

Learning Hamiltonian Systems with Mono-Implicit Runge–Kutta Methods^{*}

Håkon Noren

Department of Mathematical Sciences, Norwegian University of Science and
Technology, Trondheim, Norway
`hakon.noren@ntnu.no`

Abstract. Numerical integrators could be used to form interpolation conditions when training neural networks to approximate the vector field of an ordinary differential equation (ODE) from data. When numerical one-step schemes such as the Runge–Kutta methods are used to approximate the temporal discretization of an ODE with a known vector field, properties such as symmetry and stability are much studied. Here, we show that using mono-implicit Runge–Kutta methods of high order allows for accurate training of Hamiltonian neural networks on small datasets. This is demonstrated by numerical experiments where the Hamiltonian of the chaotic double pendulum in addition to the Fermi–Pasta–Ulam–Tsingou system is learned from data.

Keywords: Inverse problems · Hamiltonian systems · Mono-implicit Runge–Kutta · Deep neural networks.

1 Introduction

In this paper, we apply backward error analysis [11] to motivate the use of numerical integrators of high order when approximating the vector field of ODEs with neural networks. We particularly consider mono-implicit Runge–Kutta (MIRK) methods [3,1], a class of one-step methods that are explicit when solving inverse problems. Such methods can be constructed to have high order with relatively few stages, compared to explicit Runge–Kutta methods, and attractive properties such as symmetry. Here, we perform numerical experiments learning two Hamiltonian systems with MIRK methods up to order $p = 6$. To the best of our knowledge, this is the first demonstration of the remarkable capacity of numerical integrators of order $p > 4$ to facilitate the training of Hamiltonian neural networks [9] from sparse datasets, to do accurate interpolation and extrapolation in time.

Recently, there has been a growing interest in studying neural networks through the lens of dynamical systems. This is of interest both to accelerate data-driven modeling and for designing effective architectures for neural networks [10,16,8]. Considering neural network layers as the flow of a dynamical

^{*} Supported by the Research Council of Norway, through the project DynNoise: Learning dynamical systems from noisy data. (No. 339389).

system is the idea driving the study of so-called neural ODEs [4] and its discretized counter-part, residual neural networks.

Hamiltonian mechanics provide an elegant formalism that allows a wide range of energy preserving dynamical systems to be described as first order ODEs. Hamiltonian neural networks [9] aim at learning energy-preserving dynamical systems from data by approximating the Hamiltonian using neural networks. A central issue when studying neural networks and dynamical systems is which method to use when discretizing the continuous time dynamics. Several works use backward error analysis to argue for the importance of using symplectic integrators for learning the vector field of Hamiltonian systems [5,17,14]. Using Taylor expansions to derive the exact form of the inverse modified vector field allows for the construction of a correction term that cancels the error stemming from the temporal discretization, up to arbitrary order [14,6].

2 Inverse ODE problems on Hamiltonian form

We consider a first-order ODE

$$\frac{d}{dt}y(t) = f(y(t)), \quad y(t) : [0, T] \rightarrow \mathbb{R}^n, \quad (1)$$

and assume that the vector field f is unknown, whereas samples $S_N = \{y(t_n)\}_{n=0}^N$ of the solution are available, with constant step size h . Then the inverse problem aims at deriving an approximation $f_\theta \approx f$ where θ is a set of parameters to be chosen. The inverse problem can be formulated as the following optimization problem:

$$\arg \min_{\theta} \sum_{n=0}^{N-1} \left\| y(t_{n+1}) - \Phi_{h, f_\theta}(y(t_n)) \right\|, \quad (2)$$

where f_θ is a neural network approximation of f with parameters θ , and Φ_{h, f_θ} is a one-step integration method with step size h such that $y_{n+1} = \Phi_{h, f}(y_n)$. In particular, we assume that (1) is a Hamiltonian system, meaning that

$$f(y) = J \nabla H(y(t)), \quad J := \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} \in \mathbb{R}^{2d \times 2d}. \quad (3)$$

We follow the idea of Hamiltonian neural networks [9] aiming at approximating the Hamiltonian, $H : \mathbb{R}^{2d} \rightarrow \mathbb{R}$, such that H_θ is a neural network and f is approximated by $f_\theta(y) := J \nabla H_\theta(y)$. It thus follows that the learned vector field f_θ by construction is Hamiltonian.

