ADAPTIVE HIGHER ORDER REVERSIBLE INTEGRATORS FOR MEMORY EFFICIENT DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The depth of networks plays a crucial role in the effectiveness of deep learning. However, the memory requirement for backpropagation scales linearly with the number of layers, which leads to memory bottlenecks during training. Moreover, deep networks are often unable to handle time-series data appearing at irregular intervals. These issues can be resolved by considering continuous-depth networks based on the neural ODE framework in combination with reversible integration methods that allow for variable time-steps. Reversibility of the method ensures that the memory requirement for training is independent of network depth, while variable time-steps are required for assimilating time-series data on irregular intervals. However, at present, there are no known higher-order reversible methods with this property. High-order methods are especially important when a high level of accuracy in learning is required or when small time-steps are necessary due to large errors in time integration of neural ODEs, for instance in context of complex dynamical systems such as Kepler systems and molecular dynamics. The requirement of small time-steps when using a low-order method can significantly increase the computational cost of training as well as inference. In this work, we present an approach for constructing high-order reversible methods that allow adaptive time-stepping. Our numerical tests show the advantages in computational speed when applied to the task of learning dynamical systems.

1 INTRODUCTION

Deep neural networks are widely used across various learning tasks (Russakovsky et al., 2015; Esteva et al., 2017), and their depth often plays a crucial role in the effectiveness of learning. These networks have also been shown to be particularly useful in the tasks of learning models of dynamical systems (Chen et al., 2018; Raissi et al., 2018a; Rudy et al., 2019b; Schüssler et al., 2019; Rudy et al., 2019a; Liu et al., 2022; Raissi et al., 2018b). It was recently shown that the use of numerical methods for neural network architectures can provide impressive results with theoretical guarantees (Haber & Ruthotto, 2017; Chang et al., 2018; Celledoni et al., 2021; Maslovskaya & Ober-Blöbaum, 2024). In this work we use the theory of symmetric numerical methods for the construction of a new class of reversible neural networks. The new class allows memory efficient computations of gradients in training and reduced computational costs in learning models of dynamical systems, where the parameters typically need to be identified to high accuracy and the depth of the networks can be very large. Our network architecture constitutes an important contribution to ensure scalability of neural ODEs to high-dimensional dynamical systems that arise, for instance, as discretizations of systems governed by partial differential equations.

The high memory costs of computing the gradient of very deep neural networks using the backpropagation algorithms poses a significant bottleneck in their training, hindering their scalability and efficiency. To address this, a neural ODE approach combined with the adjoint method has been proposed for gradient computations (Chen et al., 2018), which avoids storing intermediate states during forward propagation, potentially making the cost of gradient computation independent of network depth. However, it was quickly realized (Gholaminejad et al.; Zhuang et al., 2020) that using this approach with arbitrary discretization methods leads to incorrect gradients.

The solution of this problem is to use reversible integrators, which ensure accurate gradient computations by reconstructing intermediate states precisely during backward integration. However, symplectic reversible integrators (Chang et al., 2018), which are a large class of well studied integra-

tors, do not allow adaptivity in the step size and require a particular structure in the neural ODE. This makes them unsuitable for use in time-series applications where data appears at irregular intervals, when time-steps need to be decreased adaptively to achieve a prescribed accuracy in the learning of dynamical systems, or when the identified model is used to predict continuous trajectories.

Adaptive time-stepping for numerical integrators for differential equations is a well-established field in numerical analysis (Hairer et al., 2013; Deuflhard & Bornemann, 2002). In adaptive time-stepping, step sizes for an integration step are selected such that an estimate for the local error is below a given error tolerance. In this way, computational cost in numerical integration can be saved when large step sizes are sufficient to obtain accurate results, while step sizes are automatically decreased when required. The only reversible methods compatible with variable time-step selection without losing the reversibility property are asynchronous leapfrog (ALF) (Mutze, 2016), which is based on the classical Verlet method (also known as leapfrog method) (Verlet, 1967), and the reversible Heun method (Kidger et al., 2021).

ALF has been used to construct neural network architectures known as MALI networks (Zhuang et al., 2021). These methods are based on operating on an augmented space: a neural ODE in the original variable z is extended to a larger space and replaced by a neural ODE in (z, v). Both methods are known to be of order of accuracy $(2, 1)$ in (z, v) (Mutze, 2016), which makes them computationally costly in learning tasks that require high accuracy in the integration of the neural ODE, in particular, in learning of dynamical systems. To be able to reach high accuracy, the lower order methods are forced to use small step sizes and, as a result, have higher computational costs. This phenomenon was highlighted in the examples in (Matsubara et al., 2021). Furthermore, we show in C.1 that in parameter identification tasks there is a direct relation between the order of a numerical integrator and the order of accuracy of identified parameters. Therefore, there is a need for higher order reversible methods, which we address in this paper.

Various machine learning techniques have emerged in the past decades for approximating models of dynamical systems (Ghadami & Epureanu, 2022). These include methods based on Gaussian processes (Bouvrie & Hamzi, 2017; Raissi & Karniadakis, 2018; Hamzi & Owhadi, 2021), sparse regression on libraries of basis functions (Brunton et al., 2016; Tran & Ward, 2017; Reinbold et al., 2021), and recurrent neural networks (RNNs) (ichi Funahashi & Nakamura, 1993; Bailer-Jones et al., 1998; Karniadakis et al., 2021). In particular, the neural ODE approach (Chen et al., 2018) identified an important connection between the RNN structure and the numerical methods available for integration of differential equations. This was generalized by universal differential equations in (Rackauckas et al., 2021) for different types of differential equations. Other generalizations include physics informed learning of dynamical systems (Greydanus et al., 2019; Cranmer et al., 2020b; Jin et al., 2020; Chen et al., 2020) and operator approximation (Chen & Chen, 1995; Lu et al., 2021; Lin et al., 2023; Boullé & Townsend, 2024). If the training data consists of time-series data that corresponds to a constant time-step, neural ODEs can be trained with a low order method and time-series can be predicted with high accuracy provided that the trained neural ODE is integrated with the same integrator that was used during training (Zhu et al., 2021; David & Méhats, 2023; Offen & Ober-Blöbaum, 2022; Ober-Blöbaum & Offen, 2023). However, in realistic examples, snapshots of trajectories with variable time-steps need to be processed (Raissi et al., 2018b; Rudy et al., 2019b; Liu et al., 2022). Moreover, a discretization-independent prediction of the system's evolution is often desired and learning of underlying differential equations requires high accuracy in the learned parameters, which requires high accuracy simulation.

We demonstrate that our approach can be applied to high-dimensional dynamical systems, including those arising from discretizations of partial differential equations. This is a highly active research area. Other approaches in this context include model order reduction based techniques and operator inference (see the review (Kramer et al., 2024) or e.g. (Sharma et al., 2024; 2023; Allen-Blanchette et al., 2020; Mason et al., 2022)), or structure-preserving approaches for discrete field theories (Qin, 2020; Offen & Ober-Blöbaum, 2024; Offen, 2024).

The main contribution of this paper is the development of a methodology to construct reversible neural networks based on higher order numerical methods. First, we prove that, in contrast to the analysis in (Mutze, 2016), ALF is of order 2 in both (z, v) at even time-steps. Using this property and the theory of composition methods, we construct a class of reversible networks of any even order. A particular architecture based on 4th order networks is compared to the already known ALF method

on examples of learning dynamical systems. The comparison of the two methods with adaptive time-stepping shows that the proposed higher order method is computationally more efficient.

2 BACKGROUND

2.1 NEURAL ODE

Assume that an unknown function $\mathcal{F} : X \to Y$ is approximated by a neural network based on training data $\{x_i, y_i = \mathcal{F}(x_i)\}_{i=1}^n$. The neural ODE approach to the deep network design employs the idea of continuous-depth networks and their discretization by a numerical method. The continuous-depth network is defined as a flow of a neural ODE of the form

$$\dot{z}(t) = f(z(t), \theta(t)), \quad z(0) = (x_1, \ldots, x_n), \tag{1}$$

on the time interval $[0, T]$ for some vector field f. Discretization of a neural ODE with a numerical method of step size h is defined as follows

$$\begin{aligned} &z_{j+1} = \sigma_h(z_j, \theta_j), \quad j = 0, \ldots, N-1, \\ &z_0 = z, \end{aligned} \tag{2}$$

where z_j is the value of the feature variable at the jth layer with the total number of layers N and $z = (x_1, \ldots, x_n)$. The step size can be chosen in an adaptive manner for each layer and it is not considered as an optimization parameter. The learning problem is to find parameters $\{\theta_j\}_{j=0}^{N-1}$ which lead to the best approximation of $\mathcal{F}$. The parameters are usually found as a solution of the following optimization problem.

$$\begin{aligned} &\min_{\{\theta_j\}} J = L(z_N, y) \\ &z_{j+1} = \sigma_h(z_j, \theta_j), \quad j = 0, \ldots, N-1, \\ &z_0 = z, \end{aligned} \tag{3}$$

where $L(\cdot, \cdot)$ is a loss function which measures the distance between the output of the network and the training data $y = (y_1, \ldots, y_N)$.

Because of the connection between network equation 2 and the corresponding neural ODE equation 1, there exists a continuous counterpart of equation 3 which makes equation 3 an approximation of an optimal control problem of the form

$$\begin{aligned} &\min_{\theta(t)} J = L(z(T), y) \\ &\dot{z}(t) = f(z(t), \theta(t)), \quad t \in [0, T], \\ &z(0) = z. \end{aligned} \tag{4}$$

Solutions of equation 3 are usually found using methods based on gradient descent. Such methods require computations of the gradients of the loss function with respect to all parameters $\{\theta_j\}_{j=0}^{N-1}$.

2.2 METHODS OF GRADIENT COMPUTATIONS

Backpropagation Let $J = L(z_N, y)$. By the chain rule, the gradient is given by

$$\frac{\partial}{\partial \theta_j} J = \nabla_z L(z_N, y)^\top \frac{\partial z_N}{\partial z_{N-1}} \cdots \frac{\partial z_{j+2}}{\partial z_{j+1}} \frac{\partial z_{j+1}}{\partial \theta_j}.$$

To avoid computationally expensive multiplication of large matrices, the formula is evaluated from the left to the right (backpropagation). This requires that the intermediate values z_j ($j = 1, \ldots, N$) are available. Their size corresponds to the width of the layer and their number to the network's depth N.

Adjoint method This approach is based on the formula for the continuous gradients via adjoint variables $p(t)$. The adjoint variables are the solution of

$$\dot{p} = -\frac{\partial}{\partial z} f(z(t), \theta(t))^\top p, \qquad p(T) = \nabla L(z(T), y), \tag{5}$$

on time interval $[0, T]$ and the differential of $J(\theta)$ with respect to $\theta(t)$ is calculated as follows

$$DJ(\theta(t)) = p^\top(t) \frac{\partial}{\partial \theta} f(z(t), \theta(t)). \tag{6}$$

This method is particularly interesting when the available memory is limited. In this case, the forward propagation is done to obtain the value of $z(T)$ and $\nabla L(z(T), y)$, there is no need to save the intermediate values z_j and the computational graph, because the backpropagation is realized by integrating the equations for $(z(t), p(t))$ numerically backward in time. The correct discretization of the state-adjoint equations leads to the same expression for the gradient as the one obtained in the backpropagation approach. Still, the values of z_j obtained by the numerical integration backward do not always coincide with the values obtained in the forward pass. This is why this approach usually leads to inexact gradients. This issue can be solved by considering reversible networks.

Checkpointing An alternative approach for memory reduction is checkpointing (Gholaminejad et al.), which stores a few intermediate states for a regeneration of the computation graph. In case of adaptive time-stepping it was described in (Zhuang et al., 2020) and further improved in (Matsubara et al., 2021) for the class of Runge-Kutta methods. This is a highly efficient approach, when used in combination with higher order methods. In this case, a small number of checkpoints is required to get a high accuracy in learning. However, when the learning task is to learn a dynamical system from long trajectories of complex systems, then the number of checkpoints becomes large and can lead to memory leaks. This is why it is important to have an alternative approach based on reversible networks with the memory costs independent from the given task.

