Hamiltonian reduction using a convolutional auto-encoder coupled to an Hamiltonian neural network

Raphaël Côte, Emmanuel Franck, Laurent Navoret, Guillaume Steimer, Vincent Vigon

Hamiltonian reduction using a convolutional auto-encoder coupled to a Hamiltonian neural network

Raphaël Côte[1], Emmanuel Franck[1, 2], Laurent Navoret[1, 2], Guillaume Steimer[1, 2], and Vincent Vigon[1, 2]

[1]Institut de Recherche Mathématique Avancée, UMR 7501 Université de Strasbourg et CNRS, 7 rue René Descartes 67000 Strasbourg, France
[2]INRIA, MACARON Project, Strasbourg, France

Abstract

The reduction of Hamiltonian systems aims to build smaller reduced models, valid over a certain range of time and parameters, in order to reduce computing time. By maintaining the Hamiltonian structure in the reduced model, certain long-term stability properties can be preserved. In this paper, we propose a non-linear reduction method for models coming from the spatial discretization of partial differential equations: it is based on convolutional auto-encoders and Hamiltonian neural networks. Their training is coupled in order to simultaneously learn the encoder-decoder operators and the reduced dynamics. Several test cases on non-linear wave dynamics show that the method has better reduction properties than standard linear Hamiltonian reduction methods.

Keywords: Hamiltonian dynamics, model order reduction, convolutional auto-encoder, Hamiltonian neural network, non-linear wave equations, shallow water equation

AMS subject classifications: 65P10, 34C20, 68T07

1 Introduction

Hamiltonian reduced order modeling techniques have been successfully developed in order to perform accelerated numerical simulations of some parameterized Hamiltonian models of large dimension [25, 16, 30]. The spatial discretization of some wave-like partial differential equations gives rise to very large such Hamiltonian systems. Reduced order models can be essential for real-time simulations or when a large number of simulation instances are required as part of a control, optimisation or uncertainty quantification algorithm. Starting from the initial model, a large differential system, the methods consist into constructing a differential system of a smaller size that can produce valid approximate solutions for a predefined range of times and parameters. Many physical models have a Hamiltonian structure and this gives the system a certain number of geometrical properties like the conservation of energy and the symplecticity of the phase space flows. In particular, the preservation of this structure at the discrete level enables to ensure large-time stability of the numerical simulations [14]. In order to build consistent and robust reduced models, it is therefore interesting to preserve this Hamiltonian structure through the reduction. The construction of reduced models can be divided in two steps: (i) find a so-called pair of encoder and decoder operators that goes from the full to the reduced variables and inversely; (ii) identify the dynamics followed by the reduced variables. The construction of the encoder and decoder operators relies on a large number of data produced by numerical simulations in the range of time and parameters of interest.

The first approach to reduce a large Hamiltonian system relies on a linear approximation: the solutions manifold is approximated with a symplectic vector space of small dimension [25]. The encoder is here a linear mapping, which is also constructed to be symplectic so that the Hamiltonian structure is preserved into the reduced model. Such symplectic mapping can be constructed from data through greedy algorithms [1] or through a Singular Value Decomposition (SVD) methodology: this is the Proper Symplectic Decomposition (PSD) proposed in [25]. In this work, several algorithms have been proposed to define approximated optimal symplectic mappings: for instance, the cotangent-lift algorithm devise a symplectic mapping which is also orthogonal and have a block diagonal structure. Then the reduced model is obtained using the Galerkin projection method: the model is constructed by supposing that a symplectic projection of the residual vanishes, where the residual stands for the error obtained after replacing the original variables by the decoded reduced variables.

Such linear reductions, however, can hardly handle non-linear dynamics: this is the case for convection-dominated or non-linear wave like problems for which the solution manifold is badly approximated by hyperplanes. In order to build more expressive reduced models, one possibility is to consider time adaptive reduced methods [22, 15]. Another widely investigated possibility is to consider non-linear reduction methods.

Regarding the construction of non-linear encoder-decoder operators, a first class of methods rely on manifold learning techniques [27, 4]. Such methods are based on the geometrical analysis of the neighbors graph of the data thanks to the computation of geodesic distances (ISOMAP method, [28]), of eigenfunctions of the graph Laplacian (EigenMaps method [2]) or of diffusion processes (DiffusionMaps method [8]). This provides reduced variables for each data that can be further interpolated using the Nyström formula [3].

Since the explosion of deep learning in the early 2010s, new dimension reduction methods grounded on neural networks have been developed. The convolutional auto-encoder architecture [12] seems particularly appropriate since its very purpose is to determine latent variables: the neural network is indeed divided into an encoder part and decoder part and

they are trained simultaneously so that the sequence of encoder and decoder is close to the identity map. This was originally developed for image generation, but has been also used for reduced order modeling for models coming from the spatial discretization of partial differential equations on a grid [19, 17, 26]. Indeed, convolutional neural networks have proved particularly effective to extract multi-scale informations of grid-structured data. Secondly, they involve far less parameters than their dense counterparts, especially when the input size of the neural network is large. In [5], the authors use auto-encoder neural networks for Hamiltonian reduction: the encoder and the decoder are weakly constrained to be symplectic thanks to a penalization term in the cost functional.

Once non-linear encoders and decoders have been devised from data, the dynamics of the reduced variables still has to be determined. Two strategies can be considered. The first one relies on a Galerkin projection of the Hamiltonian system as in the linear case [5]. Note that the reduced model is indeed Hamiltonian provided the decoder is a symplectic map. However, the reduced model still requires the evaluation of the vector field in the original large dimension space of size $2N$: this is a well-known difficulty in non-linear reduction. To overcome this difficulty, hyper-reduction methods have been proposed like the discrete empirical interpolation method (DEIM) [6]. This method has to be adapted to not destroy the geometric structure of the full order model as in [15, 23, 24] where the authors propose to apply a DEIM algorithm to the gradient of the Hamiltonian thus, with some additional strategies, preserve the geometric structure of the full order model.

Another approach is to learn about the dynamics of the reduced variables using a neural network: given the initial state, the neural network provides the full trajectory. As the learning is done directly in the reduced dimension, the obtained reduced model does not require an evaluation of non-linear terms in the original variables: this is a clear advantage of the method compared with projection-based ones. The reduced dynamics can be captured for instance by Recurrent Neural Networks (RNNs), Long Short Term Memory (LSTM) neural networks [21], or by fully connected networks [11]. This has also been considered as correction of the Galerkin-type reduced models [32].

Here we consider another strategy which consists in learning the vector field that generates the observed reduced dynamics. The neural network is trained so as to minimize its deviation from the finite difference time derivative of the reduced data obtained after encoding. As we aim at conserving the Hamiltonian structure at the reduced level, the vector field is further supposed to be associated with a reduced Hamiltonian function. Therefore, we can learn directly the Hamiltonian function instead of the vector field. This is a so-called Hamiltonian Neural Network (HNN) strategy proposed in [13] where a symplectic time integrator is used. We also note that neural networks methods has also recently be used to learn hidden or reduced dynamics which also involve dissipation [34, 35, 7, 10].

The present paper proposes to combine an auto-encoder strategy for the encoding-decoding part and a HNN method to learn the reduced dynamics: this will be referred to as the AE-HNN method. Note that there is a priori no reason for the auto-encoder neural networks to spontaneously provide reduced variables compatible with Hamiltonian dynamics. Therefore, some constraints on the auto-encoder have to be added. This can be done by imposing symplecticity weakly as in [5], where a penalization term of the symplectic constraint is added to the loss function. Here, we propose instead to train it simultaneously with the HNN. With this joint training, the auto-encoder will gradually converge to a set of reduced variables compatible with a Hamiltonian system. Of course, this means that the loss functions associated with each neural network must be weighted judiciously during training. Note that such a joint training of the encoding-decoding operators and the reduce

dynamics have been explored in [11, 34], but without considering Hamiltonian structures for the first and without symplectic time integrator for the second.

The outline of the article is as follows. In Section 2, we introduce parameterized Hamiltonian systems as well as the main steps for the construction of reduced order models. Section 3 then presents the non-linear AE-HNN reduction method. In particular we describe the architectures and the loss functions used for the trainings. Finally, Section 4 is devoted to the numerical results: we apply our reduction method on the Hamiltonian systems obtained after spatial discretization of linear, non-linear wave equations and a shallow water system and compare it with the linear PSD reduction technique.

2 Parameterized Hamiltonian systems and reduction

In this section, we introduce the notations used for the parameterized Hamiltonian systems and the main steps for the construction of a reduced model.

In the following, we often write vectors of interest with bold script letters, operators with capital italic letters, with their parameters as indices, and overline quantities when related to the reduced model.

2.1 Parameterized Hamiltonian dynamics

We consider a parameterized autonomous Hamiltonian system, whose solution, $\mathbf{y}(t;\mu) \in \mathbb{R}^{2N}$ with $N \in \mathbb{N}^*$, depends on time $t \in [0,T]$, with $T > 0$, and on a parameter $\mu \in \Xi \subset \mathbb{R}^d$, with $d \in \mathbb{N}$. The dynamics derive from a given Hamiltonian function $\mathcal{H} : \mathbb{R}^{2N} \times \Xi \to \mathbb{R}$ and writes

$$\begin{cases} \dfrac{d}{dt}\mathbf{y}(t;\mu) = J_{2N}\,\nabla_{\mathbf{y}}\mathcal{H}\left(\mathbf{y}(t;\mu);\mu\right), & \text{in } (0,T], \\ \mathbf{y}(0;\mu) = \mathbf{y}_{\text{init}}(\mu), \end{cases} \tag{1}$$

where $\mathbf{y}_{\text{init}}(\mu) \in \mathbb{R}^{2N}$ is a given initial condition and J_{2N} refers to the canonical symplectic matrix

$$J_{2N} = \begin{pmatrix} 0_N & I_N \\ -I_N & 0_N \end{pmatrix},$$

with I_N the identity matrix of dimension N.

Introducing the canonical coordinates $\mathbf{y} = (\mathbf{q},\mathbf{p})^T$, the system becomes:

$$\begin{cases} \dfrac{d}{dt}\mathbf{q}(t;\mu) = \nabla_{\mathbf{p}}\mathcal{H}\left(\mathbf{q},\mathbf{p};\mu\right), & \text{in } (0,T], \\ \dfrac{d}{dt}\mathbf{p}(t;\mu) = -\nabla_{\mathbf{q}}\mathcal{H}\left(\mathbf{q},\mathbf{p};\mu\right), & \text{in } (0,T], \\ \mathbf{q}(0;\mu) = \mathbf{q}_{\text{init}}(\mu), \\ \mathbf{p}(0;\mu) = \mathbf{p}_{\text{init}}(\mu), \end{cases} \tag{2}$$

with $\mathbf{q}_{\text{init}}, \mathbf{p}_{\text{init}} \in \mathbb{R}^N$ such that $\mathbf{y}_{\text{init}} = (\mathbf{q}_{\text{init}}, \mathbf{p}_{\text{init}})^T$. A key property of such systems is that the associated flow is symplectic, meaning that $\phi_t\left(\mathbf{y}_{\text{init}}(\mu);\mu\right) = \mathbf{y}(t;\mu)$ satisfies the relation

$$\left(\nabla_{\mathbf{y}}\phi_t\left(\mathbf{y}_{\text{init}}(\mu);\mu\right)\right)^T J_{2N}\left(\nabla_{\mathbf{y}}\phi_t\left(\mathbf{y}_{\text{init}}(\mu);\mu\right)\right) = J_{2N}.$$

One consequence is that the Hamiltonian $\mathcal{H}$ is preserved along the flow

$$\forall t \in (0,T], \mu \in \Xi, \quad \mathcal{H}\left(\mathbf{y}(t;\mu);\mu\right) = \mathcal{H}\left(\mathbf{y}_{\text{init}}(\mu);\mu\right),$$

which is of particular importance when considering physical systems.

In this work, we are specifically interested in Hamiltonian systems resulting from the space discretization of one-dimensional and two-dimensional wave-type equations. In such systems, $\mathbf{q} \in \mathbb{R}^N$ refers to the height of the wave at grid points and $\mathbf{p} \in \mathbb{R}^N$ to the velocity of the wave also at grid points. Examples will be detailed in the numerical section.

In order to provide numerical approximations of the solution, specific numerical schemes have been developed to ensure the symplectic property at the discrete level [14]. These schemes also guarantee large time stability of the numerical solutions. Here we consider the standard second-order Störmer-Verlet scheme. Denoting $\mathbf{y}_\mu^n = \left(\mathbf{q}_\mu^n, \mathbf{p}_\mu^n\right)^T \in \mathbb{R}^{2N}$ the approximate solution at time $t^n = n\Delta t$, with time step $\Delta t > 0$, one iteration of the scheme is defined by:

$$\begin{cases} \mathbf{p}_\mu^{n+\frac{1}{2}} = \mathbf{p}_\mu^n - \frac{1}{2}\Delta t\, \nabla_{\mathbf{q}} \mathcal{H}\left(\mathbf{q}_\mu^n, \mathbf{p}_\mu^{n+\frac{1}{2}}; \mu\right), \\ \mathbf{q}_\mu^{n+1} = \mathbf{q}_\mu^{n+\frac{1}{2}} + \Delta t \left[\nabla_{\mathbf{p}} \mathcal{H}\left(\mathbf{q}_\mu^n, \mathbf{p}_\mu^{n+\frac{1}{2}}; \mu\right) + \nabla_{\mathbf{p}} \mathcal{H}\left(\mathbf{q}_\mu^{n+1}, \mathbf{p}_\mu^{n+\frac{1}{2}}; \mu\right)\right], \\ \mathbf{p}_\mu^{n+1} = \mathbf{p}_\mu^{n+\frac{1}{2}} - \frac{1}{2}\Delta t\, \nabla_{\mathbf{q}} \mathcal{H}\left(\mathbf{q}_\mu^{n+1}, \mathbf{p}_\mu^{n+\frac{1}{2}}; \mu\right). \end{cases} \tag{3}$$

Under the further assumption that the Hamiltonian $\mathcal{H}$ is separable, i.e. is the sum of a function depending only $\mathbf{q}$ and another depending only $\mathbf{p}$:

$$\mathcal{H}(\mathbf{y}; \mu) = \mathcal{H}^1(\mathbf{q}; \mu) + \mathcal{H}^2(\mathbf{p}; \mu),$$

the implicit first two steps of (3) become explicit and the scheme simplifies into:

$$\begin{cases} \mathbf{p}_\mu^{n+\frac{1}{2}} = \mathbf{p}_\mu^n - \frac{1}{2}\Delta t\, \nabla_{\mathbf{q}} \mathcal{H}^1\left(\mathbf{q}_\mu^n; \mu\right), \\ \mathbf{q}_\mu^{n+1} = \mathbf{q}_\mu^n + \Delta t\, \nabla_{\mathbf{p}} \mathcal{H}^2\left(\mathbf{p}_\mu^{n+\frac{1}{2}}; \mu\right), \\ \mathbf{p}_\mu^{n+1} = \mathbf{p}_\mu^{n+\frac{1}{2}} - \frac{1}{2}\Delta t\, \nabla_{\mathbf{q}} \mathcal{H}^1\left(\mathbf{q}_\mu^{n+1}; \mu\right). \end{cases} \tag{4}$$

2.2 Hamiltonian reduced order modeling

Solving Hamiltonian systems with large dimension $2N \gg 1$ numerically can be relatively costly, and this is especially true when we want to solve a large number of them for a parametric study, for example. Therefore, methods have been developed in order to construct reduced Hamiltonian systems of smaller size $2K \ll 2N$, which capture the main dynamics for a range of times t and reduction parameters μ.

