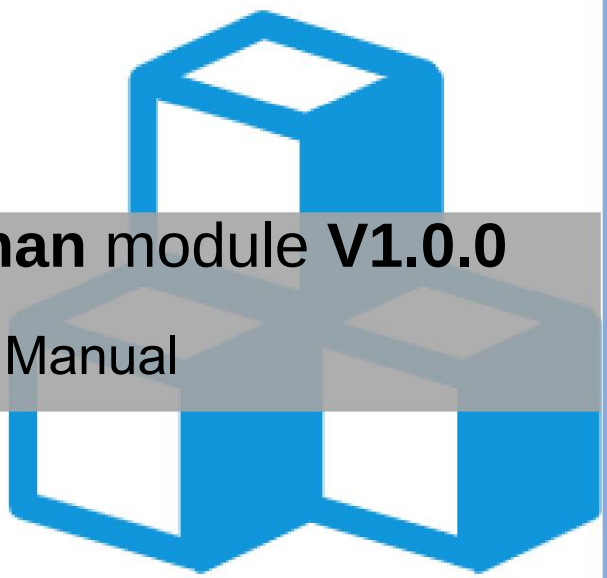
MODELING

DeepKoopman module **V1.0.0**

User Manual



Ideal

Table of contents

Chapter 1 Installation of **DeepKoopman**

1. Install the required software and hardware environment

- DeepKoopman hardware requirements

  Hard disk: 3G or more available space;

  Memory: 512M or more;

  CPU: 1.60GHz or above.

- DeepKoopman software requirements

  Python 3.8.3 or above

  - Python third-party library requirements

    Torch 1.7.0+cpu

    Numpy 1.23.0

    Scipy 1.4.1

    Matplotlib 3.2.3

    Argparse 1.1

    Control 0.8.3

  - It is recommended to use Anaconda (https://www.anaconda.com/distributionor) for installation

2. Installation of DeepKoopman

Run the following command to install the DeepKoopman module, as shown in Figure 1.1:

```
pip install deepKoopman
```

Figure 1.1 Default installation path for installing **DeepKoopman**

Required third-party modules include numpy, pytorch, scipy, control, etc., as shown in Figure 1.2:

```
import numpy as np
import torch
import torch.nn as nn
import control
import os
import argparse
import sys
import scipy
from scipy.integrate import odeint
import matplotlib.pyplot as plt
```

Figure 1.2 Installing **DeepKoopman** 's third-party module

After DeepKoopman is installed, run the following program to confirm that the module can be imported, as shown in Figure 1.3:

```
import deepKoopman

from deepKoopman import DeepKoopman
```

Figure 1.3 **DeepKoopman** installation completion test

Chapter 2 Basic Application of **DeepKoopman**

This chapter mainly introduces the basic applications of the DeepKoopman library, including functional description, framework structure, functional expansion, complete examples, functions Number description, etc. Through the introduction in this chapter, users can quickly become familiar with the functions and usage of the DeepKoopman library.

1. Function description

DeepKoopman is a system identification tool based on deep learning methods that can automatically optimize all hyperparameters according to optimization goals. An accurate system model with a global linear representation is thereby identified. DeepKoopman is a python library implemented under the Pytorch framework. It uses an algorithm based on Koopman operator theory and centers on deep learning methods, providing a weight representation of deep neural network The observation function and system matrix are further used to obtain the global linear model.

DeepKoopman supports:

- Discrete time and continuous time system identification
    - Discrete time systems and continuous time systems can be converted into each other through numerical differentiation methods.
- System identification with control input
    - For the identified system, the mature linear control method can be used to complete the control of the original nonlinear system.
- Controller based on linear quadratic programming control method (LQR)
    - The system matrix can be obtained for the identified system, thereby calculating the control gain matrix and realizing the original nonlinear system.

System closed-loop control;

- For the system that has completed the identification, the system matrix and observation function expression can be obtained. On this basis, further introduction can be Model predictive control and other control methods.

