MODELING

DeepKoopman 模块 V1.0.0

使用手册

Ideal

目录

第一章 DeepKoopman 的安装

一、 安装所需的软硬件环境

- DeepKoopman 硬件要求

  硬盘：3G 以上可用空间；

  内存：512M 以上；

  CPU：1.60GHz 或以上。

- DeepKoopman 软件要求

  Python 3.8.3 或以上

  - Python 第三方库要求

    Torch 1.7.0+cpu

    Numpy 1.23.0

    Scipy 1.4.1

    Matplotlib 3.2.3

    Argparse 1.1

    Control 0.8.3

  - 推荐使用 Anaconda（https://www.anaconda.com/distributionor）进行安装

二、 DeepKoopman 的安装

运行以下命令安装 DeepKoopman 模块，如图 1.1 所示：

```
pip install deepKoopman
```

图 1.1 安装 DeepKoopman 的缺省安装路径

需要的第三方模块包括 numpy、pytorch、scipy、control 等，如图 1.2 所示：

```
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
```

图 1.2 安装 DeepKoopman 的第三方模块

DeepKoopman 安装完成后，运行以下程序确定模块可以导入，如图 1.3 所示：

```
import deepKoopman

from deepKoopman import DeepKoopman
```

图 1.3 DeepKoopman 的安装完成测试

第二章 DeepKoopman 的基本应用

本章主要介绍 DeepKoopman 库的基本应用，包括功能描述、框架结构、功能拓展、完整示例、函数说明等。通过本章的介绍，用户可以很快的熟悉 DeepKoopman 库的功能及其使用。

一、 功能描述

DeepKoopman 是一个基于深度学习方法的系统识别工具，可以根据优化目标自动优化所有超参数，从而辨识出具有全局线性表示的准确系统模型。DeepKoopman 是在 Pytorch 框架下实现的 python 库，它使用了基于 Koopman 算子理论的算法，并以深度学习方法为中心，提供了由深度神经网络权值表示的观测函数和系统矩阵，进一步得到全局线性模型。

DeepKoopman 支持：

- 离散时间和连续时间系统辨识
  - 离散时间系统与连续时间系统可以通过数值微分方法相互转换。
- 带控制输入的系统辨识
  - 针对辨识得到的系统可以使用发展成熟的线性控制方法完成原始非线性系统的控制。
- 基于线性二次规划控制方法的控制器（LQR）
  - 针对完成辨识的系统可获得系统矩阵，从而计算出控制增益矩阵，实现原始非线性系

统的闭环控制；

- 针对完成辨识的系统可获得系统矩阵，观测函数表达，在此基础之上可以进一步引入模型预测控制等其他控制方法。

- 自动优化所有网络参数，从而获取准确的 Koopman 线性化模型
  - 根据优化目标利用优化器进行网络参数的迭代更新；
  - 通过数据自动更新提供大量训练数据与测试数据，不断丰富训练样本以提升网络训练效果，进而提高预测准确度。