2.3 Reversible neural network

A reversible network is a network with the property that there exists an explicit formula for backward propagation $z_{j+1} \mapsto z_j$ that exactly inverts a forward pass $z_j \mapsto z_{j+1}$, i.e., there exists a map $\tilde{\sigma}(\cdot)$, such that $z_j = \tilde{\sigma}(z_{j+1})$ and $z_{j+1} = \sigma_h(\tilde{\sigma}(z_{j+1}), \theta_j)$. It requires that the time-steps $t_0, \dots, t_N$ have been stored when the forward propagation was computed but it does not require storage of the (potentially very high-dimensional) intermediate values z_j. The discretizations of neural ODE equation 2, which admit this property are called reversible methods. As for now, there are only two known reversible methods allowing for an adaptive choice of the step size, namely, asynchronous leapfrog (Zhuang et al., 2021) and reversible Heun (Kidger et al., 2021).

This notion of reversibility for neural networks needs to be contrasted with the notion of time-reversibility or symmetry for numerical integrators. In the context of neural networks, reversibility means that there exists an explicit, efficient formula to invert the forward pass. In numerical integration theory, a time-reversible or symmetric numerical integrator is a formula to advance the solution of an ordinary differential equation by time h such that its inverse is obtained by substituting h by $-h$ (Hairer et al., 2006, II.3). In case of the dynamical system $f(z(t), \theta(t))$ from equation 2, if the discretization by a numerical method $z_{j+1} = \sigma_h(z_j, \theta(t_j))$ is symmetric, then it implies $z_j = \sigma_{-h}(z_{j+1}, \theta(t_{j+1}))$, or equivalently, $z_j = \sigma_{-h}(z_{j+1}, \theta_{j+1})$. Therefore, we can set $\tilde{\sigma}(\cdot) = \sigma_{-h}(\cdot, \theta_{j+1})$, which implies that the method is reversible. The symmetry of integrators is beneficial in the context of the article as inverses of the methods required for backpropagation take simple forms and efficient classical techniques to construct higher order methods (Hairer et al., 2006, II.4) apply.

Asynchronous Leapfrog (ALF) method As the optimization parameter $\theta(t)$ in the dynamics $f(z(t), \theta(t))$ depends on time, it can be seen as a part of f and written simply $f_\theta(z(t), t)$. The ALF method requires the augmentation of the pair of state and time (z, t) with the velocity v which approximates $f_\theta(z(t), t)$. We denote a step forward of the ALF method with the step size h by Ψ_h^{ALF}. Given a triple (z_j, v_j, t_j) and a step size h, the algorithm generates in the forward pass the next values $(z_{j+1}, v_{j+1}, t_{j+1})$ as follows

$$\begin{pmatrix} z_{j+1} \\ \\ v_{j+1} \end{pmatrix} = \Psi_h^{ALF}(z_j, v_j, t_j) = \begin{pmatrix} z_j + h f_\theta(z_j + \frac{h}{2} v_j, t_j + \frac{h}{2}) \\ \\ 2 f_\theta(z_j + \frac{h}{2} v_j, t_j + \frac{h}{2}) - v_j \end{pmatrix}, \quad t_{j+1} = t_j + h. \tag{7}$$

The step backward calculates (z_j, v_j, t_j) from $(z_{j+1}, v_{j+1}, t_{j+1})$ as follows

$$\begin{pmatrix} z_j \\ \\ v_j \end{pmatrix} = \Psi_{-h}^{ALF}(z_{j+1}, v_{j+1}, t_{j+1}), \quad t_j = t_{j+1} - h. \tag{8}$$

If the method is initialized at $(z_0, f(z_0, t_0), t_0)$, then ALF is a second order method in z and first order method in v, as it was shown in (Zhuang et al., 2021). The order of accuracy is the order in step size h of the error of the numerical flow compared with the exact flow of the ODE (Hairer et al., 2006). Notice that ALF is symmetric by definition.

The reversible Heun method is another reversible method based on state-space augmentation and was introduced in (Kidger et al., 2021). The method was shown to be also of second order in z and first order in v. In addition, it is a symmetric method. In the following part of the paper we will concentrate on the construction of higher order methods based on ALF, but the same can be also applied to the reversible Heun method.

3 NEW REVERSIBLE ARCHITECTURES

A general approach in numerical analysis to construct higher order symmetric methods is by composition (Hairer et al., 2006; 2013; Blanes et al., 2024a). In this case one can start with a lower order numerical method and construct a new method by composition of the lower order method with a particular choice of step sizes. This construction leads to a method of higher order of accuracy.

Numerical experiments, see Appendix A, show that ALF is of second order in the error with respect to a high accuracy solver in both z and v. This is surprising as the order of consistency of ALF in v is only 1 (Zhuang et al., 2021). Indeed, as we show below, a method consisting of a composition of two steps of ALF has order of consistency 2 in (z, v), which explains the convergence behaviour. This observation is required to apply theory for composition methods (Hairer et al., 2013; Blanes et al., 2024a; Yoshida, 1990) to (two steps of) ALF.

Theorem 3.1. *Composition of two steps of ALF methods, i.e.* $\Psi^{ALF}_{h/2} \circ \Psi^{ALF}_{h/2}$*, applied to* $\dot{z} = f(z,t)$ *provides second order accurate approximations of position* z *and velocity* $v = \dot{z}$*.*

The proof of the theorem is based on comparing the terms in the Taylor series of the exact flow of a differential equation and the numerical flow obtained by composition of two steps of the ALF method. We refer to the Appendix A.1 for the computations. The composition of two steps of the ALF method, each with time-step $\frac{h}{2}$, will be called ALF2 and denoted by Ψ^{ALF2}_h. Now we are in a classical situation, with ALF2 a one step reversible method of even order and we can apply the composition methods to construct higher order methods. In this work we consider the Yoshida approach (Yoshida, 1990). Yoshida composition permits to construct methods of a higher accuracy by composing numerical methods of order $2k$ for some integer k. It is defined by a symmetric composition of the same method Ψ^{2k} ($2k$ stands for the order) with different step sizes

$$\Psi^{\mathrm{Y}}_h = \Psi^{\mathrm{2k}}_{ah} \circ \Psi^{\mathrm{2k}}_{bh} \circ \Psi^{\mathrm{2k}}_{ah}$$

with time-steps defined by

$$a = \frac{1}{2 - 2^{\frac{1}{2k+1}}}, \qquad b = 1 - 2a.$$

Theorem 3.2 ((Yoshida, 1990))**.** *Yoshida composition of a reversible method* Ψ^{2k} *of order* $2k$ *has order* $2k+2$ *and is reversible.*

Yoshida is not the only composition method that can be used, another possible approach is Suzuki composition (Suzuki, 1991). Several approaches are reviewed in (Hairer et al., 2006; Blanes et al., 2024b).

Remark 3.3. The approach based on Yoshida composition might require checkpoints in case of certain neural ODEs, e.g., when the learning task is to learn a dispersive partial differential equation such as heat equation. This is because the Yoshida composition forces the use of negative time-steps, which can be a problem in dissipative cases, where it can lead to instability.

In the following, we will denote by $\Psi^{Y,2k}_h$ the higher order methods obtained by Yoshida composition of ALF2, where $2k$ is the order of the method. The constructed higher order reversible method can be used for the construction of a reversible network. In this case, the step forward and the step backward are defined recursively based on the steps forward and backward of a lower order method $\Psi^{Y,2k-2}_h$. The starting method of order 2 is the ALF2 method, i.e. $\Psi^{Y,2}_h = \Psi^{ALF2}_h$ by abuse of notation.

Adaptive stepping One of the main advantages in the construction of reversible methods based on ALF is that they allow for adaptive step sizes (Hairer et al., 2006). This can be done in the same manner as for ALF (Zhuang et al., 2021), where the main idea is to delete the computational graph and all the variables needed for the step size computations and only the value of the accepted new step size h_j is saved. As a result, values $h_1, \ldots, h_N$ are saved and then accessed in the integration

Algorithm 1 Step forward of $2k$-th order Yoshida

1. Input: (z_j, v_j, t_j, h_j)
2. Set $a = 1/(2 - 2^{\frac{1}{2k+1}}),\ b = 1 - 2a.$
3. Set $(\tilde{z}_1, \tilde{v}_1) = \Psi^{Y,2k-2}_{ah_j}(z_j, v_j, t_j),\ \tilde{t}_1 = t_j + ah_j$
4. Set $(\tilde{z}_2, \tilde{v}_2) = \Psi^{Y,2k-2}_{bh_j}(\tilde{z}_1, \tilde{v}_1, \tilde{t}_1),\ \tilde{t}_2 = \tilde{t}_1 + bh_j$
5. Set $(z_{j+1}, v_{j+1}) = \Psi^{Y,2k-2}_{ah_j}(\tilde{z}_2, \tilde{v}_2, \tilde{t}_2),\ t_{j+1} = \tilde{t}_2 + ah_j$

if adaptive time-stepping **then**

6a. compute the error of z_{j+1}, v_{j+1} w.r.t. the output of a $(2k+1)$st order integration method

6b. compute the new h_{j+1} following (Hairer et al., 2006)

else

6. $h_{j+1} = h_j$

end if

7. Output: $(z_{j+1}, v_{j+1}, t_{j+1}, h_{j+1})$

backward needed for gradient computations. This can be done in exactly the same manner for our Yoshida-based methods. The resulting steps forward and backward are summarized in Algorithm 1 and Algorithm 2.

Remark 3.4. Notice that even though the reversible Heun method was proved to be of order $(2, 1)$ in (z, v) in (Kidger et al., 2021), it was noted in the same paper that it gains the second order in both variables at even steps. This implies that the composition approach can be used in this case as well.

Algorithm 2 Step backward of $2k$-th order Yoshida

1. Input: $(z_{j+1}, v_{j+1}, t_{j+1}, h_{j+1})$
2. Set $a = 1/(2 - 2^{\frac{1}{2k+1}}),\ b = 1 - 2a.$
3. Set $(\tilde{z}_1, \tilde{v}_1) = \Psi^{Y,2k-2}_{-ah_{j+1}}(z_{j+1}, v_{j+1}, t_{j+1}),\ \tilde{t}_1 = t_{j+1} - ah_{j+1}$
4. Set $(\tilde{z}_2, \tilde{v}_2) = \Psi^{Y,2k-2}_{-bh_{j+1}}(\tilde{z}_1, \tilde{v}_1, \tilde{t}_1),\ \tilde{t}_2 = \tilde{t}_1 - bh_{j+1}$
5. Set $(z_j, v_j) = \Psi^{Y,2k-2}_{-ah_{j+1}}(\tilde{z}_2, \tilde{v}_2, \tilde{t}_2),\ t_j = \tilde{t}_2 - ah_{j+1}$
6. Set h_j from $h_1, \ldots, h_N$ obtained in the integration forward
7. Output: (z_j, v_j, t_j, h_j)

Gradient computations The augmentation of the feature space leads to the new variable which we denote by $\phi = (z, v)$. Then, the learning problem is formulated as follows with $P_z(\phi)$ projection of ϕ to z

$$\begin{aligned} &\min_{\{\theta_j\}} J = L(P_z(\phi_N), y) \\ &\phi_{j+1} = \Psi^{Y,2k}_h(\phi_j, \theta_j), \quad j = 0, \ldots, N-1, \\ &\phi_0 = (z, f(z, \theta_0)). \end{aligned} \tag{9}$$

Following (Griesse & Walther, 2004), the discrete version of equation 5 associated with equation 9 is given by

$$(\lambda_N)^\top = \nabla L(\phi_N), \quad \lambda_j = \left(\frac{\partial \phi_{j+1}}{\partial \phi_j}\right)^\top \lambda_{j+1}, \tag{10}$$

and the gradients are computed by

$$\frac{\partial J(\theta)}{\partial \theta_j} = \lambda_{j+1}^\top \frac{\partial \phi_{j+1}}{\partial \theta_j}. \tag{11}$$

The adjoint method for the gradient computation as in the MALI network (Zhuang et al., 2021) and the reversible Heun network (Kidger et al., 2021) is based on the propagation $\phi_j \to \phi_{j+1}$ and automatic

differentiation for the computation of the step backward of the adjoint variable following equation 10. The resulting method of gradient computation is summarized in Algorithm 3. Alternatively, the exact expression of the numerical method governing the adjoint dynamics equation 10 can be obtained, see details in Appendix B. In this case, there is no need to compute $\frac{\partial \phi_{j+1}}{\partial \phi_j}$, which makes the approach computationally more efficient and memory efficient.

