation study - the importance of the auxiliary dynamical system: Is there utility in leveraging the neural network $\phi_\theta(\cdot)$ to learn the dynamics of the time series? How well would various interpolation

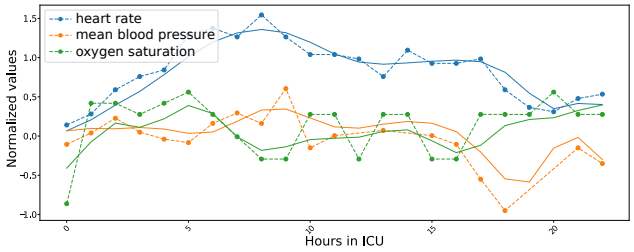

Figure 3: **PolyODE**: Reverse prediction of vitals over the 24 hours of ICU of a randomly selected test patient. We plot the true value (dots) and reconstructions (solid line) for different vitals. Reverse reconstruction is done from the last time observation. Other vitals are provided in Appendix D.

schemes for irregularly sampled observations perform in the context of reverse reconstruction and classification? In response to these questions, we first note that they do not support extrapolation and are thus incapable of forecasting the future of the time series. However, we compare the performance in terms of reverse reconstruction and classification in Table 3. We consider constant interpolation (last observation carried forward), linear interpolation and Hermite spline interpolation. Our results indicate a significant gap in performance between PolyODE and the linear and constant interpolation schemes. The Hermite spline interpolation allows us to capture most of the signal needed for the downstream classification task but results in significantly lower performance in terms of the reverse reconstruction error. These results therefore strongly support the importance of $\phi_\theta(\cdot)$ for producing informative time series embeddings. Complementary results are available in Appendix C.

Table 3: Impact of the interpolation scheme on performance.

	Downstream Classification↑			Reconstruction↓		
	SimpleTraj	Lorenz	Lorenz96	SimpleTraj	Lorenz	Lorenz96
Irregular Rate λ	0.7	0.3	0.3	0.7	0.3	0.3
Constant	0.969 ± 0.005	0.664 ± 0.033	0.862 ± 0.017	0.027 ± 0.003	0.785 ± 0.074	0.393 ± 0.017
Linear	0.969 ± 0.008	0.744 ± 0.016	0.857 ± 0.026	0.028 ± 0.005	0.787 ± 0.066	0.388 ± 0.032
Hermite Spline	0.971 ± 0.012	0.976 ± 0.000	$\mathbf{0.983 \pm 0.004}$	0.055 ± 0.016	0.135 ± 0.007	0.093 ± 0.011
PolyODE	$\mathbf{0.994 \pm 0.003}$	$\mathbf{0.992 \pm 0.000}$	$\mathbf{0.984 \pm 0.002}$	$\mathbf{0.012 \pm 0.002}$	$\mathbf{0.034 \pm 0.008}$	$\mathbf{0.038 \pm 0.008}$

Incorporating global time series uncertainty: Previous experiments demonstrate the ability of PolyODE to retain memory of the past trajectory. A similar capability can be obtained for capturing global model uncertainties over the time series history. In Figure 4, we evaluate the association between the recovered uncertainties of PolyODE and the reverse reconstruction errors. We plot the predicted uncertainties against the root mean square error (RMSE) on a logarithmic scale. We compare our approach with using the uncertainty of the model at the last time step only. We observe that the uncertainties recovered by PolyODE are significantly more correlated with the errors (Pearson-$\rho = 0.56$) compared to using the uncertainties obtained from the last time step (Pearson-$\rho = 0.11$). More details are available in Appendix F.

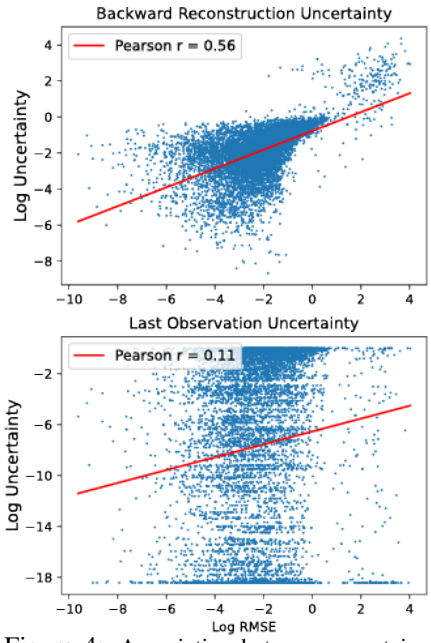

Figure 4: Association between uncertainties and reverse reconstruction errors for PolyODE (top) and classical Neural ODEs (bottom).

6 Conclusion

Producing time series representations that are easy to manipulate, representative of global dynamics, practically useful for downstream tasks and robust to irregular sampling remains an ongoing challenge. In this work, we took a step in that direction by proposing a simple but novel architecture that satisfies those requirements by design. As a Neural ODE, PolyODE inherits the ability to handle irregular time series elegantly but at the same time, PolyODE also incurs computational cost associated with numerical integration. Currently, our approach also requires a large hidden space dimension and finding methods to address this that exploit the correlation between dimensions of the time series is a fruitful direction for future work.

Reproducibility Statement Details for reproducing experiments shown are available in Appendix H. The code for reproducing all experiments will be made publicly available.

Acknowledgements

EDB is funded by a FWO-SB PhD research grant (S98819N) and a FWO research mobility grant (V424722N). RGK was supported by a CIFAR AI Chair. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute.

REFERENCES

M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

C. Canuto and A. Quarteroni. Approximation results for orthogonal polynomials in sobolev spaces. *Mathematics of Computation*, 38(157):67–86, 1982.

Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1), 2018.

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

E. De Brouwer, J. Simm, A. Arany, and Y. Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. *Advances in neural information processing systems*, 32, 2019.

E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. *Advances in Neural Information Processing Systems*, 32, 2019.

J.-Y. Franceschi, A. Dieuleveut, and M. Jaggi. Unsupervised scalable representation learning for multivariate time series. *Advances in neural information processing systems*, 32, 2019.

A. Graves, S. Fernández, and J. Schmidhuber. Multi-dimensional recurrent neural networks. In *International conference on artificial neural networks*, pages 549–558. Springer, 2007.

A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems*, 33:1474–1487, 2020.

A. Gu, K. Goel, and C. Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

P. S. Heuberger, T. J. de Hoog, P. M. Van den Hof, and B. Wahlberg. Orthonormal basis functions in time and frequency domain: Hambo transform theory. *SIAM Journal on Control and Optimization*, 42(4):1347–1373, 2003.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

P. Kidger, J. Morrill, J. Foster, and T. Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

M. Lechner and R. Hasani. Learning long-term dependencies in irregularly-sampled time series. *arXiv preprint arXiv:2006.04418*, 2020.

Y. Li, M. Anitescu, O. Roderick, and F. Hickernell. Orthogonal bases for polynomial regression with derivative information in uncertainty quantification. *Visualization of Mechanical Processes: An International Online Journal*, 1(4), 2011.

G. Lohöfer. Inequalities for the associated legendre functions. *Journal of Approximation Theory*, 95 (2):178–193, 1998.

P. Malhotra, V. TV, L. Vig, P. Agarwal, and G. Shroff. Timenet: Pre-trained deep recurrent neural network for time series classification. *arXiv preprint arXiv:1706.08838*, 2017.

J. Morrill, C. Salvi, P. Kidger, and J. Foster. Neural rough differential equations for long time series. In *International Conference on Machine Learning*, pages 7829–7838. PMLR, 2021.

M. C. Mozer, D. Kazakov, and R. V. Lindsey. Discrete event, continuous time RNNs. *arXiv preprint arXiv:1710.04110*, 2017.

