# Advanced ROMs in SciML

## Time-dependent problems and Neural Operators

Federico Pichi

02 July 2025


# Dynamic Mode Decomposition

**Remarks**:

- Data-driven surrogate models are constructed using *interpolation* and *regression* techniques to directly learn the map from inputs to outputs without constructing a reduced model and its operators.
- The notion of state is unimportant, since interpolation/regression already provide accurate results.

**Dynamic mode decomposition** (DMD) is a method to analyze the behavior of dynamical systems

$$\mathbf{x}_{n+1} = \mathbf{F}(\mathbf{x}_n), \qquad n = 0, 1, 2, \dots,$$

where $\mathbf{x} \in \Omega \subseteq \mathbb{R}^d$ denotes the state of the system, and $\mathbf{F} : \Omega \to \Omega$ governs the system's evolution.

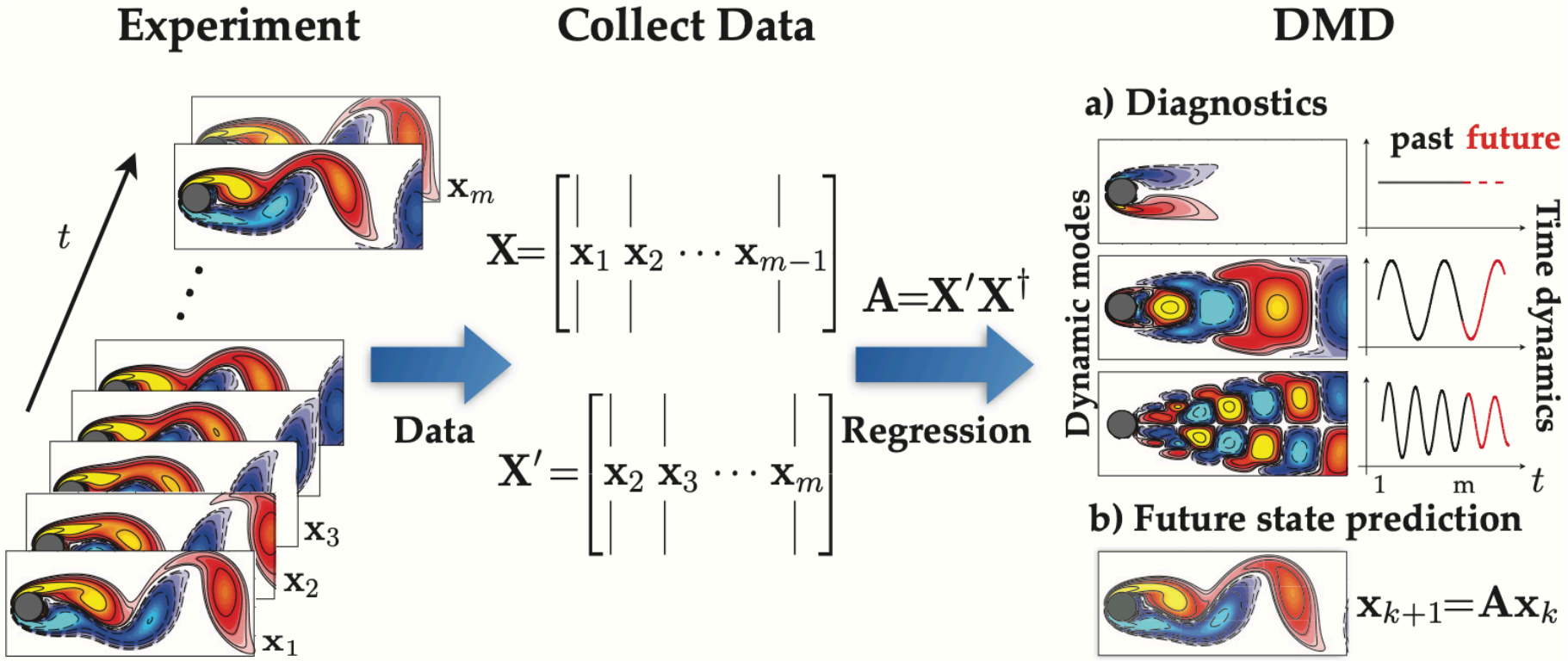
**Issue**: the knowledge is limited to discrete-time snapshots of the system, i.e., a finite dataset

$$\left\{\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\right\}_{m=1}^{M} \quad \text{such that} \quad \mathbf{y}^{(m)} = \mathbf{F}(\mathbf{x}^{(m)}), \quad m = 1, \dots, M$$

that can be written in the form of snapshot matrices

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(M)} \end{pmatrix} \in \mathbb{R}^{d \times M}, \quad \mathbf{Y} = \begin{pmatrix} \mathbf{y}^{(1)} & \mathbf{y}^{(2)} & \cdots & \mathbf{y}^{(M)} \end{pmatrix} \in \mathbb{R}^{d \times M}.$$

# Dynamic Mode Decomposition





[1] Kutz, J.N., Brunton, S.L., Brunton, B.W., Proctor, J.L., 2016. Dynamic Mode Decomposition, Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611974508

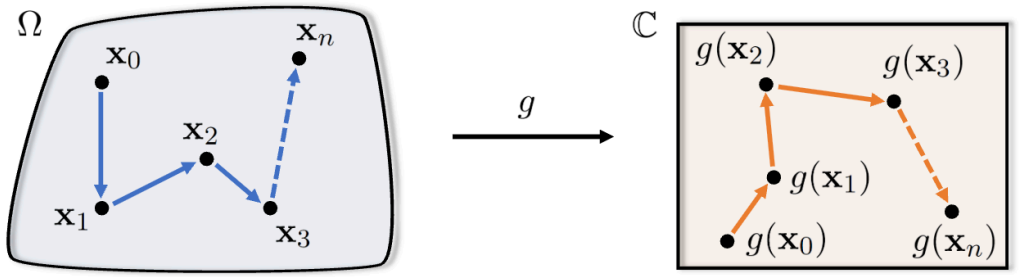
[2] Colbrook, M.J., 2023. The Multiverse of Dynamic Mode Decomposition Algorithms. https://doi.org/10.48550/arXiv.2312.00137

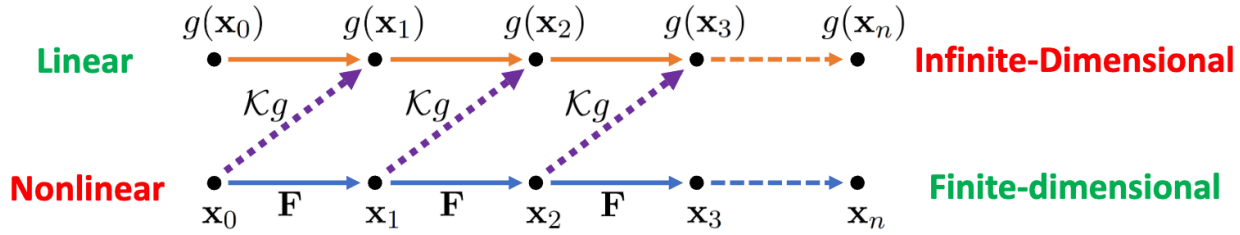
# DMD and Koopman Operators

**Koopman Operators** are closely connected to DMD, since they offer a different perspective of dynamical systems by addressing *nonlinear* problems in a *lifted* space.

**How**: lift a nonlinear system into an infinite-dimensional space of observable functions $g : \Omega \to \mathbb{C}$ using a Koopman operator $\mathcal{K}$ as: $[\mathcal{K}g](\mathbf{x}) = g(\mathbf{F}(\mathbf{x})), \quad \text{so that} \quad [\mathcal{K}g](\mathbf{x}_n) = g(\mathbf{x}_{n+1}).$

**Link**: Koopman operator theory identifies a coordinate transformation (related to the spectrum) under which even strongly nonlinear dynamics may be approximated by a linear system.



# DMD and Koopman Operators

Given the snapshot matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{C}^{d \times M}$, we seek a matrix $\mathbf{K}_{\mathrm{DMD}}$ such that $\mathbf{Y} \approx \mathbf{K}_{\mathrm{DMD}}\mathbf{X}$.

$\rightsquigarrow$ To find a suitable matrix $\mathbf{K}_{\mathrm{DMD}}$, we consider the minimization problem

$$\min_{\mathbf{K}_{\mathrm{DMD}} \in \mathbb{C}^{d \times d}} \|\mathbf{Y} - \mathbf{K}_{\mathrm{DMD}}\mathbf{X}\|_{\mathrm{F}},$$

for which, denoting by $\dagger$ the Moore-Penrose pseudoinverse, a solution is given by

$$\mathbf{K}_{\mathrm{DMD}} = \mathbf{Y}\mathbf{X}^{\dagger} \in \mathbb{C}^{d \times d}.$$

The matrices $\mathbf{X}$ and $\mathbf{Y}$ are tall and skinny ($d \gg M$), so one first projects (SVD) onto a low-dimensional subspace to reconstruct the leading eigenmodes of the matrix $\mathbf{K}_{\mathrm{DMD}}$ without explicitly computing it.

**Input:** Snapshot data $\mathbf{X} \in \mathbb{C}^{d \times M}$ and $\mathbf{Y} \in \mathbb{C}^{d \times M}$, rank $r \in \mathbb{N}$.

1: Compute a truncated SVD of the data matrix $\mathbf{X} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$, $\mathbf{U} \in \mathbb{C}^{d \times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$,$\mathbf{V} \in \mathbb{C}^{M \times r}$.
The columns of $\mathbf{U}$ and $\mathbf{V}$ are orthonormal and $\mathbf{\Sigma}$ is diagonal.

2: Compute the compression $\tilde{\mathbf{K}}_{\mathrm{DMD}} = \mathbf{U}^*\mathbf{Y}\mathbf{V}\mathbf{\Sigma}^{-1} \in \mathbb{C}^{r \times r}$.

3: Compute the eigendecomposition $\tilde{\mathbf{K}}_{\mathrm{DMD}}\mathbf{W} = \mathbf{W}\mathbf{\Lambda}$.
The columns of $\mathbf{W}$ are eigenvectors and $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues.

4: Compute the modes $\mathbf{\Phi} = \mathbf{Y}\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{W}$.

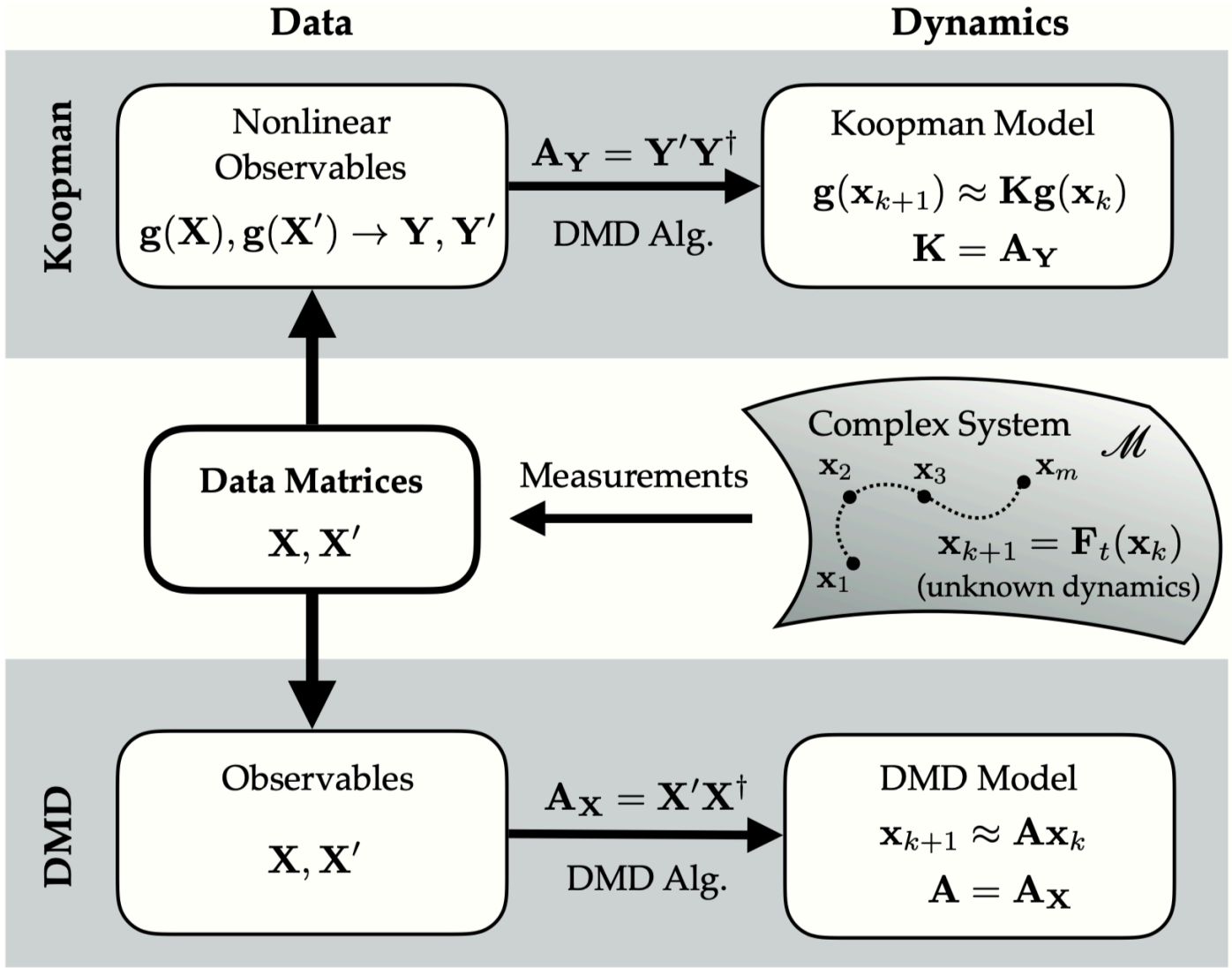
**Output:** The eigenvalues $\mathbf{\Lambda}$ and modes $\mathbf{\Phi} \in \mathbb{C}^{d \times r}$.

