Deep learning of thermodynamics-aware reduced-order models from data☆

Quercus Hernandez[a], Alberto Badías[a], David González[a], Francisco Chinesta[b], Elías Cueto[a,*]

[a] *Aragon Institute of Engineering Research (I3A), Universidad de Zaragoza, Maria de Luna 3, E-50018 Zaragoza, Spain*
[b] *ESI Chair and PIMM Lab, ENSAM ParisTech, 155 Boulevard de l'Hôpital, 75013 Paris, France*

Abstract

We present an algorithm to learn the relevant latent variables of a large-scale discretized physical system and predict its time evolution using thermodynamically-consistent deep neural networks. Our method relies on sparse autoencoders, which reduce the dimensionality of the full order model to a set of sparse latent variables with no prior knowledge of the coded space dimensionality. Then, a second neural network is trained to learn the metriplectic structure of those reduced physical variables and predict its time evolution with a so-called structure-preserving neural network. This data-based integrator is guaranteed to conserve the total energy of the system and the entropy inequality, and can be applied to both conservative and dissipative systems.

The integrated paths can then be decoded to the original full-dimensional manifold and be compared to the ground truth solution. This method is tested with two examples applied to fluid and solid mechanics.

Keywords: Deep learning; Sparse autoencoders; Thermodynamics; Model reduction; Structure preserving

1. Introduction

Physical simulation has become an indispensable tool for engineers to recreate the operative conditions of a mechanical system and make decisions about its optimal design, ranging from composite building structures to complex fluid–solid interaction CFD simulations. These phenomena are often discretized in fine meshes resulting in millions of degrees of freedom, which are computationally expensive to handle, but their solutions are contained in lower-dimensional spaces. This is the so-called manifold hypothesis [1].

☆ This project has been partially funded by the ESI Group through the ESI Chair at ENSAM Arts et Metiers Institute of Technology, and through the project 2019-0060 "Simulated Reality" at the University of Zaragoza. The support of the Spanish Ministry of Economy and Competitiveness through grant number CICYT-DPI2017-85139-C2-1-R and by the Regional Government of Aragon and the European Social Fund, are also gratefully acknowledged.

* Corresponding author.
E-mail addresses: quercus@unizar.es (Q. Hernandez), abadias@unizar.es (A. Badías), gonzal@unizar.es (D. González), francisco.chinesta@ec-nantes.fr (F. Chinesta), ecueto@unizar.es (E. Cueto).

https://doi.org/10.1016/j.cma.2021.113763

Thus, several methods try to overcome this inconvenience by reducing the dimensionality of the problem, computing a suitable reduced basis and projecting the full order model on it. The very first projection-based model order reduction (MOR) methods relied on linear transformations with some additional constraints, such as Proper Orthogonal Decomposition (POD) [2,3], Reduced-Basis technique [4] or Galerkin projection [5,6]. However, these linear mappings are only locally accurate, so they fail in modeling more complex nonlinear phenomena and sometimes require prior information about the governing equations of the problem physics.

In order to overcome these limitations, several techniques have been developed in the machine learning framework that provide nonlinear mappings, such as Locally Linear Embedding [7], Topological Data Analysis [8], kernel Principal Component Analysis [9] or Neural Networks, by means of Autoencoders [10]. In the present work we focus on this last method, which has proven to learn highly nonlinear manifolds in a wide variety of fields such as physics [11], chemistry [12], mechanics [13] or computational imaging [14]. Autoencoders used as a model reduction tool, project the original data (assumed to form a high-order manifold) to a reduced manifold. However, most of the current works rely on prior knowledge, or parametric search, of the optimal latent dimensionality of the problem. Here lies one of the key concepts of our method, which is able to learn a sparse representation of the latent space within a given reconstruction error bound.

These same machine learning tools can be used to learn the underlying physics of the problem. Very often, neural networks have been criticized for constituting a sort of black box, whose results – besides needing a big amount of data – are unpredictable. Therefore, adding previous knowledge on the physics of the problem helps to ensure the physical meaning of the results, while keeping to a minimum the amount of data needed for successful predictions. Several authors have developed frameworks for solving nonlinear PDEs with accurate results [15,16]. However, they require information about the system nature and governing equations, which are usually unknown. Some methods bypass this problem by learning energetic invariants of the system [17–19] or exploiting the symplectic structure of the problem [20–22], reporting promising and interpretable results for Hamiltonian dynamics. Nonetheless, few methods are valid for dissipative effects such as friction, heat dissipation or plasticity, which are usually found in real life engineering problems.

The authors already presented a methodology to learn the time evolution of general physical systems by enforcing the GENERIC (an acronym of General Equation for the Non-Equilibrium Reversible–Irreversible Coupling) structure of the problem [23,24], with the so-called Structure-Preserving Neural Networks [25]. This networks result in a thermodynamically-consistent integrator that is valid for both conservative (Hamiltonian) and dissipative systems. However, these networks operate only on full-order descriptions of the system, resulting in a costly procedure with limited engineering applicability for systems of tens of thousands to millions of degrees of freedom. The aim of this work is to apply this algorithm to more complex dynamical systems, combined with the nonlinear model order reduction power of autoencoders. The proposed methodology is a completely general method that is able to unveil the true effective dimensionality of the sampled data with no user intervention, and to construct from it a reduced-order integrator of the dynamics of the system with no previous knowledge on the nature of the system at hand. The resulting full-order reconstructions of the dynamics are guaranteed to conserve energy and dissipate entropy, as dictated by the laws of thermodynamics.

The outline of the paper is as follows. A brief description of the problem setup is presented in Section 2. Next, in Section 3, the methodology is presented of both the autoencoder model order reduction and the GENERIC formalism used to solve the stated problem. Two validation examples are reported: a Couette flow in a viscolastic fluid (Section 4) and a rolling hyperelastic tire (Section 5). The paper is then completed with a discussion in Section 6.

2. Problem statement

In this work we exploit the so-called "dynamical systems equivalence" of machine learning [26]. Consider a system whose governing variables will be hereafter denoted by $z \in \mathcal{M} \subseteq \mathbb{R}^D$, with $\mathcal{M}$ the state space of these variables, which is assumed to have the structure of a differentiable manifold in $\mathbb{R}^D$. The full-order model of a given physical phenomenon can be expressed as a system of differential equations encoding the time evolution of a set of governing variables z,

$$\dot{z} = \frac{dz}{dt} = F(z,t), \;\; t \in \mathcal{I} = (0,T], \;\; z(0) = z_0, \tag{1}$$

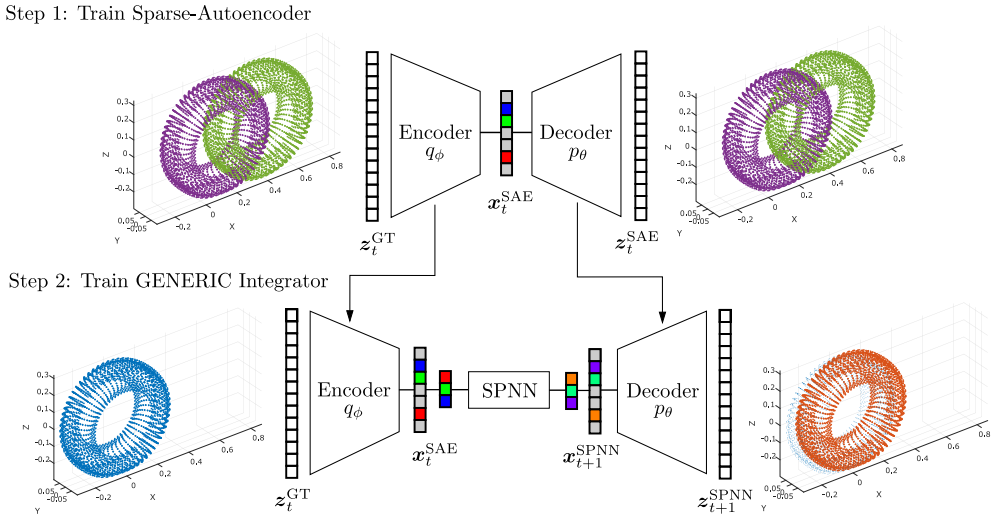

Fig. 1. Block diagram of the proposed algorithm. Snapshots of the rolling tire problem, see Section 5, have been included for illustration purposes. Step 1: A sparse autoencoder (SAE) is trained with time snapshots of a ground truth physical simulation, in order to learn an encoded representation of the full-order space. Step 2: A structure-preserving neural network (SPNN) is trained to integrate the full time evolution of the latent variables, consistently with the GENERIC structure of the underlying physics of the problem.

where $\boldsymbol{q}$ and t refer to the space and time coordinates within a domain with $n = 2, 3$ dimensions t refers to the time coordinate. The objective of the learning procedure is, therefore, to find $F(\boldsymbol{z}, t)$, the function that gives, after a prescribed time horizon T, the flow map $\boldsymbol{z}_0 \to \boldsymbol{z}(\boldsymbol{z}_0, T)$.

The dimensionality reduction technique, in addition, seeks a simplified representation of the full-order state vector $\boldsymbol{z}$ through a set of latent, reduced variables $\boldsymbol{x} \in \mathcal{N} \subseteq \mathbb{R}^d$ contained in a trial manifold with reduced dimensionality, lower than the original space $\mathcal{M}$. The mapping between both spaces can be denoted by $\phi : \mathcal{M} \subseteq \mathbb{R}^D \to \mathbb{R}^d$ with $d \ll D$. Similarly, the inverse mapping ϕ^{-1} allows to undo the transformation, returning to the original full-order space.

The goal of this paper is to find the convenient mapping ϕ for a dynamical system governed by Eq. (1) in order to efficiently learn the underlying physics in the reduced space $\mathcal{N}$ and then predict its time evolution. The solution is forced to fulfill the basic thermodynamic requirements of energy conservation and entropy inequality restrictions via the GENERIC formalism.