We first have to define an appropriate change of variable. To do that, we search for a $2K$-dimensional manifold $\widehat{\mathcal{M}}$ that approximates well the manifold

$$\mathcal{M} = \{\mathbf{y}(t; \mu) \text{ with } t \in [0, T], \mu \in \Xi\} \subset \mathbb{R}^{2N}$$

formed by the values taken by the solutions of the differential equation (1). The manifold structure results from the Cauchy-Lipshitz (Picard-Lindhöf) theorem with parameters under

some regularity assumptions of the Hamiltonian. The manifold $\widehat{\mathcal{M}}$ is defined thanks to a so-called decoding operator $\mathcal{D}_{\theta_d} : \mathbb{R}^{2K} \to \mathbb{R}^{2N}$:

$$\widehat{\mathcal{M}} = \left\{ \mathcal{D}_{\theta_d}(\overline{\mathbf{y}}) \text{ with } \overline{\mathbf{y}} \in \mathbb{R}^{2K} \right\} \subset \mathbb{R}^{2N}.$$

We also consider a pseudo-inverse operator $\mathcal{E}_{\theta_e} : \mathbb{R}^{2N} \to \mathbb{R}^{2K}$, called the encoder, which satisfies the relation

$$\mathcal{E}_{\theta_e} \circ \mathcal{D}_{\theta_d} = \mathrm{Id}_{\mathbb{R}^{2K}}.$$

To determine $\mathcal{D}_{\theta_d}$ and $\mathcal{E}_{\theta_e}$, we therefore ask for the projection operator $\mathcal{D}_{\theta_d} \circ \mathcal{E}_{\theta_e}$ onto $\widehat{\mathcal{M}}$ to be close to the identity on a data set $U \subset \mathcal{M}$:

$$\forall u \in U, \quad \mathcal{D}_{\theta_d} \circ \mathcal{E}_{\theta_e}(u) \approx u. \tag{5}$$

The data set U is composed of snapshots of the solutions at different times and various parameters, obtained with the symplectic algorithm defined above in (3); it writes

$$U = \left\{ \mathbf{y}^0_{\mu_1}, \dots, \mathbf{y}^M_{\mu_1}, \dots, \mathbf{y}^0_{\mu_P}, \dots, \mathbf{y}^M_{\mu_P} \right\},$$

where $M \in \mathbb{N}^*$ is the number of time-step chosen and $P \in \mathbb{N}^*$ the number of sampled parameters.

In addition to these approximation properties, we also ask for the reduced variables,

$$\overline{\mathbf{y}}(t;\mu) = \mathcal{E}_{\theta_e}(\mathbf{y}(t;\mu)) \in \mathbb{R}^{2K},$$

to follow a reduced Hamiltonian dynamics:

$$\begin{cases} \dfrac{d}{dt}\overline{\mathbf{y}}(t;\mu) = J_{2K} \nabla_{\overline{\mathbf{y}}} \overline{\mathcal{H}}_{\theta_h}(\overline{\mathbf{y}}(t;\mu);\mu), & \text{in } (0,T], \\ \overline{\mathbf{y}}(0;\mu) = \mathcal{E}_{\theta_e}(\mathbf{y}_{\text{init}}(\mu)), \end{cases} \tag{6}$$

where $\overline{\mathcal{H}}_{\theta_h} : \mathbb{R}^{2K} \times \Xi \to \mathbb{R}$ is a reduced Hamiltonian to be built.

The most common approach for Hamiltonian reduced order modeling is called the Proper Symplectic Decomposition (PSD) [25]. This method is briefly described in Appendix A. Although efficient for linear dynamics, it fails into reducing non-linear ones. This is why several non-linear Hamiltonian reduction techniques have been developed [5, 15]. In the next section, we present a strategy based on the coupling of an Auto-Encoder (AE) and a Hamiltonian Neural Network (HNN) method.

3 A non-linear Hamiltonian reduction method

The method proposed in this work consists in constructing the Hamiltonian reduced model via neural networks. More precisely, we aim at defining the following three neural networks:

- a decoder $\mathcal{D}_{\theta_d} : \mathbb{R}^{2K} \to \mathbb{R}^{2N}$,
- an encoder $\mathcal{E}_{\theta_e} : \mathbb{R}^{2N} \to \mathbb{R}^{2K}$,
- a reduced Hamiltonian $\overline{\mathcal{H}}_{\theta_h} : \mathbb{R}^{2K} \times \Xi \to \mathbb{R}$,

where $(\theta_d, \theta_e, \theta_h)$ stands for their parameters, such that the resulting reduced dynamics provides a good approximation of the initial one. An auto-encoder strategy will be used to define $\mathcal{D}_{\theta_d}$ and $\mathcal{E}_{\theta_e}$ while a Hamiltonian Neural Network will be considered for $\overline{\mathcal{H}}_{\theta_h}$. Note that the encoder and decoder are not enforced to be symplectic but the reduced model is.

Figure 1 illustrates how the reduced model is expected to be used for prediction. The initial condition $\mathbf{y}(t=0;\mu)$ is converted by the encoder $\mathcal{E}_{\theta_e}$ to the reduced initial condition $\overline{\mathbf{y}}(t=0;\mu)$. Then several iterations of the Störmer-Verlet scheme with the reduced Hamiltonian $\overline{\mathcal{H}}_{\theta_h}$ are performed to obtain an approximated reduced solution $\overline{\mathbf{y}}(t=T;\mu)$ at time T. Finally, by using the decoder $\mathcal{D}_{\theta_d}$, the latter is transformed into $\widehat{\mathbf{y}}(t=T;\mu) \approx \mathbf{y}(t=T;\mu)$. Note parameter μ has to be supplied to the Hamiltonian function.

In order to determine the appropriate parameters of the three neural networks $\mathcal{D}_{\theta_d}$, $\mathcal{E}_{\theta_e}$ and $\overline{\mathcal{H}}_{\theta_h}$, we have to define both their architectures and the loss functions used for their training. This section focuses on the latter.

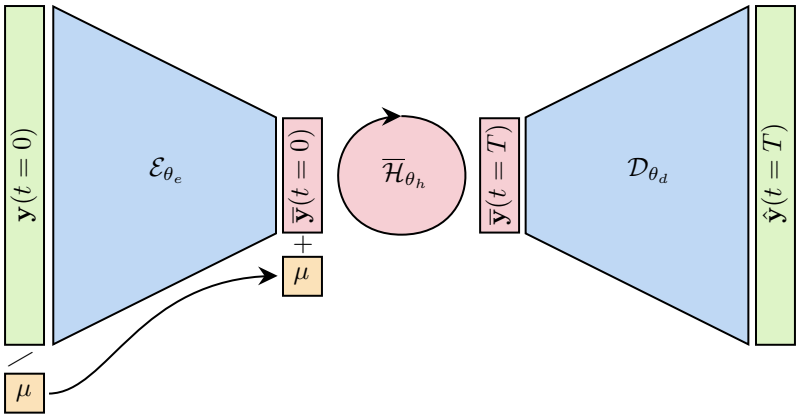


Figure 1: Prediction using the reduced model. The closed loop in the middle refers to the application of several iterations of the Störmer-Verlet scheme.

3.1 Reduction with an Auto Encoder (AE)

An auto-encoder (AE) is a classical architecture of neural networks to find a reduced representation of data [12]. It is composed of two neural networks, $\mathcal{D}_{\theta_d}$ and $\mathcal{E}_{\theta_e}$, which are trained together such as to make the projection operator $\mathcal{D}_{\theta_d} \circ \mathcal{E}_{\theta_e}$ the closest to the identity map on the training data set U. Therefore, the AE is trained so as to minimize the following loss

$$\mathcal{L}_{\mathrm{AE}}(\theta_e, \theta_d) = \sum_{\mathbf{y} \in U} \left\| \mathbf{y} - \mathcal{D}_{\theta_d}\left(\mathcal{E}_{\theta_e}\left(\mathbf{y}\right)\right) \right\|_2^2 . \tag{7}$$

To account for the particular structure of $\mathbf{y}$ made of coordinates and momenta, the encoder input is a tensor of size $(N, 2)$. This AE will be referred to as the bichannel AE.

Another choice would be to define two separate auto-encoders for coordinates and momenta: the coordinates AE is denoted $(\mathcal{E}^1_{\theta_e,1}, \mathcal{D}^1_{\theta_d,1})$ and the momenta AE is denoted $(\mathcal{E}^2_{\theta_e,2}, \mathcal{D}^2_{\theta_d,2})$; each encoder input has shape $(N, 1)$. The AEs are trained by minimizing the loss:

$$\mathcal{L}_{\mathrm{split,AE}}(\theta_e, \theta_d) = \sum_{(\mathbf{q},\mathbf{p}) \in U} \left\| \mathbf{q} - \mathcal{D}^1_{\theta_d,1}\left(\mathcal{E}^1_{\theta_e,1}\left(\mathbf{q}\right)\right) \right\|_2^2 + \left\| \mathbf{p} - \mathcal{D}^2_{\theta_d,2}\left(\mathcal{E}^2_{\theta_e,2}\left(\mathbf{p}\right)\right) \right\|_2^2 .$$

These AEs will be called the split AE. This split AE will be used to preserve the separability property of the Hamiltonian, where applicable (see Remark 1 below).

The architectures of the neural networks are chosen specifically to the Hamiltonian systems in consideration. In this work, we focus on systems resulting from the spatial discretization of wave-like equations: networks will be more efficient if they take into account the spatial structure of the data. Consequently, the encoder $\mathcal{E}_{\theta_e}$ is first composed of several pairs of convolution layer and down-sampling before ending with some dense layers, as depicted in Figure 2. As usual for AE networks, the decoder is constructed in a mirror way, i.e. starting with some dense layers and then ending with pairs of up-sampling and convolution layers in reversed size order.

More precisely, the encoder takes as input a vector $\mathbf{y} = (\mathbf{q}, \mathbf{p})$ and starts with a succession of so-called encoder blocks, made of a stride 1 convolution with kernel size 3 and a down-sampling step (stride 2 convolution with kernel size 2). An encoder block results in an output that has twice as many channels and half as many rows as the input. After possibly composing several encoder blocks, we add a last convolution layer with kernel size 3 and then a flattening operation by concatenating every channels. Then, dense layers are added until reaching the desired reduced dimension $2K$ of $\overline{\mathbf{y}}$. As already said, the decoder is built as a mirror: the flattening operation is replaced by unflattening and the encoder blocks by the decoder blocks made of an up-sampling layer of size 2 smoothed out with a convolution with kernel size 2 and a convolution layer with kernel size 3, which symmetrically results in output that has half as many channels and twice as many rows as the input. The architecture of the auto-encoder is thus defined with the number of encoder and decoder blocks and the dense layer sizes for both encoder and decoder. Figure 2 illustrates an example of auto-encoder architecture with one block for the encoder and one block for the decoder.

This AE architecture can easily be extended to 2D systems using two-dimensional convolutional layers and up and down-sampling with appropriate dimensions.

3.2 Reduced model with a Hamiltonian Neural Network (HNN)

The AE constructed in the previous section enables us to define the reduced trajectories:

$$\overline{\mathbf{y}}(t;\mu) = \mathcal{E}_{\theta_e}(\mathbf{y}(t;\mu)). \tag{8}$$

To obtain the dynamics of these reduced variables, we propose to use a Hamiltonian Neural Network strategy [13]. We thus look for a neural network function $\overline{\mathcal{H}}_{\theta_h}$, parameterized by θ_h, such that:

$$\frac{d}{dt}\overline{\mathbf{y}}(t;\mu) = J_{2K}\nabla_{\overline{\mathbf{y}}}\overline{\mathcal{H}}_{\theta_h}(\overline{\mathbf{y}}(t;\mu);\mu).$$

Note that the Hamiltonian is supposed to depend on parameter μ. We remind that $\mu \in \Xi$ stands for known parameters of the model, unlike θ_h that is the neural networks parameters to be learnt. The architecture of the reduced Hamiltonian is a classical MLP neural network. The size of the neural network is chosen to be small so that the reduced model remains competitive. This reduced dynamics are in practice defined through a time discretization. We therefore introduce the prediction operator:

$$\mathcal{P}_s\left(\overline{\mathbf{y}}; \overline{\mathcal{H}}_{\theta_h,\mu}\right),$$

which consists in performing $s \in \mathbb{N}^*$ iterations of the Störmer-Verlet scheme, defined in (3), starting from $\overline{\mathbf{y}}$ and where $\overline{\mathcal{H}}_{\theta_h,\mu}$ stands for the Hamiltonian function $\overline{\mathcal{H}}_{\theta_h}(.;\mu)$. The

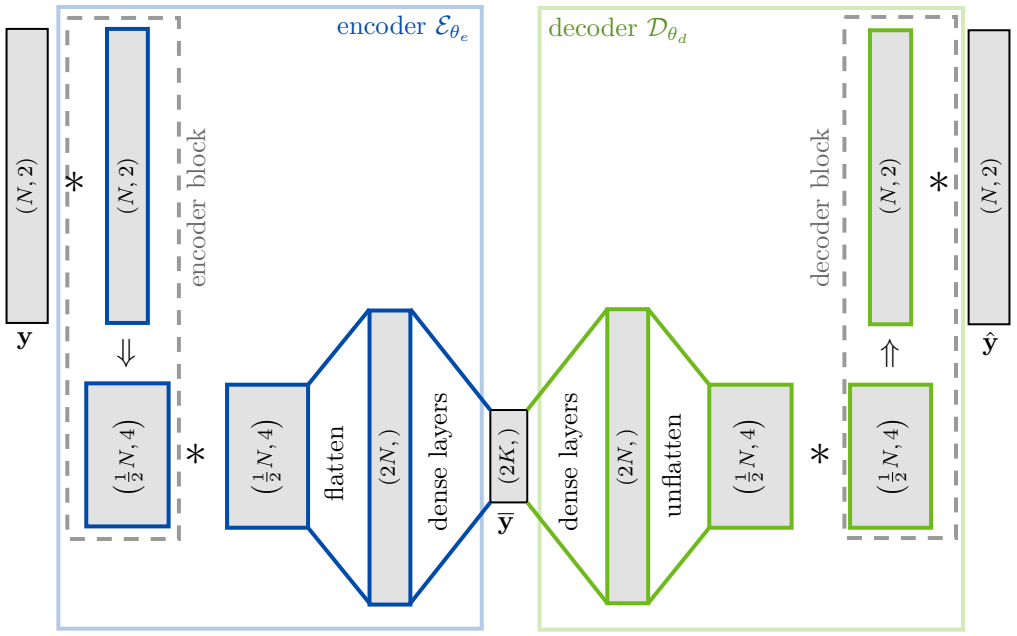


Figure 2: Auto-encoder architecture: encoder in blue, decoder in green. Symbols. $*$: stride 1 convolution with periodic padding, $\Downarrow$ down-sampling (stride 2 convolution), $\Uparrow$: up-sampling (repeat once each value along the last axis then smooth it with a kernel size 2 convolution).

number of steps s considered in this prediction is called the watch duration. This is a hyper-parameter of the method that has to be set. The parameters of the reduced Hamiltonian are finally obtained by minimizing the following loss:

$$\mathcal{L}^s_{\overline{\text{pred}}}(\theta_e, \theta_h) = \sum_{\mathbf{y}^n_\mu, \mathbf{y}^{n+s}_\mu \in U} \left\| \overline{\mathbf{y}}^{n+s}_\mu - \mathcal{P}_s\left(\overline{\mathbf{y}}^n_\mu; \overline{\mathcal{H}}_{\theta_h,\mu}\right) \right\|_2^2 \tag{9}$$

where $\mathbf{y}^n_\mu, \mathbf{y}^{n+s}_\mu \in U$ denote the sampling of random pairs $(\mathbf{y}^n_\mu, \mathbf{y}^{n+s}_\mu)$ on the data set U. In other words, random time series of size s are sampled and only the data at both ends are considered. This loss thus compares the reduced trajectories (8) with the ones obtained with the reduced Hamiltonian. The name of the loss function, "p$\overline{\text{re}}$d", refers to the prediction in the reduced variables. Note that the encoder neural network is required to obtain the reduced data $\overline{\mathbf{y}} = \mathcal{E}_{\theta_e}(\mathbf{y})$ and this is why the loss also depends on θ_e. This kind of loss function, based on a model, has been widely used in physics based deep learning methods [29].