R. K. Nagle, E. B. Saff, and A. D. Snider. *Fundamentals of differential equations and boundary value problems*. Pearson education, 2011.

D. Onken, S. W. Fung, X. Li, and L. Ruthotto. Ot-flow: Fast and accurate continuous normalizing flows via optimal transport. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9223–9232, 2021.

Y. Rubanova, R. T. Chen, and D. K. Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

T. Sauer. *Numerical analysis*. Addison-Wesley Publishing Company, 2011.

S. N. Shukla and B. Marlin. Multi-time attention networks for irregularly sampled time series. In *International Conference on Learning Representations*, 2021.

A. Voelker, I. Kajić, and C. Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems*, 32, 2019.

S. Wang, M. B. McDermott, G. Chauhan, M. Ghassemi, M. C. Hughes, and T. Naumann. Mimic-extract: A data extraction, preprocessing, and representation pipeline for mimic-iii. In *Proceedings of the ACM conference on health, inference, and learning*, pages 222–235, 2020.

J. Zhao, F. Huang, J. Lv, Y. Duan, Z. Qin, G. Li, and G. Tian. Do rnn and lstm have long memory? In *International Conference on Machine Learning*, pages 11365–11375. PMLR, 2020.

H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115, 2021.

Appendix

Table of Contents

A Derivation of the dynamics for Legendre polynomials(Gu et al., 2020)

For convenience, we repeat here the outline of the derivation in (Gu et al., 2020). We first recall the equations for the dynamics of the coefficients of the projection.

$$c_n(t) = \int f(x) p_n(x,t) \omega_t(x) dx$$

$$\frac{dc_n(t)}{dt} = \int f(x) \frac{\partial}{\partial t} p_n(x,t) \omega_t(x) dx \tag{10}$$

$$+ \int f(x) p_n(x,t) \frac{\partial}{\partial t} \omega_t(x) dx \tag{11}$$

where $p_n(x,t)$ corresponds to the orthonormal scaling of $P_n(x,t)$.

We consider a constant measure over a bounded interval of length Δ. Our time-varying weight function thus writes : $\omega(x,t) = \mathbb{I}_{[t-\Delta,t]}$. Intuitively, it enforces accurate reconstruction of the process $f(t)$ on the time interval $[t-\Delta, t]$ and disregards the more ancient history.

A constant measure over a bounded interval is reminiscent of the Legendre polynomials. Legendre polynomials satisfy

$$\int L_n(x)L_m(x)\frac{1}{\Delta}\mathbb{I}_{[-1,1]}dx = \frac{2}{2n+1}\delta_{n=m}$$

Using a straightforward change of variable, one can thus show that

$$\int l_n(x)l_m(x)\mathbb{I}_{[t-\Delta,t]}dx = \delta_{n=m} \text{ with } l_n(x,t) = (2n+1)^{\frac{1}{2}}L_n\left(\frac{2(x-t)}{\Delta}+1\right)$$

Furthermore, Legendre polynomials satisfy the following relation:

$$\begin{aligned}\frac{dL_n(x)}{dx} &= (2n-1)L_{n-1}(x) + (2n-5)L_{n-3}(x) + ... \\ &= \sum_{i=1,\text{odd}}^{n}(2(n-i)+1)L_{n-i}(x)\end{aligned}$$

We also have $L_n(1) = 1$ and $L_n(-1) = (-1)^n$.

We can use these properties to compute the integrals in Equations 10 and 11.

$$\begin{aligned}\int f(x)\frac{\partial}{\partial t}l_n(x,t)\omega(x,t)dx &= \sum_{i=1,\text{odd}}^{n}(2(n-i)+1)^{\frac{1}{2}}(2n+1)^{\frac{1}{2}}\cdot\frac{-2}{\Delta}\int f(x)l_{n-i}(x,t)\omega(x,t)dx \\ &= \sum_{i=1,\text{odd}}^{n}(2(n-i)+1)^{\frac{1}{2}}(2n+1)^{\frac{1}{2}}\cdot\frac{-2}{\Delta}\cdot c_{n-i}(t)\end{aligned}$$

$$\begin{aligned}\int f(x)l_n(x,t)\frac{\partial}{\partial t}\frac{1}{\Delta}\mathbb{I}_{[t-\Delta,t]}(x)dx &= \frac{1}{\Delta}(f(t)l_n(t,t) - f(t-\Delta)l_n(t-\Delta,t)) \\ &= \frac{1}{\Delta}(f(t)(2n+1)^{\frac{1}{2}} - f(t-\Delta)(2n+1)^{\frac{1}{2}}(-1)^n)\end{aligned}$$

Remarkably, if we approximate $f(t-\Delta)$ by its projection, $f(t-\Delta) \approx \sum_{n=0}^{N} c_n(t)l_n(t-\Delta,t)$, the equations above only depend linearly on $c_n(t)$ and the value of the process $f(t)$. After simplification, and writing $\mathbf{c}(t) \in \mathbb{R}^N$ the vector of all coefficients $c_n(t)$, this leads to the following ordinary differential equation.

$$\begin{aligned}\frac{d\mathbf{c}(t)}{dt} &= -\frac{1}{\Delta}A\mathbf{c}(t) + \frac{1}{\Delta}Bf(t) \\ A_{n,m} &= (2n+1)^{\frac{1}{2}}(2m+1)^{\frac{1}{2}}\begin{cases}1 \text{ if } m \leq n \\ (-1)^{n-m} \text{ if } m \leq n\end{cases} \\ B_n &= (2n+1)^{\frac{1}{2}}\end{aligned} \tag{12}$$

B RECONSTRUCTION ERROR OF PIECE-WISE CONTINUOUS FUNCTIONS

Again, we use the following weight function: $\omega_t(x) = \frac{1}{\Delta}\mathbb{I}_{[t-\Delta,t]}$, corresponding to the measure $d\mu_t(x) = \omega_t(x)dx$ that generates the Legendre sequence of polynomials P_n^t. We define p_n^t as the normalized version of the Legendre polynomials.

We want to bound $|| \mathbf{x}_{<t} - \hat{\mathbf{x}}_{<t} ||^2_{\mu_t}$.

The legendre polynomials form a complete orthogonal basis for the space of integrable functions with weight ω, L^2_ω. This means that $\mathbf{x}_{<t}(t) \in L^2_\omega$ can be written as:

$$\mathbf{x}_{<t} = \sum_{k=1}^{\infty} \langle \mathbf{x}_{<t}, p_k^t \rangle_{\mu_t} p_k^t \tag{13}$$

The reverse reconstruction is

$$\hat{\mathbf{x}}_{<t} = \sum_{k=1}^{N} \langle \mathbf{x}_{<t}, p_k^t \rangle_{\mu_t} p_k^t \tag{14}$$

We then have

$$\begin{aligned}
|| \mathbf{x}_{<t} - \hat{\mathbf{x}}_{<t} ||^2_{\mu_t} =& || \sum_{k=1}^{\infty} \langle \mathbf{x}_{<t}, p_k^t \rangle_{\mu_t} p_k^t - \sum_{k=1}^{N} \langle \mathbf{x}_{<t}, p_k^t \rangle_{\mu_t} p_k^t ||^2_{\mu_t} && (15)\\
=& || \sum_{k=N+1}^{\infty} \langle \mathbf{x}_{<t}, p_k^t \rangle_{\mu_t} p_k^t ||^2_{\mu_t} && (16)\\
=& \langle \sum_{k=N+1}^{\infty} \langle \mathbf{x}_{<t}, p_k^t \rangle_{\mu_t} p_k^t, \sum_{k=N+1}^{\infty} \langle \mathbf{x}_{<t}, p_k^t \rangle_{\mu_t} p_k^t \rangle_{\mu_t} && (17)\\
=& \langle \sum_{k=N+1}^{\infty} c_k^t p_k^t, \sum_{k=N+1}^{\infty} c_k^t p_k^t \rangle_{\mu_t} && (18)\\
=& \sum_{i=N+1}^{\infty} \Big(\sum_{j=N+1}^{\infty} \langle c_i^t p_i^t, c_j^t p_j^t \rangle \Big) && (19)\\
=& \sum_{i=N+1}^{\infty} \langle c_i^t p_i^t, c_i^t p_i^t \rangle && (20)\\
=& \sum_{i=N+1}^{\infty} (c_i^t)^2 \langle p_i^t, p_i^t \rangle && (21)\\
=& \sum_{i=N+1}^{\infty} (c_i^t)^2 && (22)\\
& && (23)
\end{aligned}$$