Algorithm 3 Computation of gradients

1. Input: training data z_0, initialization of parameters θ, velocity $v_0 = f(z_0, \theta_0)$
2. Propagate through the network using $\Psi_h^{Y,2k}$ to get (z_N, v_N)
3. Set $\lambda_N^z = \nabla L(z_N, y)$ and $\lambda_N^v = 0$

for `j = N-1 to 1` **do**

4. Compute ϕ_j from ϕ_{j+1} using Algorithm 2
5. Compute ϕ_{j+1} from ϕ_j using Algorithm 1 to get the computational graph
6. Compute λ_j from λ_{j+1} using equation 10 and AD to compute $\frac{\partial \phi_{j+1}}{\partial \phi_j}$
7. Compute $\frac{\partial J(\theta)}{\partial \theta_j}$ using equation 11
8. Delete $\lambda_{j+1}, \phi_{j+1}$ and the computational graphs

end for

9. Output: gradients $\frac{\partial J(\theta)}{\partial \theta_j}$ for $j = 1, \ldots, N-1$.

Costs comparison We will use the following notations: d is the dimension of z, T is the length of the time interval in the continuous-depth setting, N is the number of layers, M stands for the number of layers in f, when f is given by a neural network itself, s denotes the number of steps needed for the computation of a time-step in the adaptive step size selection, p is the order of the considered numerical method and r is the number of evaluations of f used in the numerical method (e.g. stages in Runge-Kutta methods or compositions in our approach). We show the comparison of the new proposed approach with the standard backpropagation approach, adjoint method version NODE (Chen et al., 2018), ACA approach (Zhuang et al., 2020) and MALI approach (Zhuang et al., 2021) in Table 1, which extends the Table 1 in (Zhuang et al., 2021). We use big $\mathcal{O}$ notation, when the constants depend on the learning tasks.

Computational costs The compositional structure of the proposed method directly implies that the computation costs for gradient computations are equal to the computational costs by ALF multiplied by r, the number of the compositions. Notice that N depends on the order p of the discretization method and becomes smaller when the order is higher for fixed ε and T. As a result, ALF method needs more time-steps, than higher order methods for $\varepsilon < 1$, which is related to the bias in the learned parameters in the task of identification of the parameters, as explained in Appendix C.1 and illustrated in Figure 4, and to the training error.

Memory costs The gradient computation requires to compute $\frac{\partial \phi_{j+1}}{\partial \phi_j}$ leading to the storage of all the intermediate states involved in the step forward. This increases the memory costs of MALI by a factor r, see Table 1. Notice that the approach presented in Appendix B does not require to store all the intermediate states. Indeed, a step backward of the state-adjoint system is a composition of rescaled steps backward of the ALF method. Therefore, we only need to store one intermediate state obtained in the composition at a time. This makes the method of the same memory cost as MALI. Notice that depending on the depth of the network, adaptive checkpointing as in (Matsubara et al., 2021) can be added. When no checkpoints are needed the behaviour is as in NODE and in the worst case the behaviour is as in the backprop. In general, the number of checkpoints depends on N, which depends linearly on T. Therefore, more checkpoints are needed in case of large T.

4 Experiments

4.1 Parameter identification in dynamical systems

We consider the identification problem of unknown parameters of a dynamical system. The structure of the differential equations is assumed to be known, but some parameters in the equations are unknown. The training data is given by snapshots of trajectories $\{x_l(t_i)\}_{i,l}$ with $l = 1, \ldots L,\ i = 0, \ldots, I$. The goal is to learn the parameters from the given trajectories. This class of problems can

Table 1: Comparison of costs in gradient computations for different approaches

Method	Computational costs	Memory costs	Number of epochs N in function of accuracy ε^*
Backprop	$r \times d \times M \times N \times s \times 2$	$r \times d \times M \times N \times s$	$T \times \mathcal{O}(\varepsilon^{-\frac{1}{p+1}})$
NODE	$r \times d \times M \times N \times s \times 2$	$d \times M$	$T \times \mathcal{O}(\varepsilon^{-\frac{1}{p+1}})$
ACA	$r \times d \times M \times N \times (s+1)$	$d \times (M+N)$	$T \times \mathcal{O}(\varepsilon^{-\frac{1}{p+1}})$
MALI	$d \times M \times N \times (s+2)$	$d \times (M+1)$	$T \times \mathcal{O}(\varepsilon^{-\frac{1}{3}})$
Proposed method	$r \times d \times M \times N \times (s+2)$	$r^{**} \times d \times (M+1)$	$T \times \mathcal{O}(\varepsilon^{-\frac{1}{p+1}})$

* ε is the error tolerance for the estimation of the local error in the stepsize selection
** Memory costs of the proposed method can be reduced to $r = 1$, if the gradients are computed as in AppendixB

be naturally treated using the neural ODE approach. The vector field f in equation 1 is given by the known differential equation and $\theta = \theta_1, \ldots, \theta_s$ is the set of unknown parameters. In this case, the learning problem can be stated in the form of equation 9, where the same $\theta_1, \ldots, \theta_s$ appear all at each layer. The training data $z_0 = (x_1(t_0), \ldots, x_L(t_0))$ stands for the initial points and y includes all the other points of the given trajectories. We denote by $y(t_i)$ the points in y corresponding to trajectories at time t_i for $i = 1, \ldots, I$. The loss have a particular structure in this case as it depends on the intermediate states obtained during the integration of neural ODE, namely, it depends on $(z_{N_1}, \ldots, z_{N_I})$, to measure the distance with the given trajectories points $(y(t_1), \ldots, y(t_I))$. As a result, it takes the form $L = \sum_{i=1}^{I} L_i(z_{N_i}, y(t_i))$. Because of the additive form of the loss, the gradients can be computed as a sum of the corresponding gradients of $L_1, \ldots, L_I$ as follows

$$\frac{\partial L}{\partial \theta_i} = \frac{\partial L_1}{\partial \theta_i} + \cdots + \frac{\partial L_I}{\partial \theta_i}, \qquad i = 1, \ldots, s,$$

where each of the terms in the sum is computed using Algorithm 3. The memory efficiency is still important in this case, because we do not store all the intermediate states at the propagation forward, but only the states which approximate the trajectories at the desired times $t_1, \ldots, t_I$.

Statistical inference In simulation based inference or likelihood-free inference probabilistic methods are employed to identify parameters in models based on repeated forward simulations (Cranmer et al., 2020a). Traditionally, these consider the forward pass as a black box (such as Approximate Bayesian Computation (ABC) (Rubin, 1984; Beaumont et al., 2002)) and do not require differentiability with respect to the model parameters or the inputs. This needs to be contrasted to our proposed neural network architecture, which is designed to circumvent large memory requirements in the computation of gradients when the layers are wide. Indeed, a combination of our architecture with simulation based inference models that do make use of gradients such as (Graham & Storkey, 2017) constitutes an interesting avenue for future research.

4.1.1 KEPLER PROBLEM

We consider the Kepler problem, where the dynamics describes the evolution of the position q and velocity v of a mass point moving around a much heavier body. It is modeled on the 4-dimensional space $x = (q, v) \in \mathbb{R}^2 \times \mathbb{R}^2$. The equations are defined on the time interval $[0, 1]$ as follows

$$\dot{q} = v, \quad \dot{v} = -\frac{\alpha}{\|q\|^3} q, \tag{12}$$

with an unknown parameter $\alpha \in \mathbb{R}$. The training set is given by the initial condition $x(t_0)$ and q-coordinate of 5 points on a trajectory of equation 12 generated with $\alpha = \pi/4 \approx 0.785$, i.e. $\{q(t_i)\}_{i=1}^{5}$. The task is to learn α as accurately as possible. From the training set we form $z_0 = x(t_0)$ and the corresponding $y(t_i) = q(t_i)$ for $i = 1, \ldots, 5$. We set up a learning problem in the form of equation 9 with the loss defined by $L = \sum_{i=1}^{5} \|q_{N_i} - q(t_i)\|^2$ with q_{N_i} projection of z_{N_i} to q-coordinate and N_i the number of time-steps used in the integration from t_{i-1} to t_i. We compare two algorithms for performing the numerical integration during training, namely, ALF and the Yoshida composition of ALF2 of order 4. We write Y4 for the Yoshida composition method for shortness. We test the wall-clock time required to reach loss accuracy 10^{-8} using adaptive methods. The tests are run for different initializations of α in optimization. The results can be seen in Table 2. It can be observed that in all tests, Y4 is at least four times faster than ALF. The reason of the faster training for Y4 is

Table 2: Time to reach accuracy 10^{-8} using adaptive methods in the Kepler problem

Computation time		
Initial value of α	adaptive ALF	adaptive Y4
0.1	7.68 sec	2.42 sec
0.7	4.07 sec	1.02 sec
0.75	3.26 sec	0.803 sec
0.8	2.5 sec	0.44 sec
1.3	8.39 sec	3.85 sec

Table 3: Time to reach accuracy 10^{-4} using adaptive methods in nonlinear oscillators problem

Computation time		
Mean parameter error at initialization	adaptive ALF	adaptive Y4
0.28897009	81608 sec	51720 sec
0.29821727	68645 sec	40646 sec
0.30549358	56764 sec	29990 sec
0.30289593	96301 sec	46524 sec
0.29106813	22161 sec	13790 sec

in using larger step sizes for the forward integration. The lower order method requires smaller step sizes to reach the same accuracy defined by an error tolerance and this leads to more steps in the computation of trajectories. Additional results for the Kepler problem supporting the reasoning can be found in Appendix C.1.

4.1.2 Nonlinear harmonic oscillator

In the second example we consider a system of coupled Duffing oscillators, which describes the movements of a coupled system of mass points attached with springs with nonlinear elastic forces. The dynamics of N mass points is given by the following equations

$$\dot{q}_i = v_i, \quad \dot{v}_i = -a_i q_i - b_i q_i^3 - \sum_{j=1}^{N} e_{i,j}(q_i - q_j), \qquad i = 1, \dots, N, \tag{13}$$

with the condition $e_{i,j} = e_{j,i}$. Positions of N mass points are given by $q = (q_1, \cdots, q_N) \in \mathbb{R}^N$ and velocities by $v = (v_1, \cdots, v_N) \in \mathbb{R}^N$. We set $x = (q, v) \in \mathbb{R}^{2N}$. In the numerical experiments we fix $N = 10$ and assume that parameters $a_i, b_i, e_{i,j} \in \mathbb{R}$ for $i, j = 1, \dots, 10$ are unknown. As a result, equation 13 has dimension 20 with 65 unknown parameters. The training set consists of initial and final positions of 200 trajectories, that is $z_0 = (x_1(t_0), \dots, x_{200}(t_0))$ and $y = y(t_1) = (x_1(t_1), \dots, x_{200}(t_1))$. In this setting, we compare the computational time to reach a certain training accuracy of ALF and Y4 with adaptive time-stepping and the training accuracy of ALF and Y4 with fixed step-size. We present the wall-clock times to reach the training accuracy 10^{-4} in Table 3. The time required by Y4 to reach accuracy 10^{-4} is almost two times smaller which illustrates the lower computational costs of the method. As before, the ALF method with adaptive time-stepping requires smaller step sizes and more steps are used in each epoch of the optimization. Details with additional results confirming the behaviour are presented in Appendix C.2.1.

4.2 Learning of dynamical systems parameterized by neural network

We consider a problem, where a part of the structure of the differential equations is known and the unknown part is approximated using a neural network. Our goal is to find the neural network parameterization such that the resulting trajectories of the system are as close as possible to given trajectories from the training set. As before, the problem can be treated by the neural ODE approach. In this case the vector field f in equation 1 is given by a neural network.

