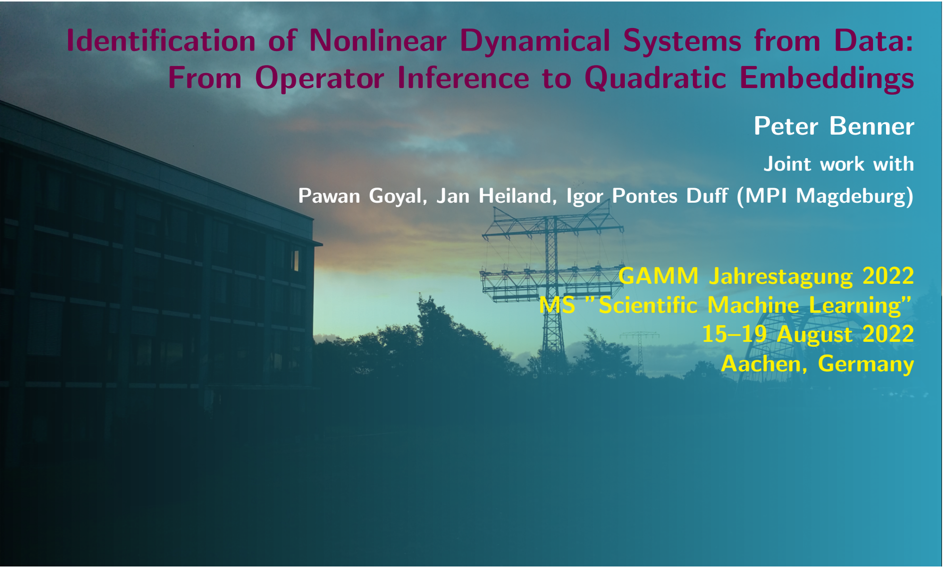
MAX PLANCK INSTITUTE
FOR DYNAMICS OF COMPLEX
TECHNICAL SYSTEMS
MAGDEBURG

CSC

COMPUTATIONAL METHODS IN
SYSTEMS AND CONTROL THEORY



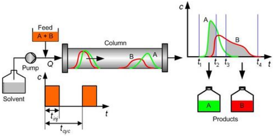
Identification of Nonlinear Dynamical Systems from Data: From Operator Inference to Quadratic Embeddings

Peter Benner

Joint work with

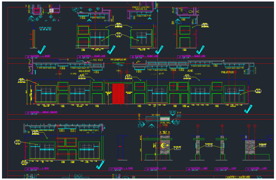
Pawan Goyal, Jan Heiland, Igor Pontes Duff (MPI Magdeburg)

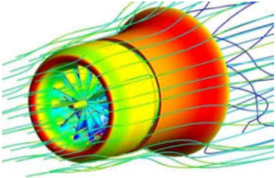
GAMM Jahrestagung 2022
MS "Scientific Machine Learning"
15–19 August 2022
Aachen, Germany



Dynamic models are important for

- analysis of transient behavior under operating conditions,
- control synthesis and design,
- parameter optimization and optimal control,
- long-time horizon prediction (health monitoring, digital twins).





Problem set-up

- Construct a mathematical model

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)),$$

describing dynamics of the process.

Our Objective

Problem set-up

- Construct a mathematical model

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)),$$

describing dynamics of the process.

- Neural network-based approaches: RNNs, LSTM, Neural ODEs, ... $\rightsquigarrow$ black-box models

Our Objective

Problem set-up

- Construct a mathematical model

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)),$$

  describing dynamics of the process.

- Neural network-based approaches: RNNs, LSTM, Neural ODEs, ... $\rightsquigarrow$ black-box models
- Engineering design e.g., control, optimization, can be difficult.

Problem set-up

- Construct a mathematical model

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)),$$

describing dynamics of the process.

- Neural network-based approaches: RNNs, LSTM, Neural ODEs, ... $\rightsquigarrow$ black-box models
- Engineering design e.g., control, optimization, can be difficult.

Goal

- Construct *simple dynamical models*, capturing important dynamic behavior in a **state-space model** that facilitates engineering tasks.

- The simplest type of model one can think of is a Linear Model
  - $\rightsquigarrow$ Many tools for optimal/feedback control, optimization, and prediction

Koopman Operator and DMD

- The simplest type of model one can think of is a Linear Model
  - $\rightsquigarrow$ Many tools for optimal/feedback control, optimization, and prediction

- Given data $\mathbf{x}(t_i)$ and its derivative $\dot{\mathbf{x}}(t_i)$, a linear model can be determined by solving

$$\min_{\mathbf{A}} \|\dot{\mathbf{X}} - \mathbf{A}\mathbf{X}\|,$$

where $\dot{\mathbf{X}} = [\dot{\mathbf{x}}(t_1), \ldots, \dot{\mathbf{x}}(t_n)]$ and $\mathbf{X} = [\mathbf{x}(t_1), \ldots, \mathbf{x}(t_n)]$.

- The simplest type of model one can think of is a Linear Model
  - ⇝ Many tools for optimal/feedback control, optimization, and prediction

- Given data $\mathbf{x}(t_i)$ and its derivative $\dot{\mathbf{x}}(t_i)$, a linear model can be determined by solving

$$\min_{\mathbf{A}} \|\dot{\mathbf{X}} - \mathbf{A}\mathbf{X}\|,$$

  where $\dot{\mathbf{X}} = [\dot{\mathbf{x}}(t_1), \ldots, \dot{\mathbf{x}}(t_n)]$ and $\mathbf{X} = [\mathbf{x}(t_1), \ldots, \mathbf{x}(t_n)]$.

- Often referred to (in a simplistic view) as Dynamic Mode Decomposition or Operator Inference.

Koopman Operator and DMD

- The simplest type of model one can think of is a Linear Model
  - ⇝ Many tools for optimal/feedback control, optimization, and prediction

- Given data $\mathbf{x}(t_i)$ and its derivative $\dot{\mathbf{x}}(t_i)$, a linear model can be determined by solving
$$\min_{\mathbf{A}} \|\dot{\mathbf{X}} - \mathbf{A}\mathbf{X}\|,$$
where $\dot{\mathbf{X}} = [\dot{\mathbf{x}}(t_1), \dots, \dot{\mathbf{x}}(t_n)]$ and $\mathbf{X} = [\mathbf{x}(t_1), \dots, \mathbf{x}(t_n)]$.