Our goal is thus to bound the series of the square of the projection coefficients. We first proceed by evaluating the expression for the coefficients.

$$c_n(t) = \langle x_{\leq t}, p_n(t) \rangle \tag{24}$$
$$= \int_{-\infty}^{t} x(s) p_n(t,s) \omega_t(s) ds \tag{25}$$
$$= \frac{1}{\Delta} \int_{t-\Delta}^{t} x(s) p_n(t,s) ds \tag{26}$$
$$= \frac{1}{\Delta} \int_{t-\Delta}^{t} x(s) (2n+1)^{\frac{1}{2}} P_n\left(\frac{2(s-t)}{\Delta} + 1\right) ds \tag{27}$$
$$= \frac{(2n+1)^{\frac{1}{2}}}{\Delta} \int_{t-\Delta}^{t} x(s) P_n\left(\frac{2(s-t)}{\Delta} + 1\right) ds \tag{28}$$
$$= \frac{(2n+1)^{\frac{1}{2}}}{2} \int_{-1}^{1} x\left(\frac{\Delta(y-1)}{2} + t\right) P_n(y) dy \quad \text{(change of variable } y = \frac{2(s-t)}{\Delta} + 1) \tag{29}$$

Where we use the definition of the normalized Legendre polynomials : $p_n(x) = P_n(x)(2n+1)^{\frac{1}{2}}$.

Now, we let $\Sigma = \cup_{i=1}^{K+1} \{a_i, b_i\}$ be the boundaries of the continuous intervals delimited by the irregular observations in y. That is, b_0 is the first discontinuity (the time of the first irregular observation in the rescaled time y, but occuring at $\frac{\Delta(b_0-1)}{2} + t$ in original time).

Because Legendre polynomials satisfy

$$P_n(s) = \frac{1}{2n+1} \frac{d}{ds} (P_{n+1}(s) - P_{n-1}(s)) \tag{30}$$

We can integrate by parts:

$$c_n(t) = \frac{(2n+1)^{\frac{1}{2}}}{2} \int_{-1}^{1} x(\frac{\Delta(y-1)}{2} + t) P_n(y) dy \tag{31}$$
$$= \frac{(2n+1)^{\frac{1}{2}}}{2} \int_{-1}^{1} x(\frac{\Delta(y-1)}{2} + t)(\frac{1}{2n+1} \frac{d}{dy} (P_{n+1}(y) - P_{n-1}(y))) dy \tag{32}$$
$$= -\frac{(2n+1)^{-\frac{1}{2}}}{2} \sum_{i=1}^{K+1} \int_{a_i}^{b_i} x'(\frac{\Delta(y-1)}{2} + t) \frac{\Delta}{2} ((P_{n+1}(y) - P_{n-1}(y))) dy \tag{33}$$
$$+ (\frac{(2n+1)^{-\frac{1}{2}}}{2} x(\frac{\Delta(y-1)}{2} + t)((P_{n+1}(y) - P_{n-1}(y))))\Big|_{-1}^{1} \tag{34}$$
$$+ \sum_{i=1}^{K} (\frac{(2n+1)^{-\frac{1}{2}}}{2} x(\frac{\Delta(y-1)}{2} + t)((P_{n+1}(y) - P_{n-1}(y))))\Big|_{b_i^-}^{b_i^+} \tag{35}$$

The second term above (Eq. 35) is trivially 0 since Legendre polynomials satisfy $P_{n+1}(1) = P_{n-1}(1) = 1$ and $P_{n+1}(-1) = P_{n-1}(-1) = (-1)^n$, so we have:

$$c_n(t) = -\frac{(2n+1)^{-\frac{1}{2}}}{2} \sum_{i=1}^{K+1} \int_{a_i}^{b_i} x'(\frac{\Delta(y-1)}{2} + t) \frac{\Delta}{2} ((P_{n+1}(y) - P_{n-1}(y))) dy \tag{36}$$
$$+ \sum_{i=1}^{K} \frac{(2n+1)^{-\frac{1}{2}}}{2} x(\frac{\Delta(y-1)}{2} + t)(P_{n+1}(y) - P_{n-1}(y))\Big|_{b_i^-}^{b_i^+} \tag{37}$$

We decompose the above expression and let

$$c_n(t) = A_n(t) + B_n(t) \tag{38}$$

with

$$A_n(t) = -\frac{(2n+1)^{-\frac{1}{2}}}{2} \sum_{i=1}^{K+1} \int_{a_i}^{b_i} x'(\frac{\Delta(y-1)}{2} + t)\frac{\Delta}{2}((P_{n+1}(y) - P_{n-1}(y)))dy \tag{39}$$

$$B_n(t) = \sum_{i=1}^{K} \frac{(2n+1)^{-\frac{1}{2}}}{2} x(\frac{\Delta(y-1)}{2} + t)(P_{n+1}(y) - P_{n-1}(y))\Big|_{b_i^-}^{b_i^+} \tag{40}$$

We then have,

$$c_n^2(t) \leq \mid A_n(t) \mid^2 + 2 \mid A_n(t) \mid\mid B_n(t) \mid + \mid B_n(t) \mid^2 \tag{41}$$

We first bound $\mid A_n(t) \mid^2$. Because we assume $x(t)$ is L-lispchitz, we have

$$\begin{aligned}
\mid A_n(t) \mid &\leq \frac{1}{2(2n+1)^{\frac{1}{2}}} \sum_{i=1}^{K+1} \mid \int_{a_i}^{b_i} \frac{\Delta}{2} x'(\frac{\Delta(y-1)}{2} + t)((P_{n+1}(y) - P_{n-1}(y)))dy \mid && (42)\\
&= \frac{L\Delta}{4(2n+1)^{\frac{1}{2}}} \sum_{i=1}^{K+1} \mid \int_{a_i}^{b_i} ((P_{n+1}(y) - P_{n-1}(y)))dy \mid && (43)\\
&\leq \frac{\Delta}{4(2n+1)^{\frac{1}{2}}} \sum_{i=1}^{K+1} \sqrt{2L^2} \sqrt{\int_{a_i}^{b_i} (P_{n+1}(y) - P_{n-1}(y))^2 dy} \quad \text{(Cauchy-Schwarz)} && (44)\\
&\leq \frac{\Delta}{4(2n+1)^{\frac{1}{2}}} \sqrt{2L^2} \sum_{i=1}^{K+1} \sqrt{\int_{a_i}^{b_i} (P_{n+1}(y) - P_{n-1}(y))^2 dy} && (45)\\
&\leq \frac{\Delta}{4(2n+1)^{\frac{1}{2}}} \sqrt{2L^2} \sum_{i=1}^{K+1} \sqrt{\int_{-1}^{1} (P_{n+1}(y) - P_{n-1}(y))^2 dy} && (46)\\
&= \frac{\Delta}{4(2n+1)^{\frac{1}{2}}} \sqrt{2L^2}(K+1) \sqrt{\int_{-1}^{1} P_{n+1}^2(y) + P_{n-1}^2(y) dy} && (47)\\
&= \frac{\Delta}{4(2n+1)^{\frac{1}{2}}} \sqrt{2L^2}(K+1) \sqrt{\frac{2}{2n+3} + \frac{2}{2n-1}} && (48)\\
&= \frac{\Delta}{2(2n+1)^{\frac{1}{2}}} L(K+1) \sqrt{\frac{1}{2n+3} + \frac{1}{2n-1}} && (49)
\end{aligned}$$