3. Methodology

The proposed algorithm divides the problem in two main steps, sketched in Fig. 1. First, the full order model is encoded to a reduced manifold with a nonlinear mapping via an autoencoder [13]. This autoencoder learns a latent representation of a state vector of a physical system, in order to handle a wide amount of simulation data in a compact form. The full order simulation data presented in this work is generated in silico, but the same procedure could be applied to measured data in a real physical system.

Secondly, a structure-preserving neural network [25] is trained with several snapshots of the physical simulation. This net works as an integrator which predicts the time evolution of the system within the GENERIC formalism [23,24]. This integration scheme preserves the thermodynamic structure of the latent variables in the reduced manifold [27] ensuring, as we said, the basic laws of thermodynamics of energy conservation and entropy inequality. These integrated variables are then projected back to the original manifold of the full order model with the decoder.

3.1. Model reduction with sparse-autoencoders

An autoencoder is a type of artificial neural network which reduces the dimensionality of an input into a coded version, which ideally contains the same information, by learning the identity function. It is composed by an encoder

q_ϕ, which maps high-dimensional data $\boldsymbol{z} \in \mathbb{R}^D$ onto a low-dimensional code $\boldsymbol{x} \in \mathbb{R}^d$ with $d \ll D$, and a decoder p_θ, which applies the inverse mapping back to the original full-order manifold,

$$q_\phi : \mathbb{R}^D \to \mathbb{R}^d, \quad \boldsymbol{x} = q_\phi(\boldsymbol{z}), \tag{2}$$

$$p_\theta : \mathbb{R}^d \to \mathbb{R}^D, \quad \hat{\boldsymbol{z}} = p_\theta(\boldsymbol{x}). \tag{3}$$

The vector $\boldsymbol{z}$ is often referred as the full order vector, whereas its coded vector $\boldsymbol{x}$ is referred as *code* or *latent variable*. In this work, we use a bottleneck architecture composed by several stacked fully-connected hidden layers N_h in both the encoder and decoder. Each layer is modeled as a multilayer perceptron (MLP), which is mathematically defined as

$$\boldsymbol{x}^{[l]} = \sigma(\boldsymbol{w}^{[l]}\boldsymbol{x}^{[l-1]} + \boldsymbol{b}^{[l]}), \tag{4}$$

where l is the index of the current layer, $\boldsymbol{x}^{[l-1]}$ and $\boldsymbol{x}^{[l]}$ are the layer input and output vector respectively, $\boldsymbol{w}^{[l]}$ is the weight matrix, $\boldsymbol{b}^{[l]}$ is the bias vector and σ is the activation function. The activation functions are usually nonlinear, allowing the encoding and decoding of complex nonlinear phenomena by stacking several layers together.

The latent vector dimensionality d in Eq. (2) is, a priori, unknown. Thus, we add a sparsity condition to the bottleneck to force the autoencoder to learn the number of latent variables needed to encode the necessary information of the full order model. Even if the latent layer has a fixed number of units N_d, the sparsity penalizer is able to find (at least a good approximation to) the intrinsic dimensionality of the low-dimensional data $\boldsymbol{x}$. Here, no prior on the reduced dimension is needed. Thus, further in this text, the autoencoder with sparsity regularization is referred as sparse autoencoder (SAE).

The loss function for our neural network is composed of two different terms:

- **Reconstruction loss**: This term minimizes the difference between the ground truth vector $\boldsymbol{z}_n^{\mathrm{GT}}$ and the autoencoder reconstruction $\boldsymbol{z}_n^{\mathrm{SAE}}$ in the snapshot n. This enforces the network to learn the identity function,

$$\mathcal{L}_n^{\mathrm{rec}} = \left(\boldsymbol{z}_n^{\mathrm{GT}} - \boldsymbol{z}_n^{\mathrm{SAE}}\right)^\top \left(\boldsymbol{z}^{\mathrm{GT}} - \boldsymbol{z}_n^{\mathrm{SAE}}\right). \tag{5}$$

- **Regularization**: In order to impose the sparsity of the latent vector, several regularizers can be used [28]. Due to the continuous nature of the physical data, it is found more convenient to use L1-norm penalizer, which enforces hard zeros in the latent variables that are not relevant,

$$\mathcal{L}_n^{\mathrm{reg}} = \sum_{i=1}^{N_d} |\boldsymbol{x}_i^{\mathrm{SAE}}|. \tag{6}$$

The temporal snapshots of the physical simulations are split in a partition of train snapshots ($N_{\mathrm{train}} = 80\%$ of the database snapshots) and test snapshots ($N_{\mathrm{test}} = 20\%$ of the database snapshots) so that $N_T = N_{\mathrm{train}} + N_{\mathrm{test}}$. The total loss function is computed as the mean squared error (MSE) of the data reconstruction loss and the sparsity regularization term for the train snapshots (N_{train}). The sparsity loss is multiplied by a regularization hyperparameter λ_r^{SAE}, which is responsible for the trade-off between the reconstruction fidelity of the autoencoder and the sparsity of the latent vector $\boldsymbol{x}$,

$$\mathcal{L}^{\mathrm{SAE}} = \frac{1}{N_{\mathrm{train}}} \sum_{n=0}^{N_{\mathrm{train}}} (\mathcal{L}_n^{\mathrm{rec}} + \lambda_r^{\mathrm{SAE}} \mathcal{L}_n^{\mathrm{reg}}). \tag{7}$$

The backpropagation algorithm [29] is then used to calculate the gradient of the loss function for each encoder and decoder parameters ϕ and θ (weight and bias vectors of both blocks), which are updated with the gradient descent technique [30]. An overview of the training algorithm of the SAE is sketched in Algorithm 1.

The SAE performance is then evaluated with the mean squared error (MSE) of the test snapshots (N_{test}) for each state variable (z),

$$\mathrm{MSE}^{\mathrm{SAE}}(z) = \frac{1}{N_{\mathrm{test}}} \sum_{n=0}^{N_{\mathrm{test}}} \varepsilon_n^2 = \frac{1}{N_{\mathrm{test}}} \sum_{n=0}^{N_{\mathrm{test}}} \left(z_n^{\mathrm{GT}} - z_n^{\mathrm{SAE}}\right)^2, \tag{8}$$

tested with two different databases of nonlinear systems. A pseudocode of the testing process of the SAE is shown in Algorithm 2.

Algorithm 1 Pseudocode for the training algorithm of the Sparse-Autoencoder.

Load database: z^{GT} (train partition);
Define network architecture: $N_{\mathrm{in}}^{\mathrm{SAE}} = N_{\mathrm{out}}^{\mathrm{SAE}} = D$, N_h^{SAE}, N_d^{SAE}, σ_j^{SAE};
Define hyperparameters: l_r^{SAE}, λ_r^{SAE};
Initialize $w_{i,j}^{\mathrm{SAE}}$, b_j^{SAE};
for each epoch **do**
  Initialize loss function: $C = 0$;
  for each train snapshot **do**
    Encoder: $\boldsymbol{x}_n^{\mathrm{SAE}} = q_\phi(\boldsymbol{z}_n^{\mathrm{GT}})$; ▷ Eq. (2)
    Decoder: $\boldsymbol{z}_n^{\mathrm{SAE}} = p_\theta(\boldsymbol{x}_n^{\mathrm{SAE}})$; ▷ Eq. (3)
    Loss function: $C \leftarrow C + \mathcal{L}_n^{\mathrm{rec}} + \lambda_r^{\mathrm{SAE}} \mathcal{L}_n^{\mathrm{reg}}$; ▷ Eqs. (5), (6)
  end for
  MSE loss function: $\mathcal{L}^{\mathrm{SAE}} \leftarrow \frac{C}{N_{\mathrm{train}}}$ ▷ Eq. (7)
  Backward propagation;
  Optimizer step;
end for

Algorithm 2 Pseudocode for the test algorithm of the Sparse-Autoencoder.

Load database: z^{GT} (test partition);
Load network parameters;
for each test snapshot **do**
  Encoder: $\boldsymbol{x}_n^{\mathrm{SAE}} = q_\phi(\boldsymbol{z}_n^{\mathrm{GT}})$; ▷ Eq. (2)
  Decoder: $\boldsymbol{z}_n^{\mathrm{SAE}} = p_\theta(\boldsymbol{x}_n^{\mathrm{SAE}})$; ▷ Eq. (3)
  Compute Squared Error: $\varepsilon_n^2 = \left(\boldsymbol{z}_n^{\mathrm{GT}} - \boldsymbol{z}_n^{\mathrm{SAE}}\right)^2$; ▷ Eq. (8)
end for
Compute $\mathrm{MSE}^{\mathrm{SAE}}$ $(\boldsymbol{z})$; ▷ Eq. (8)

Once the problem is reduced to a lower-dimensional manifold, a second neural network can be trained to learn the underlying physics of the problem, being able to integrate the whole simulation trajectory with thermodynamic consistency. This is achieved by using a structure-preserving neural network [25], and is explained in the next section.

3.2. The GENERIC formalism

There are different forms of enforcing physical meaning to the results of a particular neural network. One could be the enforcement of the structure of a particular partial differential equation, as in [15]. This is known as adding an *inductive bias* [31]. An inductive bias is a way to enforce an algorithm to prioritize one solution to another. In our case, we try to guarantee as much as possible the physical meaning of the solution, but without enforcing any particular physical law, which may be even unknown. We do this by adding a regularization term to our neural network. This regularization will enforce the fulfillment of the first and second laws of thermodynamics.

A Structure-Preserving Neural Network [25] (from now on, SPNN) is a type of artificial neural network that learns the metriplectic structure of a general dynamical system [32], with both conservative and dissipative phenomena, by imposing a GENERIC structure [23,24].

In this approach, the reversible or conservative contribution is assumed to be of Hamiltonian form, requiring an energy function $E(\boldsymbol{x})$ and a Poisson bracket $\{\boldsymbol{x}, E\}$ acting on an arbitrary state vector $\boldsymbol{x}$. Similarly, the remaining irreversible contribution to the energetic balance of the system is generated by the nonequilibrium entropy $S(\boldsymbol{x})$ with an irreversible or friction bracket $[\boldsymbol{x}, S]$.