3 Mono-implicit Runge–Kutta for inverse problems

Since the solution is known point-wise, $S_N = \{y(t_n)\}_{n=0}^N$, the points y_n and y_{n+1} can be substituted by $y(t_n)$ and $y(t_{n+1})$ when computing the next step of a one-step integration method. We denote this substitution as the *inverse injection*,

and note that this yields an interpolation condition for $f_\theta \approx f$ for each n . If we let Φ_{h,f_θ} in (2) be the so-called implicit midpoint method, we get the following expression to be minimized:

$$\left\| y(t_{n+1}) - \left(y(t_n) + hf_\theta\left(\frac{y(t_n) + y(t_{n+1})}{2}\right) \right) \right\|, \quad n = 0, \dots, N-1. \quad (4)$$

For the midpoint method, the inverse injection bypasses the computationally costly problem of solving a system of equations within each training iteration, since $y(t_{n+1})$ is known. More generally, mono-implicit Runge–Kutta (MIRK) methods constitute the class of all Runge–Kutta methods that form explicit methods under this substitution. Given vectors $b, v \in \mathbb{R}^s$ and a strictly lower triangular matrix $D \in \mathbb{R}^{s \times s}$, a MIRK method is a Runge–Kutta method where $A = D + vb^T$, and is thus given by

$$y_{n+1} = y_n + h \sum_{i=1}^s b_i k_i, \quad (5)$$

$$k_i = f\left(y_n + v_i(y_{n+1} - y_n) + h \sum_{j=1}^s d_{ij} k_j\right).$$

Let us denote \hat{y}_{n+1} and \hat{k}_i as the next time-step and the corresponding stages of a MIRK method when substituting y_n, y_{n+1} by $y(t_n), y(t_{n+1})$ on the right-hand side of (5).

Theorem 1. *Let y_{n+1} be given by a MIRK scheme (5) of order p and \hat{y}_{n+1} be given by the same method under the inverse injection. Assume that only one integration step is taken from a known initial value $y_n = y(t_n)$. Then*

$$\hat{y}_{n+1} = y_{n+1} + \mathcal{O}(h^{p+2}) \quad (6)$$

$$\text{and } \hat{y}_{n+1} = y(t_{n+1}) + \mathcal{O}(h^{p+1}). \quad (7)$$

Proof. Since the method (5) is of order p we have that

$$\begin{aligned} \hat{k}_1 &= f\left(y(t_n) + v_1(y(t_{n+1}) - y(t_n))\right) \\ &= f\left(y_n + v_1(y_{n+1} - y_n)\right) + \mathcal{O}(h^{p+1}) = k_1 + \mathcal{O}(h^{p+1}) \end{aligned}$$

The same approximation could be made for $\hat{k}_2, \dots, \hat{k}_s$, since D is strictly lower triangular, yielding $\hat{k}_i = k_i + \mathcal{O}(h^{p+1})$ for $i = 1, \dots, s$. In total, we find that

$$\begin{aligned} \hat{y}_{n+1} &= y(t_n) + h \sum_{i=1}^s b_i \hat{k}_i \\ &= y_n + h \sum_{i=1}^s b_i k_i + \mathcal{O}(h^{p+2}) \\ &= y_{n+1} + \mathcal{O}(h^{p+2}) \\ &= y(t_{n+1}) + \mathcal{O}(h^{p+1}) + \mathcal{O}(h^{p+2}). \\ &= y(t_{n+1}) + \mathcal{O}(h^{p+1}). \end{aligned}$$

For the numerical experiments, we will consider the optimal MIRK methods derived in [12]. The minimal number of stages required to obtain order p is $s = p - 1$ for MIRK methods [1]. In contrast, explicit Runge–Kutta methods need $s = p$ stages to obtain order p for $1 \leq p \leq 4$ and $s = p + 1$ stages for $p = 5, 6$ [2], meaning that the MIRK methods have significantly lower computational cost for a given order. As an example, a symmetric, A-stable MIRK method with $s = 3$ stages and of order $p = 4$ is given by

$$\begin{aligned} k_1 &= f(y_n), & k_2 &= f(y_{n+1}), \\ k_3 &= f\left(\frac{1}{2}(y_n + y_{n+1}) + \frac{h}{8}(k_1 - k_2)\right), \\ y_{n+1} &= y_n + \frac{h}{6}(k_1 + k_2 + 4k_3). \end{aligned}$$