4.2.1 Nonlinear harmonic oscillator

We consider the problem of approximating the potential function of a physical system, given by the Duffing oscillators with two mass points. Equations can be equivalently written as

$$\dot{q} = v, \quad \dot{v} = -\nabla V(q), \tag{14}$$

with $q = (q_1, q_2),\ v = (v_1, v_2)$ and $V(q)$ stands for the potential energy of the system. The learning task is to learn $V(q)$. The gradient of the potential is approximated using a neural network with 51500

Table 4: Time to get accuracy 10^{-2} by adaptive methods in oscillators problem parameterized by NN

Computation time		
Random initialization of parameters in NN	adaptive ALF	adaptive Y4
Initialization 1	2504 sec	1974 sec
Initialization 2	2961 sec	1857 sec
Initialization 3	4185 sec	2627 sec
Initialization 4	3542 sec	2125 sec
Initialization 5	3396 sec	2616 sec

Table 5: Wallclock time to get the training loss below 10^{-3} by adaptive methods in discretized PDE

Computation time		
Random initialization of parameters in NN	adaptive ALF	adaptive Y4
Initialization 1	336.6808 sec	138.5936 sec
Initialization 2	169.5010 sec	133.6825 sec
Initialization 3	180.8185 sec	140.3172 sec
Initialization 4	153.7389 sec	128.2795 sec
Initialization 5	142.7732 sec	142.9196 sec

parameters. To obtain the potential from the learned vector field, we apply numerical integration methods to the neural network approximating $-\nabla V(q)$. We compare the computational time of ALF and Y4 to reduce the value of the loss function below 10^{-2}. The results presented in Table 4 show that Y4 is faster than ALF in completing the training on different random initializations of the network parameters.

4.2.2 Discretized wave equation

In the second example we consider the 1-dimensional wave equation $u_{tt}(t,x) = u_{xx}(t,x) - \nabla V(u(t,x))$ on the spatial-temporal domain $[0,1] \times [0,0.3]$ with periodic boundary conditions in space for the potential $V(u) = \frac{1}{2}u^2$. On a spatial, equidistant, periodic mesh with mesh width $\Delta x = \frac{1}{40}$ we seek to describe the system's evolution by the first order system

$$\dot{u}_d = v_d, \quad \dot{v}_d = f(u_d), \tag{15}$$

where the unknown function f is parametrized as a fully connected ReLU neural network with one hidden layer of size 100. The dimension of (u_d, v_d) is 40. We compare the training performance of ALF and Y4. Both adaptive methods are employed with the same error tolerance. Yoshida is faster in finishing each epoch and the optimizer takes less time to minimize the training loss below 10^{-3}. The precise results are reported in Table 5 for 5 random initializations in the training. This illustrates the applicability of our method to the highly active research area of learning models of systems that are governed by partial differential equations.

5 Conclusion

In this work, we construct higher order reversible methods. These constitute explicit numerical integrators which are compatible with adaptive step-size selection strategies. The methods are employed to train deep neural networks that are based on neural ODEs. Thanks to the reversibility property, we avoid high memory requirements for backpropagation in the optimization procedure of the network parameters. Memory efficient backpropagation allows an application of deep architectures to the identification tasks of models of high-dimensional dynamical systems, which arise, for instance, as spatial discretizations of partial differential equations. As the method is based on neural ODEs, it can be trained with time-series data at irregular time-steps and can predict continuous time-series data. We showed the advantages of the newly constructed networks on the example of a network based on a 4th order method and demonstrate lower memory costs and faster training in comparison to lower order methods.

While the examples in the article focus on system identification tasks for systems governed by differential equations, extensions to neural stochastic differential equations (Kidger et al., 2021) are of interest and applications to normalizing flows or image processing (Allen-Blanchette et al., 2020) can be an exciting avenue to explore in future works.

REFERENCES

Christine Allen-Blanchette, Sushant Veer, Anirudha Majumdar, and Naomi Ehrich Leonard. LagNetViP: A Lagrangian neural network for video prediction (AAAI 2020 symposium on physics guided ai), 2020.

Coryn A L Bailer-Jones, David J C MacKay, and Philip J Withers. A recurrent neural network for modelling dynamical systems. *Network: Computation in Neural Systems*, 9(4):531, nov 1998. doi: 10.1088/0954-898X/9/4/008. URL https://dx.doi.org/10.1088/0954-898X/9/4/008.

Mark A Beaumont, Wenyang Zhang, and David J Balding. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, December 2002. ISSN 1943-2631. doi: 10.1093/genetics/162.4.2025. URL http://dx.doi.org/10.1093/genetics/162.4.2025.

Sergio Blanes, Fernando Casas, and Ander Murua. Splitting methods for differential equations. *Acta Numerica*, 33:1–161, 2024a. doi: 10.1017/S0962492923000077.

Sergio Blanes, Fernando Casas, and Ander Murua. Splitting methods for differential equations, 2024b. URL https://arxiv.org/abs/2401.01722.

Nicolas Boullé and Alex Townsend. Chapter 3 - a mathematical guide to operator learning. In Siddhartha Mishra and Alex Townsend (eds.), *Numerical Analysis Meets Machine Learning*, volume 25 of *Handbook of Numerical Analysis*, pp. 83–125. Elsevier, 2024. doi: https://doi.org/10.1016/bs.hna.2024.05.003. URL https://www.sciencedirect.com/science/article/pii/S1570865924000036.

Jake Bouvrie and Boumediene Hamzi. Kernel methods for the approximation of nonlinear systems. *SIAM Journal on Control and Optimization*, 55(4):2460–2492, 2017. doi: 10.1137/14096815X. URL https://doi.org/10.1137/14096815X.

Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016. doi: 10.1073/pnas.1517384113. URL https://www.pnas.org/doi/abs/10.1073/pnas.1517384113.

E. Celledoni, M. J. Ehrhardt, C. Etmann, R. I. Mclachlan, B. Owren, C.-B. Schonlieb, and F. Sherry. Structure-preserving deep learning. *European Journal of Applied Mathematics*, 32(5):888–936, 2021. ISSN 0956-7925. doi: 10.1017/S0956792521000139. URL https://www.cambridge.org/core/article/structurepreserving-deep-learning/15384A9F2776B2D1C1F1D3CDA390D779.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995. doi: 10.1109/72.392253.

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. In *International Conference on Learning Representations*, 2020.

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020a. doi: 10.1073/pnas.1912789117. URL https://www.pnas.org/doi/abs/10.1073/pnas.1912789117.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks, 2020b.

Wolfgang Dahmen and Arnold Reusken. *Numerik für Ingenieure und Naturwissenschaftler: Methoden, Konzepte, Matlab-Demos, E-Learning*. Springer Berlin Heidelberg, 2022. ISBN 9783662651810. doi: 10.1007/978-3-662-65181-0. URL http://dx.doi.org/10.1007/978-3-662-65181-0.

Marco David and Florian Méhats. Symplectic learning for hamiltonian neural networks. *Journal of Computational Physics*, 494:112495, 2023. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2023.112495. URL https://www.sciencedirect.com/science/article/pii/S0021999123005909.

Peter Deuflhard and Folkmar Bornemann. *Scientific Computing with Ordinary Differential Equations*. Springer New York, 2002. ISBN 9780387215822. doi: 10.1007/978-0-387-21582-2. URL http://dx.doi.org/10.1007/978-0-387-21582-2.

Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115—118, February 2017. ISSN 0028-0836. doi: 10.1038/nature21056. URL https://europepmc.org/articles/PMC8382232.

Amin Ghadami and Bogdan I. Epureanu. Data-driven prediction in dynamical systems: recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 380(2229):20210213, 2022. doi: 10.1098/rsta.2021.0213. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2021.0213.

Amir Gholaminejad, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *International Joint Conferences on Artificial Intelligence*. doi: 10.24963/ijcai.2019/103. URL https://par.nsf.gov/biblio/10322883.

Matthew M. Graham and Amos J. Storkey. Asymptotically exact inference in differentiable generative models. *Electronic Journal of Statistics*, 11(2):5105 – 5164, 2017. doi: 10.1214/17-EJS1340SI. URL https://doi.org/10.1214/17-EJS1340SI.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf.

R. Griesse and A. Walther. Evaluating gradients in optimal control: Continuous adjoints versus automatic differentiation. *Journal of Optimization Theory and Applications*, 122(1):63–86, 2004. ISSN 1573-2878. doi: 10.1023/B:JOTA.0000041731.71309.f1. URL https://doi.org/10.1023/B:JOTA.0000041731.71309.f1.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, dec 2017. doi: 10.1088/1361-6420/aa9a90. URL https://dx.doi.org/10.1088/1361-6420/aa9a90.

Ernst Hairer, Marlis Hochbruck, Arieh Iserles, and Christian Lubich. Geometric numerical integration. *Oberwolfach Reports*, 3(1):805–882, 2006.

Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2013. ISBN 9783662050187. doi: 10.1007/3-540-30666-8. URL https://doi.org/10.1007/3-540-30666-8.

Boumediene Hamzi and Houman Owhadi. Learning dynamical systems from data: A simple cross-validation perspective, part i: Parametric kernel flows. *Physica D: Nonlinear Phenomena*, 421:132817, 2021. ISSN 0167-2789. doi: https://doi.org/10.1016/j.physd.2020.132817. URL https://www.sciencedirect.com/science/article/pii/S0167278920308186.

Ken ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80125-X. URL https://www.sciencedirect.com/science/article/pii/S089360800580125X.

Pengzhan Jin, Zhen Zhang, Aiqing Zhu, Yifa Tang, and George Em Karniadakis. Sympnets: Intrinsic structure-preserving symplectic networks for identifying hamiltonian systems. *Neural Networks*, 132:166–179, 2020. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2020.08.017. URL https://www.sciencedirect.com/science/article/pii/S0893608020303063.

George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021. ISSN 2522-5820. doi: 10.1038/s42254-021-00314-5. URL https://doi.org/10.1038/s42254-021-00314-5.

Patrick Kidger, James Foster, Xuechen (Chen) Li, and Terry Lyons. Efficient and accurate gradients for neural sdes. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 18747–18761. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/9ba196c7a6e89eafd0954de80fc1b224-Paper.pdf.

Boris Kramer, Benjamin Peherstorfer, and Karen E. Willcox. Learning nonlinear reduced models from data with operator inference. *Annual Review of Fluid Mechanics*, 56 (Volume 56, 2024):521–548, 2024. ISSN 1545-4479. doi: https://doi.org/10.1146/annurev-fluid-121021-025220. URL https://www.annualreviews.org/content/journals/10.1146/annurev-fluid-121021-025220.

Guang Lin, Christian Moya, and Zecheng Zhang. Learning the dynamical response of nonlinear non-autonomous dynamical systems with deep operator neural networks. *Engineering Applications of Artificial Intelligence*, 125:106689, 2023. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2023.106689. URL https://www.sciencedirect.com/science/article/pii/S0952197623008734.

Yuying Liu, J. Nathan Kutz, and Steven L. Brunton. Hierarchical deep learning of multiscale differential equation time-steppers. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 380(2229):20210200, 2022. doi: 10.1098/rsta.2021.0200. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2021.0200.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL https://doi.org/10.1038/s42256-021-00302-5.

Sofya Maslovskaya and Sina Ober-Blöbaum. Symplectic methods in deep learning. *IFAC-PapersOnLine*, 58(17):85–90, 2024. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2024.10.118. URL https://www.sciencedirect.com/science/article/pii/S2405896324018731. 26th International Symposium on Mathematical Theory of Networks and Systems MTNS 2024.

Justice Mason, Christine Allen-Blanchette, Nicholas Zolman, Elizabeth Davison, and Naomi Leonard. Learning interpretable dynamics from images of a freely rotating 3d rigid body, 2022.

Takashi Matsubara, Yuto Miyatake, and Takaharu Yaguchi. Symplectic adjoint method for exact gradient of neural ode with minimal memory. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 20772–20784. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/adf8d7f8c53c8688e63a02bfb3055497-Paper.pdf.