We then bound $\mid B_n(t) \mid$.

$$
\begin{aligned}
\mid B_n(t) \mid &= \sum_{i=1}^{K} \frac{(2n+1)^{-\frac{1}{2}}}{2} \mid x(\frac{\Delta(y-1)}{2}+t)(P_{n+1}(y)-P_{n-1}(y))\Big|_{b_i^-}^{b_i^+} \mid \\
&= \sum_{i=1}^{K} \frac{(2n+1)^{-\frac{1}{2}}}{2} \mid x(\frac{\Delta(y-1)}{2}+t)\Big|_{b_i^-}^{b_i^+} \mid \cdot \mid (P_{n+1}(b_i)-P_{n-1}(b_i)) \mid \\
&\leq \sum_{i=1}^{K} 2 \cdot \frac{(2n+1)^{-\frac{1}{2}}}{2} \mid x(\frac{\Delta(y-1)}{2}+t)\Big|_{b_i^-}^{b_i^+} \mid \cdot \quad (\mid P_n \mid \text{is bounded by 1}) \\
&= \frac{1}{(2n+1)^{\frac{1}{2}}} \sum_{i=1}^{K} \Delta_i
\end{aligned}
$$

where we set Δ_i as the absolute value of the gaps at the observation points.

A tighter bound can be achieved by assuming that observations are confined in the $[t-0.95\Delta, t-0.05\Delta]$ region (Lohöfer, 1998). In this case, $\forall y \in [-0.9, 0.9]$, we have

$$
\begin{aligned}
\mid P_n(y) \mid &< \sqrt{\frac{2}{\pi(n+\frac{1}{2})}} \frac{1}{(1-x^2)^{\frac{1}{4}}} \\
&<= \sqrt{\frac{2}{\pi(n+\frac{1}{2})}} \frac{1}{(0.99)^{\frac{1}{4}}}
\end{aligned}
$$

So we have in this case,

$$
\begin{aligned}
\mid B_n(t) \mid &\leq \frac{1}{(2n+1)^{\frac{1}{2}}} \sqrt{\frac{2}{\pi(n+\frac{1}{2})}} \frac{1}{(0.99)^{\frac{1}{4}}} \sum_{i=1}^{K} \Delta_i \\
&< \frac{1}{2(2n+1)^{\frac{1}{2}}} \frac{1}{\sqrt{\pi(n+\frac{1}{2})}} \sum_{i=1}^{K} \Delta_i
\end{aligned}
$$

Together, we have

$$
\begin{aligned}
\sum_{n=N}^{\infty} c_n^2(t) &\leq \sum_{n=N}^{\infty} \mid A_n(t) \mid^2 + 2 \mid A_n(t) \mid \cdot \mid B_n(t) \mid + \mid B_n(t) \mid^2 \\
&= \sum_{n=N}^{\infty} \mid A_n(t) \mid^2 + \sum_{n=N}^{\infty} 2 \mid A_n(t) \mid \cdot \mid B_n(t) \mid + \sum_{n=N}^{\infty} \mid B_n(t) \mid^2
\end{aligned}
$$

The first series gives

$$
\begin{aligned}
\sum_{n=N}^{\infty} \mid A_n(t) \mid^2 &\leq \sum_{n=N}^{\infty} \frac{\Delta^2}{(2n+1)} L^2 (K+1)^2 (\frac{1}{2n+3} + \frac{1}{2n-1}) \\
&= \Delta^2 L^2 (K+1)^2 \sum_{n=N}^{\infty} \frac{1}{(2n+1)} (\frac{1}{2n+3} + \frac{1}{2n-1}) \\
&= \Delta^2 L^2 (K+1)^2 \frac{1}{2(1+2N)} \frac{1}{4N-2}
\end{aligned}
$$

The second series gives

$$\begin{aligned}
\sum_{n=N}^{\infty} 2 \mid A_n(t) \mid \cdot \mid B_n(t) \mid &\leq 2 \sum_{n=N}^{\infty} \frac{\Delta}{(2n+1)^{\frac{1}{2}}} L(K+1) \sqrt{\frac{1}{2n+3} + \frac{1}{2n-1}} \cdot \frac{1}{(2n+1)^{\frac{1}{2}}} \sum_{i=1}^{K} \Delta_i \\
&= 2\Delta \cdot L(K+1) \sum_{i=1}^{K} \Delta_i \sum_{n=N}^{\infty} \frac{1}{(2n+1)} \sqrt{\frac{1}{2n+3} + \frac{1}{2n-1}} \\
&\leq 2\Delta \cdot L(K+1) \sum_{i=1}^{K} \Delta_i \sum_{n=N}^{\infty} \frac{1}{(2n-1)} \sqrt{\frac{2}{2n-1}} \\
&= 2\sqrt{2}\Delta \cdot L(K+1) \sum_{i=1}^{K} \Delta_i \sum_{n=N}^{\infty} \frac{1}{(2n-1)^{\frac{3}{2}}} \\
&= 2\sqrt{2}\Delta \cdot L(K+1) (\sum_{i=1}^{K} \Delta_i) \xi(\frac{3}{2}, N-1)
\end{aligned}$$

Where $\xi(s, a)$ is the Hurwitz-Zeta function.

The third series gives

$$\begin{aligned}
\sum_{n=N}^{\infty} \mid B_n(t) \mid^2 &\leq \sum_{n=N}^{\infty} \frac{1}{4(2n+1)} \frac{1}{\pi(n+\frac{1}{2})} (\sum_{i=1}^{K} \Delta_i)^2 \\
&= \frac{1}{4\pi} (\sum_{i=1}^{K} \Delta_i)^2 \sum_{n=N}^{\infty} \frac{1}{(2n+1)(n+\frac{1}{2})} \\
&< \frac{1}{4\pi} (\sum_{i=1}^{K} \Delta_i)^2 \sum_{n=N}^{\infty} \frac{1}{(2n)(n)} \\
&= \frac{1}{4\pi} (\sum_{i=1}^{K} \Delta_i)^2 \frac{1}{2} \xi(2, N)
\end{aligned}$$

Plugging everything together, we then have

$$\begin{aligned}
\sum_{n=N}^{\infty} c_n^2(t) &\leq \Delta^2 L^2 (K+1)^2 \frac{1}{2(1+2N)} \frac{1}{4N-2} \\
&\quad + 2\sqrt{2}\Delta L(K+1) (\sum_{i=1}^{K} \Delta_i) \xi(\frac{3}{2}, N-1) \\
&\quad + \frac{1}{4\pi} (\sum_{i=1}^{K} \Delta_i)^2 \frac{1}{2} \xi(2, N) \\
&\leq C_0 \frac{\Delta^2 L^2 (K+1)^2}{N(2N-1)} + C_1 \Delta L(K+1) S_\Delta \xi(\frac{3}{2}, N) + C_2 S_\Delta^2 \xi(\frac{3}{2}, N)
\end{aligned}$$

as stated in Result 4.1.

C EXPERIMENTS WITH MORE IRREGULAR RATES

In Table 4, we present additional results for the Lorenz63 dataset. In Table 5, for the Lorenz96 dataset and in Table 6 for the Synthetic dataset. We compare against the same baselines and also

provide the comparison against different interpolation schemes (as described in Section **??**). We observe a similar trend for all irregular rates. Namely, PolyODEprovides better downstream classification performance as well as better reconstruction.