4 Backward error analysis

Let $\varphi_{h,f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ be the h -flow of an ODE such that $\varphi_{h,f}(y(t_0)) := y(t_0 + h)$ for an initial value $y(t_0)$. With this notation, the vector field $f_h(y)$ solving the optimization problem (2) exactly must satisfy

$$\varphi_{h,f}(y(t_n)) = \Phi_{h,f_h}(y(t_n)), \quad n = 0, \dots, N - 1. \quad (8)$$

For a given numerical one-step method Φ , the *inverse modified vector field* [18] f_h could be computed by Taylor expansions. However, since their convergence is not guaranteed, truncated approximations are usually considered. This idea builds on backward error analysis [11, Ch. IX], which is used in the case of forward problems (f is known and $y(t)$ is approximated) and instead computes the modified vector field \tilde{f}_h satisfying $\varphi_{h,\tilde{f}_h}(y(t_n)) = \Phi_{h,f}(y(t_n))$.

An important result, Theorem 3.2 in [18], which is very similar to Theorem 1.2 in [11, Ch. IX], states that if the method $\Phi_{h,f}$ is of order p , then the inverse modified vector field is a truncation of the true vector field, given by

$$f_h(y) = f(y) + h^p f_p(y) + \dots = f(y) + \mathcal{O}(h^p). \quad (9)$$

Furthermore, by the triangle inequality, we can express the objective function of the optimization problem (2) in a given point $y(t_n)$ by

$$\begin{aligned} \|y(t_{n+1}) - \Phi_{h,f_\theta}(y(t_n))\| &\leq \|\varphi_{h,f}(y(t_n)) - \Phi_{h,f_h}(y(t_n))\| \\ &\quad + \|\Phi_{h,f_h}(y(t_n)) - \Phi_{h,f_\theta}(y(t_n))\| \end{aligned}$$

In the case of formal analysis where we do not consider convergence issues and truncated approximations, the first term is zero by the definition of f_h in (8). Thus it is evident that the approximated vector field will approach the inverse modified vector field as the optimization objective tends to zero. Then, by Equation (9) it is clear that $f_\theta(y)$ will learn an approximation of $f(y)$ up to a truncation $\mathcal{O}(h^p)$, which motivates using an integrator of high order.

5 Numerical experiments

In this section, MIRK methods of order $2 \leq p \leq 6$, denoted by MIRK_p in the plots, in addition to the classic fourth-order Runge–Kutta method (RK4), is utilized for the temporal discretization in the training of Hamiltonian neural networks. We train on samples $y(t_n)$, for $t_n \in [0, 20]$, from solutions of the double pendulum (DP) problem with the Hamiltonian

$$H(y_1, y_2, y_3, y_4) = \frac{\frac{1}{2}y_3^2 + y_4^2 - y_3y_4 \cos(y_1 - y_2)}{1 + \sin^2(y_1 - y_2)} - 2 \cos(y_1) - \cos(y_2).$$

In addition, we consider the highly oscillatory Fermi–Pasta–Ulam–Tsingou (FPUT) problem with $m = 1$, meaning $y(t) \in \mathbb{R}^4$, and $\omega = 2$ as formulated in [11, Ch. I.5]. For both Hamiltonian systems, the data $S_N = \{y(t_i)\}_{i=0}^N$ is found by integrating the system using *DOP853* [7] with a tolerance of 10^{-15} for the following step sizes and number of steps: $(h, N) = (2, 10), (1, 20), (0.5, 40)$. The initial values used are $y_0^{\text{DP}} = [-0.1, 0.5, -0.3, 0.1]^T$ and $y_0^{\text{FPUT}} = [0.2, 0.4, -0.3, 0.5]^T$. The results for $[y(t)]_3$ are illustrated in Figure 1.