- 添加自定义模块。
  - 用户可以实现任何类来扩展 DeepKoopman(例如，自定义可观察对象，自定义控制器，新的系统估计器)。

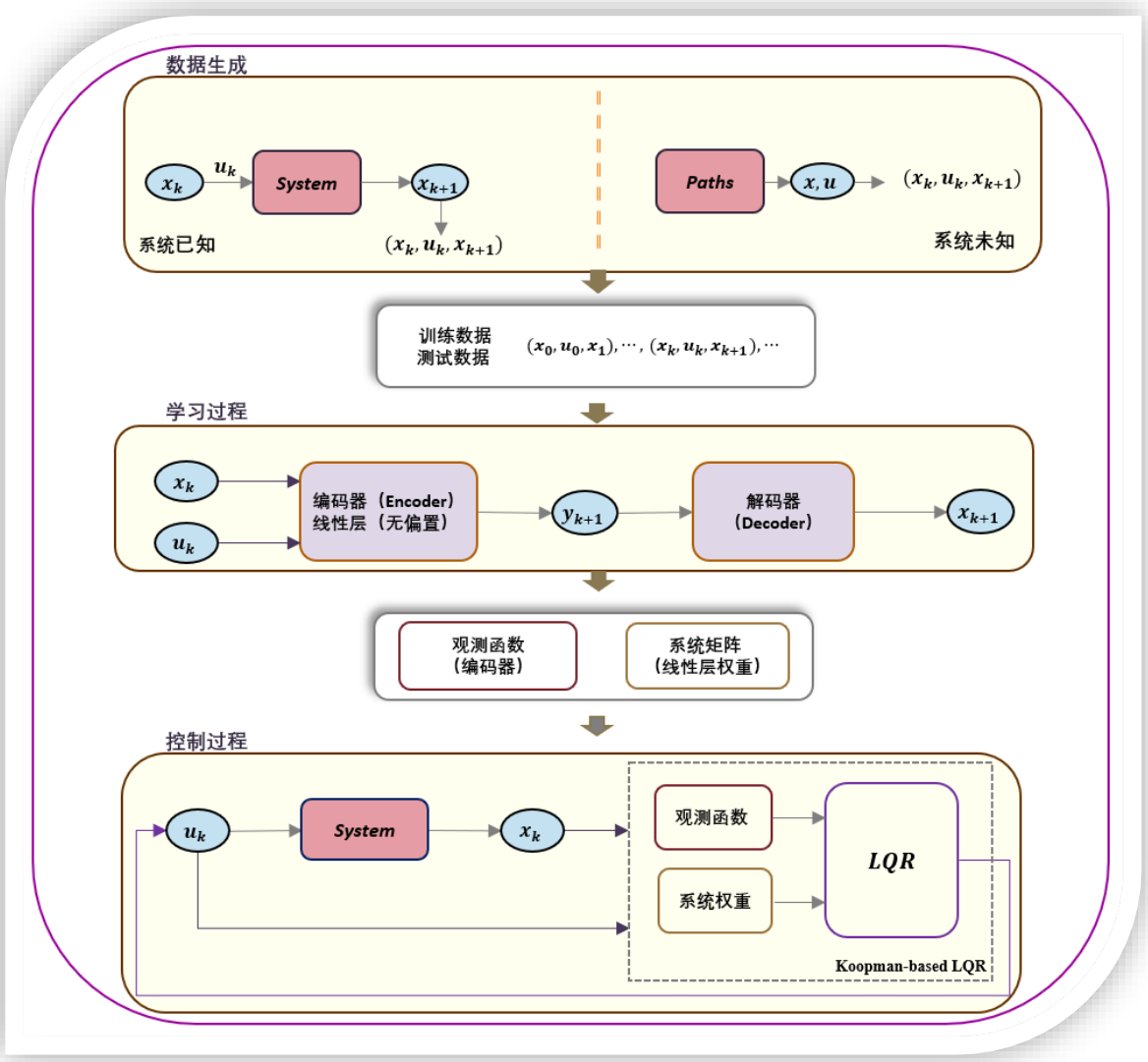

图 2.1 DeepKoopman 模块的原理框架图示，其中x和u表示系统状态和控制输入，x_k和u_k为离散形式的状态和控制输入，k为离散时间步数，System 代表系统已知，Paths 代表系统数学模型未知情况下的数据保存路径。

二、 框架结构

该库架构采用模块化设计，允许用户实现常用的已知系统模型的模型辨识与未知系统模型的系统辨识，对于系统模型已知的情况，模块可以自动生成训练数据与测试数据；对于系统模型未知的情况，用户只需要提供观测数据即可。进一步，针对完成辨识的系统可采用 LQR 控制器完成针对原始非线性系统的闭环控制，如图 2.1 所示。

三、 应用场景

该库适用于希望利用数据驱动的动态系统技术的系统工程师/研究人员。用户可以在没有预先模型的情况下对他们的系统进行数据驱动建模。

- 系统动力学预测。用户可以模拟从他们的测量中学习到的模型预测系统在很长一段时间内的演化，并实现所提供的分析基础 (例如控制输入计算，可视化)。
- 系统分析与控制。用户可以得到其系统在其原始状态或 Koopman 可观测值的线性表示。他们可以使用这种线性形式来执行诸如控制器综合和系统可达性等任务。合成控制信号，以实现期望的闭环行为，并在某些目标方面是最优的。
- 验证。证明系统的安全需求。