The GENERIC formulation of time evolution for nonequilibrium systems, described by a set of $\boldsymbol{x}$ state variables required for its complete description, is given by

$$\frac{d\boldsymbol{x}}{dt} = \{\boldsymbol{x}, E\} + [\boldsymbol{x}, S]. \tag{9}$$

For practical use, it is convenient to reformulate the brackets in two algebraic or differential operators

$$\boldsymbol{L} : T^*\mathcal{M} \to T\mathcal{M}, \quad \boldsymbol{M} : T^*\mathcal{M} \to T\mathcal{M},$$

where $T^*\mathcal{M}$ and $T\mathcal{M}$ represent, respectively, the cotangent and tangent bundles of $\mathcal{M}$. These operators inherit the mathematical properties of the original bracket formulation. The operator $\boldsymbol{L}(\boldsymbol{x})$ represents the Poisson bracket and is required to be skew-symmetric (a cosympletic matrix). Similarly, the friction matrix $\boldsymbol{M}(\boldsymbol{x})$ accounts for the irreversible part of the system and is symmetric and positive semi-definite. Then, the brackets of Eq. (9) can be replaced by their homologous matrix operators

$$\{\boldsymbol{A}, \boldsymbol{B}\} = \frac{\partial \boldsymbol{A}}{\partial \boldsymbol{x}} \boldsymbol{L} \frac{\partial \boldsymbol{B}}{\partial \boldsymbol{x}}, \qquad [\boldsymbol{A}, \boldsymbol{B}] = \frac{\partial \boldsymbol{A}}{\partial \boldsymbol{x}} \boldsymbol{M} \frac{\partial \boldsymbol{B}}{\partial \boldsymbol{x}},$$

resulting in the time-evolution equation for the state variables $\boldsymbol{x}$,

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{L} \frac{\partial E}{\partial \boldsymbol{x}} + \boldsymbol{M} \frac{\partial S}{\partial \boldsymbol{x}}. \tag{10}$$

This equation is completed with two degeneracy conditions

$$\{S, \boldsymbol{x}\} = \boldsymbol{0}, \quad [E, \boldsymbol{x}] = \boldsymbol{0}.$$

The first one states that the entropy is a degenerate functional of the Poisson bracket, showing the reversible nature of the Hamiltonian contribution to the dynamics. The second expression states the conservation of the total energy of the system with a degenerate condition of the energy with respect to the friction bracket. These restrictions can be reformulated in a matrix form in terms of the $\boldsymbol{L}$ and $\boldsymbol{M}$ operators, resulting in the following degeneracy restrictions:

$$\boldsymbol{L} \frac{\partial S}{\partial \boldsymbol{x}} = \boldsymbol{M} \frac{\partial E}{\partial \boldsymbol{x}} = 0. \tag{11}$$

The degeneracy conditions, in addition to the non-negativeness of the irreversible bracket, guarantees the first (energy conservation) and second (entropy inequality) laws of thermodynamics,

$$\frac{dE}{dt} = \{E, E\} = 0, \quad \frac{dS}{dt} = [S, S] \geq 0. \tag{12}$$

3.3. Structure-preserving neural networks

Based on this theoretical formalism, a structure-preserving neural network imposes the GENERIC thermodynamically-sound structure in discretized approach,

$$\frac{\boldsymbol{x}_{n+1} - \boldsymbol{x}_n}{\Delta t} = \mathsf{L}_n \cdot \frac{\mathsf{D}E_n}{\mathsf{D}\boldsymbol{x}_n} + \mathsf{M}_n \cdot \frac{\mathsf{D}S_n}{\mathsf{D}\boldsymbol{x}_n}, \tag{13}$$

where the time derivative is substituted by a forward Euler scheme in time increments Δt, where $\boldsymbol{x}_{n+1} = \boldsymbol{x}_{t+\Delta t}$. L_n and M_n are the discretized version of the Poisson and friction operators. $\frac{\mathsf{D}E_n}{\mathsf{D}\boldsymbol{x}_n}$ and $\frac{\mathsf{D}S_n}{\mathsf{D}\boldsymbol{x}_n}$ represent the discrete gradients of the energy and the entropy.

Manipulating algebraically Eq. (13) and including the degeneracy conditions of Eq. (11), the proposed integration scheme for predicting the dynamics of a physical system is the following

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta t \left(\mathsf{L}_n \cdot \frac{\mathsf{D}E_n}{\mathsf{D}\boldsymbol{x}_n} + \mathsf{M}_n \cdot \frac{\mathsf{D}S_n}{\mathsf{D}\boldsymbol{x}_n} \right) \tag{14}$$

subject to:

$$\mathsf{L}_n \cdot \frac{\mathsf{D}S_n}{\mathsf{D}\boldsymbol{x}_n} = 0, \quad \mathsf{M}_n \cdot \frac{\mathsf{D}E_n}{\mathsf{D}\boldsymbol{x}_n} = 0, \tag{15}$$

ensuring the thermodynamical consistency of the resulting model. From now on, the energy and entropy gradients will be shortened as $\frac{\mathrm{D}E_n}{\mathrm{D}\boldsymbol{x}_n} \equiv \mathsf{DE}_n$ and $\frac{\mathrm{D}S_n}{\mathrm{D}\boldsymbol{x}_n} \equiv \mathsf{DS}_n$.

Unlike previous work [25], the GENERIC structure is imposed to the reduced order model learnt by the sparse autoencoder, so there is no prior information about the L and M matrices. Instead, the SPNN is forced to automatically learn them on each learning set time step, L_n and M_n, with their respective skew-symmetric and symmetric conditions. Similarly, the energy and entropy gradient, DE_n and DS_n, are computed on each time step and no finite-difference approach is needed.

The structure-preserving neural network uses a feed-forward scheme [33], consisting of several fully-connected layers with no cyclic connections. The input of the neural net is the encoded vector state of a given time step $\boldsymbol{x}_n^{\mathrm{SAE}} = q_\phi(\boldsymbol{x}_n^{\mathrm{GT}})$, and the outputs are the concatenated GENERIC matrices (L_n, M_n) and energy and entropy gradient matrices (DE_n, DS_n). Then, using the GENERIC forward integration scheme in Eq. (13), the reduced state vector at the next time step $\boldsymbol{x}_{n+1}^{\mathrm{SPNN}}$ is obtained.

Following Eq. (14), the input dimension of the SPNN is the same as the dimension of the sparsified latent variables $\boldsymbol{x}_n^{\mathrm{SAE}}$ ($N_{\mathrm{in}}^{\mathrm{SPNN}} = d$). Consequently, the GENERIC matrices L_n and M_n are squared with dimension d^2 each, which can be reduced to $d \cdot (d-1)/2$ and $d \cdot (d+1)/2$ taking into account the skew-symmetric and symmetric elements respectively. Additionally, the matrix M_n is assembled by taking the absolute value of the diagonal elements of the resulting lower triangular matrix and multiplying it by its transpose. By the Cholesky factorization, this ensures that M_n is positive semidefinite. The energy and entropy gradient matrices DE_n and DS_n have the same dimension d as the state vector. The final output dimension of the integrator network is then $N_{\mathrm{out}}^{\mathrm{SPNN}} = d \cdot (d+1)/2 + d \cdot (d-1)/2 + 2 \cdot d = d \cdot (d+2)$.

The loss function for the SPNN is composed of three different terms:

- **Data loss**: The main loss condition is the agreement between the network output and the real data. It is computed as the squared error sum, computed between the predicted state vector $\boldsymbol{x}_{n+1}^{\mathrm{SPNN}}$ and the ground truth solution based on the SAE output $\boldsymbol{x}_{n+1}^{\mathrm{SAE}}$ for each time step,

$$\mathcal{L}_n^{\mathrm{data}} = \|\boldsymbol{x}_{n+1}^{\mathrm{SAE}} - \boldsymbol{x}_{n+1}^{\mathrm{SPNN}}\|_2^2. \tag{16}$$

- **Fulfillment of the degeneracy conditions**: The loss function will also account for the degeneracy conditions, Eq. (15) in order to ensure the thermodynamic consistency of the solution, implemented as the sum of the squared elements of the degeneracy vectors for each time step,

$$\mathcal{L}_n^{\mathrm{degen}} = \|\mathsf{L}_n \cdot \mathsf{DS}_n\|_2^2 + \|\mathsf{M}_n \cdot \mathsf{DE}_n\|_2^2. \tag{17}$$

  This term acts as a regularization of the loss function and, at the same time, is the responsible of ensuring thermodynamic consistency of the integration scheme. This is, in other words, our inductive bias.

- **Regularization**: In order to avoid overfitting, an extra L2 regularization term $\mathcal{L}^{\mathrm{reg}}$ is added to the loss function,

$$\mathcal{L}^{\mathrm{reg}} = \sum_{l}^{L} \sum_{i}^{n^{[l]}} \sum_{j}^{n^{[l+1]}} (w_{i,j}^{[l],\mathrm{SPNN}})^2. \tag{18}$$

The same database split procedure is followed as in the SAE, dividing the complete dataset of N_T snapshots in a partition of train snapshots ($N_{\mathrm{train}} = 80\%$ of the database snapshots) and test snapshots ($N_{\mathrm{test}} = 20\%$ of the database snapshots) so that $N_T = N_{\mathrm{train}} + N_{\mathrm{test}}$. The total loss function is computed as the mean squared error (MSE) of the data loss and degeneracy residual, in addition to the regularization term, for all the training snapshots (N_{train}) of the simulation time T. Both the data loss error and the regularization terms are weighted with two additional hyperparameters $\lambda_d^{\mathrm{SPNN}}$ and $\lambda_r^{\mathrm{SPNN}}$ respectively, which account for their relative influence in the total loss function with respect to the degeneracy constraint,

$$\mathcal{L}^{\mathrm{SPNN}} = \frac{1}{N_{\mathrm{train}}} \sum_{n=0}^{N_{\mathrm{train}}} (\lambda_d^{\mathrm{SPNN}} \mathcal{L}_n^{\mathrm{data}} + \mathcal{L}_n^{\mathrm{degen}}) + \lambda_r^{\mathrm{SPNN}} \mathcal{L}^{\mathrm{reg}}. \tag{19}$$

The usual backpropagation algorithm [29] is then used to calculate the gradient of the loss function for each net parameter (weight and bias vectors), which are updated with the gradient descent technique [30]. The training algorithm is sketched below in Algorithm 3.

