

Neural Ordinary Differential Equations with TM-Solver to Predict Time Series Data

Anna Golovkina(✉) and Anna Vashukova

Saint Petersburg State University, Saint Petersburg, Russia
a.golovkina@spbu.ru

Abstract. The paper aims to analyze the performance of Neural ODE model based solutions for time-series prediction. The original paper from 2018 gave the opportunity for continuous-time modeling of long-time processes. Since then, lots of different modifications were proposed, like ODE-RNN and p-node ODE-RNN, augmented neural ODE, Hamiltonian Neural Networks, Neural Controlled differential equations, etc. All these modifications try to modify the form of ODE lying behind the NN layers. However, the performance of these approaches is still strongly influenced by the choice of numerical solvers, with conventional methods often compromising computational efficiency, stability, or precision. This paper introduces the Taylor Mapping (TM) solver, a numerical integrator that leverages polynomial approximations of the ODE general solution for Neural ODEs. The TM-solver adaptively truncates higher-order terms to balance accuracy and computational cost. What is more, the TM-solver explicitly constructs interpretable maps to reveal local dynamics and can be integrated with automatic differentiation frameworks for efficient training. Evaluations show that the TM-solver improves Neural ODEs by increasing prediction accuracy for synthetic data and real-world datasets.

Keywords: TM-solver · Neural ODE · time-series prediction

1 Introduction

Time-series modeling and prediction, a critical task in domains such as finance, engineering, or energy forecasting, requires models that are capable of capturing complex time dynamics despite possible irregularities in data sampling and scale of temporal resolution. Conventional methods, including autoregressive models and recurrent neural architectures, often impose rigid discrete-time structures, limiting their adaptability to irregularly sampled sequences or continuous latent processes. Neural ordinary differential equations (Neural ODEs) [6] have emerged as a transformative framework for modeling temporal evolution via continuous-time dynamics governed by differential equations. By parameterizing the derivative of hidden states via a neural network, neural ODEs inherently adapt to

The authors acknowledge Saint Petersburg State University for a research project 121061000159-6.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
O. Gervasi et al. (Eds.): ICCSA 2025 Workshops, LNCS 15894, pp. 282–293, 2026.
https://doi.org/10.1007/978-3-031-97648-3_19

variable time intervals, learn smooth trajectories, and reduce architectural complexity compared to their discrete-layer counterparts.

One of the key parameters strongly influencing the performance of Neural ODEs is the choice of numerical solver that approximates the integration of the learned differential equations [19–22]. Although conventional solvers such as explicit Runge-Kutta or adaptive step size methods (e.g., Dopri5) are widely adopted [3], their computational cost, stability, and precision trade-offs pose challenges for gradient-based training and long-horizon forecasting. This work introduces the Taylor Mapping (TM) solver, a numerical integrator that combines a polynomial representation of the solution derived from Taylor series expansions [11]. Unlike generic methods for an ODE solution, the TM-solver explicitly constructs a local polynomial approximation of the ODE solution by a straightforward procedure at each step. This approach enables precise, high-order approximations of the state trajectory while keeping analytical tractability, a critical feature for efficient gradient propagation in neural ODE training.

The TM-solver offers numerous advantages for modeling time-series data. The polynomial representation facilitates the adaptive truncation of higher-order terms, striking a balance between accuracy and computational efficiency [10]. Second, the explicit computation of Taylor map coefficients (compare to the learned coefficients in traditional solver's approximation formulas) provides information on the local dynamics of the system, improving the interpretability of the learned ODE. Third, compatibility of the method with automatic differentiation frameworks (e.g., PyTorch, TensorFlow) ensures seamless integration with neural network training pipelines [16]. Thus, when combined with Neural ODEs, the TM-solver is well suited for time series applications requiring high precision and smoothness, such as forecasting chaotic systems or interpolating irregularly sampled data (e.g., from sensors).

The polynomial-based methodology of the TM-solver improves interpretability and conforms to the requirements of contemporary scientific computer infrastructure. The solver is optimized for deployment in distributed systems, including extensive climate simulations and real-time sensor networks, due to its provision of closed-form solutions and effective parallelization. This research further investigates its application to real-time vibration sensor data forecasting.

