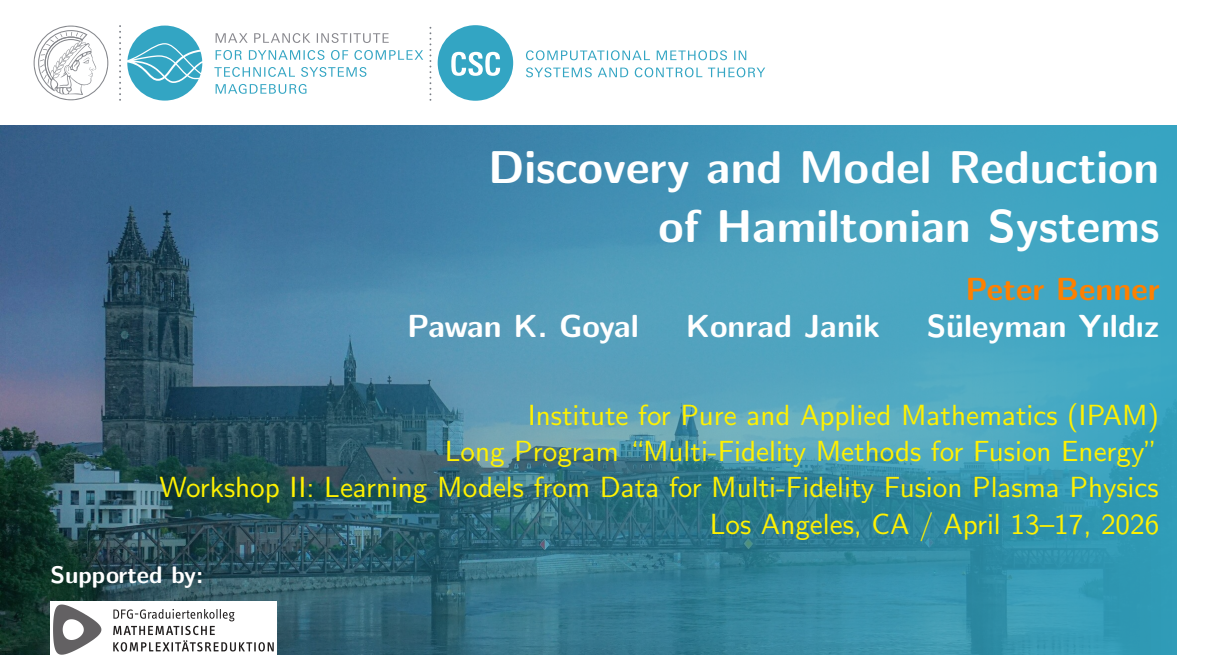

MAX PLANCK INSTITUTE
FOR DYNAMICS OF COMPLEX
TECHNICAL SYSTEMS
MAGDEBURG
CSC
COMPUTATIONAL METHODS IN
SYSTEMS AND CONTROL THEORY
Discovery and Model Reduction
of Hamiltonian Systems
Peter Benner
Pawan K. Goyal
Konrad Janik
Süleyman Yıldız
Institute for Pure and Applied Mathematics (IPAM)
Long Program "Multi-Fidelity Methods for Fusion Energy"
Workshop II: Learning Models from Data for Multi-Fidelity Fusion Plasma Physics
Los Angeles, CA / April 13–17, 2026
Supported by:
DFG-Graduiertenkolleg
MATHEMATISCHE
KOMPLEXITÄTSREDUKTION

# Outline

# Hamiltonian Systems

# Hamiltonian Systems

A Hamiltonian system is given by the following system of ODEs:

$$\begin{pmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{p}} \end{pmatrix} = \begin{pmatrix} \nabla_{\boldsymbol{p}} H(\boldsymbol{q}, \boldsymbol{p}) \\ -\nabla_{\boldsymbol{q}} H(\boldsymbol{q}, \boldsymbol{p}) \end{pmatrix} = \boldsymbol{J}_{2d} \nabla H(\boldsymbol{q}, \boldsymbol{p}), \qquad \boldsymbol{J}_{2d} := \begin{pmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{0} \end{pmatrix} \in \mathbb{R}^{2d \times 2d},$$

where $H : \mathbb{R}^{2d} \to \mathbb{R}$ is the **Hamiltonian**, $\boldsymbol{q}(0) = \boldsymbol{q}_0,\ \boldsymbol{p}(0) = \boldsymbol{p}_0$ and $\boldsymbol{q}_0, \boldsymbol{p}_0 \in \mathbb{R}^d$.

# Hamiltonian Systems

A Hamiltonian system is given by the following system of ODEs:

$$\begin{pmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{p}} \end{pmatrix} = \begin{pmatrix} \nabla_{\boldsymbol{p}} H(\boldsymbol{q}, \boldsymbol{p}) \\ -\nabla_{\boldsymbol{q}} H(\boldsymbol{q}, \boldsymbol{p}) \end{pmatrix} = \boldsymbol{J}_{2d} \nabla H(\boldsymbol{q}, \boldsymbol{p}), \qquad \boldsymbol{J}_{2d} := \begin{pmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{0} \end{pmatrix} \in \mathbb{R}^{2d \times 2d},$$

where $H : \mathbb{R}^{2d} \to \mathbb{R}$ is the **Hamiltonian**, $\boldsymbol{q}(0) = \boldsymbol{q}_0,\ \boldsymbol{p}(0) = \boldsymbol{p}_0$ and $\boldsymbol{q}_0, \boldsymbol{p}_0 \in \mathbb{R}^d$.

Its flow $\boldsymbol{\phi} : \mathbb{R} \times \mathbb{R}^{2d} \to \mathbb{R}^{2d},\ \boldsymbol{\phi}(t, \boldsymbol{x}_0) \mapsto \boldsymbol{x}(t) \equiv (\boldsymbol{q}(t)^T, \boldsymbol{p}(t)^T)^T$, has two important properties:

1. Symplecticity, i.e.,

$$\left( \frac{\partial \boldsymbol{\phi}(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \right)^T \boldsymbol{J}_{2d} \left( \frac{\partial \boldsymbol{\phi}(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \right) = \boldsymbol{J}_{2d}.$$

# Hamiltonian Systems

A Hamiltonian system is given by the following system of ODEs:

$$\begin{pmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{p}} \end{pmatrix} = \begin{pmatrix} \nabla_{\boldsymbol{p}} H(\boldsymbol{q}, \boldsymbol{p}) \\ -\nabla_{\boldsymbol{q}} H(\boldsymbol{q}, \boldsymbol{p}) \end{pmatrix} = \boldsymbol{J}_{2d} \nabla H(\boldsymbol{q}, \boldsymbol{p}), \qquad \boldsymbol{J}_{2d} := \begin{pmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{0} \end{pmatrix} \in \mathbb{R}^{2d \times 2d},$$

where $H : \mathbb{R}^{2d} \to \mathbb{R}$ is the **Hamiltonian**, $\boldsymbol{q}(0) = \boldsymbol{q}_0,\ \boldsymbol{p}(0) = \boldsymbol{p}_0$ and $\boldsymbol{q}_0, \boldsymbol{p}_0 \in \mathbb{R}^d$.

Its flow $\boldsymbol{\phi} : \mathbb{R} \times \mathbb{R}^{2d} \to \mathbb{R}^{2d},\ \boldsymbol{\phi}(t, \boldsymbol{x}_0) \mapsto \boldsymbol{x}(t) \equiv (\boldsymbol{q}(t)^T, \boldsymbol{p}(t)^T)^T$, has two important properties:

1. Symplecticity, i.e.,

$$\left( \frac{\partial \boldsymbol{\phi}(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \right)^T \boldsymbol{J}_{2d} \left( \frac{\partial \boldsymbol{\phi}(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \right) = \boldsymbol{J}_{2d}.$$

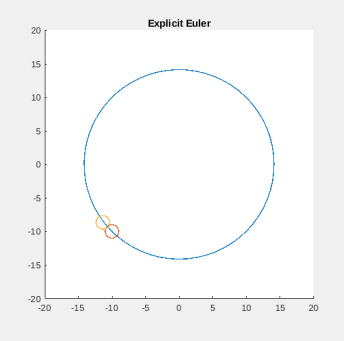
2. Conservation of the Hamiltonian (kinetic plus potential energy), i.e., $H(\boldsymbol{\phi}(t, \boldsymbol{x})) = H(\boldsymbol{x})$.

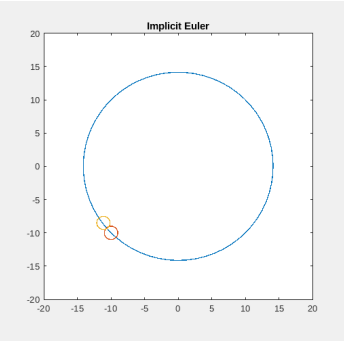
Consider the simple example of the harmonic oscillator

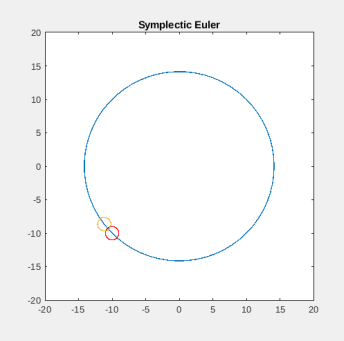
$$\dot{q} = p, \qquad q(t) = \cos(t)q_0 + \sin(t)p_0,$$
$$\dot{p} = -q, \qquad p(t) = -\sin(t)q_0 + \cos(t)p_0.$$

Consider the simple example of the harmonic oscillator

$$\dot{q} = p, \qquad q(t) = \cos(t)q_0 + \sin(t)p_0,$$
$$\dot{p} = -q, \qquad p(t) = -\sin(t)q_0 + \cos(t)p_0.$$




,

# Maxwell's Equations

Maxwell's equations

$$\nabla \cdot \boldsymbol{E} = \rho/\epsilon_0, \qquad \nabla \times \boldsymbol{E} = -\partial_t \boldsymbol{B},$$
$$\nabla \cdot \boldsymbol{B} = 0, \qquad \nabla \times \boldsymbol{B} = \mu_0 \left(\boldsymbol{J} + \epsilon_0 \partial_t \boldsymbol{E}\right).$$

# Maxwell's Equations

Maxwell's equations in vacuum, i.e., $\rho = 0,\ \boldsymbol{J} = 0$

$$\nabla \cdot \boldsymbol{E} = 0, \qquad \nabla \times \boldsymbol{E} = -\partial_t \boldsymbol{B},$$
$$\nabla \cdot \boldsymbol{B} = 0, \qquad \nabla \times \boldsymbol{B} = \mu_0 \epsilon_0 \partial_t \boldsymbol{E}.$$

# Maxwell's Equations

Maxwell's equations in vacuum, i.e., $\rho = 0,\ \boldsymbol{J} = 0$

$$\nabla \cdot \boldsymbol{E} = 0, \qquad \nabla \times \boldsymbol{E} = -\partial_t \boldsymbol{B},$$
$$\nabla \cdot \boldsymbol{B} = 0, \qquad \nabla \times \boldsymbol{B} = \mu_0 \epsilon_0 \partial_t \boldsymbol{E}.$$

The ansatz: $\boldsymbol{B} = \nabla \times \boldsymbol{A},\ \boldsymbol{E} = -\partial_t \boldsymbol{A}$ with Coulomb gauge $\nabla \cdot \boldsymbol{A} = 0$ yields

$$-\mu_0 \epsilon_0 \partial_t^2 \boldsymbol{A} = \nabla \times (\nabla \times \boldsymbol{A}).$$

# Maxwell's Equations

Maxwell's equations in vacuum, i.e., $\rho = 0,\ \boldsymbol{J} = 0$

$$\begin{aligned} \nabla \cdot \boldsymbol{E} &= 0, & \nabla \times \boldsymbol{E} &= -\partial_t \boldsymbol{B}, \\ \nabla \cdot \boldsymbol{B} &= 0, & \nabla \times \boldsymbol{B} &= \mu_0 \epsilon_0 \partial_t \boldsymbol{E}. \end{aligned}$$

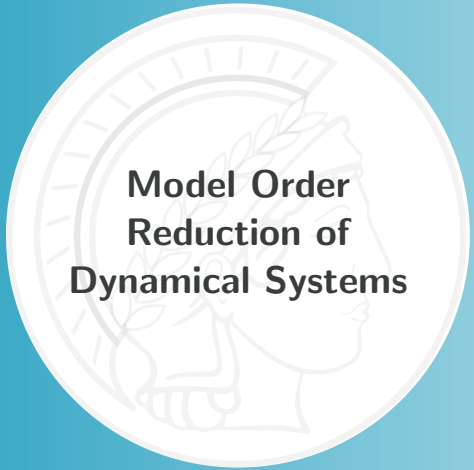
The ansatz: $\boldsymbol{B} = \nabla \times \boldsymbol{A},\ \boldsymbol{E} = -\partial_t \boldsymbol{A}$ with Coulomb gauge $\nabla \cdot \boldsymbol{A} = 0$ yields

$$-\mu_0 \epsilon_0 \partial_t^2 \boldsymbol{A} = \nabla \times (\nabla \times \boldsymbol{A}).$$

Choosing $\boldsymbol{q} = \boldsymbol{A}$ and $\boldsymbol{p} = \partial_t \boldsymbol{A}$, we obtain a Hamiltonian system with

$$H(\boldsymbol{q}, \boldsymbol{p}) = \frac{1}{2} \int_\Omega \left( c(\nabla \times \boldsymbol{q})^2 + \boldsymbol{p}^2 \right) d\boldsymbol{x}.$$

Similar geometry/structure also appears in plasma physics like Vlasov-Maxwell/Poisson systems, according to [Sonnendrücker, Kraus, Brantner, Kormann,...].