Then, we constrain the reduced trajectories to preserve the reduced Hamiltonian with the following loss function :

$$\mathcal{L}^s_{\overline{\text{stab}}}(\theta_e, \theta_h) = \sum_{\mathbf{y}^n_\mu, \mathbf{y}^{n+s}_\mu \in U} \left\| \overline{\mathcal{H}}_{\theta_h,\mu}\left(\overline{\mathbf{y}}^{n+s}_\mu\right) - \overline{\mathcal{H}}_{\theta_h,\mu}\left(\overline{\mathbf{y}}^n_\mu\right) \right\|_2^2, \tag{10}$$

where $\overline{\mathbf{y}}^{n+s}_\mu$ and $\overline{\mathbf{y}}^n_\mu$ are still obtained using the encoder $\mathcal{E}_{\theta_e}$. The aim is to ensure some stability of the reduced model, hence its name "st$\overline{\text{ab}}$" . At first sight, this loss seems redundant with the prediction-reduced loss since using Störmer-Verlet schemes in the prediction

step ensures that the reduced Hamiltonian is preserved at least approximately. Hence if the prediction-reduced loss becomes small, so does the stability loss. However, this additional loss may help the coupling to converge.

Remark 1. *The separability of the Hamiltonian could be an interesting property to preserve at the reduced level. Using a split AE to learn separately reduced coordinates and momenta, a separable reduced Hamiltonian can be designed:*

$$\overline{\mathcal{H}}_{\theta_h}(\overline{\mathbf{y}};\mu) = \overline{\mathcal{H}}^1_{\theta_h}(\bar{\mathbf{q}};\mu) + \overline{\mathcal{H}}^2_{\theta_h}(\bar{\mathbf{p}};\mu),$$

involving two neural networks. This will be referred to as the split HNN. The Störmer-Verlet scheme would then have a cost of an explicit scheme. Note however that this is not a crucial gain since the reduced models under consideration have small sizes.

Remark 2. *The HNN will produce an approximation of the Hamiltonian, whose associated flow would have generated the discrete solution we would like to fit. Note also that the data used to fit the HNN was obtained after encoding a discrete solution of the initial dynamics of the Hamiltonian. Thus, even this discrete dynamics is an approximation to the encoded continuous Hamiltonian flow. However, from a practical point of view, it is not essential to capture the underlying continuous dynamics (at the reduced level): the objective is rather to obtain a method capable of reproducing the discrete dynamics, with good geometric properties, for a given time step. So, in practice, we actually do not vary the time step.*

3.3 Strong coupling of the neural networks

The prediction-reduced and the stability-reduced losses (9) already introduce a coupling between the encoder neural networks and the reduced Hamiltonian one. To make the coupling stronger, we could ask for the trajectories in the initial variables to be well recovered. This is why we introduce the following fourth loss function named "pred" which refers to the model prediction in the FOM space:

$$\mathcal{L}^s_{\text{pred}}(\theta_e,\theta_d,\theta_h) = \sum_{\mathbf{y}^n_\mu,\mathbf{y}^{n+s}_\mu \in U} \left\| \mathbf{y}^{n+s}_\mu - \mathcal{D}_{\theta_d}\left(\mathcal{P}_s\left(\overline{\mathbf{y}}^n_\mu; \overline{\mathcal{H}}_{\theta_h,\mu}\right)\right)\right\|_2^2. \tag{11}$$

This is the only loss function that couples the three neural networks. It compares the trajectories in the initial variables with the full process of encoding, predicting in reduced variables over s iterations and then decoding.

To sum up, we use four different loss functions $\mathcal{L}_{\text{AE}}, \mathcal{L}^s_{\overline{\text{pred}}}, \mathcal{L}_{\overline{\text{stab}}}$ and $\mathcal{L}^s_{\text{pred}}$, given by (7)-(9)-(10)-(11) that couple both AE and HNN neural networks. The training aims to find the parameters $(\theta_e,\theta_d,\theta_h)$ that are a solution to the minimization problem:

$$\min_{\theta_e,\theta_d,\theta_h} \quad \omega_{\text{AE}}\,\mathcal{L}_{\text{AE}}(\theta_e,\theta_d) + \omega_{\overline{\text{pred}}}\,\mathcal{L}^s_{\overline{\text{pred}}}(\theta_e,\theta_h) + \omega_{\overline{\text{stab}}}\,\mathcal{L}^s_{\overline{\text{stab}}}(\theta_e,\theta_h) + \omega_{\text{pred}}\,\mathcal{L}^s_{\text{pred}}(\theta_e,\theta_d,\theta_h),$$

where ω_{AE}, $\omega_{\overline{\text{pred}}}$, $\omega_{\overline{\text{stab}}}$, ω_{pred} are positive weights: these are hyper-parameters of the method. The four loss functions interact during training and possibly compete with each other. Note that in the end, the only loss value that really quantifies the quality of the reduction and prediction process is the one corresponding to $\mathcal{L}^s_{\text{pred}}$. The other loss functions are only useful in the training process.

3.4 Training hyper-parameters

In addition to the parameters of the neural networks (number of layers, size of the layers), the training of the model also depends on several hyper-parameters.

Reduced dimension K. In classical reduction method, the larger K, the more accurate the reduced model. Regarding the AE-HNN method, as the approximation is truly non-linear, there may be no benefit increasing the reduced dimension. In practice, the minimum possible reduced dimension should be equal to the number of variable parameters in the model.

Watch duration in predictions. One of the hyper-parameter to set is the watch duration s in the loss functions $\mathcal{L}^s_{\overline{\text{stab}}}$, $\mathcal{L}^s_{\overline{\text{pred}}}$ and $\mathcal{L}^s_{\text{pred}}$ that make predictions. This quantity should be not too small to capture the dynamics but also not too large as the computation of the gradients of the associated loss functions may generate vanishing gradient problems. In the numerical setting, the watch duration will be typically set to $s = 16$.

Loss functions weights. Losses weights have been chosen experimentally as follows:

$$\omega_{\text{pred}} = 0.1, \quad \omega_{\text{AE}} = 0.1, \quad \omega_{\overline{\text{pred}}} = 80, \quad \omega_{\overline{\text{stab}}} = 7 \times 10^{-4}.$$

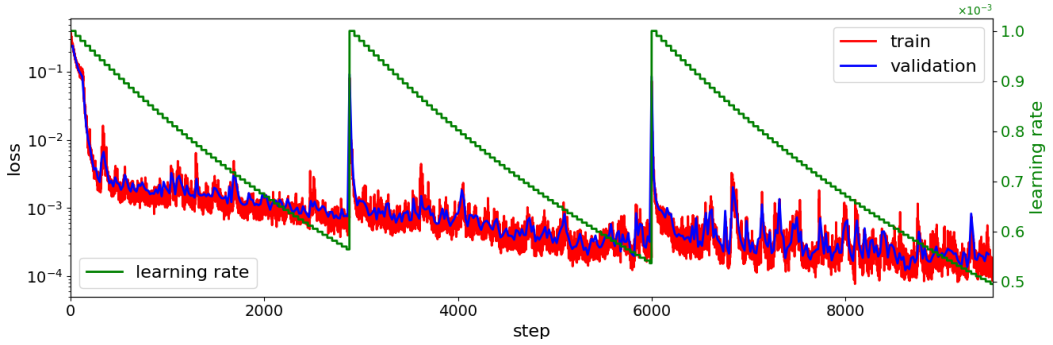
A typical loss functions history is shown in Figure 3b: each loss function is represented multiplied by its weight. We first notice that the prediction loss function $\mathcal{L}^s_{\text{pred}}$ and the auto-encoder loss function $\mathcal{L}_{\text{AE}}$ have the same magnitude and actually are almost equal: the error in prediction is mostly due to the encoder-decoder step. We keep this behavior by assigning them the same weight $\omega_{\text{pred}} = \omega_{\text{AE}} = 0.1$. This value is determined in proportion to the learning rate chosen below. As the prediction loss function $\mathcal{L}^s_{\text{pred}}$ is the most important one for the applications, we want it to dominate over the others. We therefore set the weight $\omega_{\overline{\text{pred}}} = 80$ so that the weighted reduced prediction loss function $\omega_{\overline{\text{pred}}}\mathcal{L}^s_{\overline{\text{pred}}}$ is about 10 times smaller than the previous two. Finally, we want the weighted reduced stability loss function $\omega_{\overline{\text{stab}}}\mathcal{L}^s_{\overline{\text{stab}}}$ to act as a quality control that remains small compared to the other loss functions. To this end, we set $\omega_{\overline{\text{stab}}} = 7 \times 10^{-4}$ in order to make it about 100 times smaller than the weighted reduced prediction loss.

Gradient descent. An Adam optimizer [18] is used for the training. The learning rate follows the following rule:

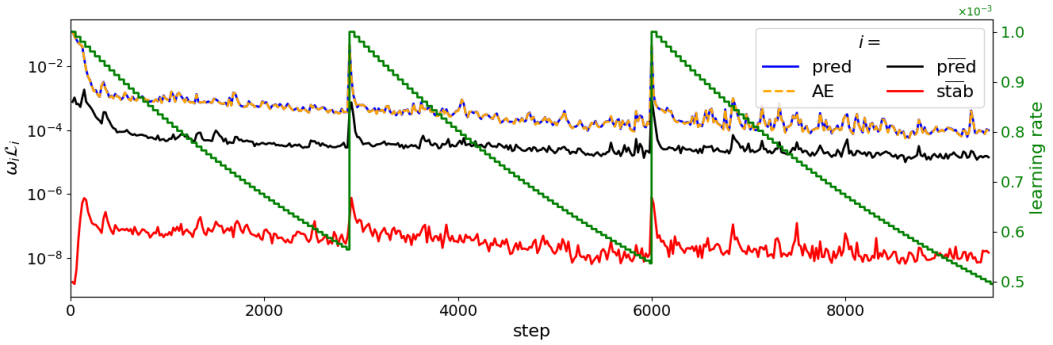
$$\rho_k = (0.99)^{k/150}\, 0.001.$$

where the division operator stands here for the integer division, and k is the train step. Thus, the learning rate is constant over 150 iterations, then decreases. It has an exponential decay with the shape of a staircase. In addition, we can reset this decay i.e. set $k = 0$ at any time if we notice that the loss is reaching a plateau. The main goal of this reset strategy is to escape poor local minima of the minimization problem with a sudden large learning rate. On Figure 3a is shown a typical training and validation loss history as functions of the training step k as well as the learning rate at each step. The resets enable us to make the training loss go from 1×10^{-3} to 5×10^{-4} and then to 1×10^{-4}. With the loss functions weights above-mentioned, we consider 1×10^{-5} to be a correct plateau value to stop the training process. The training phase lasts from 1 to 3 hours on a shared NVIDIA Tesla T4 GPU.

Pre-processing Data pre-processing is required to optimize the learning process. Here we use usual standardization techniques.



(a) Training loss function (blue) and validation loss function (red) as functions of the training step.



(b) All the weighted loss functions as functions of the training step.

Figure 3: Example of loss functions history during a training, overlaid with the evolution of the learning rate (green).

3.5 Numerical complexity

Here, we briefly compare the computational gain in using the reduced models in the online phase. For the original system, the main cost comes from evaluating the N components of the Hamiltonian gradient. If we denote by α the evaluation complexity for one component, the computational cost is therefore about $O(N\alpha)$. When using the reduced PSD model, an additional cost arises from the linear encoding-decoding operations and the computational cost is equal to $O(N\alpha + NK)$. The use of the DEIM-PSD method, as presented in the appendix A.2, allows us to rely only on m components of the gradient of the Hamiltonian and the computation time is therefore about $O(m\alpha + mK)$, which no longer depends on the N dimension. On the other hand, the reduced HNN model relies on the evaluation of the gradient of the neural network Hamiltonian, whose evaluation complexity is, to a first approximation, equivalent to a direct evaluation. Thus, if we denote by n^k the width (i.e. the number of neurons) of the k-th layer and only count the linear operations between the layers, the total complexity of the evaluation is about $O(\sum_{k=1}^{L} n^{(k-1)} n^{(k)})$. Therefore, if the width is of the order of K^2, the complexity of the evaluation is of the order of $O(K^4)$.

Depending on the values of α and m, the reduced model AE-HNN can be competitive. It should also be noted that an additional advantage of the AE-HNN reduced model is that it can naturally be evaluated for a batch of parameters in parallel.

4 Numerical results

This section is devoted to the numerical results obtained with the proposed Hamiltonian reduction method. We consider one-dimensional discretizations of three wave-type equations: the linear wave equation, the non-linear wave equation and the shallow water equation.

4.1 Wave equations

We introduce a parameterized one-dimensional wave equation:

$$\begin{cases} \partial_{tt} u(x,t;\mu) - \mu_a \, \partial_x \left(w' \left(\partial_x u(x,t;\mu), \mu_b \right) \right) + g'(u(x,t;\mu), \mu_c) = 0, & \text{in } [0,1] \times (0,T], \\ u(x,0;\mu) = u_{\text{init}}(x;\mu), & \text{in } [0,1], \\ \partial_t u(x,0;\mu) = v_{\text{init}}(x;\mu), & \text{in } [0,1], \end{cases} \tag{12}$$

complemented with periodic boundary conditions. The solution $u(x,t;\mu)$ represents the vertical displacement of a string over the interval $[0,1]$. The model depends on two given functions $w, g : \mathbb{R} \to \mathbb{R}$ and three parameters: $\mu = (\mu_a, \mu_b, \mu_c)^T \in \Xi \subset \mathbb{R}^3_+$.

Defining the vertical displacement velocity $v(x,t;\mu) = \partial_t u(x,t;\mu)$, Equation (12) can be reformulated as a first order in time system:

$$\begin{cases} \partial_t u(x,t;\mu) - v(x,t;\mu) = 0, & \text{in } [0,1] \times (0,T], \\ \partial_t v(x,t;\mu) + \mu_a \partial_x \left(w' \left(\partial_x u(x,t;\mu), \mu_b \right) \right) + g'(u(x,t;\mu), \mu_c) = 0, & \text{in } [0,1] \times (0,T] \\ u(x,0;\mu) = u_{\text{init}}(x;\mu), & \text{in } [0,1], \\ v(x,0;\mu) = v_{\text{init}}(x;\mu), & \text{in } [0,1]. \end{cases}$$

Then, we consider a spatial finite difference discretization of this system. Introducing a uniform mesh of the interval $[0,1]$, with N cells, space step $\Delta x = 1/(N-1)$ and nodes $x_i = i\Delta x$, for $i \in [\![0, N-1]\!]$, the approximate solution $(\mathbf{u}, \mathbf{v}) = (u_i(t;\mu), v_i(t;\mu))_{i \in [\![0,N-1]\!]} \in \mathbb{R}^N \times \mathbb{R}^N$ satisfies the following system:

$$\begin{cases} \dfrac{d}{dt} u_i(t;\mu) = v_i(t;\mu), & \text{in } (0,T] \\ \dfrac{d}{dt} v_i(t;\mu) = -\dfrac{\mu_a}{\Delta x} \left(w' \left(\dfrac{u_{i+1} - u_i}{\Delta x}, \mu_b \right) - w' \left(\dfrac{u_i - u_{i-1}}{\Delta x}, \mu_b \right) \right) + g'(u_i, \mu_c), & \text{in } (0,T], \\ u_i(0;\mu) = u_{\text{init}}(x_i;\mu), \\ v_i(0;\mu) = v_{\text{init}}(x_i;\mu). \end{cases} \tag{13}$$

These equations actually form a Hamiltonian system of size $2N$, with a separable Hamiltonian function ($(\mathbf{u}, \mathbf{v})$ stands for variables $(\mathbf{q}, \mathbf{p})$) given by

$$\mathcal{H}(\mathbf{u}, \mathbf{v}; \mu) = \mathcal{H}^1(\mathbf{u}; \mu) + \mathcal{H}^2(\mathbf{v}; \mu), \tag{14}$$

with

$$\mathcal{H}^1(\mathbf{u};\mu) = \Delta x \sum_{i=0}^{N-1} \left(\mu_a w \left(\frac{u_{i+1}-u_i}{\Delta x}, \mu_b \right) + \mu_a w \left(\frac{u_i - u_{i-1}}{\Delta x}, \mu_b \right) + g(u_i, \mu_c) \right),$$

$$\mathcal{H}^2(\mathbf{v};\mu) = \frac{1}{2} \Delta x \sum_{i=0}^{N-1} v_i^2.$$

In the following, we test the AE-HNN method for two choices of functions w and g. The same hyper-parameters are used in both test-cases. They are summarized in Table 1.

		wave equations	shallow water 1D	shallow water 2D
AE	type	split	bichannel	bichannel
	nb of convolution blocks (encoder)	4	4	4
	dense layers (encoder)	$[256, 128, 64, 32]$	$[256, 128, 64, 32]$	$[512, 256, 128, 64, 32]$
	activation functions	ELU	swish	ELU
HNN	type	split	standard	standard
	dense layers	$[24, 12, 12, 12, 6]$	$[40, 20, 20, 20, 10]$	$[96, 48, 48, 48, 24]$
	activation functions	tanh	swish	tanh
watch duration	s	16	48	16

Table 1: Hyper-parameters. Activation functions are used except for the last layer of the neural networks. ELU refers to the function $\text{elu}(x) = x1_{x>0} + (e^x - 1)1_{x<0}$ and swish to the function $\text{swish}(x) = x/(1+e^{-x})$. For the auto-encoder (AE), the number of convolution blocks and the sizes of the hidden of layers are those of the encoder. The decoder is constructed in a mirror way.