This work explores the synergy between Neural ODEs and the TM-solver, focusing on its capacity to (1) improve prediction accuracy by leveraging high-order Taylor terms for fine-grained trajectory approximation, (2) reduce sensitivity to solver hyperparameters through adaptive polynomial truncation, and (3) enable scalable training via gradient-compatible Taylor coefficient computation. By unifying the continuous-time modeling flexibility of neural ODEs with the analytical rigor of Taylor-based solvers, this framework aims to advance methods in time-series prediction, offering a mathematically grounded and computationally feasible tool for real-world applications. Subsequent sections evaluate the performance of the TM solver against the other Neural ODE modifications.

The rest of the paper is organized as follows: we summarize the related work in Sect. 2. Section 3 presents the description of the combined model. Section 4

introduces the details of the experiment and the experimental results; Sect. 5 explains the conclusions and future work.

2 Related Work

2.1 Neural ODE

Basic Concept. Neural Ordinary Differential Equations (NODEs) are a novel type of model that can be thought of as endless residual blocks connecting the model's input and output [6]. NODEs are an appropriate tool for learning complicated systems over extended periods of time due to their property of continuous infinite depth [20]. In recent years, the NODE has been widely researched in theory and methods to improve its efficiency and robustness. It has been shown to be effective in a variety of applications, including image recognition, point cloud learning, and time series prediction [13].

NODE concept presents deep learning as dynamic systems governed by differential equations so that the hidden state evolves via an ODE that is parameterized by a neural network:

$$\begin{aligned} \frac{d\mathbf{X}(t)}{dt} &= f(\mathbf{X}(t), t, \theta), \\ \mathbf{X}(0) &= (t_0), \end{aligned} \tag{1}$$

where the function f models the dynamics with the initial condition $f(0)$, parameter θ is utilized to optimally approximate f in the ODE model.

Thus, we can compute the terminal value $\mathbf{X}(T)$ of the process started from t_0 $(T > t_0)$ by integrating ODEs:

$$\mathbf{X}(T) = \mathbf{X}(t_0) + \int_{t_0}^{T} f(\mathbf{X}(t), \theta) dt. \tag{2}$$

To test and train the model, a loss function can be defined as the distance between the prediction and training set

$$\mathcal{L}(\mathbf{X}(T), \mathcal{Y}_{\text{train}}) = d(\mathbf{X}(T), \mathcal{Y}_{\text{train}}),$$

where $d(\cdot, \cdot)$ is a metric function for distance such as $d(x, \mathcal{Y}) = \inf_{y \in \mathcal{Y}} ||x - y||^2$. The adjoint approach [21] allows for parameter θ updates by gradient descent, which can be described as the following dynamical system:

$$\frac{d\mathcal{L}(\mathbf{X}(T))}{d\theta} = \int_{t_0}^{T} \psi(t) \frac{\partial f(\mathbf{X}(t), \theta)}{\partial \theta} dt, \tag{3}$$

where $\psi(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{X}}$ is the adjoint state that satisfies the adjoint equation:

$$\begin{aligned} \frac{d\psi(t)}{dt} &= -\psi(t) \frac{\partial f(\mathbf{X}(t), \theta)}{\partial \mathbf{X}}, \\ \psi(T) &= \frac{\partial \mathcal{L}}{\partial \mathbf{X}}(T). \end{aligned} \tag{4}$$

To sum up the points above, NODEs solve (2) to obtain the predicted value in forward inference while solving Eqs. (3) and (4) in backward training utilizing black-box ODE solvers [3,14] to update the weights.

Nevertheless, when addressing extensive machine learning models or learning long-term dependencies in intricate dynamic systems [9,17], NODEs necessitate a substantial quantity of function evaluations during both inference (1) and training (4) to attain a minimal error tolerance of ODEs, thereby impacting the computational efficiency and efficacy of ODE-based models. To alleviate the diminished efficiency caused by elevated function evaluations, dimension-lifting techniques can be incorporated with NODEs to augment the representational capability of such models [7,9]. Also, some authors propose incorporating physical laws or acceleration techniques inspired by physics [5,12]. Let us describe some of the proposed modifications in more detail below.

P-Node Neural ODE and RNN. This method combines two models, vanilla Neural ODE and RNN, giving the possibility to flexibly combine advantages of continuous and discrete models [18]. The hidden state is defined either by Eq. (1) in the case of Neural ODE or by

$$\mathbf{X_t} = \sigma(W_X \mathbf{X}_{t-1} + W_{\mathcal{Y}} \mathcal{Y}_t + b),$$

in the case of RNN. Here $\mathcal{Y}_t$ is the input data, W_X and $W_{\mathcal{Y}}$ are the weight matrices, b a bias term, σ is an activation function.