Table 4: Performance on Lorenz Attractor

Model	Downstream Classification↑			Reconstruction↓		
Irregular Rate	0.3	0.4	0.5	0.3	0.4	0.5
GRU-ODE	0.825 ± 0.031	0.834 ± 0.025	0.818 ± 0.032	0.752 ± 0.057	0.690 ± 0.097	0.683 ± 0.035
ODE-RNN	0.813 ± 0.013	0.860 ± 0.019	0.885 ± 0.011	0.674 ± 0.049	0.566 ± 0.077	0.633 ± 0.050
Neural-RDE	0.604 ± 0.046	0.681 ± 0.095	0.815 ± 0.019	0.989 ± 0.074	1.437 ± 0.933	1.030 ± 0.065
HiPPO-obs	0.837 ± 0.034	0.886 ± 0.025	0.903 ± 0.025	0.511 ± 0.043	0.368 ± 0.011	0.284 ± 0.052
HiPPO-RNN	0.804 ± 0.023	0.822 ± 0.025	0.888 ± 0.035	0.784 ± 0.122	0.656 ± 0.277	0.356 ± 0.018
S4	0.911 ± 0.005	0.921 ± 0.009	0.936 ± 0.009	0.428 ± 0.040	0.371 ± 0.025	0.321 ± 0.040
Extended-S4	0.909 ± 0.015	0.921 ± 0.009	0.933 ± 0.010	0.439 ± 0.026	0.374 ± 0.015	0.329 ± 0.033
Hermite Interpolation	0.976 ± 0.000	0.984 ± 0.000	0.9874 ± 0.000	0.135 ± 0.007	0.083 ± 0.008	0.036 ± 0.004
Linear Interpolation	0.744 ± 0.016	0.750 ± 0.019	0.774 ± 0.031	0.787 ± 0.066	0.787 ± 0.038	0.777 ± 0.036
Constant Interpolation	0.664 ± 0.033	0.630 ± 0.019	0.724 ± 0.122	0.785 ± 0.074	0.923 ± 0.128	0.612 ± 0.154
PolyODE	$\mathbf{0.992 \pm 0.000}$	$\mathbf{0.990 \pm 0.000}$	$\mathbf{0.994 \pm 0.000}$	$\mathbf{0.034 \pm 0.008}$	$\mathbf{0.023 \pm 0.006}$	$\mathbf{0.011 \pm 0.002}$

Table 5: Performance on Lorenz96 Attractor

Model	Classification↑			Regression↓		
Irregular Rate	0.3	0.4	0.5	0.3	0.4	0.5
GRU-ODE	0.925 ± 0.004	0.907 ± 0.030	0.925 ± 0.008	0.346 ± 0.072	0.270 ± 0.031	0.261 ± 0.018
ODE-RNN	0.954 ± 0.012	0.947 ± 0.007	0.927 ± 0.015	0.214 ± 0.030	0.225 ± 0.016	0.192 ± 0.022
Neural-RDE	0.606 ± 0.112	0.799 ± 0.031	0.865 ± 0.042	1.747 ± 0.472	1.853 ± 1.642	1.483 ± 0.904
HiPPO-obs	0.949 ± 0.007	0.954 ± 0.006	0.960 ± 0.006	0.247 ± 0.005	0.149 ± 0.037	0.112 ± 0.002
HiPPO-RNN	0.944 ± 0.008	0.958 ± 0.007	0.976 ± 0.010	0.198 ± 0.014	0.160 ± 0.022	0.107 ± 0.008
S4	0.948 ± 0.016	0.956 ± 0.015	0.963 ± 0.006	0.171 ± 0.008	0.174 ± 0.019	0.161 ± 0.014
ExtendedS4	0.937 ± 0.018	0.945 ± 0.014	0.948 ± 0.013	0.208 ± 0.066	0.166 ± 0.047	0.140 ± 0.002
Hermite Interpolation	0.983 ± 0.004	0.988 ± 0.001	0.992 ± 0.005	0.093 ± 0.011	0.036 ± 0.008	0.020 ± 0.002
Linear Interpolation	0.857 ± 0.026	0.886 ± 0.032	0.886 ± 0.020	0.388 ± 0.032	0.369 ± 0.008	0.351 ± 0.051
Constant Interpolation	0.862 ± 0.017	0.905 ± 0.015	0.880 ± 0.030	0.393 ± 0.017	0.371 ± 0.068	0.266 ± 0.040
PolyODE	$\mathbf{0.984 \pm 0.002}$	$\mathbf{0.994 \pm 0.002}$	$\mathbf{0.992 \pm 0.002}$	$\mathbf{0.038 \pm 0.008}$	$\mathbf{0.021 \pm 0.003}$	$\mathbf{0.011 \pm 0.002}$

Table 6: Performance on SimpleTraj

Model	Classification↑			Regression↓		
Irregular Rate	0.7	0.8	0.9	0.7	0.8	0.9
GRU-ODE	0.968 ± 0.004	0.964 ± 0.008	0.976 ± 0.013	0.057 ± 0.010	0.044 ± 0.007	0.029 ± 0.011
ODE-RNN	0.938 ± 0.034	0.950 ± 0.003	0.967 ± 0.015	0.080 ± 0.036	0.049 ± 0.007	0.031 ± 0.001
Neural-RDE	0.773 ± 0.111	0.896 ± 0.016	0.949 ± 0.041	0.167 ± 0.031	0.132 ± 0.024	0.097 ± 0.035
HiPPO-obs	0.758 ± 0.023	0.805 ± 0.010	0.889 ± 0.006	0.197 ± 0.010	0.161 ± 0.007	0.098 ± 0.006
HiPPO-RNN	0.742 ± 0.008	0.789 ± 0.016	0.902 ± 0.010	0.209 ± 0.018	0.165 ± 0.018	0.090 ± 0.024
S4	$\mathbf{0.994 \pm 0.003}$	0.998 ± 0.001	0.999 ± 0.001	0.032 ± 0.006	0.019 ± 0.003	0.015 ± 0.005
S4 extended	$\mathbf{0.995 \pm 0.002}$	0.993 ± 0.004	0.999 ± 0.001	0.017 ± 0.004	0.012 ± 0.001	0.008 ± 0.001
Hermite Interpolation	0.971 ± 0.012	0.972 ± 0.011	0.980 ± 0.011	0.055 ± 0.016	0.048 ± 0.009	0.024 ± 0.010
Linear Interpolation	0.969 ± 0.008	0.973 ± 0.010	0.986 ± 0.010	0.028 ± 0.005	0.013 ± 0.001	0.009 ± 0.002
Constant Interpolation	0.969 ± 0.005	0.980 ± 0.005	0.992 ± 0.004	0.027 ± 0.003	0.013 ± 0.002	0.006 ± 0.002
PolyODE	$\mathbf{0.994 \pm 0.003}$	$\mathbf{0.998 \pm 0.001}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.012 \pm 0.002}$	$\mathbf{0.005 \pm 0.003}$	$\mathbf{0.002 \pm 0.001}$

D OTHER ILLUSTRATIONS

We complement the analysis of the reconstruction performance of PolyODEon the MIMIC-III dataset with the trajectories of other vitals. In Figure 5, we plot the true trajectories and reconstruction of the heart rate, mean blood pressure and diastolic blood pressure. In Figure 6, we plot the true trajectories and reconstruction of the oxygen saturation, respiratory rate and blood glucose. In Figure 7, we plot the true trajectories and reconstruction of the blood urea nitrogen, the white blood cells count and body temperature. In Figure 8, we plot the true trajectories and reconstruction of the creatinine levels.