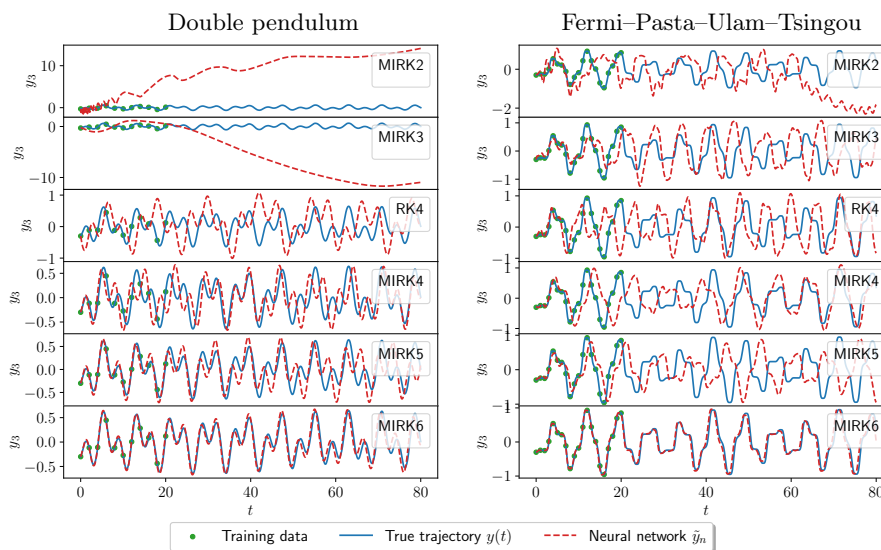


Fig. 1: Result when integrating over the learned vector fields when training on data from the double pendulum (left, $N = 10$) and the Fermi–Pasta–Ulam–Tsingou (right, $N = 20$) Hamiltonian.

After using the specified integrators in training, approximated solutions \tilde{y}_n are computed for each learned vector field f_θ again using *DOP853*, but now with step size and number of steps given by $(h_{\text{test}}, N_{\text{test}}) = (\frac{h}{20}, 4 \cdot 20N)$, enabling the

computation of the interpolation and extrapolation error:

$$e^l(\tilde{y}) = \frac{1}{M+1} \sum_{n=0}^M \|\tilde{y}_n - y(t_n)\|_2, \quad t_n \in Q^l, \quad M = |Q^l| - 1. \quad (10)$$

Here $\tilde{y}_{n+1} := \Phi_{h, f_\theta}(\tilde{y}_n)$, and $l \in \{i, e\}$ denotes interpolation or extrapolation: $Q^i = \{hn : 0 \leq hn \leq 20, n \in \mathbb{Z}_+\}$ and $Q^e = \{hn : 20 \leq hn \leq 80, n \in \mathbb{Z}_+\}$, with $h = h_{\text{test}}$. In addition, the error of the learned Hamiltonian is computed along the true trajectory $y(t_n)$ by

$$\begin{aligned} \overline{e(H_\theta)} &= \frac{1}{M+1} \sum_{n=0}^M H(y(t_n)) - H_\theta(y(t_n)), \\ e(H_\theta) &= \frac{1}{M+1} \sum_{n=0}^M \left| H(y(t_n)) - H_\theta(y(t_n)) - \overline{e(H_\theta)} \right|, \end{aligned} \quad (11)$$

for $t_n \in Q_i \cup Q_e$. The mean is subtracted since the Hamiltonian is only trained by its gradient ∇H_θ . The error terms are shown in Figure 2.

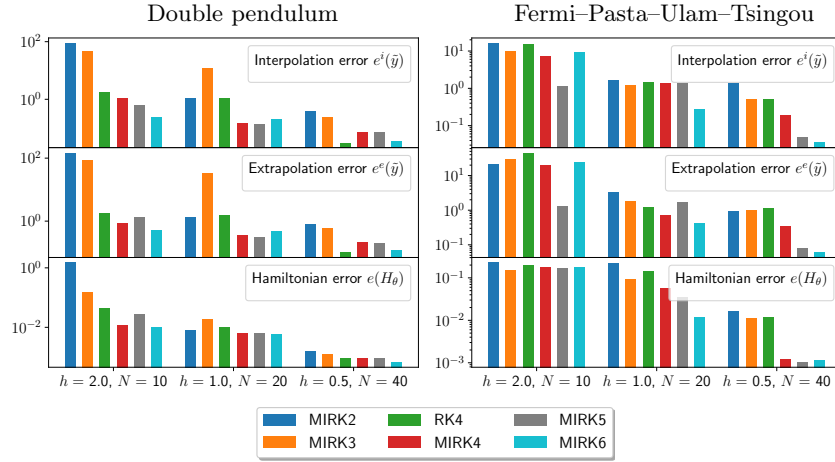


Fig. 2: Errors in interpolation, extrapolation and the Hamiltonian for the double pendulum (left) and the Fermi-Pasta-Ulam-Tsingou problem (right).