The choice between these two models is done according to the probability: RNN is chosen with probability $1 - p$, while Neural ODE – with p.

It is theoretically proven that when $p(t) = 1/t$ (here t denotes training time), the model's hidden state may be considered as a random process with all of its properties. This probability selection is further justified by the fact that the neural ODE model trains faster than the RNN during the first epochs. Furthermore, this switching allows for fewer function evaluations, lowering total training time.

Augmented Neural ODE (ANODE). This model augments the input state with extra dimensions to learn richer dynamics [7,9]. For example, a 2-dim input might be expanded to 5-dim before solving the ODE:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{X}(t)\\ \hat{\mathbf{X}}(t)\end{bmatrix} = \begin{bmatrix}f(\mathbf{X}(t), t, \theta)\\ g(\hat{\mathbf{X}}(t), t, \phi)\end{bmatrix},$$

where $\hat{\mathbf{X}}(t)$ is an augmented state. Thus, ANODE addresses original Neural ODEs problems related to complex transformations (e.g., rotations, separations) and input space's topology due to limited state space. This provides a variety of benefits; among those are improved expressivity: thanks to additional dimensions, the system is capable of modeling more complex flows. The second is the absence of over-parameterization: expanding state space helps to avoid increasing network width/depth. Additionally, ANODE outperforms vanilla Neural ODEs on some tasks like image classification that require complex feature transformations.

Hamiltonian Neural Networks (HNN). HNN enforces Hamiltonian mechanics (energy conservation) in the ODE structure by learning dynamics that preserve symplectic structure (e.g., position and momentum) [12]. In Hamiltonian mechanics q and p represent generalized coordinates and momenta, respectively. These variables are used to describe the state of the system and its dynamics:

$$\dot{q}(t) = \frac{\partial H}{\partial p}, \quad \dot{p}(t) = -\frac{\partial H}{\partial q},$$

where $H(q, p, t)$ is the Hamiltonian of the system. The Hamiltonian H is a function that describes the total energy of the system. In HNN, the Hamiltonian is parameterized by a neural network:

$$H(q, p) = \mathrm{NN}(q, p; \theta),$$

where NN is a neural network with parameters θ. This helps to model systems where it is important to preserve the physical laws while training, e.g., the energy conservation law.

Thus, HNN addresses original Neural ODE limitations; for instance, with physical plausibility, original Neural ODEs may violate physical laws. Also, standard models require large datasets to learn dynamics, while HNNs exploit the known physics priors. Thanks to physical constraints, HNN exhibits long-term stability of the predictions. All these facts result in better generalization ability and interpretability.

Neural Controlled Differential Equations (NCDE). NCDE extends Neural ODEs to handle controlled dynamical systems, where the derivative depends on an external control signal (e.g., time-series inputs) [15]. Governed by:

$$\frac{d\mathbf{X}(t)}{dt} = f(\mathbf{X}(t), u(t), t, \theta),$$

where $u(t)$ is an external control signal. This allows modeling systems where the state depends not only on the current state but also on external input data. The control action $u(t)$ can be represented as a function of time or as the output of another neural network.

Thus, compared to vanilla Neural ODE, NCDE provides an explicit mechanism to incorporate external inputs handling also time-varying, discontinuous, or sparse control signals. Thanks to the mentioned points, NCDE captures dependence on the entire history of controls (like RNNs but continuous) and models sequential data with irregular sampling.

2.2 TM-Solver

Consider the autonomous system of ordinary differential equations:

$$\frac{d\mathbf{X}}{dt} = f(\mathbf{X}(t)). \tag{5}$$

Here t is an independent variable (time), $\mathbf{X} \in \mathbb{R}^n$ is a n-dimensional state of the system, and $f : \mathbb{R}^n \to \mathbb{R}^n$ a smooth vector field. A non-autonomous system $\frac{d\mathbf{X}}{dt} = f(\mathbf{X}(t), t)$ can be brought to this form by adding a variable $X_{n+1} = t$ and appending the equation $\frac{d}{dt}t = 1$.

