Machine Learning Methods for Autonomous Ordinary Differential Equations

Maxime Bouchereau, Philippe Chartier, Mohammed Lemou, Florian Méhats

Machine Learning Methods for Autonomous Ordinary Differential Equations

Maxime Bouchereau[1] Philippe Chartier[3] Mohammed Lemou[2]
Florian Méhats[4]

[1]Université de Rennes
[2]Ravel technologies, on leave from INRIA
[3]Ravel technologies, on leave from Centre National de la recherche Scientifique (CNRS)
[4]Ravel technologies, on leave from Université de Rennes

April 14, 2023

Abstract

Ordinary Differential Equations are generally too complex to be solved analytically. Approximations thereof can be obtained by general purpose numerical methods. However, even though accurate schemes have been developed, they remain computationally expensive: In this paper, we resort to the theory of modified equations in order to obtain "on the fly" cheap numerical approximations. The recipe consists in approximating, prior to that, the modified field associated to the modified equation by *neural networks*. Elementary convergence results are then established and the efficiency of the technique is demonstrated on experiments.

Keywords: modified equation, ordinary differential equation, neural network, numerical method, convergence analysis.

1 Introduction

Ordinary Differential Equations are ubiquitous in the modelling of systems in domains of science as diverse as biology, dynamics, fluid mechanics, quantum mechanics, thermodynamics or weather forecasting. In most situations, these differential equations can not be solved analytically and necessitate the use of numerical methods so as to compute accurate approximations [14, 15].

Generally speaking, the aforementioned methods are the result of a trade-off between accuracy and computational cost. In simple words, a small approximation error requires long computations. Of course, there are various ways in which one may soften the computational constraints, e.g. by

raising the order, taking into account the structure of the problem or optimising the coefficients of the method. Excellent numerical methods are abundant in the literature and we refer to reference books [3, 5, 6, 14, 15, 19] for them.

In this paper, we choose to derive as much as possible prior information from the knowledge of the vector field of the equation in order to accelerate the solving of the equation. This is thus a two-step process: (i) first, generate data that will be used during the effective computation of the approximate solution; (ii) second, compute, from an initial value, an approximation of the solution "on the fly" as accurately as possible by using available pre-computed values. As such, the process remains too vague to become practical: this is where the crucial ingredient of the technique comes into play, that is to say modified equations.

The theory of modified equations has emerged in the context of ordinary differential equations as a powerful tool (referred to as *backward error analysis*) to explain the excellent behaviour of structure-preserving numerical methods [14]. More recently, modified fields have also been used in a dual manner, as a technique to raise the order of any existing numerical method by twisting appropriately the original vector field [3]. The idea is that, by adding ad-hoc perturbation terms to the vector field and then solving the associated differential equation, one may compute higher-order approximations. It can be indeed proved that, at least at a formal level, for any numerical method, there exists a modified equation whose solution by the aforementioned method coincides with the exact solution. The perturbation terms to be added may be obtained in analytical form as *elementary differentials* involving various derivatives of the original vector field. Complete expansions are for instance available with the help of representations by trees (known as B-series). However, computing the corresponding expressions analytically would be an extremely tedious process and this is the reason why the technique has remained confined to specific situations [8, 14, 16] of limited practical interest.

The main idea of this work is thus to combine the theory of modified equations with machine learning techniques and more precisely neural networks. Given a differential equation and a numerical method, the ad-hoc perturbations of the vector field are first *learnt* by extensive simulations and then approximated by inference from a neural network. Once this representation of the modified vector field has been obtained, it is used to solve the original equation with the same numerical method for any initial value prescribed in a *learnt* domain from the phase-space. The combined computational work ids by far greater than for any usual reasonable numerical method. Nevertheless, if one omits the time spend to learn the perturbation, an accurate solution can be obtained very cheaply as compared to well-established schemes such as those of Dormand & Prince [23].

1.1 Scope of the paper

The article is divided into two main sections: Section 2 is devoted to the exposition of the technique and its convergence analysis, while Section 3 presents numerical experiments illustrating the performances and properties of the schemes we analysed.

Subsection 2.1 exposes the general strategy which is adopted. A specific structure of the neural network is selected, in agreement with the structure of the modified field. Moreover, the details of the machine leaning method for the learning of the modified field are given here, concerning in particular the choice of the training data set and the calibration of the parameters of the neural network (approximating the modified field via *loss*-minimization).

Subsection 2.2 establishes a convergence result for the numerical integration resulting from the

combined use of neural networks and classical schemes. The essential difference with standard convergence results is reflected in the multiplicative constant of the local error which turns out to be smaller than for a direct application to the original vector field. A special focus is put on explicit Runge-Kutta methods in the same subsection, where more precise estimates are given.

Moreover, two specific schemes are studied in this subsection: the forward Euler method and a Runge-Kutta method of order 2, which are two simple occurrences of explicit methods. A short convergence analysis is undertaken for each of them, leading to improved bounds. The same methods are then used for numerical experiments which illustrate two main contributions of this work: (i) The modified field can be learnt efficiently through a neural network, as is illustrated in 3.1; (ii) The resulting numerical methods have far greater efficiency, as is illustrated on convergence curves in Subsection 3.2. A comparison with the well-known DOPRI5 method in Subsection 3.3 is particularly enlightening with this respect.

1.2 Related work

The link between differential equations and machine learning has been already explored in several publications. Two approaches are prominent. On the one hand, the ODE vector field can be learnt by the technique of MSE Loss, which can be applied to both ODEs or PDEs [25]. Let us notice also a paper of Burton et al. [4] which proposes symbolic regression for the learning of complex dynamical systems. On the other hand, the ODE vector field can be learnt by statistical methods. for instance, Expectation-Maximization is used in a paper of Nguyen et al. [21], while a paper of Raissi et al. [24] proposes Gaussian processes for linear differential equations.

Links with modified equations. Links between machine learning and modified equations have been established more recently, e.g. in the paper [27] where the theory of modified equation is used fora rigorous analysis. Offen et al. [22] consider numerical methods for Hamiltonian ODEs. The methods used therein are statistical methods, namely Gaussian processes.

Structure of Neural Networks. In order to preserve geometric properties, specific neural networks, adapted to the structure of the differential equations under consideration have been developed. A first example are Hamiltonian equations for which Hamiltonian neural networks have been developed (see [10] or [18] where irregular time observed data can be used). A second example are Poisson systems for which Jin et al. [17] developed Poisson neural networks. Another example of specific ODEs is studied in another recent paper [28], where VPNets are used to learn the volume-preserving flows.

ODEs methods for Neural Networks. In the same way as neural networks are used to solve ODEs, the reciprocal strategy can be pursued: neural networks can indeed be modelled by ODEs and their properties deduced from the corresponding ODE properties. For instance, Lu et al. [20] have developed new neural networks which are discretizations of ODEs by various numerical methods and Haber et al. [13] have developed new structures of neural networks depending on the stability properties of the ODEs. Finally, Chen et al. [9] have derived new optimization methods of the loss function from the properties of the corresponding ODEs (the neural network is here again obtained through the discretization of the ODE).

Approximation by neural networks. In order to properly approximate functions by neural networks, error estimates have been established. Anastassiou [1] stated rates of convergence for approximations of functions by networks, according to the number of parameters and the dimension. By considering neural network spaces as functional spaces, Gribonval et al. [12] have obtained

inclusions of theses spaces in Besov or Lebesgue spaces, according to the number of parameters and a given rate of convergence. In a separate work, Bach [2] has given bounds on the approximation error in a Hilbertian setting. Finally, a problem of approximation [7] which is underlined is the curse of dimensionality, where high-dimensional vector fields are approximated with a slower rate of convergence than low-dimensional vector fields, i.e. for high dimensions, more parameters and more data will be required in order to get a satisfying learning.

2 Improving the accuracy of numerical methods with machine learning

Consider an autonomous ordinary differential equation of the form

$$\left\{ \begin{array}{rcl} \dot{y}(t) & = & f(y(t)) \in \mathbb{R}^d, \quad t \in [0,T] \\ y(0) & = & y_0 \end{array} \right. ,$$

where $f : \mathbb{R}^d \to \mathbb{R}^d$ is assumed to be smooth enough. By Cauchy-Lipschitz theorem, we have existence and uniqueness of a solution for any given initial value $y_0 \in \mathbb{R}^d$. We wish to approximate the solution over $[0,T]$ at times $t_n = nh$, $0 \leqslant n \leqslant N$, where $h = \frac{T}{N}$ is the time-step and N is the number of discretization points.

2.1 General strategy

As explained in the Introduction section, we shall approximate the modified vector field with the help of a neural network.

2.1.1 Modified field

Let us consider $\Phi_h^f(\cdot)$ the numerical flow associated to a given numerical method (h is the time-step of the method) and to the vector field f, and assume that it is of order p, in the sense that[1]

$$\underset{0 \leqslant n \leqslant N}{Max} \left| \left(\Phi_h^f \right)^n (y_0) - \varphi_{nh}^f(y_0) \right| \quad \leqslant \quad Ch^p \tag{1}$$

for some constants $C > 0$. If we modify the field f used in Φ_h, i.e. if we apply the numerical flow Φ_h with the modified field $\tilde{f}_h$ instead of f, we may obtain a higher-order approximation. In fact, the theory of *modified equations* states that it is possible to construct $\tilde{f}_h$ as a series of powers of h multiplied by appropriately chosen functions (at least as a formal series), in such a way that $(\Phi_h^{\tilde{f}_h})^n(y_0)$ coincides exactly with $y(t_n)$. The structure of this modified field writes (see [14])

$$\tilde{f}_h(y) = f(y) + h^p \sum_{j=1}^{+\infty} h^{j-1} f^{[j]}(y) = f(y) + h^p \sum_{j=1}^{k-1} h^{j-1} f^{[j]}(y) + h^{k+p-1} R(y,h) \tag{2}$$

[1]Here and in the sequel, $|\cdot|$ denotes a norm on $\mathbb{R}^d$.

where the coefficient-functions $f^{[j]}$ are built upon derivatives of f. It can be shown rigorously that the truncation of this formal series (2) obtained by neglecting the $O\left(h^{k+p-1}\right)$-terms, leads to

$$(\Phi_h^{\tilde{f}_h})^n(y_0) = \varphi_{nh}^f(y_0) + O\left(h^{k+p-1}\right)$$

where k denotes the number of terms kept in $\tilde{f}_h$.