4.1.1 Reduction of the linear wave equation

Here we consider the linear wave equation:

$$\begin{cases} \partial_{tt} u(x,t;\mu) - \mu_a \, \partial_{xx} u(x,t;\mu) = 0, & \text{in } [0,1] \times (0,T], \\ u(x,0;\mu) = u_{\text{init}}(x;\mu), & \text{in } [0,1]. \end{cases}$$

corresponding to $w(x,\mu_b) = \frac{1}{2}x^2$ and $g(x,\mu_c) = 0$. In particular, we conserve only one parameter $\mu_a \in [0.2, 0.6]$, which corresponds to the square of the wave velocity. The initial condition is taken equal to:

$$u_{\text{init}}(x;\mu) = h(10|x - \tfrac{1}{2}|), \quad v_{\text{init}}(x;\mu) = 0. \tag{15}$$

where h is the compactly supported (single bump) function:

$$h(r) = \left(1 - \tfrac{3}{2}r^2 + \tfrac{3}{4}r^3\right) 1_{[0,1]}(r) + \tfrac{1}{4}(2-r)^3 1_{(1,2]}(r). \tag{16}$$

The initial condition can be observed in Figure 5a. We consider $N = 1024$ discretization points, set $T = 0.4$ and $\Delta t = 1 \times 10^{-4}$. In Figure 4, we observe the solution at final time $T = 0.4$ as a function of μ_a. Each color corresponds to a different value of μ_a. As we can expect, large values of μ_a generate waves farther from the initial condition.

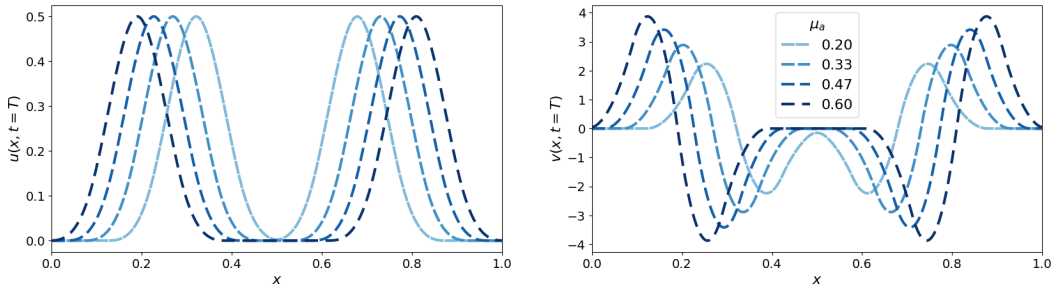


Figure 4: (Linear wave) Solution $(\mathbf{u}, \mathbf{v})$ at final time $T = 0.4$ for various parameters $\mu = \mu_a$

To train the model, we consider numerical solutions obtained with $P = 20$ different values μ_a taken regularly spaced in the interval $[0.2, 0.6]$. The validation set is obtained considering numerical solutions with 6 different parameter values in the same interval and 128 random pairs $(\mathbf{y}^n, \mathbf{y}^{n+s})$ for each validation parameter. The final value of the validation loss functions are given in Table 2.

Then we test the obtained model on 3 test values:

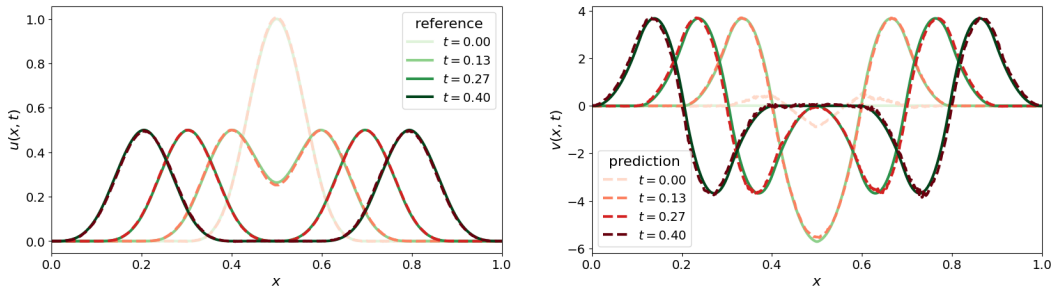
$$\mu_a = 0.2385 \text{ (test 1)}, \quad \mu_a = 0.3798 \text{ (test 2)}, \quad \mu_a = 0.5428 \text{ (test 3)}.$$

To evaluate our model, we compute the relative L^2 discrete error between the reference solution u_{ref} and the reduced model prediction u_{pred}:

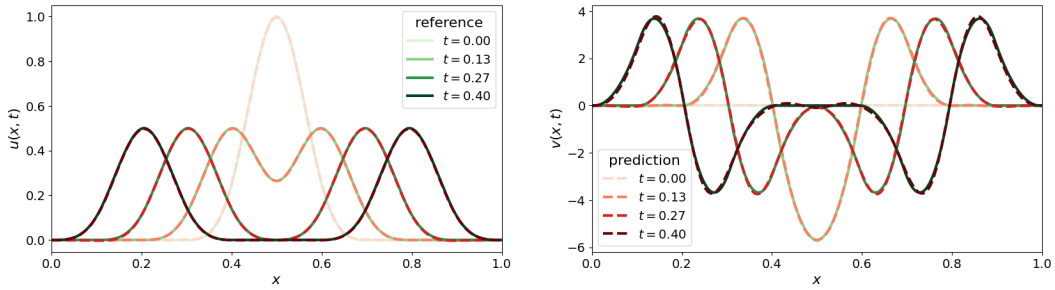
$$\text{err} = \frac{\sum_{n=1}^{M} \Delta t \left(\sum_{i=1}^{N} \Delta x \, (u_{\text{ref},i}^n - u_{\text{pred},i}^n)^2 \right)}{\sum_{n=1}^{M} \Delta t \left(\sum_{i=1}^{N} \Delta x \, (u_{\text{pred},i}^n)^2 \right)},$$

where M is the total number of time steps. Table 3 compares the errors of the AE-HNN method with the ones obtained with the PSD and the POD for different reduced dimensions K. We have colored in blue the cells of the PSD and POD methods with an error lower than the AE-HNN for $K = 1$ (in yellow). As shown in Table 3, we succeed in capturing the dynamics with a AE-HNN model of dimension $K = 1$ with a relative error equal to 1×10^{-2}. To obtain comparable performance, $K = 6$ modes are needed for the PSD and $K = 10$ modes for the POD. The reduction capability of the PSD is nevertheless quite good and this is due to the linearity of the problem. Also, the POD results show that taking into account the symplecticity of the problem enables us to improve significantly the reduction.

Figure 5 shows the simulations of the reduced models for both the AE-HNN ($K = 1$) and the PSD ($K = 6$) methods at different times for test 3. As expected, both methods provide good results.



(a) AE-HNN, $K = 1$



(b) PSD, $K = 6$

Figure 5: (Linear wave) Solution $(\mathbf{u}, \mathbf{v})$ at different times on test 3, reference solution (full lines) and prediction (dashed lines).

	$\mathcal{L}^s_{\text{pred}}$	$\mathcal{L}_{\text{AE}}$	$\mathcal{L}^s_{\overline{\text{pred}}}$	$\mathcal{L}^s_{\overline{\text{stab}}}$
linear wave	8.25×10^{-4}	8.19×10^{-4}	2.52×10^{-7}	1.54×10^{-5}
non-linear wave	3.47×10^{-5}	3.53×10^{-5}	1.84×10^{-8}	1.98×10^{-6}
shallow water 1D	6.49×10^{-5}	6.51×10^{-5}	7.70×10^{-9}	1.19×10^{-7}
shallow water 2D	9.19×10^{-5}	9.11×10^{-5}	3.80×10^{-8}	3.29×10^{-7}

Table 2: Values of the loss functions on validation data for the different test-cases.

		test 1		test 2		test 3	
		error u	error v	error u	error v	error u	error v
AE-HNN	$K=1$	1.20×10^{-2}	6.05×10^{-2}	7.07×10^{-3}	5.84×10^{-2}	1.43×10^{-2}	8.09×10^{-2}
	$K=2$	6.71×10^{-3}	2.24×10^{-2}	1.48×10^{-2}	3.99×10^{-2}	9.71×10^{-3}	2.30×10^{-2}
	$K=5$	1.67×10^{-2}	2.54×10^{-2}	1.48×10^{-2}	2.90×10^{-2}	1.79×10^{-2}	3.55×10^{-2}
PSD	$K=4$	6.70×10^{-1}	3.67×10^{-1}	7.01×10^{-1}	8.69×10^{-1}	7.30×10^{-1}	3.65×10^{-1}
	$K=5$	1.28×10^{-1}	1.34×10^{-1}	1.60×10^{-1}	1.43×10^{-1}	1.91×10^{-1}	1.52×10^{-1}
	$K=6$	4.80×10^{-3}	2.11×10^{-2}	5.52×10^{-2}	2.14×10^{-2}	5.89×10^{-3}	2.23×10^{-2}
POD	$K=6$	3.07×10^{-2}	1.08×10^{-1}	1.30×10^{-2}	2.67×10^{-2}	6.00×10^{-2}	1.54×10^{-1}
	$K=8$	1.28×10^{-2}	4.89×10^{-2}	1.12×10^{-2}	2.66×10^{-2}	3.96×10^{-2}	7.05×10^{-2}
	$K=10$	9.78×10^{-3}	4.36×10^{-2}	2.30×10^{-3}	9.07×10^{-3}	1.58×10^{-2}	5.93×10^{-2}

Table 3: (Linear wave) Relative L^2 errors for different reduced dimensions K. Blue cells correspond to POD and PSD simulations with lower errors than the corresponding AE-HNN simulation in yellow.

4.1.2 Reduction of non-linear wave equation

Now, we consider the following non-linear wave equations:

$$\begin{cases} \partial_{tt}u(x,t;\mu) - \mu_a\,\partial_{xx}u(x,t;\mu) - \mu_b\,\partial_x\left(\cos(\mu_b\partial_x u(x,t;\mu))\right) + 30\mu_c x^2 = 0, & \text{in } [0,1]\times(0,T], \\ u(x,0;\mu) = u_{\text{init}}(x;\mu), & \text{in } [0,1]. \end{cases}$$

corresponding to $w(x,\mu_b) = \frac{1}{2}x^2 + \sin(\mu_b x)$ and $g(x,\mu_c) = 10\mu_c x^3$. There are three parameters $\mu_a \in [0.2, 0.6]$, $\mu_b \in [0.025, 0.5]$ and $\mu_c \in [0.4, 2.4]$. The initial condition is given by Eq. (15)-(16).

The number of discretization points is still $N = 1024$ but the final time is taken equal to $T = 0.3$ and the time step $\Delta t = 1\times10^{-4}$. These parameters have been chosen so that the numerical simulations remain stable despite the strong non-linearities. In Figure 6, we observe the numerical solutions at final time for several sets of parameters.

Training data are obtained using $P = 20$ different values (μ_a, μ_b, μ_c) regularly spaced in the segment $[(0.2, 0.025, 0.4), (0.6, 0.5, 2.4)]$. The validation set is also made of 6 different triplets define on the same domain. At the end of the training, the validation loss takes the values given in Table 2.

The obtained model is tested on 3 sets of parameters:

$$\begin{aligned} (\mu_a,\mu_b,\mu_c) &= (0.2385, 0.088, 0.5485) \qquad \text{(test 1)}, \\ (\mu_a,\mu_b,\mu_c) &= (0.3785, 0.281, 1.354) \qquad \text{(test 2)}, \\ (\mu_a,\mu_b,\mu_c) &= (0.5528, 0.437, 2.128) \qquad \text{(test 3)}, \end{aligned}$$

and Table 4 presents the relative errors. A relative error of order 1×10^{-2} can be reached with the AE-HNN method with a reduced dimension of $K = 3$ only. In comparison, the PSD and the POD require $K = 15$ and $K = 30$ reduced dimensions to obtain similar results.

Figure 7 shows numerical solutions obtained for parameters corresponding to test 3. While the AE-HNN method with $K = 3$ remains close to the reference solution (Fig. 7a), the PSD with the same reduced dimension does not provide satisfactory results (Fig. 7b). Increasing the reduced dimension to $K = 15$ for the PSD allows us to recover comparable results (Fig. 7c).

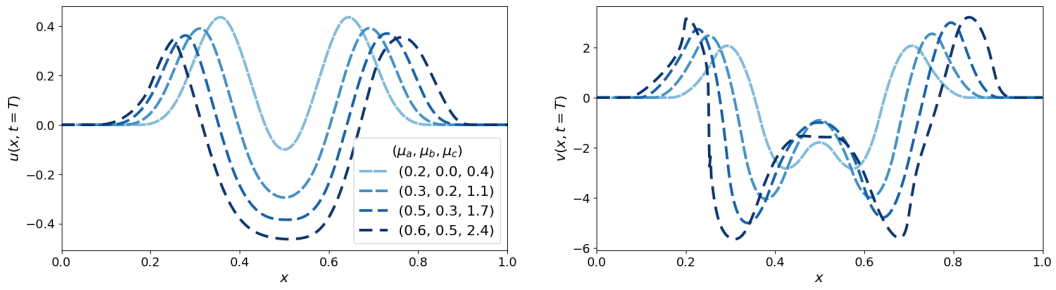
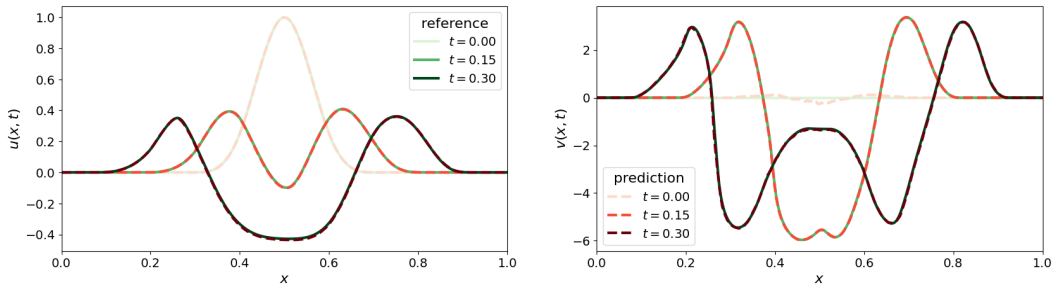