# Model Order Reduction of Dynamical Systems

## Original System

$$\Sigma : \begin{cases} \dot{x}(t) &= f(t, x(t), u(t)), \\ y(t) &= g(t, x(t), u(t)), \end{cases}$$

- states $x(t) \in \mathbb{R}^n$,
- inputs $u(t) \in \mathbb{R}^m$,
- outputs $y(t) \in \mathbb{R}^p$.

## Original System

$$\Sigma : \begin{cases} \dot{x}(t) &= f(t, x(t), u(t)), \\ y(t) &= g(t, x(t), u(t)), \end{cases}$$

- states $x(t) \in \mathbb{R}^n$,
- inputs $u(t) \in \mathbb{R}^m$,
- outputs $y(t) \in \mathbb{R}^p$.


## Reduced-Order Model (ROM)

$$\widehat{\Sigma} : \begin{cases} \dot{\hat{x}}(t) &= \widehat{f}(t, \hat{x}(t), u(t)), \\ \hat{y}(t) &= \widehat{g}(t, \hat{x}(t), u(t)), \end{cases}$$

- states $\hat{x}(t) \in \mathbb{R}^r$, $r \ll n$,
- inputs $u(t) \in \mathbb{R}^m$,
- outputs $\hat{y}(t) \in \mathbb{R}^p$.

# Model Order Reduction of Dynamical Systems

## Original System

$$\Sigma : \begin{cases} \dot{x}(t) &= f(t, x(t), u(t)), \\ y(t) &= g(t, x(t), u(t)), \end{cases}$$

- states $x(t) \in \mathbb{R}^n$,
- inputs $u(t) \in \mathbb{R}^m$,
- outputs $y(t) \in \mathbb{R}^p$.


## Reduced-Order Model (ROM)

$$\widehat{\Sigma} : \begin{cases} \dot{\hat{x}}(t) &= \widehat{f}(t, \hat{x}(t), u(t)), \\ \hat{y}(t) &= \widehat{g}(t, \hat{x}(t), u(t)), \end{cases}$$

- states $\hat{x}(t) \in \mathbb{R}^r$, $r \ll n$,
- inputs $u(t) \in \mathbb{R}^m$,
- outputs $\hat{y}(t) \in \mathbb{R}^p$.


## Goals:

$\|y - \hat{y}\| < \text{tolerance} \cdot \|u\|$ for all admissible input signals.

## Original System

$$\Sigma : \begin{cases} \dot{x}(t) &= f(t, x(t), u(t)), \\ y(t) &= g(t, x(t), u(t)), \end{cases}$$

- states $x(t) \in \mathbb{R}^n$,
- inputs $u(t) \in \mathbb{R}^m$,
- outputs $y(t) \in \mathbb{R}^p$.


## Reduced-Order Model (ROM)

$$\widehat{\Sigma} : \begin{cases} \dot{\hat{x}}(t) &= \widehat{f}(t, \hat{x}(t), u(t)), \\ \hat{y}(t) &= \widehat{g}(t, \hat{x}(t), u(t)), \end{cases}$$

- states $\hat{x}(t) \in \mathbb{R}^r$, $r \ll n$,
- inputs $u(t) \in \mathbb{R}^m$,
- outputs $\hat{y}(t) \in \mathbb{R}^p$.


## Goals:

$\|y - \hat{y}\| < \text{tolerance} \cdot \|u\|$ for all admissible input signals.

Secondary goal: reconstruct ("decode") approximation of $x$ from $\hat{x}$.

## Original System

$$\Sigma : \begin{cases} \dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t). \end{cases}$$

- states $x(t) \in \mathbb{R}^n$,
- inputs $u(t) \in \mathbb{R}^m$,
- outputs $y(t) \in \mathbb{R}^p$.


## Reduced-Order Model (ROM)

$$\widehat{\Sigma} : \begin{cases} \dot{\hat{x}}(t) &= \widehat{A}\hat{x}(t) + \widehat{B}u(t), \\ \hat{y}(t) &= \widehat{C}\hat{x}(t) + \widehat{D}u(t). \end{cases}$$

- states $\hat{x}(t) \in \mathbb{R}^r$, $r \ll n$
- inputs $u(t) \in \mathbb{R}^m$,
- outputs $\hat{y}(t) \in \mathbb{R}^p$.


## Goals:

$\|y - \hat{y}\| < \text{tolerance} \cdot \|u\|$ for all admissible input signals.

Secondary goal: reconstruct ("decode") approximation of $x$ from $\hat{x}$.

# Model Order Reduction of Linear Systems

## Model Reduction Schematically

$$E\,\dot{x}(t) = A\,x(t) + B\,u(t)$$

$$y(t) = C\,x(t) + D\,u(t)$$

MOR

$$\hat{E}\,\dot{\hat{x}}(t) = \hat{A}\,\hat{x}(t) + \hat{B}\,u(t)$$

$$\hat{y}(t) = \hat{C}\,\hat{x}(t) + \hat{D}\,u(t)$$

- $E, A \in \mathbb{R}^{n\times n}$
- $B \in \mathbb{R}^{n\times m}$
- $C \in \mathbb{R}^{p\times n}$
- $D \in \mathbb{R}^{p\times m}$

- $\hat{E}, \hat{A} \in \mathbb{R}^{r\times r}$
- $\hat{B} \in \mathbb{R}^{r\times m}$
- $\hat{C} \in \mathbb{R}^{p\times r}$
- $\hat{D} \in \mathbb{R}^{p\times m}$

# Why Model Order Reduction of Dynamical Systems?

**MOR (surrogate or reduced-order modeling)** becomes important when evaluation (simulation) of full-order model (FOM) becomes too time-consuming, e.g. in

# Why Model Order Reduction of Dynamical Systems?

**MOR (surrogate or reduced-order modeling)** becomes important when evaluation (simulation) of full-order model (FOM) becomes too time-consuming, e.g. in

- **multi-query scenarios:** the model needs to be evaluated many times for changing parameter instances (realizations), varying initial and boundary conditions, e.g., for Monte-Carlo simulations;

# Why Model Order Reduction of Dynamical Systems?

**MOR (surrogate or reduced-order modeling)** becomes important when evaluation (simulation) of full-order model (FOM) becomes too time-consuming, e.g. in

- **multi-query scenarios:** the model needs to be evaluated many times for changing parameter instances (realizations), varying initial and boundary conditions, e.g., for Monte-Carlo simulations;
- **real-time scenarios:** for control purposes (e.g., navigating vehicles, planes, ...) or in Digital Twins (for health monitoring, autonomous driving/flying, etc.), results are needed with very short response times (often, milliseconds).

# Why Model Order Reduction of Dynamical Systems?

**Example (for multi-query scenario in thermodynamics)**

Heat equation with convective transport:

$$\begin{aligned} \frac{\partial \mathbf{x}}{\partial t} &= \lambda \triangle \mathbf{x} + v \cdot \nabla \mathbf{x}, \qquad \mathbf{x} = \mathbf{x}(t,\xi), \quad \xi \in \Omega \subset \mathbb{R}^d,\ t \in (0,T],\ v \in \mathbb{R}^d,\ d = 2,3 \\ \mathbf{x}(0,\xi) &= \mathbf{x}_0(\xi) \text{ in } \Omega, \qquad \mathbf{x}(t,\xi) = \mathbf{u}(t,\xi) \text{ on } \partial\Omega, \end{aligned}$$

where $\mathbf{x}_0, \mathbf{u}$ are given functions of sufficient regularity, $\lambda > 0$ is the medium's thermal diffusivity.

# Why Model Order Reduction of Dynamical Systems?

**Example (for multi-query scenario in thermodynamics)**

Heat equation with convective transport:

$$\begin{aligned} \frac{\partial \mathbf{x}}{\partial t} &= \lambda \triangle \mathbf{x} + v \cdot \nabla \mathbf{x}, \qquad \mathbf{x} = \mathbf{x}(t,\xi), \quad \xi \in \Omega \subset \mathbb{R}^d,\ t \in (0,T],\ v \in \mathbb{R}^d,\ d = 2,3 \\ \mathbf{x}(0,\xi) &= \mathbf{x}_0(\xi) \text{ in } \Omega, \qquad \mathbf{x}(t,\xi) = \mathbf{u}(t,\xi) \text{ on } \partial\Omega, \end{aligned}$$

where $\mathbf{x}_0, \mathbf{u}$ are given functions of sufficient regularity, $\lambda > 0$ is the medium's thermal diffusivity.

**Spatial discretization** (finite elements/differences) leads to LTI system (for simplicity, assume $C =$ identity, i.e., full observation)

$$M\dot{x}(t) = (\lambda K + L(v))x(t) + Bu(t), \qquad x(0) = x_0, \quad M, K, L(v) \in \mathbb{R}^{n\times n},\ B \in \mathbb{R}^{n\times m}.$$

# Why Model Order Reduction of Dynamical Systems?

**Example (for multi-query scenario in thermodynamics)**

Heat equation with convective transport:

**Spatial discretization** (finite elements/differences) leads to LTI system (for simplicity, assume $C =$ identity, i.e., full observation)

$$M\dot{x}(t) \;=\; (\lambda K + L(v))x(t) + Bu(t), \qquad x(0) = x_0, \quad M, K, L(v) \in \mathbb{R}^{n\times n},\; B \in \mathbb{R}^{n\times m}.$$

Now assume one solve of the heat equation requires **1 min of computing time,** and we need to explore

- 20 different values of the thermal diffusivity $\lambda$ (e.g., different materials to be tested),
- 20 different wind directions and magnitudes $v$ for each $\lambda$, and
- 18 different control/forcing scenarios (i.e., different $\mathbf{u}(t, \xi)$).

$\rightsquigarrow$ 7,200 min of computing time, i.e., **5 full days for the whole computation!**

# Why Model Order Reduction of Dynamical Systems?

**Example (for multi-query scenario in thermodynamics)**

Heat equation with convective transport:

**Spatial discretization** (finite elements/differences) leads to LTI system (for simplicity, assume $C =$ identity, i.e., full observation)

$$M\dot{x}(t) \;=\; (\lambda K + L(v))x(t) + Bu(t), \qquad x(0) = x_0, \quad M, K, L(v) \in \mathbb{R}^{n\times n},\; B \in \mathbb{R}^{n\times m}.$$

Now assume one solve of the heat equation requires **1 min of computing time,** and we need to explore

- 20 different values of the thermal diffusivity $\lambda$ (e.g., different materials to be tested),
- 20 different wind directions and magnitudes $v$ for each $\lambda$, and
- 18 different control/forcing scenarios (i.e., different $\mathbf{u}(t, \xi)$).

$\rightsquigarrow$ 7,200 min of computing time, i.e., **5 full days for the whole computation!**

**Using a reduced-order model that computes the important information (quantity-of-interest) in about 1 sec, the task would be done in 2 hours!**

# Reduced-order Models from Data

**Assumption:** trajectory $x(t; u)$ is contained in low-dimensional subspace $\mathcal{V} \subset \mathbb{R}^n$.

# Reduced-order Models from Data

**Assumption:** trajectory $x(t;u)$ is contained in low-dimensional subspace $\mathcal{V} \subset \mathbb{R}^n$.

Thus, use Galerkin or Petrov-Galerkin-type projection of state-space onto $\mathcal{V}$ (trial space) along complementary subspace $\mathcal{W}$ (test space), where

$$\operatorname{range}(V) = \mathcal{V}, \quad \operatorname{range}(W) = \mathcal{W}, \quad W^T V = I_r.$$

The **reduced-order (or surrogate) model** then is

$$\hat{x} = W^T x, \quad \hat{A} := W^T A V, \quad \hat{B} := W^T B, \quad \hat{C} := CV, \quad (\hat{D} := D).$$

# Reduced-order Models from Data

**Assumption:** trajectory $x(t;u)$ is contained in low-dimensional subspace $\mathcal{V} \subset \mathbb{R}^n$.

Thus, use Galerkin or Petrov-Galerkin-type projection of state-space onto $\mathcal{V}$ (trial space) along complementary subspace $\mathcal{W}$ (test space), where

$$\operatorname{range}(V) = \mathcal{V}, \quad \operatorname{range}(W) = \mathcal{W}, \quad W^T V = I_r.$$

The **reduced-order (or surrogate) model** then is

$$\hat{x} = W^T x, \quad \hat{A} := W^T A V, \quad \hat{B} := W^T B, \quad \hat{C} := CV, \quad (\hat{D} := D).$$

But: we need the matrices $A, B, C, D$ to compute the reduced-order model!

