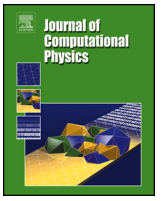
Journal of Computational Physics 500 (2024) 112738

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp



Pseudo-Hamiltonian neural networks for learning partial differential equations

Sølve Eidnes *, Kjetil Olsen Lye

Department of Mathematics and Cybernetics, SINTEF Digital, 0373 Oslo, Norway

ARTICLE INFO

Keywords:
Physics-informed machine learning
Hamiltonian neural networks
Partial differential equations
Inverse problem

ABSTRACT

Pseudo-Hamiltonian neural networks (PHNN) were recently introduced for learning dynamical systems that can be modelled by ordinary differential equations. In this paper, we extend the method to partial differential equations. The resulting model is comprised of up to three neural networks, modelling terms representing conservation, dissipation and external forces, and discrete convolution operators that can either be learned or be given as input. We demonstrate numerically the superior performance of PHNN compared to a baseline model that models the full dynamics by a single neural network. Moreover, since the PHNN model consists of three parts with different physical interpretations, these can be studied separately to gain insight into the system, and the learned model is applicable also if external forces are removed or changed.

1. Introduction

The field called physics-informed machine learning combines the strengths of physics-based models and data-driven techniques to achieve a deeper understanding and improved predictive capabilities for complex physical systems [1,2]. The rapidly growing interest in this interdisciplinary approach is largely motivated by the increasing capabilities of computers to store and process large quantities of data, along with the decreasing costs of sensors and computers that capture and handle data from physical systems. Machine learning for differential equations can broadly be divided into two categories: the forward problem, which involves predicting future states from an initial state, and the inverse problem, which entails learning a system or parts of it from data. A wealth of recent literature exists on machine learning for the forward problem in the context of partial differential equations (PDEs). The proposed methods include neural-network-based substitutes for numerical solvers [3–6], but also methods that can aid the solution process, e.g. by optimizing the discretization to be used in a solver [7]. The focus of this paper is on the inverse problem, and much of the foundation for our proposed model can be found in recent advances in learning neural network models for ordinary differential equations (ODEs). Specifically, we build on recent works on models that incorporate Hamiltonian mechanics and related structures that underlie the physical systems we seek to model.

Greydanus et al. introduced Hamiltonian neural networks (HNN) in [8], for learning finite-dimensional Hamiltonian systems from data. They assume that the data $q \in \mathbb{R}^n$, $p \in \mathbb{R}^n$ is obtained from a canonical Hamiltonian system

* Corresponding author.
E-mail address: solve.eidnes@sintef.no (S. Eidnes).

https://doi.org/10.1016/j.jcp.2023.112738
Received 19 June 2023; Received in revised form 22 December 2023; Accepted 23 December 2023

Available online 3 January 2024

$$\begin{pmatrix} \dot{q} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} 0 & I_n \\ -I_n & 0 \end{pmatrix} \begin{pmatrix} \frac{\partial H}{\partial q} \\ \frac{\partial H}{\partial p} \end{pmatrix},$$

and aim to learn a neural network model $\hat{H}_\theta$ of the Hamiltonian $H : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. This approach has since been further explored and expanded in a number of directions, which include considering control input [9], dissipative systems [10,11], constrained systems [12,13], port-Hamiltonian systems [14–16], and metriplectic system formulations [17,18]. A similar approach considering a Lagrangian formulation instead of a Hamiltonian is presented in [19]. Modifications of the models that focus on improved training from sparse and noisy data have been proposed in [20–23].

In [24], we proposed pseudo-Hamiltonian neural networks (PHNN). These can learn what we call pseudo-Hamiltonian systems, which generalizes Hamiltonian systems first to any invariant-preserving system and further allows for dissipation and external forces acting on the system. Thus we consider the formulation

$$\dot{x} = (S(x) - R(x))\nabla H(x) + f(x,t), \qquad x \in \mathbb{R}^d, \tag{1}$$

where $S(x) = -S(x)^T$, and $y^T R(x) y \geq 0$ for all y. That is, $S(x) \in \mathbb{R}^{d\times d}$ can be any skew-symmetric matrix and $R(x) \in \mathbb{R}^{d\times d}$ can be any positive semi-definite matrix. Since we put no restrictions on the external forces, a pseudo-Hamiltonian formulation can in principle be obtained for any first-order ODE, which in turn can be obtained from any arbitrary-order ODE by a variable transformation. The formulation makes it possible to learn models that can be separated into internal dynamics and the external forces, i.e. $(\hat{S}_\theta(x) - \hat{R}_\theta(x))\nabla \hat{H}_\theta(x)$ and $\hat{f}_\theta(x,t)$. This requires some sense of uniqueness in this separation, so certain restrictions need be put on the model to consider systems less general than (1). A major advantage that comes with this feature of the PHNN approach is that it makes it possible to learn a model of the system as if under ideal conditions even if data is sampled from a system affected by disturbances, if one assumes that an undisturbed system is closed and thus given only by the internal dynamics.

The motivating idea behind the present paper is to extend the framework of [24] to PDEs. In principle, one could always treat the spatially discretized PDE as a system of ODEs and apply the PHNN models of [24] to that. However, that would be disregarding certain structures we know to be present in the discretized PDE and would lead to inefficient models. Thus, compared to the ODE case, we consider different neural network architectures. Moreover, we will in the PDE setting impose some restrictions on the form of the external forces, in that we will not allow for them to depend on spatial derivatives of the solution. On the other hand, we will consider a more general form of the internal dynamics, where dissipation can result from a separate term and not just damping of the Hamiltonian. This mean that we can model metriplectic systems, in addition to Hamiltonian, port-Hamiltonian and dissipative systems.

Although HNN and extensions of this have attracted considerable attention in recent years, there has been very few studies on extending the methodology to PDEs. To our knowledge, the only prior works that consider HNN for PDEs are those of Matsubara et al. in [25] and Jin et al. in [26]. The latter has included a numerical example on the nonlinear Schrödinger equation. The former reference considers both Hamiltonian PDEs, exemplified by the Korteweg–de Vries equation, and an extension to dissipative PDEs, demonstrated on the Cahn–Hilliard equation. That paper has been a major inspiration for our work, especially on the neural network architecture we use to model the integrals in our PDE formulation. By generalizing to a wider class of PDEs that can have conservative and dissipative terms at once, and also allowing for external forces, we largely expand the utility of this learning approach.

The extension of PHNN to PDEs we propose here should naturally also be put in context with other recent advances in learning of PDEs from data. Long et al. introduced PDE-Net in [27], and together with [25] this may be the work that is most comparable to what we present here. Their model is similar to the baseline model we will compare PHNN to in this paper, albeit less general. Their approach has two components: learning neural network models for the nonlinear terms in the PDE and identifying convolution operators that correspond to appropriate finite difference approximations of the spatial derivative operators present. They do however make considerable simplifying assumptions in their numerical experiments, e.g. only considering linear terms and a forward Euler discretization in time. Other works that have received significant attention are those that have focused on identifying coefficients of terms of the PDE, both in the setting where one assumes that the terms are known and approximate them by neural network models [6] and in the setting where one also identifies the terms present from a search space of candidates using sparse regression [28–30]. There has also been considerable recent research on learning operators associated with the underlying PDEs, where two prominent methods are Fourier neural operators (FNO) [31,32] and deep operator networks (DeepONet) [33]. These operators can e.g. map from a source term to solution states or from an initial state to future states; in the latter case, learning the operator equates to solving the forward problem of the PDE. The review paper [34] summarizes the literature on operator learning and system identification of PDEs, as well as recent developments on learning order reductions.

As will be demonstrated theoretically and experimentally in this paper, assuming a pseudo-Hamiltonian structure when solving the inverse problem for PDEs has both qualitative and quantitative advantages. The latter is shown by numerical comparisons to a baseline model on five test cases. The main qualitative feature of PHNN is that it is composed of up to six trainable submodels, which after training each can be studied for an increased understanding of the system we are modelling. And if initial experiments for instance indicate that the system is Hamiltonian, we can retrain with this assumption imposed and thus learn more accurate solutions by pure HNN models. Moreover, we could train a system affected by external forces and remove these from the model after training, so that we have a model unaffected by these disturbances. The code for this paper is built on the code for PHNN for ODEs developed for [24], and we have updated the GitHub repository https://github.com/SINTEF/pseudo-hamiltonian-neural-networks and the Python package `phlearn` with this extension to the PDE case.

The rest of this paper is organised as follows. In the next section, we explore the theoretical foundations upon which our method is based. Then the pseudo-Hamiltonian formulation and the class of PDEs we will learn are presented and discussed in Section 3.

Section 4 is the centrepiece of the paper, as it is here we present the PHNN method for PDEs. We then dedicate a substantial portion of the paper to presenting and evaluating numerical results for various PDEs, in Section 5. The penultimate section is devoted to analysis of the results and our model, and a discussion of open questions to address in future research. We summarize the main results and draw some conclusions in the last section.

2. Background: derivatives, discretizations and neural networks

Before delving into the pseudo-Hamiltonian formulation and the model we propose based on this, we will review and discuss some requisites for making efficient neural network models of systems governed by PDEs.

2.1. Learning dynamical systems

Consider the first-order in time and p-order in space PDE

$$u_t = g(u^\alpha, x, t), \quad u \in H^p(\Omega), x \in \Omega \subseteq \mathbb{R}^d, t \in \mathbb{R}, \tag{2}$$

with

$$u^\alpha = \left\{ \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}} : |\alpha| \leq p \right\}, \quad \alpha \in (\mathbb{Z}^{\geq})^d.$$

We seek to train a model $\hat{g}_\theta$ of g so that solving $u_t = \hat{g}_\theta(u^\alpha, x, t)$ leads to accurate predictions of the future states of the system. The universal approximation theorem [35,36] states that g can be approximated with an arbitrarily small error by a neural network. In practise, we have to assume an abundance of observations of u_t and u^α at t and x to actually find a precise neural network approximation of g. This brings us straight to one of the fundamental challenges of machine learning of differential equations: in a typical real-world setting, we cannot expect to have data on the derivatives, neither temporal nor spatial. Thus we will have to depend on approximations obtained from discrete data. In this paper we will use sub- and superscript to denote discrete solution points in space resp. time. That is, $u^\alpha(x) = u(x, t^j)$ and $u_i(t) = u(x_i, t)$, and we will suppress the arguments when they are not necessary. Let us consider the issue of time-discretization first, an issue shared by ODEs and PDEs alike, and defer the second issue to the next subsection.

In several of the papers introducing the most prominent recent methods for learning finite-dimensional dynamical systems, e.g. the original HNN paper [8] and the first paper by Brunton et al. on system identification [34], the derivatives of the solution are assumed to be known or approximated by finite differences. Approximating the time-derivative by the forward finite difference operator is equivalent to training using the forward Euler integrator, which is also what is done in the PDE-Net papers [27,37]. However, there has been several recent papers proposing more efficient training methods that incorporate other numerical integration schemes, see e.g. [20–22]. We follow [24,38] and set up the training is such a way that we can use any mono-implicit integrator; that is, any integrator that relies explicitly on the solution in the times it integrates from and to. For the majority of the experiments in this paper, we use the implicit midpoint method, which is second-order, symplectic and symmetric. That is, we train the model $\hat{g}_\theta$ by identifying the parameters θ that minimize the loss function

$$\mathcal{L}_{g_\theta} = \left\| \frac{u^{j+1} - u^j}{\Delta t} - \hat{g}_\theta\left(\frac{(u^\alpha)^j + (u^\alpha)^{j+1}}{2}, x, \frac{t^j + t^{j+1}}{2} \right) \right\|_2^2,$$

given for one training point u^α and barring regularization for now. This yields a considerable improvement over the forward Euler method at next to no additional computational cost, since the model in both cases is evaluated at only one point at each iteration of training. The option to use other integrators, including symmetric methods of order four and six, is readily implemented in the `phlearn` package, and we do demonstrate the need for and utility of a fourth-order integrator in Section 5.4. For a thorough study of integrators especially suited for training neural network models of dynamical systems, we refer the reader to [38,39].

2.2. Spatial derivatives and convolution operators

Moving from finite-dimensional systems to infinite-dimensional systems introduces the issue of how to approximate spatial derivatives by the neural network models. Thankfully, a proposed solution to this issue can be found in recent literature, as several works have noted the connection between finite difference schemes for differential equations and the convolutional neural network models originally developed for image analysis; see [40–42,7,43].

Given a function u and a kernel or filter w, a discrete convolution is defined by

$$(u * w)(x_i) = \sum_{j=-r}^{s} w_j u(x_{i-j}), \quad r, s \geq 0. \tag{3}$$

Here $*$ is called the convolution operator, and the kernel w is a tensor containing trainable weights: $w = [w_{-r}, w_{-r+1}, \ldots, w_0, \ldots, w_{s-1}, w_s]$. If the function u is periodic, so that $u(x_i) = u(x_{i+M})$ for some M, we obtain a circular convolution, which can be expressed by a circulant matrix applied on the vector $u = \left[u_0, \ldots, u_{M-1}\right]^T$, where $u_i := u(x_i)$.

A convolutional layer in a neural network can be represented as

$$y_k(u_i) = \phi\big((u * w_k)(x_i) + b_k\big) = \phi\Big(\sum_{j=-r}^{s} w_{kj} u_{i-j} + b_k\Big),$$

where $y_k(u_i)$ is the output of the k-th feature map at point u_i, w_{kj} are the weights of the kernel w_k, b_k is the bias term, and $\phi(\cdot)$ is an activation function. The width of the layer is $r + s + 1$, and this is usually referred to as the size of the convolution kernel, or filter. For our purpose it makes sense to have either $r = 0$ or $r = s$, and the latter is the standard when convolutional neural networks are used in image analysis. Training the convolutional layer of a neural network constitutes of optimizing the weights and biases, which we collectively denoted by θ in the previous subsection.

Similarly, a finite difference approximation of the n-th order derivative of u at a point x_i can also be expressed as applying a discrete convolution:

$$\frac{\mathrm{d}^n u(x_i)}{\mathrm{d}x^n} \approx \sum_{j=-r}^{s} a_j u(x_{i-j}), \tag{4}$$

where the finite difference weights a_j depend on the spatial grid. If we assume the spatial points to be equidistributed and let $h := x_{i+1} - x_i$, we have e.g.

$$\begin{aligned}
\frac{\mathrm{d}u(x_i)}{\mathrm{d}x} &= \frac{u(x_{i+1}) - u(x_i)}{h} + \mathcal{O}(h),\\
\frac{\mathrm{d}u(x_i)}{\mathrm{d}x} &= \frac{u(x_{i+1}) - u(x_{i-1})}{2h} + \mathcal{O}(h^2),\\
\frac{\mathrm{d}^2u(x_i)}{\mathrm{d}x^2} &= \frac{u(x_{i+1}) - u(x_i) + u(x_{i-1})}{h^2} + \mathcal{O}(h^2),\\
\frac{\mathrm{d}^3u(x_i)}{\mathrm{d}x^3} &= \frac{u(x_{i+2}) - 2u(x_{i+1}) + 2u(x_{i-1}) - u(x_{i-2})}{2h^3} + \mathcal{O}(h^2).
\end{aligned}$$

Hence, a kernel size of two, with e.g. $r = 0$ and $s = 1$, is sufficient to obtain a first-order approximation of the first derivative, while a kernel size of three is sufficient and necessary to obtain second order approximations of first and second derivatives. Further, kernel size five is needed to approximate the third derivative. As noted by [42], higher-order derivatives can be approximated either by increasing the kernel size or applying multiple convolution operations. In our models, we have designed neural networks where only the first layer is convolutional, and thus the kernel size restricts the order of the derivative we can expect to learn, while it also restricts the order of the approximations of these derivatives.