2.1.2 Machine learning methods

The main idea of this paper consists in approximating the modified field $\tilde{f}_h$ by a neural network approximation $f_{app}(\cdot, h)$ whose structure mimics the structure of the theoretical modified field (2). More precisely, we shall approximate separately each function $f^{[j]}$ in (2) with a neural network. As could be anticipated, the truncation of (2) will be echoed by a similar truncation

$$f_{app}(y, h) = f(y) + h^p \sum_{j=1}^{N_t-1} h^{j-1} f_j(y) + h^{N_t+p-1} R_a(y, h), \tag{3}$$

where f_i, for $1 \leqslant i \leqslant N_t - 1$, and R_a, are Multi Layer Perceptrons. An obvious advantage of this choice is that the corresponding numerical method remains consistent. Let us further notice that learning the perturbation in this way is also well-adapted to the situations where the original vector field possesses a specific structure [10, 17, 18, 25, 28].

The complete numerical procedure can be decomposed into three main steps : Firstly, data are collected by simulating very accurately the exact flow at various points of the domain. A high number of simulations and a high accuracy are prerequisite for a good approximation of the modified field. Secondly, the different neural networks are trained separately by minimising a prescribed *Loss*-function. Eventually, given an initial data y_0, an approximation of the exact solution is obtained by applying the same numerical scheme as the one used for the training to the field f_{app}. In more details, we follow the three stages:

1. **Construction of the data set:** K initial data $y_0^{(k)}$ are randomly selected into a compact set $\Omega \subset \mathbb{R}^d$ (where we wish to simulate the solution) with uniform distribution. Then, for all $0 \leqslant k \leqslant K-1$, we compute a very accurate approximation of the the exact flow at times $h^{(k)}$ with initial condition $y_0^{(k)}$, denoted $y_1^{(k)}$. Time steps $h^{(k)}$ are selected in the domain $[h_-, h_+]$ (we actually pick up the value $\log h^{(k)}$ randomly in the domain $[\log h_-, \log h_+]$ with uniform distribution).

2. **Training of the neural networks:** We minimize the Mean Squared Error (MSE), denoted $Loss_{Train}$, which measures the difference between predicted data $\hat{y}_1^{(k,\ell)}$ and "exact data" $y_1^{(k,\ell)}$ by computing the optimal NN's parameters over K_0 data (where $1 \leqslant K_0 \leqslant K-1$) by resorting to a gradient method:

$$Loss_{Train} = \frac{1}{K_0} \sum_{k=0}^{K_0-1} \frac{1}{h^{(k)^{2p+2}}} \left| \underbrace{\Phi_{h^{(k)}}^{f_{app}(\cdot,h^{(k)})}\left(y_0^{(k)}\right)}_{=\hat{y_1}^{(k)}} - \underbrace{\varphi_{h^{(k)}}^f\left(y_0^{(k)}\right)}_{=y_1^{(k)}} \right|^2 \tag{4}$$

At the same time, we compute the value of another MSE, denoted $Loss_{Test}$, which measures the difference between predicted data $\hat{y}_1^{(k,\ell)}$ and "exact data" $y_1^{(k,\ell)}$ for a subset of the initial values which have not been used to train the NNs. The objective of this step is to estimate the performance of the training for "unknown" initial values :

$$Loss_{Test} \quad = \quad \frac{1}{K-K_0}\sum_{k=K_0}^{K-1}\frac{1}{h^{(k)2p+2}}\left|\Phi_{h^{(k)}}^{f_{app}(\cdot,h^{(k)})}\left(y_0^{(k)}\right)-\varphi_{h^{(k)}}^{f}\left(y_0^{(k)}\right)\right|^2 \tag{5}$$

If $Loss_{Train}$ and $Loss_{Test}$ exhibit the same decay pattern, one considers that there is no overfitting, that is to say that the neural network model does not fit exactly against its training data and remains able to perform accurately against unseen data, which is its main purpose.

3. **Numerical approximation:** At the end of the training, an accurate approximation $f_{app}(\cdot,h)$ of $\tilde{f}_h$ is available. It is then used to compute the successive values of $(\Phi_h^{f_{app}(\cdot,h)})^n(y_0)$ for $n=0,\ldots,N$.

2.2 Error analysis

In this subsection, we analyse the error resulting from the procedure described in previous Subsection. More specifically, we state estimates of the global error for any standard numerical method.

Theorem 1. *Let us denote $\Phi_h^{f_{app}(\cdot,h)}$, the flow of a given numerical scheme Φ_h of order p, applied to the modified field $f_{app}(\cdot,h)$ and let us consider the global error*

$$e_n \quad := \quad \left(\Phi_h^{f_{app}(\cdot,h)}\right)^n(y_0)-\left(\varphi_h^f\right)^n(y_0), \tag{6}$$

at times $t_n=nh$ for $0\leqslant n\leqslant N$. Denoting the learning error by

$$\delta \quad := \quad \underset{(y,h)\in\Omega\times[h_-,h_+]}{\text{Max}}\frac{\left|\tilde{f}_h(y,h)-f_{app}(y,h)\right|}{h^p} \tag{7}$$

and assuming that

(i) *For any pair smooth vector fields f_1 and f_2, we have*

$$\forall 0\leq h\leq h_+,\quad \left\|\Phi_h^{f_1}-\Phi_h^{f_2}\right\|_{L^\infty(\Omega)} \quad\leqslant\quad Ch\,||f_1-f_2||_{L^\infty(\Omega)} \tag{8}$$

for some positive constant C, independent of f_1 and f_2;

(ii) *For any smooth vector field f, there exists a constant $L>0$ such that*

$$\forall 0\leq h\leq h_+,\ \forall(y_1,y_2)\in\Omega^2,\quad \left|\Phi_h^f(y_1)-\Phi_h^f(y_2)\right| \quad\leqslant\quad (1+Lh)\,|y_1-y_2|\,. \tag{9}$$

Then there exist two constants $\tilde{C},\tilde{L}>0$ such that:

$$\underset{0\leqslant n\leqslant N}{Max}\,|e_n| \quad\leqslant\quad \frac{C\delta h^p}{\tilde{L}}\left(e^{\tilde{L}T}-1\right) \tag{10}$$

Proof. The arguments of the proof are completely standard ans thus omitted. ∎

Remarks. ***(i)*** *The vector fields f and $\tilde{f}_h$ are smooth, respectively by assumption and by construction. As for $f_{app}(\cdot, h)$, it is smooth as well given that it is obtained through the composition of affine functions $A_1, \cdots, A_{L+1}$ and nonlinear functions $\Sigma_1, \cdots, \Sigma_L$ (the so-called activation functions). The output of the NN thus appears to be of the form $A_{L+1} \circ \Sigma_L \circ A_L \circ \cdots \Sigma_1 \circ A_1$ (for L layers). Hence, if the activation functions are smooth, then so is $f_{app}(\cdot, h)$. This is the case for instance if the Σ_i's are the hyperbolic tangent functions.*

(ii) *Assumption (8) is straightforwardly satisfied for all known consistent methods.*

(iii) *A similar error estimate holds for a variable step-size implementation of the numerical method Φ: if we indeed use the sequence of steps $0 \leq h_j \leq h_+$, then $T = h_0 + \cdots, +h_{N-1}$ and $y^*_{n+1} = \Phi_{h_n}^{f_{app}(\cdot,h_n)}\left(y^*_n\right)$, then there exist $\tilde{C}, \tilde{L} > 0$ such that:*

$$\underset{0\leqslant n\leqslant N}{Max} \left| \Phi_{h_{n-1}}^{f_{app}(\cdot,h_{n-1})} \circ \ldots \circ \Phi_{h_0}^{f_{app}(\cdot,h_0)}(y_0) - \varphi^f_{h_{n-1}} \circ \ldots \circ \varphi^f_{h_0}(y_0) \right| \quad \leqslant \quad \frac{\tilde{C}\delta h^p}{\tilde{L}} \left(e^{\tilde{L}T} - 1 \right),$$

where $h = \underset{0\leqslant j\leqslant N-1}{\text{Max}} h_j$.