# Reduced-order Models from Data

**Assumption:** trajectory $x(t;u)$ is contained in low-dimensional subspace $\mathcal{V} \subset \mathbb{R}^n$.

Thus, use Galerkin or Petrov-Galerkin-type projection of state-space onto $\mathcal{V}$ (trial space) along complementary subspace $\mathcal{W}$ (test space), where

$$\operatorname{range}(V) = \mathcal{V}, \quad \operatorname{range}(W) = \mathcal{W}, \quad W^T V = I_r.$$

The **reduced-order (or surrogate) model** then is

$$\hat{x} = W^T x, \quad \hat{A} := W^T A V, \quad \hat{B} := W^T B, \quad \hat{C} := CV, \quad (\hat{D} := D).$$

But: we need the matrices $A, B, C, D$ to compute the reduced-order model!

Using proprietary simulation software, we would need to **intrude** the software to get the matrices

$\rightsquigarrow$ **intrusive MOR** = *learning (compact, surrogate) models from (full, detailed) models.*

This is often impossible!

# Reduced-order Models from Data

**Assumption:** trajectory $x(t;u)$ is contained in low-dimensional subspace $\mathcal{V} \subset \mathbb{R}^n$.

Thus, use Galerkin or Petrov-Galerkin-type projection of state-space onto $\mathcal{V}$ (trial space) along complementary subspace $\mathcal{W}$ (test space), where

$$\operatorname{range}(V) = \mathcal{V}, \quad \operatorname{range}(W) = \mathcal{W}, \quad W^T V = I_r.$$

The **reduced-order (or surrogate) model** then is

$$\hat{x} = W^T x, \quad \hat{A} := W^T A V, \quad \hat{B} := W^T B, \quad \hat{C} := CV, \quad (\hat{D} := D).$$

But: we need the matrices $A, B, C, D$ to compute the reduced-order model!

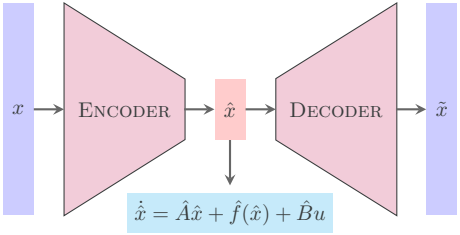
Using proprietary simulation software, we would need to **intrude** the software to get the matrices

$\rightsquigarrow$ **intrusive MOR** = *learning (compact, surrogate) models from (full, detailed) models.*

This is often impossible!

$\rightsquigarrow$ **non-intrusive MOR** = *LEARNING (compact, surrogate) MODELS FROM DATA!*
*Also called: model inference / discovery.*

In the language of artificial intelligence (AI) or machine learning (ML), model reduction can be seen as a special instance of an autoencoder:

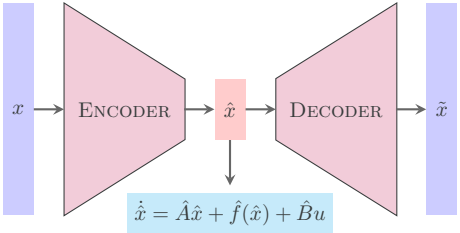



So far: both encoder and decoder use linear map to the **latent space** (our "reduced-order space"):

$$\textbf{encoder:} \quad \hat{x}(t) := W^T x(t), \qquad \textbf{decoder:} \quad \tilde{x}(t) = V\hat{x}(t),$$

where $W$ is the matrix obtained by the model reduction procedure, in non-intrusive MOR usually containing the dominant POD/PCA modes.

In the language of artificial intelligence (AI) or machine learning (ML), model reduction can be seen as a special instance of an autoencoder:




So far: both encoder and decoder use linear map to the **latent space** (our "reduced-order space"):

$$\textbf{encoder:} \quad \hat{x}(t) := W^T x(t), \qquad \textbf{decoder:} \quad \tilde{x}(t) = V\hat{x}(t),$$

where $W$ is the matrix obtained by the model reduction procedure, in non-intrusive MOR usually containing the dominant POD/PCA modes.

For nonlinear systems, nonlinear encoders and/or decoders might give better results!

# Goal

We want to do structure-preserving model order reduction, i.e., find a mapping $\boldsymbol{\psi} : \mathbb{R}^{2n} \to \mathbb{R}^{2k}$, $n \gg k$, such that the reduced-order dynamics are still governed by a Hamiltonian system

$$\dot{\tilde{\boldsymbol{x}}} = \boldsymbol{J}_{2k} \nabla \tilde{H}(\tilde{\boldsymbol{x}}), \qquad \tilde{\boldsymbol{x}} = \boldsymbol{\psi}(\boldsymbol{x}).$$

# Goal

We want to do structure-preserving model order reduction, i.e., find a mapping $\boldsymbol{\psi} : \mathbb{R}^{2n} \to \mathbb{R}^{2k}$, $n \gg k$, such that the reduced-order dynamics are still governed by a Hamiltonian system

$$\dot{\tilde{\boldsymbol{x}}} = \boldsymbol{J}_{2k} \nabla \tilde{H}(\tilde{\boldsymbol{x}}), \qquad \tilde{\boldsymbol{x}} = \boldsymbol{\psi}(\boldsymbol{x}).$$

**Theorem ([Hairer/Wanner/Lubich 2006])**

*Let $\boldsymbol{\psi} \in C^1(\mathbb{R}^{2n}; \mathbb{R}^{2k})$ be a symplectic projection. Then the Hamiltonian system $\dot{\boldsymbol{x}} = \boldsymbol{J}_{2n} \nabla H(\boldsymbol{x})$ becomes a Hamiltonian in the reduced coordinates $\tilde{\boldsymbol{x}} = \boldsymbol{\psi}(\boldsymbol{x})$*

$$\dot{\tilde{\boldsymbol{x}}} = \boldsymbol{J}_{2k} \nabla \tilde{H}(\tilde{\boldsymbol{x}}), \qquad \text{with} \quad \tilde{H}(\tilde{\boldsymbol{x}}) = H(\boldsymbol{x}).$$

# Goal

We want to do structure-preserving model order reduction, i.e., find a mapping $\boldsymbol{\psi} : \mathbb{R}^{2n} \to \mathbb{R}^{2k}$, $n \gg k$, such that the reduced-order dynamics are still governed by a Hamiltonian system

$$\dot{\tilde{\boldsymbol{x}}} = \boldsymbol{J}_{2k} \nabla \tilde{H}(\tilde{\boldsymbol{x}}), \qquad \tilde{\boldsymbol{x}} = \boldsymbol{\psi}(\boldsymbol{x}).$$

**Theorem ([Hairer/Wanner/Lubich 2006])**

*Let $\boldsymbol{\psi} \in C^1(\mathbb{R}^{2n}; \mathbb{R}^{2k})$ be a symplectic projection. Then the Hamiltonian system $\dot{\boldsymbol{x}} = \boldsymbol{J}_{2n} \nabla H(\boldsymbol{x})$ becomes a Hamiltonian in the reduced coordinates $\tilde{\boldsymbol{x}} = \boldsymbol{\psi}(\boldsymbol{x})$*

$$\dot{\tilde{\boldsymbol{x}}} = \boldsymbol{J}_{2k} \nabla \tilde{H}(\tilde{\boldsymbol{x}}), \qquad \text{with} \quad \tilde{H}(\tilde{\boldsymbol{x}}) = H(\boldsymbol{x}).$$

A map $\boldsymbol{\psi} \in C^1(\mathbb{R}^{2n}; \mathbb{R}^{2k})$ is called symplectic projection if

$$(\mathrm{d}\, \boldsymbol{\psi}_{\boldsymbol{x}}) \boldsymbol{J}_{2n} (\mathrm{d}\, \boldsymbol{\psi}_{\boldsymbol{x}})^T = \boldsymbol{J}_{2k}, \qquad \forall\, \boldsymbol{x} \in \mathbb{R}^{2n}.$$

# Proper Symplectic Decomposition (PSD) [Peng/Mohseni 2016]

**Algorithm** PSD Algorithm

**Require:** An empirical data ensemble $\{\boldsymbol{q}(t_i), \boldsymbol{p}(t_i)\}_{i=1}^{N}$.

**Ensure:** A symplectic matrix $\boldsymbol{A} \in \mathbb{M}(2n, 2k)$ in block-diagonal form.

1: Construct an extended snapshot matrix $\boldsymbol{M}_1 := [\boldsymbol{q}(t_1), \ldots, \boldsymbol{q}(t_N), \boldsymbol{p}(t_1), \ldots, \boldsymbol{p}(t_N)]$.
2: Compute the SVD of $\boldsymbol{M}_1$ to obtain a POD basis matrix $\boldsymbol{\Phi}$.
3: Construct the symplectic matrix $\boldsymbol{A} = \mathrm{diag}(\boldsymbol{\Phi}, \boldsymbol{\Phi})$.

**Algorithm** PSD Algorithm

**Require:** An empirical data ensemble $\{\boldsymbol{q}(t_i), \boldsymbol{p}(t_i)\}_{i=1}^{N}$.
**Ensure:** A symplectic matrix $\boldsymbol{A} \in \mathbb{M}(2n, 2k)$ in block-diagonal form.
1: Construct an extended snapshot matrix $\boldsymbol{M}_1 := [\boldsymbol{q}(t_1), \ldots, \boldsymbol{q}(t_N), \boldsymbol{p}(t_1), \ldots, \boldsymbol{p}(t_N)]$.
2: Compute the SVD of $\boldsymbol{M}_1$ to obtain a POD basis matrix $\boldsymbol{\Phi}$.
3: Construct the symplectic matrix $\boldsymbol{A} = \operatorname{diag}(\boldsymbol{\Phi}, \boldsymbol{\Phi})$.

PSD solves the following optimization problem

$$\begin{aligned} &\text{minimize} && \|\boldsymbol{M}_x - \boldsymbol{A}\boldsymbol{A}^{+}\boldsymbol{M}_x\|_F \\ &\text{subject to} && \boldsymbol{A} \in \mathbb{M}(2n, 2k), \end{aligned} \qquad \boldsymbol{M}_x = \big[\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_N)\big]$$

$$\mathbb{M}(2n, 2k) = \left\{ \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Phi} \end{bmatrix} \,\middle|\, \boldsymbol{\Phi}^T\boldsymbol{\Phi} = \boldsymbol{I}_k,\ \boldsymbol{\Phi} \in \mathbb{R}^{n\times k} \right\}.$$

**Algorithm** PSD Algorithm

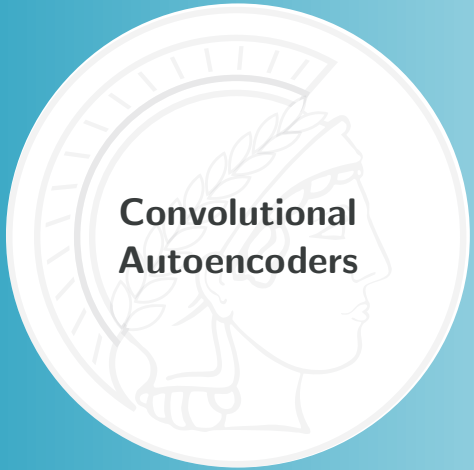
**Require:** An empirical data ensemble $\{\boldsymbol{q}(t_i), \boldsymbol{p}(t_i)\}_{i=1}^{N}$.
**Ensure:** A symplectic matrix $\boldsymbol{A} \in \mathbb{M}(2n, 2k)$ in block-diagonal form.
1: Construct an extended snapshot matrix $\boldsymbol{M}_1 := [\boldsymbol{q}(t_1), \ldots, \boldsymbol{q}(t_N), \boldsymbol{p}(t_1), \ldots, \boldsymbol{p}(t_N)]$.
2: Compute the SVD of $\boldsymbol{M}_1$ to obtain a POD basis matrix $\boldsymbol{\Phi}$.
3: Construct the symplectic matrix $\boldsymbol{A} = \operatorname{diag}(\boldsymbol{\Phi}, \boldsymbol{\Phi})$.