- Automatically optimize all network parameters to obtain an accurate Koopman linearization model
  - Use the optimizer to iteratively update network parameters according to the optimization goals;
  - Provide a large amount of training data and test data through automatic data updates, and continuously enrich training samples to improve network training efficiency. results, thereby improving prediction accuracy.

- Add custom modules.
  - Users can implement any class to extend DeepKoopman (for example, custom observable objects, custom controllers, new system estimator).

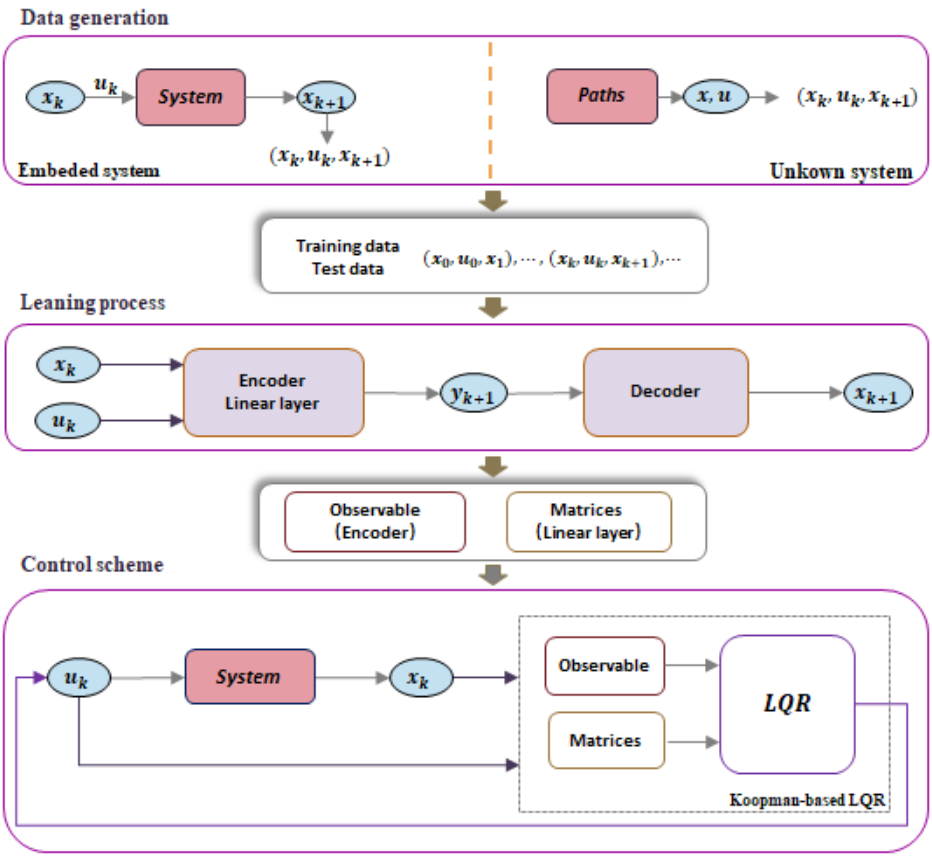


Figure 2.1 Illustration of the principle framework of the **DeepKoopman** module, where and represents the system state and control input, and is the state and control in discrete form Input, is the number of discrete time steps, System represents the known system, and Paths represents the data saving path when the system mathematical model is unknown.

2. Frame structure

The library architecture adopts a modular design, allowing users to implement model identification of commonly used known system models and systems of unknown system models. Identification, for situations where the system model is known, the module can automatically generate training data and test data; for situations where the system model is unknown, Users only need to provide observation data. Furthermore, for the system that has completed the identification, the LQR controller can be used to complete the original nonlinear The closed-loop control of the system is shown in Figure 2.1.

3. Application scenarios

This library is intended for systems engineers/researchers who wish to leverage data-driven dynamic systems techniques. Users can data-driven modeling of their systems.