## DMD and Koopman Operators

**Remarks**:

- The rank $r$ is usually chosen based on the decay of singular values of $\mathbf{X}$.
- In the context of Koopman operators, $r$ is equivalent to the size of the space spanned by basis functions, and a good choice depends on the chosen observables.
- The algorithm construct a linear model $\tilde{\mathbf{x}}_{n+1} \approx \tilde{\mathbf{K}}_{\mathrm{DMD}} \tilde{\mathbf{x}}_n$ of the dynamical system on projected coordinates $\tilde{\mathbf{x}} = \mathbf{U}^* \mathbf{x}$.
- *Centering* the data before applying DMD can be helpful if the mean-subtracted data have linearly dependent columns, especially if the dynamics are *perturbations* about an *equilibrium*.
- DMD can be interpreted as an approximation to *Koopman* spectral analysis, providing a solid theoretical foundation for applying DMD in analyzing nonlinear dynamics.

# DMD and Koopman Operators



# Operator Inference (OpInf)

**Goal**: *nonintrusive* projection-based model reduction approach for *time-dependent PDEs*.

**Issue**: for intrusive projection the full-model operators are required, but in many situations the full model is given as a *black-box* that computes trajectories and outputs for given initial conditions and inputs.

**Strategy**: *OpInf* infers approximations of the reduced operators from the initial conditions, inputs, trajectories of the states, and outputs without requiring the full-model operators.

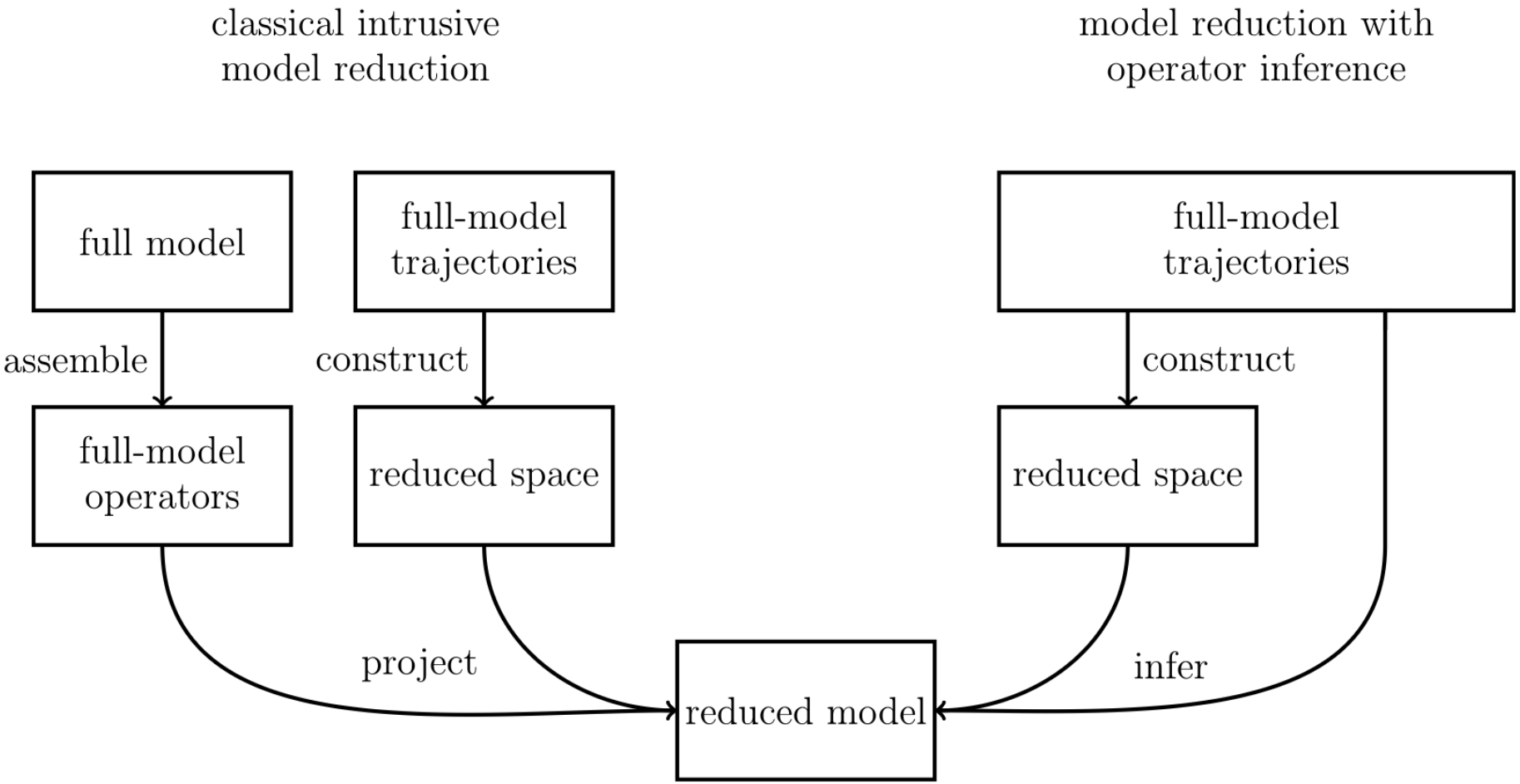
**Remark**: applicable to *linear* PDEs or low-order polynomial *nonlinear* terms.

**How**: The inferred operators are the solution of a *least-squares* problem.

**Convergence**: The operators *converge*, with sufficient state trajectory data, in the Frobenius norm, to the reduced operators that would be obtained via an intrusive projection of the full-model operators.

[1] Peherstorfer, B., & Willcox, K. (2016). Data-driven operator inference for nonintrusive projection-based model reduction. Computer Methods in Applied Mechanics and Engineering. https://doi.org/10.1016/j.cma.2016.03.025

# Operator Inference (OpInf)



## OpInf and DMD

- DMD derives an approximation of the HF operator from HF trajectories, and it corresponds to the approximation that *best* fits the trajectories in the $L^2$ *norm*.
- For *linear* full model and inputs that are *constant over time*, the inferred operator is the same operator as the one obtained via DMD.
- DMD derives a *linear* approximation of the full model, whereas OpInf can handle full models that have *polynomial nonlinear terms*.
- DMD is typically applied to full models with *parameter-independent* operators only, while here one can consider parametric extentions or compute multiple inferred operators and then *interpolate* between them.

## OpInf

Let $K \in \mathbb{N}$ and consider a dynamical system of the form

$$\hat{\boldsymbol{x}}_{k+1}(\boldsymbol{\mu}) = \boldsymbol{f}(\hat{\boldsymbol{x}}_k(\boldsymbol{\mu}), \boldsymbol{u}_k(\boldsymbol{\mu}); \boldsymbol{\mu}) \,, \qquad k = 0, \dots, K-1$$

with state $\hat{\boldsymbol{x}}_k(\boldsymbol{\mu}) \in \mathbb{R}^N$ of dimension $N \in \mathbb{N}$ and input $\boldsymbol{u}_k(\boldsymbol{\mu}) \in \mathbb{R}^p$ of dimension $p \in \mathbb{N}$ at time steps $k = 0, \dots, K-1$, where the parameter $\boldsymbol{\mu} \in \mathcal{D} \subset \mathbb{R}^d$ is independent of the time step, the initial condition is $\hat{\boldsymbol{x}}_0 \in \mathbb{R}^N$, and the function $\boldsymbol{f} : \mathbb{R}^N \times \mathbb{R}^p \times \mathcal{D} \to \mathbb{R}^N$ describes the dynamics of system.

For systems with polynomial nonlinearities of order $\ell \in \mathbb{N}$, i.e., denoting $N_i = \binom{N+i-1}{i}$ for $i \in \mathbb{N}$, there exists $\boldsymbol{A}_i(\boldsymbol{\mu}) \in \mathbb{R}^{N \times N_i}$ for $i = 1, \dots, \ell$, and $\boldsymbol{B}(\boldsymbol{\mu}) \in \mathbb{R}^{N \times p}$ for $\boldsymbol{\mu} \in \mathcal{D}$ such that

$$\boldsymbol{f}(\hat{\boldsymbol{x}}_k(\boldsymbol{\mu}), \boldsymbol{u}_k(\boldsymbol{\mu}); \boldsymbol{\mu}) = \sum_{i=1}^{\ell} \boldsymbol{A}_i(\boldsymbol{\mu}) \hat{\boldsymbol{x}}_k^i(\boldsymbol{\mu}) + \boldsymbol{B}(\boldsymbol{\mu}) \boldsymbol{u}_k(\boldsymbol{\mu}) \,, \qquad k = 0, \dots, K-1 \,.$$

The vector $\hat{\boldsymbol{x}}_k^i(\boldsymbol{\mu}) \in \mathbb{R}^{N_i}$ is the $i$-th power of $\hat{\boldsymbol{x}}_k$, which is constructed from the Kronecker product $\hat{\boldsymbol{x}}_k(\boldsymbol{\mu}) \otimes \cdots \otimes \hat{\boldsymbol{x}}_k(\boldsymbol{\mu})$ by removing all duplicate entries due to commutativity of the multiplication.

## Standard ROMs

- Define $\boldsymbol{X}(\boldsymbol{\mu}) = [\hat{\boldsymbol{x}}_0(\boldsymbol{\mu}), \ldots, \hat{\boldsymbol{x}}_{K-1}(\boldsymbol{\mu})] \in \mathbb{R}^{N \times K}$ and $\boldsymbol{Y}(\boldsymbol{\mu}) = [\hat{\boldsymbol{x}}_1(\boldsymbol{\mu}), \ldots, \hat{\boldsymbol{x}}_K(\boldsymbol{\mu})] \in \mathbb{R}^{N \times K}$, which differ in their start and end index, and $\boldsymbol{X}^i(\boldsymbol{\mu}) = [\hat{\boldsymbol{x}}_0^i(\boldsymbol{\mu}), \ldots, \hat{\boldsymbol{x}}_{K-1}^i(\boldsymbol{\mu})] \in \mathbb{R}^{N \times K}$, for $i = 1, \ldots, \ell$, be the $i$-th powers of the states at times $k = 0, \ldots, K-1$.
- If operators $\boldsymbol{A}_1(\boldsymbol{\mu}), \ldots, \boldsymbol{A}_\ell(\boldsymbol{\mu}), \boldsymbol{B}(\boldsymbol{\mu})$ for $\boldsymbol{\mu} \in \mathcal{D}$ are available, then traditional projection-based model reduction can be applied to find a reduced model. Let $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_m \in \mathcal{D}$ and let $\boldsymbol{X}(\boldsymbol{\mu}_1), \ldots, \boldsymbol{X}(\boldsymbol{\mu}_m) \in \mathbb{R}^{N \times K}$ be the corresponding trajectories of length $K$, applying proper orthogonal decomposition (POD) to the snapshot matrix $[\boldsymbol{X}(\boldsymbol{\mu}_1), \ldots, \boldsymbol{X}(\boldsymbol{\mu}_m)] \in \mathbb{R}^{N \times mK}$ yields an orthonormal basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r$, with $r \ll N$, that spans an $r$-dimensional subspace $\mathcal{V}_r \subset \mathbb{R}^N$ via the matrix $\boldsymbol{V}_r = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r] \in \mathbb{R}^{N \times r}$.
- For $j = 1, \ldots, m$, the reduced operators are constructed via projection $\tilde{\boldsymbol{A}}_1(\boldsymbol{\mu}_j) = \boldsymbol{V}_r^T \boldsymbol{A}_1(\boldsymbol{\mu}_j) \boldsymbol{V}_r$, and $\tilde{\boldsymbol{B}}(\boldsymbol{\mu}_j) = \boldsymbol{V}_r^T \boldsymbol{B}(\boldsymbol{\mu}_j)$, and similarly for $\tilde{\boldsymbol{A}}_2(\boldsymbol{\mu}_j) \in \mathbb{R}^{r \times r_2}, \ldots, \tilde{\boldsymbol{A}}_\ell(\boldsymbol{\mu}_j) \in \mathbb{R}^{r \times r_\ell}$.