PSD solves the following optimization problem

$$\begin{aligned} &\text{minimize} && \|\boldsymbol{M}_x - \boldsymbol{A}\boldsymbol{A}^{+}\boldsymbol{M}_x\|_F \\ &\text{subject to} && \boldsymbol{A} \in \mathbb{M}(2n, 2k), \end{aligned} \qquad \boldsymbol{M}_x = \big[\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_N)\big]$$

$$\mathbb{M}(2n, 2k) = \left\{ \begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Phi} \end{bmatrix} \,\middle|\, \boldsymbol{\Phi}^T\boldsymbol{\Phi} = \boldsymbol{I}_k,\ \boldsymbol{\Phi} \in \mathbb{R}^{n\times k} \right\}.$$

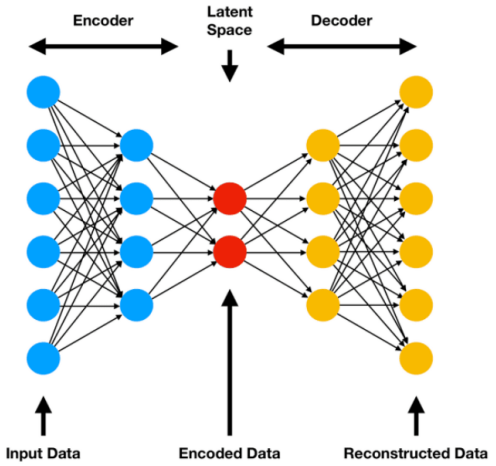
(Linear) model order reduction is done via symplectic projection with $\boldsymbol{A}$, i.e., $\hat{\boldsymbol{x}} = \boldsymbol{A}^{+}\boldsymbol{x}$.



# Convolutional Autoencoders

# Autoencoder

An autoencoder $\psi = \boldsymbol{D} \circ \boldsymbol{E} : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ consists of an encoder $\boldsymbol{E} : \mathbb{R}^n \to \mathbb{R}^k$, that maps the data to a latent space (here $\mathbb{R}^{2k}$) and a decoder $\boldsymbol{D} : \mathbb{R}^k \to \mathbb{R}^n$, which maps back to the original space.



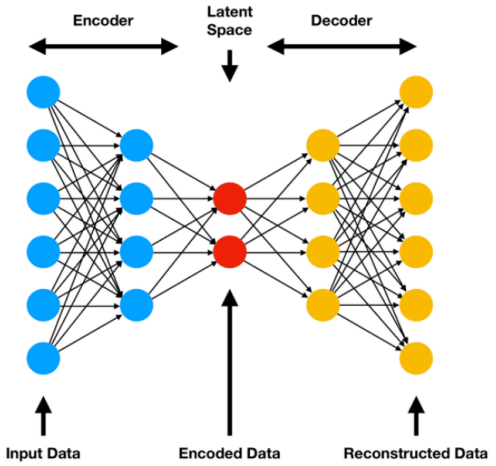

[1] https://www.programming-ocean.com/ai-architecutres/auto-encoder-achitecture.php

# Autoencoder

An autoencoder $\psi = \boldsymbol{D} \circ \boldsymbol{E} : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ consists of an encoder $\boldsymbol{E} : \mathbb{R}^n \to \mathbb{R}^k$, that maps the data to a latent space (here $\mathbb{R}^{2k}$) and a decoder $\boldsymbol{D} : \mathbb{R}^k \to \mathbb{R}^n$, which maps back to the original space.

The autonecoder is usually trained to optimize the reconstruction loss

$$\mathcal{L} = \|\psi(\boldsymbol{X}) - X\|^2 .$$




[1] https://www.programming-ocean.com/ai-architecutres/auto-encoder-achitecture.php

# 1D Convolutional Layer

Continuous convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)\,dy$$

[1] https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks

# 1D Convolutional Layer

Continuous convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)\,dy$$

Discrete convolution on $[-M, M]$:

$$(f * g)(n) = \sum_{m=-M}^{M} f(m)g(n-m)$$

[1] https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks

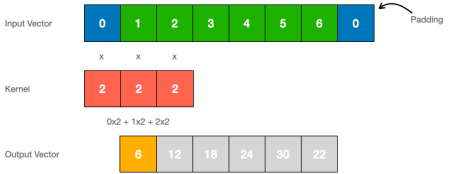
# 1D Convolutional Layer

Continuous convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)\,dy$$

Discrete convolution on $[-M, M]$:

$$(f * g)(n) = \sum_{m=-M}^{M} f(m)g(n-m)$$




**Padding**

```
nn.Conv1d(in_channels=1, out_channels=1,
  kernel_size=3, padding=1,
  bias=False
)
```

**Size**

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel\_size} - 1) - 1}{\text{stride}} + 1 \right\rfloor$$

$$6 = \frac{6 + (2 * 1) - 1 * (3 - 1) - 1}{1} + 1$$

jinglescode.github.io

[1] https://jinglescode.github.io/2020/11/01/how-convolutional-layers-work-deep-learning-neural-networks

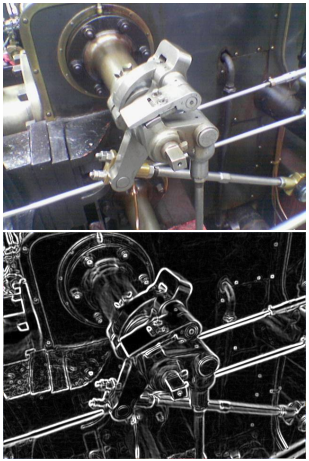
# 1D Convolutional Layer

Any 1D convolutional layer with zero-padding can be written as

$$\begin{bmatrix} \boldsymbol{T}_{1,1} & \cdots & \boldsymbol{T}_{1,C_{\text{in}}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{T}_{C_{\text{out}},1} & \cdots & \boldsymbol{T}_{C_{\text{out}},C_{\text{in}}} \end{bmatrix} \boldsymbol{x} = \boldsymbol{y}, \qquad \boldsymbol{T}_{i,j} = \begin{bmatrix} \boldsymbol{w}_{i,j,m} & \cdots & \boldsymbol{w}_{i,j,l} & & & & \\ \vdots & \ddots & & \ddots & & & \\ \boldsymbol{w}_{i,j,1} & \cdots & \boldsymbol{w}_{i,j,m} & \cdots & \boldsymbol{w}_{i,j,l} & & \\ & \ddots & & \ddots & & \ddots & \\ & & \boldsymbol{w}_{i,j,1} & \cdots & \boldsymbol{w}_{i,j,m} & \cdots & \boldsymbol{w}_{i,j,l} \\ & & & \ddots & & \ddots & \\ & & & & \boldsymbol{w}_{i,j,1} & \cdots & \boldsymbol{w}_{i,j,m} \end{bmatrix} \in \mathbb{R}^{N \times N}$$

for $i = 1, \ldots, C_{\text{out}},\ j = 1, \ldots, C_{\text{in}},\ m = (l+1)/2$ for odd kernel length $l$ and where $\boldsymbol{w} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times l}$ contains the weights of the convolutional layer and $\boldsymbol{x}, \boldsymbol{y}$ are the vectorized input and output signals.

# 2D Convolutional Layer



| | | | | |
|---|---|---|---|---|
| $3_0$ | $3_1$ | $2_2$ | 1 | 0 |
| $0_2$ | $0_2$ | $1_0$ | 3 | 1 |
| $3_0$ | $1_1$ | $2_2$ | 2 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 2 | 0 | 0 | 0 | 1 |

| | | |
|---|---|---|
| 12.0 | 12.0 | 17.0 |
| 10.0 | 17.0 | 19.0 |
| 9.0 | 6.0 | 14.0 |

Source: `https://en.wikipedia.org/wiki/Sobel_operator`

[1] Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv:1603.07285.

# 2D Convolutional Layer

$$\tau_{i,j,k} = \begin{bmatrix} w_{i,j,m_1,k} & \cdots & w_{i,j,l_1,k} & & & & \\ \vdots & \ddots & & \ddots & & & \\ w_{i,j,1,k} & \cdots & w_{i,j,m_1,k} & \cdots & w_{i,j,l_1,k} & & \\ & \ddots & & \ddots & & \ddots & \\ & & w_{i,j,1,k} & \cdots & w_{i,j,m_1,k} & \cdots & w_{i,j,l_1,k} \\ & & & \ddots & & \ddots & \\ & & & & w_{i,j,1,k} & \cdots & w_{i,j,m_1,k} \end{bmatrix}$$

# 2D Convolutional Layer

$$\boldsymbol{T}_{ij} = \begin{bmatrix} \boldsymbol{\tau}_{i,j,m_2} & \cdots & \boldsymbol{\tau}_{i,j,l_2} & & & & \\ \vdots & \ddots & & \ddots & & & \\ \boldsymbol{\tau}_{i,j,1} & \cdots & \boldsymbol{\tau}_{i,j,m_2} & \cdots & \boldsymbol{\tau}_{i,j,l_2} & & \\ & \ddots & & \ddots & & \ddots & \\ & & \boldsymbol{\tau}_{i,j,1} & \cdots & \boldsymbol{\tau}_{i,j,m_2} & \cdots & \boldsymbol{\tau}_{i,j,l_2} \\ & & & \ddots & & \ddots & \\ & & & & \boldsymbol{\tau}_{i,j,1} & \cdots & \boldsymbol{\tau}_{i,j,m_2} \end{bmatrix},$$

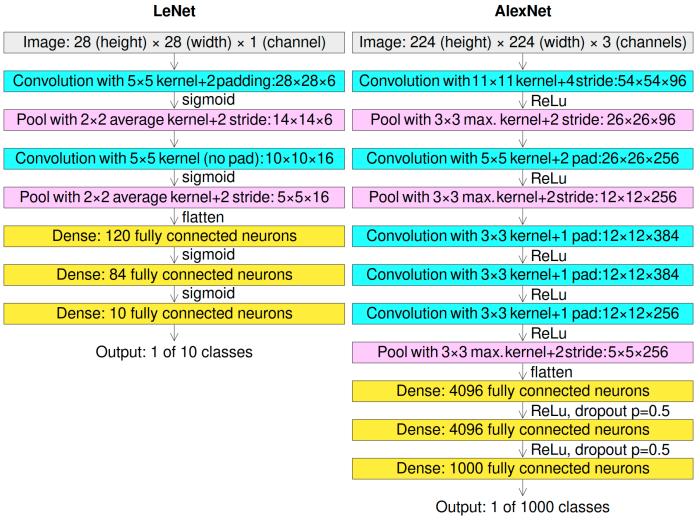
# 2D Convolutional Layer

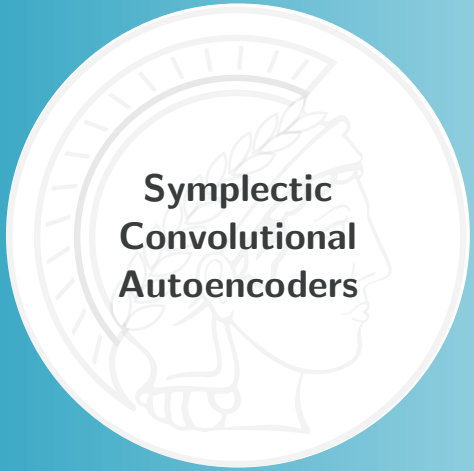
$$\boldsymbol{T}_{ij} = \begin{bmatrix} \boldsymbol{\tau}_{i,j,m_2} & \cdots & \boldsymbol{\tau}_{i,j,l_2} & & & & \\ \vdots & \ddots & & \ddots & & & \\ \boldsymbol{\tau}_{i,j,1} & \cdots & \boldsymbol{\tau}_{i,j,m_2} & \cdots & \boldsymbol{\tau}_{i,j,l_2} & & \\ & \ddots & & \ddots & & \ddots & \\ & & \boldsymbol{\tau}_{i,j,1} & \cdots & \boldsymbol{\tau}_{i,j,m_2} & \cdots & \boldsymbol{\tau}_{i,j,l_2} \\ & & & \ddots & & \ddots & \\ & & & & \boldsymbol{\tau}_{i,j,1} & \cdots & \boldsymbol{\tau}_{i,j,m_2} \end{bmatrix}, \qquad \begin{bmatrix} \boldsymbol{T}_{1,1} & \cdots & \boldsymbol{T}_{1,C_{\text{in}}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{T}_{C_{\text{out}},1} & \cdots & \boldsymbol{T}_{C_{\text{out}},C_{\text{in}}} \end{bmatrix} \boldsymbol{x} = \boldsymbol{y}.$$

# Convolutional Autoencoder

A convolutional autoencoder usually consists of four different kinds of layers:

- Convolutional layers
- Activation layers
- Pooling layers
- Fully connected linear layers





# Symplectic Convolutional Autoencoders

Shear maps are symplectic maps given by

$$f_{\text{up}}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{p}\end{bmatrix} = \begin{bmatrix}\boldsymbol{I} & \nabla V\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{p}\end{bmatrix} := \begin{pmatrix}\boldsymbol{q}+\nabla V(\boldsymbol{p})\\ \boldsymbol{p}\end{pmatrix},$$

$$f_{\text{low}}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{p}\end{bmatrix} = \begin{bmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \nabla V & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{p}\end{bmatrix} := \begin{pmatrix}\boldsymbol{q}\\ \boldsymbol{p}+\nabla V(\boldsymbol{q})\end{pmatrix}$$

for some $V : \mathbb{R}^d \to \mathbb{R}$.