- Often referred to (in a simplistic view) as Dynamic Mode Decomposition or Operator Inference.

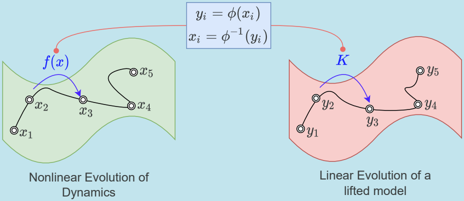
- Once a linear model is learned and verified, it can be deployed for control and design tasks.

Koopman Operator and DMD

- The simplest type of model one can think of is a Linear Model
  - $\rightsquigarrow$ Many tools for optimal/feedback control, optimization, and prediction
- Given data $\mathbf{x}(t_i)$ and its derivative $\dot{\mathbf{x}}(t_i)$, a linear model can be determined by solving
$$\min_{\mathbf{A}} \|\dot{\mathbf{X}} - \mathbf{A}\mathbf{X}\|,$$
where $\dot{\mathbf{X}} = [\dot{\mathbf{x}}(t_1), \ldots, \dot{\mathbf{x}}(t_n)]$ and $\mathbf{X} = [\mathbf{x}(t_1), \ldots, \mathbf{x}(t_n)]$.
- Often referred to (in a simplistic view) as Dynamic Mode Decomposition or Operator Inference.
- Once a linear model is learned and verified, it can be deployed for control and design tasks.
- However, several challenges remain:
  - Often, one cannot measure the full state $\mathbf{x} \rightsquigarrow$ partial measurements!
  - Real-world processes are often nonlinear, thus learning a linear model may not be sufficient to characterize complex dynamic behavior.

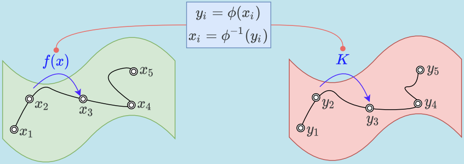
Koopman Operator in Nutshell (Koopman 1931)

A nonlinear dynamical system $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$ can be written as a linear system in a infinite dimensional Hilbert space.



Koopman Operator in Nutshell (Koopman 1931)

A nonlinear dynamical system $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$ can be written as a linear system in a infinite dimensional Hilbert space.



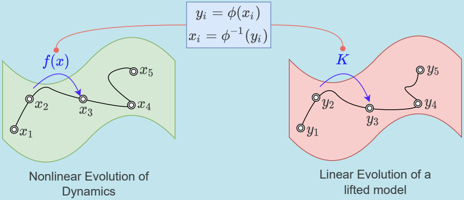
Extended DMD (Williams et al. 2015)

- The aim is to approximate infinite dimensional Koopman linear operator via a finite dimensional one.

Koopman Operator and DMD

Koopman Operator in Nutshell (Koopman 1931)

A nonlinear dynamical system $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$ can be written as a linear system in a infinite dimensional Hilbert space.



Extended DMD (Williams et al. 2015)

- The aim is to approximate infinite dimensional Koopman linear operator via a finite dimensional one.
- For this, often hand-design observables are needed,
  - but challenging design decisions need to be taken, and it still is an approximation.

Our "Holy Grail"

- Try to re-write a nonlinear system using a simple structure in finite dimensions.
  - In Koopman theory, the structure is that of a linear system and it is infinite dimensional.

Our "Holy Grail"

- Try to re-write a nonlinear system using a simple structure in finite dimensions.
  - In Koopman theory, the structure is that of a linear system and it is infinite dimensional.
- Mapping from the observables to the state is linear (at least for good low-dimensional linear representation).

Our "Holy Grail"

- Try to re-write a nonlinear system using a simple structure in finite dimensions.
  - In Koopman theory, the structure is that of a linear system and it is infinite dimensional.
- Mapping from the observables to the state is linear (at least for good low-dimensional linear representation).

Lifting Principle

- Try to re-write a nonlinear system using a simple structure in finite dimensions.
  - In Koopman theory, the structure is that of a linear system and it is infinite dimensional.
- Mapping from the observables to the state is linear (at least for good low-dimensional linear representation).

Lifting Principle

- McCormick proposed a convex relaxation to solve nonlinear non-convex optimization problems. (McCormick 1976)

Our "Holy Grail"

- Try to re-write a nonlinear system using a simple structure in finite dimensions.
  - In Koopman theory, the structure is that of a linear system and it is infinite dimensional.
- Mapping from the observables to the state is linear (at least for good low-dimensional linear representation).

Lifting Principle

- McCormick proposed a convex relaxation to solve nonlinear non-convex optimization problems. (McCormick 1976)
- Key ingredient is lifting the optimization problem to a higher dimensional space using auxiliary variables (similar to observables in Koopman theory).

- Try to re-write a nonlinear system using a simple structure in finite dimensions.
  - In Koopman theory, the structure is that of a linear system and it is infinite dimensional.
- Mapping from the observables to the state is linear (at least for good low-dimensional linear representation).

Lifting Principle

- McCormick proposed a convex relaxation to solve nonlinear non-convex optimization problems. (McCormick 1976)
- Key ingredient is lifting the optimization problem to a higher dimensional space using auxiliary variables (similar to observables in Koopman theory).
- This ideas has been further developed for learning dynamical systems.

Lifted Nonlinear Dynamical Models using Lifting principle

- Consider a nonlinear system of the generic form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}),$$

where $\mathbf{x} \in \mathbb{R}^n$, and the function $\mathbf{f}(\cdot)$ is assumed to be smooth enough.

Lifting Principle for Dynamical Systems

Lifted Nonlinear Dynamical Models using Lifting principle

- Consider a nonlinear system of the generic form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}),$$

where $\mathbf{x} \in \mathbb{R}^n$, and the function $\mathbf{f}(\cdot)$ is assumed to be smooth enough.

- Then, there exists a lifting $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^m$, and its "left" inverse mapping $\mathcal{L}^\sharp : \mathbb{R}^m \to \mathbb{R}^n$, resulting in a quadratic model

$$\dot{\mathbf{y}}(t) = \mathbf{A}\mathbf{y} + \mathbf{H}\left(\mathbf{y}(t) \otimes \mathbf{y}(t)\right) + \mathbf{B},$$

where $\mathbf{y}(t) = \mathcal{L}(\mathbf{x}(t))$ and $\mathcal{L}^\sharp\left(\mathbf{y}(t)\right) = \mathbf{x}(t)$.