## Standard ROMs

- Denoting the reduced state by $\tilde{\boldsymbol{x}}_k(\boldsymbol{\mu}_j) \in \mathbb{R}^r$, its $i$-th power as $\tilde{\boldsymbol{x}}_k^i(\boldsymbol{\mu}_j) \in \mathbb{R}^{r_i}$ for $i \in \mathbb{N}$, and the initial condition as $\tilde{\boldsymbol{x}}_0(\boldsymbol{\mu}_j) = \boldsymbol{V}_r^T \boldsymbol{x}_0(\boldsymbol{\mu}_j)$, the reduced model for $\boldsymbol{\mu}_j$ is given by

$$\begin{aligned}\tilde{\boldsymbol{x}}_{k+1}(\boldsymbol{\mu}_j) &= \tilde{\boldsymbol{f}}(\tilde{\boldsymbol{x}}_k(\boldsymbol{\mu}_j), \boldsymbol{u}_k(\boldsymbol{\mu}_j); \boldsymbol{\mu}_j) \\ &= \sum_{i=1}^{\ell} \tilde{\boldsymbol{A}}_i(\boldsymbol{\mu}_j)\tilde{\boldsymbol{x}}_k^i(\boldsymbol{\mu}_j) + \tilde{\boldsymbol{B}}(\boldsymbol{\mu}_j)\boldsymbol{u}_k(\boldsymbol{\mu}_j)\,, \qquad k = 0, \ldots, K-1.\end{aligned}$$

- Once the reduced models $\tilde{\boldsymbol{f}}(\cdot, \cdot; \boldsymbol{\mu}_1), \ldots, \tilde{\boldsymbol{f}}(\cdot, \cdot; \boldsymbol{\mu}_m)$ are constructed for all $m$ parameters $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_m$, a reduced model for $\boldsymbol{\mu} \in \mathcal{D}$ is derived by element-wise interpolation of the reduced operators corresponding to $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_m$.

## Operator Inference

1. State trajectories $\boldsymbol{X}(\boldsymbol{\mu}_1), \ldots, \boldsymbol{X}(\boldsymbol{\mu}_m)$ and $\boldsymbol{Y}(\boldsymbol{\mu}_1), \ldots, \boldsymbol{Y}(\boldsymbol{\mu}_m)$ are obtained by querying the system at parameters $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_m \in \mathcal{D}$ to derive a reduced space spanned by the columns of $\boldsymbol{V}_r = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r]$.
2. Project the trajectories onto the reduced space $\mathcal{V}_r$ spanned by the columns of $\boldsymbol{V}_r$

$$\breve{\boldsymbol{X}}(\boldsymbol{\mu}_j) = \boldsymbol{V}_r^T \boldsymbol{X}(\boldsymbol{\mu}_j)\,, \qquad \breve{\boldsymbol{Y}}(\boldsymbol{\mu}_j) = \boldsymbol{V}_r^T \boldsymbol{Y}(\boldsymbol{\mu}_j)\,, \qquad j = 1, \ldots, m\,.$$

3. The operators

$$\hat{\boldsymbol{A}}_1(\boldsymbol{\mu}_j) \in \mathbb{R}^{r \times r_1}, \ldots, \hat{\boldsymbol{A}}_\ell(\boldsymbol{\mu}_j) \in \mathbb{R}^{r \times r_\ell}, \hat{\boldsymbol{B}}(\boldsymbol{\mu}_j) \in \mathbb{R}^{r \times p}$$

are learned via least-squares regression

$$\min_{\hat{\boldsymbol{A}}_1(\boldsymbol{\mu}_j), \ldots, \hat{\boldsymbol{A}}_\ell(\boldsymbol{\mu}_j), \hat{\boldsymbol{B}}(\boldsymbol{\mu}_j)} \sum_{k=0}^{K-1} \left\| \sum_{i=1}^{\ell} \hat{\boldsymbol{A}}_i(\boldsymbol{\mu}_j) \breve{\boldsymbol{x}}_k^i(\boldsymbol{\mu}_j) + \hat{\boldsymbol{B}}(\boldsymbol{\mu}_j) \boldsymbol{u}_k(\boldsymbol{\mu}_j) - \breve{\boldsymbol{x}}_{k+1}(\boldsymbol{\mu}_j) \right\|_2^2 .$$

## Operator Inference

4. Obtain the model for $j = 1, \ldots, m$:

$$\begin{aligned}\hat{\boldsymbol{x}}_{k+1}(\boldsymbol{\mu}_j) &= \sum_{i=1}^{\ell} \hat{\boldsymbol{A}}_i(\boldsymbol{\mu}_j)\hat{\boldsymbol{x}}_k^i(\boldsymbol{\mu}_j) + \hat{\boldsymbol{B}}(\boldsymbol{\mu}_j)\boldsymbol{u}_k(\boldsymbol{\mu}_j) \\ &= \hat{\boldsymbol{f}}(\hat{\boldsymbol{x}}_k(\boldsymbol{\mu}_j), \boldsymbol{u}_k(\boldsymbol{\mu}_j); \boldsymbol{\mu}_j).\end{aligned}$$

**Remark**:

- The least-squares problem is solved for each parameter $\boldsymbol{\mu}_j$ with $j = 1, \ldots, m$.
- The state of the learned model at time $k$ is $\hat{\boldsymbol{x}}_k(\boldsymbol{\mu}_j) \in \mathbb{R}^r$ and is obtained by time stepping.
- The projected state $\breve{\boldsymbol{x}}_k(\boldsymbol{\mu}_j)$ is obtained by projecting the high-dimensional state $\boldsymbol{x}_k(\boldsymbol{\mu}_j)$ at time $k$.
- The initial condition is $\hat{\boldsymbol{x}}_0(\boldsymbol{\mu}_j) = \boldsymbol{V}_r^T \boldsymbol{x}_0(\boldsymbol{\mu}_j)$, and to obtain a model for $\boldsymbol{\mu} \in \mathcal{D}$, the operators of the learned models corresponding to $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_m$ are interpolated as in traditional model reduction.

## Operator Inference

It is convenient to reformulate the problem for each $j = 1, \ldots, m$ as

$$\min_{\hat{\boldsymbol{O}}(\boldsymbol{\mu}_j)} \left\| \breve{\boldsymbol{D}}^T(\boldsymbol{\mu}_j)\hat{\boldsymbol{O}}^T(\boldsymbol{\mu}_j) - \breve{\boldsymbol{Y}}^T(\boldsymbol{\mu}_j) \right\|_F^2 \qquad \text{and} \qquad \breve{\boldsymbol{D}}(\boldsymbol{\mu}_j) = \begin{bmatrix} \breve{\boldsymbol{X}}(\boldsymbol{\mu}_j) \\ \breve{\boldsymbol{X}}^2(\boldsymbol{\mu}_j) \\ \vdots \\ \breve{\boldsymbol{X}}^\ell(\boldsymbol{\mu}_j) \\ \boldsymbol{U}(\boldsymbol{\mu}_j) \end{bmatrix} \in \mathbb{R}^{\sum_{i=1}^{\ell} n_i + p \times K}$$

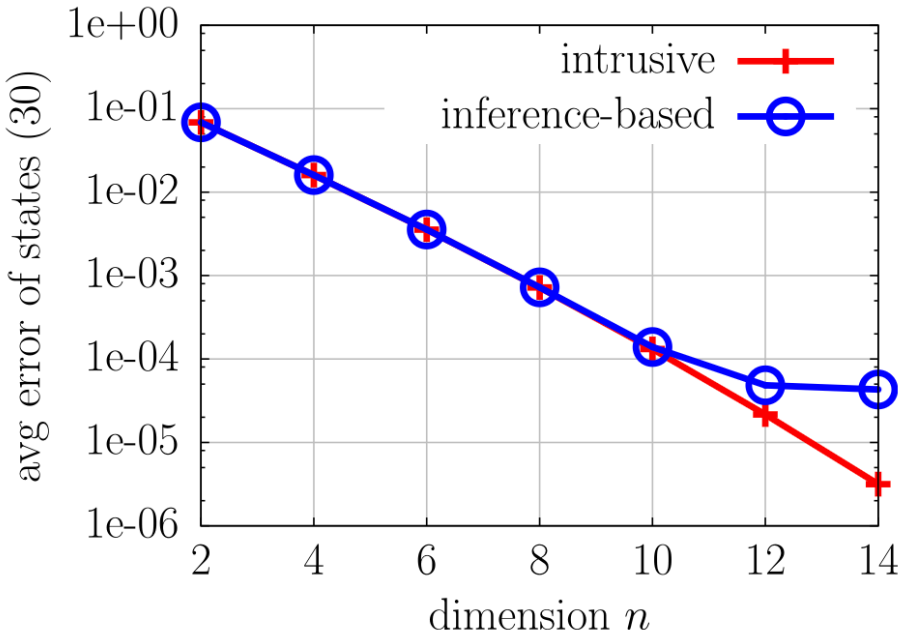
where the operator matrix is defined as $\hat{\boldsymbol{O}}(\boldsymbol{\mu}_j) = \begin{bmatrix} \hat{\boldsymbol{A}}_1(\boldsymbol{\mu}_j) & \ldots & \hat{\boldsymbol{A}}_\ell(\boldsymbol{\mu}_j) & \hat{\boldsymbol{B}}(\boldsymbol{\mu}_j) \end{bmatrix} \in \mathbb{R}^{r \times \sum_{i=1}^{\ell} n_i + p}$, $\breve{\boldsymbol{X}}^i(\boldsymbol{\mu}_j) = [\breve{\boldsymbol{x}}_0^i(\boldsymbol{\mu}_j), \ldots, \breve{\boldsymbol{x}}_{K-1}^i(\boldsymbol{\mu}_j)] \in \mathbb{R}^{r_i \times K}$, and $\boldsymbol{U}(\boldsymbol{\mu}_j) = [\boldsymbol{u}_0(\boldsymbol{\mu}_j), \ldots, \boldsymbol{u}_{K-1}(\boldsymbol{\mu}_j)] \in \mathbb{R}^{p \times K}$.

# Operator Inference (Burgers equation)

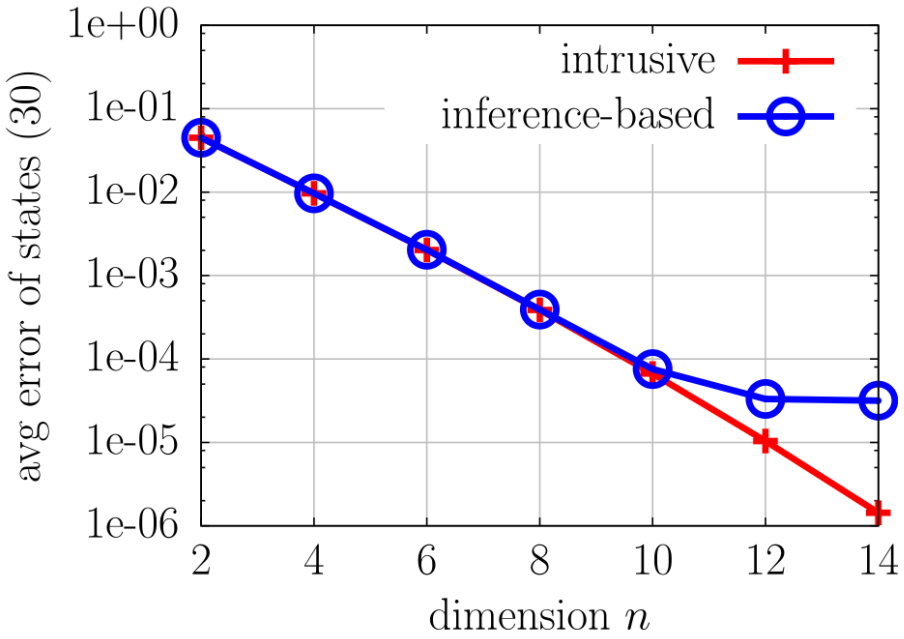
Let's consider the Burgers equation with nonlinear term of second order:

$$\partial_t \boldsymbol{u}(x,t;\mu) + \boldsymbol{u}(x,t;\mu)\partial_x \boldsymbol{u}(x,t;\mu) - \mu \partial_{xx}^2 \boldsymbol{u}(x,t;\mu) = 0,$$

where $x \in \Omega = [0,1]$ is the spatial coordinate, $t \in [0,T]$ the time, and $\mu \in \mathcal{D} = [0.1,1]$ the parameter.