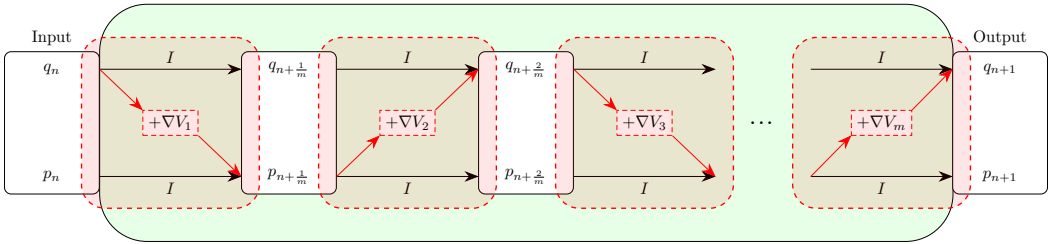
Shear maps are symplectic maps given by

$$f_{\text{up}}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{p}\end{bmatrix} = \begin{bmatrix}\boldsymbol{I} & \nabla V\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{p}\end{bmatrix} := \begin{pmatrix}\boldsymbol{q}+\nabla V(\boldsymbol{p})\\ \boldsymbol{p}\end{pmatrix},$$

$$f_{\text{low}}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{p}\end{bmatrix} = \begin{bmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \nabla V & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{p}\end{bmatrix} := \begin{pmatrix}\boldsymbol{q}\\ \boldsymbol{p}+\nabla V(\boldsymbol{q})\end{pmatrix}$$

for some $V : \mathbb{R}^d \to \mathbb{R}$.

SympNets consist of compositions of shear maps



LA-SympNets have linear and activation modules.

- Linear modules $v_i \in \mathcal{M}_{\mathsf{L}}$: Composition of $k$ linear layers with $\nabla V_j(\boldsymbol{x}) := \boldsymbol{S}_j \boldsymbol{x}$ with trainable parameters $\boldsymbol{S}_j = \boldsymbol{S}_j^T,\ \boldsymbol{S}_j \in \mathbb{R}^{d \times d}$:

$$v_i \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} := \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0}/\boldsymbol{S}_k \\ \boldsymbol{S}_k/\boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \cdots \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{S}_2 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{S}_1 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}.$$

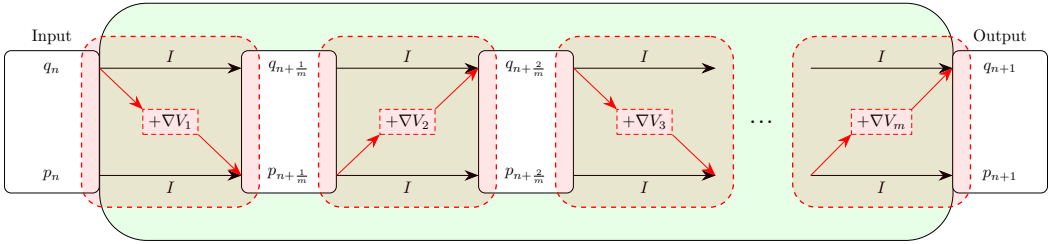
LA-SympNets have linear and activation modules.

- Linear modules $v_i \in \mathcal{M}_{\mathsf{L}}$: Composition of $k$ linear layers with $\nabla V_j(\boldsymbol{x}) := \boldsymbol{S}_j \boldsymbol{x}$ with trainable parameters $\boldsymbol{S}_j = \boldsymbol{S}_j^T,\ \boldsymbol{S}_j \in \mathbb{R}^{d \times d}$:

$$v_i \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} := \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0}/\boldsymbol{S}_k \\ \boldsymbol{S}_k/\boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \cdots \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{S}_2 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{S}_1 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}.$$

- Activation modules $w_i \in \mathcal{M}_{\mathsf{A}}$: $\nabla V_i(\boldsymbol{x}) = \tilde{\sigma}_{\boldsymbol{a}_i, \boldsymbol{b}_i}(x) := \operatorname{diag}(\boldsymbol{a}_i)\, \sigma(\boldsymbol{x} + \boldsymbol{b}_i)$ with trainable parameters $\boldsymbol{a}_i, \boldsymbol{b}_i \in \mathbb{R}^d$ and activation function $\sigma$:

$$w_i^{\mathsf{up}} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \tilde{\sigma}_{\boldsymbol{a}_i, \boldsymbol{b}_i} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}, \qquad w_i^{\mathsf{low}} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \tilde{\sigma}_{\boldsymbol{a}_i, \boldsymbol{b}_i} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}.$$

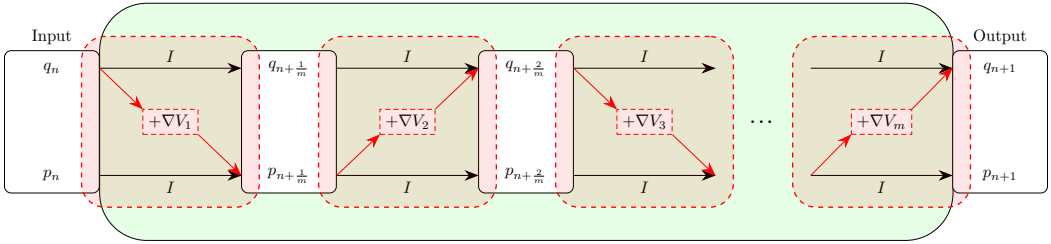



LA-SympNets are given by finite compositions of the respective modules

$$\Psi_{\mathsf{LA}} := \{\psi = v_{k+1} \circ w_k \circ v_k \circ \ldots \circ w_1 \circ v_1 \,|\, v_1, \ldots, v_{k+1} \in \mathcal{M}_{\mathsf{L}},\ w_1, \ldots, w_k \in \mathcal{M}_{\mathsf{A}}\}\,.$$

# Symplectic Convolutional Autoencoders

## LA-SympNets [Jin et al. 2020] in a nutshell




LA-SympNets are given by finite compositions of the respective modules

$$\Psi_{\mathsf{LA}} := \{\psi = v_{k+1} \circ w_k \circ v_k \circ \ldots \circ w_1 \circ v_1 \,|\, v_1, \ldots, v_{k+1} \in \mathcal{M}_{\mathsf{L}},\ w_1, \ldots, w_k \in \mathcal{M}_{\mathsf{A}}\}\,.$$

LA-SympNets can approximate arbitrary symplectic maps.

[Jin et al. 2020, B./Janik 2025 (symplectic universal approximation theorem)].

**Note:** original SympNet formulation required data on uniform time grid, generalized in [B./Janik 2025].

# Symplectic Convolutional Layers

The idea behind the symplectic convolutional layers is replacing the symmetric matrices $\boldsymbol{S}_j$ with symmetric Toeplitz matrices $\boldsymbol{T}_j = \boldsymbol{T}_j^T$

$$v_i \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} := \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0}/\boldsymbol{T}_k \\ \boldsymbol{T}_k/\boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \cdots \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{T}_2 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{T}_1 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}.$$

# Symplectic Convolutional Layers

The idea behind the symplectic convolutional layers is replacing the symmetric matrices $\boldsymbol{S}_j$ with symmetric Toeplitz matrices $\boldsymbol{T}_j = \boldsymbol{T}_j^T$

$$v_i \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} := \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0}/\boldsymbol{T}_k \\ \boldsymbol{T}_k/\boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \cdots \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{T}_2 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{T}_1 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}.$$

Since $\boldsymbol{0}$ and $\boldsymbol{I}$ are Toeplitz matrices, too, $v_i$ is a composition of 1D convolutions with 2 input and 2 output channels.

# Symplectic Convolutional Layers

The idea behind the symplectic convolutional layers is replacing the symmetric matrices $\boldsymbol{S}_j$ with symmetric Toeplitz matrices $\boldsymbol{T}_j = \boldsymbol{T}_j^T$

$$v_i \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} := \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0}/\boldsymbol{T}_k \\ \boldsymbol{T}_k/\boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \cdots \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{T}_2 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{T}_1 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}.$$

Since $\boldsymbol{0}$ and $\boldsymbol{I}$ are Toeplitz matrices, too, $v_i$ is a composition of 1D convolutions with 2 input and 2 output channels.

What if we want to change the number of channels?

# Symplectic Convolutional Layers

Assume that we want to double the number of channels and we have two input channels

$$\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \mapsto \begin{bmatrix} \boldsymbol{q}_1 \\ \boldsymbol{p}_1 \\ \boldsymbol{q}_2 \\ \boldsymbol{p}_2 \end{bmatrix} = \left[ \begin{bmatrix} \boldsymbol{I} & \boldsymbol{T}_1 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{T}_2 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \right].$$

# Symplectic Convolutional Layers

Assume that we want to double the number of channels and we have two input channels

$$\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \mapsto \begin{bmatrix} \boldsymbol{q}_1 \\ \boldsymbol{p}_1 \\ \boldsymbol{q}_2 \\ \boldsymbol{p}_2 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{T}_1 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \\ \begin{bmatrix} \boldsymbol{I} & \boldsymbol{T}_2 \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \end{bmatrix}.$$

Reordering leads to

$$\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \mapsto \begin{bmatrix} \boldsymbol{q}_1 \\ \boldsymbol{q}_2 \\ \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{T}_1 \\ \boldsymbol{I} & \boldsymbol{T}_2 \\ \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}.$$

# Symplectic Convolutional Layers

To make the mapping symplectic we need to scale the identity matrices

$$\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \mapsto \begin{bmatrix} c\boldsymbol{I} & \boldsymbol{T}_1 \\ c\boldsymbol{I} & \boldsymbol{T}_2 \\ \boldsymbol{0} & c\boldsymbol{I} \\ \boldsymbol{0} & c\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix},$$

where $2c^2 = 1$.

# Symplectic Convolutional Layers

To make the mapping symplectic we need to scale the identity matrices

$$\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \mapsto \begin{bmatrix} c\boldsymbol{I} & \boldsymbol{T}_1 \\ c\boldsymbol{I} & \boldsymbol{T}_2 \\ \boldsymbol{0} & c\boldsymbol{I} \\ \boldsymbol{0} & c\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix},$$

where $2c^2 = 1$. In general, the *set of 1D symplectic convolutional lifting matrices* with $C_{\text{in}}$ input channels and $C_{\text{out}}$ output channels $\mathbb{C}_{\text{1D}}(C_{\text{out}}, C_{\text{in}}, N)$ is given by

$$\mathbb{C}_{\text{1D}}(C_{\text{out}}, C_{\text{in}}, N) := \left\{ \begin{bmatrix} c\boldsymbol{I} & \boldsymbol{T}_1 \\ \vdots & \vdots \\ c\boldsymbol{I} & \boldsymbol{T}_d \\ \boldsymbol{0} & c\boldsymbol{I} \\ \vdots & \vdots \\ \boldsymbol{0} & c\boldsymbol{I} \end{bmatrix} \in \mathbb{R}^{C_{\text{out}}N \times C_{\text{in}}N} \;\middle|\; \begin{array}{l} \boldsymbol{T}_i \in \mathbb{T}^{\text{1D}}_{\text{sym}}(C_{\text{in}}/2, N), \\ c = \sqrt{\frac{1}{d}}, \\ d = \frac{C_{\text{out}}}{C_{\text{in}}} \end{array} \right\} \cup \left\{ \begin{bmatrix} c\boldsymbol{I} & \boldsymbol{0} \\ \vdots & \vdots \\ c\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{T}_1 & c\boldsymbol{I} \\ \vdots & \vdots \\ \boldsymbol{T}_d & c\boldsymbol{I} \end{bmatrix} \in \mathbb{R}^{C_{\text{out}}N \times C_{\text{in}}N} \;\middle|\; \begin{array}{l} \boldsymbol{T}_i \in \mathbb{T}^{\text{1D}}_{\text{sym}}(C_{\text{in}}/2, N), \\ c = \sqrt{\frac{1}{d}}, \\ d = \frac{C_{\text{out}}}{C_{\text{in}}} \end{array} \right\},$$

