Homework: Data-Driven Methods for Complex Dynamical Systems

Lu Wan

June 21, 2020

Abstract

In many scientific research, it is essential to find a model from a bunch of data even without any information of governing equations of the system. When the dynamical system has non-linearity and time-variant characteristics, the conventional regression methods may not predict the model very well. The main focus of this course is on characterizing complex systems through dimensional reduction, sparse sampling, and dynamical systems modeling by many algorithms, such as Singular Value Decomposition (SVD), Dynamic Mode Decomposition (DMD), Sparse Identification of Nonlinear Dynamics (SINDy), Koopman Operator Theory, Delay Coordinates and the Proper Orthogonal decomposition (POD). Besides all the mentioned algorithms, machine learning and data analysis are explored in many aspects, such as regression and model selection, clustering and classification and neural network and deep learning. All the code is on Github, https://github.com/LuWan-dot/AMATH582-upload.

1 Introduction and Overview

In this section, a brief and basic introduction of the course is presented. The goal is to build a model from a bunch of data which could be from simulation or measurement. How to find out a reduced space model which can represent the system and in order to predict the future is a key goal in this course. In the introduction section, main parts in the lecture are briefly presented and explained, and details could be found in the book[1]. Four parts of this book are briefly reviewed and the main ideas from lecture are explained in this sections. These four chapters are:

1. Dimensionality Reduction and Transforms.
2. Machine Learning and Data Analysis.
3. Dynamic and control.
4. Reduced Order Models

Besides, many key themes are discussed in this course. First, how to find the *dominant low-dimensional patterns* of complex systems is sometimes the first step to do with the data. This low-dimensional patterns could enables efficient sensing and compact representations for modeling and control. The proper orthogonal decomposition (POD) including SVD is the main tool of dimension reduction. This pattern extraction has relationship with the second theme of how to find the proper coordinate to represent the model simply. Many techniques in the *coordinate transforms* are familiar to engineering, such as, spectral decompositions, the Fourier transform, generalized functions, etc. In this book, many data-driven coordinate transforms are explained, like, DMD, SINDy, Koopman, etc. The third theme is *dynamical systems and control* in chapter 3. However, in class, more time was spent on data-driven dynamical systems than control. when handling big data, first to apply the coordinate transforms to find approximated low-dimensional model and then to train a Neural Network (NN) based on this model is a kind of good method. The fourth themes is *data-driven applied optimization*, such as, finding optimal low-dimensional patterns, optimal sensor placement, machine learning optimization and optimal control. In class, the case about optimization of sensor placement was explained. In the following subsections, these four chapters are briefly summarized for future reference, some of key points are also explained in these subsections.

1.1 Dimensionality Reduction and Transforms

There are three parts in this chapter: SVD, Fourier and wavelet transforms, and sparsity and compressed sensing. Any signal could be described by modes and the values on that modes. Fourier and wavelet are two types of coordinates transformation which utilize sinusoidal and and wavelet as basis modes. The desired coordinate systems are simplify, decouple and amenable to computation and analysis.

Besides, the SVD is a generalized concept of the fast Fourier transform (FFT). The SVD provides a numerically stable matrix decomposition that can be used for a variety of purposes and is guaranteed to exist. It could be used to obtain low-rank approximations to matrices and to perform pseudo-inverses of non-square matrices to find the solution of a system of equations Ax=b. Some of large matrices of complex systems actually contains few dominant patterns that explain the high-dimensional data. This SVD is a numerically robust and efficient method of extracting these patterns from data. Therefore, SVD is usually the first step when analysing data of dynamic systems in some algorithems like DMD, POD, classification methods, etc.

1.2 Machine Learning and Data Analysis

Machine learning is based on optimization techniques for data. The goal is to find both low-rank subspace for optimally embedding the data, as well as regression methods for clustering and classification of different data types. Data mining, i.e. a principle set of mathematical methods for extracting meaningful features from data, as well as binning the data into distinct and meaningful patterns that can be exploited for decision making.