(a) Training parameters $\mu_1, \ldots, \mu_M$.




(b) Test parameters $\mu_{M+1}, \ldots, \mu_{M+M_{\text{test}}}$.

## Sparse identification of nonlinear dynamical systems (SINDy)

**Goal**: Discover governing physical equations from measurement data for nonlinear dynamical systems.

**Assumption**: Only a few important terms govern the dynamics, i.e. the equations are sparse in the space of possible functions.

**Outcome**: The resulting models are parsimonious, balancing model complexity with descriptive ability avoiding overfitting, capable of generalizing to parameterized, time-varying, or externally forced systems.

**How**: Sparse regression combined with machine learning.
In particular, we use sparse regression to determine the fewest terms in the dynamic governing equations required to accurately represent the data.

## SINDy

Let us start from the dynamical system:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t)) \\ \boldsymbol{x}(t_0) = \boldsymbol{x}_0, \end{cases}$$

where $\boldsymbol{f} : I \times \Omega \to \mathbb{R}^N$, with $I \subseteq \mathbb{R}^+$ some interval, $(t_0, \boldsymbol{x}_0) \in I \times \Omega$ and $\boldsymbol{x} : I \to \Omega \subseteq \mathbb{R}^N$.
The vector $\boldsymbol{x}(t) = [x_1(t),\, x_2(t),\, \cdots,\, x_N(t)]^T \in \mathbb{R}^N$ represents the state of the system at time $t$, with $N$ being the spatial dimension of the system, whereas the function $\boldsymbol{f}(t, \boldsymbol{x})$ represents the dynamics.

**Aim**: We have a set of snapshots describing the system, and we are interested in learning $\boldsymbol{f}$ from the data.

**Assumption**: To find the function $\boldsymbol{f}$ describing the dynamics, we assume that it can be expressed as a linear combination of terms coming from a library of candidate functions with a few active terms (sparse).

# SINDy

1. Express the snapshot matrix (being $N_t$ the number of snapshots) as:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}(t_1)^T \\ \vdots \\ \boldsymbol{x}(t_{N_t})^T \end{bmatrix} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_N(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_{N_t}) & x_2(t_{N_t}) & \cdots & x_N(t_{N_t}) \end{bmatrix} \in \mathbb{R}^{N_t \times N}.$$

2. We denote by $\dot{\boldsymbol{X}}$ the matrix containing the corresponding time derivatives (approximated if unknown), and we introduce $\boldsymbol{\Theta}(\boldsymbol{x}^T) \in \mathbb{R}^r$ as the vector of symbolic functions to construct the library.

$\rightsquigarrow$ Compute $\boldsymbol{f}$ by solving for $\boldsymbol{\Xi}$ the equation

$$\dot{\boldsymbol{X}} = \boldsymbol{\Theta}(\boldsymbol{X})\boldsymbol{\Xi},$$

where $\boldsymbol{\Theta}(\boldsymbol{X}) \in \mathbb{R}^{N_t \times r}$ is the dictionary matrix containing the values at different time instances of the $r$ candidate functions of the columns of $\boldsymbol{X}$, and $\boldsymbol{\Xi} = \{\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_N\} \in \mathbb{R}^{r \times N}$ is such that its $k-$th column describes the active terms in the equation $\dot{x}_k(t) = f_k(t, \boldsymbol{x}(t))$, i.e.

$$\dot{x}_k(t) = f_k(t, \boldsymbol{x}(t)) = \boldsymbol{\Theta}(\boldsymbol{x}^T(t))\boldsymbol{\xi}_k, \quad k = 1, \dots, N.$$

# SINDy

**Example**:
Consider the dictionary/data matrix $\boldsymbol{\Theta}(\boldsymbol{X})$ given by constant function, polynomials up to a given degree and trigonometric functions as:

$$\boldsymbol{\Theta}(\boldsymbol{X}) = \begin{bmatrix} | & | & | & & | & | & | & | & \\ \mathbf{1} & \boldsymbol{X} & \boldsymbol{X}^2 & \cdots & \sin(\boldsymbol{X}) & \cos(\boldsymbol{X}) & \sin(2\boldsymbol{X}) & \cos(2\boldsymbol{X}) & \cdots \\ | & | & | & & | & | & | & | & \end{bmatrix},$$

where higher polynomials are denoted as $\boldsymbol{X}^2$, $\boldsymbol{X}^3$ and so on. For instance, by $\boldsymbol{X}^2$ we denote the quadratic functions of the state variable $\boldsymbol{x}$:

$$\boldsymbol{X}^2 = \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & \cdots & x_2^2(t_1) & x_2(t_1)x_3(t_1) & \cdots \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & \cdots & x_2^2(t_2) & x_2(t_2)x_3(t_2) & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\ x_1^2(t_{N_t}) & x_1(t_{N_t})x_2(t_{N_t}) & \cdots & x_2^2(t_{N_t}) & x_2(t_{N_t})x_3(t_{N_t}) & \cdots \end{bmatrix}.$$

# SINDy

**Remarks**:

- Continuous candidate functions to be able to evaluate them pointwise.
- The choice of the candidate functions is crucial and heavily problem-dependent, but can be guided by prior knowledge of the underlying system.
- The problem results in an overdetermined system of equations.
- To solve it for $\boldsymbol{\Xi}$, each column requires a distinct optimization problem to find the vector of coefficients $\xi_k$ for the $k-$th row equation.
- For sparsity, one should use a sparse regression algorithm for identifying $\xi_k$, with $k \in \{1, \dots, N\}$.
- In the snapshots matrix $\boldsymbol{X}$ the times $t_1, \dots, t_{N_t}$ do not need to be ordered.
- It is possible to collect snapshots coming from the evolution of the system starting from different initial conditions (parametric extension)

## SINDy

Assume that $\boldsymbol{\Theta}(\boldsymbol{X}) \in \mathbb{R}^{m \times r}$, with $r$ the cardinality of the library and $m \gg r$, meaning that the number of snapshots of the system is much bigger that the number of candidate nonlinear functions.

**Issue**: Due to the lack of analytical solutions and/or of uncertainty in the observations, $\boldsymbol{X}$ and $\dot{\boldsymbol{X}}$ might be affected by some noise, so that the equation does not hold exactly.

**Strategy**: *LASSO* approach, a statistical learning method (computationally expensive for large dataset) solving the minimization problem with $L^1$ regularization term to the regression (promote sparsity):

$$\arg \min_{\boldsymbol{\Xi} \in \mathbb{R}^{r \times N}} \|\dot{\boldsymbol{X}} - \boldsymbol{\Theta}(\boldsymbol{X})\boldsymbol{\Xi}\|_2 + \lambda \|\boldsymbol{\Xi}\|_1 .$$

**Remark**: Choosing the value of $\lambda \in \mathbb{R}^+$ is a trade-off between the model's complexity and accuracy.

## SINDy (Lotka-Volterra)

Let us consider the Lotka-Volterra system of equations, which reads as follows:

$$\begin{cases} x'(t) = \alpha x(t) - \beta x(t) y(t) \\ y'(t) = \delta x(t) y(t) - \gamma y(t) \end{cases}$$

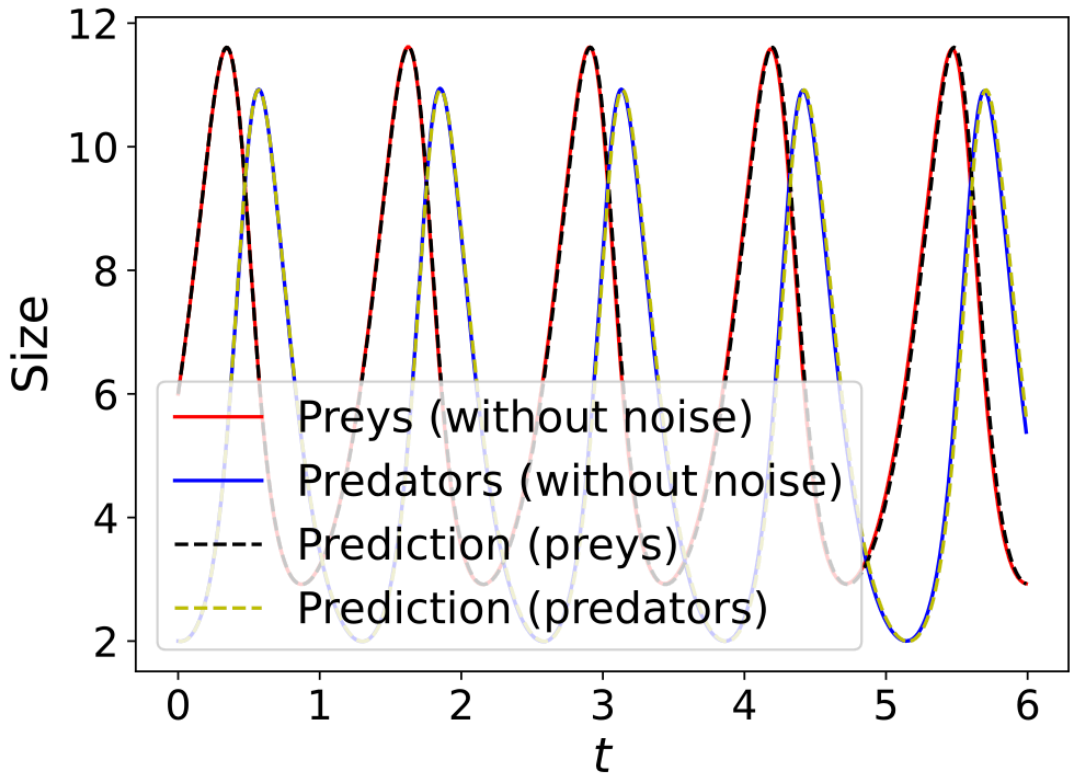
with $\alpha = 4.2, \beta = 0.8, \gamma = 6.3, \delta = 1 \in \mathbb{R}^+$, initial condition $[x_0, y_0] = [6, 2]^T$, $x(t)$ and $y(t)$ represent, respectively, the size of the population of preys and predators at time $t \in [0, 6]$.

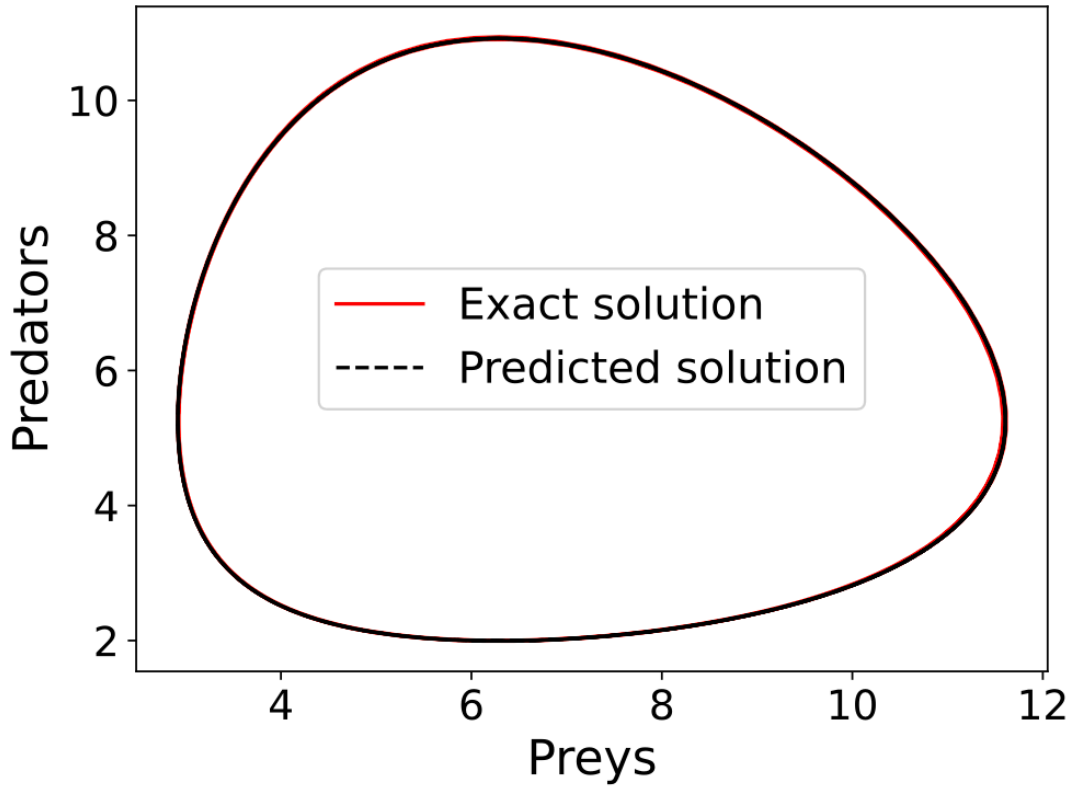
Training a model exploiting a library of polynomials of degree at most $2$, one obtains the equations:

$$\begin{cases} x'(t) = \tilde{\alpha} x(t) + -\tilde{\beta} x(t) y(t) \\ y'(t) = -\tilde{\gamma} y(t) + \tilde{\delta} x(t) y(t), \end{cases}$$

where $\tilde{\alpha} = 4.197, \tilde{\beta} = 0.8, \tilde{\gamma} = 6.291$, and $\tilde{\delta} = 0.999$, hence, the terms appearing in the polynomial library coincide exactly with the ones in the actual equation, and the coefficients almost coincide.