We now focus on numerical schemes belonging to the class of explicit Runge-Kutta methods, as this allows to specify some of the constants of previous Theorem. Following Remark (i), we shall assume that $f_{app}(\cdot, h)$ is well-defined and smooth on the compact set $\Omega \times [t_-, t_+]$, so that it is Lipschitz with Lipschitz constant

$$\lambda \quad := \quad \underset{h\in H}{Max} \left\| df_{app}(\cdot, h) \right\|_{L^\infty(\Omega)}. \tag{11}$$

Corollary 1. *Suppose that the numerical scheme Φ_h from Theorem 1 is the Runge-Kutta method with Butcher tableau (A, b) where $A = (a_{i,j})_{1\leqslant j\leqslant i\leqslant s} \in \mathcal{M}_s(\mathbb{R})$ and $b = (b_j)_{1\leqslant j\leqslant s} \in \mathbb{R}^s$. Assume further that A is strictly lower triangular, so that the scheme is explicit, and that it is of order p. Then inequality (11) of Theorem 1 holds with $\tilde{C} = \alpha$ and $\tilde{L} = \alpha\lambda$ where*

$$\alpha \quad = \quad ||b||_1 \left(1 + \lambda h_+ \, ||A||_\infty \, e^{\lambda h_+ ||A||_\infty} \right).$$

Remarks. ***1.*** *As the approximation $f_{app}(\cdot, h)$ of $\tilde{f}_h$ contains an $O(h^p)$-error term, the order of the new numerical procedure coincides with the order of the underlying scheme Φ_h. However, as soon as the NN becomes large, δ is small enough for the combined procedure to be significantly more accurate then the simple application of Φ_h.*

2. *For $N_t = 1$, with 1 hidden layer, we have density of MLP's in $C^1(\Omega)$ for the Sobolev norm $W^{1,\infty}$ [18]. If we have $\underset{h\in H}{\text{Max}} \left\| \frac{|\tilde{f}_h(\cdot) - f_{app}(\cdot,h)|}{h^p} \right\|_{W^{1,\infty}(\Omega)} \leqslant \delta$, then we get*

$$\lambda \quad \leqslant \quad \underset{h\in H}{Max} \left\| d\tilde{f}_h \right\|_{L^\infty(\Omega)} + \delta h_+^p. \tag{12}$$

3. *Error estimates for the Forward Euler and the sol-called RK2 methods may be slightly improved. One has indeed*

$$\underset{0\leqslant n\leqslant N}{Max} \left| e^*_n \right| \leqslant \frac{\delta h}{\lambda} \left(e^{\lambda T} - 1 \right) \quad \text{and} \quad \underset{0\leqslant n\leqslant N}{Max} \left| e^*_n \right| \leqslant \frac{\delta h^2}{\lambda} \left(e^{\lambda\left(1+\frac{\lambda h_+}{2}\right)T} - 1 \right),$$

for respectively Forward Euler and RK2 methods.

2.3 An alternative method for parallel training

In this subsection, we show how to learn the modified field in an alternative way. The main idea consists in training separately each term (say for instance of the modified field for the forward Euler method), by creating different data sets for different time steps $h_1 < \cdots < h_{N_h}$:

$$y_j \quad = \quad y_0 + h_j f(y_0) + h_j^2 f^{[1]}(y_0) + \cdots + h_j^{N_t} f^{[N_t-1]}(y_0) + h_j^{N_t+1} R(y_0, h_j). \tag{13}$$

Note that in contrast with previous method, the step-size is not chosen at random for each initial value. We then obtain a linear system which can be solved by using the *generalized inverse* of a matrix. The solution of this linear system encompasses the values of $f^{[1]}(y_0)$, ..., $f^{[N_t-1]}(y_0)$ and $R(y_0, h_1)$, ..., $R(y_0, h_{N_h})$ which correspond to data usable for learning each term of the modified field separately.

The main advantages of this method are a shorter training-time (at least on a parallel machine), thus allowing for a larger number of data, and a smaller computational time (again on a parallel machine) when it comes to obtaining the numerical solution from trained values.

3 Numerical experiments

In order to illustrate our theoretical results, we have tested the method given in Section 2.1 for two simple dynamical systems used from simple physics:

1. **The Non-linear Pendulum:** This system describes the movement of a pendulum under the influence of gravity. It is governed by the equations

$$\begin{cases} \dot{y_1} &= -\sin(y_2) \\ \dot{y_2} &= y_1 \end{cases},$$

   where y_2 denotes the angle of the pendulum with respect to the vertical and y_1 its angular velocity. Note that the system is Hamiltonian, see [10, 14, 18]. Parameters are given in Appendix B.2.1 for the forward Euler method, Appendix B.2.3 for the Runge-Kutta 2 method and Appendix B.2.4 for the midpoint rule.

2. **The Rigid Body system:** This is a three-dimensional system which describes the angular rotation of a solid in the physical space

$$\begin{cases} \dot{y_1} &= \left(\frac{1}{I_3} - \frac{1}{I_2}\right) y_2 y_3 \\ \dot{y_2} &= \left(\frac{1}{I_1} - \frac{1}{I_3}\right) y_1 y_3 \\ \dot{y_3} &= \left(\frac{1}{I_2} - \frac{1}{I_1}\right) y_1 y_2 \end{cases},$$

   where y_1, y_2 and y_3 denote the angular momenta, and I_1, I_2 and I_3 the momenta of inertia [14] (we take here $I_1 = 1$, $I_2 = 2$, $I_3 = 3$). It possesses two invariants, the so-called Casimir $C(y) = \frac{1}{2}|y|^2$ and the energy $H(y) = \frac{1}{2}\left(\frac{y_1^2}{I_1} + \frac{y_2^2}{I_2} + \frac{y_3^2}{I_3}\right)$. Hence, the solution lies at the

intersection of the sphere $|y|^2 = |y(0)|^2$ and of the ellipsoïd $\frac{y_1^2}{I_1} + \frac{y_2^2}{I_2} + \frac{y_3^2}{I_3} = 2H(y(0))$. The domain Ω used for training is thus chosen accordingly. Parameters are given in Appendix B.2.2.

For the *Forward Euler* and *RK2* methods, the modified field (2) can be computed by recursive formulae based on various derivatives of f, see for instance [8, 14]. It is represented by a series whose general term $f^{[j]}$ has an explicit -though complicated- expression, which can be compared with its numerical counterpart, obtained by learning it from the data set. For all $y \in \mathbb{R}^d$, $1 \leqslant j \leqslant k-1$, we have on the one hand

$$f^{[1]}(y) = \frac{1}{2} df(y) f(y) \tag{14}$$

$$f^{[j]}(y) = \frac{1}{j+1} df^{[j-1]}(y) f(y) \tag{15}$$

for the Forward Euler method, and on the other hand

$$f^{[1]}(y) = \frac{1}{24} d(df \cdot f)(y) f(y) + \frac{1}{8} df(y)^2 f(y) \tag{16}$$

$$f^{[2]}(y) = \frac{1}{24} d\left(d\left(df \cdot f\right) \cdot f\right) f(y) - \frac{1}{2} df(y) f^{[1]}(y) - \frac{1}{2} df^{[1]}(y) f(y). \tag{17}$$

for the Runge-Kutta 2 method. Formulas associated to the midpoint method are given in [8]. Truncating the formal power series (2) then gives an approximation of the theoretical modified field, which serves as a reference

$$\tilde{f}_h(y) = f(y) + h^p \sum_{j=1}^{k-1} h^{j-1} f^{[j]}(y) + O\left(h^{k+p-1}\right). \tag{18}$$

Here, p is the order of the numerical method under consideration.

3.1 Approximation of the modified field

In this subsection, we study the approximation error between the learned modified field (3) and the theoretical modified field (2) for the nonlinear Pendulum. We observe the learning error w.r.t. both space and time step variables. More precisely, we plot the function

$$g_h^k : x \mapsto \frac{1}{h^p} \left| \tilde{f}_h^{\,k}(x) - f_{app}(x, h) \right| \tag{19}$$

for $k = 4$ over the domain $\Omega = [-2, 2]^2$ in order to study the learning error in space, where the O-term in $\tilde{f}_h(y)$ is simply neglected. We furthermore represent $\underset{\Omega}{\mathrm{Max}} g_h^k$ for several values of time steps h, in order to study the learning error in function of the the time step. Note that we clearly get the expected order of convergence of $f_{app}(\cdot, h)$ towards the modified field $\tilde{f}_h$, with the exception of a plateau for small values of h .

Figures 1,2 and 3 show that the error g_h^4 is globally constant at the center of the domain and grows near its boundaries (see [10] where a similar behaviour is observed).

Altogether, these experiments confirm that the modified field can be appropriately learned with our neural network.

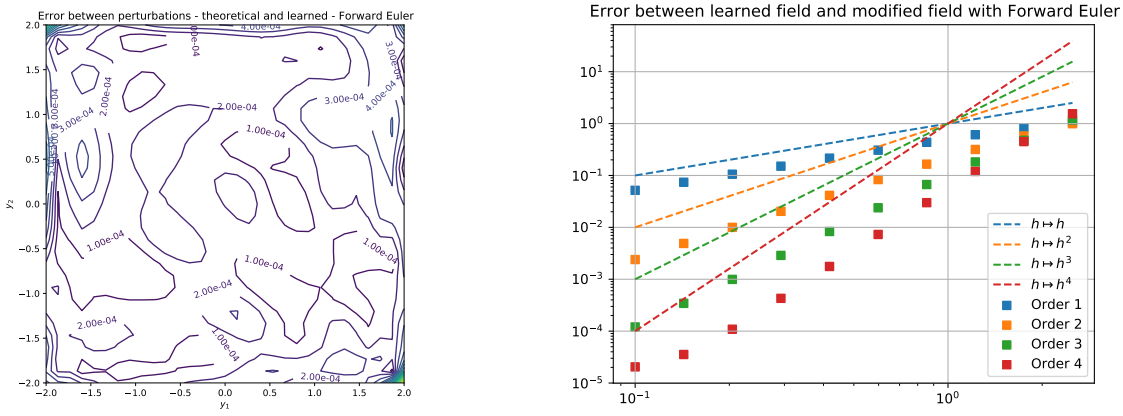

Figure 1: Forward Euler method. Left: Difference between $\tilde{f}_h^{\,4}$ and $f_{app}(\cdot, h)$ for $h = 0.1$. Right: Error between $\tilde{f}_h^{\,k}$ and $f_{app}(\cdot, h)$ for $1 \leqslant k \leqslant 4$.