Algorithm 3 Pseudocode for the train algorithm of the SPNN.

Load train database: z^{SAE} (train partition), Δt;
Define network architecture: $N_{\text{in}}^{\text{SPNN}} = d$, $N_{\text{out}}^{\text{SPNN}} = d \cdot (d+3)$, N_h^{SPNN}, σ_j^{SPNN};
Define hyperparameters: l_r^{SPNN}, λ_d^{SPNN}, λ_r^{SPNN};
Initialize $w_{i,j}^{\text{SPNN}}$, b_j^{SPNN};
for each epoch **do**
  Initialize loss function: $C = 0$;
  for each train snapshot **do**
    Encoder: $\boldsymbol{x}_n^{\text{SAE}} = q_\phi(\boldsymbol{z}_n^{\text{GT}})$; ▷ Eq. (3)
    Forward propagation: $[\mathsf{L}_n, \mathsf{M}_n, \mathsf{DE}_n, \mathsf{DS}_n] \leftarrow \text{SPNN}(\boldsymbol{x}_n^{\text{SAE}})$; ▷ Eq. (4)
    Time step integration: $\boldsymbol{x}_{n+1}^{\text{SPNN}} \leftarrow \boldsymbol{x}_n^{\text{SAE}} + \Delta t\ (\mathsf{L}_n \cdot \mathsf{DE}_n + \mathsf{M}_n \cdot \mathsf{DE}_n)$; ▷ Eq. (13)
    Update loss function: $C \leftarrow C + \lambda_d^{\text{SPNN}} \mathcal{L}_n^{\text{data}} + \mathcal{L}_n^{\text{degen}}$; ▷ Eqs. (16), (17)
  end for
  MSE loss function: $\mathcal{L}^{\text{SPNN}} \leftarrow \frac{C}{N_{\text{train}}} + \lambda_r^{\text{SPNN}} \mathcal{L}^{\text{reg}}$ ▷ Eqs. (18), (19)
  Backward propagation;
  Optimizer step;
end for

The testing consists of the full time integration of the initial state vector z_0 at $t = 0$ along the complete simulation time interval $\mathcal{I} = (0, T]$, reproducing the problem statement established in Eq. (1). Thus, the net performance is evaluated with the mean squared error (MSE) of the SPNN state variable predictions and the ground truth solution for the complete set of snapshots N_T,

$$\text{MSE}^{\text{SPNN}}(z) = \frac{1}{N_T} \sum_{n=0}^{N_T} \varepsilon_n^2 = \frac{1}{N_T} \sum_{n=0}^{N_T} \left(\boldsymbol{x}_n^{\text{GT}} - \boldsymbol{z}_n^{\text{SPNN}}\right)^2, \tag{20}$$

tested for the same nonlinear systems trained in the SAE training phase. A pseudocode of the testing process of the SPNN is shown in Algorithm 4.

Algorithm 4 Pseudocode for the test algorithm of the complete integration scheme of the SPNN.

Load database: z^{GT}, Δt;
Load network parameters;
Initialize state vector: $\boldsymbol{z}_0^{\text{SAE}} = \boldsymbol{z}_0^{\text{SPNN}} = \boldsymbol{z}_0^{\text{GT}}$;
Initialize encoded state vector: $\boldsymbol{x}_0^{\text{SAE}} = \boldsymbol{x}_0^{\text{SPNN}} = q_\phi(\boldsymbol{z}_0^{\text{GT}})$; ▷ Eq. (2)
for each snapshot **do**
  Forward propagation: $[\mathsf{L}_n, \mathsf{M}_n, \mathsf{DE}_n, \mathsf{DS}_n] \leftarrow \text{SPNN}(\boldsymbol{x}_n^{\text{SPNN}})$; ▷ Eq. (4)
  Time step integration: $\boldsymbol{x}_{n+1}^{\text{SPNN}} \leftarrow \boldsymbol{x}_n^{\text{SPNN}} + \Delta t\ (\mathsf{L}_n \cdot \mathsf{DE}_n + \mathsf{M}_n \cdot \mathsf{DE}_n)$; ▷ Eq. (13)
  Update state vector: $\boldsymbol{x}_n^{\text{SPNN}} \leftarrow \boldsymbol{x}_{n+1}^{\text{SPNN}}$;
  Update snapshot: $n \leftarrow n + 1$;
  Decoder: $\boldsymbol{z}_{n+1}^{\text{SPNN}} = p_\theta(\boldsymbol{x}_{n+1}^{\text{SPNN}})$; ▷ Eq. (3)
  Compute Squared Error: $\varepsilon_{n+1}^2 = \left(\boldsymbol{z}_{n+1}^{\text{GT}} - \boldsymbol{z}_{n+1}^{\text{SPNN}}\right)^2$; ▷ Eq. (20)
end for
Compute $\text{MSE}^{\text{SPNN}}(z)$; ▷ Eq. (20)

The SPNN is compared on each example with a baseline unconstrained neural network which directly predicts the time evolution of the latent vector x_{t+1} from the current snapshot x_t, with a similar training and integration scheme as depicted in Algorithms 3 and 4.

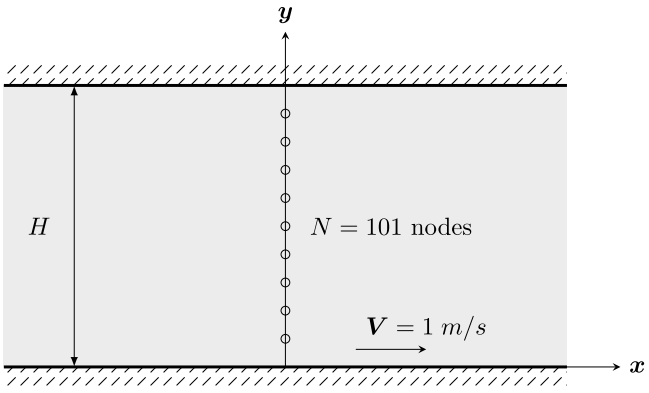

Fig. 2. Couette flow in an Olroyd-B fluid. Horizontal position, velocity, internal energy and conformation tensor shear component are tracked for the total of 100 nodes (excluded the $y = H$ node).

4. Validation examples: Couette flow of an Oldroyd-B fluid

4.1. Description

The first example is a shear (Couette) flow of an Oldroyd-B fluid model. This is a constitutive model for viscoelastic fluids. It arises from the consideration of linear elastic dumbbells as a proxy representation of polymeric chains immersed in a solvent.

The problem is solved by the CONNFFESSIT technique [34], based on the Fokker–Planck equation [35]. This equation is solved by converting it in its corresponding Itô stochastic differential equation,

$$\begin{aligned} dr_x &= \left(\frac{\partial u}{\partial y} r_y - \frac{1}{2\mathrm{We}} r_x\right) dt + \frac{1}{\sqrt{\mathrm{We}}} dV_t, \\ dr_y &= -\frac{1}{2\mathrm{We}} r_y dt + \frac{1}{\sqrt{\mathrm{We}}} dW_t, \end{aligned} \tag{21}$$

where $\boldsymbol{r} = [r_x,\ r_y]^\top$, $r_x = r_x(y, t)$ and assuming a Couette flow so that $r_y = r_y(t)$ depends only on time, "We" stands for the Weissenberg number and V_t, W_t are two independent one-dimensional Brownian motions. This equation is solved via Monte Carlo techniques, by replacing the mathematical expectation by the empirical mean.

The model relies on the microscopic description of the state of the dumbbells. Thus, it is particularly useful to base the microscopic description on the evolution of the conformation tensor $\boldsymbol{c} = \langle \boldsymbol{r}\boldsymbol{r} \rangle$, this is, the second moment of the dumbbell end-to-end distance distribution function. This tensor is in general not experimentally measurable and plays the role of an internal variable. The expected xy stress component tensor will be given by

$$\tau = \frac{\epsilon}{\mathrm{We}} \frac{1}{K} \sum_{k=1}^{K} r_x r_y,$$

where K is the number of simulated dumbbells and $\epsilon = \frac{\nu_p}{\nu_p}$ is the ratio of the polymer to solvent viscosities.

The state variables chosen for the full order model are the position of the fluid on each node of the mesh $\boldsymbol{q}$, see Fig. 2, its velocity $\boldsymbol{v}$ in the x direction, internal energy e and the conformation tensor shear component τ for all the nodes of the mesh,

$$\mathcal{S} = \{\boldsymbol{z} = (q_i, v_i, e_i, \tau_i, i = 1, 2, \ldots, N) \in (\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R})^N\}, \tag{22}$$

resulting in a full-order model of $D = 4 \cdot N$ dimensions.

4.2. Database and hyperparameters

The training database for this Olroyd-B model is generated in MATLAB with a multiscale approach [35] in dimensionless form. The fluid is discretized in the vertical direction with $N = 100$ elements (101 nodes) in a total

Table 1
Left: Mean squared error of the SAE reconstruction (MSE^{SAE}) for the 4 state variables of the Olroyd-B Couette flow example, reported only for the test snapshots. Right: Mean squared error of the same reduction using a Proper Orthogonal Decomposition algorithm (MSE^{POD}).

State variable (z_i)	MSE^{SAE}	MSE^{POD}
q [-]	$2.52 \cdot 10^{-6}$	$7.87 \cdot 10^{-6}$
v [-]	$7.27 \cdot 10^{-5}$	$4.31 \cdot 10^{-5}$
e [-]	$1.89 \cdot 10^{-6}$	$7.33 \cdot 10^{-6}$
τ [-]	$7.22 \cdot 10^{-6}$	$2.07 \cdot 10^{-5}$

height of $H = 1$. A total of 10,000 dumbbells were considered at each nodal location in the model. The lid velocity is set to $V = 1$, the viscolastic Weissenberg number We $= 1$ and Reynolds number of Re $= 0.1$. The simulation time of the movement is $T = 1$ in time increments of $\Delta t = 0.0067$ ($N_T = 150$ snapshots).