There are two broad categories for machine learning: *supervised machine learning* and *unsupervised machine learning*. The main different between these two categories is whether the training data is labeled or not. In supervised machine learning, the labeled training data is used as the input of the training model, and the optimized model trained on the labeled data is used for prediction and classification using new data. There are many sub-topics in supervised machine learning, such as *active learning* (when limited sets of labeled data for training), *semi-supervised learning* (when some of input data is not labeled) and *reinforcement learning* (rewards or punishments are the training labels that help shape the regression architecture in order to build the best model). On the contrary, there are no labels of training data for unsupervised learning algorithms. It must find patterns in the data in a principled way in order to determine how to cluster data and generate labels for predicting and classifying new data.

All of this machine learning architecture is finding low-rank feature spaces that are informative and interpretable. The goal is to construct and exploit the intrinsic low-rank feature space of a given data set.In the following sub-sections, the most commonly used supervised and unsupervised machine learning methods exploited in class are briefly listed here.

1.2.1 Unsupervised machine learning

1. K-means clustering.

   The K-means algorithm assumes one is given a set of vector valued data with the goal of partitioning m observations into k clusters. Each observation is labeled as belonging to a cluster with the nearest mean, which serves as a proxy (prototype) for that cluster.

2. Hierarchical Clustering: Dendrogram.

   Dendrograms are created from a simple hierarchical algorithm, allowing one to efficiently visualize if data is clustered without any labeling or supervision. There are two types of hierarchical clustering: Agglomerative (bottom-up approach), and Divisive (top-down approach).

3. Mixture Models and the Expectation-Maximization Algorithm.

   This method is known as *Gaussian mixture models* (GMM), when the models are assumed to be Gaussian distributions.

1.2.2 Supervised machine learning

1. Linear Discriminant Analysis (LDA)

   The goal of LDA is to find a linear combination of features that characterizes or separates two or more classes of objects or events in the data.

2. Support Vector Machines (SVM)

   The SVM is a core machine learning tool along with *random forest* algorithm. With enough training data, the SVM can now be replaced with deep neural nets. The simply one is linear SVM, when data is complex, nonlinear SVM could be used, which includes nonlinear features and enrich the new space. And Kernel methods for SVM could reduce the computation in nonlinear SVM which enrichs higher dimensions.

1.2.3 Neural Networks

Neural Networks (NN) can be regarded as a map from input to output. The NN is built on trained data and validated with withdraw data, and used to predict with new data. Since the NNs have significant potential for overfitting of data, the cross-validation must be considered. NNs offer an amazingly flexible architecture for performing a diverse set of mathematical tasks, for instance, it can be used for future state predictions of dynamical systems.

1.3 Dynamic and control

There are four parts in this chapter in the book [1], data-driven dynamical system, linear control theory, balanced models for control and data-driven control. During the lecture, the first part, data-driven dynamical was delivered with more time. Some methods like DMD, SINDy, Koopman operator are briefly reviewed in section section.

1.4 Reduced Order Models

The proper orthogonal decomposition (POD) is the SVD algorithm applied to partial differential equations (PDEs). The POD technique seeks to take advantage of this fact in order to produce low-rank dynamical systems capable of accurately modeling the full spatio-temporal evolution of the governing complex system.

2 Theoretical Background

2.1 SVD

The SVD is easily computed in many languages. For example, if we have a large data set $\mathbf{X} \in \mathbf{C^{n\times m}}$

$$M = \begin{bmatrix} | & | & & & | \\ x_1 & x_2 & ... & ... & x_m \\ | & | & & & | \end{bmatrix} \tag{1}$$

We have n states and m snapshots, after the decomposition of X, we have three matrices, $\mathbf{U} \in \mathbf{C^{n\times n}}$, $\mathbf{V} \in \mathbf{C^{m\times m}}$ and $\mathbf{\Sigma} \in \mathbf{C^{n\times m}}$.

$$\mathbf{X} = \mathbf{U\Sigma V}^* \tag{2}$$

Since the SVD provides a hierarchy of low-rank approximations, a rank r approximation could be obtained by keeping the leading r singular values and vectors and discarding the rest.

$$\mathbf{X} \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^* \tag{3}$$

And these truncation matrices are $\tilde{\mathbf{U}} \in \mathbf{C^{n\times r}}$, $\tilde{\mathbf{V}} \in \mathbf{C^{m\times r}}$ and $\tilde{\mathbf{\Sigma}} \in \mathbf{C^{r\times r}}$. The columns of the matrix $\mathbf{U}$ provide an orthonormal basis for the column space of $\mathbf{X}$. Similarly, the columns of $\mathbf{V}$ provide an orthonormal basis for the row space of X. If the columns of $\mathbf{X}$ are spatial measurements in times, then $\mathbf{U}$ encode spatial patterns, and V encode temporal patterns.