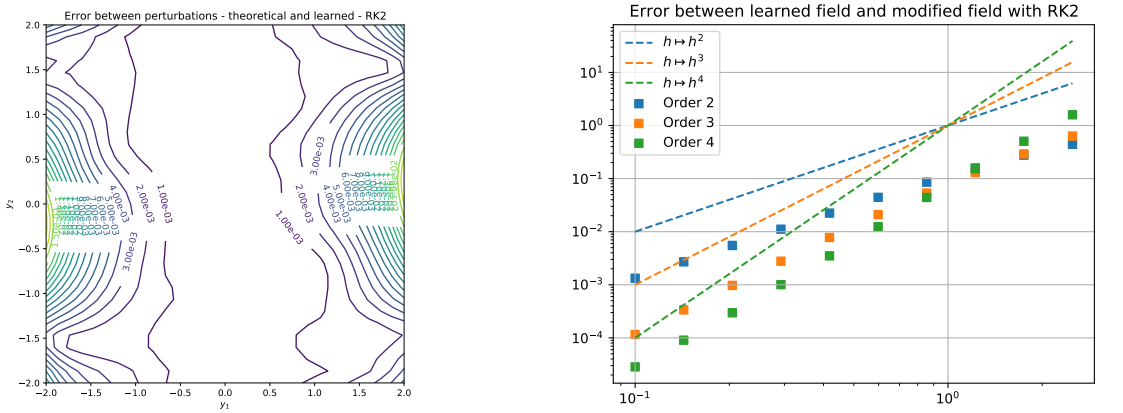

Figure 2: Runge-Kutta 2 method. Left: Difference between $\tilde{f}_h^{\,4}$ and $f_{app}(\cdot, h)$ for $h = 0.1$. Right: Error between $\tilde{f}_h^{\,k}$ and $f_{app}(\cdot, h)$ for $1 \leqslant k \leqslant 4$.

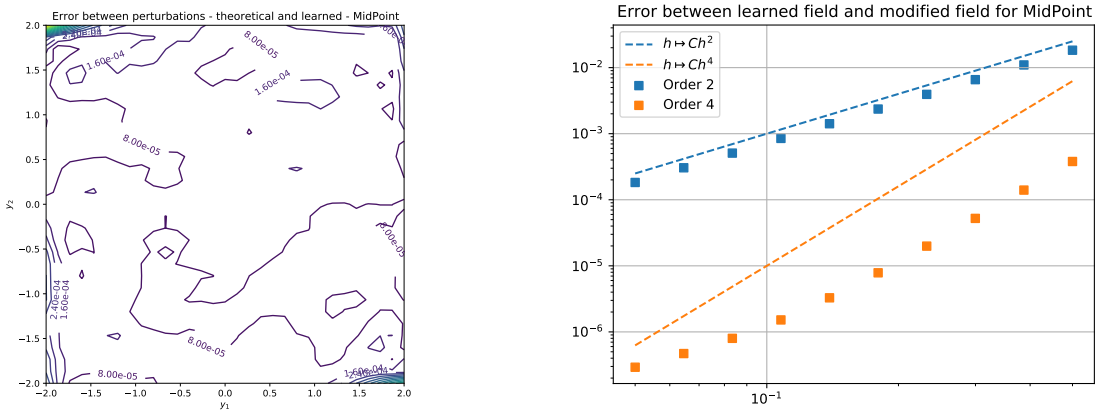

Figure 3: Midpoint method. Left: Difference between $\tilde{f}_h{}^4$ and $f_{app}(\cdot, h)$ for $h = 0.05$. Right: Error between $\tilde{f}_h{}^k$ and $f_{app}(\cdot, h)$ for $1 \leqslant k \leqslant 4$. Note that owing to the structure of the modified field for midpoint method (see [8, 14]), terms for odd powers of h vanish.

3.2 Loss decay and Integration of ODE's

Now, in order to compare, for a given method, the integration of a dynamical system with the original field and with the learned modified field, we will solve the nonlinear Pendulum with the Forward Euler, Runge-Kutta 2 and midpoint methods and the Rigid Body system with the Forward Euler method. However, prior to that, we study the decays of the $Loss$-functions for the training and testing data sets ($Loss_{Train}$ and $Loss_{Test}$). Their similarity is a good indication that there is no overfitting (the size of the training data set is thus appropriately estimated). As the MSE $Loss$ is used, it gives an idea of the value of the square of the learning error.

Figures 4, 5 and 6 show a more accurate numerical integration by using the corresponding learned modified field $f_{app}(\cdot, h)$ than using f. Moreover, exact flow and numerical flow with $f_{app}(\cdot, h)$ seem identical due to the small numerical error.

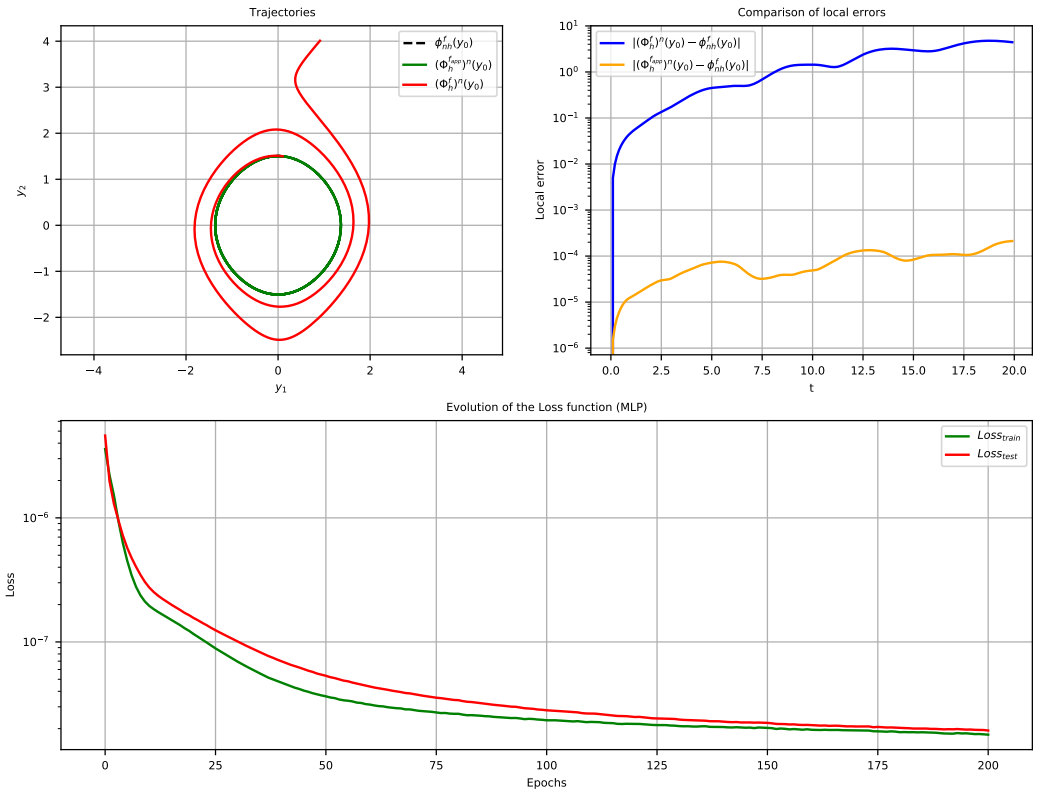

Figure 4: Comparison between $Loss$ decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, red: numerical flow with f, green: numerical flow with $f_{app}(\cdot, h)$) and local error (blue: exact flow and numerical flow with f, yellow: exact and numerical flow with $f_{app}(\cdot, h)$) for the nonlinear pendulum with Forward Euler method.

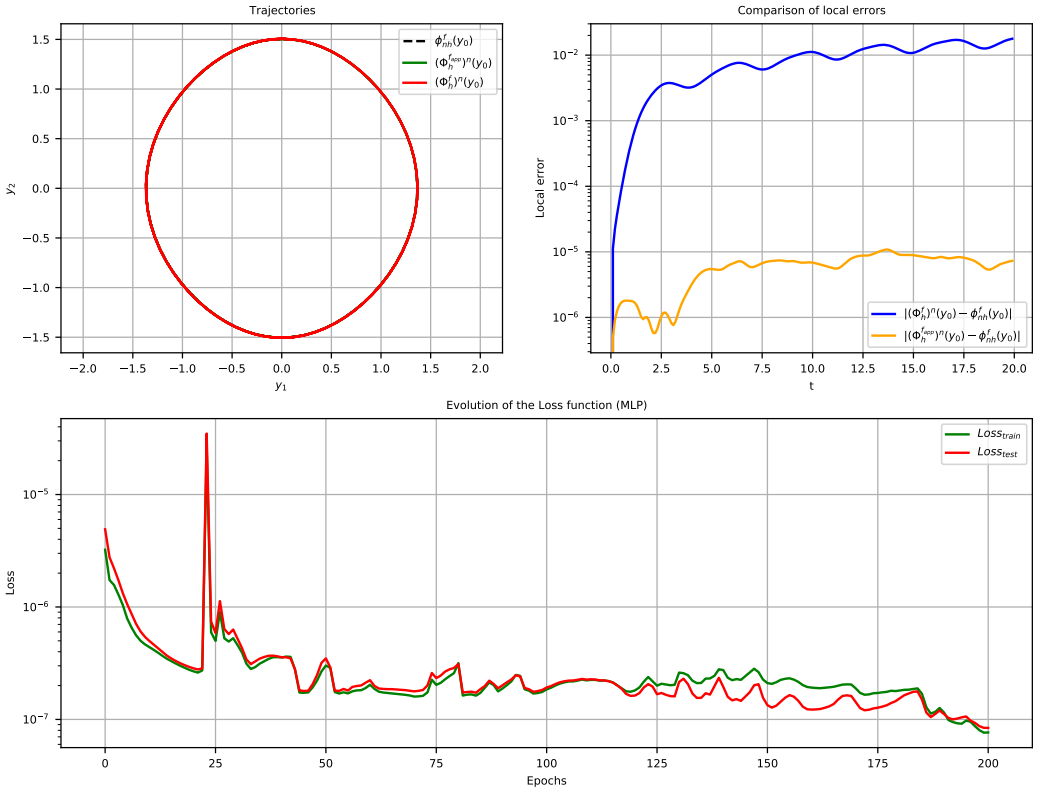

Figure 5: Comparison between $Loss$ decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, red: numerical flow with f, green: numerical flow with $f_{app}(\cdot, h)$) and local error (blue: exact flow and numerical flow with f, yellow: exact and numerical flow with $f_{app}(\cdot, h)$) for the nonlinear pendulum with Runge-Kutta 2 method.