- System dynamics prediction. Users can simulate models learned from their measurements to predict the evolution of systems over long periods of time, and implement the provided analytical basis (e.g. control input calculations, visualization).
- System analysis and control. Users can obtain linear representations of their systems in their original state or Koopman observables. They can Use this linear form to perform tasks such as controller synthesis and system reachability. synthesize control signals to achieve the desired Closed-loop behavior and optimal with respect to some goals.
- Verification. Demonstrate the security requirements of the system.

4. Complete example

The DeepKoopman module can directly input parameters to build a model. You can give the system name of the system that the module already contains or give In the case of fixed trajectory training data (array list, .npy format file), the dynamic system can be learned from the data in one call, as shown in the figure 2.2-As shown in Figure 2.5:

DeepKoopman provides the following variable parameters to adjust the network framework of the network model, so that it can continuously optimize the network hyperparameters to Obtain the optimal network model and further obtain the optimal global linear model to complete system model prediction and control.

Table 2.1 Parameter representation and default values

Parameter name	representation	default value
model_name	system name	vdp (optional: duffing; toy; pendulum; unknown_system)
max_iterMaximum	number of iterations	50 (the data will be automatically updated once an iteration is completed)

epoch	number of optimizations in each iteration	50
hidden_dim	ascending state dimension	8
stable_dim	network hidden layer depth	64
batch_size	batch training data size	128
nx	Original system state dimension	2
nu	Control input dimensions	1
time	The sampling period	50
steps	Sampling step size	0.01
ntraj	Number of sampling trajectories	200
mode	Train or load a saved network model	train

```
"""
====###Case1###==== default systems: vdp;duffing;pendulum;toy;robot
"""
"""
build NNKOOPMAN model for system with equations
parameters:
                model_name: systems:vdp;duffing;toy;pendulum;
                max_iter: max iterations
                epoch: training epoch in one iteration
                hidden_dim: dimension of the lifted state
                stable_dim: size of hidden layers
                batch_size: batch size
                nx: dimension of the original state
                nu: dimension of the control input
                time: sampling period
                steps: sampling interval
                ntaj: number of trajectories
                mode: training or load model
"""
parser = argparse.ArgumentParser()
parser.add_argument("--model_name", default='vdp')
parser.add_argument("--max_iter", default=50)
parser.add_argument("--epoch", default=50, type=int)
parser.add_argument("--hidden_dim", default=8, type=int)
parser.add_argument("--stable_dim", default=64, type=int)
parser.add_argument("--batch_size", default=128, type=int)
parser.add_argument("--nx", default=2, type=int)
parser.add_argument("--nu", default=1, type=int)
parser.add_argument("--time", default=50, type=int)
parser.add_argument("--steps", default=0.01)
parser.add_argument("--ntaj", default=200)
parser.add_argument("--mode", default="train")
# parser.add_argument("--mode", '-false')
args = parser.parse_args()
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
```

Figure 2.2 Example of using system dynamics prediction - model establishment (system model is known)

```
"""
train NNKOOPMAN model for system with equations
save model weights;
load model weights;
build LQR controller;
"""
if args.mode== "train":
      model.train(args.max_iter, args.epoch,0.001)
      model.save()
else:
      model.load()
      sat_true,sat_pre, error1, error2=model.pre()
      model.pre_plot(sat_true,sat_pre, error1, error2)
      ##################LQR
      """
     LQR control:
           K gain matrix; xx closed-system states
     """
      # A, B = model.get_system()
      Q = np.eye(args.hidden_dim)
      R = np.array([[0.1]])
      # K, _, _ = control.lqr(A, B, Q, R)
      ref=[0.0, 0.0]
      x_0=[0.5,-0.5]
      nsim=200
      K,xx,uu=model.policy_rollout(Q, R, ref, x_0, nsim)
      model.policy_plot(xx,uu)
```