2.3. Variational derivative

Given the function H depending on u, x and the first derivative u_x, let $\mathcal{H}$ be the integral of H over the spatial domain:

$$\mathcal{H}[u] = \int_\Omega H(x, u, u_x)\,\mathrm{d}x. \tag{5}$$

The variational derivative, or functional derivative, $\frac{\delta \mathcal{H}}{\delta u}[u]$ of $\mathcal{H}$ is defined by the property

$$\left\langle \frac{\delta \mathcal{H}}{\delta u}[u], v \right\rangle_{L_2} = \frac{\mathrm{d}}{\mathrm{d}\epsilon}\bigg|_{\epsilon=0} \mathcal{H}[u + \epsilon v] \quad \forall v \in H^p(\Omega). \tag{6}$$

When $\mathcal{H}$ as here only depends on first derivatives, the variational derivative can be calculated explicitly by the relation

$$\frac{\delta \mathcal{H}}{\delta u}[u] = \frac{\partial H}{\partial u} - \frac{\mathrm{d}}{\mathrm{d}x}\frac{\partial H}{\partial u_x},$$

assuming enough regularity in H.

3. Pseudo-Hamiltonian formulation of PDEs

In this paper we consider the class of PDEs that can be written on the form

$$u_t = S(u^\alpha, x)\frac{\delta \mathcal{H}}{\delta u}[u] - R(u^\alpha, x)\frac{\delta \mathcal{V}}{\delta u}[u] + f(u^\alpha, x, t), \tag{7}$$

where $S(u^\alpha, x)$ and $R(u^\alpha, x)$ are operators that are skew-symmetric resp. positive semi-definite with respect to the L^2 inner product, $\mathcal{H}$ and $\mathcal{V}$ are integrals of the form (5) and $f : \mathbb{R} \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$. To be consistent with our previous work [24] and to make clear the connection to the vast recent literature on Hamiltonian neural networks, we say that (7) is the class of *pseudo-Hamiltonian PDEs*. This marks a generalization of the definition used in [24], in addition to the extension to infinite-dimensional systems, in that we here allow for $\mathcal{H}$ and $\mathcal{V}$ to be two different integrals.

The naming of this class is a challenge, since similar but not identical classes have been called by a myriad of names in the literature. Ignoring the term f, the class could be referred to as metriplectic PDEs, where the name is a portmanteau of metric and symplectic [44,45]. Examples of metriplectic PDEs are the viscous Burgers' equation and the Navier–Stokes equation. The formulation is also similar to an infinite-dimensional variant of the General Equation for Non-Equilibrium Reversible-Irreversible Coupling (GENERIC) formalism from thermodynamics [46,47], except for f and the fact that $R(u^\alpha, x)$ is positive instead of negative semi-definite. Furthermore, the GENERIC formalism requires the degeneracy conditions

$$R(u^\alpha, x)\frac{\delta \mathcal{H}}{\delta u}[u] = S(u^\alpha, x)\frac{\delta \mathcal{V}}{\delta u}[u] = 0$$

to be satisfied. We do not assume this to be satisfied and thus do not impose this condition on our model, but we consider that a highly relevant future extension of our work. In the finite-dimensional case, neural networks that preserve the GENERIC formalism have been studied in [48].

In the case $\mathcal{V} = 0$ and $f(u^\alpha, x, t) = 0$, we have the class of integral-preserving PDEs, which encompasses all (non-canonical) Hamiltonian PDEs [49]. That is, given the appropriate boundary conditions, e.g. periodic, the PDE will preserve the integral $\mathcal{H}$, usually labelled the integral of motion, of the system. This follows from the skew-symmetry of S:

$$\frac{\mathrm{d}\mathcal{H}}{\mathrm{d}t} = \left\langle \frac{\delta \mathcal{H}}{\delta u}[u], \frac{\partial u}{\partial t} \right\rangle_{L^2} = \left\langle \frac{\delta \mathcal{H}}{\delta u}[u], S(u^\alpha, x)\frac{\delta \mathcal{H}}{\delta u}[u] \right\rangle_{L^2} = 0.$$

If S in addition satisfies the Jacobi identity and thus defines a Poisson bracket, $\mathcal{H}$ is a Hamiltonian of the system [50]. If $\mathcal{H} = 0$ and $f(u^\alpha, x, t) = 0$ but $\mathcal{V} \geq 0$, the PDE (7) will dissipate the integral $\mathcal{V}$, and $\mathcal{V}$ may be called a Lyapunov function.

The general pseudo-Hamiltonian formulation does not in itself have a geometric structure. The motivation for still considering this formulation is two-fold: i) to develop a general machine learning model where geometric structures can be imposed to handle different system classes, including Hamiltonian, port-Hamiltonian, dissipative and metriplectic PDEs, with and without external forces; ii) to obtain grey-box models with parts that can be studied separately to understand more about the system.

Example 1. Consider the KdV–Burgers (or viscous KdV) equation [51]

$$u_t + \eta u u_x - \nu u_{xx} - \gamma^2 u_{xxx} = 0. \tag{8}$$

This is a metriplectic PDE that can be written on the form (16) with A and R both being the identity operator I, $S = \frac{\partial}{\partial x}$ and $f(u, x, t) = 0$, and

$$\mathcal{H} = -\int_\Omega \left(\frac{\eta}{6}u^3 + \frac{\gamma^2}{2}u_x^2 \right) dx \tag{9}$$

and

$$\mathcal{V} = \frac{\nu}{2}\int_\Omega u_x^2 \, dx. \tag{10}$$

We see this connection by deriving the variational derivatives

$$\frac{\delta \mathcal{H}}{\delta u}[u] = -(\frac{\eta}{2}u^2 - \gamma^2 u_{xx}) \tag{11}$$

and

$$\frac{\delta \mathcal{V}}{\delta u}[u] = -\nu u_{xx}. \tag{12}$$

We have that (8) reduces to the inviscid Burgers' equation for $\eta = -1$ and $\nu = \gamma = 0$, the viscous Burgers' equation for $\eta = -1$, $\nu \neq 0$ and $\gamma = 0$, and the Korteweg–de Vries (KdV) equation for $\nu = 0$, $\eta \neq 0$ and $\gamma \neq 0$.

3.1. Spatial discretization

In this section and this section only we will use boldface notation for vectors, to distinguish continuous functions and parameters from their spatial discretizations. Assume that values of u are obtained at grid points $\mathbf{x} = [x_0, \ldots, x_M]^T$. Following [52], we interpret these as quadrature points with non-zero quadrature weights $\kappa = [\kappa_0, \ldots, \kappa_M]^T$, and approximate the L_2 inner product by a weighted discrete inner product:

$$\langle u, v \rangle = \int_\Omega u(x)v(x)\,\mathrm{d}x \approx \sum_{i=0}^{M} \kappa_i u(x_i)v(x_i) = u^T \mathrm{diag}(\kappa) v =: \langle u, v \rangle_\kappa.$$

Let $\mathbf{p}$ denote the discretization parameters that consist of $\mathbf{x}$ and the associated κ. Then, assuming that there exists a consistent approximation $\mathcal{H}_\mathbf{p}(\mathbf{u})$ to $\mathcal{H}[u]$ that depends on u evaluated at $\mathbf{x}$, we define the discretized variational derivative by the analogue to (6)

$$\left\langle \frac{\delta \mathcal{H}_{\mathbf{p}}}{\delta \mathbf{u}}(\mathbf{u}), \mathbf{v} \right\rangle_{\kappa} = \frac{\mathrm{d}}{\mathrm{d}\epsilon}\bigg|_{\epsilon=0} \mathcal{H}_{\mathbf{p}}(\mathbf{u} + \epsilon \mathbf{v}) \quad \forall \mathbf{v} \in \mathbb{R}^{M+1}.$$

Thus, as shown in [52], we have a relationship between the discretized variational derivative and the gradient:

$$\frac{\delta \mathcal{H}_{\mathbf{p}}}{\delta \mathbf{u}}(\mathbf{u}) = \mathrm{diag}(\kappa)^{-1} \nabla_{\mathbf{u}} \mathcal{H}_{\mathbf{p}}(\mathbf{u}).$$

Furthermore, we approximate $S(u^\alpha, x)$ and $R(u^\alpha, x)$ by matrices $S_d(\mathbf{u})$ and $R_d(\mathbf{u})$ that are skew-symmetric resp. positive semi-definite with respect to $\langle \cdot, \cdot \rangle_\kappa$. Then a spatial discretization of (7) is given by

$$\mathbf{u}_t = S_d(\mathbf{u}) \frac{\delta \mathcal{H}_{\mathbf{p}}}{\delta \mathbf{u}}(\mathbf{u}) - R_d(\mathbf{u}) \frac{\delta \mathcal{V}_{\mathbf{p}}}{\delta \mathbf{u}}(\mathbf{u}) + \mathbf{f}(\mathbf{u}, \mathbf{x}, t),$$

which may equivalently be written as

$$\mathbf{u}_t = S_{\mathbf{p}}(\mathbf{u}) \nabla_{\mathbf{u}} \mathcal{H}_{\mathbf{p}}(\mathbf{u}) - R_{\mathbf{p}}(\mathbf{u}) \nabla_{\mathbf{u}} \mathcal{V}_{\mathbf{p}}(\mathbf{u}) + \mathbf{f}(\mathbf{u}, \mathbf{x}, t), \tag{13}$$

where $S_{\mathbf{p}}(\mathbf{u}) := S_d(\mathbf{u}) \, \mathrm{diag}(\kappa)^{-1}$ and $R_{\mathbf{p}}(\mathbf{u}) := R_d(\mathbf{u}) \, \mathrm{diag}(\kappa)^{-1}$ are skew-symmetric resp. positive semi-definite by the standard definitions for matrices.

Thus, upon discretizing in space, we obtain a system of ODEs (13) that is on a form quite similar to the generalized pseudo-Hamiltonian formulation considered in [24]. In fact, if $\mathcal{V} = \mathcal{H}$, we obtain the system

$$\mathbf{u}_t = (S_{\mathbf{p}}(\mathbf{u}) - R_{\mathbf{p}}(\mathbf{u})) \nabla_{\mathbf{u}} \mathcal{H}_{\mathbf{p}}(\mathbf{u}) + \mathbf{f}(\mathbf{u}, \mathbf{x}, t).$$

Still, we do not recommend applying the PHNN method of [24] on this directly without taking into consideration what we know about $\mathcal{H}_{\mathbf{p}}$. Specifically, we want to exploit that it is a discrete approximation of the integral (5), and can thus be expected to be given by a sum of M terms that each depend in the same way on u_i and the neighbouring points u_{i-1} and u_{i+1}. Hence, as discussed in the next section, we will employ convolutional neural networks with weight sharing across the spatial discretization points.

Example 2. Consider again the KdV–Burgers equation (8), on the domain $\Omega = [0, P]$ with periodic boundary conditions $u(0,t) = u(P,t)$. We assume that the $M+1$ grid points are equidistributed and define $h := x_{i+1} - x_i = P/M$. We approximate the integrals (9) and (10) by

$$\mathcal{H}_{\mathbf{p}} = -h \sum_{i=0}^{M-1} \left(\frac{\eta}{6} u_i^3 + \frac{\gamma}{2} \left(\delta_f u_i \right)^2 \right) \tag{14}$$

and

$$\mathcal{V}_{\mathbf{p}} = \frac{\nu}{2} h \sum_{i=0}^{M-1} \kappa_i \left(\delta_f u_i \right)^2, \tag{15}$$

where the operator δ_f denotes forward difference, i.e. $\delta_f u_i = (u_{i+1} - u_i)/h$. Furthermore, we approximate ∂_x by the matrix corresponding to the central difference approximation δ_c defined by $\delta_c u_i = (u_{i+1} - u_{i-1})/(2h)$, i.e.

$$S_d = \frac{1}{2h} \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & -1 \\ -1 & 0 & 1 & 0 & \cdots & \\ 0 & -1 & 0 & 1 & 0 & \cdots \\ & & \ddots & \ddots & \ddots & \\ & \cdots & 0 & -1 & 0 & 1 \\ 1 & 0 & \cdots & 0 & -1 & 0 \end{pmatrix} \in \mathbb{R}^{M \times M},$$

where the first and last rows are adjusted according to the periodic boundary conditions.

To obtain (13), we have that $S_{\mathbf{p}} = \frac{1}{h} S_d$ and $R_{\mathbf{p}} = \frac{1}{h} I$, with I being the identity matrix, and take the gradients of the approximated integrals to find

$$\nabla_u \mathcal{H}_{\mathbf{p}} = -h \left(\frac{\eta}{2} \mathbf{u}^2 - \gamma^2 \delta_c^2 \mathbf{u} \right),$$
$$\nabla_u \mathcal{V}_{\mathbf{p}} = -h \, \nu \, \delta_c^2 \mathbf{u},$$

where $\mathbf{u}^2$ and $\mathbf{u}^3$ denote the element-wise square and cube of $\mathbf{u}$, and $\delta_c^2 := \delta_f \delta_b$ denotes the second-order difference operator approximating the second derivative by $\delta_c^2 u_i = (u_{i+1} - 2u_i + u_{i-1})/(2h)$. Observe that $\frac{\delta \mathcal{H}_{\mathbf{p}}}{\delta \mathbf{u}}(\mathbf{u}) = \frac{1}{h} \nabla_{\mathbf{u}} \mathcal{H}_{\mathbf{p}}(\mathbf{u})$ and $\frac{\delta \mathcal{V}_{\mathbf{p}}}{\delta \mathbf{u}}(\mathbf{u}) = \frac{1}{h} \nabla_{\mathbf{u}} \mathcal{V}_{\mathbf{p}}(\mathbf{u})$ are consistent discrete approximations of (11) and (12). Moreover, they are second-order approximations of these variational derivatives, even though (14) and (15) are only first-order approximations of the integrals (9) and (10).

3.2. Restricting the class by imposing assumptions

Without imposing any further restrictions, the formulation (7) can be applied to any PDE that is first-order in time and will not be unique for any system. Any contribution from the two first terms on the left-hand side could also be expressed in f, and even if we restrict this term, the operators S and R are generally not uniquely defined for a corresponding integral. In the remainder of this paper, we will consider the cases where the operators S and R are linear and independent of x and u, we assume R to be symmetric, and we will not let f depend on derivatives of u. Furthermore, we apply the symmetric positive semi-definite operator A to the equation and require that this commutes with R and S. We redefine $S := AS$, $R := AR$ and $f(u,x,t) := Af(u,x,t)$, and thus get

$$Au_t = S\frac{\delta \mathcal{H}}{\delta u}[u] - R\frac{\delta \mathcal{V}}{\delta u}[u] + f(u,x,t), \tag{16}$$

where the new S is still skew-symmetric and the new R is still symmetric and positive semi-definite.