The database consists of the state vector, Eq. (22), of the 100 nodal trajectories (excluding the node at $y = H$, for which a no-slip condition $v = 0$ has been imposed) for each snapshot of the simulation. This database is split in 120 train snapshots and 30 test snapshots.

The SAE input and output sizes are $N_{\text{in}}^{\text{SAE}} = N_{\text{out}}^{\text{SAE}} = D = 4 \cdot N = 400$. The number of hidden layers in both the encoder and decoder is $N_h^{\text{SAE}} = 2$ with 160 neurons each, ReLU activation functions and linear in the first and last layer. The number of bottleneck variables is set to $N_d = 10$. It is initialized according to the Kaiming method [36], with normal distribution and the optimizer used is Adam [37], with a learning rate of $l_r^{\text{SAE}} = 10^{-4}$. The sparsity parameter is set to $\lambda_r^{\text{SAE}} = 10^{-4}$. The training process (Algorithm 1) is able to sparsify the bottleneck variables of the Olroyd-B model with only $d = 4$ latent variables, which are the input variables used in the structure preserving-neural network.

Thus, the SPNN input and output size are $N_{\text{in}}^{\text{SPNN}} = d = 4$ and $N_{\text{out}}^{\text{SPNN}} = d \cdot (d + 2) = 24$. The number of hidden layers is $N_h^{\text{SPNN}} = 5$ with 24 neurons each, ReLU activation functions and linear in the last layer. The same initialization method and optimizer are used as in the SAE network, with a learning rate of $l_r^{\text{SPNN}} = 10^{-5}$. The weight decay and the data weight are set to $\lambda_r^{\text{SPNN}} = 10^{-5}$ and $\lambda_d^{\text{SPNN}} = 10^3$ respectively.

The unconstrained network training parameters are analogous to the structure-preserving network, except for the output size $N_{out}^{UC} = N_{in}^{UC} = 4$. Several network architectures were tested, and the lowest error is achieved with $N_h = 5$ hidden layers and 25 neurons each layer.

4.3. Results

Fig. 3 shows the time evolution of the SAE bottleneck variables after the complete training process. The sparsity constraint forces the unnecessary latent variables to vanish, remaining a learnt latent dimensionality of $d = 4$ relevant variables from a starting bottleneck dimension of $N_d = 10$ (Fig. 3, Top Left). This compares advantageously with the obtained dimensionality $d = 6$ of our previous work [38]. Table 1 shows the mean squared error of the SAE reconstruction, computed with Algorithm 2, and an equal reduction using Proper Orthogonal Decomposition. Then, the SPNN is able to integrate the whole trajectory of the relevant latent variables in the reduced manifold in good agreement with the original SAE reduction (Fig. 3, Top Right). The integration scheme also ensures that the time derivative of the energy (dE/dt) and entropy (dS/dt) of the system remain equal to zero or greater than zero respectively, in fulfillment with the first and second law of thermodynamics (Fig. 3, Bottom Left), computed with Eq. (12).

Fig. 4 presents the time evolution of the decoded state variables of the Olroyd-B Couette flow for 4 different nodes computed with the presented integration scheme and the ground truth. The results show a good agreement in the transient response of the Couette flow, even for the high nonlinearities of the internal energy and the conformation tensor shear component. The mean squared error of the total integration scheme, computed with Algorithm 4, for the 4 state variables is reported in Table 2 using the Structure-preserving neural network (SPNN) and the unconstrained approach (UC). Our neural network achieves less error than the unconstrained one, showing the importance of adding the physical constrains to the learning process, and this difference becomes greater in the second example.

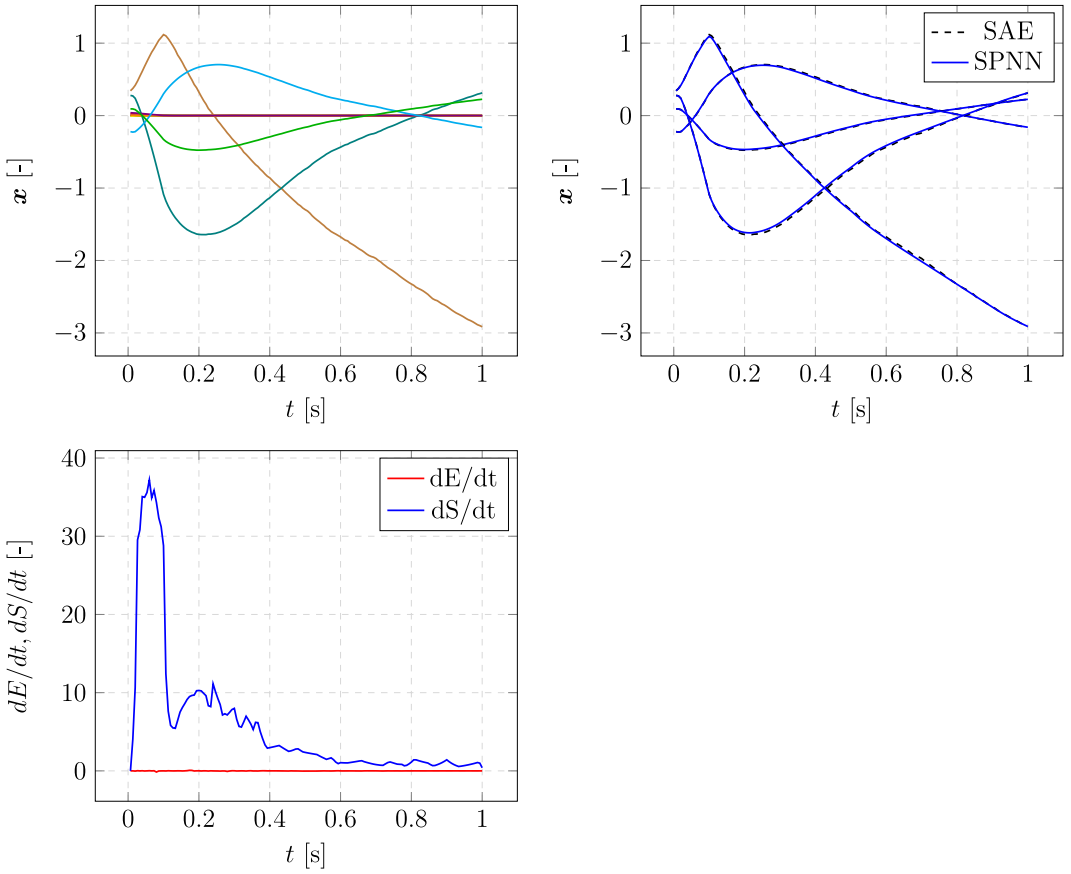

Fig. 3. Top Left: Time evolution of the latent variables encoded with the sparse autoencoder (SAE) in the Olroyd-B fluid problem. The bottleneck has $N_d = 10$ neurons and the learning algorithm automatically sparsifies them to a dimensionality of $d = 4$ relevant latent variables. Top Right: Time evolution of the relevant latent variables integrated in time by the structure-preserving neural network (SPNN). Bottom: Evolution of the time derivative of the energy (dE/dt) and entropy (dS/dt) of the latent variables.

Table 2
Mean squared error of the SPNN integration scheme and the unconstrained (UC) approach for the 4 state variables of the Olroyd-B Couette flow example, reported for the complete trajectory.

State variable (z_i)	MSE^{SPNN}	MSE^{UC}
q [-]	$1.78 \cdot 10^{-5}$	$7.96 \cdot 10^{-5}$
v [-]	$3.34 \cdot 10^{-5}$	$3.48 \cdot 10^{-5}$
e [-]	$5.60 \cdot 10^{-6}$	$5.67 \cdot 10^{-5}$
τ [-]	$2.19 \cdot 10^{-5}$	$1.22 \cdot 10^{-4}$

5. Rolling hyperelastic tire

5.1. Description

The second example is a simulation of the transient response of a 175 SR14 rolling tire ($D_{\text{tire}} = 0.66$ m) impacting with a curb ($h_{\text{curb}} = 0.025$ m). The tire is initially preloaded with an inflation load of 200 kPa, simulating the internal air pressure, and a footprint load of 3300 N in the vertical axis, simulating the weight of the vehicle distributed equally in the tires. The free rolling conditions are determined in a separated analysis, corresponding to $\omega = 8.98$ rad/s for a translational horizontal velocity of $v_0 = 10$ km/h (see Fig. 5).