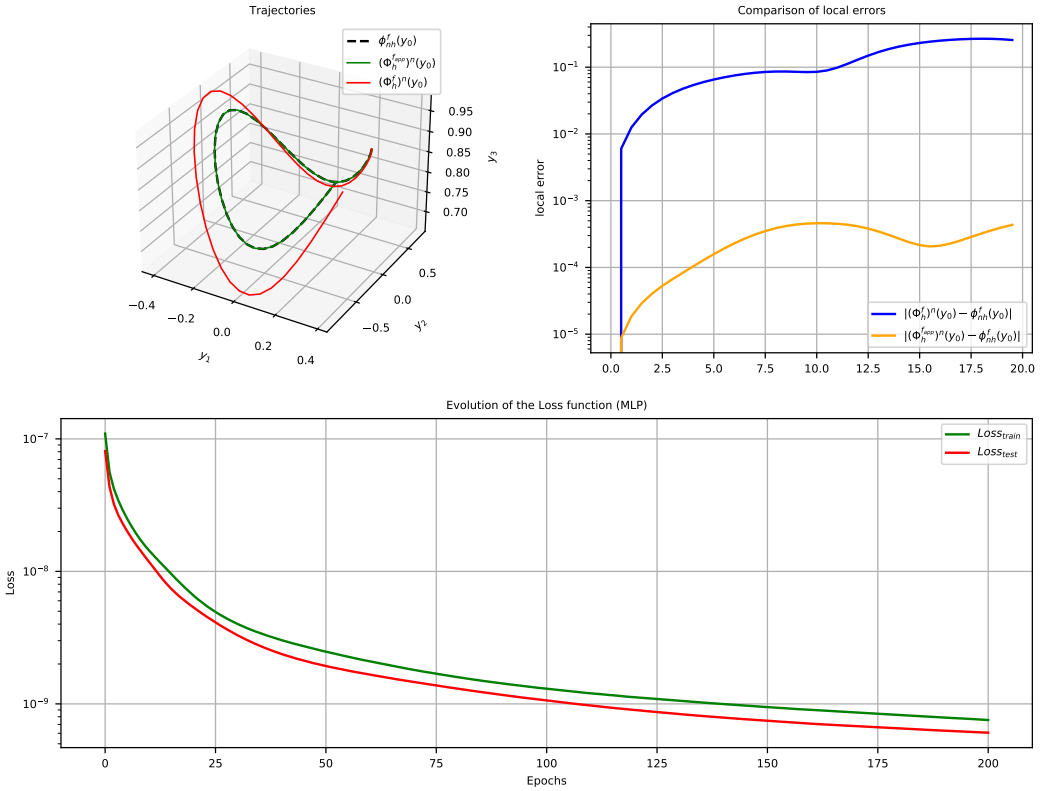

Figure 6: Comparison between $Loss$ decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, red: numerical flow with f, green: numerical flow with $f_{app}(\cdot, h)$) and local error (blue: exact flow and numerical flow with f, yellow: exact and numerical flow with $f_{app}(\cdot, h)$) for the Rigid Body system with Forward Euler method.

As the midpoint method is a symmetric and symplectic method, it is known to preserve accurately the geometric properties of the model. In order to evaluate the extent to which this feature persists in our context, we simply plot the value of the Hamiltonian along the numerical solution obtained from learned data. We test this method for the pendulum system, which is hamiltonian. Figure 7 shows a smaller error for integration with $f_{app}(\cdot, h)$ by using the midpoint method than integration with f. Moreover, the hamiltonian function of the pendulum system, given by

$$H : y \quad \mapsto \quad \frac{1}{2}y_1^2 + (1 - \cos(y_2)) \tag{20}$$

is preserved by the midpoint method with $f_{app}(\cdot, h)$ with smaller oscillations than midpoint with f. Preservation is better than DOPRI5 too, which is a non-symplectic method, as shown in Figure 8.

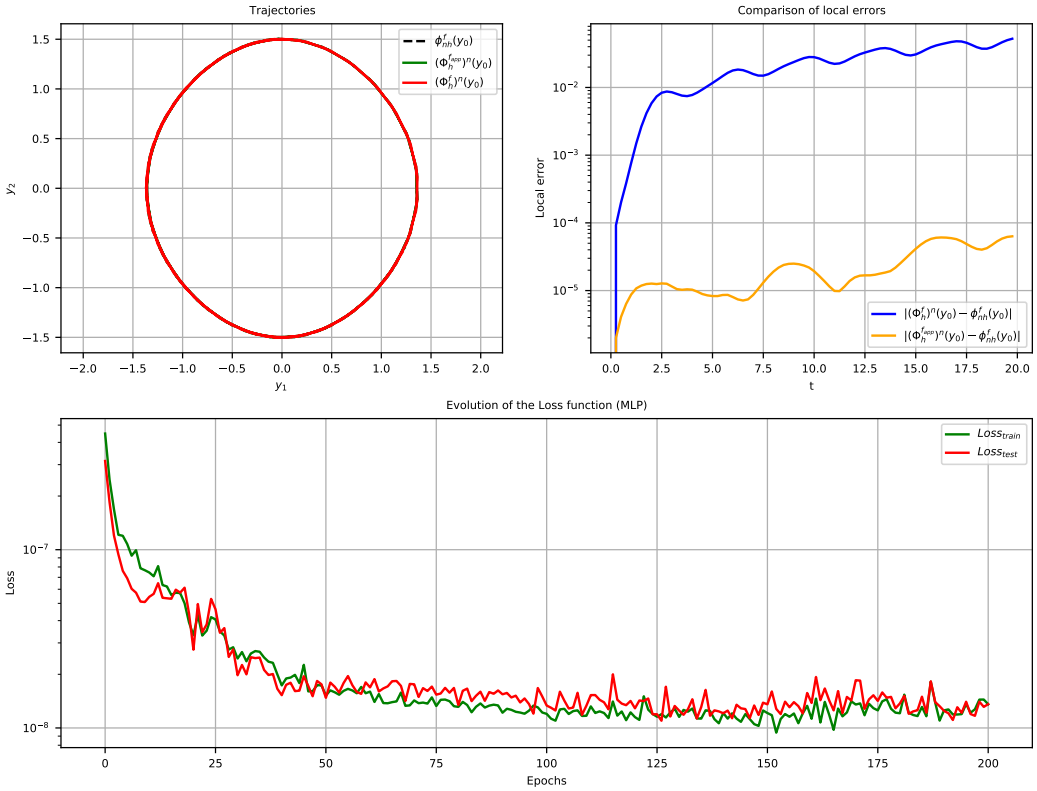

Figure 7: Comparison between $Loss$ decays (green: $Loss_{Train}$, red: $Loss_{Test}$), trajectories (dashed dark: exact flow, red: numerical flow with f, green: numerical flow with $f_{app}(\cdot, h)$) and local error (blue: exact flow and numerical flow with f, yellow: exact and numerical flow with $f_{app}(\cdot, h)$) for the nonlinear pendulum with midpoint method.

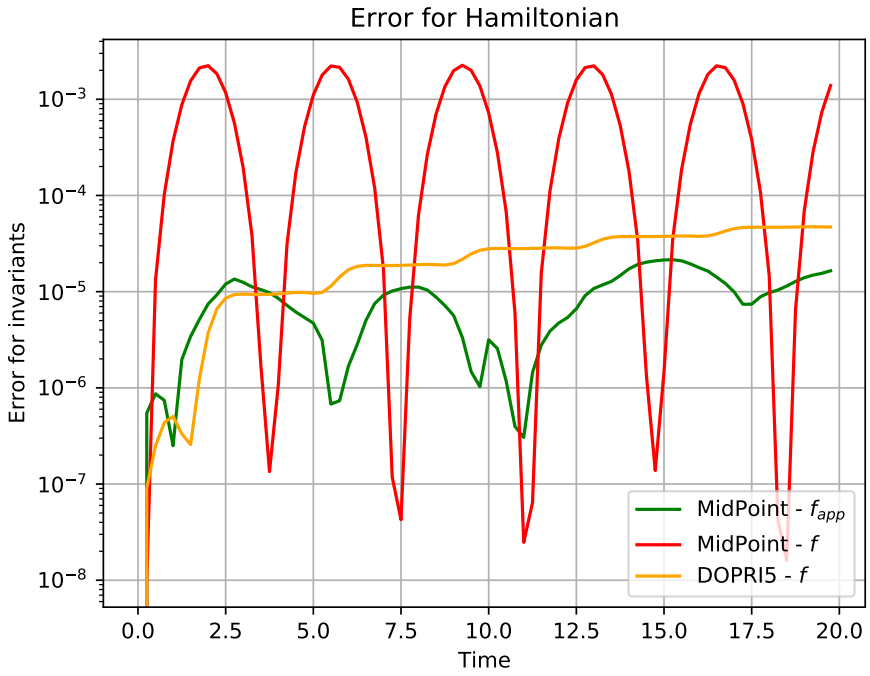

Figure 8: Evolution of the error between Hamiltonian $H : y \mapsto (1 - \cos(y_2)) + \frac{1}{2}y_1^2$ over the numerical flow and Hamiltonian at $t = 0$, $H(y_0)$.