四、 完整示例

DeepKoopman 模块可以直接输入参数建立模型，可以在给出模块已经包含系统的系统名称或者给定轨迹训练数据(数组列表，.npy 格式文件)的情况下，一次调用就可以从数据中学习动态系统，如图 2.2-图 2.5 所示：

DeepKoopman 提供以下可变参数来调整网络模型的网络框架，从而能够不断优化网络超参数，以得到最优的网络模型，进一步得到最佳的全局线性模型完成系统模型预测与控制。

表 2.1 参数表示及默认值

参数名称	表示	默认值
model_name	系统名称	vdp(可选：duffing;toy;pendulum;unknown_system)
max_iter	最大迭代次数	50（完成一次迭代会自动更新一次数据）

epoch	每次迭代中的优化次数	50
hidden_dim	升维状态维数	8
stable_dim	网络隐藏层深度	64
batch_size	批量训练数据大小	128
nx	原系统状态维数	2
nu	控制输入维数	1
time	采样周期	50
steps	采样步长	0.01
ntraj	采样轨迹数目	200
mode	训练或者加载保存的网络模型	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)
```

图 2.2 系统动力学预测使用示例-模型建立（系统模型已知）

```
"""
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)
```

图 2.3 系统动力学预测使用示例-模型训练、模型存储、模型加载、系统模型预测和系统控制（系统模型已知）

```
"""
====###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)
```

图 2.4 系统动力学预测使用示例-模型建立（系统模型未知）