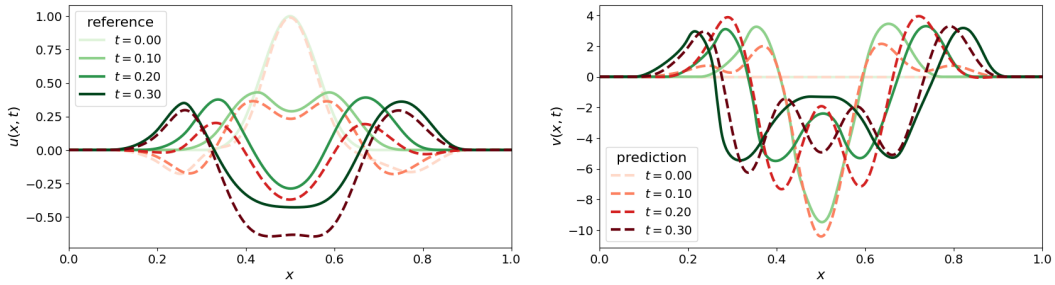
Figure 6: (Non-linear wave) final condition $(\mathbf{u}, \mathbf{v})(t = T; \mu)$ for various parameters $\mu = (\mu_a, \mu_b, \mu_c)$

		test 1		test 2		test 3	
		error u	error v	error u	error v	error u	error v
AE-HNN	$K = 3$	4.34×10^{-3}	1.17×10^{-2}	5.82×10^{-2}	8.65×10^{-2}	1.06×10^{-2}	1.36×10^{-2}
PSD	$K = 3$	4.04×10^{-1}	2.55×10^{-1}	4.21×10^{-1}	2.49×10^{-1}	4.33×10^{-1}	2.73×10^{-1}
	$K = 10$	3.26×10^{-2}	4.92×10^{-2}	3.84×10^{-2}	4.53×10^{-2}	5.63×10^{-2}	4.94×10^{-2}
	$K = 15$	8.48×10^{-3}	1.66×10^{-2}	6.45×10^{-3}	1.66×10^{-2}	5.29×10^{-3}	1.87×10^{-2}
POD	$K = 3$	1.86×10^{-1}	3.16×10^{-1}	5.72×10^{-2}	6.66×10^{-2}	2.44×10^{-1}	2.89×10^{-1}
	$K = 20$	2.08×10^{-2}	4.00×10^{-2}	3.10×10^{-2}	7.00×10^{-2}	3.66×10^{-3}	1.01×10^{-2}
	$K = 30$	5.36×10^{-3}	2.09×10^{-2}	9.54×10^{-3}	2.26×10^{-2}	8.87×10^{-3}	1.93×10^{-2}

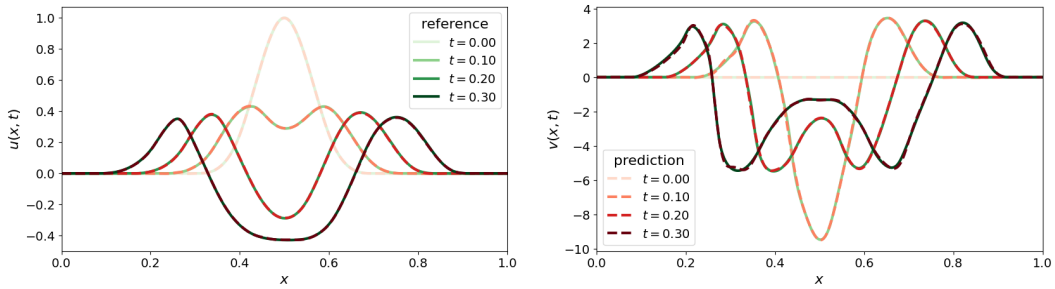
Table 4: (Non-linear wave) Relative L^2 errors for different reduced dimensions K and different parameters. Blue cells correspond to POD and PSD simulations with lower errors than the corresponding AE-HNN simulation in yellow.



(a) AE-HNN, $K = 3$



(b) PSD, $K = 3$



(c) PSD, $K = 15$

Figure 7: (Non-linear wave) Solutions $(\mathbf{u}, \mathbf{v})$ at different times on test 3. Reference solution in solid lines and prediction in dashed lines.

4.1.3 Comparison with a non Hamiltonian reduction

In this section, we want to highlight the importance of the Hamiltonian framework for the reduced dynamics. Indeed, if the Hamiltonian structure is ignored, then the reduced system can be written:

$$\begin{cases} \dfrac{d}{dt}\overline{\mathbf{y}}(t;\mu) = \overline{\mathcal{F}}(\overline{\mathbf{y}}(t;\mu);\mu), \quad \forall t \in (0,T], \\ \overline{\mathbf{y}}(0;\mu) = \mathcal{E}(\mathbf{y}_{\text{init}}(\mu)), \end{cases} \tag{17}$$

where $\overline{\mathcal{F}} : \mathbb{R}^K \times \Xi \to \mathbb{R}^K$ is the reduced vector field. Then the reduced dynamics can be learnt by designing a neural network approximation $\overline{\mathcal{F}}_{\theta_f}$, with a classical MLP architecture.

To this aim, we can consider the following loss function:

$$\mathcal{L}^s_{\text{Flow}}(\theta_e, \theta_f) = \sum_{\mathbf{y}^n_\mu, \mathbf{y}^{n+s}_\mu \in U} \left\| \overline{\mathbf{y}}^{n+s}_\mu - \mathcal{P}_s\left(\overline{\mathbf{y}}^n_\mu; \overline{\mathcal{F}}_{\theta_f}\right) \right\|_2^2, \tag{18}$$

where the prediction operator is here a simple RK2 scheme. We discard the stability loss function $\mathcal{L}^s_{\overline{\text{stab}}}$ and keep all the other hyper-parameters identical to those of the previous section. Coupled with the auto-encoder, this reduction method will be referred to as AE-Flow.

We compare the two resulting method on the non-linear wave test-case considered in Section 4.1.2. The HNN parameters are unchanged. For a fair comparison, the flow neural network $\mathcal{F}_{\theta_f}$ has the same amount of parameters: the hidden layers have $[32, 24, 16, 16]$ units, which results in 1 886 parameters instead of 1 728 for the HNN. We train both models up to reach a loss value of about 1×10^{-5}. The obtained validation loss functions are as follows

$$\mathcal{L}^s_{\text{pred}} = 9.46 \times 10^{-5}, \quad \mathcal{L}_{\text{AE}} = 9.88 \times 10^{-5}, \quad \mathcal{L}^s_{\text{Flow}} = 7.92 \times 10^{-7}.$$

They are of comparable magnitude to that of the AE-HNN method in Table 2.

Relative errors are provided in Table 5. The errors of the AE-HNN method are about 5 times smaller than those of the AE-Flow method. Figure 8 shows the time evolution of the relative error for test case 3 and Figure 9 depicts the solutions at different times. We observe that the AE-HNN solution remains close to the reference while the AE-Flow solution drifts from it. Furthermore, we compare in Figure 10 the time evolution of the Hamiltonian for the reference solution, the AE-HNN solution, and the AE-Flow solution on test case 3. We observe that the AE-HNN method results in a better preservation of the Hamiltonian than the AE-Flow solution.

	test 1		test 2		test 3	
	error u	error v	error u	error v	error u	error v
AE-HNN	4.35×10^{-3}	1.18×10^{-2}	5.82×10^{-2}	8.66×10^{-2}	1.05×10^{-2}	1.36×10^{-2}
AE-Flow	5.11×10^{-2}	5.56×10^{-2}	1.27×10^{-1}	1.50×10^{-1}	9.99×10^{-2}	1.07×10^{-1}

Table 5: (Non linear wave) Relative L^2 errors for the AE-Flow and the AE-HNN

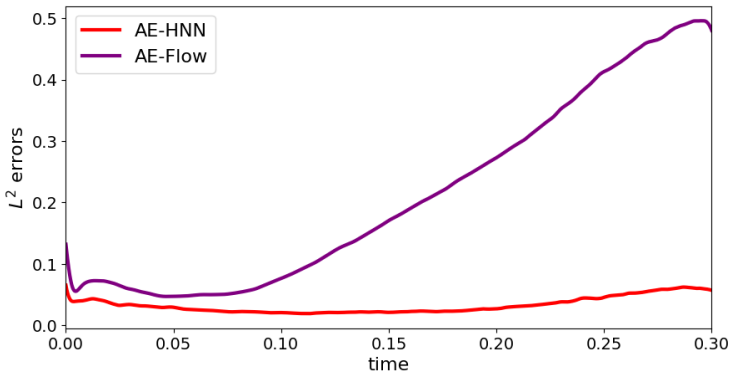


Figure 8: (Non linear wave) L^2 errors on test 3 as a function of time for the AE-Flow (purple) and the AE-HNN (red)

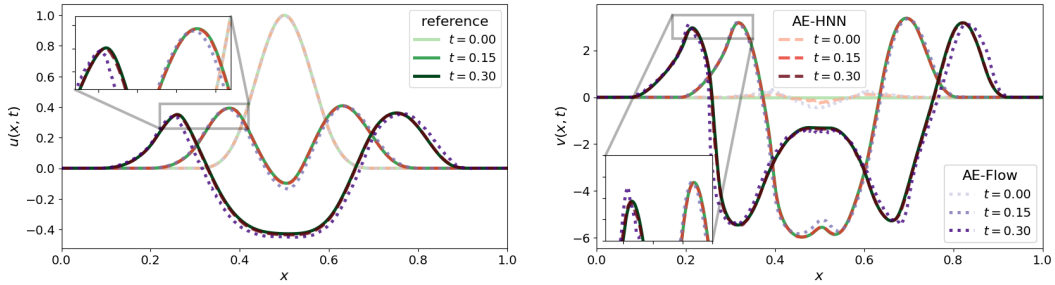


Figure 9: (Non linear wave) $(\mathbf{u}, \mathbf{v})$ for different times on test 3, reference solution (green), AE-HNN solution (red) and AE-Flow solution (purple)

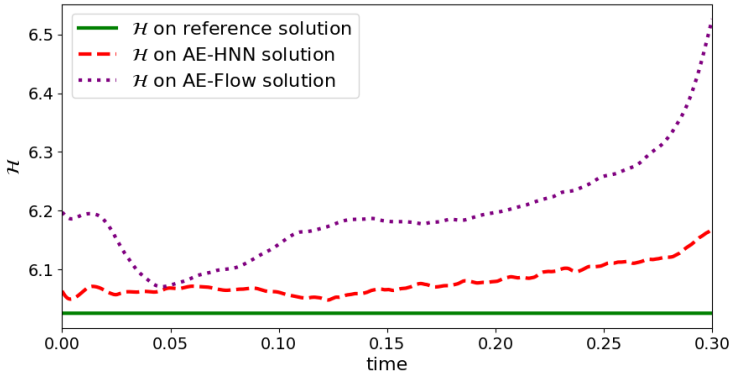


Figure 10: (Non linear wave) Time evolution of the Hamiltonian on test 3 for the reference solution (green), the AE-HNN solution (red) and the AE-Flow solution (purple).

4.1.4 Gain in computation time

In this section, we compare the actual computation time on an Intel Xeon CPU. To obtain a fair comparison, we use the same implementation of the Störmer-Verlet algorithm for all the methods. The only difference between the PSD and the AE-HNN methods lies in the gradient computations. For the former, we use the explicit expression of the reduced Hamiltonian as defined in Appendix A and for the latter, we use the HNN backpropagation algorithm. We use 32 bits floating-point numbers and include the encoding or decoding of the data in the computation time. We consider the non-linear wave test case with the following parameters: $N = 1024, K = 3, T = 0.4$ and $\Delta t = 1 \times 10^{-4}$.

The reference solution without reduction is computed in $2\,200 \pm 14$ ms. In comparison, the AE-HNN method spends 452 ± 20 ms, which is five times faster. On the contrary, the PSD method spends $2\,382 \pm 12$ ms, which is a bit slower than the reference solution. This results from the evaluation of the gradient of the Hamiltonian in the non-reduced dimension. Hyper-reduction technic like Hamiltonian DEIM would accelerate the computations [15]. However, this could also deteriorate the precision, and one should then consider larger reduced dimensions.

4.2 1D shallow water system

We consider the one-dimensional shallow water system under the following formulation:

$$\begin{cases} \partial_t \chi(x,t;\mu) + \partial_x \left((1+\chi(x,t;\mu))\, \partial_x \phi(x,t;\mu) \right) = 0, & \text{in } [-1,1] \times (0,T], \\ \partial_t \phi(x,t;\mu) + \frac{1}{2} (\partial_x \phi(x,t;\mu))^2 + \chi(x,t;\mu) = 0, & \text{in } [-1,1] \times (0,T], \\ \chi(x,0;\mu) = \chi_{\text{init}}(x;\mu), & \text{in } [-1,1], \\ \phi(x,0;\mu) = \phi_{\text{init}}(x;\mu), & \text{in } [-1,1], \end{cases}$$

where $\chi(x,t;\mu)$ denotes the perturbation of the equilibrium and $\phi(x,t;\mu)$ is the scalar velocity potential. Periodic boundary conditions are considered. This system admits a Hamiltonian function given by:

$$\mathcal{H}_{\text{cont}}(\chi,\phi) = \frac{1}{2} \int_{-1}^{1} \left((1+\chi)(\partial_x \phi)^2 + \chi^2 \right) dx.$$

Like the wave equation, we consider a spatial discretization of this model, still using a uniform mesh grid with N nodes $(x_i)_{i \in [\![0,N-1]\!]}$ on $[-1,1]$. The approximate solution is denoted $\boldsymbol{\chi} = (\chi_i)_{i \in [\![0,N-1]\!]}, \boldsymbol{\phi} = (\phi_i)_{i \in [\![0,N-1]\!]}$. We then introduce the discrete Hamiltonian function:

$$\mathcal{H}(\boldsymbol{\chi},\boldsymbol{\phi}) = \frac{\Delta x}{2} \sum_{i=1}^{N} (1+\chi_i) \left(\frac{\phi_{i+1} - \phi_{i-1}}{\Delta x} \right)^2 + \chi_i^2, \tag{19}$$

and the resulting discrete shallow water equation:

$$\begin{cases} \dfrac{d}{dt} \boldsymbol{\chi}(t;\mu) = -D_x^2 \boldsymbol{\phi}(t;\mu) - D_x(\boldsymbol{\chi}(t;\mu) \odot D_x \boldsymbol{\phi}(t;\mu)), \\ \dfrac{d}{dt} \boldsymbol{\phi}(t;\mu) = -\dfrac{1}{2} (D_x \boldsymbol{\phi}(t;\mu)) \odot (D_x \boldsymbol{\phi}(t;\mu)) - \boldsymbol{\chi}(t;\mu), \\ \chi_i(0;\mu) = \chi_{\text{init}}(x_i;\mu), \\ \phi_i(0;\mu) = \phi_{\text{init}}(x_i;\mu), \end{cases} \tag{20}$$

where $\odot$ denotes the element-wise vector multiplication and D_x the centered second-order finite difference matrix with periodic boundary condition

$$D_x = \frac{1}{2\Delta x} \begin{pmatrix} 0 & 1 & & -1 \\ -1 & & \ddots & \\ & \ddots & & 1 \\ 1 & & -1 & 0 \end{pmatrix}.$$

We note that Hamiltonian (19) is not separable. As a consequence, the numerical resolution of (20) with the Störmer-Verlet scheme is implicit.Therefore, applying a reduction is all the more attractive.

4.2.1 Reduction of the system

The number of discretization points is taken equal to $N = 1024$, the final time equal to $T = 0.5$ and the time step $\Delta t = 2 \times 10^{-4}$. The initial condition is parameterized with 2 parameters $\mu = (c, \sigma) \in [0, 0.2] \times [0.2, 0.05]$

$$\chi_{\text{init}}(x; \mu) = \frac{0.02}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2\right), \quad \phi_{\text{init}}(x; \mu) = 0.$$

For the training data, we use 20 different solution parameters (c, σ): we take regularly spaced values in the segment $[(0, 0.2), (0.2, 0.05)]$. Let us observe the influence of the parameters on the solutions at initial and final time on Figure 11. Non-linear patterns appear for small values of the standard deviation σ.

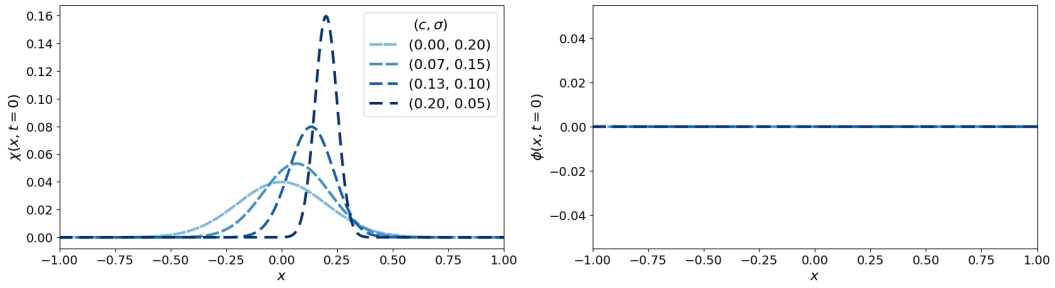
The set of hyper-parameters is almost identical to the previous test cases except for the HNN architecture, activation functions and the watch duration, which here is taken equal to 48. They are gathered in Table 1. As in the previous sections, we inspect the validation loss functions and obtain value given in Table 2.