For the fixed time t, $\mathbf{Y}(t)$ can be regarded as a function of its initial value $\mathbf{X}(t_0) = \mathbf{X}_0$:

$$\mathbf{X}(t, \mathbf{X}_0) = \mathcal{M}_F^t(\mathbf{X}_0) = \mathbf{X}_0 + \int_{t_0}^{t} F(\tau, \mathbf{X}(\tau))d\tau. \tag{6}$$

Mapping $\mathcal{M}_F^t$ defines the system's state propagator from the time moment t_0 to t. In general, we pick a numerical integrator Φ_h that approximates $\mathcal{M}_F^t(\mathbf{X}_0)$ and compose it to produce the numerical solution. The pleiades of traditional solvers are based on dividing the interval $[t_0, t]$ into several steps and numerical computing integrals in (6). For example, the common choice is the Runge-Kutta method [2]:

$$\mathbf{X}(t_{k+1}) = \mathbf{X}(t_k) + h\sum_{i=1}^{I} b_i f(t_k, \mathbf{X}(v_i)), \tag{7}$$

$$v_i = \mathbf{X}(t_k) + h\sum_{j=1}^{I} a_{ij} f(v_j), i = 1..I.$$

Here h is a discrete step and coefficients a_{ij} and b_i with $i, j = 1..I$ fully describe particular method. For instance, the simplest procedure is the explicit Euler method, where a single step takes the following form:

$$\mathbf{X}(t_{k+1}) = \mathbf{X}(t_k) + hF(t_k, \mathbf{X}(t_k)). \tag{8}$$

The formula (7)–(8) shows that the state vector values at the previous time step are typically not multiplicatively separable in the nonlinear function f. This indicates that numerical computation should be repeated for each new initial condition when solving a forward problem for (5).

TM-solver aims to build an expression of the flow (similar idea to [4]) $\mathcal{M}_F^t(\mathbf{X}_0)$ in the polynomial form

$$\mathbf{X}(t) = \sum_{i=1}^{m} R^{1i}(t, t_0)\mathbf{X}_0^{[i]}, \tag{9}$$

where $\mathbf{X}^{[k]}$ is k-th Kronecker power of the vector $\mathbf{X}$ (means $\mathbf{X} \otimes \mathbf{X}^{[k-1]}$ after removing duplicate terms). The advantage of representation (9) is that it is linear with respect to the matrices $R^{1i}(t, t_0)$.

The procedure to construct unknown $R^{1i}(t, t_0)$, $i = \overline{1..m}$ matrices based on the system (5) is described in detail in [10].

Using such a method for ODE solution in NODE can bring several potential advantages:

- (Closed-Form Efficiency) Polynomial solutions provide exact or approximate closed-form expressions, eliminating iterative numerical integration. This reduces computational costs during inference; additionally, evaluation of polynomials is typically faster than solving an ODE step-by-step.
- (Simpler Gradient Computation) Gradients might be calculated simply using symbolic differentiation of a polynomial, eliminating the requirement for adjoint sensitivity approaches. This prevents memory-intensive backpropagation using solver stages.
- (Interpretability) In contrast to black-box numerical solvers, polynomial solutions are explicit and provide insights into learnt dynamics.
- (Stability) Polynomial approximations can help to avoid the stability concerns that come with iterative approaches, particularly for stiff systems.

3 Model

Despite the mentioned advantages of TM-solver, when it is applied to Neural ODE there could be some potential problems [8]. For instance, most ODEs do not admit exact polynomial solutions; also, Neural ODEs learn arbitrary dynamics via flexible neural networks. If we force the network to produce ODEs solvable by polynomials, it would severely limit expressivity, undermining their universal approximation capability. At the same time, higher accuracy might require high-degree polynomials, thus introducing computational complexity and numerical instability. Furthermore, fixed polynomial forms lack the adaptive step-sizing required to handle irregular or stiff dynamics, which is a strength of conventional solvers such as dopri5.

Thus, inspired by the p-Node Neural ODE + RNN, the following hybrid architecture can be proposed:

1. Hybrid solver system:
   - *TM-solver*: constructs solution in the form (9) according to the deterministic procedure for the given ODE;
   - *Dopri*: a traditional adaptive ODE solver (e.g., dopri5) for regions where polynomial approximations fail;
   - *Gating Network*: a classifier based on MLP that decides which solver to use for a given input/time window.
2. Neural ODE core: the main network that defines dynamics (1).