Lifted Nonlinear Dynamical Models using Lifting principle

- Consider a nonlinear system of the generic form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}),$$

  where $\mathbf{x} \in \mathbb{R}^n$, and the function $\mathbf{f}(\cdot)$ is assumed to be smooth enough.
- Then, there exists a lifting $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^m$, and its "left" inverse mapping $\mathcal{L}^\sharp : \mathbb{R}^m \to \mathbb{R}^n$, resulting in a quadratic model

$$\dot{\mathbf{y}}(t) = \mathbf{A}\mathbf{y} + \mathbf{H}\left(\mathbf{y}(t) \otimes \mathbf{y}(t)\right) + \mathbf{B},$$

  where $\mathbf{y}(t) = \mathcal{L}(\mathbf{x}(t))$ and $\mathcal{L}^\sharp\left(\mathbf{y}(t)\right) = \mathbf{x}(t)$.

- Such a lifting concept was first developed by (Savageau/Voit 1987) for control purposes.
- Also used for model reduction for nonlinear systems (Gu 2009, B./Breiten 2015).

Lifted Nonlinear Dynamical Models using Lifting principle

- Consider a nonlinear system of the generic form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}),$$

where $\mathbf{x} \in \mathbb{R}^n$, and the function $\mathbf{f}(\cdot)$ is assumed to be smooth enough.

- Then, there exists a lifting $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^m$, and its "left" inverse mapping $\mathcal{L}^\sharp : \mathbb{R}^m \to \mathbb{R}^n$, resulting in a quadratic model

$$\dot{\mathbf{y}}(t) = \mathbf{A}\mathbf{y} + \mathbf{H}\left(\mathbf{y}(t) \otimes \mathbf{y}(t)\right) + \mathbf{B},$$

where $\mathbf{y}(t) = \mathcal{L}(\mathbf{x}(t))$ and $\mathcal{L}^\sharp\left(\mathbf{y}(t)\right) = \mathbf{x}(t)$.

- Such a lifting concept was first developed by (Savageau/Voit 1987) for control purposes.
- Also used for model reduction for nonlinear systems (Gu 2009, B./Breiten 2015).
- Recently, it has become popular using terminology Lift and Learn by Willcox, Peherstorfer, Qian, Krämer, . . . (Qian et al. 2019).

An illustration

- Consider the simple pendulum model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\sin(x_2) \\ x_1 \end{bmatrix}.$$

An illustration

- Consider the simple pendulum model:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-\sin(x_2)\\ x_1\end{bmatrix}.$$

- Lifted coordinates (observables) and the corresponding inverse transformation:

$$\mathcal{L}: \begin{bmatrix}x_1\\ x_2\end{bmatrix} \mapsto \begin{bmatrix}x_1\\ x_2\\ \sin(x_2)\\ \cos(x_2)\end{bmatrix} =: \begin{bmatrix}y_1\\ y_2\\ y_3\\ y_4\end{bmatrix}, \qquad \mathcal{L}^\sharp: \begin{bmatrix}y_1\\ y_2\\ y_3\\ y_4\end{bmatrix} \mapsto \begin{bmatrix}y_1\\ y_2\end{bmatrix} \equiv \begin{bmatrix}x_1\\ x_2\end{bmatrix}.$$

An illustration

- Consider the simple pendulum model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\sin(x_2) \\ x_1 \end{bmatrix}.$$

- Lifted coordinates (observables) and the corresponding inverse transformation:

$$\mathcal{L} : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mapsto \begin{bmatrix} x_1 \\ x_2 \\ \sin(x_2) \\ \cos(x_2) \end{bmatrix} =: \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}, \qquad \mathcal{L}^\sharp : \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \mapsto \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \equiv \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

- Consequently, we can write the dynamics in the variables y_i as a quadratic system:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \dot{y}_4 \end{bmatrix} = \begin{bmatrix} -y_3 \\ y_1 \\ y_1 y_4 \\ -y_1 y_3 \end{bmatrix}.$$

An illustration

- Consider the simple pendulum model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\sin(x_2) \\ x_1 \end{bmatrix}.$$

- Lifted coordinates (observables) and the corresponding inverse transformation:

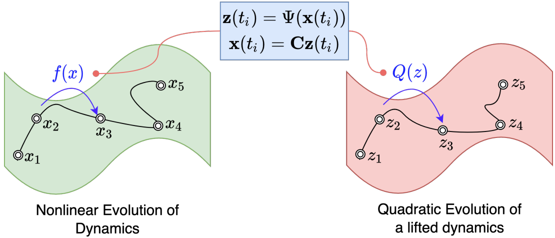
$$\mathcal{L} : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mapsto \begin{bmatrix} x_1 \\ x_2 \\ \sin(x_2) \\ \cos(x_2) \end{bmatrix} =: \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}, \qquad \mathcal{L}^\sharp : \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \mapsto \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \equiv \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

- Consequently, we can write the dynamics in the variables y_i as a quadratic system:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \dot{y}_4 \end{bmatrix} = \begin{bmatrix} -y_3 \\ y_1 \\ y_1 y_4 \\ -y_1 y_3 \end{bmatrix}.$$

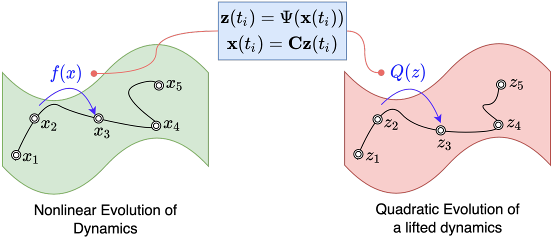
- Note that the inverse mapping is indeed linear.

- Using observables—inspired by *lifting principle*—we can write nonlinear systems as quadratic systems
  - which are finite dimensional
  - for which we can reconstruct full-state using a linear projection (restriction) of observables.
  - so that for a given nonlinear system, lifted observables are easy to determine.



Lifting Principle for Dynamical Systems

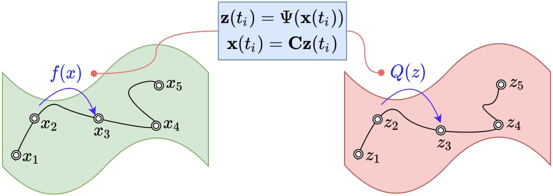
- Using observables—inspired by *lifting principle*—we can write nonlinear systems as quadratic systems
  - which are finite dimensional
  - for which we can reconstruct full-state using a linear projection (restriction) of observables.
  - so that for a given nonlinear system, lifted observables are easy to determine.