We choose three different sets of parameters to test the AE-HNN method:

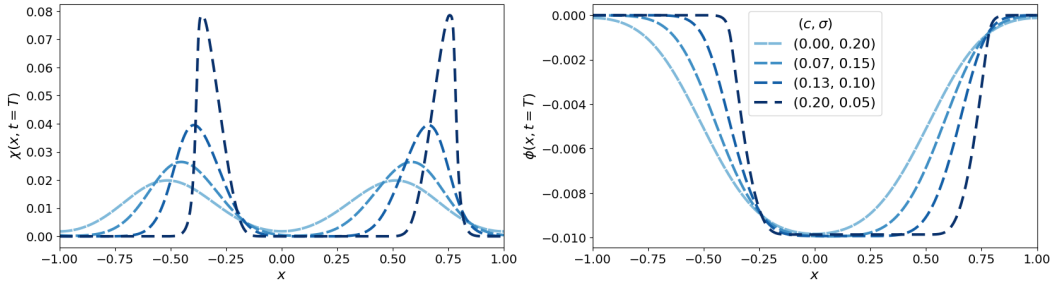
$$\begin{aligned} &(c, \sigma) = (0.105, 0.11), && \text{(test 1)} \\ &(c, \sigma) = (0.195, 0.053), && \text{(test 2)} \\ &(c, \sigma) = (0.21, 0.045). && \text{(test 3)} \end{aligned}$$

As in the previous test cases, we compute the relative errors of the AE-HNN method with respect to the reference solution (see Table 6) and compare them to the results obtained with the PSD and POD methods for different values of K.

The AE-HNN method achieves a mean error of about 3×10^{-2} with a reduced dimension of $K = 4$ only. To achieve a similar performance, the PSD requires a reduced dimension of $K = 24$ and the POD need a value larger than $K = 32$. Table 6 also shows that the AE-HNN method has a different behavior with respect to K than the POD and PSD methods. For the latter, increasing K improves accuracy (at the expense of computation time). With the AE-HNN method, increasing K improves the encoder-decoder performance but makes it more difficult to learn the reduced dynamics for the HNN neural network. Therefore, the HNN performance requires a balance between an adequate compression and a low-dimensional reduced model.



(a) $t = 0$



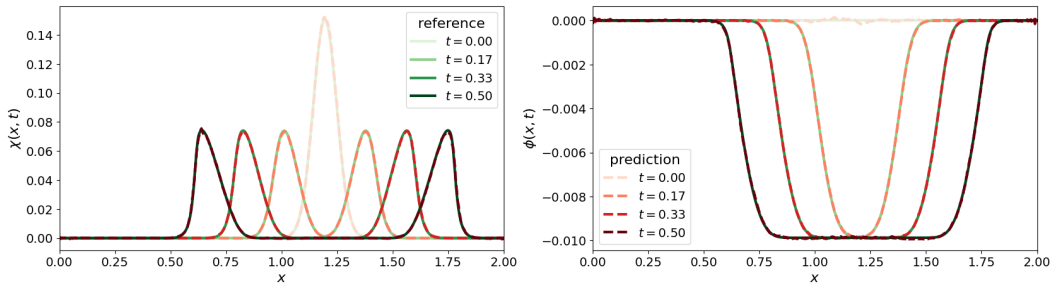
(b) $t = 0.5$

Figure 11: (Shallow water 1D) Solutions $(\boldsymbol{\chi}, \boldsymbol{\phi})$ at initial time $t = 0$ and final time $t = 0.5$ for various parameters (c, σ).

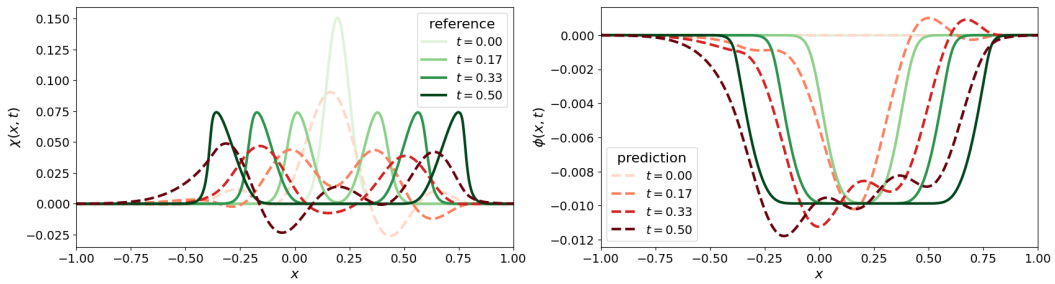
Then, we compare the solutions obtained with the AE-HNN method and the PSD in Figure 12. For a given reduced dimension $K = 4$, the AE-HNN method solution remains close to the reference as expected (Fig. 12a) while the PSD solution oscillates and stays far from the reference solution, even for the initial condition (Fig. 12b). When increasing the dimension of the reduced model to $K = 24$ for the PSD method (Fig. 12c), we recover similar results as the ones obtained with the AE-HNN.

		test 1		test 2		test 3	
		error χ	error ϕ	error χ	error ϕ	error χ	error ϕ
AE-HNN	$K = 4$	5.86×10^{-2}	2.89×10^{-2}	1.34×10^{-2}	5.00×10^{-3}	8.12×10^{-2}	2.27×10^{-2}
	$K = 6$	8.46×10^{-2}	5.17×10^{-2}	2.07×10^{-2}	7.77×10^{-3}	9.70×10^{-2}	2.80×10^{-2}
	$K = 8$	6.26×10^{-2}	3.54×10^{-2}	2.21×10^{-2}	1.20×10^{-2}	1.45×10^{-1}	4.99×10^{-2}
PSD	$K = 10$	7.11×10^{-2}	1.71×10^{-2}	1.64×10^{-1}	2.74×10^{-2}	3.08×10^{-1}	5.89×10^{-2}
	$K = 14$	2.67×10^{-2}	5.36×10^{-3}	7.98×10^{-2}	1.00×10^{-2}	2.08×10^{-1}	3.17×10^{-2}
	$K = 24$	1.19×10^{-2}	3.43×10^{-3}	2.69×10^{-2}	4.20×10^{-3}	9.56×10^{-2}	1.06×10^{-2}
POD	$K = 14$	1.13×10^{-1}	4.14×10^{-2}	1.22×10^{-1}	4.10×10^{-2}	6.98×10^{-1}	2.99×10^{-1}
	$K = 24$	4.22×10^{-2}	9.91×10^{-3}	3.31×10^{-2}	7.81×10^{-3}	3.10×10^{-1}	7.96×10^{-2}
	$K = 32$	8.70×10^{-3}	1.67×10^{-3}	1.32×10^{-2}	2.31×10^{-3}	1.35×10^{-1}	2.43×10^{-2}

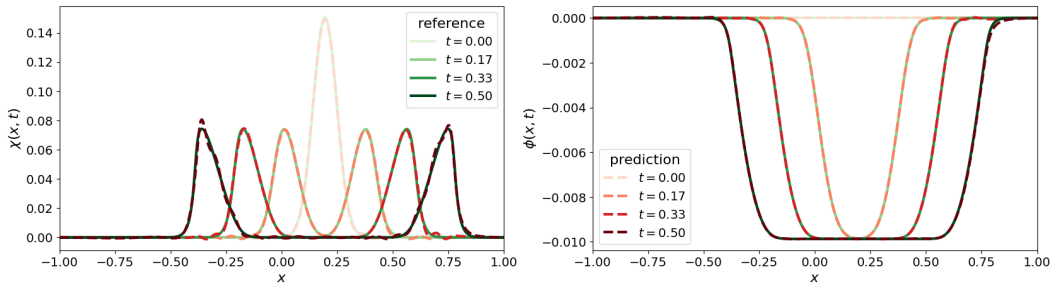
Table 6: (Shallow water 1D) Relative L^2 errors for different reduced dimensions K. Blue cells correspond to POD and PSD simulations with lower errors than the corresponding AE-HNN simulation in yellow.



(a) AE-HNN, $K = 4$



(b) PSD, $K = 4$



(c) PSD, $K = 24$

Figure 12: (Shallow water 1D) $(\boldsymbol{\chi}, \boldsymbol{\phi})$ as a at different times on test 2 and $K = 8$, reference solution (full lines) and prediction (dashed lines).

4.2.2 Comparison with a non Hamiltonian reduction

In this section, we perform a study similar to that performed in Section 4.1.3 to show the importance of having a Hamiltonian AE-HNN reduction instead of a classical AE-Flow reduced model. We consider the same test case as in Section 4.2.1 with the same hyper-parameters described in Table 1. We stop both AE-HNN and AE-Flow training after reaching a validation loss value of 3×10^{-5}. The obtained validation loss functions for the AE-Flow are as follows

$$\mathcal{L}^s_{\text{pred}} = 7.02\times10^{-5}, \quad \mathcal{L}_{\text{AE}} = 6.97\times10^{-5}, \quad \mathcal{L}^s_{\text{Flow}} = 2.85\times10^{-7}.$$

They are of comparable magnitude to that of the AE-HNN method in Table 2. Relative errors are provided in Table 7 for the different test values. The AE-HNN method is about 4 times more precise than the AE-Flow method. Figure 13 shows the time evolution of the L^2 errors for test case 2 and Figure 14 depicts the solution at different times. The AE-Flow drifts away from the reference solution while the AE-HNN remains close to it. Finally, Figure 15 shows that the AE-HNN method preserves the Hamiltonian more effectively than the AE-Flow method.

	test 1		test 2		test 3	
	error χ	error ϕ	error χ	error ϕ	error χ	error ϕ
AE-HNN	5.86×10^{-2}	2.89×10^{-2}	1.34×10^{-2}	5.00×10^{-3}	8.12×10^{-2}	2.27×10^{-2}
AE-Flow	6.63×10^{2}	3.66×10^{-2}	1.05×10^{-1}	3.61×10^{-2}	1.73×10^{-1}	5.12×10^{-2}

Table 7: (Shallow water 1D) Relative L^2 errors for the AE-Flow and the AE-HNN

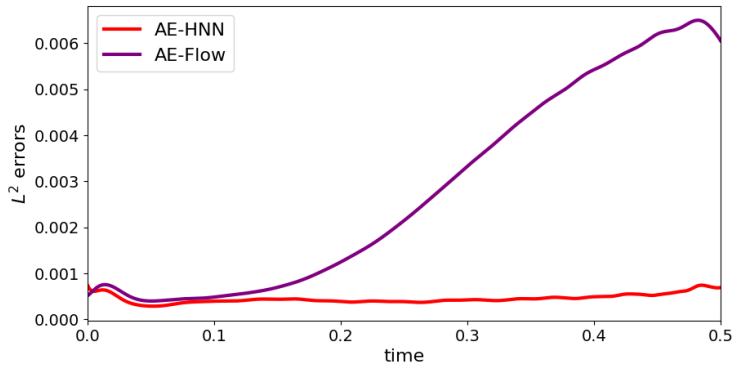


Figure 13: (Shallow water 1D) L^2 errors on test 2 as a function of time for the AE-Flow (purple) and the AE-HNN (red)

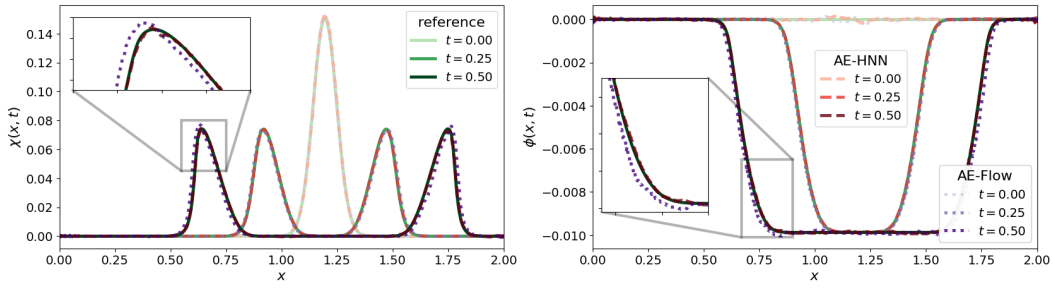


Figure 14: (Shallow water 1D) $(\boldsymbol{\chi}, \boldsymbol{\phi})$ for different times on test 2, reference solution (green), AE-HNN solution (red) and AE-Flow solution (purple)

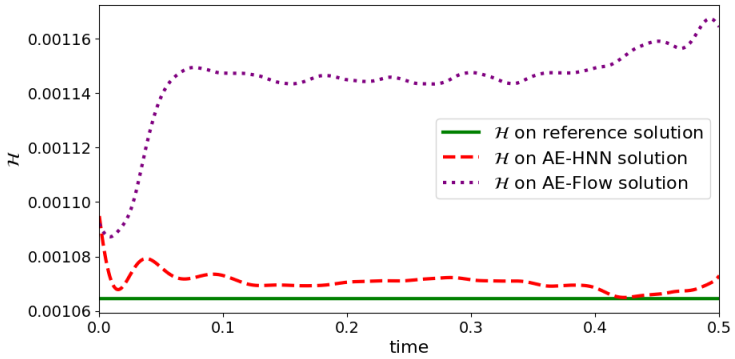


Figure 15: (Shallow water 1D) Time evolution of the Hamiltonian on test 2 for the reference solution (green), the AE-HNN solution (red) and the AE-Flow solution (purple).

4.3 2D shallow water system

We consider the two-dimensional shallow water system on a square $\Omega = [-1, 1]^2$ under the following formulation:

$$\begin{cases} \partial_t \chi(\mathbf{x}, t; \mu) + \nabla \cdot ((1 + \chi(\mathbf{x}, t; \mu)) \nabla \phi(\mathbf{x}, t; \mu)) = 0, & \text{in } \Omega \times (0, T], \\ \partial_t \phi(\mathbf{x}, t; \mu) + \frac{1}{2} |\nabla \phi(\mathbf{x}, t; \mu)|^2 + \chi(\mathbf{x}, t; \mu) = 0, & \text{in } \Omega \times (0, T], \\ \chi(\mathbf{x}, 0; \mu) = \chi_{\text{init}}(\mathbf{x}; \mu), & \text{in } \Omega, \\ \phi(\mathbf{x}, 0; \mu) = \phi_{\text{init}}(\mathbf{x}; \mu), & \text{in } \Omega, \end{cases} \tag{21}$$

where $\chi(\mathbf{x}, t; \mu)$ denotes the perturbation of the equilibrium and $\phi(\mathbf{x}, t; \mu)$ is the scalar velocity potential. Periodic boundary conditions are considered. This system admits a Hamiltonian function given by:

$$\mathcal{H}_{\text{cont}}(\chi, \phi) = \frac{1}{2} \int_\Omega \left((1 + \chi) |\nabla \phi|^2 + \chi^2 \right).$$

We consider a spatial discretization of the domain Ω with a regular mesh of $N = M^2$ cells of size $\Delta x = 2/(M-1)$ in each direction. The discrete Hamiltonian function is

$$\mathcal{H}(\boldsymbol{\chi}, \boldsymbol{\phi}) = \frac{1}{2} \sum_{i,j=0}^{M-1} \left((1 + \chi_{i,j}) \left[\left(\frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} \right)^2 + \left(\frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y} \right)^2 \right] + \chi_{i,j}^2 \right)$$

with $\chi_{i,j}(t;\mu) \approx \chi(\mathbf{x}_{i,j}, t; \mu)$ (resp. $\phi_{i,j}(t;\mu) \approx \phi(\mathbf{x}_{i,j}, t; \mu)$), with $\mathbf{x}_{i,j} = (-1,-1) + (i\Delta x, j\Delta x)$. The resulting discrete system reads

$$\begin{cases} \dfrac{d}{dt}\boldsymbol{\chi}(t;\mu) = -D_x \left([1 + \boldsymbol{\chi}(t;\mu)] \odot D_x \boldsymbol{\phi}(t;\mu) \right) - D_y \left([1 + \boldsymbol{\chi}(t;\mu)] \odot D_y \boldsymbol{\phi}(t;\mu) \right), \\ \dfrac{d}{dt}\boldsymbol{\phi}(t;\mu) = -\frac{1}{2} \left[\left(D_x \boldsymbol{\phi}(t;\mu) \right)^2 + \left(D_y \boldsymbol{\phi}(t;\mu) \right)^2 \right] - \boldsymbol{\chi}(t;\mu), \\ \chi_m(0;\mu) = \chi_{\text{init}}(\mathbf{x}_m;\mu), \\ \phi_m(0;\mu) = \chi_{\text{init}}(\mathbf{x}_m;\mu), \end{cases}$$

where D_x and D_y are respectively the centered finite difference operators along the x and y axis.