# SINDy (Lotka-Volterra)




**Remark**: Choosing a different stepsize $dt$ or constant $\lambda$ new terms can appear and the structure is lost.

## SINDy (Lotka-Volterra + noise)

We perturb the snapshot matrix $\boldsymbol{X}$ with some noise (training and derivatives)

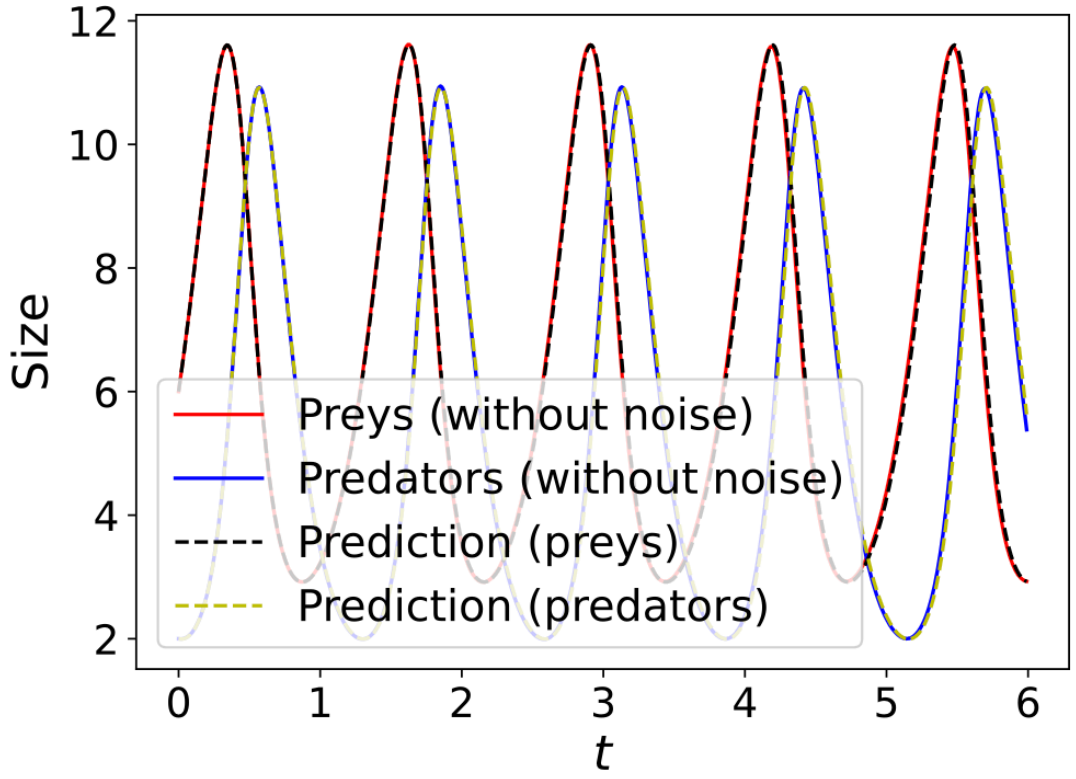
$$\tilde{\boldsymbol{X}} = \boldsymbol{X} + \eta \boldsymbol{Z},$$

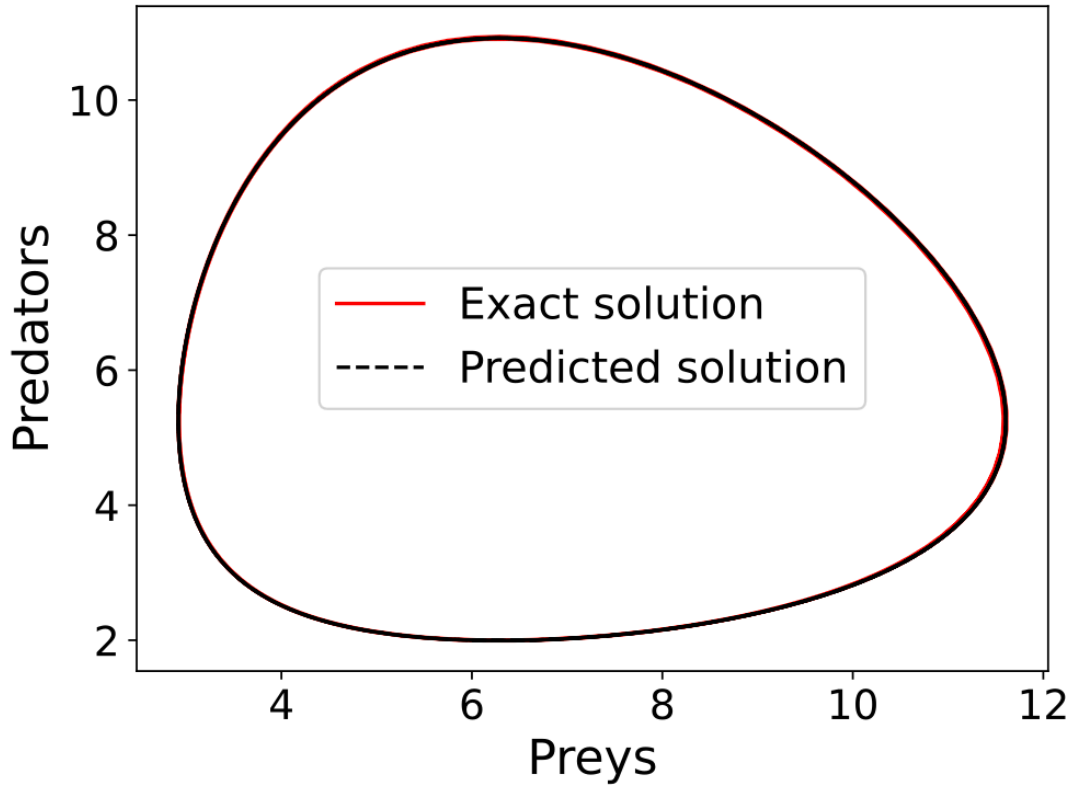
with $\boldsymbol{Z}$ a matrix of i.i.d. Gaussian entries with zero mean and unit variance, where $\eta$ is the noise magnitude.

- For a low value of $\eta = 0.05$, the learned system is not very different from the real one $\tilde{\alpha} = 4.195 \approx 4.2 = \alpha, \tilde{\beta} = 0.799 \approx 0.8 = \beta, \tilde{\gamma} = 6.276 \approx 6.3 = \gamma$ and $\tilde{\delta} = 0.997 \approx 1 = \delta$.
- If the magnitude of the noise is increased from $\eta = 0.05$ to $\eta = 0.1$, SINDy struggles to identify the correct terms identifying a further constant term

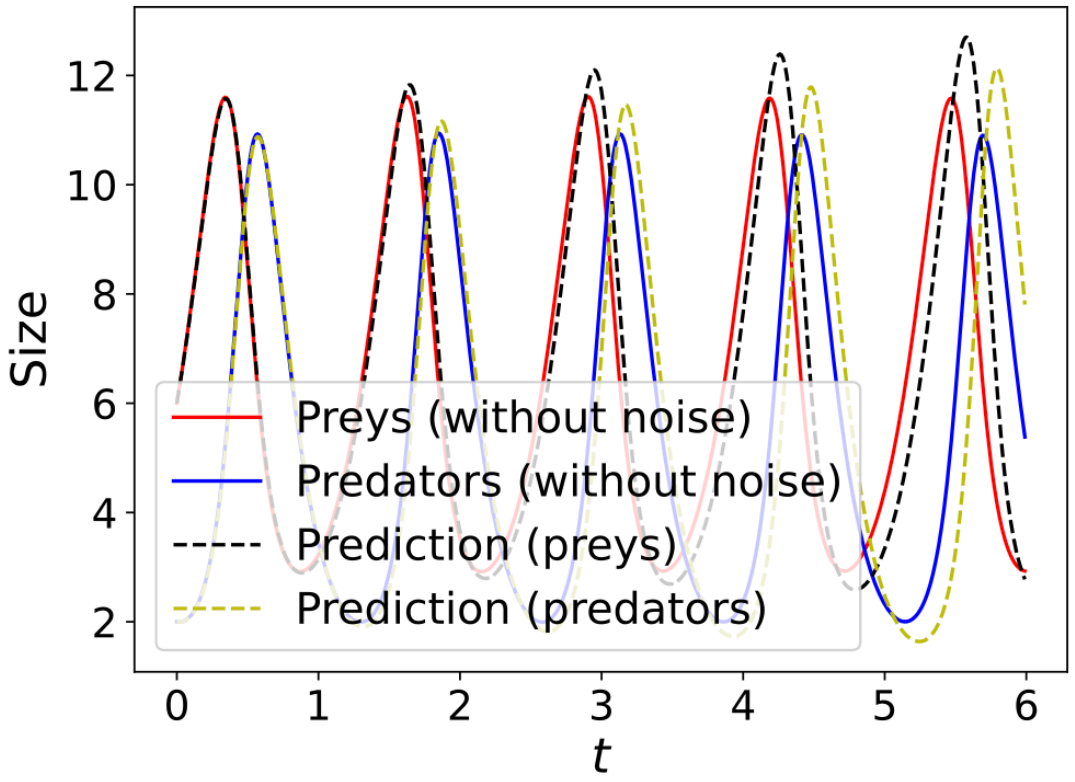
$$\begin{cases} x'(t) = -0.345 + 4.192x(t) + -0.794x(t)y(t) \\ y'(t) = -6.278y(t) + 0.997x(t)y(t). \end{cases}$$

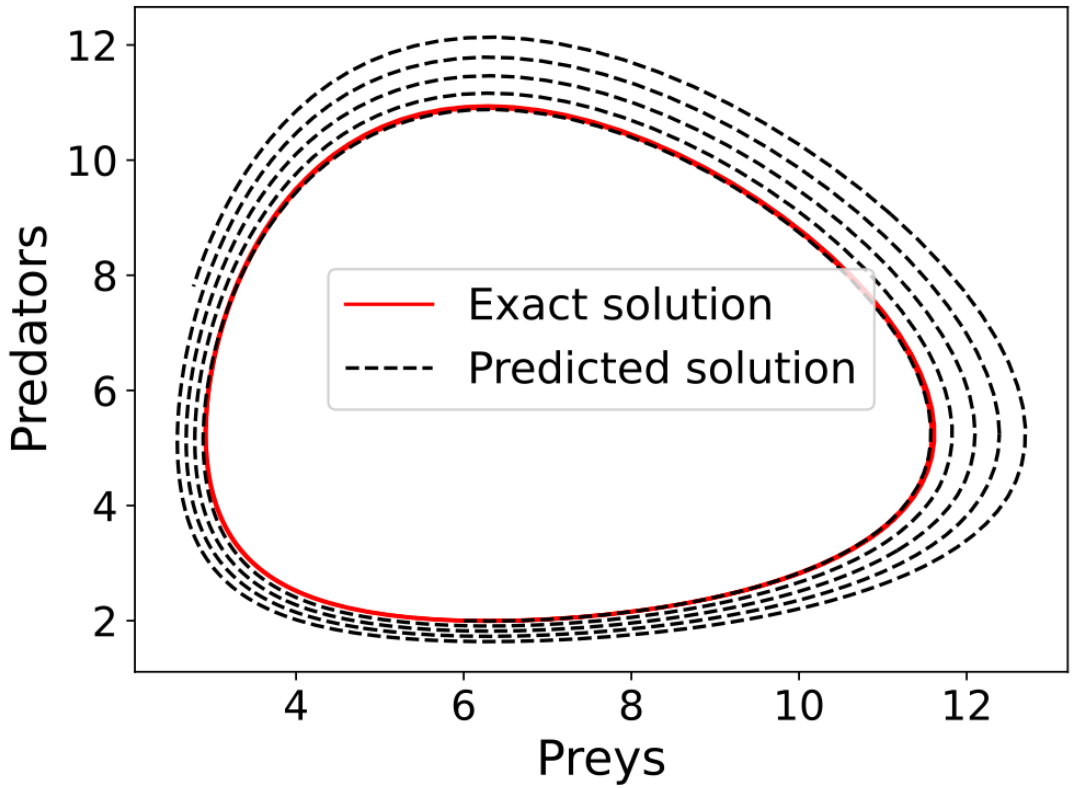
## SINDy (Lotka-Volterra + noise $\eta = 0.05$)



## SINDy (Lotka-Volterra + noise $\eta = 0.1$)



## SINDy for parameterized problems

The SINDy method can be generalized to the case of parameterized dynamical systems:

$$\begin{cases} \dot{\boldsymbol{x}}(t;\boldsymbol{\mu}) = \boldsymbol{f}(t, \boldsymbol{x}(t;\boldsymbol{\mu});\boldsymbol{\mu}) \\ \boldsymbol{x}(t_0;\boldsymbol{\mu}) = \boldsymbol{x}_0, \end{cases}$$

where $\boldsymbol{\mu} \in \mathbb{R}^p$ is a vector consisting of $p$ (possibly) time dependent parameters and/or forcing terms.