In the following we will denote the identity operator by I and the zero operator by 0, so that $Iv = v$ and $0v = 0$ for any $v \in L_2$. We note that the zero operator is positive semi-definite, symmetric and skew-symmetric, while the identity operator is symmetric and positive semi-definite, but not skew-symmetric.

4. The PHNN model for PDEs

Since we assume that the operators A, S and R are independent of x and u, the discretization of these operators will necessarily result in circulant matrices, given that u is periodic. That is, they can be viewed as discrete convolution operators. We thus set $\hat{A}_\theta^{[k_1]}$, $\hat{S}_\theta^{[k_2]}$ and $\hat{R}_\theta^{[k_3]}$ to be trainable convolution operators, where k_1, k_2 and k_3 denote the kernel sizes, and we impose symmetry on $\hat{A}_\theta^{[k_1]}$ and $\hat{R}_\theta^{[k_3]}$ and skew-symmetry on $\hat{S}_\theta^{[k_2]}$. Furthermore, we let $\hat{\mathcal{H}}_\theta$ and $\hat{\mathcal{V}}_\theta$ be two separate neural networks that take input vectors of length M, the number of spatial discretization points, and output a scalar. The neural network $\hat{f}_\theta$ can take input vectors representing both u, x and t and outputs a vector of length M.

The full pseudo-Hamiltonian neural network model for PDEs is then given by

$$\hat{g}_\theta(u,x,t) = (\hat{A}_\theta^{[k_1]})^{-1}\Big(\hat{S}_\theta^{[k_2]}\nabla\hat{\mathcal{H}}_\theta(u) - \hat{R}_\theta^{[k_3]}\nabla\hat{\mathcal{V}}_\theta(u) + k_4\hat{f}_\theta(u,x,t)\Big), \tag{17}$$

where we also have introduced k_4, which should be 1 or 0 depending on whether or not we want to learn a force term. Given a set of N training points $\{(u^{j_n}, u^{j_n+1}, t^{j_n})\}_{n=1}^N$ varying across time and different stochastic realizations of initial conditions, we let the loss function be defined as

$$\mathcal{L}_{g_\theta}(\{(u^{j_n}, u^{j_n+1}, t^{j_n})\}_{n=1}^N) = \frac{1}{N}\sum_{n=1}^N \left| \frac{u^{j_n+1} - u^{j_n}}{\Delta t} - \hat{g}_\theta\Big(\frac{u^{j_n} + u^{j_n+1}}{2}, x, \frac{t^{j_n} + t^{j_n+1}}{2}\Big)\right|^2, \tag{18}$$

if the implicit midpoint integrator is used. For the experiments in Section 5.4 we use the fourth-order symmetric integrator introduced in [24], and the loss function is amended accordingly.

4.1. Implementation

PHNN is comprised of up to six trainable models; the framework is very flexible and assumptions may be imposed so one or more of the parts do not have to be learned. Moreover, careful considerations should be made on how to best model the different parts. In the following, we explain how we have set up the models in our code.

4.1.1. Modelling $\mathcal{H}$ and $\mathcal{V}$

The networks $\hat{\mathcal{H}}_\theta$ and $\hat{\mathcal{V}}_\theta$ take inputs of dimension M, the number of spatial discretization points, and consist of one convolutional layer with kernel size two followed by linear layers corresponding to convolutional layers with kernel size one, and then in the last layer performs a summation of the M inputs to one scalar. Following each of the first two layers, we apply the tanh activation function. To impose the periodic boundary conditions, we pad the input to the convolutional layer by adding $u(P) = u(0)$ at the end of the array of the discretized u. A similar technique was suggested in [25], although they use a kernel of size three on the first convolutional layer. We opt to have a smaller filter, since kernel size two is sufficient to learn the forward difference approximation of the first derivative in the integrals, which in turn is sufficient to obtain second order approximation of the resulting variational derivative; this is shown for the KdV–Burgers equation in Example 2. If we want to be able to learn derivatives of order two in the integral, we would need kernel size three, and to pad the input with one element on each side. If we want to learn derivatives of order three or four, or if we want to learn third- or fourth-order approximations of the derivatives, we would need the kernel size of the convolutional layer to be five, and to pad the input by two elements on each side. This adjustment can easily be made in our code. For the examples in this paper, we would not gain anything by increasing the kernel size, because we only have up to first derivatives in the integrals and because the training data is generated using second order spatial discretizations. On the other hand, having a kernel of size two simplifies the learning and may facilitate superior performance over a model that does not rely on a pseudo-Hamiltonian structure and have to approximate up to third derivatives by convolutional neural networks.

4.1.2. Modelling A, S, R and f

In (17), $k = (k_1, k_2, k_3, k_4)$ are hyperparameters that determine the expressiveness of the model. Setting $k_1 = k_2 = k_3 = M$ and $k_4 = 1$ means that we can approximate the general system (16), while setting $k_1 = k_3 = 1$, $k_2 = 3$ and $k_4 = 0$ would be sufficient to learn a model for the discretized KdV–Burgers system (8). In fact, if we set $k_3 = 3$, the skew-symmetric operator $\hat{S}_\theta^{[k_2]}$ is uniquely defined up to a multiplicative constant, since we require $w_0 = 0$ and $w_1 = -w_{-1}$. Moreover, since this constant would only amount to a scaling between the operator and the discrete variational derivative it is applied to, $\hat{S}_\theta^{[k_2]}$ does not have to be trained in this case; determining w_1 would just lead to a scaling of the second-order approximation of the first derivative in space that could be compensated by a scaling of $\mathcal{H}$. Similarly, if the kernel size of A or R is 3, we could set $w_0 = 1$ and learn a single parameter $w_1 = w_{-1}$ for each of these when training the model. This corresponds to learning a linear combination of the identity and the second-order approximation of the second derivative in space.

We model f by the neural network $\hat{f}_\theta$ that may take either of the variables u, x and t as input. This has three linear layers, i.e. convolutional layers with kernel size one, with the tanh activation function after each of the first two. If $\hat{f}_\theta$ depends on x, periodicity on the domain $[0, P]$ is imposed in a similar fashion as suggested in [53–55] for hard-constraining periodic boundary conditions in physics-informed neural networks and DeepONet. That is, we replace the input x by the first two Fourier basis functions, $\sin\left(\frac{2\pi}{P}x\right)$ and $\cos\left(\frac{2\pi}{P}x\right)$, which is sufficient for expressing any x-dependent periodic function.

In the numerical experiments of the next section, we do not consider systems where A and R are anything other than linear combinations of the identity and the spatial second derivative, or S is anything other than the first derivative in space. Thus we set $k = [3, 3, 3, 1]$ in our most general model, which is expressive enough to learn all the systems we consider. We also consider what we call *informed* PHNN models where we assume to have prior knowledge of the operators, affecting k, and also what variables f depend on.

4.1.3. Leakage of constant

If R is the identity, or a linear combination of the identity with differential operators, the separation between the dissipation term and the external force term in (16) is at best unique up to a constant, which means that there may be leakage of a constant between the two last terms of the PHNN model. Hence, we must make some assumptions about these terms to separate them as desired. If we want the external force term to be small, we may use regularization and penalize large values of $\|\hat{f}_\theta\|$ during training. The option to do this is implemented in the `phlearn` package. However, for the numerical experiments in the next section, we have instead opted to assume that the dissipative term should be zero for the zero solution, and thus correct the two terms in question after training so that it adheres to this without changing the full model. That is, if we have the model

$$\hat{g}_\theta^{\text{pre}}(u, x, t) = (\hat{A}_\theta^{[k_1]})^{-1}\left(\hat{S}_\theta^{[k_2]}\nabla\hat{\mathcal{H}}_\theta(u) - \hat{R}_\theta^{[k_3]}\nabla\hat{\mathcal{V}}_\theta^{\text{pre}}(u) + k_4\hat{f}_\theta^{\text{pre}}(u, x, t)\right), \tag{19}$$

when the last training step is performed, we set

$$\begin{aligned}\nabla\hat{\mathcal{V}}_\theta(u) &:= \nabla\hat{\mathcal{V}}_\theta^{\text{pre}}(u) - k_4\nabla\hat{\mathcal{V}}_\theta^{\text{pre}}(0),\\ \hat{f}_\theta(u, x, t) &:= \hat{f}_\theta^{\text{pre}}(u, x, t) - \hat{R}_\theta^{[k_3]}\nabla\hat{\mathcal{V}}_\theta^{\text{pre}}(0)\end{aligned}$$

to get our final model (17), which is equivalent to (19). Then we may remove the dissipation or external forces from the model simply by setting $k_3 = 0$ or $k_4 = 0$. Note, however, that this correction may not work as expected if the zero solution is far outside the domain of the training data, since the neural network $\hat{\mathcal{V}}_\theta^{\text{pre}}$ like most neural networks generally extrapolates poorly. In that case, regularization is to be preferred.

4.1.4. Algorithms

We refer to Algorithm 1 and Algorithm 2 for the training of the PHNN and baseline models, respectively.

Data: Observations $D = \{(t_1, \vec{x}^1, \vec{u}^1), \ldots, (t_N, \vec{x}^N, \vec{u}^N)\}$
Data: Number of epochs K
Data: Batch size M_b
Data: Initial CNN $\hat{H}_\theta, \hat{V}_\theta$
Data: Initial DNN $\hat{f}_\theta$
Data: Matrices $\hat{A}_\theta^{[k_1]}$, $\hat{S}_\theta^{[k_2]}$ and $\hat{R}_\theta^{[k_3]}$
Data: g_θ defined in (17)
Data: Loss function $\mathcal{L}_{g_\theta}$ defined in (18)
Result: Parameters θ for g_θ
for k *in* $1 \ldots K$ **do**
  for *batch in Batches* **do**
    $B := \{(u^{j_m}, u^{j_m+1}, t^{j_m})\}_{m=1}^{M_b} \leftarrow$ DrawRandomBatch(D, M_b);
    Step using $\mathcal{L}_{g_\theta}(B)$ and $\nabla_\theta\mathcal{L}_{g_\theta}(B)$
  end
end

Algorithm 1: The training phase of the PHNN algorithm.

Data: Observations $D = \{(t_1, \vec{x}^1, \vec{u}^1), \dots, (t_N, \vec{x}^N, \vec{u}^N\}$
Data: Number of epochs K
Data: Batch size M_b
Data: Initial CNN g_θ
Data: Loss function $\mathcal{L}_{g_\theta}$ defined in (18)
Result: Parameters θ for g_θ
for k *in* $1 \dots K$ **do**
  for *batch in Batches* **do**
    $B := \{(u^{j_m}, u^{j_m+1}, t^{j_m})\}_{m=1}^{M_b} \leftarrow$ DrawRandomBatch(D, M_b);
    Step using $\mathcal{L}_{g_\theta}(B)$ and $\nabla_\theta \mathcal{L}_{g_\theta}(B)$
  end
end

Algorithm 2: The training phase of the baseline algorithm.

5. Numerical experiments

In this section, we test how PHNN models perform on a variety of problems with different properties. To our knowledge, there are no existing methods in the literature for which it is natural to compare the PHNN method across a variety of PDEs. We consider thus first a purely Hamiltonian PDE problem, to be able to compare PHNN to the method of Matsubara et al. [25]. We also compare our models to the system identification method PDE-FIND [28] for this problem. For problems with damping and external forces, we have developed our own baseline model that does not have the pseudo-Hamiltonian structure but is otherwise as similar as possible to the PHNNs and is trained in the same way. We test either two or three PHNNs for each problem, in addition to a baseline model. The models we test on all problems are:

- PHNN (general): A PHNN model with kernel sizes $k = [3, 3, 3, 1]$ and an $\hat{f}_\theta$ that depends on u, x and t;
- PHNN (informed): A PHNN model where the operators A, S and R are known a priori and $\hat{f}_\theta$ depends only on the variable(s) which f depend on;
- Baseline: A model consisting of one neural network that takes u, x and t as input, where the output is of the same dimension as u. The network consists of two parts: first a five-layer deep neural network with a hidden dimension of 20 and the tanh activation function, then a convolutional layer with kernel size five and activation function tanh, followed by two additional layers with hidden dimension 100 and the tanh activation function. The first five layers are meant to approximate any non-linear function (say $u \mapsto \frac{1}{2}u^2$ in the case of Burgers' equation), while the convolutional layer is supposed to represent a finite-difference approximation of the spatial derivatives. The baseline model needs a kernel size of five to approximate the third and fourth derivatives present in the first two resp. last example we consider.

The GitHub repository https://github.com/SINTEF/pseudo-hamiltonian-neural-networks includes notebooks to run experiments on all the systems we consider in the following. To reproduce the exact results we present in this section and the next, we refer the reader to https://doi.org/10.5281/zenodo.10419436.

5.1. The KdV equation

If $\nu = 0$ in (8), we get the KdV equation

$$u_t + \eta u u_x - \gamma^2 u_{xxx} = 0. \tag{20}$$

Furthermore, we let $\eta = 6$ and $\gamma = 1$ and assume periodic solutions $u(0, t) = u(P, t)$ on the domain $[0, P]$, with $P = 20$. We generate training data from initial conditions

$$u(x, 0) = 2 \sum_{l=1}^{2} c_l^2 \operatorname{sech}^2 \left(c_l \left(\left(x + \frac{P}{2} - d_l P \right) \bmod P - \frac{P}{2} \right) \right), \tag{21}$$

where c_1, c_2 and d_1, d_2 are randomly drawn from the uniform distributions $\mathcal{U}(\frac{1}{2}, 2)$ and $\mathcal{U}(0, 1)$ respectively. That is, the initial states are two waves of height $2c_1^2$ and $2c_2^2$ centred at $d_1 P$ and $d_2 P$, with periodicity imposed. The system is integrated from 20 different random initial states from time $t = 0$ to time $t = 0.2$. This is done with a time step $\Delta t = 0.0025$, but then only every fourth step is used as training data. The test data is obtained from 10 random initial states integrated and evaluated at every time step $\Delta t = 0.001$ to $t = 2$.

In this case, where the data actually represents a purely Hamiltonian PDE system, the general PHNN model performs much worse than the informed PHNN model, which in this case becomes a pure HNN model. The DGNet method of Matsubara et al. [25] can give accurate results, as is evident in Fig. 1, but it performs generally worse than our method on varied test data, and also requires much more time to train, as the numbers in Table 1 show. The sparse regression method PDE-FIND [28] is not able to find accurate models from the training data considered here.

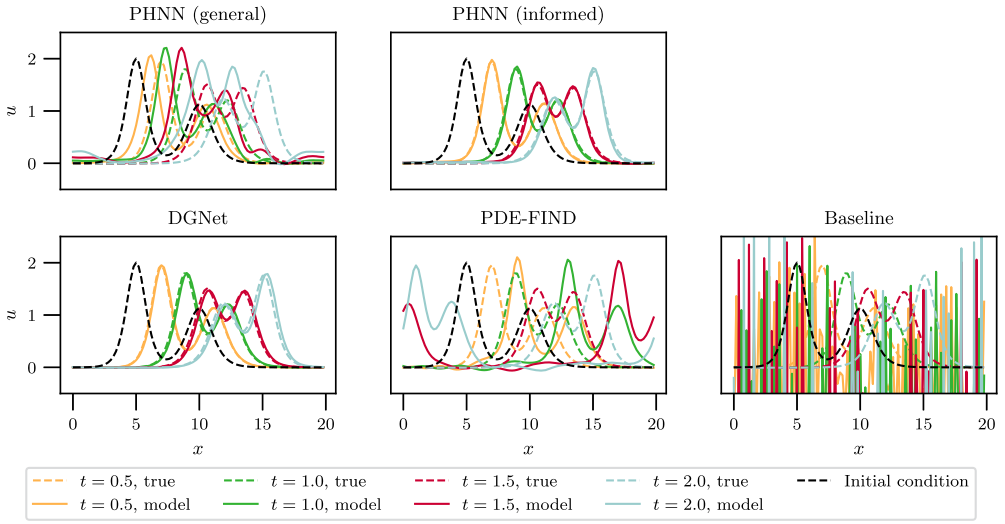


Fig. 1. Predictions of the KdV equation (20) by two PHNN models and our baseline model, compared to DGNet [25] and PDE-FIND [28]. The training data consist of 420 states, with 20 different initial conditions and 21 points equidistributed in time from $t = 0$ to $t = 0.2$, and the neural network models are all trained for 5000 epochs.