4.3.1 Reduction of the system

We consider $M = 64$ cells per direction, The final time is set to $T = 15$ and the time step to $\Delta t = 1 \times 10^{-3}$. In this test case, we choose to use an implicit midpoint scheme [14]. The initial condition, parameterized with two parameters $\mu = (\alpha, \beta) \in [0.2, 0.5] \times [1, 1.7]$, is chosen equal to

$$\chi_{\text{init}}(\mathbf{x};\mu) = \alpha \exp\left(-\beta\, \mathbf{x}^T \mathbf{x}\right), \quad \phi_{\text{init}}(\mathbf{x};\mu) = 0.$$

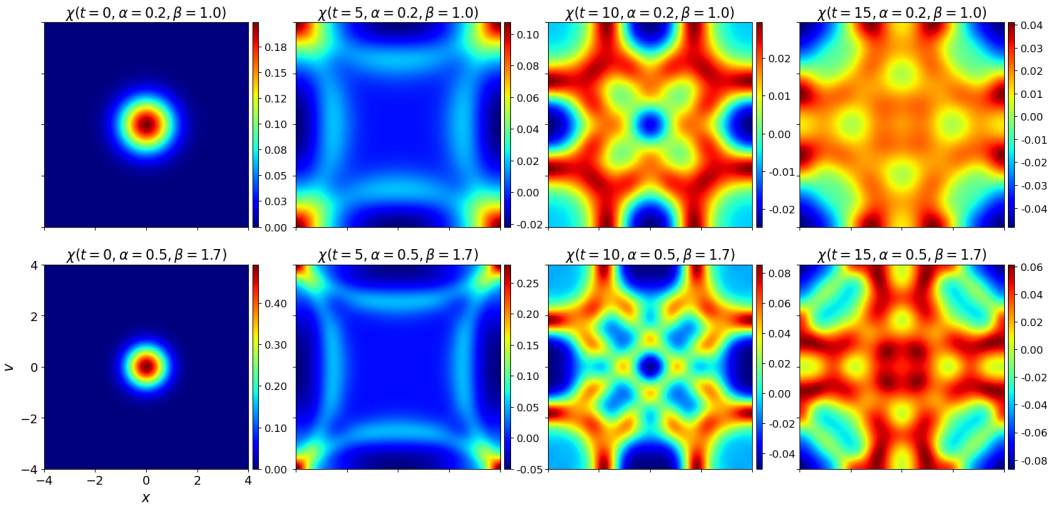
For the training data, we use 20 different couples of parameter (α, β) regularly spaced in the segment $[(0.2, 0.5), (1, 1.7)]$. Figure 15 shows the time evolution for two couples of parameters: $(\alpha, \beta) = (0.2, 1)$ and $(\alpha, \beta) = (0.5, 1.7)$.

As data are two-dimensional, we use a 2D variant of the convolutional AE: convolutional layers are two-dimensionals, up and down-sampling are extended in 2D. All the neural network hyper-parameters are given in Table 1.

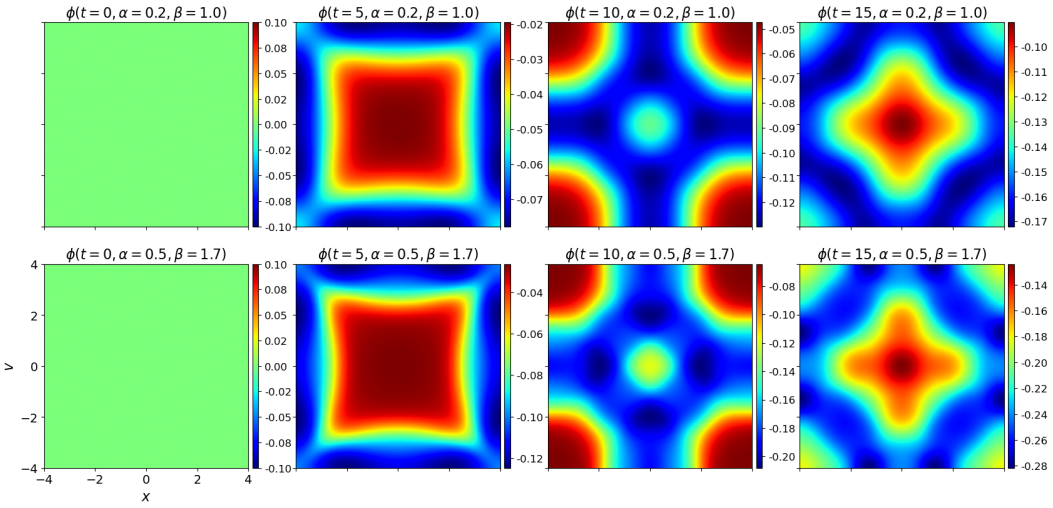
We choose three different sets of parameters to test the AE-HNN method

$$\begin{aligned} &(\alpha, \beta) = (0.35, 1.35), && \text{(test 1)} \\ &(\alpha, \beta) = (0.41, 1.49), && \text{(test 2)} \\ &(\alpha, \beta) = (0.51, 1.72). && \text{(test 3)} \end{aligned}$$

As in the previous test cases, we compute relative errors of the AE-HNN method with respect to the reference solution and compare them to the results obtained with the PSD and POD methods for different values of K in Table 8. Regarding the AE-HNN method, a fixed reduced dimension of $K = 4$ if sufficient to obtain a precise reduced model while, for the same reduced dimension, the PSD solution is far from the reference solution as it can be observed in Fig. 16. When increasing the reduced dimension to $K = 30$, the PSD produces similar results as the ones obtained with the AE-HNN with only $K = 4$. Regarding the POD reduced model, even a reduced dimension of $K = 35$ does not provide the targeted precision. Indeed, the non-symplecticity of the reduced model produces some instabilities as it can be observed on test 3 in Fig. 17.



(a) $\boldsymbol{\chi}(t;\mu)$



(b) $\boldsymbol{\phi}(t;\mu)$

Figure 15: (Shallow water 2D) Solutions $(\boldsymbol{\chi}, \boldsymbol{\phi})$ at different times $t \in \{0, 5, 10, 15\}$ for various parameters $(\alpha, \beta) \in \{(0.2, 1), (0.5, 1.8)\}$.

		test 1		test 2		test 3	
		error χ	error ϕ	error χ	error ϕ	error χ	error ϕ
AE-HNN	$K=4$	4.68×10^{-2}	1.37×10^{-2}	1.73×10^{-2}	5.03×10^{-3}	3.33×10^{-2}	8.92×10^{-3}
	$K=5$	2.32×10^{-2}	6.25×10^{-3}	7.38×10^{-2}	1.62×10^{-2}	1.39×10^{-1}	2.98×10^{-2}
PSD	$K=6$	3.48×10^{-1}	3.96×10^{-2}	3.82×10^{-1}	4.50×10^{-2}	4.33×10^{-1}	5.41×10^{-2}
	$K=20$	5.05×10^{-2}	3.39×10^{-3}	6.52×10^{-2}	4.55×10^{-3}	9.49×10^{-2}	7.13×10^{-3}
	$K=30$	1.37×10^{-2}	9.20×10^{-4}	1.87×10^{-2}	1.19×10^{-3}	3.09×10^{-2}	2.01×10^{-3}
POD	$K=10$	4.51×10^{-1}	3.69×10^{-2}	4.81×10^{-1}	4.47×10^{-2}	5.33×10^{-1}	6.02×10^{-2}
	$K=16$	1.19×10^{-1}	6.31×10^{-3}	4.04×10^{-1}	1.76×10^{-2}	1.01×10^{0}	4.52×10^{-2}
	$K=35$	3.94×10^{-2}	1.83×10^{-3}	5.74×10^{-2}	2.67×10^{-3}	5.05×10^{-1}	2.52×10^{-2}

Table 8: (Shallow water 2D) Relative L^2 errors for different reduced dimensions K. Blue cells correspond to POD and PSD simulations with lower errors than the corresponding AE-HNN simulation in yellow.

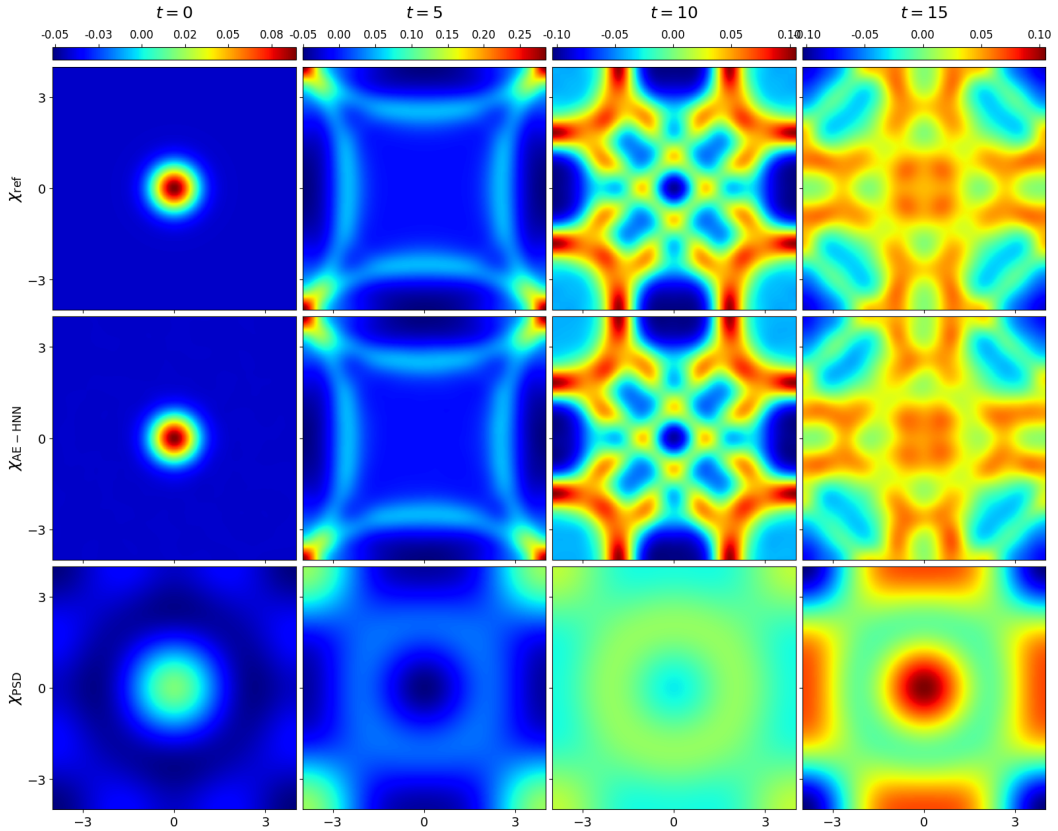


Figure 16: (Shallow water 2D) Solutions $\boldsymbol{\chi}(t;\mu)$ at different times $t \in \{0, 5, 10, 15\}$ on test 3 with $K = 4$, reference solution (top line), AE-HNN solution (middle line) and PSD solution (bottom line).

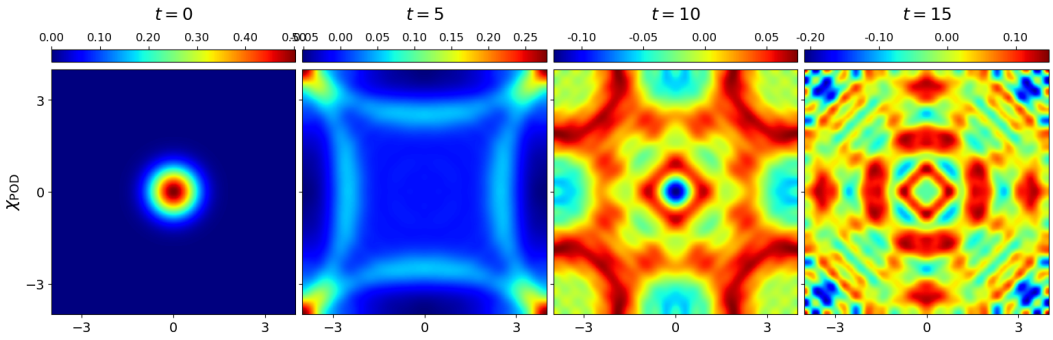


Figure 17: (Shallow water 2D) POD solution $\boldsymbol{\chi}(t;\mu)$ at different times $t \in \{0, 5, 10, 15\}$ on test 3 with $K = 35$.

4.3.2 Comparison with a symplectic DEIM hyper-reduction

While the AE-HNN would require a smaller reduced dimension than the PSD for a given targeted precision, practical efficiency still need to be compared. As the PSD still requires to come back to the original $2N$ dimension for non-linear models, we also compare the AE-HNN with the Discrete Empirical Interpolation Method (DEIM) version of the PSD as proposed in [25]: it relies on an interpolation of m given components (among the $2N$ ones). For the sake of completeness, the method is briefly presented in Annex A.2.

First, we test the precision of this DEIM approximation as a function of m. We consider the test-case of the previous subsection, with reduced dimension equal to $K = 30$. We compute the discrete L^2 error on the 3 test cases:

$$\overline{\mathrm{err}}_u^n = \frac{1}{3} \sum_{\mu \in \text{ test}} \left(\sum_{i=1}^{N} \Delta x \, (u_{\mathrm{ref},i}^{\mu,n} - u_{\mathrm{pred},i}^{\mu,n})^2 \right).$$

where $\overline{u}_{\mathrm{ref},i}$ refers to the reference solution computed with the original model and $u_{\mathrm{pred},i}^{\mu,n}$ the one computed with the reduced one. We observe the time evolution of this error for different values of m on Fig 18. As expected, the DEIM method deteriorates the reduced model solution with respect to the sole PSD and increasing m decreases the error. In practice, the value of m is set so that the DEIM does not deteriorate the solution too much.