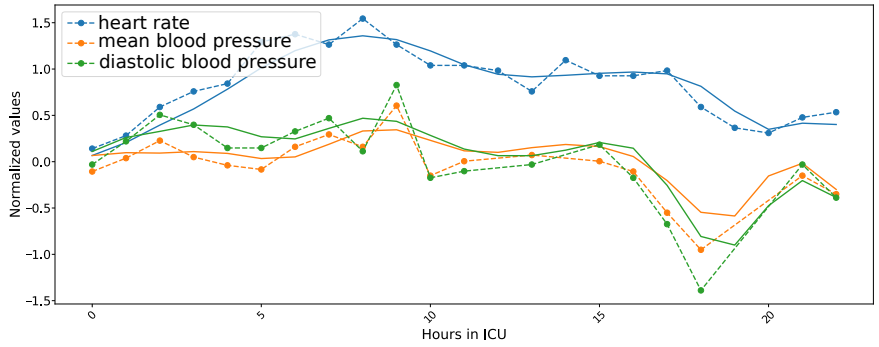

Figure 5: **PolyODE**: Reverse reconstruction of vitals over the 24 hours of ICU of a randomly selected patient from the test. We plot the true value (dots) and reconstructions (solid line) for heart rate, mean blood pressure and diastolic blood pressure. Reverse reconstruction is done from the last time observation.

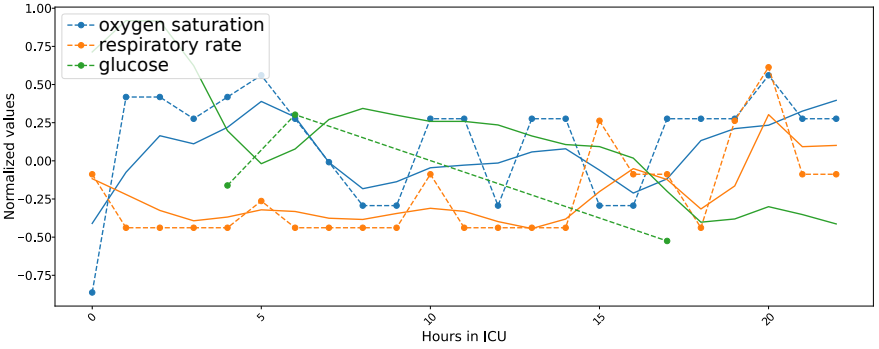

Figure 6: **PolyODE**: Reverse prediction of vitals over the 24 hours of ICU of a randomly selected patient from the test. We plot the true value (dots) and reconstructions (solid line) for oxygen saturation, respiratory rate and blood glucose. Reverse reconstruction is done from the last time observation.

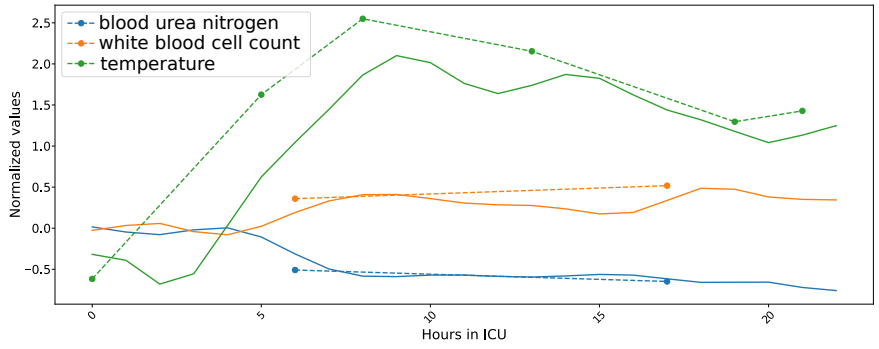

Figure 7: **PolyODE**: Reverse reconstructioon of vitals over the 24 hours of ICU of a randomly selected patient from the test. We plot the true value (dots) and reconstructions (solid line) for blood urea nitrogen, white blood cell count and body temperature. Reverse reconstruction is done from the last time observation.

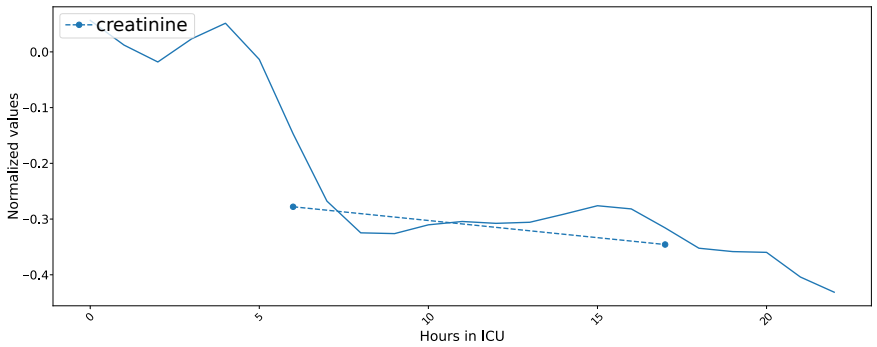

Figure 8: **PolyODE**: Reverse reconstruction of vitals over the 24 hours of ICU of a randomly selected patient from the test. We plot the true value (dots) and reconstructions (solid line) for creatinine. Reverse reconstruction is done from the last time observation.

E Details on the backward reconstruction process

Reconstructions in PolyODE are fixed operators, as suggested by Equation 3. The reconstruction is obtained by using the orthogonal polynomial coefficients at a particular step. This procedure ensures accurate backward prediction performance, as supported by Result 4.1.

In contrast, classical NODE reconstructions (such as GRU-ODE or ODE-RNN, and depicted in Figure 1) are computed using backward integration of the hidden process. That is, conditioned on the hidden at time , one can reconstruct the time series at time using :

$$\hat{\mathbf{x}}(t') = g(\mathbf{h}(t')) \tag{50}$$

$$\mathbf{h}(t') = \mathbf{h}(t) + \int_t^{t'} \Phi(\mathbf{h}(s))ds \tag{51}$$

where ϕ is the neural network characterizing the NODE.

We note that this reconstruction exists and is unique if ϕ is continuous in t and Lipschitz continuous in $\mathbf{h}$, according to the Picard-Lindelöf theorem Nagle et al. (2011). As neural networks parametrized with continuous activation functions (*e.g.* hyperbolic tangent or sigmoid) are Lipschitz continuous, we have that the above reconstruction exists and is unique for these activation functions.

F Uncertainty experiment details

In this experiment, we use a similar system of ordinary differential equations but extend it with other coefficient processes to model the uncertainties over time.

$$\begin{cases} \frac{d\mathbf{c}^1(t)}{dt} & = A_\mu \mathbf{c}^1(t) + B_\mu g_1(\mathbf{h}(t)) \\ & \vdots \\ \frac{d\mathbf{c}^d(t)}{dt} & = A_\mu \mathbf{c}^d(t) + B_\mu g_d(\mathbf{h}(t)) \\ \frac{d\mathbf{c}^{1,\sigma}(t)}{dt} & = A_\mu \mathbf{c}^{1,\sigma}(t) + B_\mu g_1^\sigma(\mathbf{h}(t)) \\ & \vdots \\ \frac{d\mathbf{c}^{d,\sigma}(t)}{dt} & = A_\mu \mathbf{c}^{d,\sigma}(t) + B_\mu g_d^\sigma(\mathbf{h}(t)) \\ \frac{d\mathbf{h}(t)}{dt} & = \phi_\theta(\mathbf{h}(\mathbf{t})) \end{cases} \tag{52}$$

where $\mathbf{c}^{i,\sigma}$ are the coefficients used for the uncertainty modeling. We also introduce another neural network $g^{\sigma}(\cdot)$ that produces the standard deviation of observations. We then model the output distribution of the predictions as a normal distribution with mean $\hat{\mu}_{\mathbf{x}}(t) = g(\mathbf{h}(t))$ and standard deviation $\hat{\sigma}_{\mathbf{x}}(t) = g^{\sigma}(\mathbf{h}(t))$. We can then reconstruct the past means and uncertainty estimates using

$$\hat{\mu}_{\mathbf{x}_{<t,j}} = \sum_{n=0}^{N} c_n^j(t) \cdot P_n^t \cdot \frac{1}{\alpha_n^t} \tag{53}$$

$$\hat{\sigma}_{\mathbf{x}_{<t,j}} = \sum_{n=0}^{N} c_n^{j,\sigma}(t) \cdot P_n^t \cdot \frac{1}{\alpha_n^t} \tag{54}$$