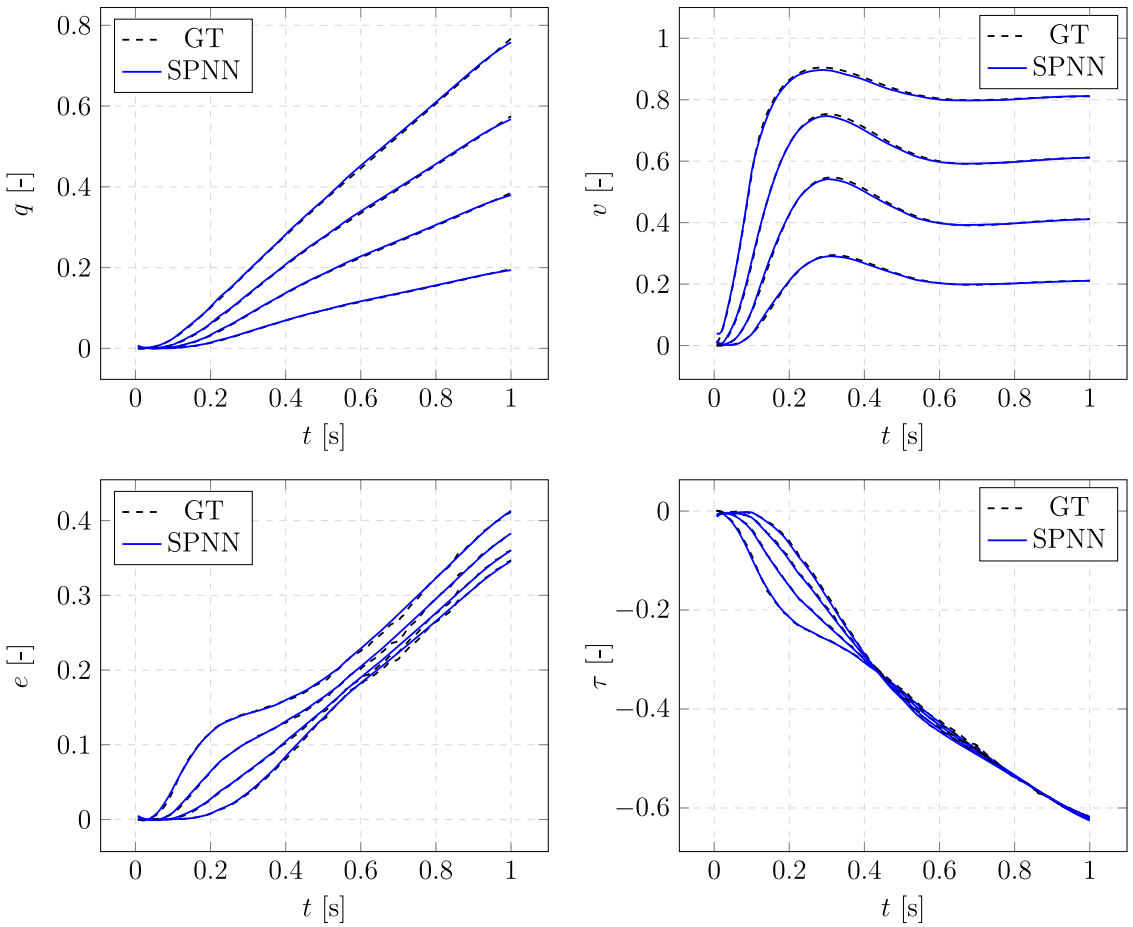

Fig. 4. Results of the complete integration scheme (SPNN) with respect to the ground truth simulation (GT) for 4 different nodes of the Olroyd-B fluid database.

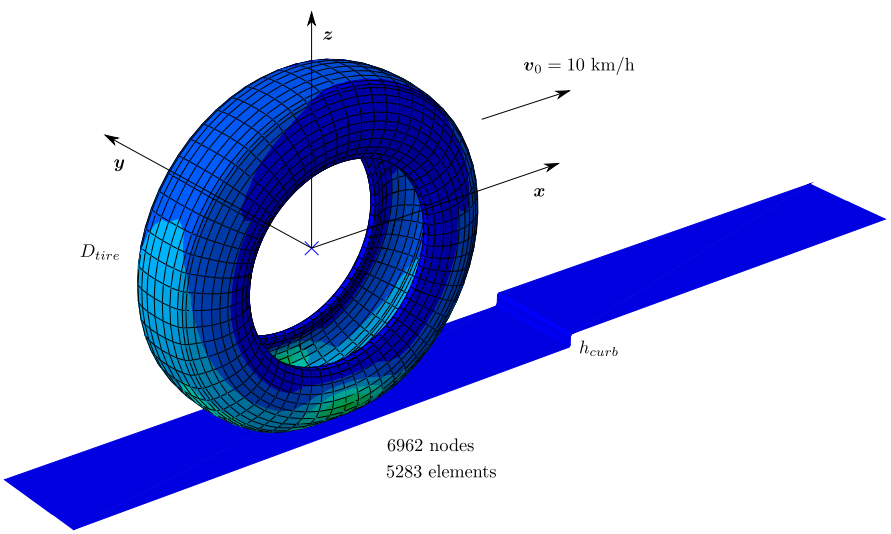

Fig. 5. Hyperelastic tire rolling towards a curb. 3D position, 3D velocity and Cauchy stress tensor components are tracked for the total of 4140 selected nodes.

The tread and sidewalls of the tire are made of rubber, modeled as an incompressible hyperelastic material with a viscolastic component described by a one-term Prony series of the dimensionless shear relaxation modulus,

$$g_R(t) = 1 - \bar{g}_1(1 - e^{\frac{-t}{\tau_1}}),$$

with relaxation coefficient of $\bar{g}_1 = 0.3$ and relaxation time of $\tau_1 = 0.1$ s. The belts and carcass of the tire are constructed from fiber-reinforced rubber composites, modeled as a linear elastic material, with a 20° orientation of the reinforcing belt.

The state variables chosen for the full order model are the 3D position $\boldsymbol{q}_i$, velocity $\boldsymbol{v}_i$ and the 6 different components of the Cauchy stress tensor $\boldsymbol{\sigma}_i$ for each i node of the studied mesh subset N,

$$\mathcal{S} = \{\boldsymbol{z} = (\boldsymbol{q}_i, \boldsymbol{v}_i, \boldsymbol{\sigma}_i, i = 1, 2, \ldots, N) \in (\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^6)^N\}, \tag{23}$$

resulting in a full-order model of $D = 12 \cdot N$ dimensions.

5.2. Database and hyperparameters

The training database for this rolling tire simulation is generated by finite element simulation. The full-order model is discretized with 5283 elements in a total of 6962 nodes. The simulation time of the movement is $T = 0.5$ s in time increments of $\Delta t = 0.0025$ s ($N_T = 200$ snapshots). The database consists of the normalized state vector (Eq. (23)) of a subset of $N = 4140$ relevant nodes in every time step snapshot. The total state vector snapshots are randomly split in 160 train snapshots and 40 test snapshots.

The SAE architecture for this second example is slightly modified in order to handle the high dimensionality of the problem. The three physical variables ($\boldsymbol{q}$, $\boldsymbol{v}$, and $\boldsymbol{\sigma}$) are encoded and decoded independently, due to their very different nature. In this way, three bottleneck latent vectors are obtained. The input and output sizes of the three SAEs are $N^{\text{SAE}}_{\text{in},q} = N^{\text{SAE}}_{\text{out},q} = 3 \cdot N = 12420$ for the position variable, $N^{\text{SAE}}_{\text{in},v} = N^{\text{SAE}}_{\text{out},v} = 3 \cdot N = 12420$ and $N^{\text{SAE}}_{\text{in},\sigma} = N^{\text{SAE}}_{\text{out},\sigma} = 6 \cdot N = 24840$ for the stress tensor.

The number of hidden layers in both the encoder and decoder is $N^{\text{SAE}}_h = 2$ in the three variables with 40 neurons each in position and velocity, and 80 neurons in the stress tensor, with ReLU activation functions and linear in the first and last layers. The number of bottleneck variables is set to $N_{d,q} = 10$ for the position, $N_{d,v} = 10$ for velocity and $N_{d,\sigma} = 20$ for the stress tensor. Thus, the total dimensionality of the bottleneck latent vector is $N_d = N_{d,q} + N_{d,v} + N_{d,\sigma} = 40$.

In the same way as we do in the first example, the nets are initialized according to the Kaiming method [36], with normal distribution and the optimizer used is Adam [37], with a learning rate of $l^{\text{SAE}}_r = 10^{-4}$. The sparsity parameter, in this case, is set to $\lambda^{\text{SAE}}_r = 10^{-2}$. The training process (Algorithm 1) is able to sparsify the bottleneck variables of the rolling tire model with only $d_q = 4$ position, $d_q = 3$ velocity and $d_\sigma = 2$ stress tensor latent variables. So, the learnt dimensionality of the reduced model is $d = d_q + d_v + d_\sigma = 9$, which are the input variables used in the structure preserving-neural network.

Thus, the SPNN input and output sizes are $N^{\text{SPNN}}_{\text{in}} = d = 9$ and $N^{\text{SPNN}}_{\text{out}} = d \cdot (d + 2) = 99$. The number of hidden layers is $N^{\text{SPNN}}_h = 5$ with 198 neurons each, with ReLU activation functions and linear in the last layer. The same initialization method and optimizer are used as in the SAE network, with a learning rate of $l^{\text{SPNN}}_r = 10^{-5}$. The weight decay and the data weight are set to $\lambda^{\text{SPNN}}_r = 10^{-4}$ and $\lambda^{\text{SPNN}}_d = 10^3$ respectively.

The unconstrained network training parameters are analogous to the structure-preserving network, except for the output size $N^{UC}_{out} = N^{UC}_{in} = 9$. Several network architectures were tested, and the lowest error is achieved with $N_h = 5$ hidden layers and 45 neurons each layer.

5.3. Results

Fig. 6 shows the time evolution of the SAE bottleneck variables ($\boldsymbol{x}_q$, $\boldsymbol{x}_v$ and $\boldsymbol{x}_\sigma$) after the complete training process. The sparsity constraint forces the unnecessary latent variables to vanish, remaining a learnt latent dimensionality of $d_q = 4$, $d_v = 3$ and $d_\sigma = 2$ relevant variables from a starting bottleneck dimension of $N_{d,q} = 10$, $N_{d,v} = 10$ and $N_{d,\sigma} = 20$ respectively (Fig. 6). The mean squared error of the SAE reconstruction, computed with Algorithm 2, and a equal reduction with a Proper Orthogonal Decomposition is shown in Table 3. Then, the SPNN is able to integrate the whole trajectory of the relevant latent variables in the reduced manifold in good agreement