Ulrich Mutze. An asynchronous leapfrog method ii, 2016.

Sina Ober-Blöbaum and Christian Offen. Variational learning of euler–lagrange dynamics from data. *Journal of Computational and Applied Mathematics*, 421:114780, March 2023. ISSN 0377-0427. doi: 10.1016/j.cam.2022.114780. URL http://dx.doi.org/10.1016/j.cam.2022.114780.

C. Offen and S. Ober-Blöbaum. Symplectic integration of learned hamiltonian systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(1), 1 2022. ISSN 1089-7682. doi: 10.1063/5.0065913. URL http://dx.doi.org/10.1063/5.0065913.

Christian Offen. Machine learning of discrete field theories with guaranteed convergence and uncertainty quantification, 2024. URL https://arxiv.org/abs/2407.07642.

Christian Offen and Sina Ober-Blöbaum. Learning of discrete models of variational pdes from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 34(1), January 2024. ISSN 1089-7682. doi: 10.1063/5.0172287. URL http://dx.doi.org/10.1063/5.0172287.

Hong Qin. Machine learning and serving of discrete field theories. *Scientific Reports*, 10(1), November 2020. ISSN 2045-2322. doi: 10.1038/s41598-020-76301-0. URL http://dx.doi.org/10.1038/s41598-020-76301-0.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal differential equations for scientific machine learning, 2021. URL https://arxiv.org/abs/2001.04385.

Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2017.11.039. URL https://www.sciencedirect.com/science/article/pii/S0021999117309014.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems, 2018a. URL https://arxiv.org/abs/1801.01236.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018b.

Patrick A. K. Reinbold, Logan M. Kageorge, Michael F. Schatz, and Roman O. Grigoriev. Robust learning from noisy, incomplete, high-dimensional experimental data via physically constrained symbolic regression. *Nature Communications*, 12(1):3219, 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-23479-0. URL https://doi.org/10.1038/s41467-021-23479-0.

Donald B. Rubin. Bayesianly Justifiable and Relevant Frequency Calculations for the Applied Statistician. *The Annals of Statistics*, 12(4):1151 – 1172, 1984. doi: 10.1214/aos/1176346785. URL https://doi.org/10.1214/aos/1176346785.

Samuel H. Rudy, J. Nathan Kutz, and Steven L. Brunton. Deep learning of dynamics and signal-noise decomposition with time-stepping constraints. *Journal of Computational Physics*, 396: 483–506, 2019a. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.06.056. URL https://www.sciencedirect.com/science/article/pii/S0021999119304644.

Samuel H. Rudy, J. Nathan Kutz, and Steven L. Brunton. Deep learning of dynamics and signal-noise decomposition with time-stepping constraints. *Journal of Computational Physics*, 396: 483–506, 2019b. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.06.056. URL https://www.sciencedirect.com/science/article/pii/S0021999119304644.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. ISSN 1573-1405. doi: 10.1007/s11263-015-0816-y. URL https://doi.org/10.1007/s11263-015-0816-y.

Max Schüssler, Tobias Münker, and Oliver Nelles. Deep recurrent neural networks for nonlinear system identification. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 448–454, 2019. doi: 10.1109/SSCI44817.2019.9003133.

Harsh Sharma, Hongliang Mu, Patrick Buchfink, Rudy Geelen, Silke Glas, and Boris Kramer. Symplectic model reduction of hamiltonian systems using data-driven quadratic manifolds. *Computer Methods in Applied Mechanics and Engineering*, 417:116402, 2023. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2023.116402. URL https://www.sciencedirect.com/science/article/pii/S0045782523005261.

Harsh Sharma, David A. Najera-Flores, Michael D. Todd, and Boris Kramer. Lagrangian operator inference enhanced with structure-preserving machine learning for nonintrusive model reduction of mechanical systems. *Computer Methods in Applied Mechanics and Engineering*, 423:116865, 2024. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2024.116865. URL https://www.sciencedirect.com/science/article/pii/S004578252400121X.

Masuo Suzuki. General theory of fractal path integrals with applications to many-body theories and statistical physics. *Journal of Mathematical Physics*, 32(2):400–407, 02 1991. ISSN 0022-2488. doi: 10.1063/1.529425. URL https://doi.org/10.1063/1.529425.

Giang Tran and Rachel Ward. Exact recovery of chaotic systems from highly corrupted data. *Multiscale Modeling & Simulation*, 15(3):1108–1129, 2017. doi: 10.1137/16M1086637. URL https://doi.org/10.1137/16M1086637.

Loup Verlet. Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159:98–103, Jul 1967. doi: 10.1103/PhysRev.159.98. URL https://link.aps.org/doi/10.1103/PhysRev.159.98.

Haruo Yoshida. Construction of higher order symplectic integrators. *Physics Letters A*, 150(5):262–268, 1990. ISSN 0375-9601. doi: https://doi.org/10.1016/0375-9601(90)90092-3. URL https://www.sciencedirect.com/science/article/pii/0375960190900923.

Aiqing Zhu, Pengzhan Jin, Beibei Zhu, and Yifa Tang. Inverse modified differential equations for discovery of dynamics, 2021.

Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ODE. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 11639–11649. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/zhuang20a.html.

Juntang Zhuang, Nicha C. Dvornek, Sekhar Tatikonda, and James S. Duncan. Mali: A memory efficient and reverse accurate integrator for neural odes, 2021.

A Error analysis of ALF2

Example A.1. Consider a simple example of a differential equation on $\mathbb{R}$ given by

$$\dot{z} = z^2 + t + \sin(zt) + \frac{1}{z^2+1}. \tag{16}$$

We solve the equation numerically using the ALF method and compare with a solution of high accuracy for different step sizes. The results are plotted in Figure 1 and show the second order behaviour in both (z, v) variables.

A.1 Proof of Theorem 3.1

We show that the local error of ALF2 in (z, v) is of order $O(h^3)$. Let us consider the Taylor expansion of the exact flow $(z(t), v(t))$ around $(z(t_0), v(t_0) = f(z_0, t_0))$.

$$z(t_0 + h) = z_0 + hf(z_0, t_0) + \frac{h^2}{2}\left(\frac{\partial f}{\partial z}(z_0, t_0) \circ f(z_0, t_0) + \frac{\partial f}{\partial t}(z_0, t_0)\right) + O(h^3),$$

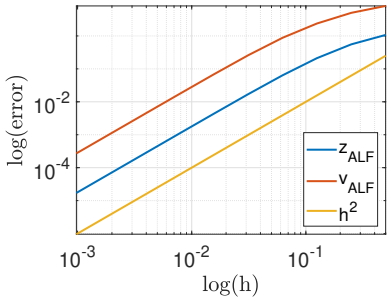

Figure 1: Log-log plot of the global error of trajectories $(z(t), v(t))$ of equation 16 defined on time interval $[0, 1.0]$ and obtained by ALF with h ranging from 0.5 to 10^{-3}.

$$v(t_0+h) = f(z_0,t_0) + h\left(\frac{\partial f}{\partial z}(z_0,t_0)\circ f(z_0,t_0) + \frac{\partial f}{\partial t}(z_0,t_0)\right) +$$
$$\frac{h^2}{2}(\frac{\partial^2 f}{\partial z^2}(z_0,t_0)(f(z_0,t_0),f(z_0,t_0)) + \frac{\partial f}{\partial z}(z_0,t_0)\circ\frac{\partial f}{\partial z}(z_0,t_0)\circ f(z_0,t_0) + 2\frac{\partial^2 f}{\partial z\partial t}(z_0,t_0)\circ f(z_0,t_0)$$
$$+ \frac{\partial f}{\partial z}(z_0,t_0)\circ\frac{\partial f}{\partial t}(z_0,t_0) + \frac{\partial^2 f}{\partial t^2}(z_0,t_0)) + O(h^3).$$

Now we consider the same for the numerical flow obtained with ALF2, that is composition of two steps of ALF each with the step size $\frac{h}{2}$. One step of ALF2 from (z_0, v_0) leads to (z_1, v_1) of the form

$$z_1(h) = z_0 + \frac{h}{2}(f(z_0 + \frac{h}{4}f(z_0,t_0), t_0 + \frac{h}{4}) + f(z_0 + hf(z_0 + \frac{h}{4}f(z_0,t_0), t_0 + \frac{h}{4})$$
$$- \frac{h}{4}f(z_0,t_0), t_0 + \frac{3h}{4})),$$

and

$$v_1(h) = v_0 + 2(f(z_0 + hf(z_0 + \frac{h}{4}f(z_0,t_0), t_0 + \frac{h}{4}) - \frac{h}{4}v_0, t_0 + \frac{3h}{4}) - f(z_0 + \frac{h}{4}f(z_0,t_0), t_0 + \frac{h}{4})).$$

Writing down the Taylor expansion in h for $(z_1(h), v_1(h))$ we find exactly the same terms as in $(z(t_0+h), v(t_0+h))$ up to terms of the third order $O(h^3)$. The computations to obtain the Taylor expansion of (z_1, v_1) were done using Maple software. This implies that the local error of ALF2 is of the 3rd order, and therefore, the global error is of order 2. This completes the proof.

B NUMERICAL METHOD FOR THE ADJOINT

The expression of equation 10 for ϕ_{k+1} obtained from ϕ_k by a step forward of ALF2 can be interpreted as a rescaled step backward of ALF2 applied to the state-adjoint dynamics. Let us introduce a map W_h depending on h, which acts on $(z, v, \lambda^z, \lambda^v)$ as follows. It only transforms λ^v multiplying it by $-\frac{h^2}{16}$, that is

$$W_\alpha(z, v, \lambda^z, \lambda^v) = \begin{pmatrix} \mathrm{Id} & 0 & 0 & 0 \\ 0 & \mathrm{Id} & 0 & 0 \\ 0 & 0 & \mathrm{Id} & 0 \\ 0 & 0 & 0 & \alpha\mathrm{Id} \end{pmatrix} \begin{pmatrix} z \\ v \\ \lambda^z \\ \lambda^v \end{pmatrix}.$$

Theorem B.1. *The step backward of the discretized state-adjoint system associated to the ALF2 method satisfies*

$$(z_k, v_k, \lambda^z_k, \lambda^v_k) = W^{-1}_{\frac{-h^2}{16}} \circ \Psi^{ALF2}_{-h} \circ W_{\frac{-h^2}{16}}(z_{k+1}, v_{k+1}, \lambda^z_{k+1}, \lambda^v_{k+1}), \tag{17}$$

where Ψ^{ALF2}_{-h} is applied to the state-adjoint equations of the augemented system for $\phi = (z, v)$

$$\dot\phi(t) = \tilde f(\phi(t), \theta(t)), \quad \dot\lambda = -\frac{\partial}{\partial\phi}\tilde f(\phi(t),\theta(t))^\top\lambda,$$

with $\tilde f(\phi, \theta) = (f(z,\theta), \frac{\partial}{\partial z}f(z,\theta)f(z,\theta))$.