The training procedure includes the following steps aside from those provided by the original model for each input/time window $[t_i, t_{i+1}]$:

- Extract dynamics features $\varphi_i = \text{feature_extractor}(f_\theta, \mathcal{Y}_i, t_i)$
- Compute gating probability $p_i = \text{gating_network}(\varphi_i)$
- **if** $p_i > \tau$ (threshold): use TM-solver to predict $\mathcal{Y}_{i+1}$
- **else**: use Dopri to compute $\mathcal{Y}_{i+1}$.

4 Results

The mentioned models in the first section were compared to the proposed TM-solver-based NODE in this paper for two testing scenarios: univariate and multivariate time-series prediction.

4.1 Univariate Time-Series Prediction

We consider two datasets in the univariate time-series scenario: synthetic data defined with a simple formula:

$$\sin(t) + 0.1 \cdot Random(t),$$

and the time-series data of weather conditions in Delhi [1]. For testing, we chose humidity index only.

Table 1 presents the results of experiments on synthetic data and real data (humidity in Delhi) [1]. For each method, the training time and the value of the loss function (MSE) at the test set are indicated.

Table 1. Result of experiments with univariate time series prediction

Method	Synthetical data		Real data	
	train time (s)	MSE	train time (s)	MSE
Neural ODE	12.21	0.0127	31.05	0.0157
P-Node + RNN	10.27	0.0142	30.01	0.0126
Augmented ODE	9.58	0.0123	40.18	0.0089
HNN	7.05	0.0135	33.97	0.0104
NCDE	11.94	0.0193	41.88	0.0150
TM-solver-NODE	10.21	0.0134	35.03	0.0122

Figures 1–2 illustrate the best of the analyzed models (ANODE) with respect to the MSE metric.

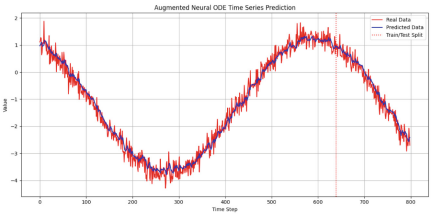


Fig. 1. ANODE for synthetic data.

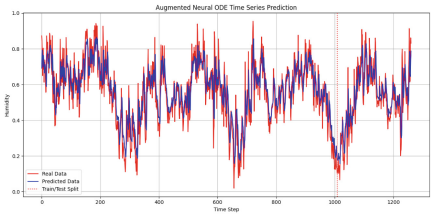


Fig. 2. ANODE for humidity data.

4.2 Multivariate Time-Series Prediction

As for an example of multivariate time-series data, we considered the real dataset collected at the vibration testbed at SPbSU. The dataset consists of acceleration of vibrations measured within three perpendicular directions (X, Y, Z) for 4 points of interest. Figure 3 illustrates the phase portrait of the vibration for the first point.

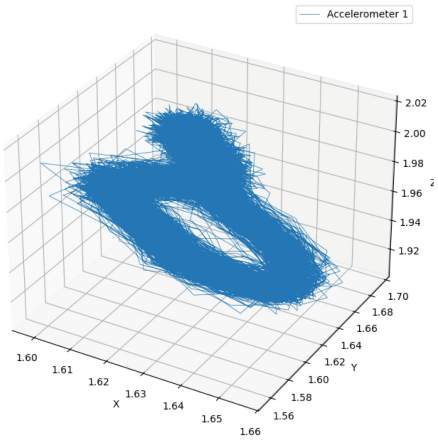


Fig. 3. Vibration phase portrait for the first point of measurements.

Table 2 aggregates the computational results for the different models being investigated in the current research. The best results, both with respect to the training time and MSE Hamiltonian neural network demonstrates. Figure 4 shows the prediction for the vibration signal with HNN model.

Table 2. Result of experiments with multivariate time series prediction

Method	train time (s)	MSE
Neural ODE	31.61	0.0159
P-Node + RNN	55.17	0.0124
Augmented ODE	90.71	0.0123
HNN	2.77	0.0095
NCDE	32.84	0.0247
TM-solver-NODE	11.04	0.0112