**Assumption**: The library functions can depend on the parameter $\boldsymbol{\mu}$ and the snapshot matrix ($N_\mu$ parametric samples, $N_t$ time instances) is defined as:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}(t_1, \boldsymbol{\mu}_1)^T \\ \vdots \\ \boldsymbol{x}(t_{N_t}, \boldsymbol{\mu}_1)^T \\ \boldsymbol{x}(t_1, \boldsymbol{\mu}_2)^T \\ \vdots \\ \boldsymbol{x}(t_{N_t}, \boldsymbol{\mu}_{N_\mu})^T \end{bmatrix} \in \mathbb{R}^{N_t N_\mu \times N},$$

## SINDy for parameterized problems

We can write the problem as

$$\dot{\boldsymbol{X}} = \boldsymbol{\Theta}(\boldsymbol{X}; \boldsymbol{\mu})\boldsymbol{\Xi},$$

in which $\boldsymbol{\Theta}(\boldsymbol{X};\ \boldsymbol{\mu})$ is the dictionary/data matrix containing the candidate (parametrized) functions.

To capture the dynamics, the library should include functions both of the state $\boldsymbol{x}$ and of the parameter $\mu$.

1. An analytic library of candidate functions of the state variables $\boldsymbol{\Theta}(\boldsymbol{x}^T) \in (C^0[\mathbb{R}^N, \mathbb{R}])^r$ is given, where $r$ represents the cardinality of the library.
2. Given $N_\mu$ the number of parameter instances, for $m = 1, \dots, N_\mu$ we can consider the snapshot matrix $\boldsymbol{X}_m \in \mathbb{R}^{N_t \times N}$ with all the snapshots for parameter $\mu_m$, and the matrix $\dot{\boldsymbol{X}}_m \in \mathbb{R}^{N_t \times N}$.
3. For each instance of the parameter $\mu_m$, we can build the dictionary/data matrix $\boldsymbol{\Theta}_m(\boldsymbol{X}_m) \in \mathbb{R}^{N_t \times r}$.
4. choose a set of $s$ candidate functions for the parameter $\mu$ and, proceeding as in the case of the state variables, we can build the analytic library $\boldsymbol{\Theta}(\mu) \in (C^0[\mathbb{R}, \mathbb{R}])^s$, and the corresponding dictionary/data matrix $\boldsymbol{\Theta}(\boldsymbol{\mu}) \in \mathbb{R}^{N_\mu \times s}$, where $\boldsymbol{\mu} = [\mu_1, \dots, \mu_{N_\mu}]^T$.

## SINDy for parameterized problems

**Example**: polynomials up to some degrees or other functions:

$$\boldsymbol{\Theta}(\boldsymbol{\mu}) = \begin{bmatrix} 1 & \mu_1 & \mu_1^2 & \mu_1^3 & \cdots \\ 1 & \mu_2 & \mu_2^2 & \mu_2^3 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & \mu_{N_\mu} & \mu_{N_\mu}^2 & \mu_{N_\mu}^3 & \cdots \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_{\mu_1} \\ \boldsymbol{\theta}_{\mu_2} \\ \vdots \\ \boldsymbol{\theta}_{\mu_{N_\mu}} \end{bmatrix} \in \mathbb{R}^{N_\mu \times s}.$$

Once this has been done, we can then immediately obtain an expression for the library of candidate functions in the parametric case by considering for the the symbolic and matriz library, respectively

$$\boldsymbol{\Theta}(\boldsymbol{x}^T;\, \mu) = \boldsymbol{\Theta}(\boldsymbol{x}^T) \otimes \boldsymbol{\Theta}(\mu), \quad \text{and} \quad \boldsymbol{\Theta}(\boldsymbol{X}; \boldsymbol{\mu}) = \begin{bmatrix} \boldsymbol{\theta}_{\mu_1} \otimes \boldsymbol{\Theta}(\boldsymbol{X}_1) \\ \boldsymbol{\theta}_{\mu_2} \otimes \boldsymbol{\Theta}(\boldsymbol{X}_2) \\ \vdots \\ \boldsymbol{\theta}_{\mu_{N_\mu}} \otimes \boldsymbol{\Theta}(\boldsymbol{X}_{N_\mu}) \end{bmatrix}.$$

## SINDy for PDEs

**Issue**: Given a grid on the computational domain, we can interpret the equation as a set of ordinary differential equations on the mesh points, where $N$ now represents the number of dofs.

**Idea**: Coupling ML techniques and SINDy for identification at the reduced dimension.

**Autoencoder** $(\boldsymbol{\varphi}, \boldsymbol{\psi})$, where $\boldsymbol{\varphi}(\cdot) = \boldsymbol{\varphi}(\cdot; \boldsymbol{W}_\varphi) : \mathbb{R}^N \to \mathbb{R}^n$ is the FNN encoder with weights $\boldsymbol{W}_\varphi$ and $\boldsymbol{\psi}(\cdot) = \boldsymbol{\psi}(\cdot; \boldsymbol{W}_\psi) : \mathbb{R}^n \to \mathbb{R}^N$ is the FNN decoder with weights $\boldsymbol{W}_\psi$ such that

$$\boldsymbol{\psi}\left(\boldsymbol{\varphi}(\boldsymbol{y}; \boldsymbol{W}_\varphi); \boldsymbol{W}_\psi\right) \approx \boldsymbol{y}, \quad \forall \boldsymbol{y} \in \mathbb{R}^N.$$

In the new coordinates $\boldsymbol{z}$ obtained through $\boldsymbol{\varphi}$, the system can then be written as

$$\begin{cases} \dot{\boldsymbol{z}}(t; \boldsymbol{\mu}) = \tilde{\boldsymbol{f}}(t, \boldsymbol{z}(t; \boldsymbol{\mu}); \boldsymbol{\mu}) \\ \boldsymbol{z}(t_0; \boldsymbol{\mu}) = \boldsymbol{z}_0, \end{cases}$$

in which $\tilde{\boldsymbol{f}}(\cdot\,; \boldsymbol{\mu}) : I \times \mathbb{R}^n \to \mathbb{R}^n$, $(t_0, \boldsymbol{z}_0) \in I \times \mathbb{R}^n$ identifies the dynamics of the low-dimensional system and $\dot{\boldsymbol{z}}(t) = \dot{\boldsymbol{\varphi}}(\boldsymbol{x}(t)) = \nabla_{\boldsymbol{x}} \boldsymbol{\varphi}(\boldsymbol{x}(t)) \dot{\boldsymbol{x}}(t)$ following the chain rule.

# SINDy-AE architecture

**Goal**: Find the correct set of latent coordinates for the system identification.

**How**: Two optimization problems related to the autoencoder network in $\boldsymbol{X}$ and SINDy in $\boldsymbol{z}$

$$\arg\min_{\boldsymbol{W}_\varphi, \boldsymbol{W}_\psi} \|\boldsymbol{\psi}\left(\boldsymbol{\varphi}(\boldsymbol{X};\, \boldsymbol{W}_\varphi);\, \boldsymbol{W}_\psi\right) - \boldsymbol{X}\|_2, \quad \text{and} \quad \arg\min_{\boldsymbol{\Xi}} \|\dot{\boldsymbol{Z}} - \boldsymbol{\Theta}(\boldsymbol{Z};\boldsymbol{\mu})\boldsymbol{\Xi}\|_2 + \lambda\|\boldsymbol{\Xi}\|_1,$$

given a library of candidate functions $\boldsymbol{\Theta}$, and $\boldsymbol{Z} = \boldsymbol{\varphi}(\boldsymbol{X};\, \boldsymbol{W}_\varphi)$ the encoded snapshot matrix.

**Training**: The optimization tasks are addressed at the same time, and $\boldsymbol{\Xi}$ are treated as network weights. The optimization problem which needs to be solved is given by

$$\min_{\boldsymbol{W}_\varphi, \boldsymbol{W}_\psi, \boldsymbol{\Xi}} \Bigg( \underbrace{\|\boldsymbol{X} - \boldsymbol{\psi}(\boldsymbol{\varphi}(\boldsymbol{X};\boldsymbol{W}_\varphi);\boldsymbol{W}_\psi)\|_2}_{\text{Autoencoder loss}} + \underbrace{\lambda_1\|\dot{\boldsymbol{Z}} - \boldsymbol{\Theta}(\boldsymbol{Z};\boldsymbol{\mu})\boldsymbol{\Xi}\|_2 + \lambda_2\|\boldsymbol{\Xi}\|_1}_{\text{Sparse regression loss}} + \underbrace{\lambda_3\|\dot{\boldsymbol{X}} - \nabla_{\boldsymbol{z}}\boldsymbol{\psi}(\boldsymbol{Z};\boldsymbol{W}_\psi)\boldsymbol{\Theta}(\boldsymbol{Z};\boldsymbol{\mu})\boldsymbol{\Xi}\|_2}_{\text{Consistency losses}} \Bigg)$$

# Physics-Informed Neural Networks (PINNs)

**What**: PINNs consist of differential equation residual (loss) terms, initial and boundary conditions.
The network is parametrized by weights $\theta$ and maps the inputs (variables $\boldsymbol{x}, \boldsymbol{t}$) into outputs (the field $\boldsymbol{u}$).

**Training**: Automatic differentiation computes the derivative of the output field $\boldsymbol{u}$ using the given equations.