Proof. In order to find the expression for ALF2, we first determine the expression for ALF and use the chain rule. Let us compute $\frac{\partial \phi_{k+1}}{\partial \phi_k}$ for ALF method, where $\phi_{k+1} = (z_{k+1}, v_{k+1})$ and $\phi_k = (z_k, v_k)$. Differentiating equation 7 with respect to (z_k, v_k), we obtain

$$\frac{\partial \phi_{k+1}}{\partial \phi_k} = \begin{pmatrix} \mathrm{Id} + h\frac{\partial f}{\partial z}(z_k + \frac{h}{2}v_k, t_k + \frac{h}{2}) & \frac{h^2}{2}\frac{\partial f}{\partial z}(z_k + \frac{h}{2}v_k, t_k + \frac{h}{2}) \\ 2\frac{\partial f}{\partial z}(z_k + \frac{h}{2}v_k, t_k + \frac{h}{2}) & h\frac{\partial f}{\partial z}(z_k + \frac{h}{2}v_k, t_k + \frac{h}{2}) - \mathrm{Id} \end{pmatrix}.$$

This implies

$$\begin{aligned} \lambda_k^z &= \left(\mathrm{Id} + h\frac{\partial}{\partial z}f(z^k + \frac{h}{2}v^k, t^k + \frac{h}{2})\right)\lambda_{k+1}^z + 2\frac{\partial}{\partial z}f(z^k + \frac{h}{2}v^k, t^k + \frac{h}{2})\lambda_{k+1}^v, \\ \lambda_k^v &= \frac{h^2}{2}\frac{\partial}{\partial z}f(z^k + \frac{h}{2}v^k, t^k + \frac{h}{2})\lambda_{k+1}^z + \left(h\frac{\partial}{\partial z}f(z^k + \frac{h}{2}v^k, t^k + \frac{h}{2}) - \mathrm{Id}\right)\lambda_{k+1}^v. \end{aligned} \tag{18}$$

Notice that equation 18 can be equivalently written as

$$\begin{aligned} \lambda_k^z &= \lambda_{k+1}^z + h\frac{\partial}{\partial z}f(z^k + \frac{h}{2}v^k, t^k + \frac{h}{2})\left(\lambda_{k+1}^z + \frac{2}{h}\lambda_{k+1}^v\right), \\ \lambda_k^v &= -2\frac{\partial}{\partial z}f(z^k + \frac{h}{2}v^k, t^k + \frac{h}{2})\left(-\frac{h^2}{4}\lambda_{k+1}^z - \frac{h}{2}\lambda_{k+1}^v\right) - \lambda_{k+1}^v. \end{aligned} \tag{19}$$

Let us now introduce $\tilde{\lambda}_k^v = -\frac{4}{h^2}\lambda_k^v$. Then equations take the following form

$$\begin{aligned} \lambda_k^z &= \lambda_{k+1}^z + h\frac{\partial}{\partial z}f(z^k + \frac{h}{2}v^k, t^k + \frac{h}{2})\left(\lambda_{k+1}^z - \frac{h}{2}\tilde{\lambda}_{k+1}^v\right), \\ \tilde{\lambda}_k^v &= -2\frac{\partial}{\partial z}f(z^k + \frac{h}{2}v^k, t^k + \frac{h}{2})\left(\lambda_{k+1}^z - \frac{h}{2}\tilde{\lambda}_{k+1}^v\right) - \tilde{\lambda}_{k+1}^v. \end{aligned} \tag{20}$$

Taking into account that $z_k + \frac{h}{2}v_k = z_{k+1} - \frac{h}{2}v_{k+1}$ from the construction of equation 7-equation 8, we conclude that variables $(\lambda_k^z, -\frac{4}{h^2}\lambda_k^v)$ follow the backward integration with ALF method and its step backward defined by equation 8 applied to the continuous equations of the adjoint equation 5. As a result, the step backward of the adjoint variables λ can be expressed as

$$(\lambda_k^z, \lambda_k^v) = \widehat{W}_{\frac{-h^2}{4}}^{-1} \circ \widehat{\Psi}_{-h}^{ALF}(z_{k+1}, v_{k+1}) \circ \widehat{W}_{\frac{-h^2}{4}}(\lambda_{k+1}^z, \lambda_{k+1}^v),$$

where $\widehat{W}_\alpha$ is a projection of W_α to variables $(\lambda_k^z, \lambda_k^v)$ and $\widehat{\Psi}_{-h}^{ALF}(z_{k+1}, v_{k+1})$ stands for a projection of the backward ALF step to (λ^z, λ^v), which is still a function of (z_{k+1}, v_{k+1}). To deduce the formula for the ALF2 method, we use its composition structure, namely, $\Psi_h^{ALF2} = \Psi_{h/2}^{ALF} \circ \Psi_{h/2}^{ALF}$. This implies

$$\frac{\partial \phi_{k+1}}{\partial \phi_k} = \frac{\partial}{\partial \phi_k}\left(\Psi_{h/2}^{ALF} \circ \Psi_{h/2}^{ALF}\right) = \left(\frac{\partial \Psi_{h/2}^{ALF2}}{\partial \phi}(\phi_{k+\frac{1}{2}})\right) \circ \left(\frac{\partial \Psi_{h/2}^{ALF2}}{\partial \phi}(\phi_k)\right)$$

with

$$\phi_{k+\frac{1}{2}} = \Psi_{h/2}^{ALF2}(\phi_k) = \Psi_{-h/2}^{ALF2}(\phi_{k+1}).$$

As a result,

$$\begin{aligned} \left(\frac{\partial \phi_{k+1}}{\partial \phi_k}\right)^\top &= \left(\frac{\partial \Psi_{h/2}^{ALF2}}{\partial \phi}(\phi_k)\right)^\top \circ \left(\frac{\partial \Psi_{h/2}^{ALF2}}{\partial \phi}(\phi_{k+\frac{1}{2}})\right)^\top \\ &= \widehat{W}_{\frac{-h^2}{16}}^{-1} \circ \widehat{\Psi}_{-h/2}^{ALF}(\phi_{k+\frac{1}{2}}) \circ \widehat{W}_{\frac{-h^2}{16}} \circ \widehat{W}_{\frac{-h^2}{16}}^{-1} \circ \widehat{\Psi}_{-h/2}^{ALF}(\phi_{k+1}) \circ \widehat{W}_{\frac{-h^2}{16}} \\ &= \widehat{W}_{\frac{-h^2}{16}}^{-1} \circ \widehat{\Psi}_{-h/2}^{ALF}(\Psi_{-h/2}^{ALF2}(\phi_{k+1})) \circ \widehat{\Psi}_{-h/2}^{ALF}(\phi_{k+1}) \circ \widehat{W}_{\frac{-h^2}{16}} \\ &= \widehat{W}_{\frac{-h^2}{16}}^{-1} \circ \widehat{\Psi}_{-h}^{ALF2}(z_{k+1}, v_{k+1}) \circ \widehat{W}_{\frac{-h^2}{16}}. \end{aligned}$$

The resulting equations for the backward step of the state-adjoint system are given in equation 17. This completes the proof of Theorem B.1. □

The formula for the adjoint of Yoshida methods Φ^Y_{2k} follows from the composition structure of the method and is presented the following theorem. With a slight abuse of notation we denote Φ^{ALF2} by Φ^Y_2.

Theorem B.2. *Assume that the discrete one step method in equation 2 is given for $k \geq 2$ by*

$$\Psi^Y_{2k}(h) = \Psi^Y_{2k-2}(ah) \circ \Psi^Y_{2k-2}(bh) \circ \Psi^Y_{2k-2}(ah), \qquad \phi_{k+1} = \Psi^Y_{2k}(h) \circ \phi_k. \tag{21}$$

Then the state-adjoint backward step can be computed recursively as follows

$$(\phi_k, \lambda_k) = \widetilde{\Psi}^Y_{2k-2}(ah) \circ \widetilde{\Psi}^Y_{2k-2}(bh) \circ \widetilde{\Psi}^Y_{2k-2}(ah)(\phi_{k+1}, \lambda_{k+1}), \tag{22}$$

with $\widetilde{\Psi}^Y_{2k-2}$ the map, which defines the backward step of state-adjoint system of the method Φ^Y_{2k-2}.

Proof. The proof is by induction on k in the considered method Ψ^Y_{2k} and is based on the composition structure of Ψ^Y_{2k} in equation 21. Let $k = 2$, then $\Psi^Y_4(h) = \Psi^{ALF2}_{ah} \circ \Psi^{ALF2}_{bh} \circ \Psi^{ALF2}_{ah}$. By the chain rule we have

$$\left(\frac{\partial \phi_{k+1}}{\partial \phi_k}\right)^\top = \left(\frac{\partial \Psi^{ALF2}_{ah}}{\partial \phi}(\phi_{k+\frac{1}{3}})\right)^\top \circ \left(\frac{\partial \Psi^{ALF2}_{bh}}{\partial \phi}(\phi_{k+\frac{2}{3}})\right)^\top \circ \left(\frac{\partial \Psi^{ALF2}_{ah}}{\partial \phi}(\phi_{k+1})\right)^\top$$

with

$$\phi_{k+\frac{2}{3}} = \Psi^{ALF2}_{-ah}(\phi_{k+1}),$$
$$\phi_{k+\frac{1}{3}} = \Psi^{ALF2}_{-bh} \circ \Psi^{ALF2}_{-ah}(\phi_{k+1}).$$

By construction of the backward step of the state adjoint system by ALF2 shown in equation 17, we have

$$\left(\frac{\partial \phi_{k+1}}{\partial \phi_k}\right)^\top = Pr_\lambda\left(W^{-1}_{ah} \circ \Psi^{ALF2}_{-ah} \circ W_{ah}\right) \circ Pr_\lambda\left(W^{-1}_{bh} \circ \Psi^{ALF2}_{-bh} \circ W_{bh}\right) \circ$$
$$\circ Pr_\lambda\left(W^{-1}_{ah} \circ \Psi^{ALF2}_{-ah} \circ W_{ah}\right),$$

where $\widehat{\Psi}^{ALF2}_{h_i} = Pr_\lambda\left(W^{-1}_{h_i} \circ \Psi^{ALF2}_{-h_i} \circ W_{h_i}\right)$, $h_i \in \{ah, bh\}$ defines a step backward with the ALF2 method with step-size h_i in the adjoint variable. This proves the Theorem for $k = 2$. Let us assume now that the statement of the theorem holds for $k = k_0$ and we consider the adjoint method for $\Psi^Y_{2k_0}(h) = \Psi^Y_{2k_0-2}(ah) \circ \Psi^Y_{2k_0-2}(bh) \circ \Psi^Y_{2k_0-2}(ah)$. As before, applying the chain rule and the assumption of the induction, it follows that

$$\left(\frac{\partial \phi_{k+1}}{\partial \phi_k}\right)^\top = \left(\frac{\partial \Psi^Y_{2k_0-2}(ah)}{\partial \phi}(\tilde{\phi}_{k+\frac{1}{3}})\right)^\top \circ \left(\frac{\partial \Psi^Y_{2k_0-2}(bh)}{\partial \phi}(\tilde{\phi}_{k+\frac{2}{3}})\right)^\top \circ \left(\frac{\partial \Psi^Y_{2k_0-2}(ah)}{\partial \phi}(\phi_{k+1})\right)^\top$$
$$= \widehat{\widetilde{\Psi}^Y}_{2k_0-2}(ah) \circ \widehat{\widetilde{\Psi}^Y}_{2k_0-2}(bh) \circ \widehat{\widetilde{\Psi}^Y}_{2k_0-2}(ah),$$

where we used the notation

$$\tilde{\phi}_{k+\frac{2}{3}} = \Psi^Y_{2k_0-2}(-ah)(\phi_{k+1}), \quad \tilde{\phi}_{k+\frac{1}{3}} = \Psi^Y_{2k_0-2}(-bh) \circ \Psi^Y_{2k_0-2}(-ah)(\phi_{k+1}),$$

and $\widehat{\widetilde{\Psi}^Y}_{2k_0-2}$ the projection of the step backward associated to the state-adjoint system and $\Psi^Y_{2k_0-2}$ method. This completes the induction step and the proof. □

In case of $k = 2$, Theorem B.2 in combination with equation 17 leads to the following expression

$$\widetilde{\Psi}^Y_4 = W^{-1}_{\frac{-(ah)^2}{16}} \circ \Psi^{ALF2}_{-ah} \circ W_{\frac{a^2}{b^2}} \circ \Psi^{ALF2}_{-bh} \circ W_{\frac{b^2}{a^2}} \circ \Psi^{ALF2}_{-ah} \circ W_{\frac{-(ah)^2}{16}}.$$

The obtained results lead to the Algorithm 4 for the computation of gradients.

C Details of numerical experiments

In all the numerical experiments, our implementation of the Yoshida composition method uses the code of the MALI network (Zhuang et al., 2021). We use the steps forward and backward of the ALF method as composition steps to compute ALF2 and its Yoshida composition.