- For given nonlinear dynamical models, we can determine suitable observables.

Lifting Principle for Dynamical Systems

- Using observables—inspired by *lifting principle*—we can write nonlinear systems as quadratic systems
  - which are finite dimensional
  - for which we can reconstruct full-state using a linear projection (restriction) of observables.
  - so that for a given nonlinear system, lifted observables are easy to determine.



Nonlinear Evolution of Dynamics

Quadratic Evolution of a lifted dynamics

- For given nonlinear dynamical models, we can determine suitable observables.
- However, our goal itself is to learn dynamical models from data.

Problem Statement (for fast decay of Kolmogorov n-width) (Goyal/Benner 2022)

Given data $\{\mathbf{x}(t_1), \ldots, \mathbf{x}(t_{\mathcal{N}})\}$ and derivative information $\{\dot{\mathbf{x}}(t_1), \ldots, \dot{\mathbf{x}}(t_{\mathcal{N}})\}$, we seek to identify

Problem Statement (for fast decay of Kolmogorov n-width) **(Goyal/Benner 2022)**

Given data $\{\mathbf{x}(t_1), \ldots, \mathbf{x}(t_\mathcal{N})\}$ and derivative information $\{\dot{\mathbf{x}}(t_1), \ldots, \dot{\mathbf{x}}(t_\mathcal{N})\}$, we seek to identify

- observables $\mathbf{z} := \psi(\mathbf{x})$ such that

$$\begin{aligned}\dot{\mathbf{z}}(t) &= \mathbf{A}\mathbf{z}(t) + \mathbf{H}\left(\mathbf{z}(t) \otimes \mathbf{z}(t)\right) + \mathbf{B} =: \mathcal{Q}(\mathbf{z}),\\ \mathbf{x}(t) &= \mathbf{C}\mathbf{z}(t).\end{aligned}$$

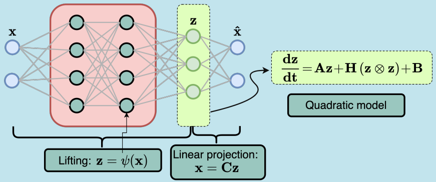
Problem Statement (for fast decay of Kolmogorov n-width) **(Goyal/Benner 2022)**

Given data $\{\mathbf{x}(t_1), \ldots, \mathbf{x}(t_\mathcal{N})\}$ and derivative information $\{\dot{\mathbf{x}}(t_1), \ldots, \dot{\mathbf{x}}(t_\mathcal{N})\}$, we seek to identify

- observables $\mathbf{z} := \psi(\mathbf{x})$ such that

$$\dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{H}\left(\mathbf{z}(t) \otimes \mathbf{z}(t)\right) + \mathbf{B} =: \mathcal{Q}(\mathbf{z}),$$
$$\mathbf{x}(t) = \mathbf{C}\mathbf{z}(t).$$

- Since we do not have any prior information, we learn $\psi(\cdot)$ using a neural network.



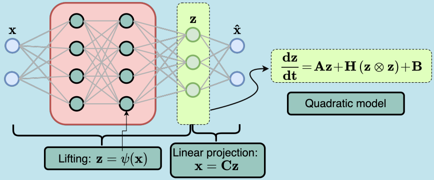
Problem Statement (for fast decay of Kolmogorov n-width) **(Goyal/Benner 2022)**

Given data $\{\mathbf{x}(t_1), \ldots, \mathbf{x}(t_\mathcal{N})\}$ and derivative information $\{\dot{\mathbf{x}}(t_1), \ldots, \dot{\mathbf{x}}(t_\mathcal{N})\}$, we seek to identify

- observables $\mathbf{z} := \psi(\mathbf{x})$ such that

$$\dot{\mathbf{z}}(t) = \mathbf{A}\mathbf{z}(t) + \mathbf{H}\left(\mathbf{z}(t) \otimes \mathbf{z}(t)\right) + \mathbf{B} =: \mathcal{Q}(\mathbf{z}),$$
$$\mathbf{x}(t) = \mathbf{C}\mathbf{z}(t).$$

- Since we do not have any prior information, we learn $\psi(\cdot)$ using a neural network.
- We learn parameters of neural network $\psi(\cdot)$ and the system matrices $\{\mathbf{A}, \mathbf{H}, \mathbf{B}, \mathbf{C}\}$ simultaneously.



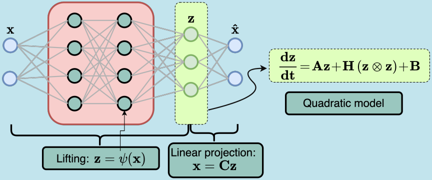
Lifting Principle for Dynamical Systems

Loss function (Goyal/B. 2022)

- Compute $\dot{\mathbf{z}}$ using $\dot{\mathbf{x}}$ by the chain rule:

$$\mathcal{L}_{\dot{\mathbf{z}}\dot{\mathbf{x}}} = \left\| \left(\nabla_{\mathbf{x}} \mathbf{z}\right) \dot{\mathbf{x}} - \mathcal{Q}(\mathbf{z}) \right\|$$

where $\mathcal{Q}(\mathbf{z}) := \left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z} \otimes \mathbf{z}\right) + \mathbf{B}\right)$ and $\mathbf{z} = \Psi(\mathbf{x})$.

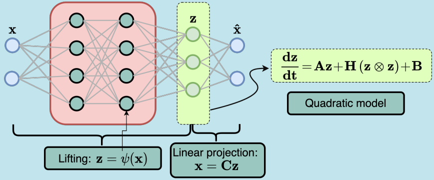


Loss function (Goyal/B. 2022)

- Compute $\dot{\mathbf{z}}$ using $\dot{\mathbf{x}}$ by the chain rule:

$$\mathcal{L}_{\dot{\mathbf{z}}\dot{\mathbf{x}}} = \|\left(\nabla_{\mathbf{x}}\mathbf{z}\right)\dot{\mathbf{x}} - \mathcal{Q}(\mathbf{z})\|$$

where $\mathcal{Q}(\mathbf{z}) := \left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z}\otimes\mathbf{z}\right) + \mathbf{B}\right)$
and $\mathbf{z} = \Psi(\mathbf{x})$.