Figure 2.3 Examples of system dynamics prediction usage - model training, model storage, model loading, system model prediction and system control (system model is known)

```
"""
====###Case2###==== unknown dynamics, please provide training data and test data
"""
"""
build NNKOOPMAN model for system with equations
model name:
      unknown system
provide data paths:
      path1 state_trainning_data
      path2 controlinput_training_data
      path3 state_test_data
      path4 controlinput_test_data
"""
parser = argparse.ArgumentParser()
parser.add_argument("--model_name", default='unknown_system')
parser.add_argument("--max_iter", default=50)
parser.add_argument("--epoch", default=50)
parser.add_argument("--hidden_dim", default=8, type=int)
parser.add_argument("--stable_dim", default=64, type=int)
parser.add_argument("--batch_size", default=128, type=int)
parser.add_argument("--nx", default=2, type=int)
parser.add_argument("--nu", default=1, type=int)
parser.add_argument("--time", default=50, type=int)
parser.add_argument("--steps", default=0.01)
parser.add_argument("--ntaj", default=200)
parser.add_argument("--mode", default="train")
# parser.add_argument("--mode", '-false')
args = parser.parse_args()
path1='path_for_state_data'
path2='path_for_controlinput_data'
path3='path_for_state_data'
path4='path_for_controlinput_data'
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
```

Figure 2.4 Example of using system dynamics prediction - model establishment (system model is unknown)

```
"""
train DEEPKOOPMAN model for unknown systen with measurements
save model weights;
load model weights;
build LQR controller;
"""
if args.mode == "train":
        model.train(args.max_iter, 0.001, path1,path2, path3,path4,)
        model.save()
else:
        model.load()
        sat_true,sat_pre, error1, error2=model.pre()
        model.pre_plot(sat_true,sat_pre, error1, error2)
        model.save_weights()
        ###################LQR
        A, B = model.get_system()
        Q = np.eye(args.hidden_dim)
        R = np.array([[0.1]])
        K, _, _ = control.lqr(A, B, Q, R)
```

Figure 2.5 Examples of system dynamics prediction usage - model training, model storage, model loading, system model prediction and system control (system model is unknown)

5. Function description

The DeepKoopman module includes network functions, system functions, training functions, model storage and loading, etc. It is simple and convenient to use. It can be quickly trained to obtain the prediction model of the system, and can further output the prediction results and control results of the system dynamics, and Corresponding illustrations can be given.

Deepkoopman.encoder

Item description
Representation: ascending dimension function **Function:** Upgrade the nonlinear state into a high-dimensional linear space **Input:** nonlinear system state **Output:** Upgraded dimension state, that is, the system state in the linear system based on koopman.

- Example

```
model = DeepKoopman(args.nx,args.nu,args.model_name, args.hidden_dim)
gt = DeepKoopman.encoder(xt)
```

Input the current state xt to the encoder to get the dimensionally enhanced state gt.

DeepKoopman.decoder

Item description
Represents: reconstruction function **Function:** Map the upgraded system state to the original nonlinear space **Input:** system status after dimensionality upgrade **Output:** original nonlinear state

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
xt_ = DeepKoopman.decoder(gt)
```

Input the elevated dimension state gt to the decoder and obtain the original state estimate xt_.

DeepKoopman.vdp

Item description
Represents: van der pol system **Function:** Obtain the motion trajectory of the van der pol system under a given initial state and a given control input trace **Input:** initial state, control input **Output:** Display the system status value of the system motion trajectory

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
obs=DeepKoopman.vdp(obs_old, dt, action)
```

Among them, obs_old is the state of the previous moment, dt is the sampling step size, action is the control input, and the next moment of the van der Pol system is obtained.

Status obs.

- Reference

[1] S. Sinha, SP Nandanoori, J. Drgona, and D. Vrabie, “Data-driven stabilization of discrete-time control-affine nonlinear systems: A Koopman operator approach,” in 2022 European Control Conference (ECC), 2022 , pp. 552–559.

[2] S. Daniel-Berhe and H. Unbehauen, “Experimental physical parameter estimation of a thyristor driven DC-motor using the HMF-method,” Control Engineering Practice, vol. 6, no. 5, pp. 615–626, 1998.