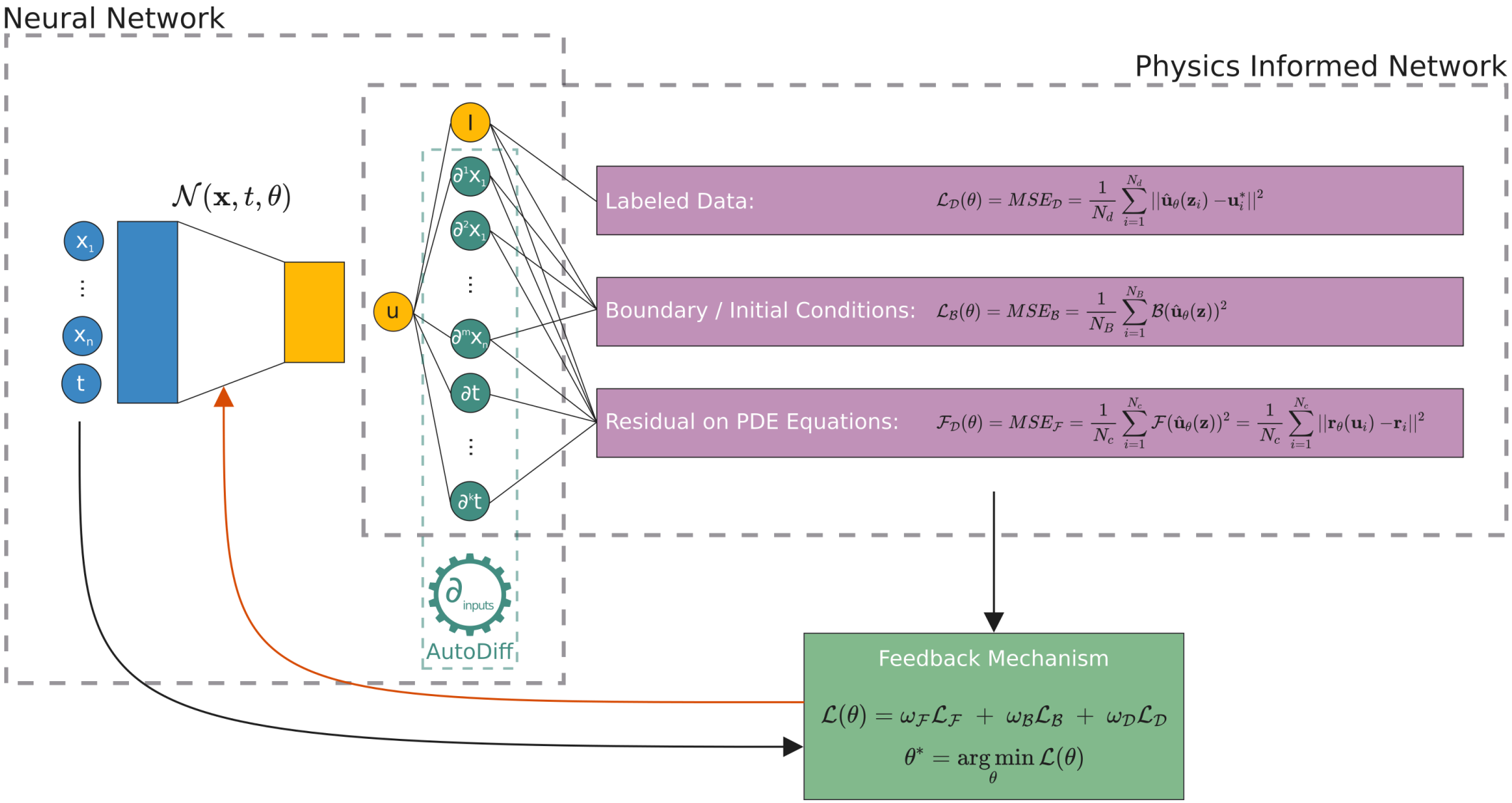
**Constraints**: The boundary/initial condition is evaluated (or hard-encoded in the NN).

**Data**: In case there is any data available, the labeled data observations are exploited.

[1] Cuomo, S., Di Cola, V.S., Giampaolo, F., Rozza, G., Raissi, M., Piccialli, F., 2022. Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next. J Sci Comput 92, 88. https://doi.org/10.1007/s10915-022-01939-z

[2] Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics 378, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

# Physics-Informed Neural Networks (PINNs)



## Physics-Informed Neural Networks (PINNs)

The PINN methodology determines the parameters $\theta$ of the NN, $\hat{\boldsymbol{u}}_\theta$, by minimizing a loss function, i.e.

$$\theta = \arg\min_{\theta}[\omega_{\mathcal{F}}\mathcal{L}_{\mathcal{F}}(\theta) + \omega_{\mathcal{B}}\mathcal{L}_{\mathcal{B}}(\theta) + \omega_d\mathcal{L}_{data}(\theta)]$$

The three terms of $\mathcal{L}$ refer to the errors in describing the initial $\mathcal{L}_i$ or boundary condition $\mathcal{L}_b$, both indicated as $\mathcal{L}_{\mathcal{B}}$, the loss respect the partial differential equation $\mathcal{L}_{\mathcal{F}}$, and the validation of known data points $\mathcal{L}_{data}$.

1. The first term, $\mathcal{L}_{\mathcal{F}}$, represents the loss produced by a mismatch with the governing differential equations $\mathcal{F}$, and enforces the differential equation $\mathcal{F}$ at the collocation points.

$$\mathcal{L}_{\mathcal{F}}(\theta) = MSE_{\mathcal{F}} = \frac{1}{N_c}\sum_{i=1}^{N_c} \|\mathcal{F}(\hat{\boldsymbol{u}}_\theta(\boldsymbol{z}_i)) - \boldsymbol{f}(\boldsymbol{z}_i)\|^2.$$

## Physics-Informed Neural Networks (PINNs)

2. The second term forces $\hat{\boldsymbol{u}}_\theta$ to mach the measurements of $\boldsymbol{u}$ over provided points $(\boldsymbol{z}, \boldsymbol{u}^*)$, which can be given as synthetic data or actual measurements

$$\mathcal{L}_{\mathcal{B}}(\theta) = MSE_{\mathcal{B}} = \frac{1}{N_B} \sum_{i=1}^{N_B} \|\mathcal{B}(\hat{\boldsymbol{u}}_\theta(\boldsymbol{z})) - \boldsymbol{g}(\boldsymbol{z}_i)\|^2$$

3. The third term is the loss due to mismatch with the boundary or initial conditions, $\mathcal{B}(\hat{\boldsymbol{u}}_\theta) = \boldsymbol{g}$

$$\mathcal{L}_{data}(\theta) = MSE_{data} = \frac{1}{N_d} \sum_{i=1}^{N_d} \|\hat{\boldsymbol{u}}_\theta(\boldsymbol{z}_i) - \boldsymbol{u}_i^*\|^2.$$

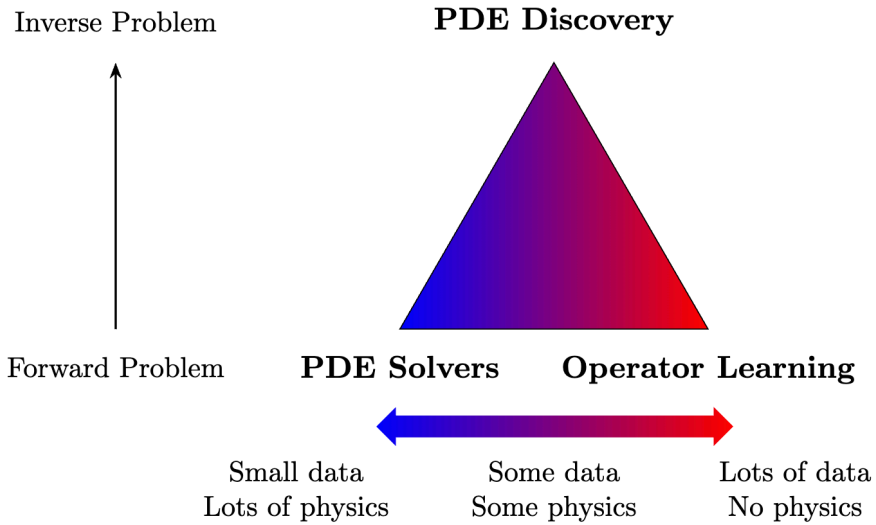
The gradients in $\mathcal{F}$ are derived using automated differentiation, so that the resulting predictions are driven to inherit any physical attributes imposed by the PDE constraint.

# Neural Operators

**Idea**: Operator learning aims to discover or approximate an unknown operator $\mathcal{A}$, which often takes the form of the solution operator associated with a differential equation.

**Problem**: Given pairs of data $(f, u)$, where $f \in \mathcal{U}$ and $u \in \mathcal{V}$ are from function spaces on a spatial domain $\Omega \subset \mathbb{R}^d$, and a (potentially nonlinear) operator $\mathcal{A} : \mathcal{U} \to \mathcal{V}$ such that $\mathcal{A}(f) = u$.

**Goal**: Find an approximation $\hat{\mathcal{A}}$ of $\mathcal{A}$ such that for any new data $f' \in \mathcal{U}$, we have $\hat{\mathcal{A}}(f') \approx \mathcal{A}(f')$.




[1] Boullé, N., Townsend, A., 2024. Chapter 3 - A mathematical guide to operator learning, in: Mishra, S., Townsend, A. (Eds.), Handbook of Numerical Analysis, Numerical Analysis Meets Machine Learning. Elsevier, pp. 83–125. https://doi.org/10.1016/bs.hna.2024.05.003

# Neural Operators

1. Represent $\hat{\mathcal{A}}$ as a neural operator, where inputs and outputs are functions, not vectors.
2. Discretize the functions at sensor points $x_1, \ldots, x_m \in \Omega$, and parametrize the neural operator with a set of parameters $\theta \in \mathbb{R}^N$
3. Formulate an optimization problem to find the best parameters:

$$\min_{\theta \in \mathbb{R}^N} \sum_{(f,u) \in \text{data}} L(\hat{\mathcal{A}}(f;\theta), u),$$

where $L$ is a loss function that measures the discrepancy between $\hat{\mathcal{A}}(f;\theta)$ and $u$, and the sum is over all available training data pairs $(f, u)$.

### Challenges

- select an appropriate neural operator architecture for $\hat{\mathcal{A}}$,
- the computational complexities of solving the optimization problem,
- the ability to generalize to new data.

# Deep Operator Networks (DeepONets)

- Learning nonlinear operators and capturing the relationships between input and output functions.
- They extend the capabilities of traditional deep learning techniques by leveraging the expressive power of neural networks to approximate any functional maps from one function space to another.
- A key theoretical motivation for DeepONet is the universal operator approximation theorem

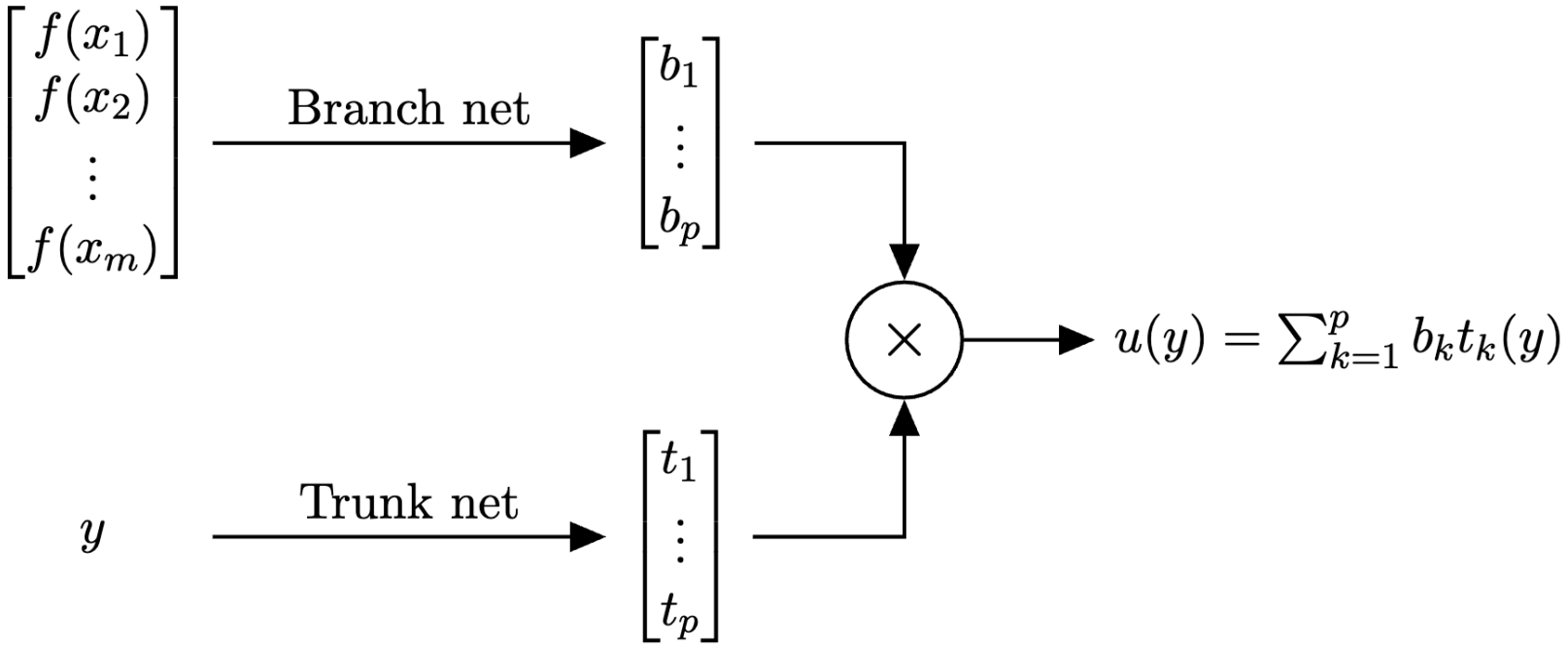
$\rightsquigarrow$ A DeepONet is a two-part deep learning network consisting of a *branch* network and a *trunk* network. The branch net encodes the operator's input functions $f$ into compact, fixed-size latent vectors $b_1(f(x_1), \ldots, f(x_m)), \ldots, b_p(f(x_1), \ldots, f(x_m))$, where $\{x_i\}_{i=1}^m$ are the sensor points at which the input functions are evaluated. The trunk net decodes these latent vectors to produce the final output function at the location $y \in \Omega$ as

$$\mathcal{N}(f)(y) = \sum_{k=1}^{p} b_k(f(x_1), \ldots, f(x_m)) t_k(y).$$

[1] Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.E., 2021. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nat Mach Intell. https://doi.org/10.1038/s42256-021-00302-5

# Deep Operator Networks (DeepONets)

A DeepONet parametrizes a neural operator using a branch network and a truncation (trunk) network. The branch network encodes the input function $f$ as a vector of $p$ features, which is then multiplied by the trunk network to yield a rank-$p$ representation of the solution $u$.