- Compute $\dot{\mathbf{x}}$ using $\dot{\mathbf{z}}$:

$$\dot{\mathbf{x}} = \mathbf{C}\dot{\mathbf{z}} = \mathbf{C}\left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z}\otimes\mathbf{z}\right) + \mathbf{B}\right), \text{ yielding}$$

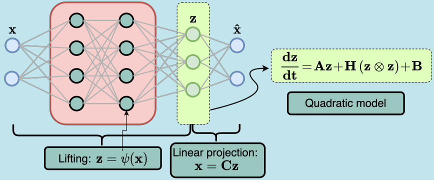
$$\mathcal{L}_{\dot{\mathbf{x}}\dot{\mathbf{z}}} = \|\dot{\mathbf{x}} - \mathbf{C}\left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z}\otimes\mathbf{z}\right) + \mathbf{B}\right)\|.$$

Loss function (Goyal/B. 2022)

- Compute $\dot{\mathbf{z}}$ using $\dot{\mathbf{x}}$ by the chain rule:

$$\mathcal{L}_{\dot{\mathbf{z}}\dot{\mathbf{x}}} = \| \left(\nabla_{\mathbf{x}}\mathbf{z}\right) \dot{\mathbf{x}} - \mathcal{Q}(\mathbf{z}) \|$$

  where $\mathcal{Q}(\mathbf{z}) := \left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z} \otimes \mathbf{z}\right) + \mathbf{B}\right)$ and $\mathbf{z} = \Psi(\mathbf{x})$.



- Compute $\dot{\mathbf{x}}$ using $\dot{\mathbf{z}}$:

$$\dot{\mathbf{x}} = \mathbf{C}\dot{\mathbf{z}} = \mathbf{C}\left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z} \otimes \mathbf{z}\right) + \mathbf{B}\right), \text{ yielding}$$

$$\mathcal{L}_{\dot{\mathbf{x}}\dot{\mathbf{z}}} = \|\dot{\mathbf{x}} - \mathbf{C}\left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z} \otimes \mathbf{z}\right) + \mathbf{B}\right)\|.$$

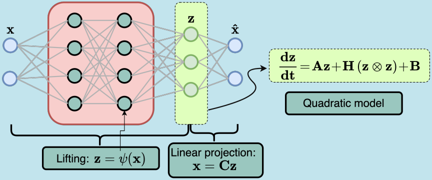
- Autoencoder loss: $\mathcal{L}_{\text{encdec}} = \|\mathbf{x} - \mathbf{C}\Psi(\mathbf{x})\|$.

Loss function (Goyal/B. 2022)

- Compute $\dot{\mathbf{z}}$ using $\dot{\mathbf{x}}$ by the chain rule:

$$\mathcal{L}_{\dot{\mathbf{z}}\dot{\mathbf{x}}} = \| \left(\nabla_{\mathbf{x}}\mathbf{z}\right)\dot{\mathbf{x}} - \mathcal{Q}(\mathbf{z})\|$$

where $\mathcal{Q}(\mathbf{z}) := \left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z}\otimes\mathbf{z}\right) + \mathbf{B}\right)$ and $\mathbf{z} = \Psi(\mathbf{x})$.



- Compute $\dot{\mathbf{x}}$ using $\dot{\mathbf{z}}$:

$$\dot{\mathbf{x}} = \mathbf{C}\dot{\mathbf{z}} = \mathbf{C}\left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z}\otimes\mathbf{z}\right) + \mathbf{B}\right), \text{ yielding}$$

$$\mathcal{L}_{\dot{\mathbf{x}}\dot{\mathbf{z}}} = \|\dot{\mathbf{x}} - \mathbf{C}\left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z}\otimes\mathbf{z}\right) + \mathbf{B}\right)\|.$$

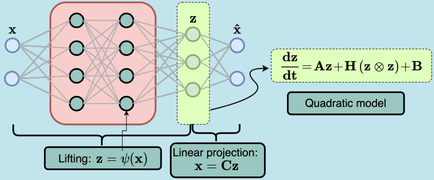
- Autoencoder loss: $\mathcal{L}_{\text{encdec}} = \|\mathbf{x} - \mathbf{C}\Psi(\mathbf{x})\|$.
- Total loss is $\mathcal{L} = \mathcal{L}_{\text{encdec}} + \mathcal{L}_{\dot{\mathbf{x}}\dot{\mathbf{z}}} + \mathcal{L}_{\dot{\mathbf{z}}\dot{\mathbf{x}}}$.

Loss function (Goyal/B. 2022)

- Compute $\dot{\mathbf{z}}$ using $\dot{\mathbf{x}}$ by the chain rule:

$$\mathcal{L}_{\dot{\mathbf{z}}\dot{\mathbf{x}}} = \| \left(\nabla_{\mathbf{x}}\mathbf{z}\right)\dot{\mathbf{x}} - \mathcal{Q}(\mathbf{z})\|$$

where $\mathcal{Q}(\mathbf{z}) := \left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z}\otimes\mathbf{z}\right) + \mathbf{B}\right)$ and $\mathbf{z} = \Psi(\mathbf{x})$.



- Compute $\dot{\mathbf{x}}$ using $\dot{\mathbf{z}}$:

$$\dot{\mathbf{x}} = \mathbf{C}\dot{\mathbf{z}} = \mathbf{C}\left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z}\otimes\mathbf{z}\right) + \mathbf{B}\right), \text{ yielding}$$

$$\mathcal{L}_{\dot{\mathbf{x}}\dot{\mathbf{z}}} = \|\dot{\mathbf{x}} - \mathbf{C}\left(\mathbf{A}\mathbf{z} + \mathbf{H}\left(\mathbf{z}\otimes\mathbf{z}\right) + \mathbf{B}\right)\|.$$

- Autoencoder loss: $\mathcal{L}_{\text{encdec}} = \|\mathbf{x} - \mathbf{C}\Psi(\mathbf{x})\|$.
- Total loss is $\mathcal{L} = \mathcal{L}_{\text{encdec}} + \mathcal{L}_{\dot{\mathbf{x}}\dot{\mathbf{z}}} + \mathcal{L}_{\dot{\mathbf{z}}\dot{\mathbf{x}}}$.