Algorithm 4 Computation of gradients

1. Input: training data z_0, initialization of parameters θ, velocity $v_0 = f(z_0, \theta_0)$
2. Propagate through the network using Ψ_{2k}^{Y} to get (z_N, v_N)
3. Set $\lambda_N^z = \nabla L(z_N)$ and $\lambda_N^v = 0$

for `k = N to 1` **do**

4. Compute ϕ_k, λ_k from $\phi_{k+1}, \lambda_{k+1}$ using equation 17 and equation 22.
5. Compute $\frac{\partial J(\theta)}{\partial \theta_k}$ using equation 11

end for

6. Output: gradients $\frac{\partial J(\theta)}{\partial \theta_k}$ for $k = 1, \dots N-1$.

C.1 KEPLER PROBLEM

The training data for the comparison of the computational time in Table 2 is given by a trajectory x of equation 12 with initial condition $x_0 = (0.75, 0, 0, \frac{0.9\pi}{4}\sqrt{\frac{5}{3}})$ on time interval $[0, T] = [0, 1]$, which is an elliptic orbit. The trajectory is obtained by numerical integration using `sci.integrate.odeint` with relative and absolute tolerances 10^{-7} and 10^{-8} respectively and maximum step size 10^{-5}. The optimizer used in the training is `SGD` from `PyTorch` with initial learning rate 0.1 and scaled by 0.95 for each epoch. For completeness, we show the evaluation of the parameter error across the learning displayed as a function of time in Figure 2 and as a function of epochs in Figure 3. In the plots we show the results obtained with ALF, Y4 and also Runge-Kutta 4(5) (RK45), the latter is not a reversible method and requires storage of the intermediate states obtained during the integration forward. This implies additional memory consumption, namely, at each epoch the algorithm saves 8 additional states obtained during integration forward, making the memory consumption of the training higher. The four plots in Figures 3 and 2 are obtained for different initializations of the parameters α_0 in the learning, namely, $\alpha_0 = 1.3, 0.1, 0.7, 0.75$.

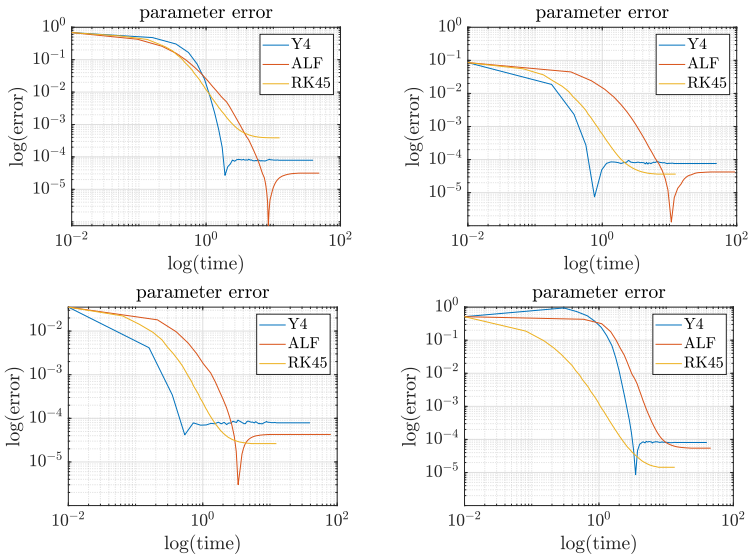

Figure 2: Error of the learned parameter with respect to the ground truth α as a function of time.

The error landscape in Figure 4 is obtained by considering 81 trajectories obtained using the same integration method as for the time comparison explained above for 81 different initial conditions $(x_0)_i$ in a neighborhood of x_0, given by a 4-dimensional box of diameter 0.4 around x_0. The points $(x_0)_i$ are chosen on a grid with a step size 0.1, which includes x_0 as its point. The behaviour of ALF and Y4 with adaptive stepping can be better understood when looking at fixed step methods, when the step size $h_i = h$ is fixed for all the steps. The loss landscape visualized in Figure 4 for fixed step ALF and fixed step Y4 shows that the minimum value of the loss is achieved at a better precision

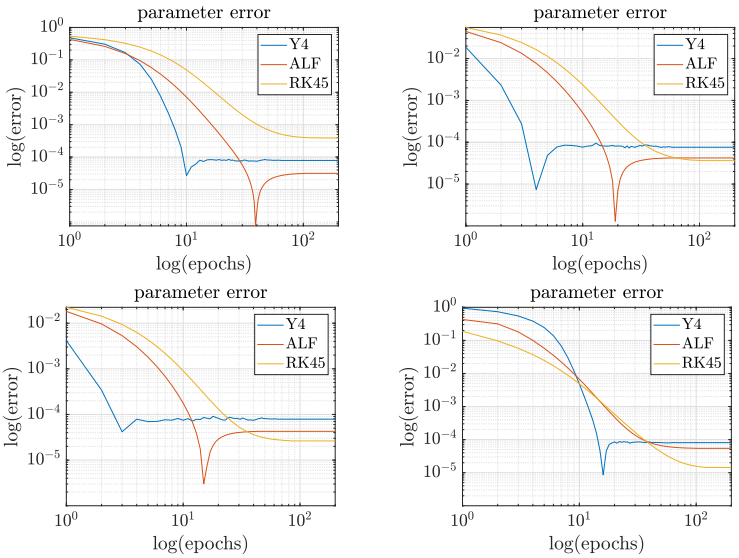

Figure 3: Error of the learned parameter with respect to the ground truth α as a function of epochs.

of the true parameter for the higher order methods than for the lower order method, which will be explained in more detail below. The loss visualized in Figure 4 as a function of α is

$$L(\alpha_k) = \frac{1}{81} \sum_{i=1}^{81} \sum_{j=1}^{5} \sum \| (q_{N_j}(\alpha_k))_i - q(t_j, (x_0)_i) \|^2$$

with q_{N_j} projection of z_{N_j} to q-coordinate and α_k taking 300 values in $[\frac{\pi}{4} - 10^{-4}, \frac{\pi}{4} + 10^{-4}]$. Here $(q_{N_j}(\alpha))_i$ is obtained by numerical integration of equation 12.

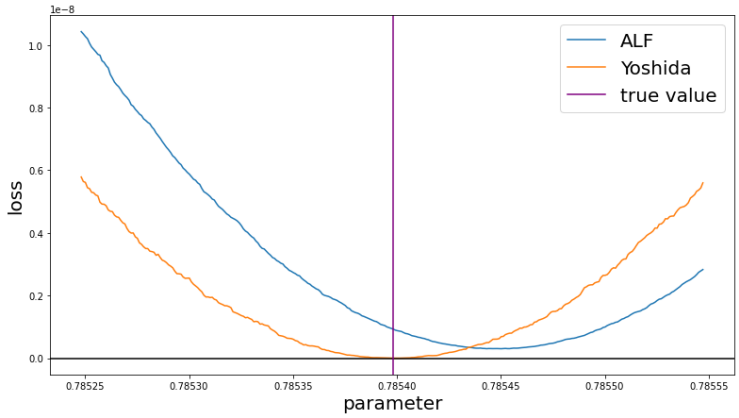

Figure 4: Error landscape of ALF and Y4 methods for Kepler problem showing the loss computed for the parameters in a neighbourhood of the true value of α displayed by a vertical line.

If $(q_{N_j}(\alpha))_i$ was obtained by exact integration of equation 12 and in the absence of noise and round-off errors, true parameter values constitute minima for L. We interpret the application of a numerical integrator as a perturbation of size $\mathcal{O}(h^p)$ to the exact $(q_{N_j}(\alpha))_i$, where h is the step size of the integration and p the order of the numerical method. This yields a perturbation $\tilde{L}$ of L of size $\mathcal{O}(h^{2p})$ in case of the mean-square loss. Thus, assuming that the local minima of L at the true parameter value is non-degenerate, $\tilde{L}$ has a local minimum within a ball around the true parameter of size $\mathcal{O}(h^p)$. This follows from classical discussions on the numerical conditioning of computing zeros of a function as, for instance, in (Dahmen & Reusken, 2022, §5.2). This provides a direct relation of the order of an integration method and the accuracy of identified parameters.

Notice that in the adaptive-step size context the perturbation of L and, thus, the error of its minima are controlled by the provided error tolerance. However, the discussion shows that in order to be able to expect the same accuracy in the parameter identification, neural ODEs based on lower-order methods require more integration steps than neural ODEs based on high-order methods.

The above $\mathcal{O}(h^p)$ error relation in the parameter estimation constitutes an asymptotic upper bound. In Geometric Numerical Integration errors of numerical integrators can enter in highly symmetric way (Hairer et al., 2013). In symplectic integration of Hamiltonian systems, for instance, energy errors enter in an unbiased form. If the sought parameter is related to the geometric structure that is preserved by the geometric numerical integrator, parameters can potentially be estimated to higher accuracy than expected by the order of the numerical integrator. This, together with backward error analysis techniques, was used in (Offen & Ober-Blöbaum, 2022), for instance, to accurately identify a Hamiltonian function of a dynamical system even though a low order method was used to discretize the dynamical system. These techniques, however, are tailored to the geometric problem at hand, while the approach of this article considers a more general case.

C.2 NONLINEAR HARMONIC OSCILLATOR

There are two settings considered for the learning of the dynamics equation 13. In the first setting, the learning problem is the parameter identification as presented in Section 4.1.2. In the second case we consider the parametrization of the potential by a neural network as described in Section 4.2.1. Here we give more details on both problems.

C.2.1 IDENTIFICATION OF PARAMETERS

In the experiments for the time comparison shown in Table 3 we consider a set of 200 trajectories in the training data with the initial conditions generated by the Halton sequence in a 20-dimensional box around zero vector x_0 with diameter 2.0. The trajectories are obtained by numerical integration using `sci.integrate.odeint` with relative and absolute tolerances 10^{-13} and 10^{-14} respectively. In the training we use `AdamW` optimizer from `PyTorch` with learning rate scheduler `ExponentialLR`. The results shown in Table 3 are obtained with different learning rates, namely, the first two with the initial learning rate 10^{-2} and $\gamma = 0.995$, the last three with the initial learning rate 10^{-1} and $\gamma = 0.998, 0.997, 0.99$ for the tree results respectively. At each epoch we consider all 200 trajectories, so that the loss is $L = \frac{1}{200}\sum_{i=1}^{200} \|(z_N)_i - x(T,(x_0)_i)\|^2$ with $T = 0.5$. While Table 3 compares the training time of ALF and Y4, it is also important to compare their performance in the learned parameters. In Figure 5 we show the results in the error of the learned parameters as a function of computational time measured at each epoch of ALF, Y4 and also RK45, which is not reversible. We can see that Y4 in not only faster than ALF in the training but the same also holds for the error in the learned parameters. While RK45 is the fastest to get to accurate parameters, it also requires the storing of 80 additional states during integration at each epoch, which means a considerable contribution to the memory costs. To better understand the reasons of the faster learning of Y4 than ALF, we show in Figure 6 the computation time accumulated at each epoch of the training. The computational time per epoch is smaller for Y4, which contributes to the faster convergence in the training. In both Figures 5 and6, the four plots correspond to different random initializations of the parameters in the optimization.

In addition to results obtained for adaptive stepping, we test ALF and Y4 with the step size fixed to $h = 0.1$ and the training until either the training accuracy reaches 10^{-4} or the number of epochs reaches 500. Figure 7 shows that ALF is stuck at the training accuracy 10^{-2} and the training stops because of reaching 500 epochs, while Y4 converges to accuracy 10^{-4} with 181 epochs. The same behaviour is observed for different parameter initialization. Decreasing the step size to $h = 0.01$ permits ALF to reach accuracy 10^{-4}. The results obtained in Figure 7 show that with a fixed step size the lower order method is unable to achieve an accuracy better than 10^{-2} in training loss, whereas Y4 reaches accuracy 10^{-4}. This illustrates what happens in the case of the adaptive time-stepping. A lower order method needs to reduce the step size to get to better accuracy. This implies more steps in the integration, and therefore, slower computations.