Eventually, we study the global error between the exact flow and the approximation obtained from the original field, as well as the error between the exact flow and the numerical flow obtained form the learned modified field. The errors are plot as functions of the step-size and the curves are in perfect agreement with the estimates of previous theorems (for the Forward Euler, Runge-Kutta 2 and midpoint methods).

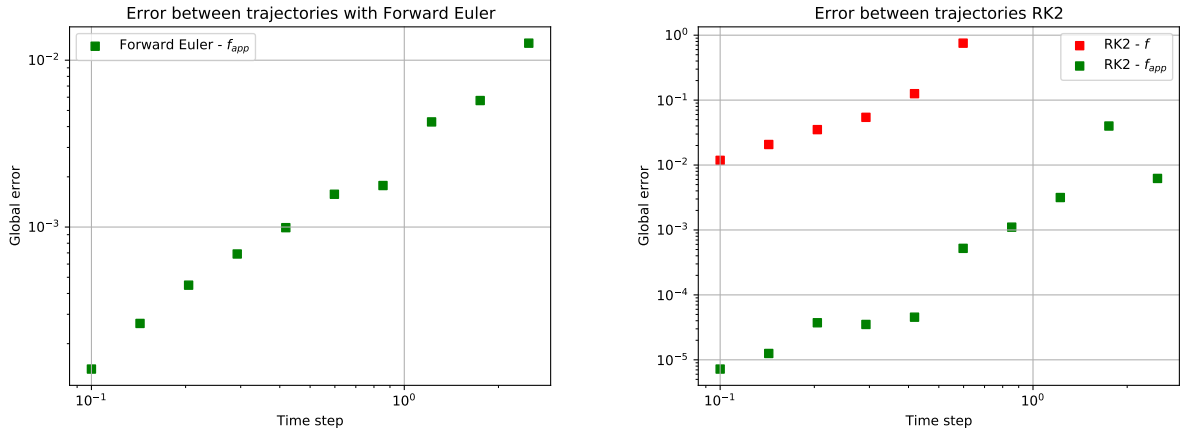

Figure 9: Integration errors (green: integration with f, red: integration with $f_{app}(\cdot, h)$). Left: Nonlinear Pendulum with Forward Euler. Right: Nonlinear Pendulum with Runge-Kutta 2.

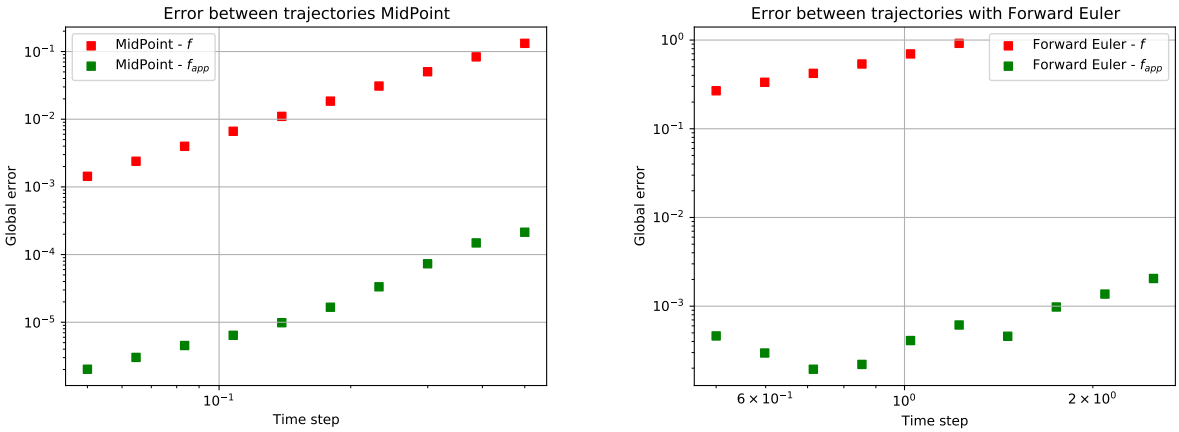

Figure 10: Integration errors (green: integration with f, red: integration with $f_{app}(\cdot, h)$). Left: midpoint method for the nonlinear Pendulum. Right: Forward Euler method for the Rigid Body System.

Note that the estimate of Theorem 1 is confirmed by Figures 9 and 10 , with a smaller multiplicative constant for the RK2 method.

3.3 Computational times for explicit methods

In this subsection, we plot efficiency curves (global error w.r.t. computational time). As the main goal of this paper is to design cheaper and/or more accurate solvers, we shall compare our results with the state-of-the-art Dormand & Prince methods [15].

Figures 11 and 12 show numerical errors for explicit methods (Forward Euler and Runge-Kutta 2) with $f_{app}(\cdot, h)$ can be smaller than numerical errors for DOPRI5 with f, especially for large time steps.

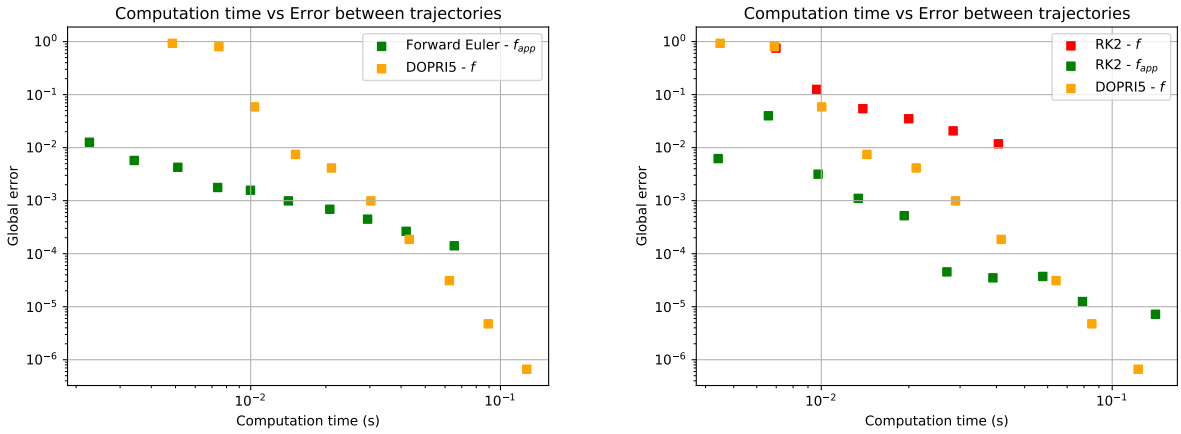

Figure 11: Comparison between computational time and integration error (red: numerical method with f, green: integration with $f_{app}(\cdot, h)$, yellow: integration with DOPRI5). Left: Nonlinear Pendulum with Forward Euler. Right: Nonlinear Pendulum with Runge-Kutta 2.

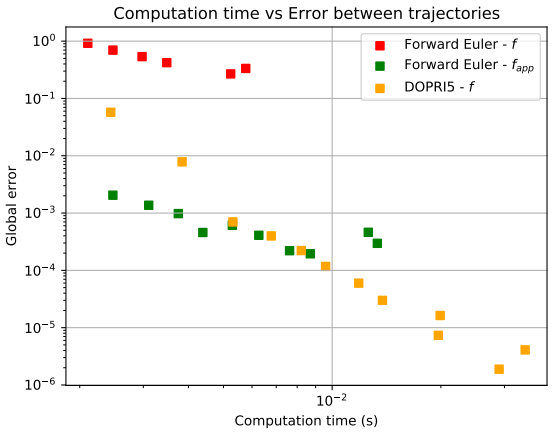

Figure 12: Comparison between computational time and integration error (red: numerical method with f, green: integration with $f_{app}(\cdot, h)$, yellow: integration with DOPRI5) for Rigid Body system with Forward Euler method.

Besides, we compare the integration error with the learned modified field and the integration error using truncated modified field at the order k $\tilde{f}_h^{\,k}$. For the forward Euler method, use this numerical method with $\tilde{f}_h^{\,k}$ will give a numerical method of order k, called *modified Euler* [10, 14].