- Note that once we have all these parameters, we need encoder (neural network) only to get initial condition for $\mathbf{z}$.
- The rest of the model is very classical state-space quadratic model
  $\rightsquigarrow$ can be used for engineering design.

Lambda–Omega reaction–diffusion example

- The governing equations are

$$
\begin{aligned}
u_t &= (1-(u^2+v^2))u + \beta(u^2+v^2)v + d_1(u_{xx}+u_{yy}),\\
v_t &= -\beta(u^2+v^2)u + (1-(u^2+v^2))v + d_2(v_{xx}+v_{yy}).
\end{aligned}
$$

- We take 100×100 grid and collect 100 data points in time $t = [0, 5]$.

Lambda–Omega reaction–diffusion example

- The governing equations are

$$u_t = (1 - (u^2 + v^2))u + \beta(u^2 + v^2)v + d_1(u_{xx} + u_{yy}),$$
$$v_t = -\beta(u^2 + v^2)u + (1 - (u^2 + v^2))v + d_2(v_{xx} + v_{yy}).$$

- We take 100×100 grid and collect 100 data points in time $t = [0, 5]$.
- We consider the first 75 points for training and the last 25 for testing.

Lambda–Omega reaction–diffusion example

- The governing equations are

$$
\begin{aligned}
u_t &= (1-(u^2+v^2))u + \beta(u^2+v^2)v + d_1(u_{xx}+u_{yy}),\\
v_t &= -\beta(u^2+v^2)u + (1-(u^2+v^2))v + d_2(v_{xx}+v_{yy}).
\end{aligned}
$$

- We take 100×100 grid and collect 100 data points in time $t = [0, 5]$.
- We consider the first 75 points for training and the last 25 for testing.
- The data are high-dimensional $(2 \cdot 10^4)$ and exhibit an exponential decay of singular values, so we compress the data using projection onto the first two POD models.

Lambda–Omega reaction–diffusion example

- The governing equations are

$$u_t = (1 - (u^2 + v^2))u + \beta(u^2 + v^2)v + d_1(u_{xx} + u_{yy}),$$
$$v_t = -\beta(u^2 + v^2)u + (1 - (u^2 + v^2))v + d_2(v_{xx} + v_{yy}).$$

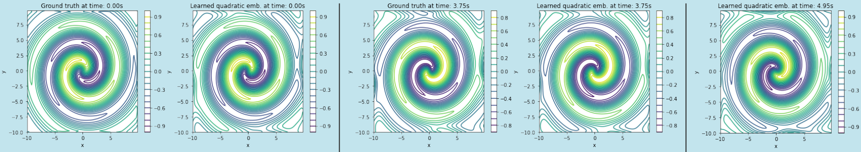
- We take 100×100 grid and collect 100 data points in time $t = [0, 5]$.
- We consider the first 75 points for training and the last 25 for testing.
- The data are high-dimensional $(2 \cdot 10^4)$ and exhibit an exponential decay of singular values, so we compress the data using projection onto the first two POD models.
- We learn a quadratic model of $\dim = 2$ using the projected data as input.

Lambda–Omega reaction–diffusion example

- The governing equations are

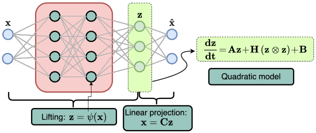
$$
\begin{aligned}
u_t &= (1-(u^2+v^2))u + \beta(u^2+v^2)v + d_1(u_{xx}+u_{yy}),\\
v_t &= -\beta(u^2+v^2)u + (1-(u^2+v^2))v + d_2(v_{xx}+v_{yy}).
\end{aligned}
$$

- We take 100×100 grid and collect 100 data points in time $t = [0, 5]$.
- We consider the first 75 points for training and the last 25 for testing.
- The data are high-dimensional $(2 \cdot 10^4)$ and exhibit an exponential decay of singular values, so we compress the data using projection onto the first two POD models.
- We learn a quadratic model of $\dim = 2$ using the projected data as input.



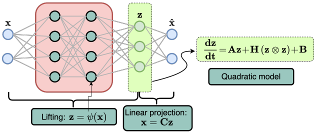
training | testing truth known | testing truth unknown

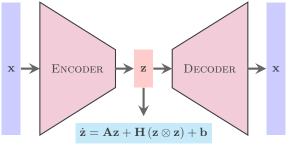
1. Recall that we have a linear projection from $\mathbf{z} \mapsto \mathbf{x}$. It works good only if we have a fast decay of singular values of our high-dimensional data
   - However, there is no suitable low-dimensional linear subspace for advection-dominant problems.
     - ⇝ Slow decay of Komologov n-width.
     - ⇝ Need a large-dimensional $\mathbf{z}$, meaning engineering studies can still be intractable.



1. Recall that we have a linear projection from $\mathbf{z} \mapsto \mathbf{x}$. It works good only if we have a fast decay of singular values of our high-dimensional data
   - However, there is no suitable low-dimensional linear subspace for advection-dominant problems.
     - $\leadsto$ Slow decay of Komologov n-width.
     - $\leadsto$ Need a large-dimensional $\mathbf{z}$, meaning engineering studies can still be intractable.

Remedy: Use a nonlinear decoder using neural networks (e.g., convolutional NNs for structured data)

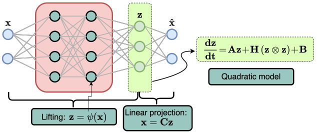


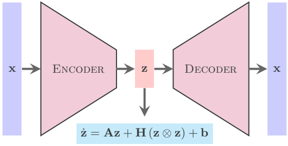


1. Recall that we have a linear projection from $\mathbf{z} \mapsto \mathbf{x}$. It works good only if we have a fast decay of singular values of our high-dimensional data
   - However, there is no suitable low-dimensional linear subspace for advection-dominant problems.
     - ⇝ Slow decay of Komologov n-width.
     - ⇝ Need a large-dimensional $\mathbf{z}$, meaning engineering studies can still be intractable.

> Remedy: Use a nonlinear decoder using neural networks (e.g., convolutional NNs for structured data)

2. Moreover, to train networks, we need to determine derivative of output w.r.t. inputs.
   - If $\dim(\mathbf{x})$ is large, then derivative computations using, e.g., `autograd` become computationally very expensive.