DeepKoopman.duffing

Item description
Represents: duffing system **Function:** Obtain a given initial state and the motion trajectory of the duffing system under a given control input **Input:** initial state, control input **Output:** Display the system status value of the system motion trajectory

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name,  args.hidden_dim)
obs=DeepKoopman.duffing(obs_old, dt, action)
```

Among them, obs_old is the state of the previous moment, dt is the sampling step size, action is the control input, and the next moment state of the duffing system is obtained.

obs.

- Reference

[3] S. Klus, F. Nuske, S. Peitz, JH Niemann, and C. Schutte, “Data-driven approximation of the Koopman generator: Model reduction, system identification, and control,” Physica D: Nonlinear Phenomena, vol. 406, p. 132416, 2020.

[4] N. Takeishi, Y. Kawahara, and T. Yairi, “Learning Koopman invariant subspaces for dynamic mode decomposition,” in Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 1130–1140.

DeepKoopman.toy

Item description
Represents: toy system **Function:** Obtain a given initial state and the motion trajectory of the toy system under a given control input **Input:** initial state, control input **Output:** Display the system status value of the system motion trajectory

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name,  args.hidden_dim)
obs=DeepKoopman.toy(obs_old, dt, action)
```

Among them, obs_old is the state of the previous moment, dt is the sampling step size, action is the control input, and the next moment state obs of the toy system is obtained.

- Reference

[5] B. Lusch, J. Kutz, and S. Brunton, “Deep learning for universal linear embeddings of nonlinear dynamics,” Nature Communications, vol. 9, p. 4950, 2018.

[6] SL Brunton, BW Brunton, JL Proctor, E. Kaiser, and JN Kutz, “Chaos as an intermittent forced linear system,” Nature Communications, vol. 8, no. 1, p. 19, 2017.

DeepKoopman.pendulum

Item description
Represents: pendulum system
Function: Obtain the motion trajectory of the pendulum system under a given initial state and a given control input
Input: initial state, control input
Output: Display the system status value of the system motion trajectory

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
obs=DeepKoopman.pendulum(obs_old, dt, action)
```

Among them, obs_old is the state of the previous moment, dt is the sampling step size, action is the control input, and the next moment state of the pendulum system is obtained.

obs.

- Model reference

[7] Ch. 2 - The Simple Pendulum (mit.edu)

DeepKoopman.save

Item description
Representation: network model parameter storage
Function: Store network encoder and decoder parameters
Input: None
Output: .pt format file storing network parameters

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
model.train(args.max_iter, args.epoch, 0.001)
model.save()
```

Among them, max_iter is the maximum number of iterations, epoch is the number of updates for each iteration, and 0.001 is the learning rate.

DeepKoopman.load

Item description
Representation: Download network model parameters **Function:** Export network encoder and decoder parameters **Input:** None **Output:** Export encoder decoder with fixed network parameters that can be used for further linear systems system construction for model approximation

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
model.load()
```

DeepKoopman.get_system

Item description
Representation: Linear system system matrix acquisition **Function:** Obtain AB matrix **Input:** None **Output:** matrix AB

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
A, B = model.get_system()
```

DeepKoopman.random_rollout

Item description
Representation: data generation **Function:** Generate network training data and network test data **Input:** data size, including sampling period, sampling step size, and number of initial states **Output:** training data and test data

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
x_train, u_train=self.random_rollout()
x_test, u_test=self.random_rollout()
```

DeepKoopman.sample

Item description
Represents: random sampling **Function:** Randomly sample training data and test data, The sample size is given by batchsize size **Input:** batchsize value **Output:** training data and test data of batchsize size are used for network model training and network model type test

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
xt, ut, xt1 = model.sample(batch_size, x_train, u_train)
```

DeepKoopman.get_data

Item description
Represents: data acquisition **Function:** Obtain system data provided by the user. When the system is an unknown system that is not included in the module, you can Network model training is performed through system data provided by users. **Input:** data storage path **Output:** System data includes system status data, corresponding control input data, etc.