For both test problems the Hamiltonian neural networks have 3 layers with a width of 100 neurons and $\tanh(\cdot)$ as the activation function. Experiments are implemented using PyTorch [15] and the optimization problem is solved using the quasi-Newton L-BFGS algorithm [13] for 100 epochs without batching. The implementation of the experiments could be found in the following repository github.com/hakonnoeren/learning_hamiltonian_mirk.

6 Conclusion

The mono-implicit Runge–Kutta methods enable the combination of high order and computationally efficient training of Hamiltonian neural networks. The importance of high order is demonstrated by the remarkable capacity of MIRK6 in learning a trajectory of the chaotic double pendulum and the Fermi–Pasta–Ulam–Tsingou Hamiltonian systems from just 11 and 21 points, see Figure 1. In most cases the error, displayed in Figure 2, is decreasing when increasing the order. Additionally MIRK4 displays superior performance comparing with the explicit method RK4 of same order. Even though the numerical experiments show promising results, the theoretical error analysis in this work is rudimentary at best. Future work should consider this in greater detail, perhaps along the lines of [18].

Acknowledgments

The author wishes to express gratitude to Elena Celledoni and Sølve Eidnes for constructive discussions and helpful suggestions while working on this paper.

References

1. Burrage, K., Chipman, F., Muir, P.H.: Order results for mono-implicit Runge–Kutta methods. *SIAM journal on numerical analysis* **31**(3), 876–891 (1994)
2. Butcher, J.C.: *Numerical methods for ordinary differential equations*. John Wiley & Sons (2016)
3. Cash, J.R.: A class of implicit Runge–Kutta methods for the numerical integration of stiff ordinary differential equations. *Journal of the ACM (JACM)* **22**(4), 504–511 (1975)
4. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. *Advances in neural information processing systems* **31** (2018)
5. Chen, Z., Zhang, J., Arjovsky, M., Bottou, L.: Symplectic recurrent neural networks. In: *International Conference on Learning Representations* (2020), <https://openreview.net/forum?id=BkgYPREtPr>
6. David, M., Méhats, F.: Symplectic learning for Hamiltonian neural networks. arXiv preprint arXiv:2106.11753 (2021)
7. Dormand, J., Prince, P.: A family of embedded Runge–Kutta formulae. *Journal of Computational and Applied Mathematics* **6**(1), 19–26 (1980). [https://doi.org/https://doi.org/10.1016/0771-050X\(80\)90013-3](https://doi.org/https://doi.org/10.1016/0771-050X(80)90013-3), <https://www.sciencedirect.com/science/article/pii/0771050X80900133>
8. E, W.: A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics* **5**(1), 1–11 (2017). <https://doi.org/10.1007/s40304-017-0103-z>, <https://doi.org/10.1007/s40304-017-0103-z>
9. Greydanus, S., Dzamba, M., Yosinski, J.: Hamiltonian neural networks. *CoRR abs/1906.01563* (2019), <http://arxiv.org/abs/1906.01563>
10. Haber, E., Ruthotto, L.: Stable architectures for deep neural networks. *Inverse problems* **34**(1), 014004 (2017)

11. Hairer, E., Lubich, C., Wanner, G.: Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations; 2nd ed. Springer, Dordrecht (2006). <https://doi.org/10.1007/3-540-30666-8>
12. Muir, P.H.: Optimal discrete and continuous mono-implicit Runge-Kutta schemes for BVODEs. *Adv. Comput. Math.* **10**(2), 135–167 (1999). <https://doi.org/10.1023/A:1018926631734>, <https://doi.org/10.1023/A:1018926631734>
13. Nocedal, J., Wright, S.J.: Numerical optimization. Springer (1999)
14. Offen, C., Ober-Blöbaum, S.: Symplectic integration of learned Hamiltonian systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **32**(1), 013122 (2022)
15. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* **32**, 8026–8037 (2019)
16. Ruthotto, L., Haber, E.: Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision* **62**(3), 352–364 (2020)
17. Zhu, A., Jin, P., Tang, Y.: Deep Hamiltonian networks based on symplectic integrators. arXiv preprint arXiv:2004.13830 (2020)
18. Zhu, A., Jin, P., Zhu, B., Tang, Y.: On numerical integration in neural ordinary differential equations. In: *International Conference on Machine Learning*. pp. 27527–27547. PMLR (2022)