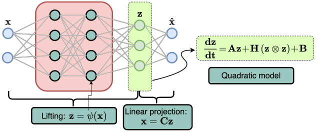
Challenges and Possible Solutions

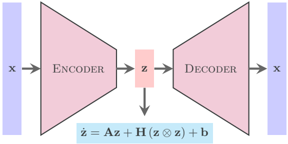
1. Recall that we have a linear projection from $\mathbf{z} \mapsto \mathbf{x}$. It works good only if we have a fast decay of singular values of our high-dimensional data
   - However, there is no suitable low-dimensional linear subspace for advection-dominant problems.
     - ⇝ Slow decay of Komologov n-width.
     - ⇝ Need a large-dimensional $\mathbf{z}$, meaning engineering studies can still be intractable.

Remedy: Use a nonlinear decoder using neural networks (e.g., convolutional NNs for structured data)

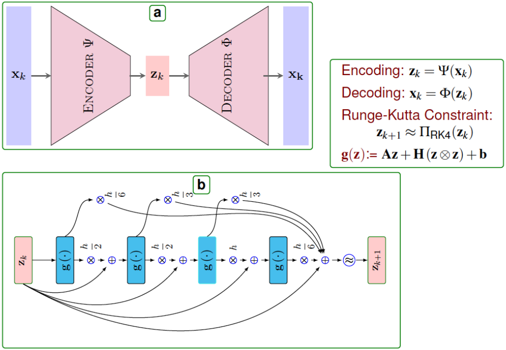
2. Moreover, to train networks, we need to determine derivative of output w.r.t. inputs.
   - If $\dim(\mathbf{x})$ is large, then derivative computations using, e.g., `autograd` become computationally very expensive.

Remedy: Embed a numerical integrator





- Combining all, we have



- We focus on a Runge-Kutta scheme, but any integrator including adaptive ones can be utilized using Neural ODEs. (Chen et al. 2018)
- Once such an architecture is framed, we can learn encoder, decoder, and a quadratic model.

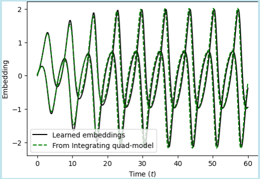
- One dimensional model with a single reaction, describing dynamics of the species concentration $\psi(x,t)$ and temperature $\theta(x,t)$ via

$$\frac{\partial \psi}{\partial t} = \frac{1}{\mathrm{Pe}}\frac{\partial^2 \psi}{\partial x^2} - \frac{\partial \psi}{\partial x} - \mathcal{D}\mathcal{F}(\psi,\theta;\gamma),$$
$$\frac{\partial \theta}{\partial t} = \frac{1}{\mathrm{Pe}}\frac{\partial^2 \theta}{\partial x^2} - \frac{\partial \theta}{\partial x} - \beta(\theta - \theta_{\mathsf{ref}}) + \mathcal{B}\mathcal{D}\mathcal{F}(\psi,\theta;\gamma),$$

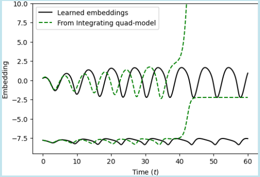
  with spatial variable $x \in (0,1)$, time $t > 0$ and Arrhenius reaction term

$$\mathcal{F}(\psi,\theta;\gamma) = \psi \exp\left(\gamma - \frac{\gamma}{\theta}\right).$$

- Collect snapshots in time $\mathbf{T} = [0, 10]$.
- We learn 2-dimensional embeddings using convolutional autoencoder.
- For comparison, we also compute a 2-dimensional model using POD projection (classical OpInf).

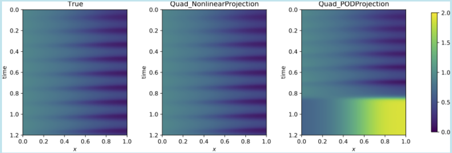


(a) convolutional autoencoder

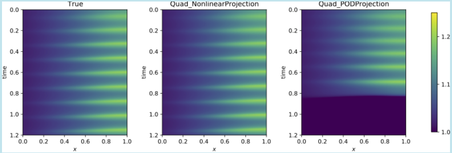


(b) POD

Figure: 2-dimensional embeddings.



(a) Concentration on the full-grid.



(b) Temperature on the full-grid.

Figure: Comparison of the convolutional autoencoders and POD-based approaches.

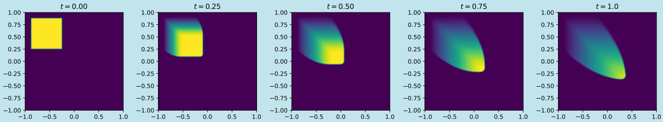
- Governing equation:

$$\frac{\partial u(x,t)}{\partial t} + \left(\frac{1}{2}, \frac{1}{2}\right)^\top \cdot \nabla u(x,t)^2 = 0 \quad \forall (x,t) \in \Omega \times [0,T].$$

- Governing equation:

$$\frac{\partial u(x,t)}{\partial t} + \left(\frac{1}{2}, \frac{1}{2}\right)^\top \cdot \nabla u(x,t)^2 = 0 \quad \forall (x,t) \in \Omega \times [0,T].$$

- We collect snapshots 100 snapshots in $[0,1]$ by taking 512 points in x and y directions $\rightsquigarrow$ full dimensional model with $262\,144$ DoFs.

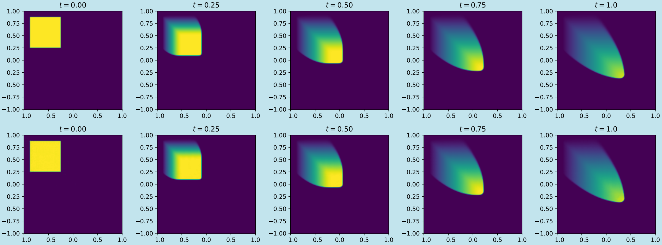


Numerical Example
2D Burgers equation

- Governing equation:

$$\frac{\partial u(x,t)}{\partial t} + \left(\frac{1}{2}, \frac{1}{2}\right)^\top \cdot \nabla u(x,t)^2 = 0 \quad \forall (x,t) \in \Omega \times [0,T].$$

- We collect snapshots 100 snapshots in $[0,1]$ by taking 512 points in x and y directions $\rightsquigarrow$ full dimensional model with $262\,144$ DoFs.
- We learn one-dimensional quadratic model, and encoder and decoder are convolutional neural networks. (Goyal/B. 2021)