```
"""
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)
```

图 2.5 系统动力学预测使用示例-模型训练、模型存储、模型加载、系统模型预测和系统控制（系统模型未知）

五、 函数说明

DeepKoopman 模块包含网络函数、系统函数、训练函数、模型存储加载等多个函数，使用简单便捷，可以快速训练，得到系统的预测模型，进一步可以输出系统动力学的预测结果以及控制结果，并可以给出相应图示。

Deepkoopman.encoder

条目	描述
表示：	升维函数
作用：	将非线性状态升维到高维线性空间中
输入：	非线性系统状态
输出：	升维状态，即基于 koopman 的线性系统中的系统状态。

- Example

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

输入当前状态 xt 到 encoder,得到升维状态 gt。

DeepKoopman.decoder

条目	描述
表示：	重构函数
作用：	将升维后的系统状态映射到原来的非线性空间
输入：	升维后的系统状态
输出：	原始非线性状态

- Example

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

输入升维状态 gt 到 decoder，得到原始状态估计 xt_。

DeepKoopman.vdp

条目	描述
表示：	van der pol 系统
作用：	得到给定初始状态，给定控制输入下的 van der pol 系统的运动轨迹
输入：	初始状态，控制输入
输出：	显示系统运动轨迹的系统状态值

- Example

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

其中 obs_old 为前一刻状态，dt 为采样步长，action 为控制输入，得到 van der Pol 系统下一时刻状态 obs。

- Reference

[1] S. Sinha, S. P. 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

条目	描述
表示：	duffing 系统
作用：	得到给定初始状态，给定控制输入下的 duffing 系统的运动轨迹
输入：	初始状态，控制输入
输出：	显示系统运动轨迹的系统状态值

- Example

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

其中 obs_old 为前一刻状态，dt 为采样步长，action 为控制输入，得到 duffing 系统下一时刻状态 obs。

- Reference

[3] S. Klus, F. Nuske, S. Peitz, J. H. 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

条目	描述
表示：	toy 系统
作用：	得到给定初始状态，给定控制输入下的 toy 系统的运动轨迹
输入：	初始状态，控制输入
输出：	显示系统运动轨迹的系统状态值

- Example

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

其中 obs_old 为前一刻状态，dt 为采样步长，action 为控制输入，得到 toy 系统下一时刻状态 obs。

- 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] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz, “Chaos as an intermittently forced linear system,” Nature Communications, vol. 8, no. 1, p. 19, 2017.

DeepKoopman.pendulum

条目	描述
表示：	pendulum 系统
作用：	得到给定初始状态，给定控制输入下的 pendulum 系统的运动轨迹
输入：	初始状态，控制输入
输出：	显示系统运动轨迹的系统状态值

- Example

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

其中 obs_old 为前一刻状态，dt 为采样步长，action 为控制输入，得到 pendulum 系统下一时刻状态 obs。

- **模型参考**

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

DeepKoopman.save

条目	描述
表示：	网络模型参数储存
作用：	存储网络 encoder、decoder 参数
输入：	无
输出：	存储网络参数的.pt 格式文件

- Example

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

其中 max_iter 为最大迭代次数，epoch 为每次迭代式的更新次数，0.001 为学习率。

DeepKoopman.load

条目	描述
表示：	网络模型参数下载
作用：	导出网络 encoder、decoder 参数
输入：	无
输出：	导出具有固定网络参数的 encoder decoder 可用于进一步的线性系统构造用于模型近似

- Example

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

DeepKoopman.get_system

条目	描述
表示：	线性系统系统矩阵获取
作用：	获得 A B 矩阵
输入：	无
输出：	矩阵 A B

- Example

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

DeepKoopman.random_rollout

条目	描述
表示：	数据生成
作用：	生成网络训练数据与网络测试数据
输入：	数据尺寸，包括采样周期，采样步长，初始状态数目
输出：	训练数据与测试数据

- 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

条目	描述
表示：	随机采样
作用：	对训练数据和测试数据进行随机采样，采样尺寸为给定的 batchsize 大小
输入：	batchsize 值
输出：	batchsize 大小的训练数据与测试数据用于网络模型训练与网络模型测试

- 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

条目	描述
表示：	数据获取
作用：	获取用户提供的系统数据。当系统为模块不包含的未知系统时，可通过用户提供的系统数据来进行网络模型训练。
输入：	数据存储路径
输出：	系统数据包括系统状态数据，对应控制输入数据等。

- 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

条目	描述

表示：	网络训练
作用：	利用训练数据与优化器根据损失函数最小化进行网络模型参数的迭代更新
输入：	迭代次数，学习率，当系统模式为未知时，需要额外输入数据存储路径
输出：	网络模型

- 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

条目	描述
表示：	网络前向传播函数
作用：	获取各个网络模块（编码器和解码器）的输出表示
输入：	网络输入
输出：	各个网络模块的输出

- 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

条目	描述
表示：	尺度损失函数
作用：	计算尺度损失，用于构造总损失
输入：	网络输出
输出：	尺度损失

- Example

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

DeepKoopman.controllability

条目	描述
表示：	系统能控性判据
作用：	计算基于 Koopman 算子的线性系统的能控性矩阵，获得升维线性系统的能控性
输入：	A B 矩阵
输出：	系统能控性

- Example

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

DeepKoopman.pre

条目	描述
表示：	系统状态预测
作用：	通过训练完成的网络模型预测原始非线性状态的运动轨迹
输入：	初始状态，预测步长
输出：	系统预测状态，真实状态，预测误差

- 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

条目	描述
表示：	系统状态预测可视化
作用：	输出系统真实状态预测状态对比图

输入：	真实状态，预测状态
输出：	状态所有维度的预测和实际轨迹对比图

- 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

条目	描述
表示：	总损失函数
作用：	计算神经网络总损失
输入：	各网络模块输入输出
输出：	状态所有维度的预测和实际轨迹对比图

- 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

条目	描述
表示：	LQR 控制计算
作用：	根据 LQR 控制规则计算控制输入
输入：	Q，R 矩阵，参考点，初始状态，控制步长
输出：	LQR 控制输入

- 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

条目	描述
表示：	控制输入及对应状态值可视化
作用：	展示 LQR 控制输入值及控制下的状态值
输入：	LQR 控制输入值，控制下的状态值
输出：	LQR 控制输入及对应状态值图示

- 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] S. L. Brunton, B. W. Brunton, J. L. Proctor, K. J. Nathan, and H. A.Kestler, “Koopman invariant subspaces and finite linear representationsof nonlinear dynamical systems for control,” Plos One, vol. 11, no. 2, p. e0150171, 2016.

[9] P. J. 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] J. L. Proctor, S. L. Brunton, and J. N. Kutz, “Generalizing Koopman theory to allow for inputs and control,” SIAM Journal on Applied Dynamical Systems, vol. 17, p. 909–930, 2016.