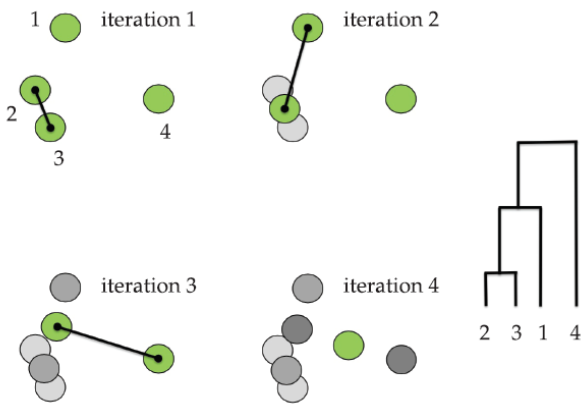


Figure 1: Illustration of the agglomerative hierarchical clustering applied to four data points [1].

2.2 Unsupervised learning: K-means clustering

The k-means algorithm is iterative, first assuming initial values for the mean of each cluster and then updating the means until the algorithm has converged. the algorithm proceeds as follows:

1. given initial values for k distinct means, compute the distance of each observation x_j to each of the k means.
2. Label each observation as belonging to the nearest mean.
3. Once labeling is completed, find the *center-of-mass* (mean) for each group of label points. These new means are then used to start back at step (1) in the algorithm.

2.3 Unsupervised learning: Dendrogram

The illustrated way to explain the dendrogram in class is shown in Fig. 1. In the algorithm, the distance between the four data points is computed. Initially the Euclidian distance between points 2 and 3 is closest. Points 2 and 3 are now merged into a point mid-way between them and the distances are once again computed. The dendrogram on the right shows how the process generates a summary (dendrogram) of the hierarchical clustering. Note that the length of the branches of the dendrogram tree are directly related to the distance between the merged points.

2.4 Unsupervised learning: Gaussian Mixture Models

The basic assumption in this method is that data observations x_j are a mixture of a set of k processes that combine to form the measurement. Like k-means and hierarchical clustering, the GMM model we fit to the data requires that we specify the number of mixtures k and the individual statistical properties of each mixture that best fit the data. GMMs are especially useful since the assumption that each mixture model has a Gaussian distribution implies that it can be completely characterized by two parameters: the mean and the variance.

2.5 Supervised learning: Linear Discriminant Analysis

The LDA algorithm aims to solve an optimization problem to find a subspace whereby the different labeled data have clear separation between their distribution of points. This then makes classification easier because an optimal feature space has been selected. The goal of LDA is two-fold: find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data.

The illustration of LDA is shown in Fig. 2. A general projection could lead to very poor discrimination between the data. LDA can separates the probability distribution functions in an optimal way.

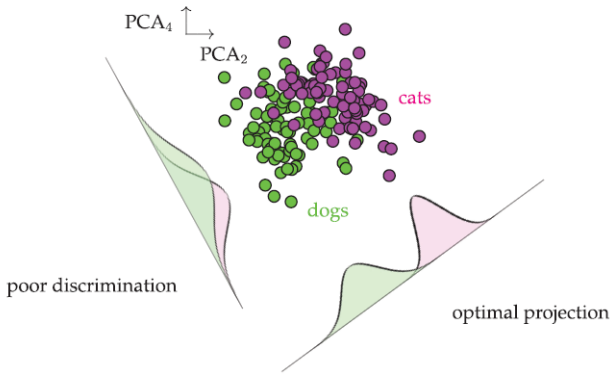


Figure 2: Illustration of LDA [1].

2.6 Supervised learning: Support Vector Machines

Linear SVM: The optimization problem associated with SVM is to not only optimize a decision line which makes the fewest labeling errors for the data, but also optimizes the largest margin between the data.