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
x_train, u_train=self.get_data(path1, path2)
x_test, u_test=self.get_data(path3, path4)
```

DeepKoopman.train

Item description

Represents: network training

Function: Use the training data and the optimizer to iterate the network model parameters based on the loss function minimization.

generation update

Input: number of iterations, learning rate, when the system mode is unknown, additional input data storage is required

path

Output: network model

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
if args.model_name== "unknown_system":
      model.train(args.max_iter, 0.001, path1, path2, path3, path4,)
else:
      model.train(args.max_iter, args.epoch, 0.001)
```

DeepKoopman.forward

Item description
Represents: network forward propagation function
Function: Get the output representation of each network module (encoder and decoder)
Input: network input
Output: the output of each network module

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
gt, gt1, gt1_, xt_, xt1_ = model.forward(xt, ut, xt1)
```

DeepKoopman.scale_loss

Item description
Representation: scale loss function
Function: Calculate scale loss, used to construct total loss
Input: network output
Output: scale loss

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
loss = model.scale_los(x, y)
```

DeepKoopman.controllability

Item description
Represents: system controllability criterion **Function:** Calculate the controllability matrix of a linear system based on the Koopman operator and obtain an ascending-dimensional linear system system controllability **Input:** AB matrix **Output:** system controllability

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
rank = model.controllability(A, B)
```

DeepKoopman.pre

Item description
Represents: system status prediction **Function:** Predict the motion trajectory of the original nonlinear state through the trained network model **Input:** initial state, prediction step size **Output:** system predicted state, true state, prediction error

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
sat_true, sat_pre, error1, error2=model.pre(obs_old0=[-0.5, 0.5], kk=1000)
```

DeepKoopman.pre_plot

Item description
Representation: system state prediction visualization **Function:** Output system real state prediction state comparison chart

Input: real state, predicted state **Output:** Comparison of predicted and actual trajectories in all dimensions of the state

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
model.pre_plot(sat_true, sat_pre, error1, error2)
```

DeepKoopman.total_loss_fn

Item description
Representation: total loss function **Function:** Calculate the total loss of the neural network **Input:** input and output of each network module **Output:** Comparison of predicted and actual trajectories in all dimensions of the state

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
total_loss=model.total_loss_fn( xt, ut, xt1,gt, gt1, gt1_,xt_, xt1_)
```

DeepKoopman.policy_rollout

Item description
Represents: LQR control calculation **Function:** Calculate control input according to LQR control rules **Input:** Q, R matrix, reference point, initial state, control step size **Output:** LQR control input

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
K,xx,uu=model.policy_rollout(Q, R, ref, x_0, nsim)
```

DeepKoopman.policy_plot

Item description
Representation: visualization of control inputs and corresponding status values
Function: Display LQR control input value and status value under control
Input: LQR control input value, status value under control
Output: LQR control input and corresponding status value icon

- Example

```
model = DeepKoopman(args.nx, args.nu, args.model_name, args.hidden_dim)
K, xx, uu=model.policy_rollout(Q, R, ref, x_0, nsim)
model.policy_plot(xx, uu)
```

Overall reference

[8] SL Brunton, BW Brunton, JL Proctor, KJ Nathan, and HAKestler, “Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control,” Plos One, vol. 11, no. 2, p. e0150171, 2016.

[9] PJ Schmid and J. Sesterhenn, “Dynamic mode decomposition of numerical and experimental data,” Journal of Fluid Mechanics, vol. 656, no. 10, pp. 5–28, 2010.

[10] M. Korda and I. Mezic, “Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control,” Automatica, vol. 93, pp. 149–160, 2016.

[11] JL Proctor, SL Brunton, and JN Kutz, “Generalizing Koopman theory to allow for inputs and control,” SIAM Journal on Applied Dynamical Systems, vol. 17, p. 909–930, 2016.