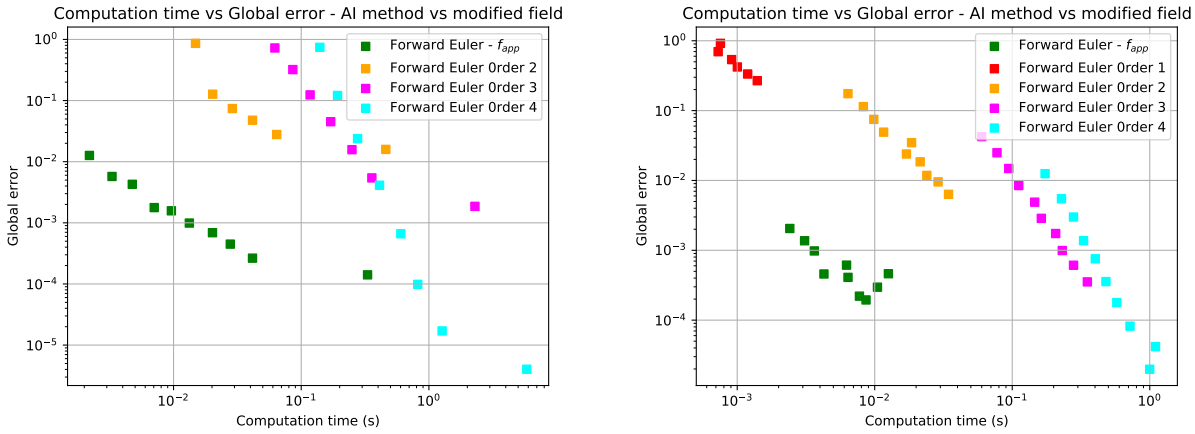

Figure 13: Computational time versus integration error for the Forward Euler method (red: with f, yellow: with $\tilde{f}_h^{\,2}$, magenta: with $\tilde{f}_h^{\,3}$, cyan: with $\tilde{f}_h^{\,4}$, green: with $f_{app}(\cdot, h)$. Left: Nonlinear Pendulum. Right: Rigid Body system.

Figure 13 shows for both systems that integration with $\tilde{f}_h^{\,k}$ is more expensive in time or less accurate that integration with $f_{app}(\cdot, h)$. Moreover, we always conserve an interesting integration error whereas the other methods are not accurate at all for small computational time.

3.4 Evaluation of alternative method

In this subsection, a comparison between the two learning techniques is illustrated. For the comparison to be fair, we adapt the volume of data so that the training times are nearly identical.

Figure 14 shows that both methods are able to generalize for smaller time steps than those used for training. In the two cases, traditional and alternative method with parallel training give approximately identical results

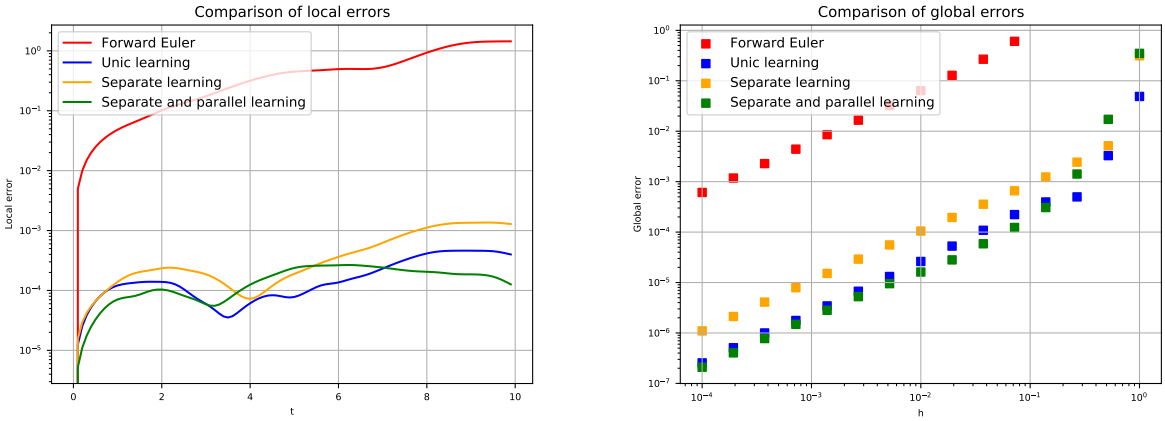

Figure 14: Comparison between forward Euler, traditional and alternative method for simple pendulum. Left: Comparison of local errors. Right: Comparison of integration errors.

4 Conclusions

The numerical experiments presented in this paper demonstrate that learning the modified vector field is very beneficial in terms of efficiency. Clearly, the training of the network is an overload which should be considered separately for real-time applications. Another interesting outcome of our study is the fact that even when explicit formulae of the various terms of the modified field are available, it is advantageous to use its learned counterpart. Eventually, even though the technique used here is not fitted to situations where invariants should be conserved (learning directly the modified Hamiltonian would definitely be a better option), the learned vector field approximately retains the properties of its exact counterpart. It remains to be emphasized that the problems that we solved here are of small dimension and further studies with larges systems (for instance originating from PDEs) are needed.

Acknoledgements

The authors of this paper would like thank Pierre Navaro for his advises in the implementation aspects.

Bibliography

[1] Anastassiou, G. *Quantitative approximations*. Chapman and Hall/CRC, 2000.

[2] Bach, F. *Learning Theory from First Principles*. Draft of a book, version of Sept, 6, 2021.

[3] Beyn, W. J., Dieci, L., Guglielmi, N., Hairer, E., Sanz-Serna, J. M., Zennaro, M. Current Challenges in Stability Issues for Numerical Differential Equations. Cetraro: Springer, 2011.

[4] Brunton, S. L., Proctor, J. L., Kutz, J. N. *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*. Proceedings of the national academy of sciences, 113(15), 3932-3937, 2016.

[5] Butcher, J. C. Numerical methods for ordinary differential equations. John Wiley & Sons, 2016.

[6] Casas, F., Martínez, V. (Eds.). Advances in Differential Equations and Applications. Springer International Publishing, 2014.

[7] Champagne, J.E. *Les réseaux de neurones multi-couches, le comment et le pourquoi*, Notes et commentaires au sujet des conférences de S. Mallat du Collège de France, 2019.

[8] Chartier, P., Hairer, E., Vilmart, G. *Numerical integrators based on modified differential equations. Mathematics of computation*, 76(260), 1941-1953, 2007.

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

[10] David, M., Méhats, F. *Symplectic learning for Hamiltonian neural networks.* arXiv preprint arXiv:2106.11753, 2021.

[11] De Ryck, T., Lanthaler, S., & Mishra, S. (2021). On the approximation of functions by tanh neural networks. Neural Networks, 143, 732-750.

[12] Gribonval, R., Kutyniok, G., Nielsen, M., Voigtlaender, F. Approximation spaces of deep neural networks. *Constructive approximation*, 55(1), 259-367, 2022.

[13] Haber, E., Ruthotto, L. *Stable architectures for deep neural networks*. Inverse problems, 34(1), 014004, 2017.

[14] Hairer, E., Lubich, C., Wanner, G. *Geometric Numerical integration: structure-preserving algorithms for ordinary differential equations*. Springer, 2006.

[15] Hairer, E., Nørsett, S. P., Wanner, G. *Solving ordinary differential equations. 1, Nonstiff problems.* Springer-Vlg, 1993.

[16] Hairer, E., Vilmart, G. *Preprocessed discrete Moser–Veselov algorithm for the full dynamics of a rigid body*. Journal of Physics A: Mathematical and General, 39(42), 13225, 2006.

[17] Jin, P., Zhang, Z., Kevrekidis, I. G., Karniadakis, G. E. *Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks*. IEEE Transactions on Neural Networks and Learning Systems, 2022.

[18] Jin, P., Zhang, Z., Zhu, A., Tang, Y., Karniadakis, G. E. *SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems*. Neural Networks, 132, 166-179, 2020.

[19] Leimkuhler, B., Reich, S. *Simulating hamiltonian dynamics* (No. 14). Cambridge university press, 2004.

[20] Lu, Y., Zhong, A., Li, Q., Dong, B. *Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations*. In International Conference on Machine Learning (pp. 3276-3285). PMLR, July 2018.

[21] Nguyen, D., Ouala, S., Drumetz, L., Fablet, R. *Em-like learning chaotic dynamics from noisy and partial observations*. arXiv preprint arXiv:1903.10335, 2019.

[22] Offen, C., Ober-Blöbaum, S. *Symplectic integration of learned Hamiltonian systems*. Chaos: An Interdisciplinary Journal of Nonlinear Science, 32(1), 013122, 2022.

[23] Prince, P. J., Dormand, J. R. *High order embedded Runge-Kutta formulae*. Journal of computational and applied mathematics, 7(1), 67-75, 1981.

[24] Raissi, M., Perdikaris, P., Karniadakis, G. E. *Machine learning of linear differential equations using Gaussian processes*. Journal of Computational Physics, 348, 683-693, 2017.

[25] Raissi, M., Perdikaris, P., Karniadakis, G. E. *Multistep neural networks for data-driven discovery of nonlinear dynamical systems*. arXiv preprint arXiv:1801.01236, 2018.

[26] Regazzoni, F., Dede, L., Quarteroni, A. Machine learning for fast and reliable solution of time-dependent differential equations. Journal of Computational physics, 397, 108852, 2019.

[27] Zhu, A., Jin, P., Zhu, B., Tang, Y. On Numerical Integration in Neural Ordinary Differential Equations. arXiv preprint arXiv:2206.07335, 2022.

[28] Zhu, A., Zhu, B., Zhang, J., Tang, Y., Liu, J. *VPNets: Volume-preserving neural networks for learning source-free dynamics*. arXiv preprint arXiv:2204.13843, 2022.