Table 1
Mean and standard deviation of the MSE from $t = 0$ to $t = 2$ for 10 models of each type tested on 10 different initial conditions, compared to the numerical solution of the exact KdV equation (20). The runtime provided is the average training time in seconds for each method, on two 2.4 GHz CPU cores of Intel Xeon Gold 6126.

	5000 epochs			20000 epochs		
	mean MSE	std MSE	Runtime	mean MSE	std MSE	Runtime
PHNN (general)	1.75e+02	3.62e+02	2691	6.52e+01	9.27e+01	10603
PHNN (informed)	1.34e+00	5.96e+00	1172	7.23e-01	3.70e+00	4587
DGNet [25]	4.58e+01	2.45e+02	9555	3.45e+01	1.99e+02	37227
PDE-FIND [28]	Inf	Inf	2	Inf	Inf	2
Baseline	1.34e+03	3.67e+03	1022	1.03e+03	2.20e+03	4245

5.2. The KdV–Burgers equation

Consider now the KdV–Burgers equation from Examples 1 and 2, but with external forces:

$$u_t + \eta(\frac{1}{2}u^2)_x - \nu u_{xx} - \gamma^2 u_{xxx} = f(x,t). \tag{22}$$

The spatial domain is still $[0, P]$ with $u(0,t) = u(P,t)$ and $P = 20$. We let $\eta = 6$, $\nu = 0.3$, $\gamma = 1$ and $f(x,t) = \sin\left(\frac{2\pi x}{P}\right)$, and generate again training data from random initial conditions (21).

We compare the general and informed PHNN models to the baseline model on (22) with

$$f(x,t) = \frac{3}{5}\sin\left(\frac{4\pi}{P}x - t\right). \tag{23}$$

Ten models of each type are trained using training sets consisting of 410 states, obtained from integrating 10 random initial states of the form (21) and evaluating the solution at every time step $\Delta t = 0.05$ until $t = 2$. We train on a larger time domain here than we did for the KdV equation to learn the explicit dependence of f on t.

Even when the baseline model is able to approximate the dynamics well, the training of the model generally converges more slowly than the PHNN models. After training 10 models of each type for 5000 epochs, the PHNN models are consistently outperforming the baseline model; see Figs. 2 and 3.

To test the accuracy of the models after the training has converged, we then train the PHNN models for 20 000 epochs and the baseline models for 50 000 epochs. At every epoch, the models are validated by integrating them to time $t = 2$ starting at three random initial states and calculating the average mean squared error (MSE) from these. The model with the lowest validation score after the last epoch is saved as the final model. When being tested on an initial state well within the domain the training data is sampled from, all models perform well; see Figs. 4 and 5.

When they are tested on a wide range of initial states, some of the models struggle to give stable and accurate solutions. We observe that the PHNN models are quite sensitive to variations in the initialization of the learnable parameters of the model, an issue

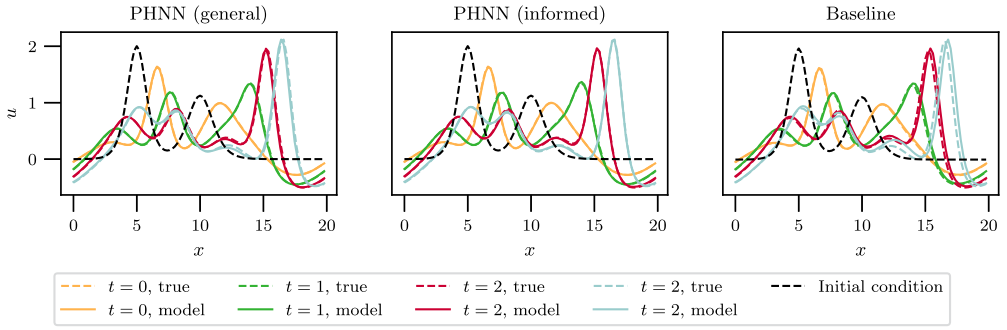


Fig. 2. Predictions of the forced KdV–Burgers system (22) with force (23), obtained from the best of 10 models of each model type, after being trained for 5000 epochs, as evaluated by the mean MSE at $t = 2$ on predictions from 10 random initial states.

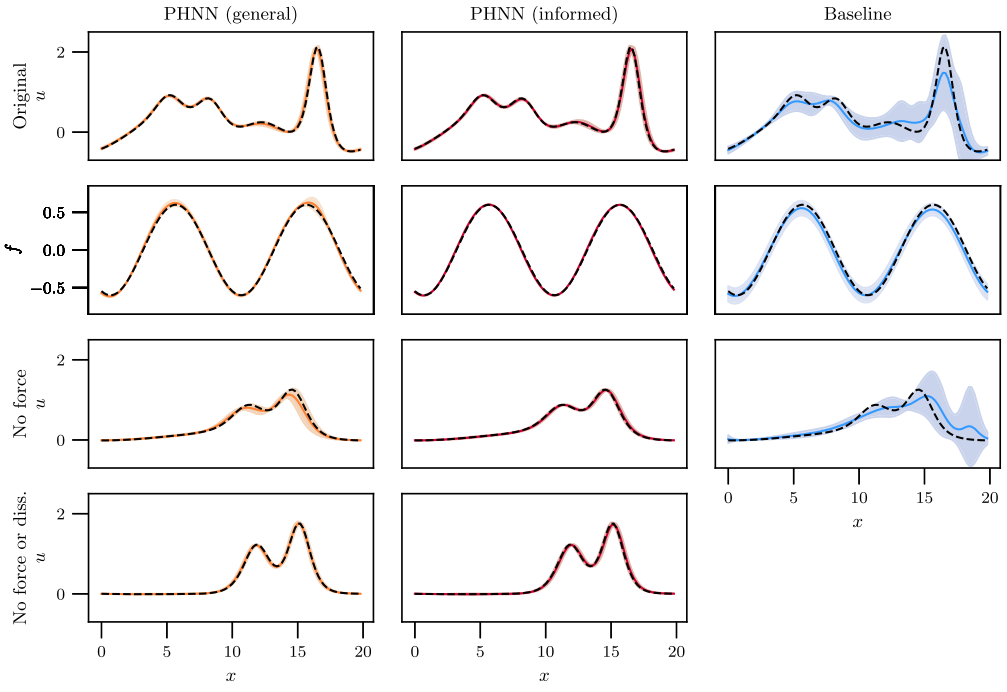


Fig. 3. Solutions of the various models, after being trained for 5000 epochs, of the forced KdV–Burgers system (22) with f given by (23) at time $t = 2$. The line and the shaded area are the mean resp. standard deviation of 10 models of each type. The dashed black line is the ground truth. *Upper row:* The original system (22) that the models are trained on. *Second row:* The learned force approximating f in (22). *Third row:* Predictions with the force f removed from the models. *Lower row:* Predictions with the external force and the dissipation term removed from the models.

we discuss further in Section 6.1. Hence we get a large average MSE from these models when applying them on 10 different initial states, as is evident from Table 2. However, the best PHNN models perform well on all the test cases; of the 30 models trained, 10 of each type, the seven models with the lowest average MSE on 10 test sets are all PHNN models. Thus it would be advisable to run several PHNN models with different initalizations of the neural networks and disregard those models who behave vastly different from the others. We demonstrate this by picking out the three most similar models of each type, as measured by their predictions on 10 different random initial conditions, and evaluate the mean error on those; see Table 2.

Figs. 3 and 5 demonstrate one of the main qualitative features of the PHNN models: we can remove the force and dissipation from the model and still get an accurate solution of the system without these. In these figures, we have also extracted the external force part from the baseline model by

$$\hat{f}_\theta(x,t) := \hat{g}_\theta(u,x,t) - \hat{g}_\theta(u,0,0). \tag{24}$$

This works here, since the external force is independent of the solution states and the integrals are zero when $u = 0$, but it would not be an option in general. Moreover, we note that there is no way to separate the conservation and dissipation terms of the baseline model. This can however be done with the PHNN models, so that we also have a model for the energy-preserving KdV equation.

Table 2
Mean and standard deviation of the MSE at $t = 2$, for 10 models of each type and for the three most similar of each type, trained on the KdV–Burgers equation with the external force (23).

	10 models		Three models	
	mean	std	mean	std
PHNN (general)	1.00e+01	1.42e+01	4.32e-01	6.09e-03
PHNN (informed)	3.42e+01	3.44e+01	4.25e-01	6.73e-03
Baseline	2.23e+00	1.61e+00	5.82e-01	1.53e-01

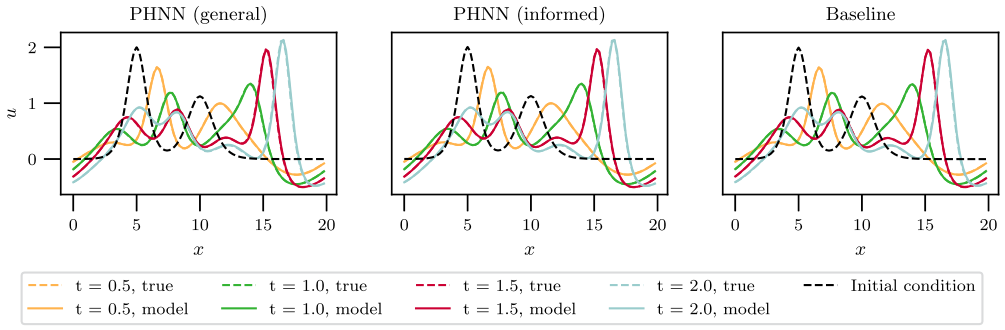


Fig. 4. Predictions of the forced KdV–Burgers system (22) with force (23), obtained from the best of 10 models of each model type, as evaluated by the mean MSE at $t = 2$ on predictions from 10 random initial states.

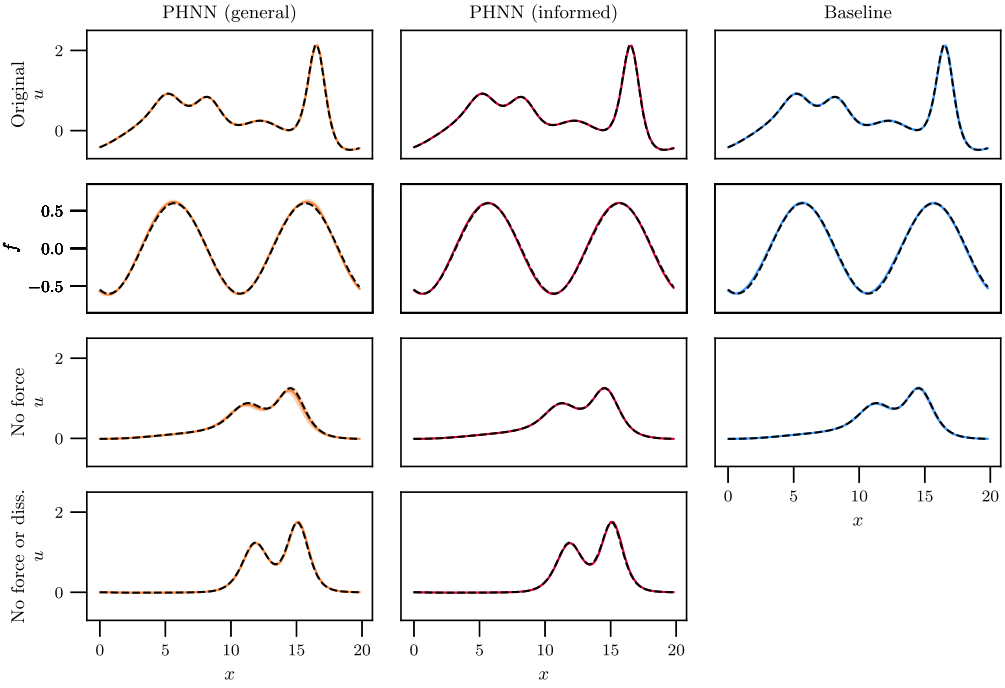


Fig. 5. Solutions of the various learned models of the forced KdV–Burgers system (22) with f given by (23) at time $t = 2$. The line and the shaded area, barely visible in these plots, are the mean resp. standard deviation of 10 models of each type. The dashed black line is the ground truth. *Upper row:* The original system (22) that the models are trained on. *Second row:* The learned force approximating f in (22). *Third row:* Predictions with the force f removed from the models. *Lower row:* Predictions with the external force and the dissipation term removed from the models.

When the external force is explicitly dependent on time, we are generally not able to learn a model that is accurate beyond the temporal domain of the training data. For autonomous systems, we can make due with less training data. Consider thus instead of (23) the external force

$$f(x) = \frac{3}{5}\sin\left(\frac{4\pi}{P}x\right). \tag{25}$$

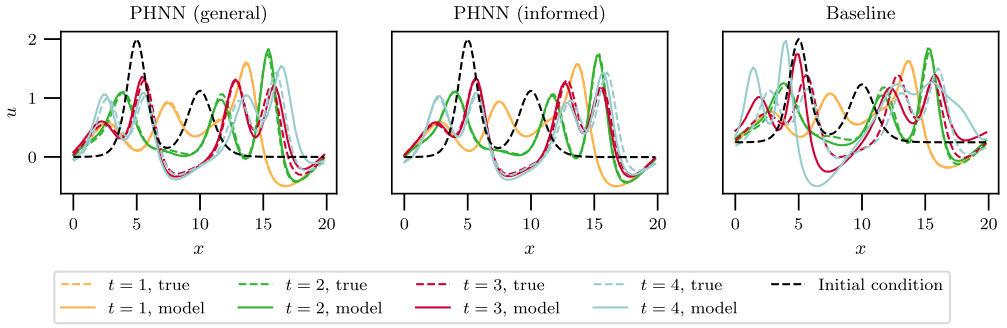


Fig. 6. Predictions of the forced KdV–Burgers system (22) with f given by (25) obtained from the best of 10 models of each model type, as evaluated by the mean MSE at $t = 0.5$ on predictions from 10 random initial states.