Nonlinear SVM: In order to build model of restrictive and high-dimensional space data, the feature space for SVM must be enriched. Nonlinear SVM maps the data into a nonlinear higher-dimensional space, so as to include nonlinear features and build hyperplanes in the new space. And the optimization of clusters is made in the higher-dimensional new space.

Kernel Methods for SVM: The higher-dimensions in nonlinear SVM leads to an expensive computation. Kernel function allows us to represent Taylor series expansions of a large (infinite) number of observable in a compact way. The kernel function enables one to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between all pairs of data in the feature space.

2.7 Neural networks and Deep Learning

The NN can be multi-layers, the activation functions in each layer should be chosen. Importantly, the chosen activation function will be differentiated in order to be used in gradient descent algorithms for optimization. Many functions like, linear, binary step, logistic, TanH, rectified linear unit (ReLU) are either differentiable or piecewise differentiable.

The NN is built when all weights of the network are determined. In practice, the objective function chosen for optimization is a proxy due to the ability to differentiate the objective function. A loss function can be chosen so as to approximate the true objective. The backpropagation (Backprop) algorithm exploits the compositional nature of NNs in order to frame an optimization problem for determining the weights of the network. Backprop relies on a simple mathematical principle: the chain rule for differentiation.

Backprop allows for an efficient computation of the objective function's gradient. While, another algorithm is also critical for enabling the training of NNs, which is *stochastic gradient descent* (SGD). SGD provides a more rapid evaluation of the optimal network weights. Since SGD does not estimate the gradient using all n data points. Only a single, randomly chosen data point, or a subset for batch gradient desent, is used to approximate the gradient at each step of the iteration.

Deep convolutional neural networks (DCNN) is fundamental building block of deep learning methods. The prototypical structure of a DCNN includes convolutional layers(each convolution window transforms the data into a new node through a given activeation function) and pooling layers (to progressively reduce the spatial size of the representation in order to reduce the number of parameters and computation in the network).

NN can also be used to predict the future state of dynamical systems. The goal is to train a NN to learn an update rule which advances the state space from x_k to x_{k+1}, where k denotes the state of the system at time t_k. The NN is trained with input and output data, while, input data is a matrix of the system at x_k and the output is the corresponding state of the system x_{k+1} advanced Δt.

2.8 Dynamic Mode Decomposition (DMD)

DMD is based on proper orthogonal decomposition (POD), which utilizes the computationally efficient singular value decomposition, so that it scales well to provide effective dimensionality reduction in high-dimensional systems. DMD not only provides dimensionality reduction in terms of a reduced set of modes, but also provides a model for how these modes evolve in time.

The first step is to reshape the snapshots of the state of a system evolving in time to two matrices. **X** and **X'**. The DMD algorithm seeks the leading decomposition (eigenvalues and eigenvectors) of the best-fit linear operator A that relates the two snapshot matrices in time: $\mathbf{X'} \approx A\mathbf{X}$. The four steps and the method to write the spectral expansion in continuous time can be found in chapter 7.2 in book [1].

2.9 Sparse Identification of Nonlinear Dynamics (SINDy)

The SINDy leverages the fact that many dynamical systems have dynamics f with only a few active terms in the space of possible rignt-hand side functions. We seek to approximate f by a generalized linear model with the fewest nonzero terms in ξ as possible. A library of candidate nonlinear function $\Xi(X)$ may be constructed from the data X.

$$\frac{\mathrm{d}}{\mathrm{d}t}x = \mathbf{f(x)} \tag{4}$$

$$\mathbf{f(x)} \approx \sum_{k=1}^{p} \theta_k(x)\xi_k = \Theta(x)\xi \tag{5}$$

2.10 Koopman Operator Theory

Koopman theory is that any nonlinear dynamical system could be represented in terms of an infinite-dimensional linear operator. In practical, obtaining finite-dimensional, matrix approximations of the Koopman operator is the focus of intense research efforts and holds the promise of enabling globally linear representations of nonlinear dynamical systems.

The Koopman operator is linear, which is appealing, and a prmary motivation to adopt the Koopman framework is the ability to simplify the dynamics through the eigen-decomposition of the operator.