A Proof of Theorem 1

Let us consider, for time step h and for all $n \in [[0, N]]$ (where $h = T/N$), the numerical scheme

$$y_0^* = y_0, \qquad y_{n+1}^* = \Phi_h^{f_{app}(\cdot,h)}\left(y_n^*\right)$$

The consistency error is of the form

$$\begin{aligned}
\varepsilon_n^* &:= y(t_{n+1}) - \Phi_h^{f_{app}(\cdot,h)}\left(y(t_n)\right) \\
&= \underbrace{y(t_{n+1}) - \Phi_h^{\tilde{f}_h}\left(y(t_n)\right)}_{=0 \text{ (Modified field)}} + \Phi_h^{\tilde{f}_h}\left(y(t_n)\right) - \Phi_h^{f_{app}(\cdot,h)}\left(y(t_n)\right) \\
&= \Phi_h^{\tilde{f}_h}\left(y(t_n)\right) - \Phi_h^{f_{app}(\cdot,h)}\left(y(t_n)\right)
\end{aligned}$$

so that, taking (7) and (8) into account, we have

$$\left|\varepsilon_n^*\right| \leqslant C\delta h^{p+1}.$$

Upon using (6), (21) and (21), the local truncation error can then be written as

$$e_{n+1}^* = \Phi_h^{f_{app}(\cdot,h)}(y_n^*) - \Phi_h^{f_{app}(\cdot,h)}(y(t_n)) - \varepsilon_n^*,$$

and from (9), we get

$$\left|e^*_{n+1}\right| \leqslant \left(1+\overline{L}h\right)\left|e^*_n\right| + \left|\varepsilon^*_n\right| \leqslant \left(1+\overline{L}h\right)\left|e^*_n\right| + C\delta h^{p+1}$$

where $\overline{L} := \underset{h\in[h_-,h_+]}{Max} L_{f_{app}(\cdot,h)}$. A discrete Grönwall lemma then leads to

$$\left|e^*_n\right| \leqslant C\delta h^{p+1}\sum_{j=0}^{n-1} e^{\overline{L}(n-j-1)h} \leqslant C\delta h^{p+1}\frac{e^{\overline{L}nh}-1}{e^{\overline{L}h}-1} \leqslant \frac{C\delta h^p}{\overline{L}}\left(e^{\overline{L}T}-1\right).$$

B Choice of the parameters

B.1 Link between learning error and parameters

The influence of the number of parameters over the learning error has been studied. In particular, we have studied the effect of the number of neurons and hidden layers. The dynamical system chosen for the test is the non-linear pendulum while the numerical method is simply the Forward Euler method. In order to approximate the learning error, we compute the value

$$\delta \approx \underset{h\in H^*}{Max}\frac{1}{h}\left\|\tilde{f}_h{}^4(x_{i,j}) - f_{app}(x_{i,j},h)\right\|_{l^\infty} \tag{21}$$

where $(x_{i,j})_{0\leqslant i,j\leqslant 40}$ is a uniform grid on the square $\Omega = [-2,2]^2$, $H^* = (e^{h^*_j})_{0\leqslant j\leqslant 14}$ where $(h^*_j)_{0\leqslant j\leqslant 14}$ is a uniform discretization of $[\log(h^-),\log(h^+)]$ and $\tilde{f}_h{}^4$ corresponds to the field (2) for $k=4$ with R neglected, computed via the formulas given in [8, 14].

As observed in Figure 15, a plateau appears when the number of parameters is large. This plateau has a lower value for a larger number of data. Moreover, we observe that deep networks are more efficient than shallow networks in order to learn the good vector field.

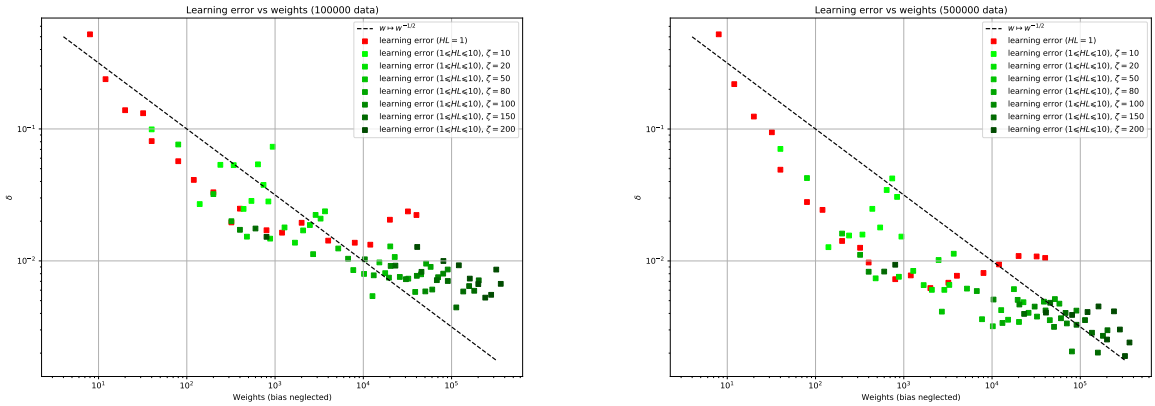

Figure 15: Learning error δ versus number of parameters w in the neural network (number of neurons ζ and hidden layers HL). Shallow network is plot with red points whereas deep network is plot with green points (light green for 10 neurons to dark green for 200 neurons). Bias are neglected. Learning error is plot for 100 000 data (left) and 500 000 data (right). 200 epochs are used. $[h^-, h^+] = [10^{-2}, 10^{-1}]$. The curve of $w \mapsto w^{\frac{1}{2}}$ has been added for comparison purposes.

B.2 Parameters selected in our numerical experiments

For the training, the optimizer *Adam* of Pytorch is used. Besides, the mini-batching option is activated as it appears to be more efficient. Hyperbolic tangent tanh was selected as activation function.

B.2.1 Nonlinear Pendulum - Forward Euler

Parameters	
# Math Parameters:	
Dynamical system:	Pendulum
Numerical method:	Forward Euler
Interval where time steps are selected:	$[h_-, h_+] = [0.1, 2.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.1$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 25\,000\,000$
Proportion of data for training:	80% - $K_0 = 20\,000\,000$
Number of terms in the perturbation (MLP's):	$N_t = 1$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	300
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training: 10 h 14 min 17 s

B.2.2 Rigid body system - Forward Euler

Casimir invariant introduced at the beginning of the section 3 allows to chose the training data in the spherical crown $\left\{x \in [-2, 2]^2 : 0.98 \leqslant |x| \leqslant 1.02\right\}$

Parameters	
# Math Parameters:	
Dynamical system:	Rigid Body
Numerical method:	Forward Euler
Interval where time steps are selected:	$[h_-, h_+] = [0.5, 2.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.5$
# AI Parameters:	
Domain where data are selected:	$\Omega = \left\{x \in [-2, 2]^2 : 0.98 \leqslant \lvert x\rvert \leqslant 1.02\right\}$
Number of data:	$K = 100\,000\,000$
Proportion of data for training:	80% - $K_0 = 80\,000\,000$
Number of terms in the perturbation (MLP's):	$N_t = 1$
Hidden layers per MLP:	2
Neurons on each hidden layer:	250
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	300
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training: 1 Day 21 h 59 min 51 s

B.2.3 Nonlinear Pendulum - Runge-Kutta 2

Parameters	
# Math Parameters:	
Dynamical system:	Pendulum
Numerical method:	RK2
Interval where time steps are selected:	$[h_-, h_+] = [0.1, 2.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.1$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 100\,000\,000$
Proportion of data for training:	80% - $K_0 = 80\,000\,000$
Number of terms in the perturbation (MLP's):	$N_t = 1$
Hidden layers per MLP:	2
Neurons on each hidden layer:	250
Learning rate:	$5 \cdot 10^{-4}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	300
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training: 3 Days 2 h 54 min 35 s

B.2.4 Nonlinear Pendulum - midpoint

Parameters	
# Math Parameters:	
Dynamical system:	Pendulum
Numerical method:	midpoint
Interval where time steps are selected:	$[h_-, h_+] = [0.05, 0.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.25$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^2$
Number of data:	$K = 20\,000\,000$
Proportion of data for training:	80% - $K_0 = 16\,000\,000$
Number of terms in the perturbation (MLP's):	$N_t = 1$
Hidden layers per MLP:	2
Neurons on each hidden layer:	200
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	300
Epochs:	200
Epochs between two prints of loss value:	20

Computational time for training: 9 h 47 min 51 s

B.3 Nonlinear Pendulum - Comparison between traditional and alternative method

For the standard method, data are created by simulating several solutions with the same initial condition for different time steps.

Parameters	
# Math Parameters:	
Dynamical system:	Pendulum
Numerical method:	Forward Euler
Interval where time steps are selected:	$[h_-, h_+] = [0.01, 0.5]$
Time for ODE simulation:	$T = 20$
Time step for ODE simulation:	$h = 0.1$
# AI Parameters:	
Domain where data are selected:	$\Omega = [-2, 2]^2$
Number of data	
- Traditional method:	$(K, N_h) = (50\,000, 5)$
- Alternative method:	$(K, N_h) = (76\,129, 5)$
- Alternative method (parallel training):	$(K, N_h) = (105\,735, 5)$
Proportion of data for training:	80%
- Traditional method:	$(K_0, N_h) = (40\,000, 5)$
- Alternative method:	$(K_0, N_h) = (60\,903, 5)$
- Alternative method (parallel training):	$(K_0, N_h) = (84\,588, 5)$
Number of terms in the perturbation (MLP's):	$N_t = 3$
Hidden layers per MLP:	2
Neurons on each hidden layer:	50
Learning rate:	$2 \cdot 10^{-3}$
Weight decay:	$1 \cdot 10^{-9}$
Batch size (mini-batching for training):	100
Epochs:	200
Epochs between two prints of loss value:	20

For the alternative method with parallel training, the total computational time for training correspond to the maximum of all training times for each term of the modified field.

Computational time for training			
Method	Traditional	Alternative	Alternative (parallel training)
f_1		2 min 54 s	3 min 24 s
f_2		2 min 22 s	3 min 28 s
R		13 min 23 s	17 min 9 s
Total	**18 min 33 s**	**18 min 41 s**	**17 min 9 s**