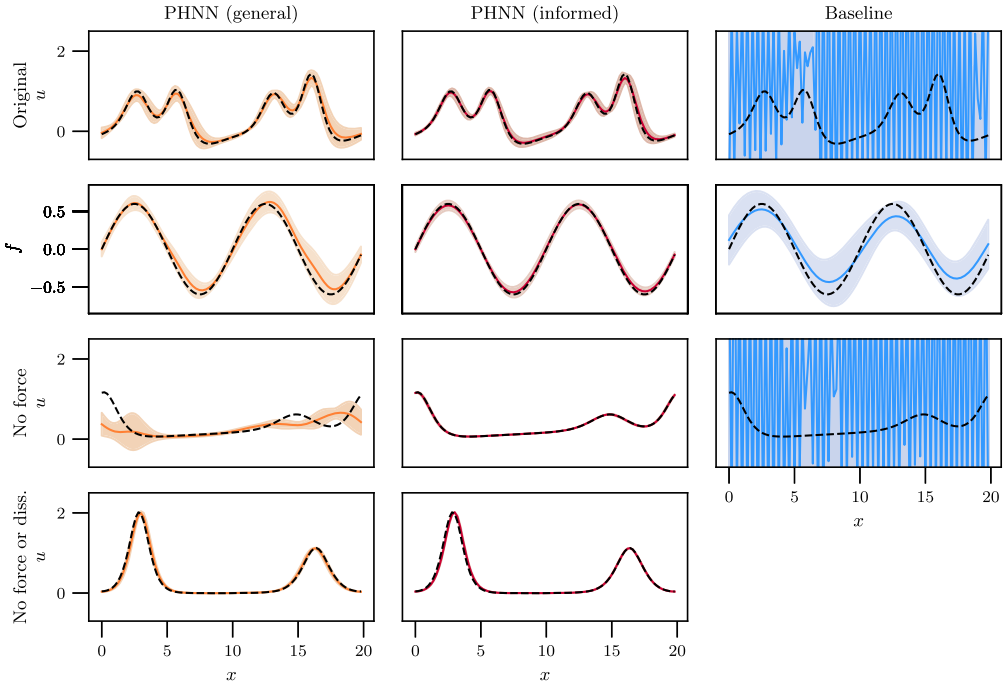


Fig. 7. Solutions of the various learned models of the forced KdV–Burgers system with f given by (25). The line and the shaded area is the mean resp. standard deviation of predictions at $t = 4$ of 10 models of each type. The dashed black line is the ground truth. *Upper row:* The original system. *Second row:* The learned force approximating f in (25). *Third row:* Predictions with the external force f removed from the models. *Lower row:* Predictions with the force and the dissipation term removed.

We train models for (22) with this f, where now we have training sets consisting of 60 states, from solutions obtained at every time step $\Delta t = 0.1$ from $t = 0$ to $t = 0.5$. As can be seen in Figs. 6 and 7, the PHNN models perform better than the baseline model, and especially so with increasing time. The worst-performing baseline models become unstable before the final test time $t = 4$. We also see that the most general PHNN model struggles to correctly separate the external force from the viscosity term with this amount of training data. However, this is not an issue when the model is informed that the force is purely dependent on the spatial variable.

5.3. The forced BBM equation

The Benjamin–Bona–Mahony (BBM) equation was introduced as an improvement on the KdV equation for modelling waves on a shallow surface [56,57]. We consider this equation with a time- and state-dependent source term:

$$u_t - u_{xxt} + u_x + uu_x = f(u,t), \tag{26}$$

which can be written on the form (16) with $A = 1 - \frac{\partial^2}{\partial x^2}$, $S = \frac{\partial}{\partial x}$ and $R = 0$. This requires

Table 3
Mean and standard deviation of the MSE at $t = 10$, for 10 models of each type and for the three most similar of each type, trained on the BBM equation with an external force.

	10 models		Three models	
	mean	std	mean	std
PHNN (general)	6.13e-01	8.04e-01	1.95e-01	1.02e-01
PHNN (no diss. term)	1.19e-01	7.75e-02	4.79e-02	2.55e-02
PHNN (informed)	1.46e-01	4.10e-01	3.45e-03	3.33e-03
Baseline	4.81e+00	4.79e+00	8.61e-01	3.29e-01

$$\mathcal{H} = \frac{1}{2}\int_{\Omega} \left(u^2 + \frac{1}{3}u^3\right) \mathrm{d}x.$$

As for the KdV–Burgers equation, we train the model on a forced system starting with a two-soliton initial condition. In this case, the initial states are given by

$$u(x,0) = 3\sum_{l=1}^{2}(c_l - 1)\,\mathrm{sech}^2\left(\frac{1}{2}\sqrt{1-\frac{1}{c_l}}\left(\left(x + \frac{P}{2} - d_l P\right) \bmod P - \frac{P}{2}\right)\right), \tag{27}$$

i.e. two waves of amplitude $3(c_1 - 1)$ and $3(c_2 - 1)$ centred at $d_1 P$ and $d_2 P$, where c_1, c_2 and d_1, d_2 are randomly drawn from the uniform distributions $\mathcal{U}(1,4)$ and $\mathcal{U}(0,1)$ respectively, and with periodicity imposed on $\Omega = [0, P]$. We set $P = 50$ in the numerical experiments. Furthermore, we let

$$f(u,t) = \frac{1}{10}\sin(t)u. \tag{28}$$

In addition to the three models described in the introduction of this section, we also test a model that is identical to the most general PHNN model except that it does not include a dissipation term. We do this because it is not clearly defined whether or how (28) should be separated into a term that is constantly dissipative and one that is not. The most general model does learn that the system has a non-zero dissipative term; however, this term added to the learned force term is close to the ground truth force term. This is due to a leakage of a term αu for some random constant α between the terms, similar to the constant leakage described in Section 4.1.3, so that we learn an approximated integral $\hat{\mathcal{V}}_\theta = \alpha\frac{\Delta x}{2}\sum_{i=0}^{M} u_i^2$ with $\hat{R}_\theta^{[3]} = I$ and a corresponding external force $\hat{f}_\theta = (\frac{1}{20}\sin(t) + \alpha)u$. This leakage could be combated by regularization, i.e. by penalizing the mean absolute value of the dissipation term. We do not do that in the numerical experiments presented here, but instead opt to also learn a model without the dissipation term and compare to this.

For every type of model, we train 10 distinct models with random initializations for a total of 20 000 epochs for the PHNN models and 50 000 epochs for the baseline model. We use training data comprising 260 states, obtained from integrating 10 randomly drawn initial states with time step $\Delta t = 0.4$ from time $t = 0$ to time $t = 10$. A validation score at each epoch is generated by integrating the models to time $t = 1$ starting at three initial states and calculating the mean MSE, and then the model with the lowest validation score is kept. After training is done, we integrate the models starting from 10 new arbitrary initial conditions to determine the average MSE at $t = 10$. By the error of the models as reported in Table 3, we see that all PHNN models perform better than the baseline model. Interestingly, the average MSE is lowest for the PHNN model where A and S has to be learned but R is known to be zero. However, the best PHNN model of all 30 models trained is one of those informed of these operators, as seen in Fig. 8. Note that the baseline model cannot be expected to learn a perfect model for this example, since the discrete approximation of $A^{-1}S = (1 - \frac{\partial^2}{\partial x^2})^{-1}\frac{\partial}{\partial x}$ used when generating the training data is a discrete convolution operator with kernel size bigger than five. The PHNN models are more stable and behave especially well with increasing time compared to the baseline model. Beyond $t = 10$, the accuracy of all models quickly deteriorates. We attribute this to the poor extrapolation abilities of neural networks; the models are not able to learn how the time-dependent f behaves beyond the temporal domain $t \in [0, 10]$ of the training data. Fig. 9 shows the external forces learned by the PHNNs, and how well the models predict the system with these forces removed. For the general PHNN model, we have in this case also removed the dissipation term.

5.4. The Perona–Malik equation

In addition to modelling physical systems, PDEs can be used for image restoration and denoising. For instance, if the heat equation is applied to a greyscale digital image, where the state u gives the intensity of each pixel, it will smooth out the image with increasing time. The Perona–Malik equation for so-called anisotropic diffusion is designed to smooth out noise but not the edges of an image [58]. Several variations of the equation exist. We consider the one-dimensional case, with a space-dependent force term, given by

$$u_t + \left(\frac{u_x}{1 + u_x^2}\right)_x = f(x). \tag{29}$$

This is a PDE of the type (16) with $A = I$, $S = 0$ and $R = I$, and

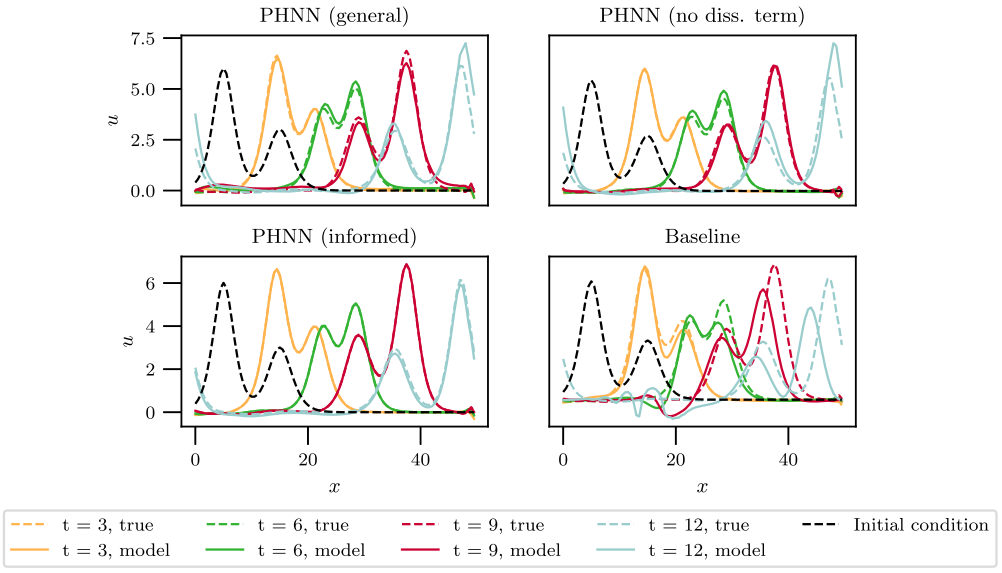


Fig. 8. Predictions of the forced BBM system (26) obtained from the best of 10 models of each model type, as evaluated by the mean MSE at $t = 10$ on predictions from 10 random initial states.

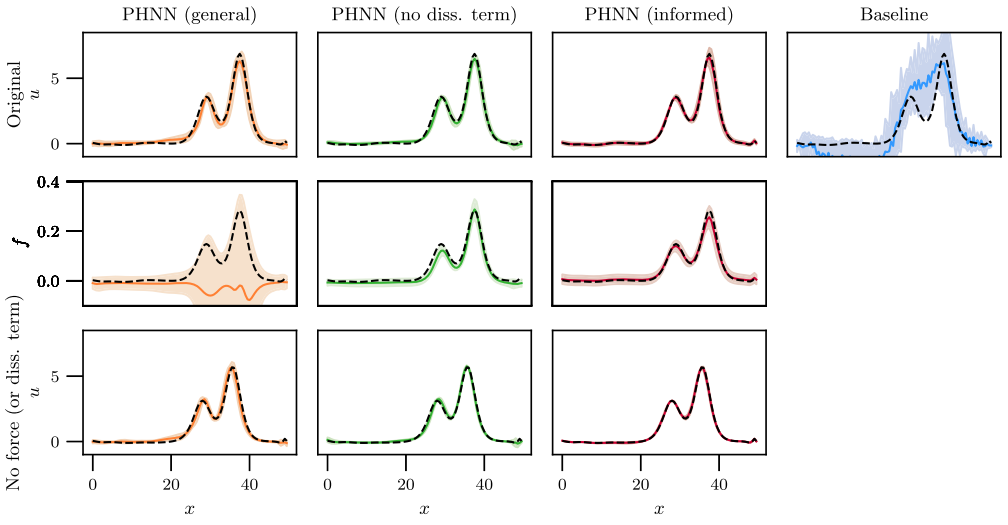


Fig. 9. Solutions of the learned BBM system obtained from the different models. The line and the shaded area is the mean resp. standard deviation of predictions at $t = 9$ of 10 models of each type. The dashed black line is the ground truth. *Upper row:* The original system (26) that the models are trained on. *Middle row:* The learned force approximating f in (26). *Lower row:* Predictions with the force f removed from the models.

$$\mathcal{V}[u] = \frac{1}{2}\int_{\Omega} \ln(1 + u_x^2)\,\mathrm{d}x.$$

Note that the equation can be written on the form $u_t = \frac{\partial}{\partial x}\phi[u] + f(x)$ for $\phi[u] = -\frac{u_x}{1+u_x^2}$, but this $\phi[u]$ is not the variational derivative of any integral. We consider (29) on the domain $[0, P]$ with $P = 6$, and set

$$f(x) = 10\sin\left(\frac{4\pi}{P}x\right) \tag{30}$$

for the following experiments. The initial conditions are given by

$$u(x,0) = a - \sum_{l=1}^{2}\left(h_l\Big(\tanh\left(b(x-d_l)\right) - \tanh\left(b(x-P+d_l)\right)\Big)\right) + c\sin^2(r\pi x)\sin(s\pi x) \tag{31}$$

where $a \in \mathcal{U}(-5,5)$, $b \in \mathcal{U}(20,40)$, $c \in \mathcal{U}(0.05,0.15)$, $d_l \in \mathcal{U}(0.3,3)$, $h_l \in \mathcal{U}(0.5,1.5)$, $r \in \mathcal{U}(0.5,3)$ and $s \in \mathcal{U}(10,20)$.

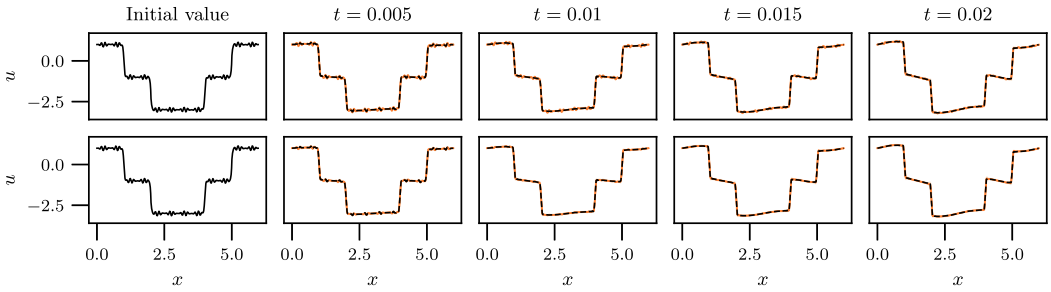


Fig. 10. The result at different times from integrating the mean of five general PHNN models trained on the Perona–Malik system (29) using two different integration schemes in the training. The solution of the learned models is in yellow, while the dashed black line is the solution of the exact PDE. *Upper row:* The second-order midpoint method. *Lower row:* The fourth-order symmetric method SRK4. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

Table 4
Mean and standard deviation of the MSE at end time $t = 0.02$, for 10 models of each type and for the three most similar of each type, trained on the Perona–Malik equation with an external force, and evaluated on predictions from 10 random initial states.