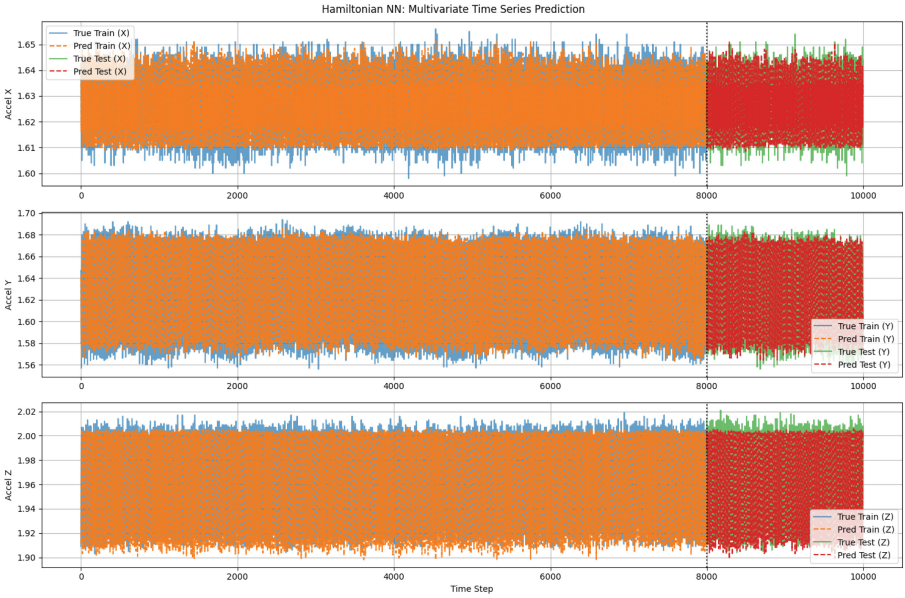


Fig. 4. HNN for the vibration acceleration signal.

5 Conclusion

In this study, we examined how well neural ODE-based models performed in time series forecasting, focusing on how the Taylor Mapping (TM) solver enhanced numerical integration. We considered many versions, such as Neural ODE, P-Node + RNN, Augmented Neural ODE (ANODE), Hamiltonian Neural Networks (HNN), Neural Controlled Differential Equations (NCDE), and the proposed TM-solver-NODE, using both synthetic and real univariate and multivariate datasets. Thanks to the enlarged state space, ANODE achieved the greatest prediction accuracy. Nevertheless, TM-solver-NODE showed competitive performance with clear benefits: its adaptable truncation balanced accuracy and computational cost, its polynomial-based solver structure allowed for interpretable local dynamics, and its smooth integration with automatic differentiating frameworks allowed for effective training.

The proposed strategy to unify the advantages of both approaches is a hybrid design that uses a gate network to dynamically select between a TM solver and conventional solvers (Dopri). The analytical rigor of the TM solver makes it a useful tool for the cases when interpretability and stability are required, including modeling stiff systems or irregularly sampled data, even if HNN and P-Node+RNN are better in terms of training efficiency. Future research might introduce an adaptive truncation criterion to higher-order approximations in TM solver method. Due to the combination of computationally efficient and interpretable solutions joint with the flexibility of continuous modeling, the proposed

approach demonstrates an interesting potential of hybrid numerical approaches to develop neural ODEs.

Future directions include optimizing the Neural ODE type models for distributed memory systems and exploring its synergy with GPU-accelerated automatic differentiation frameworks, further bridging the gap between machine learning and scientific computing.