We then compare the computational times of the AE-HNN method with respect to the PSD algorithm with and without symplectic DEIM hyper-reduction. We set $K = 4$ and $m = 32$ so that the DEIM error is smaller than the PSD error. Numerical integration are performed with an implicit midpoint scheme as above-mentioned. The original model and the PSD without DEIM are solved on an Intel Xeon CPU while the AE-HNN method is executed on a NVIDIA Tesla T4 GPU. We also use the Strang splitting method [14] where the linear part is solved explicitly and the non-linear one is solved with numerical integration except for the AE-HNN method which cannot be split. The computational time equals 101.0 s for the original model, 107.0 s for the PSD and 57.4 s for the DEIM-PSD. As expected, the PSD is not time efficient in a non-linear case due to the decompression-compression of the solution at each time step. An efficient DEIM-PSD implementation allows a satisfactory speed-up of a factor 1.76. Regarding the AE-HNN model, it is solved in 26.5 s, which corresponds to a speed up of 3.81. This could be further improved. Indeed,

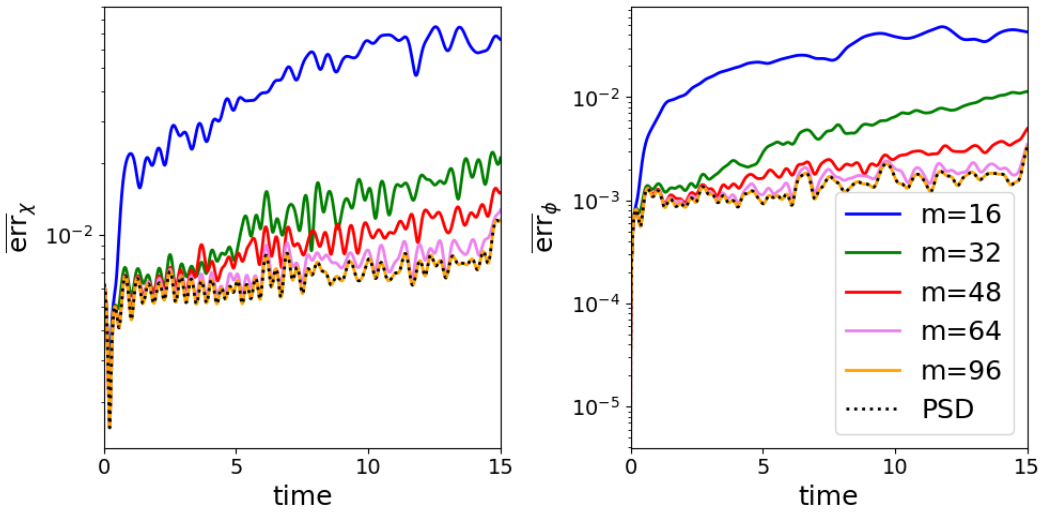


Figure 18: (Shallow water 2D) $\overline{\text{err}}$ for solutions $\boldsymbol{\chi}(t;\mu)$ (left) and $\boldsymbol{\phi}(t;\mu)$ (right) for different $m \in \{0, 5, 10, 15\}$ with $K = 30$.

the neural networks are executed on a GPU but the non-linear solver used in the implicit part of the scheme is executed on a CPU, which slows down the reduced order model. In practice, a SciPy [31] implementation of the nonlinear solver [9] is used. More than 62% of the non-linear solver computation time takes place on the CPU.

5 Conclusion

We have developed a new Hamiltonian reduction method. It is based on an auto-encoder (AE) to transform initial variables into reduced variables and vice versa, and on a Hamiltonian Neural Network (HNN) to learn the Hamiltonian reduced dynamics. Using a set of coupled loss functions, we are able to learn a reduced model (AE-HNN), which has an Hamiltonian structure. It already has better reduction properties than the PSD in linear test-cases, but the gain is much larger in non-linear test-cases, as expected. Due to its Hamiltonian structure, the reduced model also shows good stability properties. Two-dimensional test-cases show that the AE-HNN method has better computational performance than the PSD-DEIM method.

The question remains of how to improve the quality of the approximation. Indeed, unlike the PSD method, increasing the reduced dimension K does not always provide better results. The quality of the approximation could rather be increased by modifying the architecture of the neural networks (increasing the number of layers and the number of neurons per layer), which implies increasing the number of trainable parameters. However, up to our knowledge, there is unfortunately a lack of theoretical results about a systematic way to improve such approximations. Further studies need to be carried out.

Obviously, the results will have to be extended to partial differential equations in three spatial dimension. As convolution layers are used in the auto-encoder, the increase of dimension should not require too large an increase in the size of the neural networks. In

consequence, we expect that the computational gain will be even larger for such dynamics. Other extensions would be to consider time-dependent reductions as in [15] and adapt the method for the reduction of large Hamiltonian differential equations that do not have spatial structures [15].

Acknowledgements. This research was funded in part by l'Agence Nationale de la Recherche (ANR), project ANR-21-CE46-0014 (Milk).

A Linear reduction

A.1 Proper Symplectic Decomposition

Here we briefly present the Proper Symplectic Decomposition (PSD) [25]. In this approach, we assume that the decoder is a linear operator

$$\mathcal{D}_{\theta_d}(\overline{\mathbf{y}}) = A\overline{\mathbf{y}},$$

with $A \in M_{2N,2K}(\mathbb{R})$. To be consistent with the notations, we keep the subscript θ_d to refer to the decoder parameters, which here are the coefficients of the matrix A. The trial manifold $\widehat{\mathcal{M}}$ is a hyperplane. We further assume that A is symplectic, i.e. satisfies $A^T J_{2N} A = J_{2K}$, and we denote by $\mathrm{Sp}_{2N,2K}(\mathbb{R})$ the set of symplectic matrices. The encoder is thus defined as the symplectic inverse of A:

$$\mathcal{E}_{\theta_e}(\mathbf{y}) = A^+\mathbf{y},$$

with $A^+ = J_{2K}^T A^T J_{2N} \in \mathrm{Sp}_{2K,2N}(\mathbb{R})$ and where $\theta_e = \theta_d$ also denotes the coefficients of the matrix A. Then the projection operator onto $\widehat{\mathcal{M}}$ writes AA^+ and A is determined by solving the minimization problem

$$\min_{A \in \mathrm{Sp}_{2N,2K}(\mathbb{R})} \|Y - AA^+Y\|_F^2,$$

where Y refers to the snapshot matrix:

$$Y = \left[\mathbf{y}_1, \ldots, \mathbf{y}_p\right] \in M_{2N,p}(\mathbb{R}), \tag{22}$$

where $(\mathbf{y}_i)_{i \in [\![1,p]\!]}$ are p values obtained from numerical simulations of the original problem at different times and for different parameters, and $\|X\|_F = \sqrt{\sum_{ij} X_{ij}^2}$ is the Frobenius norm of $X \in M_{2N,p}(\mathbb{R})$.

An approximate solution to this problem can be obtained with the cotangent lift algorithm [25]. First, the snapshot matrix Y is reshaped into

$$\tilde{Y} = \left[\mathbf{q}_1, \ldots, \mathbf{q}_p, \mathbf{p}_1, \ldots, \mathbf{p}_p\right] \in \mathcal{M}_{N,2p}(\mathbb{R}).$$

Its Singular Value Decomposition (SVD) is computed leading to $\tilde{Y} = U\Sigma V^T$, where Σ is the diagonal matrix of the singular values and U (resp. V) the matrix of the left (resp. right) singular vectors. Then, we define $\Phi = U[:, :K]$ composed of the K-th left eigenvectors associated with the K-th largest singular values and we consider the following orthogonal symplectic matrix:

$$A = \begin{pmatrix} \Phi & 0 \\ 0 & \Phi \end{pmatrix}.$$

The reduced model satisfied by the reduced variable $\overline{\mathbf{y}} = A^+\mathbf{y}$ can be obtained by multiplying (1) by A^+ so that

$$\frac{d}{dt}\overline{\mathbf{y}} = A^+ J_{2N}\,\nabla_{\mathbf{y}}\mathcal{H}(\mathbf{y}) = J_{2K}A^T\,\nabla_{\mathbf{y}}\mathcal{H}(\mathbf{y}) \approx J_{2K}\,\nabla_{\overline{\mathbf{y}}}(\mathcal{H}\circ A)\,(\overline{\mathbf{y}})\,,$$

where we use the expression of A^+, the identity $J_{2N}^2 = -\mathrm{Id}$ and the approximation $\mathbf{y} \approx A\overline{\mathbf{y}} = AA^+\mathbf{y}$. The reduced model involves the following reduced Hamiltonian

$$\overline{\mathcal{H}}_{\theta_h} = \mathcal{H}\circ A,$$

where $\theta_h = \theta_e = \theta_d$ still denotes the coefficients of the matrix A, and whose evaluation requires to compute the non-reduced quantity $A\overline{\mathbf{y}} \in \mathbb{R}^N$. To avoid this evaluation in large dimension, approximations of the reduced Hamiltonian can be constructed using the discrete empirical interpolation method (DEIM) [33, 25].

The non-symplectic version of the algorithm is called the Proper Orthogonal Decomposition (POD) [20], where the matrix A is obtained using the K first singular vectors of the SVD of the original snapshot matrix Y defined in (22).

A.2 Symplectic Discrete Empirical Interpolation Method

In this appendix, we shortly present the symplectic version of the Discrete Empirical Interpolation Method (DEIM) proposed in [25]. We suppose that the Hamiltonian function $\mathcal{H}$ can be split into a linear and a non-linear part. Its gradient then reads as

$$\nabla_{\mathbf{y}}\mathcal{H}(\mathbf{y}) = L\mathbf{y} + f_N(\mathbf{y}).$$

where $L \in \mathcal{M}_{2N,2N}(\mathbb{R})$ and f_N is a non-linear function. As in Sec. A.1, we apply a symplectic Galerkin projection with a linear decoder operator $A \in \mathcal{M}_{2N,2K}(\mathbb{R})$ to derive the reduced model

$$\frac{d}{dt}\overline{\mathbf{y}} = J_{2K}A^T\,\nabla_{\mathbf{y}}\mathcal{H}(A\overline{\mathbf{y}}) = J_{2K}(A^TLA)\overline{\mathbf{y}} + J_{2K}A^T f_N(A\overline{\mathbf{y}})$$

While the matrix (A^TLA) can be pre-computed, the non-linear part cannot be simplified and the computational cost still depends on the initial dimension N. The DEIM strategy consists into approximating this term by using m components of f_N only. The selected components are denoted $\beta_1, \ldots, \beta_m \in [\![1, 2N]\!]$ and the selection matrix is $P = (\mathbf{e}_{\beta_1}, \ldots, \mathbf{e}_{\beta_m}) \in \mathcal{M}_{2N,m}(\mathbb{R})$ where $\mathbf{e}_{\beta_i}$ is the β_i-th element of the canonical basis. The approximation is then done as follows: first consider $\Psi \in \mathcal{M}_{2N,m}(\mathbb{R})$, obtained after performing a SVD of samples of $f_N(\mathbf{y})$ (as presented in Section A) and retaining only the m-th largest modes. Then, the indices $\beta_1, \cdots, \beta_m$ are selected with a greedy algorithm applied on Ψ and described in [25]. Next, we use the following approximation

$$f_N(\mathbf{y}) \approx \Psi\left[\left(P^T\Psi\right)^{-1}P^T f_N(\mathbf{y})\right].$$

This approximation is chosen so that it belongs to the range of Ψ and the m selected components exactly coincide with those of $f_N(\mathbf{y})$, which can be checked after multiplication by P^T. The matrix $(P^T\Psi)$ is assumed to be invertible. Hence, the reduced model reads

$$\frac{d}{dt}\overline{\mathbf{y}}(t) = J_{2K}(A^TLA)\overline{\mathbf{y}} + J_{2K}\left(A^T\Psi\left(P^T\Psi\right)^{-1}\right)g_m(\overline{\mathbf{y}})$$

with $g_m(\overline{\mathbf{y}}) = P^T f_N(A\overline{\mathbf{y}})$. The matrix $(A^T\Psi\left(P^T\Psi\right)^{-1})$ can be precomputed. If each component of f_N depends only on a few number of components of $A\overline{\mathbf{y}}$, then the evaluation complexity of g_m becomes a function of m instead of N.

References

[1] Babak Maboudi Afkham and Jan S. Hesthaven. Structure preserving model reduction of parametric hamiltonian systems. *SIAM Journal on Scientific Computing*, 39(6):A2616–A2644, January 2017.

[2] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[3] Y. Bengio, O. Delalleau, N. Le Roux, J.-F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links spectral embedding and kernel PCA. *Neural computation*, 16(10):2197–2219, 2004.

[4] S. Bhattacharjee and K. Matous. A nonlinear manifold-based reduced order model for multiscale analysis of heterogeneous hyperelastic materials. *Journal of Computational Physics*, 313, 2016.

[5] P. Buchfink, S. Glas, and B. Haasdonk. Symplectic model reduction of Hamiltonian systems on nonlinear manifolds and approximation with weakly symplectic autoencoder. *SIAM Journal on Scientific Computing*, 45(2):A289–A311, 2023.

[6] S. Chaturantabut and D.C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[7] Xiaoli Chen, Beatrice W. Soh, Zi-En Ooi, Eleonore Vissol-Gaudin, Haijun Yu, Kostya S. Novoselov, Kedar Hippalgaonkar, and Qianxiao Li. Constructing custom thermodynamics using deep learning, 2023.

[8] R.R. Coifman and S. Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006. Special Issue: Diffusion Maps and Wavelets.

[9] William Cruz, José Martínez, and Marcos Raydan. Spectral residual method without gradient information for solving large-scale nonlinear systems of equations. *Math. Comput.*, 75:1429–1448, 07 2006.

[10] Moritz Flaschel, Siddhant Kumar, and Laura De Lorenzis. Automated discovery of generalized standard material models with euclid. *Computer Methods in Applied Mechanics and Engineering*, 405:115867, February 2023.

[11] S. Fresca, L. Dede', and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *J. Sci. Comput.*, 87:1–36, 2021.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[13] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

[14] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration, Structure-preserving algorithms for ordinary differential equations*, volume 31. Springer-Verlag Berlin, 2006.

[15] J. Hesthaven, C. Pagliantini, and N. Ripamonti. Adaptive symplectic model order reduction of parametric particle-based Vlasov–Poisson equation. *Mathematics of Computation*, 2023.

[16] J.S. Hesthaven, C. Pagliantini, and G. Rozza. Reduced basis methods for time-dependent problems. *Acta Numerica*, 31:265–345, 2022.

[17] Y. Kim, Y. Choi, D. Widemann, and T. Zohdi. A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder. *Journal of Computational Physics*, 451:110841, 2022.

[18] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[19] K. Lee and K.T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020.

[20] Y.C. Liang, H.P. Lee, S.P. Lim, W.Z. Lin, K.H. Lee, and C.G. Wu. Proper orthogonal decomposition and its applications - part i: Theory. *Journal of Sound and Vibration*, 252:527–544, 05 2002.

[21] R. Maulik, B. Lusch, and P. Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids*, 33(3):037106, 2021.

[22] C. Pagliantini. Dynamical reduced basis methods for hamiltonian systems. *Numerische Mathematik*, 148(2):409–448, 2021.

[23] Cecilia Pagliantini and Federico Vismara. Fully adaptive structure-preserving hyper-reduction of parametric hamiltonian systems, 2023.

[24] Cecilia Pagliantini and Federico Vismara. Gradient-preserving hyper-reduction of nonlinear dynamical systems via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 45(5):A2725–A2754, October 2023.

[25] L. Peng and K. Mohseni. Symplectic model reduction of Hamiltonian systems. *SIAM Journal on Scientific Computing*, 38(1):A1–A27, 2016.

[26] F. Romor, G. Stabile, and G. Rozza. Non-linear manifold reduced-order models with convolutional autoencoders and reduced over-collocation method. *J. Sci. Comput.*, 94(3):74, 2023.

[27] B.E. Sonday, A. Singer, C.W. Gear, and I.G. Kevrekidis. Manifold learning techniques and model reduction applied to dissipative pdes. *arXiv e-prints*, pages arXiv–1011, 2010.

[28] J. B. Tenenbaum, V. Silva, and J. C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[29] N. Thuerey, P. Holl, M. Mueller, P. Schnell, F. Trost, and K. Um. Physics-based deep learning. *arXiv preprint arXiv:2109.05237*, 2021.

[30] T.M. Tyranowski and M. Kraus. Symplectic model reduction methods for the Vlasov equation. *Contributions to Plasma Physics*, 63(5-6):e202200046, 2023.

[31] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[32] Q. Wang, N. Ripamonti, and J.S. Hesthaven. Recurrent neural network closure of parametric POD-Galerkin reduced-order models based on the Mori-Zwanzig formalism. *Journal of Computational Physics*, 410:109402, 2020.

[33] Z. Wang. Structure-Preserving Galerkin POD-DEIM Reduced-Order Modeling of Hamiltonian Systems, 2021.

[34] Haijun Yu, Xinyuan Tian, Weinan E, and Qianxiao Li. Onsagernet: Learning stable and interpretable dynamics using a generalized onsager principle. *Physical Review Fluids*, 6(11), November 2021.

[35] Zhen Zhang, Yeonjong Shin, and George Em Karniadakis. Gfinns: Generic formalism informed neural networks for deterministic and stochastic dynamical systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 380(2229), June 2022.