	10 models		Three models	
	mean	std	mean	std
PHNN (general)	5.36e-04	1.58e-04	4.01e-04	3.98e-05
PHNN (informed)	4.14e-03	1.03e-03	3.85e-03	5.03e-04
Baseline	3.09e-03	3.08e-04	3.04e-03	4.01e-05

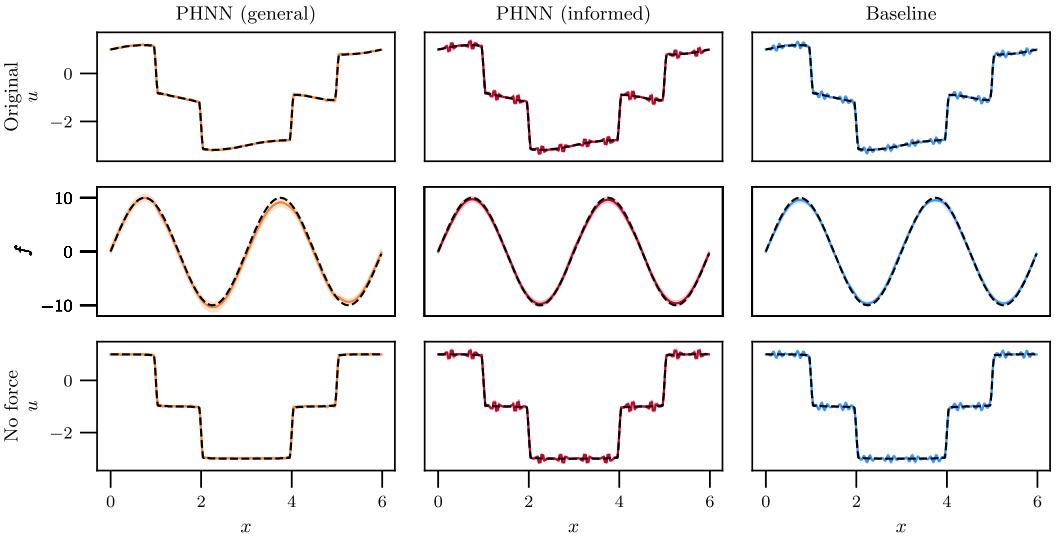


Fig. 11. Perona–Malik at time $t = 0.02$, models and exact. The line plot is the average of 10 models of each type, while the shaded region indicates the standard deviation. *Upper row:* The original system (32) that the models are trained on. *Middle row:* The learned force approximating (30). *Lower row:* Predictions with the force f removed from the models.

We train 10 models of each type for 10 000 epochs, on 20 pairs of data at time $t = 0$ and $t = 0.02$. This corresponds to the original noisy image and an image where the noise is almost completely removed, as judged by visual inspection. Because the step between these states is quite large, a high-order integrator is required to get an accurate approximation of the time-derivative. Indeed, models trained with the second-order implicit midpoint method fail to remove the noise as fast or accurately as the ground truth (29). Thus we use instead the fourth-order symmetric Runge–Kutta method (SRK4) introduced in [24]. This requires roughly four times the computational cost per epoch as using the midpoint method, but gives a considerably improved performance, as demonstrated in Fig. 10.

Table 4 and Fig. 11 report the result of applying the learned models on an original noisy state (31) with $a = 1$, $b = 30$, $c = 0.15$, $d_1 = 1$, $d_2 = 2$, $h_1 = h_2 = 1$, $r = 2$ and $s = 15$. Interestingly, the general PHNN model performs better than the informed one. Moreover, the PHNN models perform better when the kernel size of the first convolutional layer of $\hat{\mathcal{V}}_\theta$ is three instead of two. This indicates

Table 5
Mean and standard deviation of the MSE at $t = 0.02$, for 10 models of each type and for the three most similar of each type, trained on the forced Cahn–Hilliard system (32) and evaluated on predictions from 10 random initial states.

	10 models		Three models	
	mean	std	mean	std
PHNN (general)	1.14e+00	8.70e-01	4.61e-01	2.57e-01
PHNN (lean)	2.69e-01	1.36e-01	2.12e-01	3.42e-02
PHNN (informed)	2.49e-02	2.88e-04	2.48e-02	6.46e-05
Baseline	2.77e-01	7.75e-02	1.99e-01	2.24e-02

that the model does not learn the Perona–Malik equation but rather a different PDE that denoises the image. This may be as expected when we only train on initial states and end states. An odd-numbered filter size is the norm when convolutional neural networks are used for imaging tasks, since this helps to maintain spatial symmetry, and the improved performance with a kernel of size three in $\hat{\mathcal{V}}_\theta$ can perhaps be related to this.

5.5. The Cahn–Hilliard equation

The Cahn–Hilliard equation was originally developed for describing phase separation [59], but has applications also in image analysis, and specifically image inpainting [60,61]. Machine learning of pattern-forming PDEs, which include the Cahn–Hilliard and Allen–Cahn equations, has been studied in [62]. Results on applying PHNN to the Allen–Cahn equation is included in our GitHub repository. However, here we only consider the Cahn–Hilliard equation, with an external force, given by

$$u_t - (\nu u + \alpha u^3 + \mu u_{xx})_{xx} = f(u, x). \tag{32}$$

This is a dissipative PDE if the external force is zero, and it can be written on the form (16) with $A = I$, $S = 0$ and $R = -\frac{\partial^2}{\partial x^2}$, and

$$\mathcal{V}[u] = \frac{1}{2}\int_\Omega \left(\nu u^2 + \frac{1}{2}\alpha u^4 - \mu u_x^2\right) dx.$$

In the experiments, we set $\nu = -1$, $\alpha = 1$ and $\mu = -\frac{1}{1000}$, and

$$f(u, x) = \begin{cases} 30u & \text{if } 0.3 < x < 0.7, \\ 0 & \text{otherwise.} \end{cases}$$

The initial conditions of the training data are

$$u(x, 0) = \sum_{l=1}^{2} \left(a_l \sin\left(c_l \frac{2\pi}{P} x\right) + b_l \cos\left(d_l \frac{2\pi}{P} x\right)\right) \tag{33}$$

on the domain $[0, P]$ with $P = 1$, where a_l, b_l, c_l and d_l are random parameters from the uniform distributions $\mathcal{U}(0, \frac{1}{5})$, $\mathcal{U}(0, \frac{1}{20})$, $\mathcal{U}(1, 6)$ and $\mathcal{U}(1, 6)$, respectively.

In addition to the models described in the introduction of this section, we also train a "lean" model, with $k = [1, 0, 3, 1]$ but no prior knowledge of how R looks. For each model type, 10 randomly initialized models are trained for 20 000 (for the PHNN models) or 50 000 (for the baseline models) epochs on different randomly drawn data sets consisting of a total of 300 states, at times $t = 0$, $t = 0.004$ and $t = 0.008$. At each epoch, the model is evaluated by comparing to the ground truth solution of three states at $t = 0.008$, and the model with the lowest MSE on this validation set is kept. The resulting 10 models are then evaluated on 10 random initial conditions and the mean MSE in the last time step is calculated from this. The mean and standard deviation from all 10 models of each type, and the three most similar of each type, are given in Table 5. The prediction of the model of each type with the lowest mean MSE is shown in Fig. 12. In Fig. 13 we give the results of the average of all models, and the standard deviation. Here we also show the learned external force and the prediction when this is removed from the model. The initial state of the plots in Figs. 12 and 13 is (33) with $a_1 = 0.1$, $a_2 = 0.06$, $b_1 = 0.01$, $b_2 = 0.02$, $c_1 = 2$, $c_2 = 5$, $d_1 = 1$, $b_2 = 2$.

We see from Fig. 12 that the most general PHNN model may model the system moderately well, but it is highly sensitive to variations in the training data and the initialization of the neural networks in the model; from Fig. (13) and Table 5 we see that this model may produce unstable predictions. In any case, the PHNN models struggle to learn the external force of this problem accurately without knowing R, which we see by comparing the predictions of *PHNN (lean)* and *PHNN (informed)* in Fig. 13, where the difference between the models is that the former has to learn an approximation $\hat{R}_\theta^{[2]}$ of R and is not informed that f is not explicitly time-dependent.

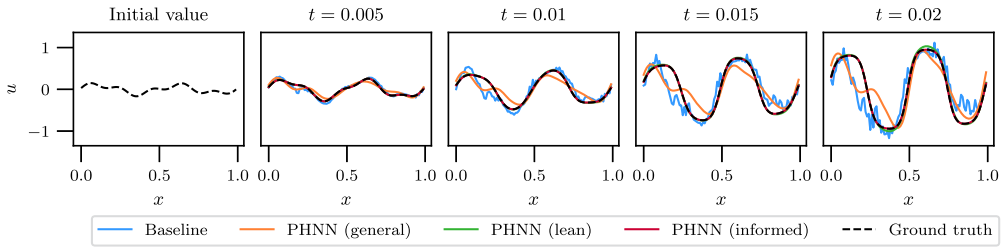


Fig. 12. Predictions of the forced Cahn–Hilliard system obtained from the best of 10 models of each model type, as evaluated by the mean MSE at $t = 0.02$ on predictions from 10 random initial states.

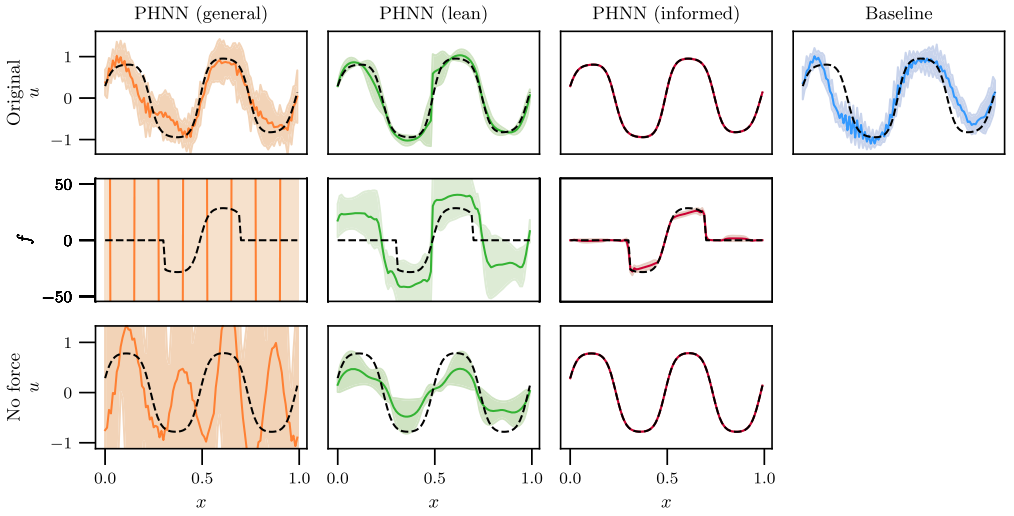


Fig. 13. Mean and standard deviation of predictions at $t = 0.02$ obtained from integrating 10 models of each type, for the Cahn–Hilliard problem (32). The dashed black line is the ground truth. *Upper row:* The original system (32) that the models are trained on. *Middle row:* The learned force approximating f in (32). *Lower row:* Predictions with the force f removed from the models.

6. Analysis of the models and further work

Here we provide some preliminary analysis of the PHNN models, which lays the groundwork for further analysis and development to be performed in the future.

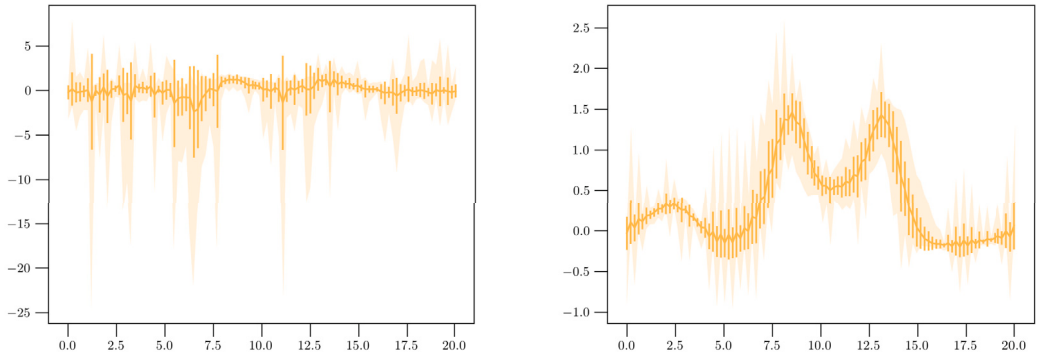
6.1. Stability with respect to initial neural network

The training of neural networks is often observed to be quite sensitive to the initial guesses for the weights and biases of the network. Here we test this sensitivity for both the general PHNN model and the baseline model on the KdV–Burgers experiment in Section 5.2. We keep the training data fixed and re-generate the initial weights for the neural networks and rerun the training procedure described in Algorithms 1 and 2. In Fig. 14 we plot the solution at the final time together with the standard deviation and the pointwise maximum and minimum values for both the baseline model and the PHNN approach, where the standard deviation and maximum/minimum is computed across an ensemble of different initial weights for the deep neural network. In Fig. 15 we plot the L^2 error at the final time step against the exact solution for varying number of epochs, where the shaded areas represent the maximum and minimum values of an ensemble of varying initial weights of the neural network.

6.2. Spatial discretization and training data

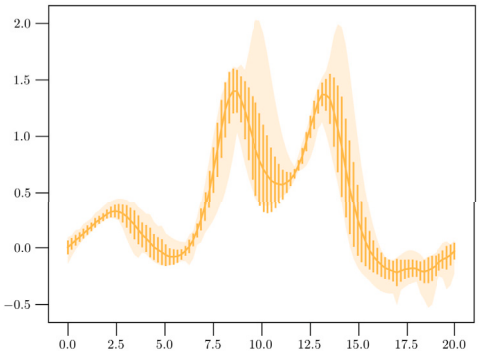
We will strive to develop PHNN further to make the models discretization invariant. For now, we settle with noting that this is already a property of our model in certain cases; a sufficiently well-trained informed PHNN model will be discretization invariant if the involved integrals do not depend on derivatives. Of the examples considered in this paper, that applies to the BBM equation, the inviscid Burgers' equation, and the Cahn–Hilliard equation if $\mu = 0$ in (32). Fig. 16 shows how the learned BBM system can be discretized and integrated on spatial grids different from where there was training data.

For the experiments in the Section 5, we generated training data using first and second order finite difference operators to approximate the spatial derivatives. For the experiments in subsections 5.2 to 5.5, we further trained our models on the same spatial grid as the data was generated on, thus making it possible to learn convolution operators of kernel size two or three that perfectly

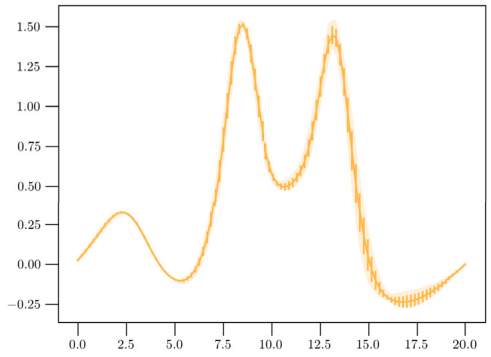


(a) Baseline, 1000 epochs, $T = 1$.

(b) PHNN, 1000 epochs, $T = 1$.



(c) Baseline, 10 000 epochs, $T = 1$.



(d) PHNN, 10 000 epochs, $T = 1$.

Fig. 14. Stability comparison of the baseline model and the general PHNN model. We retrain the models 20 times and compute the pointwise mean (plotted as a solid line) together with the standard deviation (plotted as error bars) and the pointwise maximum and minimum value (plotted as the shaded area).