References

1. Delhi Weather Data. https://www.kaggle.com/datasets/mahirkukreja/delhi-weather-data
2. Solving Ordinary Differential Equations I, Springer Series in Computational Mathematics, vol. 8. Springer Berlin Heidelberg, Berlin, Heidelberg (1993). https://doi.org/10.1007/978-3-540-78862-1
3. Atkinson, K., Han, W., Stewart, D.E.: Numerical Solution of Ordinary Differential Equations. John Wiley & Sons (2009)
4. Biloš, M., Sommer, J., Rangapuram, S.S., Januschowski, T., Günnemann, S.: Neural Flows: Efficient Alternative to Neural ODEs (2021). https://doi.org/10.48550/arXiv.2110.13040, http://arxiv.org/abs/2110.13040, arXiv:2110.13040 [cs]
5. Botev, A., Jaegle, A., Wirnsberger, P., Hennes, D., Higgins, I.: Which priors matter? Benchmarking models for learning latent dynamics (2021). https://doi.org/10.48550/arXiv.2111.05458, http://arxiv.org/abs/2111.05458, arXiv:2111.05458 [stat]
6. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural Ordinary Differential Equations (2019). https://doi.org/10.48550/arXiv.1806.07366, http://arxiv.org/abs/1806.07366, arXiv:1806.07366 [cs]
7. Dupont, E., Doucet, A., Teh, Y.W.: Augmented Neural ODEs. In: Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)
8. Fronk, C., Petzold, L.: Interpretable polynomial neural ordinary differential equations. Chaos **33**(4) (2023). https://doi.org/10.1063/5.0130803, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10076068/
9. Gholami, A., Keutzer, K., Biros, G.: ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs (2019). https://doi.org/10.48550/arXiv.1902.10298, http://arxiv.org/abs/1902.10298, arXiv:1902.10298 [cs]
10. Golovkina, A., Kozynchenko, V.: Neural network representation for ordinary differential equations. In: Artificial Intelligence in Models, Methods and Applications, Studies in Systems, Decision and Control, vol. 457. Springer International Publishing (2022)
11. Golovkina, A., Kozynchenko, V., Kulabukhova, N.: Reconstruction and identification of dynamical systems based on Taylor maps. In: Gervasi, O., et al. (eds.) Computational Science and Its Applications – ICCSA 2021, vol. 12956, pp. 360–369. Springer International Publishing, Cham (2021).https://doi.org/10.1007/978-3-030-87010-2_26, https://link.springer.com/10.1007/978-3-030-87010-2_26, series Title: Lecture Notes in Computer Science
12. Greydanus, S., Dzamba, M., Yosinski, J.: Hamiltonian Neural Networks (2019). https://doi.org/10.48550/arXiv.1906.01563, arXiv:1906.01563 [cs]
13. Huang, Z., Sun, Y., Wang, W.: Learning Continuous System Dynamics from Irregularly-Sampled Partial Observations (2020). https://doi.org/10.48550/arXiv.2011.03880

14. Kidger, P.: On Neural Differential Equations (2022). https://doi.org/10.48550/arXiv.2202.02435, http://arxiv.org/abs/2202.02435
15. Kidger, P., Morrill, J., Foster, J., Lyons, T.: Neural Controlled Differential Equations for Irregular Time Series (2020). https://doi.org/10.48550/arXiv.2005.08926, http://arxiv.org/abs/2005.08926, arXiv:2005.08926 [cs]
16. Klimenko, I., Golovkina, A., Ruzhnikov, V.: Polynomial neural layers for numerical modeling of dynamical processes. In: Gervasi, O., Murgante, B., Rocha, A.M.A.C., Garau, C., Scorza, F., Karaca, Y., Torre, C.M. (eds.) Computational Science and Its Applications – ICCSA 2023 Workshops, vol. 14109, pp. 261–273. Springer Nature Switzerland, Cham (2023https://doi.org/10.1007/978-3-031-37120-2_17, https://link.springer.com/10.1007/978-3-031-37120-2_17, series Title: Lecture Notes in Computer Science
17. Lechner, M., Hasani, R.: Learning Long-Term Dependencies in Irregularly-Sampled Time Series (2020). https://doi.org/10.48550/arXiv.2006.04418, http://arxiv.org/abs/2006.04418, arXiv:2006.04418 [cs]
18. Margasov, A.: Neural ordinary differential equations and their probabilistic extension. In: Proceedings of the Komi Science Centre of the Ural Division of the Russian Academy of Sciences, vol. 6, pp. 14–19 (2021). https://doi.org/10.19110/1994-5655-2021-6-14-19, https://www.elibrary.ru/item.asp?id=47501514
19. Pal, A., Ma, Y., Shah, V., Rackauckas, C.: Opening the Blackbox: Accelerating Neural Differential Equations by Regularizing Internal Solver Heuristics (2022). https://doi.org/10.48550/arXiv.2105.03918, http://arxiv.org/abs/2105.03918, arXiv:2105.03918 [cs]
20. Poli, M., Massaroli, S., Yamashita, A., Asama, H., Park, J.: Hypersolvers: Toward Fast Continuous-Depth Models (2020). https://doi.org/10.48550/arXiv.2007.09601, http://arxiv.org/abs/2007.09601
21. Pontryagin, L.: Mathematical Theory of Optimal Processes. Routledge, 1 edn. (2018). https://doi.org/10.1201/9780203749319, https://www.taylorfrancis.com/books/9780203749319
22. Zhu, A., Jin, P., Zhu, B., Tang, Y.: On Numerical Integration in Neural Ordinary Differential Equations (2022). https://doi.org/10.48550/arXiv.2206.07335, http://arxiv.org/abs/2206.07335, arXiv:2206.07335 [cs]