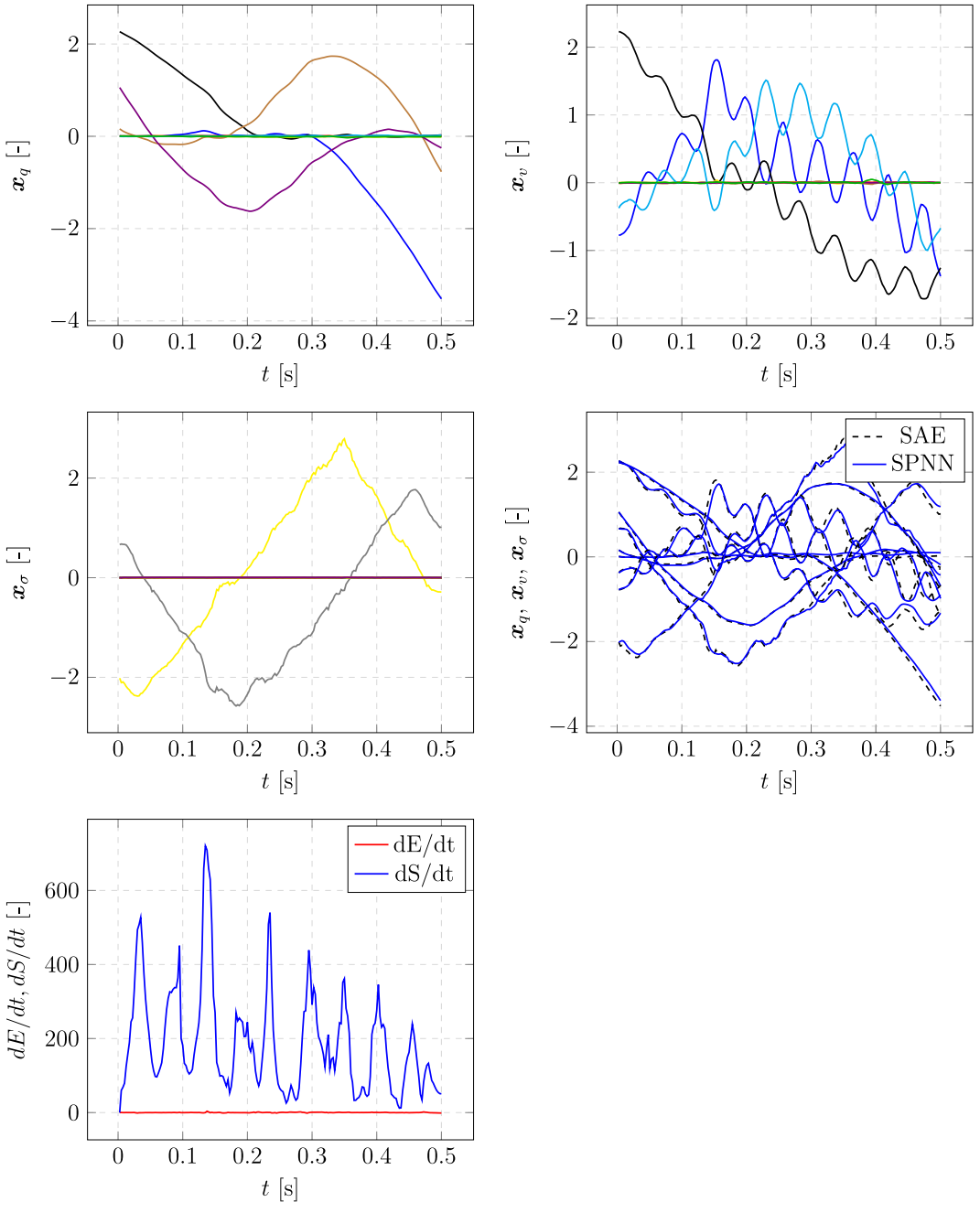

Fig. 6. Time evolution of the latent variables encoded with the sparse autoencoder (SAE) in the hyperelastic rolling tire problem. The bottleneck has $N_d = 40$ neurons and the learning algorithm sparsifies them to a dimensionality of $d = 9$ relevant latent variables. Bottom Middle Right: Time evolution of the relevant latent variables integrated in time by the structure-preserving neural network (SPNN). Bottom: Evolution of the time derivative of the energy (dE/dt) and entropy (dS/dt) of the latent variables.

with the original SAE reduction (Fig. 6, Bottom Middle Right). Also, the integration scheme fulfills the first and second laws of thermodynamics (Fig. 6, Bottom) computed with Eq. (12)

Fig. 7 presents the time evolution of the decoded state variables q_3, v_3, σ_{33} and σ_{23} of the rolling hyperelastic tire for 4 different nodes computed with the presented integration scheme and the ground truth. The mean squared error of the total integration scheme, computed with Algorithm 4, for the 12 state variables is reported in Table 4

Table 3
Left: Mean squared error of the SAE reconstruction (MSE^{SAE}) for the 12 state variables of the rolling tire example, reported only for the test snapshots. Right: Mean squared error of the same reduction using a Proper Orthogonal Decomposition algorithm (MSE^{POD}).

State variable (z_i)	MSE^{SAE}	MSE^{POD}
q_1 [m]	$2.37 \cdot 10^{-5}$	$1.30 \cdot 10^{-3}$
q_2 [m]	$3.69 \cdot 10^{-7}$	$6.27 \cdot 10^{-7}$
q_3 [m]	$3.06 \cdot 10^{-5}$	$6.55 \cdot 10^{-5}$
v_1 [m/s]	$1.00 \cdot 10^{-3}$	$3.32 \cdot 10^{-2}$
v_2 [m/s]	$4.54 \cdot 10^{-5}$	$2.37 \cdot 10^{-2}$
v_3 [m/s]	$3.70 \cdot 10^{-3}$	$6.91 \cdot 10^{-2}$
σ_{11} [MPa]	$2.41 \cdot 10^{-4}$	$3.74 \cdot 10^{-4}$
σ_{22} [MPa]	$2.10 \cdot 10^{-4}$	$4.34 \cdot 10^{-4}$
σ_{33} [MPa]	$3.35 \cdot 10^{-4}$	$6.40 \cdot 10^{-4}$
σ_{12} [MPa]	$6.73 \cdot 10^{-5}$	$1.17 \cdot 10^{-4}$
σ_{13} [MPa]	$1.80 \cdot 10^{-4}$	$3.24 \cdot 10^{-4}$
σ_{23} [MPa]	$2.95 \cdot 10^{-5}$	$5.86 \cdot 10^{-5}$

Table 4
Mean squared error of the SPNN integration scheme and the unconstrained (UC) approach for the 12 state variables of the rolling tire example, reported for the complete trajectory.

State variable (z_i)	MSE^{SPNN}	MSE^{UC}
q_1 [m]	$2.07 \cdot 10^{-4}$	$2.76 \cdot 10^{-1}$
q_2 [m]	$8.09 \cdot 10^{-7}$	$1.57 \cdot 10^{-5}$
q_3 [m]	$6.25 \cdot 10^{-5}$	$5.75 \cdot 10^{-2}$
v_1 [m/s]	$7.95 \cdot 10^{-3}$	4.26
v_2 [m/s]	$3.79 \cdot 10^{-5}$	$7.00 \cdot 10^{-4}$
v_3 [m/s]	$1.78 \cdot 10^{-2}$	4.39
σ_{11} [MPa]	$2.04 \cdot 10^{-4}$	$4.71 \cdot 10^{-3}$
σ_{22} [MPa]	$1.76 \cdot 10^{-4}$	$4.07 \cdot 10^{-3}$
σ_{33} [MPa]	$2.70 \cdot 10^{-4}$	$5.35 \cdot 10^{-3}$
σ_{12} [MPa]	$6.05 \cdot 10^{-5}$	$1.50 \cdot 10^{-3}$
σ_{13} [MPa]	$1.48 \cdot 10^{-4}$	$3.03 \cdot 10^{-3}$
σ_{23} [MPa]	$2.73 \cdot 10^{-5}$	$6.67 \cdot 10^{-4}$

using the Structure-preserving neural network (SPNN) and the unconstrained approach (UC). In this example, the error achieved by our method is several orders of magnitude less than the naive approach.

6. Conclusions

In this work, we develop a technique to learn the latent dimensionality of a physical system from data and obtain a thermodynamics-aware time integrator, which guarantees the fulfillment of the laws of thermodynamics. This technique is applied to two different physical systems. The Couette flow in a viscoelastic fluid is reduced from $D = 400$ dimensions to $d = 4$ dimensions, whereas the rolling tire is reduced from $D = 49680$ dimensions to $d = 9$ dimensions. The physically informed integrator is then able to predict the full time evolution of the set of state variables with similar precision reported in previous work [25,39].

If compared to previous works of the authors in the field, the use of autoencoders to unveil the dimensionality of the embedding manifold clearly outperforms the results obtained by classical (linear) model order reduction techniques, specially in highly nonlinear state variables such as the rolling tire velocity. In addition, it is worth highlighting the fact that the method is able to detect the true dimensionality of the data, with no need to call to different codes rely on additional methods, such as k-PCA or topological data analysis, for instance for this purpose. The right thermodynamic setting also ensures the consistency and stability of the full-order dynamics,

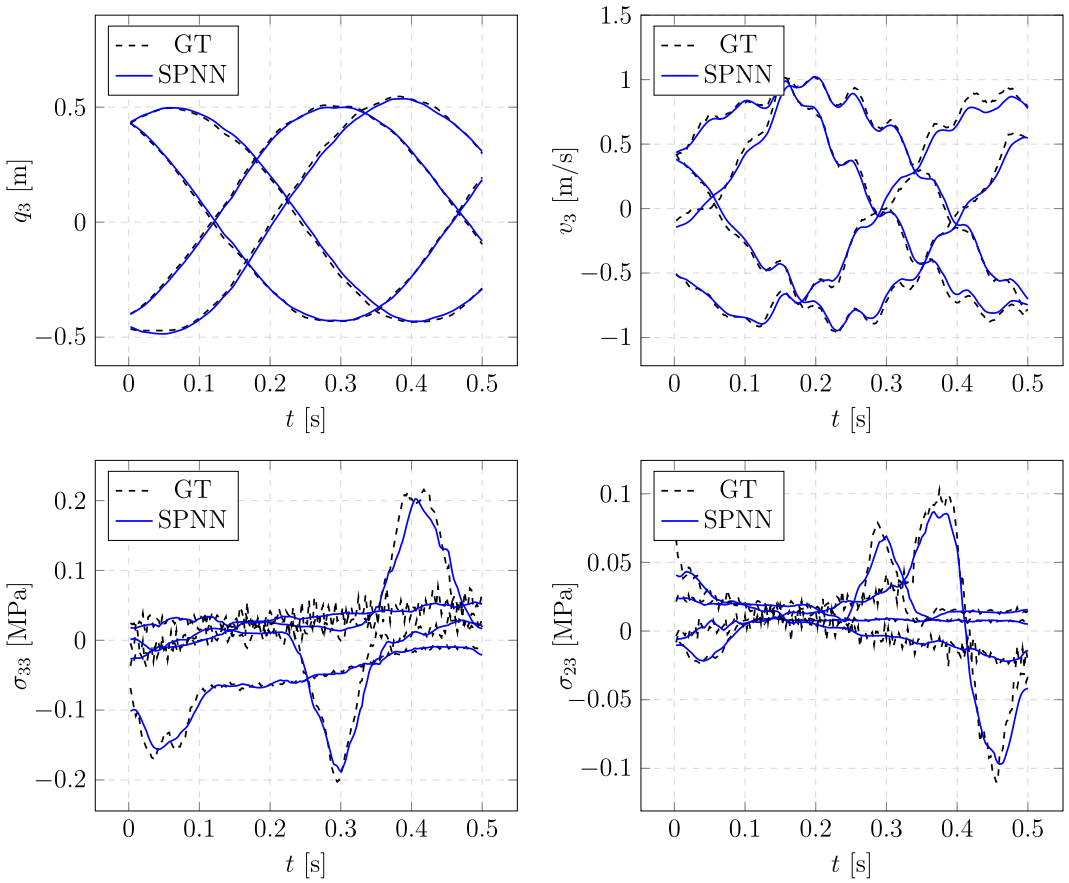

Fig. 7. Results of the complete integration scheme (SPNN) with respect to the ground truth simulation (GT) for 4 different nodes and 4 different variables (q_3, v_3, σ_{33} and σ_{23}) of the hyperelastic rolling tire database.