# Symplectic Convolutional Layers

To make the mapping symplectic we need to scale the identity matrices

$$\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \mapsto \begin{bmatrix} c\boldsymbol{I} & \boldsymbol{T}_1 \\ c\boldsymbol{I} & \boldsymbol{T}_2 \\ \boldsymbol{0} & c\boldsymbol{I} \\ \boldsymbol{0} & c\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix},$$

where $2c^2 = 1$. In general, the *set of 1D symplectic convolutional lifting matrices* with $C_{\text{in}}$ input channels and $C_{\text{out}}$ output channels $\mathbb{C}_{\text{1D}}(C_{\text{out}}, C_{\text{in}}, N)$ is given by

$$\mathbb{C}_{\text{1D}}(C_{\text{out}}, C_{\text{in}}, N) := \left\{ \begin{bmatrix} c\boldsymbol{I} & \boldsymbol{T}_1 \\ \vdots & \vdots \\ c\boldsymbol{I} & \boldsymbol{T}_d \\ \boldsymbol{0} & c\boldsymbol{I} \\ \vdots & \vdots \\ \boldsymbol{0} & c\boldsymbol{I} \end{bmatrix} \in \mathbb{R}^{C_{\text{out}}N \times C_{\text{in}}N} \;\middle|\; \begin{array}{l} \boldsymbol{T}_i \in \mathbb{T}^{\text{1D}}_{\text{sym}}(C_{\text{in}}/2, N), \\ c = \sqrt{\frac{1}{d}}, \\ d = \frac{C_{\text{out}}}{C_{\text{in}}} \end{array} \right\} \cup \left\{ \begin{bmatrix} c\boldsymbol{I} & \boldsymbol{0} \\ \vdots & \vdots \\ c\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{T}_1 & c\boldsymbol{I} \\ \vdots & \vdots \\ \boldsymbol{T}_d & c\boldsymbol{I} \end{bmatrix} \in \mathbb{R}^{C_{\text{out}}N \times C_{\text{in}}N} \;\middle|\; \begin{array}{l} \boldsymbol{T}_i \in \mathbb{T}^{\text{1D}}_{\text{sym}}(C_{\text{in}}/2, N), \\ c = \sqrt{\frac{1}{d}}, \\ d = \frac{C_{\text{out}}}{C_{\text{in}}} \end{array} \right\},$$

We can reduce the number of channels by using the symplectic inverse of the symplectic convolutional lift matrices.

# Symplectic Activation Layers

We use the same symplectic activation layers as the SympNets, because they only perform component-wise operations.

# Symplectic Activation Layers

We use the same symplectic activation layers as the SympNets, because they only perform component-wise operations.

Activation modules $w_i \in \mathcal{M}_{\mathsf{A}}$:

$$\nabla V_i(\boldsymbol{x}) = \tilde{\sigma}_{\boldsymbol{a}_i,\boldsymbol{b}_i}(x) := \operatorname{diag}(\boldsymbol{a}_i)\,\sigma(\boldsymbol{x} + \boldsymbol{b}_i)$$

with trainable parameters $\boldsymbol{a}_i, \boldsymbol{b}_i \in \mathbb{R}^d$ and activation function $\sigma$.

$$w_i^{\mathsf{up}} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \tilde{\sigma}_{\boldsymbol{a}_i,\boldsymbol{b}_i} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}, \qquad w_i^{\mathsf{low}} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \tilde{\sigma}_{\boldsymbol{a}_i,\boldsymbol{b}_i} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}.$$

# Symplectic Activation Layers

We use the same symplectic activation layers as the SympNets, because they only perform component-wise operations.

Activation modules $w_i \in \mathcal{M}_{\mathsf{A}}$:

$$\nabla V_i(\boldsymbol{x}) = \tilde{\sigma}_{\boldsymbol{a}_i,\boldsymbol{b}_i}(x) := \operatorname{diag}(\boldsymbol{a}_i)\,\sigma(\boldsymbol{x} + \boldsymbol{b}_i)$$

with trainable parameters $\boldsymbol{a}_i, \boldsymbol{b}_i \in \mathbb{R}^d$ and activation function $\sigma$.

$$w_i^{\mathsf{up}} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \tilde{\sigma}_{\boldsymbol{a}_i,\boldsymbol{b}_i} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}, \qquad w_i^{\mathsf{low}} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \tilde{\sigma}_{\boldsymbol{a}_i,\boldsymbol{b}_i} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}.$$

We combine symplectic convolutional layers and symplectic activation layers to symplectic convolutional blocks

$$\Psi_{\mathsf{CB}} = \left\{ \boldsymbol{\psi} = v_{k+1} \circ w_k \circ v_k \circ \ldots \circ w_1 \circ v_1 \,\middle|\, v_i \in \mathbb{C}_{\mathsf{1D}}(C^i_{\mathsf{out}}, C^i_{\mathsf{in}}, N), w_i \in \mathcal{M}_{\mathsf{A}} \right\}.$$

# PSD-like Layers

Proper symplectic decomposition (PSD) [Peng/Mohseni 2016] uses orthogonal symplectic projection matrices

$$\mathbb{M}(2n, 2k) := Sp(2k, \mathbb{R}^n) \cap \left\{ \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi} \end{bmatrix} \middle| \, \boldsymbol{\Phi} \in \mathbb{R}^{n \times k} \right\} = \left\{ \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi} \end{bmatrix} \middle| \, \boldsymbol{\Phi}^T \boldsymbol{\Phi} = \boldsymbol{I}_k, \boldsymbol{\Phi} \in \mathbb{R}^{n \times k} \right\}.$$

# PSD-like Layers

Proper symplectic decomposition (PSD) [Peng/Mohseni 2016] uses orthogonal symplectic projection matrices

$$\mathbb{M}(2n, 2k) := Sp(2k, \mathbb{R}^n) \cap \left\{ \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi} \end{bmatrix} \middle| \, \boldsymbol{\Phi} \in \mathbb{R}^{n \times k} \right\} = \left\{ \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi} \end{bmatrix} \middle| \, \boldsymbol{\Phi}^T \boldsymbol{\Phi} = \boldsymbol{I}_k, \boldsymbol{\Phi} \in \mathbb{R}^{n \times k} \right\}.$$

Let $n, k \in \mathbb{N},\ n \geq k$, then the *set of PSD-like modules* is given by

$$\begin{aligned} \mathcal{M}_{\mathsf{PSD}} &:= \left\{ x \mapsto A^+ x \,\middle|\, A \in \mathbb{M}(2n, 2k) \right\} \\ &= \left\{ x \mapsto \begin{bmatrix} \boldsymbol{\Psi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Psi} \end{bmatrix} x \middle| \, \boldsymbol{\Psi}\boldsymbol{\Psi}^T = \boldsymbol{I}_k,\ \boldsymbol{\Psi} \in \mathbb{R}^{k \times n} \right\}. \end{aligned}$$

# PSD-like Layers

Proper symplectic decomposition (PSD) [Peng/Mohseni 2016] uses orthogonal symplectic projection matrices

$$\mathbb{M}(2n, 2k) := Sp(2k, \mathbb{R}^n) \cap \left\{ \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi} \end{bmatrix} \middle| \, \boldsymbol{\Phi} \in \mathbb{R}^{n \times k} \right\} = \left\{ \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi} \end{bmatrix} \middle| \, \boldsymbol{\Phi}^T \boldsymbol{\Phi} = \boldsymbol{I}_k, \boldsymbol{\Phi} \in \mathbb{R}^{n \times k} \right\}.$$

Let $n, k \in \mathbb{N},\ n \geq k$, then the *set of PSD-like modules* is given by

$$\begin{aligned} \mathcal{M}_{\text{PSD}} &:= \left\{ x \mapsto A^+ x \,\middle|\, A \in \mathbb{M}(2n, 2k) \right\} \\ &= \left\{ x \mapsto \begin{bmatrix} \boldsymbol{\Psi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Psi} \end{bmatrix} x \,\middle|\, \boldsymbol{\Psi}\boldsymbol{\Psi}^T = \boldsymbol{I}_k,\ \boldsymbol{\Psi} \in \mathbb{R}^{k \times n} \right\}. \end{aligned}$$

[Brantner/Kraus 2023] showed that PSD-like modules can be used in autoencoder architectures for dimension reduction.

# Symplectic Pooling

Consider the following one-channel 1D input signal as an example, to derive the general definition:

$$\boldsymbol{q} = [2, 1, 3, 5]^T.$$

A 1D `max-pooling` operation $\mathrm{MaxPool}$ with a stride and kernel size of two, no padding and dilation of one can be equivalently written by multiplication with its Jacobian at $x$:

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{so that} \quad \boldsymbol{\Phi}(\boldsymbol{q})\boldsymbol{q} = \begin{bmatrix} 2 \\ 5 \end{bmatrix}.$$

# Symplectic Pooling

Consider the following one-channel 1D input signal as an example, to derive the general definition:

$$\boldsymbol{q} = [2, 1, 3, 5]^T.$$

A 1D `max-pooling` operation $\mathrm{MaxPool}$ with a stride and kernel size of two, no padding and dilation of one can be equivalently written by multiplication with its Jacobian at $x$:

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{so that} \quad \boldsymbol{\Phi}(\boldsymbol{q})\boldsymbol{q} = \begin{bmatrix} 2 \\ 5 \end{bmatrix}.$$

Apply same pooling operation to 2 different channels to preserve the symplecticity condition

$$\boldsymbol{P}\left(\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}\right) = \begin{bmatrix} \boldsymbol{\Phi}(\boldsymbol{q}) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Phi}(\boldsymbol{q}) \end{bmatrix}.$$

# Symplectic Pooling

**Definition (Symplectic pooling)**

Let $\mathrm{MaxPool}$ be the `max-pooling` operation with dilation of one, padding of zero and stride and kernel size $k \in \mathbb{N}$. Furthermore, let $\boldsymbol{\Phi}(\boldsymbol{x}) \in \mathbb{R}^{N/k \times N}$ be the Jacobian of $\mathrm{MaxPool}$ at $\boldsymbol{x} \in \mathbb{R}^N$, where $k$ divides $N$, i.e., $k \mid N$. The *symplectic pooling modules* for two input channels are given by

$$p_{\mathsf{up}} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} := \begin{bmatrix} \boldsymbol{\Phi}(\boldsymbol{q})\boldsymbol{q} \\ \boldsymbol{\Phi}(\boldsymbol{q})\boldsymbol{p} \end{bmatrix}, \qquad p_{\mathsf{low}} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} := \begin{bmatrix} \boldsymbol{\Phi}(\boldsymbol{p})\boldsymbol{q} \\ \boldsymbol{\Phi}(\boldsymbol{p})\boldsymbol{p} \end{bmatrix}.$$

We denote the *set of symplectic pooling modules* by

$$\mathcal{M}_{\mathsf{P}} := \{p \mid p \text{ is a symplectic pooling module}\}.$$

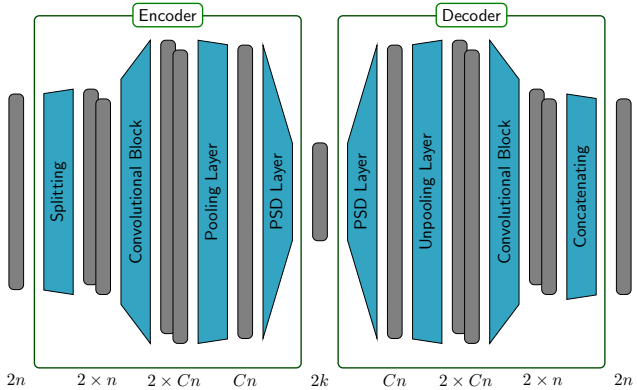



Figure: Schematic description of the 1D symplectic autoencoder architecture used for the wave equation and NLS examples. $2n$ is the full state dimension, while $2k$ describes the size of the latent space. $C = \Pi_i \frac{C_{\text{out}}^{(i)}}{C_{\text{in}}^{(i)}}$ is the product of the quotients of the output and input channels of all the convolutional layers.