In the top panel of Figure 4, we plot the tuples $(\hat{\sigma}_{\mathbf{x}_{<t_T}}, \|\mathbf{x}_{<t_T} - \hat{\mu}_{\mathbf{x}_{<t_T}}\|_2)$ for 128 time series in the test set, for all observations before time $t = t_T$ and reconstructing from $t = t_T$. In the bottom panel, we plot the tuples $(\hat{\sigma}_{\mathbf{x}}(t_T), \|\mathbf{x}_{<t} - \hat{\mu}_{\mathbf{x}_{<t}}\|_2)$ for the same time series and observations. $\hat{\sigma}_{\mathbf{x}}(t_T)$ is the predicted uncertainty at the last time step while $\hat{\sigma}_{\mathbf{x}_{<t_T}}$ is the reconstruction of the uncertainties over the past trajectories.

G Datasets details

G.1 Synthetic Univariate

We validate our approach using a univariate synthetic time series generated by the equations:

$$x(t) = \sin(x + \phi) * \cos(3 * (x + \phi)) \quad \text{with} \quad \phi \sim \mathcal{N}(0, 2\pi)$$

We simulate 1000 realizations from this process from $t = 0$ to $t = 10$ and sample from it at irregularly spaced time points using a Poisson point process with rate $\lambda[\frac{1}{s}]$. For each generated irregularly sampled time series $\mathbf{x}$, we create a binary label $y = \mathbb{I}[x(5) > 0.5]$.

G.2 Lorenz63

We simulate 1000 realizations from the Lorenz system for 10000 time steps with a time interval of 0.01 seconds. We then rescale the time axis to $[0, 10]$ seconds. We sample irregularly spaced time points using a Poisson point process with rate $\lambda[\frac{1}{s}]$. For each generated irregularly sampled time series $\mathbf{x}$, we create a binary label $y = \mathbb{I}[x^3(6) > 0.]$. We select only the two first dimensions of the system (x^0, x^1).

G.3 Lorenz96

We simulate 1000 realizations from the Lorenz96 system for 10000 time steps with a time interval of 0.01 seconds. We then rescale the time axis to $[0, 10]$ seconds. We sample irregularly spaced time points using a Poisson point process with rate $\lambda[\frac{1}{s}]$. For each generated irregularly sampled time series $\mathbf{x}$, we create a binary label $y = \mathbb{I}[x^5(6) > 0.]$.We select only the four first dimensions of the system $(x^0, ..., x^3)$.

G.4 MIMIC-III

We use a pre-processed version of the MIMIC-III dataset (Johnson et al., 2016; Wang et al., 2020). This consists of the first 24 hours of follow-up for ICU patients. We use the following longitudinal variables: heart rate, mean and diastolic blood pressure, oxygen saturation, respiratory rate, glucose, blood urea nitrogen, white blood cell count, temperature and creatinine. For each time series, the label y is the in-hospital mortality.

H IMPLEMENTATION DETAILS

In table 7, we display the hyper-parameters used to train our model in the experiments. For the downstream classification experiments, we use a two layers multi-layer perceptron with hidden dimension 32 and a binary cross entropy loss. We train our model using Adam optimizer and do not use dropout or weight decay but use early stopping. We train our model on the train set and select the model weights at epoch with lowest validation loss. We then evaluate this model on a held out test set. For uncertainty estimation of the results, we use a 3-fold cross validation approach.

Hyperparameter	Description	Lorenz63	Lorenz96	Synthetic	MIMIC-III
N	Number of coefficients	32	32	32	18
d_h	Hidden dimension	32	32	32	18
B	batch size	128	128	128	256
l_r	learning rate	0.001	0.001	0.001	0.001
δ_t	Integration step size	0.05	0.05	0.5	0.05
N_{obs}	Number of observations per time series if $\lambda = 1.0$	100	100	10	/
Δ	Width of the weight function	5	5	5	10
t_T	Maximum time value	10	10	10	20

Table 7: Hyper-parameters used for training PolyODE in the experiments.

H.1 S4 HYPER-PARAMETERS SEARCH

For the results reported for the S4 baseline we performed an hyper-parameter search on the number of state-dimensions and model-dimensions.

We used $[64, 128, 256, 512]$ for the model dimension and $[64, 128]$ for the state dimension. We reported the results for the best performing set of hyper-parameters on the validation set.

I NUMERICAL INTEGRATION DETAILS

I.1 OVERVIEW OF NUMERICAL SOLVERS

Training and inference of PolyODE requires numerically integrating the underlying neural ODE. Different choices of numerical integrators are possible. For completeness, we give a brief overview of three numerical integration methods below: the Euler method, the Dormand-Prince method (Dopri-5) and the Adams-Moulton method.

All methods aim at solving an initial value problem such as

$$\frac{dy(t)}{dt} = f(t, y(t)) \quad \text{s.t.} \quad y(t_0) = y_0. \tag{55}$$

That is, based on y_0 and f, one wishes to compute values of y at an arbitrary times t^*.

Euler method The Euler method divides the interval between t_0 and t^* in smaller intervals of fixed size $h : [t_0, t_h], [t_h, t_{2h}], ..., [t_{t^*-h}, t_{t^*}]$

One then uses the following recurrence equation to evaluate the value of y until $t = t^*$:

$$y_{n+1} = y_n + hf(t_n, y_n).$$

Dormand-Prince method The Dormand-Prince (Dopri-5) method, also known as the adaptive Runge-Kutta 4(5) method allows for automatically choosing the step size of the integration step by effectively jointly running two numerical solvers.

Each of these solvers is a Runge-Kutta solver (one of order 4 and the other of order 5).

For a step size h, a Runge-Kutta integrator of order n uses the following recurrence equation:

$$y_{n+1} = y_n + hf(t_n, y_n).$$

with

$$k_1 = f(t_n, y_n), \tag{56}$$
$$k_2 = f(t_n + c_2 h, y_n + h(a_{21} k_1)), \tag{57}$$
$$k_s = f(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + ... + a_{s,s-1} k_{s-1} \tag{58}$$

The error of the integrator is then computed by comparing the results of both solvers. If the error is larger than a pre-specified threshold, the integration step is reduced until an acceptable precision is obtained.

Adams-Moulton Adams-Moulton methods are implicit methods. They differ from the Euler and Dopri-5 methods which are explicit. Explicit methods can compute the value of y explicitly using a recurrence equation (*i.e.* only using past value of y). In contrast implicit methods use future values of the target function y in the integration step, which requires solving an equation involving the state of the system.