after projecting back the reduced-order results to the physical space, achieving better results than an unconstrained approach with no physical restrictions.

Some of the future work, including several improvements to the proposed algorithm, are listed below.

- **Database**: A limitation of the present work is the use of synthetic instead of experimental data. A research field is opened to test the limits of the presented methodology applied to real captured data, and to study the influence of noise in the measurements.
- **Autoencoder latent space**: The latent space of the reduction step can not only be sparsified, but also regularized using variational inference via Variational Autoencoders [40]. This Bayesian approach is convenient in cases where the latent variables are sampled or interpolated, and lead to smoother transitions between them. A future line of this work is to explore VAEs applied to physical systems and study its influence in the latent space topology and extrapolability.
- **Nets Architecture**: The solution of many physical systems has highly spatio-temporal correlations. Thus, convolutional [41] and graph-based [42] neural networks could be a more optimized approach, reducing the network complexity and speeding up the learning process.
- **Energy and Entropy gradients**: A modification can be performed in the structure-preserving neural network in order to output directly the energy and entropy functions. Then, the energy and entropy gradients of the GENERIC formulation can be computed via automatic differentiation with respect to the network input, the state vector. This way, the energy and entropy gradients are forced to be integrable [43].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Charles Fefferman, Sanjoy Mitter, Hariharan Narayanan, Testing the manifold hypothesis, J. Amer. Math. Soc. 29 (4) (2016) 983–1049.

[2] Siamak Niroomandi, Icíar Alfaro, Elías Cueto, Francisco Chinesta, Real-time deformable models of non-linear tissues by model reduction techniques, Comput. Methods Programs Biomed. 91 (3) (2008) 223–231.

[3] Juan Du, Fangxin Fang, Christopher C. Pain, I.M. Navon, Jiang Zhu, David A. Ham, Pod reduced-order unstructured mesh modeling applied to 2d and 3d fluid flow, Comput. Math. Appl. 65 (3) (2013) 362–379.

[4] Christophe Prud'Homme, Dimitrios V. Rovas, Karen Veroy, Luc Machiels, Yvon Maday, Anthony T. Patera, Gabriel Turinici, Reliable real-time solution of parametrized partial differential equations: Reduced-basis output bound methods, J. Fluids Eng. 124 (1) (2002) 70–80.

[5] Clarence W. Rowley, Tim Colonius, Richard M. Murray, Model reduction for compressible flows using pod and galerkin projection, Physica D 189 (1–2) (2004) 115–129.

[6] P.E. Farrell, J.R. Maddison, Conservative interpolation between volume meshes by local galerkin projection, Comput. Methods Appl. Mech. Engrg. 200 (1–4) (2011) 89–100.

[7] Alberto Badías, Sarah Curtit, David González, Icíar Alfaro, Francisco Chinesta, Elías Cueto, An augmented reality platform for interactive aerodynamic design and analysis, Internat. J. Numer. Methods Engrg. 120 (1) (2019) 125–138.

[8] Beatriz Moya, David. González, Icíar Alfaro, Francisco. Chinesta, E. Cueto, Learning slosh dynamics by means of data, Comput. Mech. 64 (2) (2019) 511–523.

[9] Beatriz Moya, Iciar Alfaro, David Gonzalez, Francisco Chinesta, Elías Cueto, Physically sound self-learning digital twins for sloshing fluids, PLOS ONE 15 (6) (2020) e0234569.

[10] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT press, 2016.

[11] Marco Farina, Yuichiro Nakai, David Shih, Searching for new physics with deep autoencoders, Phys. Rev. D 101 (7) (2020) 075021.

[12] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, Alexander Gaunt, Constrained graph variational autoencoders for molecule design, in: Advances in Neural Information Processing Systems, 2018, pp. 7795–7804.

[13] Kookjin Lee, Kevin T. Carlberg, J. Comput. Phys. 404 (2020) 108973.

[14] Julio Marco, Quercus Hernandez, Adolfo Munoz, Yue Dong, Adrian Jarabo, Min H. Kim, Xin Tong, Diego Gutierrez, Deeptof: off-the-shelf real-time correction of multipath interference in time-of-flight imaging, ACM Trans. Graph. (ToG) 36 (6) (2017) 1–12.

[15] Maziar Raissi, Paris Perdikaris, George E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.

[16] Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, David Duvenaud, Learning differential equations that are easy to solve, in: Neural Information Processing Systems, 2020.

[17] Tom Bertalan, Felix Dietrich, Igor Mezić, Ioannis G. Kevrekidis, On learning hamiltonian systems from data, Chaos 29 (12) (2019) 121107.

[18] Samuel Greydanus, Misko Dzamba, Jason Yosinski, Hamiltonian neural networks, in: Advances in Neural Information Processing Systems, 2019, pp. 15379–15389.

[19] Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, Irina Higgins, Hamiltonian generative networks, 2019, arXiv preprint arXiv:1909.13789.

[20] Yaofeng Desmond Zhong, Biswadip Dey, Amit Chakraborty, Symplectic ode-net: Learning hamiltonian dynamics with control, 2019, arXiv preprint arXiv:1909.12077.

[21] Yunjin Tong, Shiying Xiong, Xingzhe He, Guanghan Pan, Bo Zhu, Symplectic neural networks in taylor series form for hamiltonian systems, 2020, arXiv preprint arXiv:2005.04986.

[22] Pengzhan Jin, Zhen Zhang, Ioannis G. Kevrekidis, George Em Karniadakis, Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks, 2020, arXiv preprint arXiv:2012.03133.

[23] Hans Christian Öttinger, Miroslav Grmela, Dynamics and thermodynamics of complex fluids. ii. illustrations of a general formalism, Phys. Rev. E 56 (6) (1997) 6633.

[24] Miroslav Grmela, Hans Christian Öttinger, Dynamics and thermodynamics of complex fluids. i. development of a general formalism, Phys. Rev. E 56 (6) (1997) 6620.

[25] Quercus Hernandez, Alberto Badias, David Gonzalez, Francisco Chinesta, Elias Cueto, Structure-preserving neural networks, 2020, arXiv preprint arXiv:2004.04653.

[26] E. Weinan, A proposal on machine learning via dynamical systems, Commun. Math. Stat. 5 (1) (2017) 1–11.

[27] Hans Christian Öttinger, Preservation of thermodynamic structure in model reduction, Phys. Rev. E 91 (3) (2015) 032147.

[28] Andrew Ng, et al., Sparse autoencoder, CS294A Lect. Notes 72 (2011) (2011) 1–19.

[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, Adam Lerer, Automatic differentiation in pytorch. Autodiff workshop: The future of gradient-based machine learning software and techniques, 2017.

[30] Sebastian Ruder, An overview of gradient descent optimization algorithms, 2016, arXiv preprint arXiv:1609.04747.

[31] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al., Relational inductive biases, deep learning, and graph networks, 2018, arXiv preprint arXiv:1806.01261.
[32] Philip J. Morrison, A paradigm for joined hamiltonian and dissipative systems, Physica D 18 (1–3) (1986) 410–419.
[33] Jürgen Schmidhuber, Deep learning in neural networks: An overview, Neural networks 61 (2015) 85–117.
[34] Manuel Laso, Hans Christian Öttinger, Calculation of viscoelastic flow using molecular models: the connffessit approach, J. Non-Newton. Fluid Mech. 47 (1993) 1–20.
[35] Claude Le Bris, Tony Lelievre, Multiscale modelling of complex fluids: a mathematical initiation, in: Multiscale Modeling and Simulation in Science, Springer, 2009, pp. 49–137.
[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, ICCV, 2015, pp. 1026–1034.
[37] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[38] David González, Francisco Chinesta, Elías Cueto, Thermodynamically consistent data-driven computational mechanics, Contin. Mech. Thermodyn. 31 (1) (2019) 239–253.
[39] D. González, F. Chinesta, E. Cueto, Consistent data-driven computational mechanics, AIP Conf. Proc. 1960 (1) (2018) 090005.
[40] Diederik P. Kingma, Max Welling, Auto-encoding variational bayes, 2013, arXiv preprint arXiv:1312.6114.
[41] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, Ken Perlin, Accelerating eulerian fluid simulation with convolutional networks, 2016, arXiv preprint arXiv:1607.03597.
[42] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, Maosong Sun, Graph neural networks: A review of methods and applications, 2018, arXiv preprint arXiv:1812.08434.
[43] Gregory H. Teichert, A.R. Natarajan, A. Ven Van der, Krishna Garikipati, Machine learning materials physics: Integrable deep neural networks enable scale bridging by learning free energy functions, Comput. Methods Appl. Mech. Engrg. 353 (2019) 201–216.