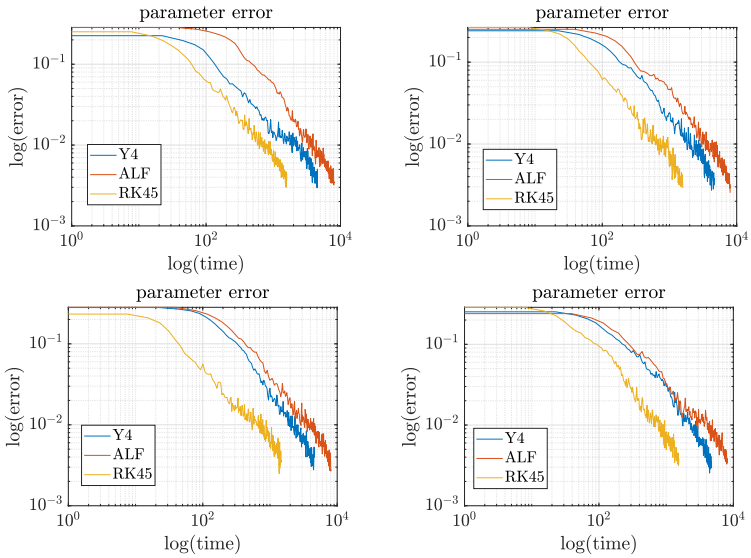

Figure 5: Error in learned parameters as a function of time.

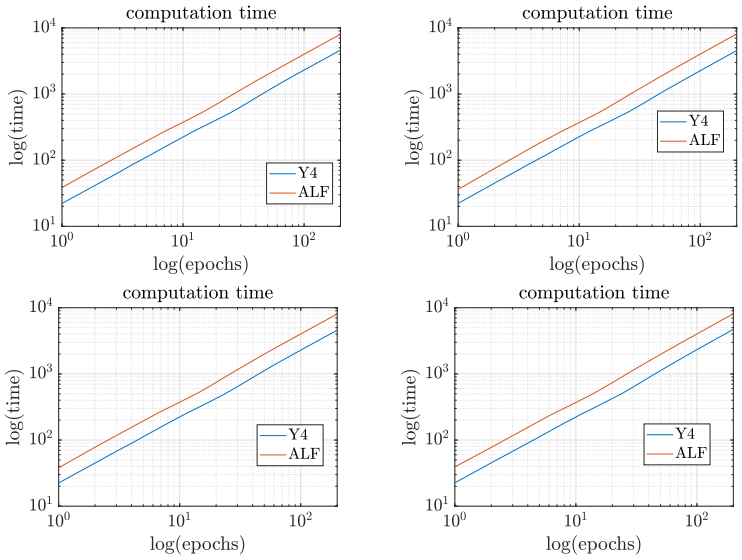

Figure 6: Time of computation in function of epochs. When the curve is positioned lower, the corresponding algorithm is faster.

C.2.2 Neural network parametrization

The goal is to find the unknown potential governing equation 14. For this we assume a particular form of the potential, namely,

$$V(q) = \sum_{i=1}^{s} \sum_{j=1}^{n} c_{i,j} \sigma_i(q_j) + \sum_{i=1}^{d} \sum_{j=1}^{n} \sum_{k=j+1}^{n} C_{i,j,k} \Sigma_i(\|q_j - q_k\|),$$

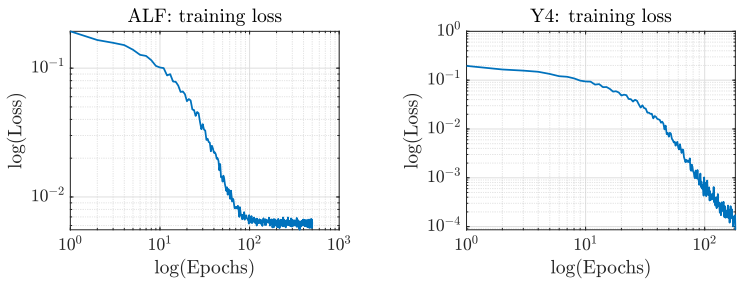

Figure 7: Training loss is displayed in logarithmic scale for the parameter identification in case of coupled oscillation for ALF and Y4 with fixed step size $h = 0.1$.

where σ_i stand for different single particle potentials and Σ_i for double particle potentials. In the case considered above, we have

$$c_{1,1} = \frac{a_1}{2},\ c_{1,2} = \frac{a_2}{2},\ \sigma_1(q) = q^2,$$
$$c_{2,1} = \frac{b_1}{4},\ c_{2,2} = \frac{b_2}{4},\ \sigma_2(q) = q^4,$$
$$C_{1,1,2} = \frac{e}{2},\ \Sigma_1(x) = x^2.$$

In the learning problem, we assume that functions σ_1, σ_2 and Σ_1 are unknown as well as parameters a_1, a_2, b_1, b_2, e. We parameterize the derivatives $\frac{a_1}{2}\sigma_1'$, $\frac{a_2}{2}\sigma_1'$, $\frac{b_1}{4}\sigma_2'$, $\frac{b_2}{4}\sigma_2'$ and $\frac{e}{2}\Sigma_1'$ by neural networks each and use them to model the dynamics in equation 14. We use 5 neural networks, which we denote by $\xi_1, \xi_2, \xi_3, \xi_4, \xi_5$. All of them have the same architecture $q \mapsto W_1 \tanh(W_2 \tanh(W_3 q))$, where W_1 is a matrix of parameters of size 1×100, matrix W_2 is of size 100×100 and W_3 is of size 100×1. The resulting dynamics is defined by

$$\dot{q}_1 = v_1, \quad \dot{v}_1 = -\xi_1(q_1) - \xi_3(q_1) - \xi_5(q_1 - q_2),$$
$$\dot{q}_2 = v_2, \quad \dot{v}_2 = -\xi_2(q_2) - \xi_4(q_2) - \xi_5(q_2 - q_1).$$

The equations parameterized by neural networks are then integrated using ALF or Yoshida composition of ALF2 at each epoch in the training. The training data is set to be a set of 1000 trajectories with the initial conditions generated by the Halton sequence in a 4-dimensional box around $x_0 = (0.8, -0.4, 0.0, 0.0)$ with diameter 2.0. The optimizer is `AdamW` with initial learning rate 10^{-3} and scheduler `ExponentialLR` with $\gamma = 0.995$. In addition, we consider batches of 300 trajectories at each epoch with the resulting loss function of the same form as in the case of the parameter identification problem.

C.3 Discretized wave equations

For generation of the training data, we consider the wave equation with potential $V(u) = \frac{1}{2}u^2$. The true motions can be expressed in the time-dependent Fourier series as

$$u(t, x) = \sum_{m=-\infty}^{\infty} \hat{u}_m(t) e^{2\pi i m x / L}, \quad L = 1$$

where the Fourier coefficients evolve as

$$\hat{u}_m(t) = \gamma_m^{-1} \hat{v}_{m,0} \sin(\gamma_m t) + \hat{u}_{m,0} \cos(\gamma_m t), \quad \gamma_m = \sqrt{1 + \frac{4\pi^2}{L^2} m^2}.$$

Here $\hat{u}_{m,0}$, $\hat{v}_{m,0}$ are the Fourier coefficients of an initial wave $u(0, x)$ and velocity $u_t(0, x)$, respectively. Notice that a Fourier coefficient $\hat{u}_m(t)$ remains exactly zero over time if and only if $\hat{u}_{m,0} = 0 = \hat{v}_{m,0}$. Training data to initial data with only finitely many nonzero Fourier coefficients can, therefore, be obtained to machine precision by a spectral method. Alternatively, solutions can be computed by an application of the 5-point stencil as described in Example 7 (16) in (Offen & Ober-Blöbaum, 2024) on a fine mesh with discretization parameters $\Delta t = 1/160$, $\Delta x = 1/80$ and then subsampled to a mesh with $\Delta t = 1/40$, $\Delta x = 1/20$. In our case both methods yield the same

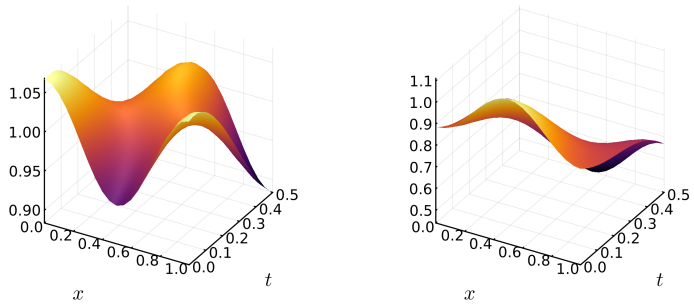

Figure 8: Two samples of the training data used in section C.3.

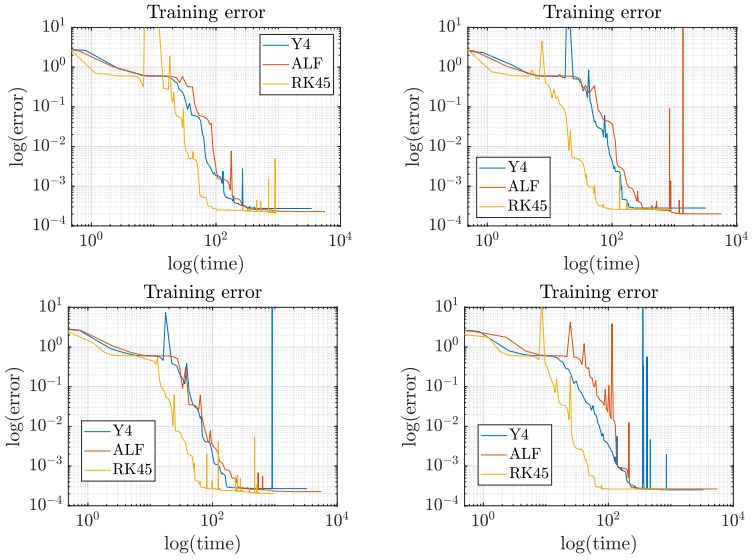

Figure 9: Time of computation in function of epochs. Lower curve means faster computations.

training data up to a maximum error of order $1\mathrm{e}-4$. In the training data creation, we sample initial $\hat{u}_{m,0}$, $\hat{v}_{m,0}$ from a standard normal distribution. It is then weighted by e^{-4m^8} such that effectively only the first two Fourier modes are active. See figure 8 for a plot of two of the solutions to the wave equation that were used to create the training data set. In the training, we consider initial and final points of 50 trajectories on time interval $[0, 0.3]$ and 30 unseen trajectories in the testing. We use the optimiser LBFGS with the default values of the parameters. In the numerical tests, we compare the behaviour of ALF, Y4 and Runge-Kutta 4(5). It can be seen in Figure 9 that Y4 reaches the lowest values in the training loss faster than ALF. While RK45 is fastest, it also consumes more memory, which can make a crucial difference in high dimensional systems. We also report a lower time of computations per epoch for Y4 with respect to the results by ALF in Figure 10.

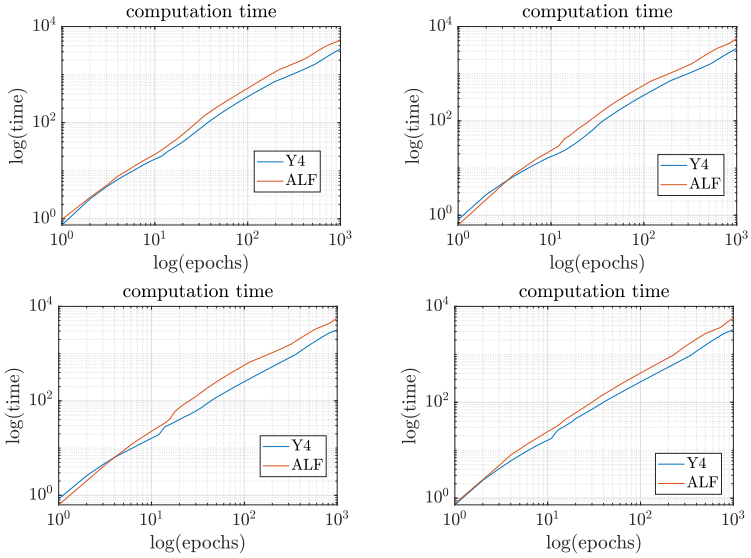

Figure 10: Time of computation in function of epochs. When the curve is positioned lower, the corresponding algorithm is faster.