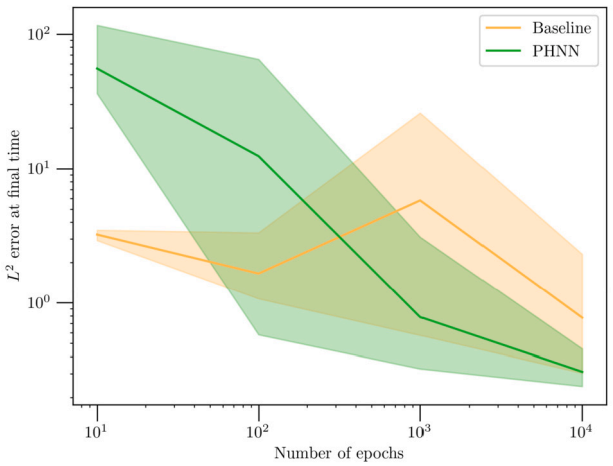


Fig. 15. Convergence of the solution with respect to the number of training epochs. Here, the shaded area represents the maximum and minimum errors obtained, and the solid middle line represents the mean value across different initial weights.

capture the operators in the data. In a real-world scenario, this is unrealistic, as we would have to deal with discretization of a continuous system in space, as well as in time. We chose to disregard this issue in the experiments, to give a clearer comparison between PHNN and the baseline model not clouded by the error from the spatial discretization that would affect both. However, for the experiments in Section 5.1 we tested our models on data generated on a spatial grid of four times as many discretization points, to not give the PHNNs and our baseline model an unfair advantage over the other methods. In this scenario, the data is generated from a more accurate approximation of the differential operators than what is possible to capture by the convolution operators. The

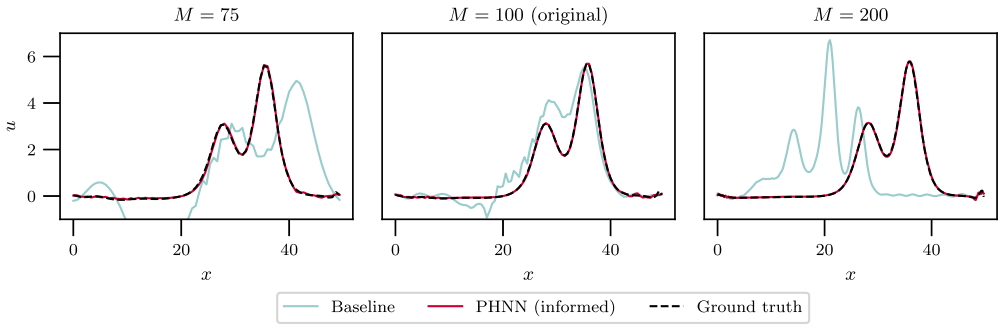


Fig. 16. The solution at $t = 10$ obtained from models of the BBM equation (26) with $f = 0$, learned from data discretized on $M + 1 = 101$ equidistributed points on the domain $[0, 50]$. The M above the plots indicates the number of equidistributed discretization points used in the integration. The PHNN model was trained for 10 000 epochs, while the baseline model was trained for 100 000 epochs, on 20 pairs of states at time $t = 0$ and $t = 0.4$, with initial states (27).

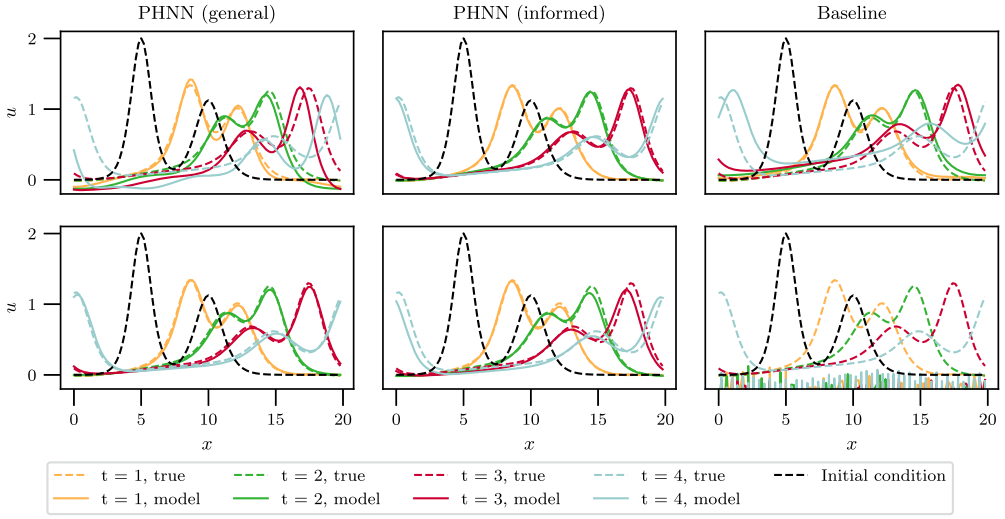


Fig. 17. Solution obtained from models of the KdV–Burgers equation (8), i.e. without a force term, learned from 410 training states, with 10 different initial conditions and points equidistributed in time between $t = 0$ and $t = 2$. *Upper row:* Models trained on data generated on a spatial grid with $M = 100$, same as used for training. *Lower row:* Models trained on data generated on a spatial grid with $M = 400$ and then downsampled to a grid with $M = 100$.

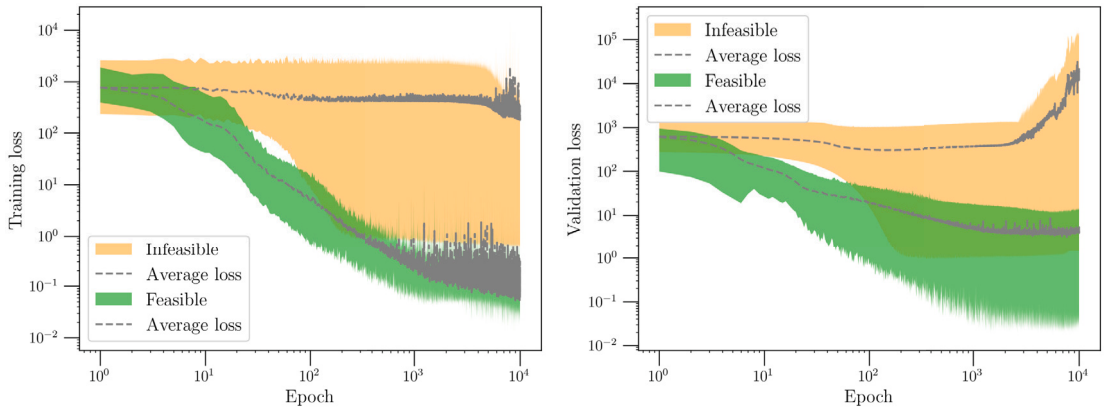
results in Section 5.1 indicate that PHNN tackles this challenge better than the baseline model, and we observe the same for the KdV–Burgers equation in Fig. 17. The PHNN models do appear to work well even with the introduction of approximation error in the spatial discretization. A more thorough study of this issue is required to gain a good understanding of how to best handle the spatial discretization.

6.3. Sensitivity to the kernel size hyperparameter

We note that for a new data set it is impossible to know the kernel size parameters a priori. However, in this subsection we will see that one can often distinguish feasible and infeasible kernel size parameters from simple heuristics applied to the training and validation loss.

First we generate 400 data points from the KdV equation as described in Section 5.2. We remind the reader that the kernel sizes $k = (k_1, k_2, k_3, k_4)$ are given as positive integers where the interesting values are typically 1 or 3. We then train the PHNN on 16 different kernel size tuples, that is we train on every possible kernel size in $\{1, 3\}^4$. Following the discussion in Section 4.1.2, we know that the feasible kernel sizes for the KdV equation are the tuples $k \in \{1, 3\}^4$ such that $k_2 = 3$. Therefore we define the *feasible* set of kernel sizes to be exactly those tuples where the second component is 3, and the infeasible set to be the complement of this set in $\{1, 3\}^4$.

We then train on this data using 200 points for training and 200 for validation. The result is plotted in Fig. 18. From the figure it becomes apparant that even by just looking at the training and validation data, there is a clear distinction between the feasible and infeasible kernel size hyperparameters, where infeasible kernel sizes simply do not reach convergence. We stress that this is done by only considering observation data. In other words, the training and validation loss acts as a discriminator between the feasible and infeasible kernel sizes.



(a) Training loss. (b) Validation loss.

Fig. 18. The loss per epoch of the training data (left) and validation data (right) for different kernel sizes for the KdV equation. The shaded area represents the interval between the minimum and maximum loss for the respective kernel size set.

An important consequence is that when encountering new data sets for which the kernel size parameter is not given, one can train on a larger set of kernel size parameters and select the ones where one does reach convergence in the training and validation loss.

6.4. Learning more complicated skew-symmetric operators

As we noted in Section 3.2, the general pseudo-Hamiltonian formulation (7) is not unique for any system. The term f ensures this, but even with $f = 0$ and $\mathcal{V} = 0$, the integral-preserving formulation

$$u_t = S(u^\alpha, x)\frac{\delta \mathcal{H}}{\delta u}[u] \tag{34}$$

is not unique. For a given $\mathcal{H}$, the corresponding skew-symmetric operator is not necessarily uniquely given. Furthermore, a PDE system may have several preserved integrals. For instance, the KdV equation (20) forms a completely integrable system, and can thus be written on the form (34) for infinitely many different $\mathcal{H}$ [63]. However, there are only two known *Hamiltonian* formulations of the KdV equation, where $S(u^\alpha, x)$ in addition to being skew-symmetric satisfies the Jacobi identity and is called the Poisson operator [50]. These are given by the pairs $S = \frac{\partial}{\partial x}$ and the energy functional (9), and $S = -\frac{1}{3}\eta(\partial_x u + u\partial_x) + \gamma^2\partial_{xxx}$ and the momentum

$$\mathcal{H} = \frac{1}{2}\int_\Omega u^2\, dx. \tag{35}$$

Because we restricted $\hat{S}_\theta^{[k_2]}$ to be constant in our models in Section 5, we achieved uniqueness and learned the formulation where $\mathcal{H}$ represents the energy. To learn the formulation where $\mathcal{H}$ represents momentum, we would have to let $\hat{S}_\theta^{[k_2]}$ depend on u, and we would need $k_2 \geq 5$ to learn the third spatial derivative. Furthermore, we would need to restrict the kernel of the convolutional layer in $\hat{\mathcal{H}}_\theta$ to be of size 1, so that this could not learn the energy functional. That is, the alternative Hamiltonian formulation could be learned by restricting $\hat{\mathcal{H}}_\theta$ more and $\hat{S}_\theta^{[k_2]}$ less. An exploration of using our models to learn alternative pseudo-Hamiltonian formulations of the same system is a planned future direction.

Such an exploration will also involve considering more PDE systems. One interesting candidate is the modified Korteweg–de Vries (mKdV) equation [64]

$$u_t + \eta u^2 u_x - \gamma^2 u_{xxx} = 0, \tag{36}$$

which introduces a more complicated S yet. For the momentum (35), the corresponding Poisson operator is in this case $S = -\frac{2}{3}\eta\partial_x u\partial_x^{-1}u\partial_x + \gamma^2\partial_{xxx}$, and thus includes both derivatives, an antiderivative and the system state u [65]. Setting $k_2 = M$ and letting $\hat{S}_\theta^{[M]}$ depend on u, it would be expressive enough to learn a consistent discretization of this S. How to impose uniqueness on this formulation is not trivial, and one of the things we will investigate in future work.

6.5. Proof of convergence in the idealized case

In this section we show a simplified error estimate for learning the right hand side of an ODE. Consider thus a model ODE of the form

$$\begin{cases} \dot{u}(t) &= g(u(t), t) \\ u(0) &= u_0, \end{cases} \tag{37}$$

where $u : [0, T) \to \mathbb{R}^M$ and $g : \mathbb{R}^M \to \mathbb{R}^M$. Note that the spatially discretized equation (13) can be cast into this form. In a certain sense, both the baseline model and the PHNN model tries to identify g by minimizing the L^p-norm of the observations $u(t^j)$ and the predictions $u_\theta(t^j)$. On a high level, this gives us a sequence $u_\theta \to u$ in L^p, but as is well-known, this would not be enough to conclude anything about the convergence of $g_\theta \to g$, since L^p convergence in general does not imply convergence of the derivatives. However, by utilizing the fact that we have a certain control over the discretized temporal derivatives in the learning phase, we can show that the $g_\theta \to g$ in the same L^p norm, *provided the training loss is small enough*. The following theorem makes this precise.

Theorem 1. *Let $\Delta t > 0$, and $g, \tilde{g} : \mathbb{R}^M \to \mathbb{R}^M$. Assume that $u : [0, T) \to \mathbb{R}^M$ solves (37) and that $\tilde{u}^1, \dots, \tilde{u}^N \in \mathbb{R}^M$ obey*[1]

$$\frac{\tilde{u}^{j+1} - u^j}{\Delta t} = \tilde{g}(u^j) \qquad \text{for } j = 0, \dots, N-1. \tag{38}$$

Then,

$$\left(\Delta t \sum_{j=1}^{N-1} \left(g(u(t^j), t^j) - \tilde{g}(u^j, t^j) \right)^p \right)^{1/p} \leq \frac{1}{\Delta t} \left(\sum_{j=1}^{N} \Delta t \left| u^j - \tilde{u}^j \right|^p \right)^{1/p} + C_g \Delta t.$$

Proof. Define $u^0, \dots, u^N \in \mathbb{R}^M$ as

$$u^j := u(t^n) \qquad j = 1, \dots, N. \tag{39}$$

By a Taylor expansion, we have

$$\frac{u^{j+1} - u^j}{\Delta t} = g(u^n) + \left(\left[\frac{\partial g}{\partial u}(u(\xi), \xi) \right] g(u(\xi, \xi)) + \frac{\partial g}{\partial t}(u(\xi), \xi) \right) \Delta t$$

for $\xi \in [t^j, t^{j+1}]$, $j = 0, \dots, N-1$. Hence, we get

$$\left(\sum_{j=0}^{N-1} \Delta t \left| g(u(t^j), t^j) - \tilde{g}(u^j, t^j) \right|^p \right)^{1/p} \leq \left(\sum_{j=1}^{N} \Delta t \left| \frac{u^j - u^{j-1}}{\Delta t} - \frac{\tilde{u}^j - u^{j-1}}{\Delta t} \right|^p \right)^{1/p} + C_g \Delta t.$$

We furthermore have

$$\begin{aligned} \left(\sum_{j=1}^{N} \Delta t \left| \frac{u^j - u^{j-1}}{\Delta t} - \frac{\tilde{u}^j - u^{j-1}}{\Delta t} \right|^p \right)^{1/p} &= \left(\sum_{j=1}^{N} \Delta t \left| \frac{u^j - \tilde{u}^j}{\Delta t} - \frac{u^{j-1} - u^{j-1}}{\Delta t} \right|^p \right)^{1/p} \\ &= \left(\sum_{j=1}^{N} \Delta t \left| \frac{u^j - \tilde{u}^j}{\Delta t} \right|^p \right)^{1/p} \\ &= \frac{1}{\Delta t} \left(\sum_{j=1}^{N} \Delta t \left| u^j - \tilde{u}^j \right|^p \right)^{1/p}. \quad \square \end{aligned}$$

Remark 1. We note that (38) means that the sequence $\{\tilde{u}^j\}_j$ is the learning data obtained by either the baseline or PHNN algorithm using the forward Euler integrator.