However, obtaining Koopman eigenfunctions from data or from analytic expressions is a central applied challenge in modern dynamical systems. A set of Koopman eigenfunctions may be used to generate more eigenfunctions like the first question in this homework.

3 Algorithm Implementation and Development

3.1 Least square curve fitting

In the 3rd question of part 1, we need to use data to fit values of 4 parameters in Lotka-Volterra equations. Here are some steps of this algorithm.

1. Define the Lotka-Volterra equations in a form ready for solution by ode45. The function is `LVequation()` in Appendix A.

2. Define the variable r for the solution of 4 parameters. In this question, r is 4*1 matrix.

3. Since the objective function for this problem is the sum of squares of the differences between the ODE solution with parameter r and the provided population data. In order to express the objective function, in this step, we define the function that computes the ODE solution using parameters r. This function is `solpts = RtoODE(r,tspan,y0)` in Appendix A.

4. To use `RtoODE` in an objective function, convert the function to an optimization expression by using `fcn2optimexpr`.

5. Express the objective function as the sum of squared differences between the ODE solution and the solution with true parameters.

6. Create an optimization problem with the objective function `obj`. And view the problem.

7. Give the initial guess $r0$ for the solver and call `solve`.

8. Use the solution to reconstruct the data and compare with original data.

3.2 Extended DMD

Extended DMD (eDMD) is considered in 3rd question of part 1. The extended DMD algorithm is essentially the same as DMD, expect that instead of performing regression on direct measurements of the state, regression is performed on an augmented vector containing nonlinear measurements of the state. In this question, the state xy is used besides states x and y. However, this set of Koopman eigenfunctions is used to generate more eigenfunctions. It establishes a commutative monoid under point-wise multiplication.

3.3 SINDy with interpolation

Since the population data is given by years, the results of SINDy is not correct due to the derivatives are computed with nearby points. In order to avoid it, the interpolation method is used before SINDy. Spline interpolation using not-a-knot end conditions. The interpolated value at a query point is based on a cubic interpolation of the values at neighboring grid points in each respective dimension.

4 Computational Results

4.1 Part1: Models of population data

There are five methods used to find the model of the population of lynx and snowshoe. The time-delay model is the best fit nonlinear, dynamic systems model to the given data.

4.1.1 Q1.1: Basic DMD model

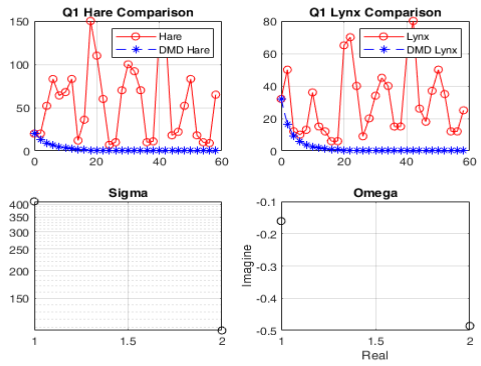
The basic DMD model was applied and the code is in List 1. The results are shown in Figure 3 . The left-bottom figure is the Ξ of the SVD result of matrix X. The prediction results of basic DMD is not satisfying, since the decay due to the real part of Ω is too strong, the predicted model failed after the known input first data.

4.1.2 Q1.2: Time-delay DMD model

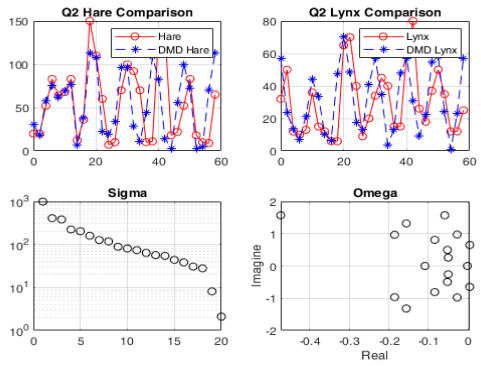
Time-delay DMD model is applied with 10 times delay coordinate. In this time-delay coordinate, the Ξ shows that we can reconstruct the data with top 18 modes. Compared the DMD model with original data, it shows better prediction than basic DMD model. And there are 3 elements of ω with zero real parts and other elements are symmetrically distributed.