# Numerical Examples

# Wave Equation

First, we consider the one-dimensional linear wave equation

$$\begin{aligned} u_{tt}(x,t) &= cu_{xx}(x,t), \\ u(x,0) &= u^0(x), \\ u_t(x,0) &= u_t^0(x), \qquad x \in \Omega. \end{aligned}$$

# Wave Equation

First, we consider the one-dimensional linear wave equation

$$\begin{aligned} u_{tt}(x,t) &= cu_{xx}(x,t), \\ u(x,0) &= u^0(x), \\ u_t(x,0) &= u_t^0(x), \qquad x \in \Omega. \end{aligned}$$

To demonstrate the Hamiltonian form of the wave equation, let us define the variables $p = u_t$ and $q = u$, which yields the Hamiltonian form

$$\frac{\partial z}{\partial t} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \frac{\delta H}{\delta z}, \quad z = \begin{bmatrix} q \\ p \end{bmatrix},$$

# Wave Equation

First, we consider the one-dimensional linear wave equation

$$\begin{aligned} u_{tt}(x,t) &= cu_{xx}(x,t), \\ u(x,0) &= u^0(x), \\ u_t(x,0) &= u_t^0(x), \qquad x \in \Omega. \end{aligned}$$

To demonstrate the Hamiltonian form of the wave equation, let us define the variables $p = u_t$ and $q = u$, which yields the Hamiltonian form

$$\frac{\partial z}{\partial t} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \frac{\delta H}{\delta z}, \quad z = \begin{bmatrix} q \\ p \end{bmatrix},$$

where $\delta$ denotes the variational derivative and the Hamiltonian is given as

$$H(z) = \frac{1}{2} \int_\Omega c q_x^2 + p^2 \, dx.$$

# Wave Equation

We discretize the space using a structure-preserving finite difference approach with $N$ equidistant grid points, which leads to the following semi-discrete Hamiltonian:

$$H = \sum_{i=1}^{N} \Delta x \left[ \frac{1}{2} \boldsymbol{p}_i^2 + \frac{c(\boldsymbol{q}_{i+1} - \boldsymbol{q}_i)^2}{2\Delta x^2} + \frac{c(\boldsymbol{q}_i - \boldsymbol{q}_{i-1})^2}{2\Delta x^2} \right]$$

where $\boldsymbol{p}_i = u_t(x_i, t), \boldsymbol{q}_i = u(t, x_i)$, and $x_i = i\Delta x$.

# Wave Equation

We discretize the space using a structure-preserving finite difference approach with $N$ equidistant grid points, which leads to the following semi-discrete Hamiltonian:

$$H = \sum_{i=1}^{N} \Delta x \left[ \frac{1}{2} \boldsymbol{p}_i^2 + \frac{c(\boldsymbol{q}_{i+1} - \boldsymbol{q}_i)^2}{2\Delta x^2} + \frac{c(\boldsymbol{q}_i - \boldsymbol{q}_{i-1})^2}{2\Delta x^2} \right]$$

where $\boldsymbol{p}_i = u_t(x_i, t), \boldsymbol{q}_i = u(t, x_i)$, and $x_i = i\Delta x$. The semi-discrete Hamiltonian form of the wave equation is expressed as follows:

$$\frac{d\boldsymbol{z}}{dt} = \boldsymbol{K}\boldsymbol{z}, \quad \boldsymbol{z} = \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}, \quad \boldsymbol{K} = \begin{bmatrix} \boldsymbol{0}_N & \boldsymbol{I}_N \\ c\boldsymbol{D}_{xx} & \boldsymbol{0}_N \end{bmatrix}.$$

# Wave Equation

To preserve the symplectic structure of the wave equation, we discretize it using the symplectic Euler method:

$$\boldsymbol{q}^{n+1} = \boldsymbol{q}^n + \Delta t \cdot \frac{\partial H}{\partial p}(\boldsymbol{q}^n, \boldsymbol{p}^n),$$

$$\boldsymbol{p}^{n+1} = \boldsymbol{p}^n - \Delta t \cdot \frac{\partial H}{\partial q}(\boldsymbol{q}^{n+1}, \boldsymbol{p}^n),$$

where the superscript $n$ denotes the time step $n$.

# Wave Equation

To preserve the symplectic structure of the wave equation, we discretize it using the symplectic Euler method:

$$\boldsymbol{q}^{n+1} = \boldsymbol{q}^n + \Delta t \cdot \frac{\partial H}{\partial p}(\boldsymbol{q}^n, \boldsymbol{p}^n),$$
$$\boldsymbol{p}^{n+1} = \boldsymbol{p}^n - \Delta t \cdot \frac{\partial H}{\partial q}(\boldsymbol{q}^{n+1}, \boldsymbol{p}^n),$$

where the superscript $n$ denotes the time step $n$. This yields the explicit scheme:

$$\boldsymbol{z}^{n+1} = \begin{bmatrix} \boldsymbol{I}_N & \Delta t \boldsymbol{I}_N \\ \boldsymbol{0}_N & \boldsymbol{I}_N \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_N & \boldsymbol{0}_N \\ c\Delta t \boldsymbol{D}_{xx} & \boldsymbol{I}_N \end{bmatrix} \boldsymbol{z}^n. \tag{1}$$

Notice that (1) has a symplectic structure that can be modeled through a symplectic convolutional module $y \in \mathcal{M}_{\text{Conv}}$.

# Numerical Setup

We discretize our examples according to the table below.

Table: Discretization

| | 1D SympCAE | | 2D SympCAE |
|---|---|---|---|
| | Wave Eqn. | NLS Eqn. | SG Eqn. |
| $N$ | 1024 | 1024 | 10000 |
| $N_t$ | 1024 | 200 | 100 |
| $T$ | 5 | 5 | 20 |

# Numerical Setup

We discretize our examples according to the table below.

Table: Discretization

| | 1D SympCAE | | 2D SympCAE |
|---|---|---|---|
| | Wave Eqn. | NLS Eqn. | SG Eqn. |
| $N$ | 1024 | 1024 | 10000 |
| $N_t$ | 1024 | 200 | 100 |
| $T$ | 5 | 5 | 20 |

- We try tor recover the dynamics using SympCAE and PSD.

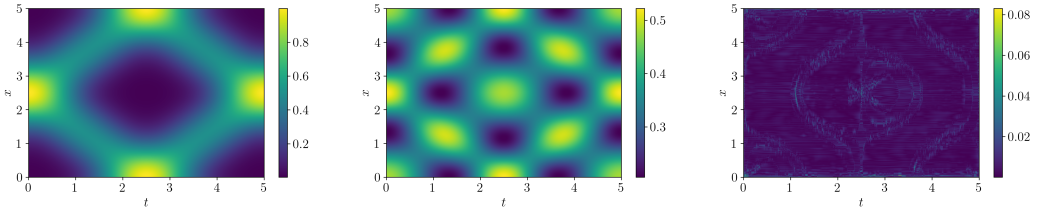
# Numerical Setup

We discretize our examples according to the table below.

Table: Discretization

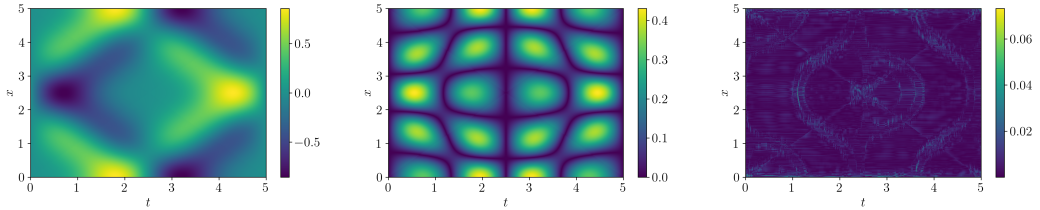
| | 1D SympCAE | | 2D SympCAE |
|---|---|---|---|
| | Wave Eqn. | NLS Eqn. | SG Eqn. |
| $N$ | 1024 | 1024 | 10000 |
| $N_t$ | 1024 | 200 | 100 |
| $T$ | 5 | 5 | 20 |

- We try tor recover the dynamics using SympCAE and PSD.
- Using latent trajectories obtained from SympCAE, we train a SympNet to extrapolate in time.




(a) Ground truth (b) Absolute PSD error (c) Absolute SympCAE error

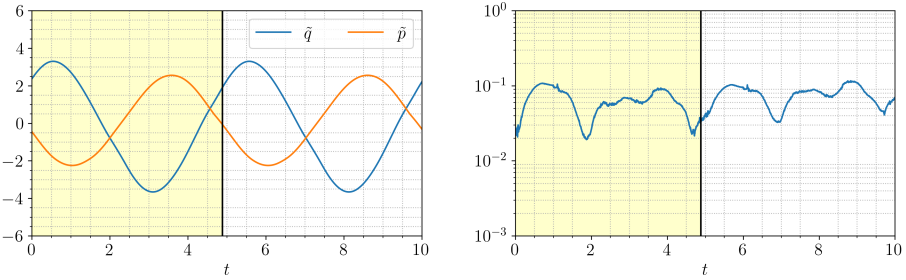
Figure: Linear wave equation: Absolute pointwise error between the ground truth solution for the state $q$ and the reconstructed solutions at latent dimension $r = 1$.




(a) Ground truth (b) Absolute PSD error (c) Absolute SympCAE error

Figure: Linear wave equation: Absolute pointwise error between the ground truth solution for the state $p$ and the reconstructed solutions at latent dimension $r = 1$.

# Wave Equation




(a) Latent trajectories

(b) Relative reconstruction error

Figure: A SympNet approach for learning the latent dynamics of the wave equation obtained by the SympCAE encoder. The vertical black lines separate the training and testing intervals.

# Wave Equation

To test the accuracy of the autoencoders, we use the following relative Frobenius error:

$$\varepsilon = \frac{\|\boldsymbol{X} - \boldsymbol{Y}\|_F}{\|\boldsymbol{X}\|_F}.$$

Table: Linear wave equation: The table shows the performance of the autoencoders obtained using PSD and SympCAE in capturing the dynamics of the ground truth model in terms of relative reconstruction error at latent dimensions $r = 1, 2, 3$. The best result for each latent dimension is highlighted in bold.

| $r$ | $\varepsilon_{\text{PSD}}$ | $\varepsilon_{\text{SympCAE}}$ |
|---|---|---|
| 1 | $7.28 \cdot 10^{-1}$ | $\mathbf{1.47 \cdot 10^{-2}}$ |
| 2 | $3.60 \cdot 10^{-1}$ | $\mathbf{1.21 \cdot 10^{-2}}$ |
| 3 | $7.20 \cdot 10^{-2}$ | $\mathbf{9.25 \cdot 10^{-3}}$ |

# Nonlinear Schrödinger Equation

We consider the NLS equation (2), which is one-dimensional, but nonlinear

$$\begin{aligned} \imath u_t(x,t) + \alpha u_{xx}(x,t) + \beta |u(x,t)|^2 u(x,t) &= 0, \\ u(x,0) &= u^0(x), \quad x \in \Omega. \end{aligned} \tag{2}$$

# Nonlinear Schrödinger Equation

We consider the NLS equation (2), which is one-dimensional, but nonlinear

$$\begin{aligned} \imath u_t(x,t) + \alpha u_{xx}(x,t) + \beta |u(x,t)|^2 u(x,t) &= 0, \\ u(x,0) &= u^0(x), \quad x \in \Omega. \end{aligned} \tag{2}$$

The canonical Hamiltonian form of the NLS equation (2) appears after expressing the complex-valued solution $u$ in terms of its imaginary and real parts as $u = p + \imath q$, which yields:

$$\begin{aligned} q_t &= p_{xx} + \beta(p^2 + q^2)p, \\ p_t &= -q_{xx} - \beta(p^2 + q^2)q, \end{aligned}$$

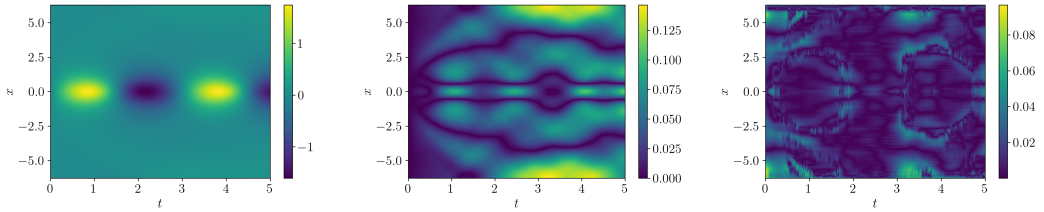
with the Hamiltonian

$$H(u) = \frac{1}{2} \int_\Omega (q_x^2 + p_x^2) + \frac{\beta}{2}(q^2 + p^2)^2 \, dx.$$

# Nonlinear Schrödinger Equation

To obtain a structure-preserving discretization in space, we discretize the space using the central difference approach as in the previous example, which yields the following system of ODEs:

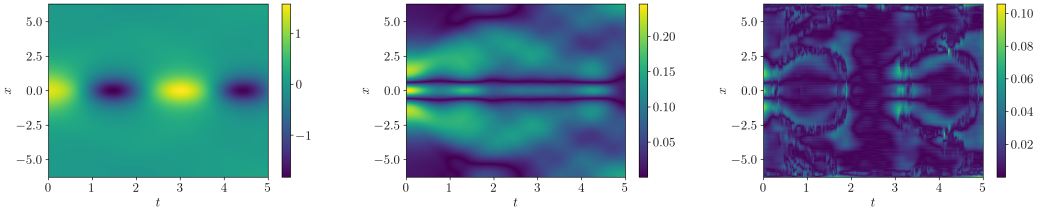
$$\boldsymbol{q}_t = \boldsymbol{D}_{xx}\boldsymbol{p} + \beta\left(\boldsymbol{p} \odot \boldsymbol{p} + \boldsymbol{q} \odot \boldsymbol{q}\right) \odot \boldsymbol{p},$$
$$\boldsymbol{p}_t = -\boldsymbol{D}_{xx}\boldsymbol{q} + \beta\left(\boldsymbol{p} \odot \boldsymbol{p} + \boldsymbol{q} \odot \boldsymbol{q}\right) \odot \boldsymbol{q},$$




(a) Ground truth (b) Absolute error PSD (c) Absolute error CNN

Figure: NLS equation: Absolute pointwise error for latent dimension $r = 1$ for the state $q$.