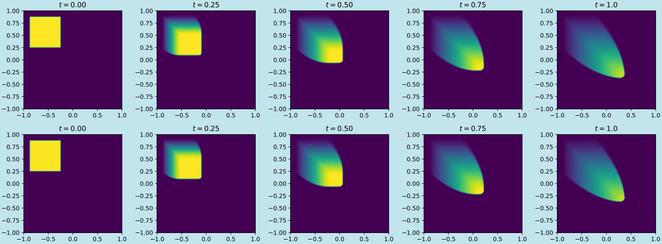
Numerical Example

2D Burgers equation

- Governing equation:

$$\frac{\partial u(x,t)}{\partial t} + \left(\frac{1}{2}, \frac{1}{2}\right)^\top \cdot \nabla u(x,t)^2 = 0 \quad \forall (x,t) \in \Omega \times [0,T].$$

- We collect snapshots 100 snapshots in $[0,1]$ by taking 512 points in x and y directions $\rightsquigarrow$ full dimensional model with $262\,144$ DoFs.
- We learn one-dimensional quadratic model, and encoder and decoder are convolutional neural networks. (Goyal/B. 2021)



- Note that for rich dynamics,we may need to increase the dimension of the quadratic embeddings.

Contributions

- Discussed lifting principle for nonlinear dynamical systems.

Contributions

- Discussed lifting principle for nonlinear dynamical systems.
- Lifting allows us to write nonlinear systems as quadratic systems using observables (or lifted variables).

  $\rightsquigarrow$ Notion of quadratic embeddings.

Contributions

- Discussed lifting principle for nonlinear dynamical systems.
- Lifting allows us to write nonlinear systems as quadratic systems using observables (or lifted variables).

  $\rightsquigarrow$ Notion of quadratic embeddings.
- To determine embeddings, we make use of neural networks (e.g., CNNs).

Contributions

- Discussed lifting principle for nonlinear dynamical systems.
- Lifting allows us to write nonlinear systems as quadratic systems using observables (or lifted variables).

  $\rightsquigarrow$ Notion of quadratic embeddings.
- To determine embeddings, we make use of neural networks (e.g., CNNs).
- For high-dimensional data with slow decay of singular values, we utilize nonlinear decoders, which allows identification of PDE models with slowly decaying Kolmogorov n-width.

Outlook

Contributions

- Discussed lifting principle for nonlinear dynamical systems.
- Lifting allows us to write nonlinear systems as quadratic systems using observables (or lifted variables).

  $\rightsquigarrow$ Notion of quadratic embeddings.
- To determine embeddings, we make use of neural networks (e.g., CNNs).
- For high-dimensional data with slow decay of singular values, we utilize nonlinear decoders, which allows identification of PDE models with slowly decaying Kolmogorov n-width.

Open work

- Extensions to Hamiltonian, parametric, and control systems.

Contributions

- Discussed lifting principle for nonlinear dynamical systems.
- Lifting allows us to write nonlinear systems as quadratic systems using observables (or lifted variables).

  $\rightsquigarrow$ Notion of quadratic embeddings.
- To determine embeddings, we make use of neural networks (e.g., CNNs).
- For high-dimensional data with slow decay of singular values, we utilize nonlinear decoders, which allows identification of PDE models with slowly decaying Kolmogorov n-width.

Open work

- Extensions to Hamiltonian, parametric, and control systems.
- Stability guarantees of the quadratic model for the embeddings?

Contributions

- Discussed lifting principle for nonlinear dynamical systems.
- Lifting allows us to write nonlinear systems as quadratic systems using observables (or lifted variables).

  $\rightsquigarrow$ Notion of quadratic embeddings.
- To determine embeddings, we make use of neural networks (e.g., CNNs).
- For high-dimensional data with slow decay of singular values, we utilize nonlinear decoders, which allows identification of PDE models with slowly decaying Kolmogorov n-width.

Open work

- Extensions to Hamiltonian, parametric, and control systems.
- Stability guarantees of the quadratic model for the embeddings?
- Work on real-engineering (reactor model) and investigate how to use more physics e.g., mass/energy conservation laws!

Outlook

Contributions

- Discussed lifting principle for nonlinear dynamical systems.
- Lifting allows us to write nonlinear systems as quadratic systems using observables (or lifted variables).

  $\rightsquigarrow$ Notion of quadratic embeddings.
- To determine embeddings, we make use of neural networks (e.g., CNNs).
- For high-dimensional data with slow decay of singular values, we utilize nonlinear decoders, which allows identification of PDE models with slowly decaying Kolmogorov n-width.

Open work

- Extensions to Hamiltonian, parametric, and control systems.
- Stability guarantees of the quadratic model for the embeddings?
- Work on real engineering (reactor model) and investigate how to use more physics e.g., mass/energy conservation laws!

Thank you for your attention!!

Selected References (Alphabetical)

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018).
Neural ordinary differential equations.
In *Advances Neural Inform. Processing Sys.*, pages 6571–6583.

Folkestad, C., Pastor, D., Mezic, I., Mohr, R., Fonoberova, M., and Burdick, J. (2020).
Extended dynamic mode decomposition with learned Koopman eigenfunctions for prediction and control.
In *American Control Conference (ACC)*, pages 3906–3913. IEEE.

Goyal, P. and Benner, P. (2021).
Learning low-dimensional quadratic-embeddings of high-fidelity nonlinear dynamics using deep learning.
e-print 2111.12995, arXiv.

Gu, C. (2011).
QLMOR: A projection-based nonlinear model order reduction approach using quadratic-linear representation of nonlinear systems.
IEEE Trans. Comput. Aided Des. Integr. Circuits. Syst., 30(9):1307–1320.

Koopman, B. O. (1931).
Hamiltonian systems and transformation in Hilbert space.
Proc. Nat. Acad. Sci. U.S.A., 17(5):315.

Lusch, B., Kutz, J. N., and Brunton, S. L. (2018).
Deep learning for universal linear embeddings of nonlinear dynamics.
Nature Commu., 9(1):1–10.

Qian, E., Kramer, B., Peherstorfer, B., and Willcox, K. (2020).
Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems.
Physica D: Nonlinear Phenomena, 406:132401.

Williams, M. O., Kevrekidis, I. G., and Rowley, C. W. (2015).
A data–driven approximation of the Koopman operator: Extending dynamic mode decomposition.
J. Nonlinear Science, 25(6):1307–1346.