Remark 2. In the above theorem, $\left(\sum_{i=1}^{N} \Delta t \left| u^j - \tilde{u}^i \right|^p \right)^{1/p}$ is proportional to the training loss. In other words, the error in the approximation of g we get is bounded by $1/\Delta t \cdot$ (Training loss). Hence, to achieve an accuracy ϵ in g, we need to train to a training loss of $\epsilon \Delta t$.

7. Conclusions

One of the advantages of PHNN is that it facilitates incorporating prior knowledge and assumptions into the models. The advantage of this is evident from the experiments in Section 5; the informed PHNN model performs consistently very well. We envision that our models can be used in an iterative process where you start by the most general model and as you learn more about the system from this, priors can be imposed, eventually resulting in models that share certain geometric properties with the underlying system.

[1] This is essentially saying they are obtained during training for the loss function.

As discussed in Section 6.1, the PHNN models are highly sensitive to variations in the initialized parameters of the neural networks. By the numerical results, the general PHNN model in particular seems to be more sensitive to this than the baseline model. However, the best trained PHNN models outperform the baseline models across the board. In a practical setting, where the ground truth is missing, we could train a number of models with different initialization of the neural networks and disregard those that deviate greatly from the others.

The aim of this paper has been to introduce a new method, demonstrate some of its advantages and share the code for the interested reader to study and develop further. It is the intent of the authors to also continue the work on these models. For one, we will be doing further analysis to address the issues raised in the previous section and improve the training of the models under various conditions. Secondly, we would like to extend the code to also work on higher-dimensional PDEs, and consider more advanced problems. One of the most promising uses of the methodology may be on image denoising and inpainting, motivated by the results of sections 5.4 and 5.5. Lastly, the pseudo-Hamiltonian formulation could be used with other machine learning models than neural networks. Building on [66], we will develop methods for identifying analytic terms for one or several or all of the parts of the pseudo–Hamiltonian model (17), and compare the performance to existing system identification methods like [28–30]. We are especially intrigued by the possibility to identify the integrals of (16), while the external forces might be best modelled by a neural network.

CRediT authorship contribution statement

Sølve Eidnes: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing. **Kjetil Olsen Lye:** Methodology, Software, Writing – original draft, Writing – review & editing.

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.

Data availability

Complete code to reproduce the results of the paper are available through Zenodo and linked to in the paper.

Acknowledgement

This work was supported by the research project PRAI (Prediction of Riser-response by Artificial Intelligence) financed by the Research Council of Norway with Equinor, BP, Subsea7, Kongsberg Maritime and Aker Solutions, project no. 308832. The authors are grateful to Brynjulf Owren for illuminating discussions, and to the anonymous reviewers for their insightful comments and suggestions. Furthermore, the authors thank Katarzyna Michałowska and Signe Riemer-Sørensen for helpful comments on the manuscript, Eivind Bøhn for help with coding issues, and Benjamin Tapley for both.

References

[1] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (2021) 422–440.
[2] J. Willard, X. Jia, S. Xu, M. Steinbach, V. Kumar, Integrating scientific knowledge with machine learning for engineering and environmental systems, ACM Comput. Surv. 55 (2022), https://doi.org/10.1145/3514228.
[3] W. E, J. Han, A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Commun. Math. Stat. 5 (2017) 349–380, https://doi.org/10.1007/s40304-017-0117-6.
[4] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (2018) 1–12, https://doi.org/10.1007/s40304-018-0127-z.
[5] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364, https://doi.org/10.1016/j.jcp.2018.08.029.
[6] M. Raissi, P. Perdikaris, G.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, https://doi.org/10.1016/j.jcp.2018.10.045.
[7] Y. Bar-Sinai, S. Hoyer, J. Hickey, M.P. Brenner, Learning data-driven discretizations for partial differential equations, Proc. Natl. Acad. Sci. USA 116 (2019) 15344–15349, https://doi.org/10.1073/pnas.1814058116.
[8] S. Greydanus, M. Dzamba, J. Yosinski, Hamiltonian neural networks, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019.
[9] Y.D. Zhong, B. Dey, A. Chakraborty, Symplectic ODE-net: learning Hamiltonian dynamics with control, in: International Conference on Learning Representations, 2020.
[10] Y.D. Zhong, B. Dey, A. Chakraborty, Dissipative SymODEN: encoding Hamiltonian dynamics with dissipation and control into deep learning, in: ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020.
[11] S. Greydanus, A. Sosanya, Dissipative Hamiltonian neural networks: learning dissipative and conservative dynamics separately, arXiv preprint, arXiv:2201.10085, 2022.
[12] M. Finzi, K.A. Wang, A.G. Wilson, Simplifying Hamiltonian and Lagrangian neural networks via explicit constraints, Adv. Neural Inf. Process. Syst. 33 (2020) 13880–13889.
[13] E. Celledoni, A. Leone, D. Murari, B. Owren, Learning Hamiltonians of constrained mechanical systems, J. Comput. Appl. Math. 417 (2023) 114608, https://doi.org/10.1016/j.cam.2022.114608.

[14] S.A. Desai, M. Mattheakis, D. Sondak, P. Protopapas, S.J. Roberts, Port-Hamiltonian neural networks for learning explicit time-dependent dynamical systems, Phys. Rev. E 104 (2021) 034312, https://doi.org/10.1103/PhysRevE.104.034312.
[15] T. Duong, N. Atanasov, Hamiltonian-based neural ODE networks on the SE(3) manifold for dynamics learning and control, in: Robotics: Science and Systems (RSS), 2021.
[16] T. Duong, N. Atanasov, Adaptive control of $SE(3)$ Hamiltonian dynamics with learned disturbance features, IEEE Control Syst. Lett. 6 (2022) 2773–2778, https://doi.org/10.1109/lcsys.2022.3177156.
[17] K. Lee, N. Trask, P. Stinis, Machine learning structure preserving brackets for forecasting irreversible processes, Adv. Neural Inf. Process. Syst. 34 (2021) 5696–5707.
[18] Q. Hernández, A. Badías, F. Chinesta, E. Cueto, Port-metriplectic neural networks: thermodynamics-informed machine learning of complex physical systems, Comput. Mech. (2023) 1–9.
[19] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, S. Ho, Lagrangian neural networks, in: ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020.
[20] Z. Chen, J. Zhang, M. Arjovsky, L. Bottou, Symplectic recurrent neural networks, in: International Conference on Learning Representations, 2019.
[21] P. Jin, Z. Zhang, A. Zhu, Y. Tang, G.E. Karniadakis, SympNets: intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems, Neural Netw. 132 (2020) 166–179.
[22] M. David, F. Méhats, Symplectic learning for Hamiltonian neural networks, arXiv preprint, arXiv:2106.11753, 2021.
[23] Y. Chen, T. Matsubara, T. Yaguchi, Neural symplectic form: learning Hamiltonian equations on general coordinate systems, Adv. Neural Inf. Process. Syst. 34 (2021).
[24] S. Eidnes, A.J. Stasik, C. Sterud, E. Bøhn, S. Riemer-Sørensen, Pseudo-Hamiltonian neural networks with state-dependent external forces, Physica D 446 (2023) 133673, https://doi.org/10.1016/j.physd.2023.133673.
[25] T. Matsubara, A. Ishikawa, T. Yaguchi, Deep energy-based modeling of discrete-time physics, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 13100–13111.
[26] P. Jin, Z. Zhang, I.G. Kevrekidis, G.E. Karniadakis, Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks, IEEE Trans. Neural Netw. Learn. Syst. (2022).
[27] Z. Long, Y. Lu, X. Ma, B. Dong, PDE-Net: learning PDEs from data, in: International Conference on Machine Learning, PMLR, 2018, pp. 3208–3216.
[28] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. 3 (2017) e1602614.
[29] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proc. R. Soc. A, Math. Phys. Eng. Sci. 473 (2017) 20160446, https://doi.org/10.1098/rspa.2016.0446.
[30] K. Kaheman, J.N. Kutz, S.L. Brunton, SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics, Proc. R. Soc. A, Math. Phys. Eng. Sci. 476 (2020) 20200279, https://doi.org/10.1098/rspa.2020.0279.
[31] A. Anandkumar, K. Azizzadenesheli, K. Bhattacharya, N. Kovachki, Z. Li, B. Liu, A. Stuart, Neural operator: graph kernel network for partial differential equations, in: ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020.
[32] Z. Li, N.B. Kovachki, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, et al., Fourier neural operator for parametric partial differential equations, in: International Conference on Learning Representations, 2021.
[33] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229.
[34] S.L. Brunton, J.N. Kutz, Machine learning for partial differential equations, arXiv preprint, arXiv:2303.17078, 2023.
[35] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[36] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314, https://doi.org/10.1007/BF02551274.
[37] Z. Long, Y. Lu, B. Dong, PDE-Net 2.0: learning PDEs from data with a numeric-symbolic hybrid deep network, J. Comput. Phys. 399 (2019) 108925, https://doi.org/10.1016/j.jcp.2019.108925.
[38] H. Noren, Learning Hamiltonian systems with mono-implicit Runge–Kutta methods, arXiv preprint, arXiv:2303.03769, 2023.
[39] H. Noren, S. Eidnes, E. Celledoni, Learning dynamical systems from noisy data with inverse-explicit integrators, arXiv preprint, arXiv:2306.03548, 2023.
[40] J.-F. Cai, B. Dong, S. Osher, Z. Shen, Image restoration: total variation, wavelet frames, and beyond, J. Am. Math. Soc. 25 (2012) 1033–1089, https://doi.org/10.1090/S0894-0347-2012-00740-1.
[41] B. Dong, Q. Jiang, Z. Shen, Image restoration: wavelet frame shrinkage, nonlinear evolution PDEs, and beyond, Multiscale Model. Simul. 15 (2017) 606–660, https://doi.org/10.1137/15M1037457.
[42] L. Ruthotto, E. Haber, Deep neural networks motivated by partial differential equations, J. Math. Imaging Vis. 62 (2020) 352–364, https://doi.org/10.1007/s10851-019-00903-1.
[43] E. Celledoni, J. Jackaman, D. Murari, B. Owren, Predictions based on pixel data: insights from PDEs and finite differences, arXiv preprint, arXiv:2305.00723, 2023.
[44] P. Guha, Metriplectic structure, Leibniz dynamics and dissipative systems, J. Math. Anal. Appl. 326 (2007) 121–136, https://doi.org/10.1016/j.jmaa.2006.02.023.
[45] A.M. Bloch, P.J. Morrison, T.S. Ratiu, Gradient flows in the normal and Kähler metrics and triple bracket generated metriplectic systems, in: Recent Trends in Dynamical Systems, in: Springer Proc. Math. Stat., vol. 35, Springer, Basel, 2013, pp. 371–415.
[46] M. Grmela, H.C. Öttinger, Dynamics and thermodynamics of complex fluids. I. Development of a general formalism, Phys. Rev. E (3) 56 (1997) 6620–6632, https://doi.org/10.1103/PhysRevE.56.6620.
[47] H.C. Öttinger, M. Grmela, Dynamics and thermodynamics of complex fluids. II. Illustrations of a general formalism, Phys. Rev. E (3) 56 (1997) 6633–6655, https://doi.org/10.1103/PhysRevE.56.6633.
[48] Z. Zhang, Y. Shin, G.E. Karniadakis, GFINNs: GENERIC formalism informed neural networks for deterministic and stochastic dynamical systems, Philos. Trans. R. Soc. A 380 (2022) 20210207, https://doi.org/10.1098/rsta.2021.0207.
[49] B. Leimkuhler, S. Reich, Simulating Hamiltonian Dynamics, Cambridge Monographs on Applied and Computational Mathematics, vol. 14, Cambridge University Press, Cambridge, 2004.
[50] P.J. Olver, Applications of Lie Groups to Differential Equations, second ed., Graduate Texts in Mathematics, vol. 107, Springer-Verlag, New York, 1993.
[51] M. Wang, Exact solutions for a compound KdV-Burgers equation, Phys. Lett. A 213 (1996) 279–287, https://doi.org/10.1016/0375-9601(96)00103-X.
[52] S. Eidnes, B. Owren, T. Ringholm, Adaptive energy preserving methods for partial differential equations, Adv. Comput. Math. 44 (2018) 815–839, https://doi.org/10.1007/s10444-017-9562-8.
[53] D. Zhang, L. Guo, G.E. Karniadakis, Learning in modal space: solving time-dependent stochastic PDEs using physics-informed neural networks, SIAM J. Sci. Comput. 42 (2020) A639–A665, https://doi.org/10.1137/19M1260141.
[54] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, S.G. Johnson, Physics-informed neural networks with hard constraints for inverse design, SIAM J. Sci. Comput. 43 (2021) B1105–B1132, https://doi.org/10.1137/21M1397908.
[55] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, G.E. Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data, Comput. Methods Appl. Mech. Eng. 393 (2022) 114778, https://doi.org/10.1016/j.cma.2022.114778.
[56] D.H. Peregrine, Calculations of the development of an undular bore, J. Fluid Mech. 25 (1966) 321–330.

[57] T.B. Benjamin, J.L. Bona, J.J. Mahony, Model equations for long waves in nonlinear dispersive systems, Philos. Trans. R. Soc. Lond., Ser. A 272 (1972) 47–78, https://doi.org/10.1098/rsta.1972.0032.
[58] P. Perona, J. Malik, Scale-space and edge detection using anisotropic diffusion, IEEE Trans. Pattern Anal. Mach. Intell. 12 (1990) 629–639.
[59] J.W. Cahn, J.E. Hilliard, Free energy of a nonuniform system. I. Interfacial free energy, J. Chem. Phys. 28 (1958) 258–267.
[60] M. Burger, L. He, C.-B. Schönlieb, Cahn-Hilliard inpainting and a generalization for grayvalue images, SIAM J. Imaging Sci. 2 (2009) 1129–1167, https://doi.org/10.1137/080728548.
[61] C.-B. Schönlieb, Partial Differential Equation Methods for Image Inpainting, Cambridge Monographs on Applied and Computational Mathematics, vol. 29, Cambridge University Press, New York, 2015.
[62] H. Zhao, B.D. Storey, R.D. Braatz, M.Z. Bazant, Learning the physics of pattern formation from images, Phys. Rev. Lett. 124 (2020) 060201.
[63] C.S. Gardner, Korteweg-de Vries equation and generalizations. IV. The Korteweg-de Vries equation as a Hamiltonian system, J. Math. Phys. 12 (1971) 1548–1551, https://doi.org/10.1063/1.1665772.
[64] R.M. Miura, Korteweg-de Vries equation and generalizations. I. A remarkable explicit nonlinear transformation, J. Math. Phys. 9 (1968) 1202–1204, https://doi.org/10.1063/1.1664700.
[65] J.P. Wang, A list of $1+1$ dimensional integrable equations and their properties, J. Nonlinear Math. Phys. 9 (2002) 213–233, https://doi.org/10.2991/jnmp.2002.9.s1.18, recent advances in integrable systems (Kowloon, 2000).
[66] S. Holmsen, S. Eidnes, S. Riemer-Sørensen, Pseudo-Hamiltonian system identification, arXiv preprint, arXiv:2305.06920, 2023.