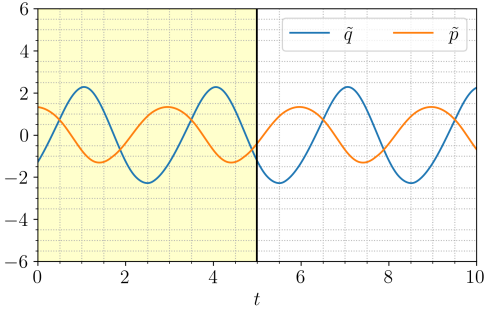
# Nonlinear Schrödinger equation



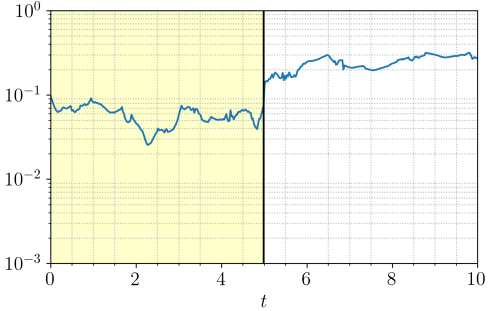

(a) Ground truth (b) Absolute error PSD (c) Absolute error CNN

Figure: NLS equation: Absolute pointwise error for latent dimension $r = 1$ for the state $p$.

# Nonlinear Schrödinger equation




(a) Latent trajectories




(b) Relative reconstruction error

Figure: A SympNet approach for learning the latent dynamics of the NLS equation obtained by the SympCAE encoder. The vertical black lines separate the training and testing intervals.

# Nonlinear Schrödinger Equation

Table: NLS equation: The table shows the performance of PSD and SympCAE in capturing the dynamics of the ground truth model in terms of relative reconstruction error at latent dimensions $r = 1, 2, 3$. The best result for each latent dimension is highlighted in bold.

| $r$ | $\varepsilon_{\text{PSD}}$ | $\varepsilon_{\text{SympCAE}}$ |
|---|---|---|
| 1 | $1.85 \cdot 10^{-1}$ | $\mathbf{4.44 \cdot 10^{-2}}$ |
| 2 | $1.04 \cdot 10^{-1}$ | $\mathbf{1.35 \cdot 10^{-2}}$ |
| 3 | $5.21 \cdot 10^{-2}$ | $\mathbf{1.82 \cdot 10^{-2}}$ |

# Sine-Gordon Equation

We consider the two-dimensional sine-Gordon equation

$$\begin{aligned} u_{tt}(x,y,t) &= u_{xx}(x,y,t) + u_{yy}(x,y,t) - \sin(u(x,y,t)), \\ u(x,y,0) &= u^0(x,y), \\ u_t(x,y,0) &= u_t^0(x,y), \quad x,y \in \Omega. \end{aligned}$$

# Sine-Gordon Equation

We consider the two-dimensional sine-Gordon equation

$$\begin{aligned} u_{tt}(x,y,t) &= u_{xx}(x,y,t) + u_{yy}(x,y,t) - \sin(u(x,y,t)), \\ u(x,y,0) &= u^0(x,y), \\ u_t(x,y,0) &= u_t^0(x,y), \quad x,y \in \Omega. \end{aligned}$$

We obtain the Hamiltonian form of the SG equation by introducing $q(x,y,t) = u(x,y,t)$ and $p(x,y,t) = u_t(x,y,t)$, which leads to the following conservative form:

$$\begin{aligned} q_t(x,y,t) &= p(x,y,t), \\ p_t(x,y,t) &= q_{xx}(x,y,t) + q_{yy}(x,y,t) - \sin(q(x,y,t)). \end{aligned}$$

# Sine-Gordon Equation

We consider the two-dimensional sine-Gordon equation

$$\begin{aligned} u_{tt}(x,y,t) &= u_{xx}(x,y,t) + u_{yy}(x,y,t) - \sin(u(x,y,t)), \\ u(x,y,0) &= u^0(x,y), \\ u_t(x,y,0) &= u_t^0(x,y), \quad x,y \in \Omega. \end{aligned}$$

We obtain the Hamiltonian form of the SG equation by introducing $q(x,y,t) = u(x,y,t)$ and $p(x,y,t) = u_t(x,y,t)$, which leads to the following conservative form:

$$\begin{aligned} q_t(x,y,t) &= p(x,y,t), \\ p_t(x,y,t) &= q_{xx}(x,y,t) + q_{yy}(x,y,t) - \sin(q(x,y,t)). \end{aligned}$$

Discretization in space leads to

$$\begin{aligned} \boldsymbol{q}_t &= \boldsymbol{p}, \\ \boldsymbol{p}_t &= \boldsymbol{D}_{xx}\boldsymbol{q} + \boldsymbol{D}_{yy}\boldsymbol{q} - \sin(\boldsymbol{q}). \end{aligned}$$

# Sine-Gordon Equation

The corresponding Hamiltonian is given by

$$H = \frac{1}{2}\left(\boldsymbol{p}^T\boldsymbol{p} - \boldsymbol{q}^T\boldsymbol{D}\boldsymbol{q}\right) + \sum_{i=1}^{N}(1 - \cos(\boldsymbol{q}_i)),$$

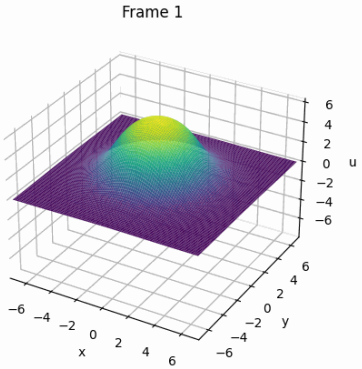
with $\boldsymbol{D} = \boldsymbol{D}_{xx} + \boldsymbol{D}_{yy}$.

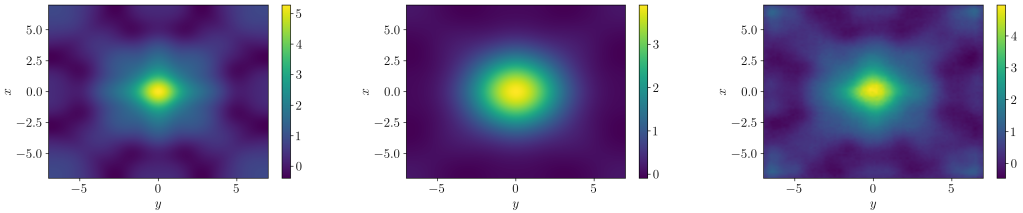
The corresponding Hamiltonian is given by

$$H = \frac{1}{2}\left(\boldsymbol{p}^T\boldsymbol{p} - \boldsymbol{q}^T\boldsymbol{D}\boldsymbol{q}\right) + \sum_{i=1}^{N}(1-\cos(\boldsymbol{q}_i)),$$

with $\boldsymbol{D} = \boldsymbol{D}_{xx} + \boldsymbol{D}_{yy}$.
We use the spatial domain
$\Omega = (-7,7)\times(-7,7)$ and the initial conditions

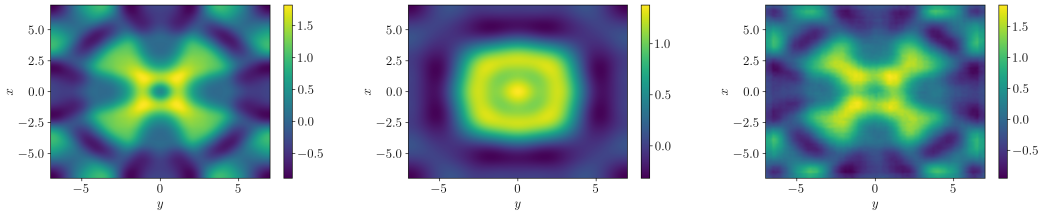
$$u^0(x,y) = 4\tan^{-1}\left(\exp\left(3-\sqrt{x^2+y^2}\right)\right),$$
$$u_t^0(x,y) = 0.$$






(a) Ground truth (b) PSD (c) SympCAE

Figure: SG equation: Comparison of the solutions of $q$ at final time $t = 20$ for latent time dimension $r = 3$.




(a) Ground truth (b) PSD (c) SympCAE

Figure: SG equation: Comparison of the solutions of $p$ at final time $t = 20$ for latent time dimension $r = 3$.

# Sine-Gordon Equation

Table: SG equation: The table shows the performance of PSD and SympCAE in capturing the dynamics of the ground truth model in terms of relative reconstruction error at latent dimensions $r = 1, 2, 3$. The best result for each latent dimension is highlighted in bold.

| $r$ | $\varepsilon_{\text{PSD}}$ | $\varepsilon_{\text{SympCAE}}$ |
|---|---|---|
| 1 | $3.74 \cdot 10^{-1}$ | $\mathbf{1.35 \cdot 10^{-1}}$ |
| 2 | $3.07 \cdot 10^{-1}$ | $\mathbf{5.15 \cdot 10^{-2}}$ |
| 3 | $2.55 \cdot 10^{-1}$ | $\mathbf{7.86 \cdot 10^{-2}}$ |

# Conclusions

# Conclusions

- We proposed a nonlinear symplectic convolutional autoencoder (SympCAE)

# Conclusions

- We proposed a nonlinear symplectic convolutional autoencoder (SympCAE) by using
  - Symplectic convolutional layers
  - Symplectic activation layers
  - PSD-like layers
  - Symplectic pooling layers

# Conclusions

- We proposed a nonlinear symplectic convolutional autoencoder (SympCAE) by using
  - Symplectic convolutional layers
  - Symplectic activation layers
  - PSD-like layers
  - Symplectic pooling layers
- We tested SympCAE on dimensionality reduction tasks for different Hamiltonian PDEs:
  - Linear wave equation
  - Nonlinear Schrödinger equation
  - 2D Sine-Gordon equation

# Conclusions

- We proposed a nonlinear symplectic convolutional autoencoder (SympCAE) by using
  - Symplectic convolutional layers
  - Symplectic activation layers
  - PSD-like layers
  - Symplectic pooling layers
- We tested SympCAE on dimensionality reduction tasks for different Hamiltonian PDEs:
  - Linear wave equation
  - Nonlinear Schrödinger equation
  - 2D Sine-Gordon equation

Future work:

- noise, parameters
- 3D extension
- symplectic transformers

# Conclusions

- We proposed a nonlinear symplectic convolutional autoencoder (SympCAE) by using
  - Symplectic convolutional layers
  - Symplectic activation layers
  - PSD-like layers
  - Symplectic pooling layers
- We tested SympCAE on dimensionality reduction tasks for different Hamiltonian PDEs:
  - Linear wave equation
  - Nonlinear Schrödinger equation
  - 2D Sine-Gordon equation

Future work:

- noise, parameters
- 3D extension
- symplectic transformers

The paper is available at arXiv:2508.19842

# References I


L. Peng & K. Mohseni (2016). Symplectic model reduction of Hamiltonian systems. *SIAM Journal on Scientific Computing* 38:A1–A27.

P. Jin, Z. Zhang, A. Zhu, Y.Tang, & G. E. Karniadakis (2020). SympNets: intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems. *Neural Networks* 132:166–179.

H. Sharma, H. Mu, P. Buchfink, R. Geelen, S. Glas, & B. Kramer (2023). Symplectic model reduction of Hamiltonian systems using data-driven quadratic manifolds. *Computer Methods in Applied Mechanics and Engineering* 417:116402

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

B. Brantner & M. Kraus (2023). Symplectic autoencoders for model reduction of Hamiltonian systems. arXiv:2312.10004.

S. Yildiz, P. Goyal, T. Bendokat, & P. Benner (2024). Data-Driven Identification of Quadratic Representations for Nonlinear Hamiltonian Systems Using Weakly Symplectic Liftings. *Journal of Machine Learning for Modeling and Computing* 5(2):45-71.

# References II


P. Goyal, S. Yildiz, & P. Benner (2025). Deep learning for structure-preserving universal stable Koopman-inspired embeddings for nonlinear canonical Hamiltonian dynamics. *Machine Learning: Science and Technology* 6(1):015063.

S. Yildiz, P. Goyal, & P. Benner (2025). Structure-preserving learning for multi-symplectic PDEs. *Advanced Modeling and Simulation in Engineering Sciences* 12(1):6.

K. Janik & P. Benner (2025). Time-adaptive SympNets for separable Hamiltonian systems. arXiv:2509.16026