# Deep Operator Networks (DeepONets)

**Remarks**:

- **Low-rank approximation and SVD**. One can view the trunk network as learning a basis of functions $\{t_k\}_{k=1}^p$, while the branch network learns the coefficients $\{b_k\}_{k=1}^p$.
- A desirable property for a neural operator architecture is to be **discretization invariant** in the sense that the model can act on any discretization of the source term and be evaluated at any domain point.
- DeepONets can be evaluated at any location of the output domain, but are not discretization invariant in their original formulation as the branch network is evaluated at specific points of the input domain.
- The training of DeepONets involves minimizing the mean-squared error between the predicted output $\mathcal{N}(f)(y)$ and the actual output of $u$ the operator at random locations $\{y_j\}_{j=1}^n$, i.e.,

$$\min_{\theta\in\mathbb{R}^N} \frac{1}{|\text{data}|} \sum_{(f,u)\in\text{data}} \frac{1}{n}\sum_{j=1}^{n} |\mathcal{N}(f)(y_j) - u(y_j)|^2.$$

## DeepONets (diffusion-reaction)

A diffusion-reaction system with a source term $u(x)$ is described by

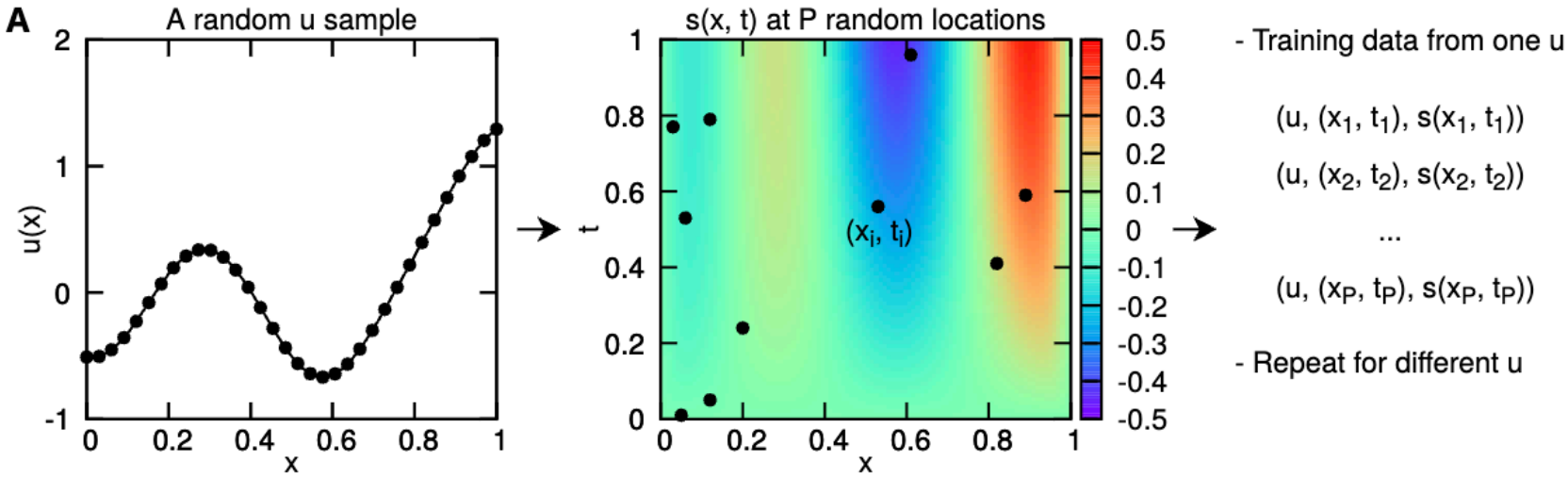
$$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x), \quad x \in [0,1], t \in [0,1],$$

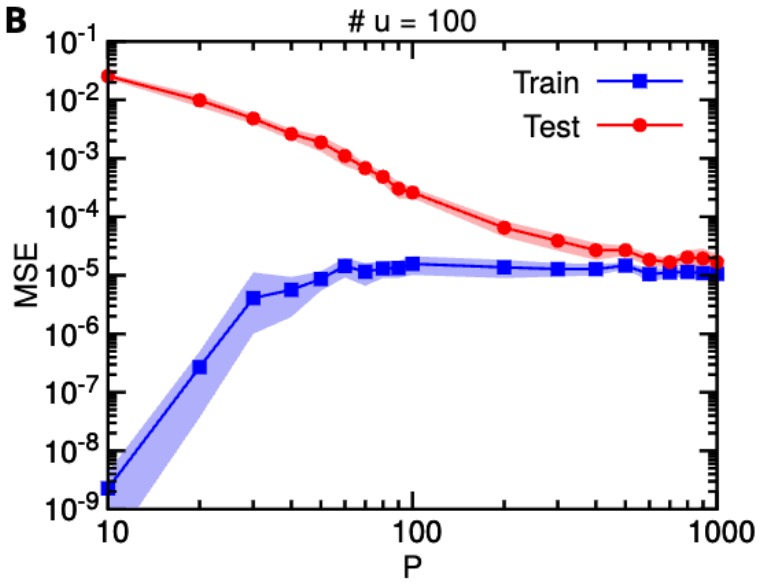
with zero initial/boundary conditions, diffusion coefficient $D = 0.01$, and reaction rate $k = 0.01$.

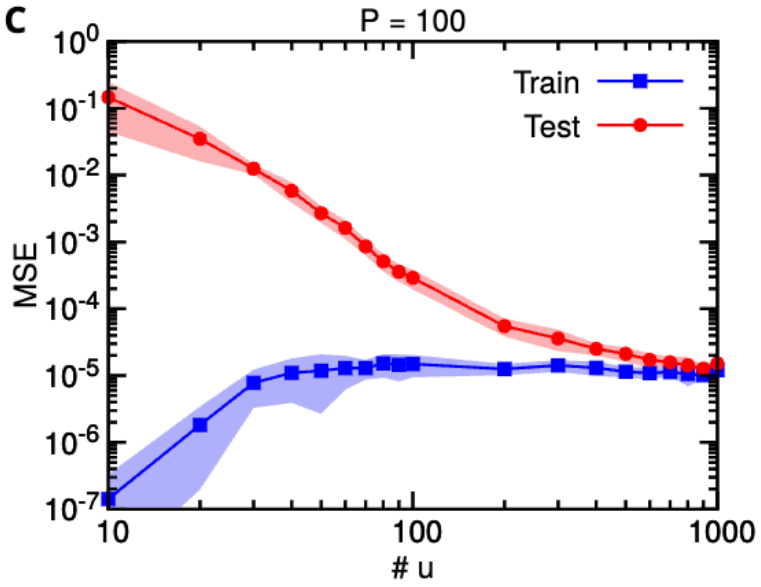
**Goal**: We use DeepONets to learn the operator mapping from $u(x)$ to the PDE solution $s(x,t)$.

**Data** 100 by 100 grid, and then for each $s$ we randomly select $P$ points out of these $10000 = 100 \times 100$ grid points (size $P$ by the number of $u$ samples).

# Deep Operator Networks (DeepONets)



## References


- Kutz, J.N., Brunton, S.L., Brunton, B.W., Proctor, J.L., 2016. Dynamic Mode Decomposition, Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611974508
- Baddoo, P.J., Herrmann, B., McKeon, B.J., Nathan Kutz, J., Brunton, S.L., 2023. Physics-informed dynamic mode decomposition. Proc. R. Soc. A. 479, 20220576. https://doi.org/10.1098/rspa.2022.0576
- Colbrook, M.J., 2023. The Multiverse of Dynamic Mode Decomposition Algorithms. https://doi.org/10.48550/arXiv.2312.00137
- Colbrook, M.J., Townsend, A., 2021. Rigorous data-driven computation of spectral properties of Koopman operators for dynamical systems.
- Geelen, R., Willcox, K., 2022. Localized non-intrusive reduced-order modelling in the operator inference framework. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 380, 20210206. https://doi.org/10.1098/rsta.2021.0206
- Geelen, R., Wright, S., Willcox, K., 2023. Operator inference for non-intrusive model reduction with quadratic manifolds. Computer Methods in Applied Mechanics and Engineering 403, 115717. https://doi.org/10.1016/j.cma.2022.115717
- Huhn, Q.A., Tano, M.E., Ragusa, J.C., Choi, Y., 2022. Parametric Dynamic Mode Decomposition for Reduced Order Modeling.
- Schmid, P.J., 2022. Dynamic Mode Decomposition and Its Variants. Annu. Rev. Fluid Mech. 54, 225–254. https://doi.org/10.1146/annurev-fluid-030121-015835
- Sun, S., Feng, L., Chan, H.S., Miličić, T., Vidaković-Koch, T., Benner, P., 2023. Parametric Dynamic Mode Decomposition for nonlinear parametric dynamical systems.
- Ichinaga, S.M., Andreuzzi, F., Demo, N., Tezzele, M., Lapo, K., Rozza, G., Brunton, S.L., Kutz, J.N., 2024. PyDMD: A Python package for robust dynamic mode decomposition. https://doi.org/10.48550/arXiv.2402.07463
- Peherstorfer, B., 2019. Sampling low-dimensional Markovian dynamics for pre-asymptotically recovering reduced models from data with operator inference. https://doi.org/10.48550/arXiv.1908.11233

## References


- Bakarji, J., Champion, K., Kutz, J.N., Brunton, S.L., 2022. Discovering Governing Equations from Partial Measurements with Deep Delay Autoencoders.
- Conti, P., Gobat, G., Fresca, S., Manzoni, A., Frangi, A., 2023. Reduced order modeling of parametrized systems through autoencoders and SINDy approach: continuation of periodic solutions. Computer Methods in Applied Mechanics and Engineering 411, 116072.
- Forootani, A., Benner, P., 2024. GN-SINDy: Greedy Sampling Neural Network in Sparse Identification of Nonlinear Partial Differential Equations. https://doi.org/10.48550/arXiv.2405.08613
- Fukami, K., Murata, T., Zhang, K., Fukagata, K., 2021. Sparse identification of nonlinear dynamics with low-dimensionalized flow representations. https://doi.org/10.48550/arXiv.2010.12177
- Gao, M.L., Kutz, J.N., Font, B., 2025. Mesh-free sparse identification of nonlinear dynamics. https://doi.org/10.48550/arXiv.2505.16058
- He, X., Tran, A., Bortz, D.M., Choi, Y., 2024. WgLaSDI: Weak-Form Greedy Latent Space Dynamics Identification.
- Zheng, H., Lin, G., 2024. LES-SINDy: Laplace-Enhanced Sparse Identification of Nonlinear Dynamical Systems. https://doi.org/10.48550/arXiv.2411.01719
- Conti, P., Kneifl, J., Manzoni, A., Frangi, A., Fehr, J., Brunton, S.L., Kutz, J.N., 2024. VENI, VINDy, VICI: a variational reduced-order modeling framework with uncertainty quantification. https://doi.org/10.48550/arXiv.2405.20905
- Tomada, L., Khamlich, M., Pichi, F., Rozza, G., 2025. Sparse Identification for bifurcating phenomena in Computational Fluid Dynamics. https://doi.org/10.48550/arXiv.2502.11194
- Coscia, D., Ivagnes, A., Demo, N., Rozza, G., 2023. Physics-Informed Neural networks for Advanced modeling. Journal of Open Source Software 8, 5352. https://doi.org/10.21105/joss.05352
- Silva, B.M. de, Champion, K., Quade, M., Loiseau, J.-C., Kutz, J.N., Brunton, S.L., 2020. PySINDy: A Python package for the sparse identification of nonlinear dynamical systems from data. Journal of Open Source Software 5, 2104. https://doi.org/10.21105/joss.02104