Adams-Moulton methods exist for different orders, depending on the number of intermediate steps. We list here orders 0,1,2 and 4.

$$y_n = y_{n-1} + hf(t_n, y_n) \tag{59}$$
$$y_{n+1} = y_n + \frac{1}{2} h(f(t_{n+1}, y_{n+1}) + f(t_n, y_n)) \tag{60}$$
$$y_{n+2} = y_{n+1} + h(\frac{5}{12} f(t_{n+2}, y_{n+2}) + \frac{8}{12} f(t_{n+1}, y_{n+1}) - \frac{1}{12}(ft_n, y_n)) \tag{61}$$
$$y_{n+4} = y_{n+3} + h(\frac{251}{270} f(t_{n+4}, y_{n+4}) + \frac{646}{720} f(t_{n+3}, y_{n+3}) - \frac{264}{720}(ft_{n+2}, y_{n+2}) \tag{62}$$
$$+ \frac{106}{720}(ft_{n+1}, y_{n+1}) - \frac{19}{720}(ft_n, y_n)) \tag{63}$$

In our experiments, we used the Adams-Moulton method of order 4.

I.2 Comparison of the different numerical integrators

We compare the performance of each integrator in terms of the compute time and the forecasting validation loss. In Table 8 we report the time duration required to train the PolyODE for 250 epochs on the Lorenz dataset as well as the lowest forecasting MSE achieved during training. We used a step size of 0.05 for the Euler and the Adams-Moulton method. A graphical comparison of the computation times is also provided in Figure 9. We observe that the Euler method achieves very poor performance, suggesting divergence in the numerical integration. The Dopri-5 achieves similar performance as the Adams-Moulton but requires much more time. This significantly higher computation time is attributed to a poorly managed integration error, which requires a constant adjustment of the integration step. In contrast, we see that the implicit Adams-Moulton method provides both short training times and good forecasting performance. We attribute this phenomenon to the stiffness ratio of the matrix A_μ, as detailed below.

I.3 Stiffness ratio of the matrix A_μ

Part of the neural ODE system of PolyODE is linear homogeneous. In that system, the spectral characterization of the matrix A_μ is crucial in understanding the stability properties of that ODE. Indeed, the characteristics of the matrix A_μ point to a stiff behavior of the system, which hampers the stability of the explicit numerical integrators (such as Dopri-5 or Euler).

Solver	Computation Time [s]	Validation Loss [MSE]
Euler	816 ± 16	1519 ± 633
Dopri-5	23286 ± 2419	0.093 ± 0.012
Adams-Moulton	1911 ± 92	0.099 ± 0.016

Table 8: Numerical comparison of different solvers in terms of computation time and validation loss achieved. The computation time corresponds to the time required to train PolyODE for 250 epochs on the Lorenz dataset. The validation loss is the lowest mean-square error on forecasting achieved during training.

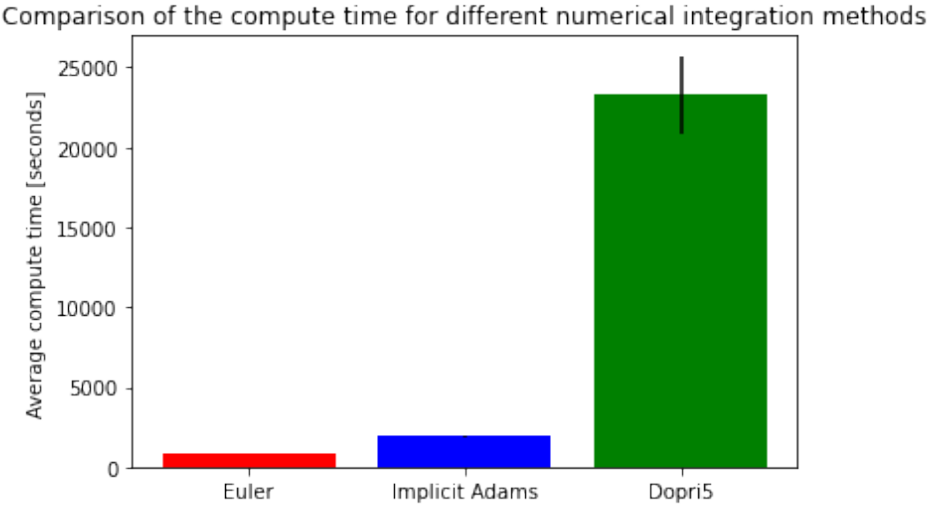

Figure 9: Graphical comparison of the different numerical solvers with respect to computation time. The computation time corresponds to the time required to train PolyODE for 250 epochs on the Lorenz dataset.

In particular, all real parts of the eigenvalues of A_μ are negative. What is more, the stiffness ratio grows with the number of projection coefficients N. The stiffness ratio is defined as the ratio between the magnitude of the eigenvalue with the largest real part and the magnitude of the eigenvalue with the smallest real part. In Figure 10, we show the magnitude of the real parts of the eigenvalues for 3 different N. In Figure 11, we show the stiffness ratio of A_μ in function of the number of projection coefficients. We observe a larger stiffness ratio as the number of coefficients grows large, therefore pointing to more numerical instability with explicit methods.

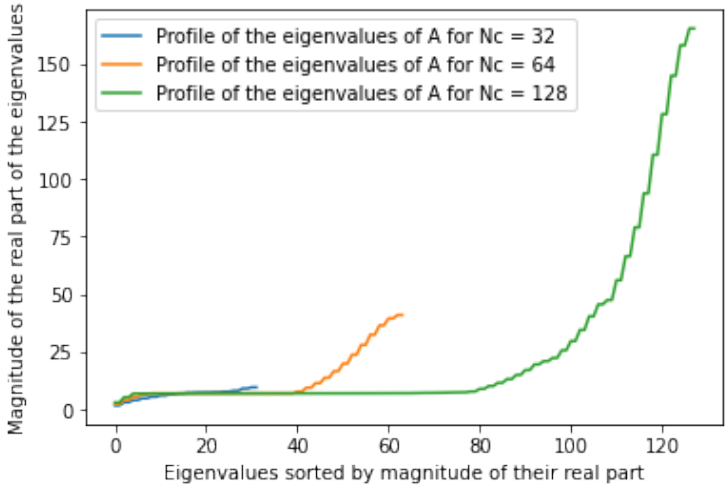

Figure 10: Magnitude of the real parts of the eigenvalues of the matrix A_μ in ascending order for different values of N.

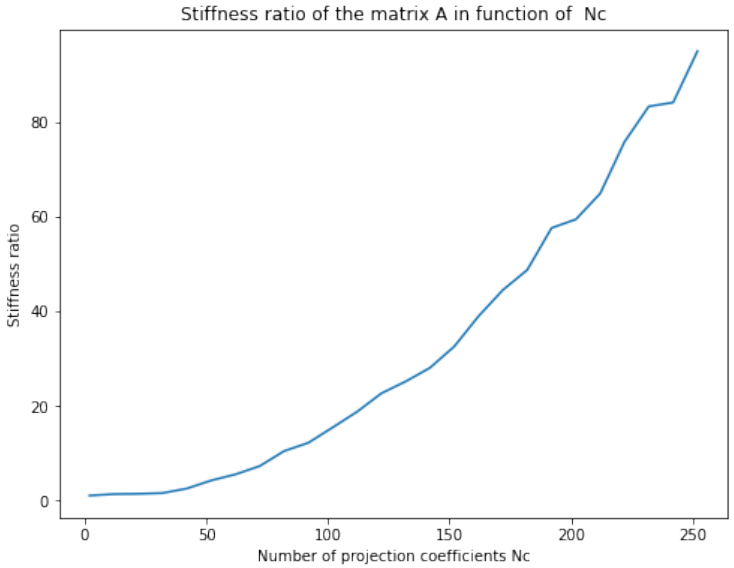

Figure 11: Evolution of the stiffness ratio, defined as the ratio between the largest and smallest magnitude of the real part of the eigenvalues of the matrix A_μ for different values of N.