4.1.3 Q1.3: Least Square Curve Fitting

Since the Lotka-Volterra equations are provided, we can use the least square curve fitting method to find the best fit parameters of b, p, r, and d. The algorithm could be found in chapter 3. In Figure 4a. Some oscillation of prediction model but it could not show the exact result as the original data. And the results are sensitive to the initial guess, which could lead the results to completely different model. Therefore, this curve fitting method is better than basic DMD, but has no compatition with the time-delay DMD, which is better. This told us that even though we knew the similar equation to model the population phenomenon, we can not expect that this equation is sure to help us to build the model from the original data.



(a) Basic DMD model of Q1.1

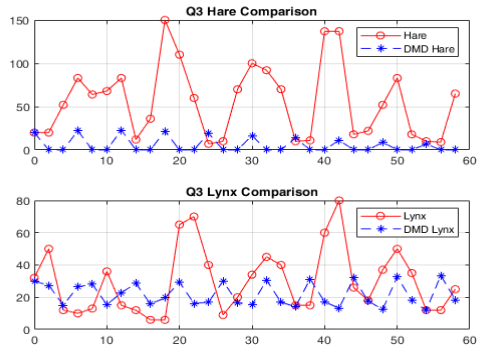


(b) Time-delay DMD model of Q1.2

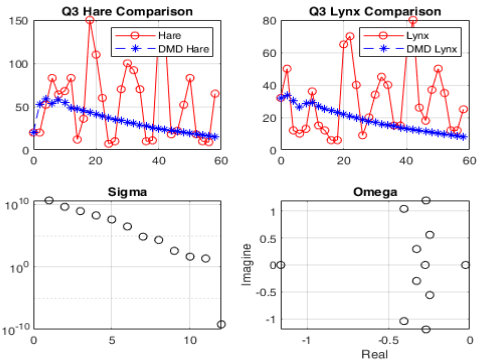
Figure 3: DMD model of part 1

4.1.4 Q1.3: Extended DMD

We could find that in the equation, xy is also an important element in the differential equations. Since we knew that from last section, the least square curve fitting failed when we want to find best fit value for a specific equations. Although this equation could not fit the data perfectly, maybe we could get some ideas from this equation. Since we have a sense that this data may have latent varable like xy. In order to consider this in modeling, the Koopman coordinate is considered before DMD, the eDMD method is used. However, xy generates more eigenfunction elements, only a set of eigenfunction are selected. However, the result in Figure 4b shows that this eDMD doesn't work on this data, maybe the reason is that only s mall set of eigenfunction is selcted compared to supposed infinite elements.



(a) Least square curve fitting method of Q1.3



(b) Extended DMD of Q1.3

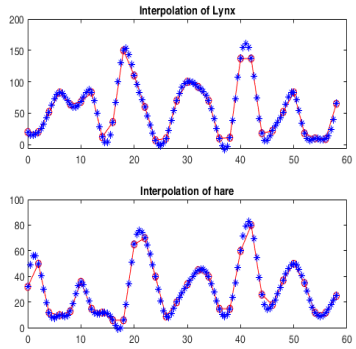
Figure 4: Curve fitting and eDMD model of part 1

4.1.5 Q1.4: Sparse Regression model

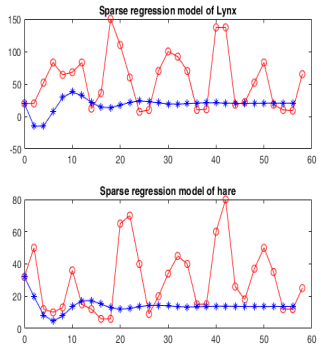
In order to find the sparse regression model, SINDy method is applied. However, the results is way far from the original data due to the derivatives computed with large time interval data. In order to find the correct model, interpolation is used before SINDy. The curious thing is that in the sparse regression model, it has Constant term, but without the term xy, it is kind of strange to me. The sparse regression model is better than the basic DMD, but after time 20, it keeps constant due to the constant term and without the term xy in sparse regression model.

```
newout =

  11×3 cell array

    {0×0 char}    {'xdot'   }    {'ydot'   }
    {'1'      }    {[19.0732]}    {[-1.3923]}
    {'x'      }    {[      0]}    {[ 0.1501]}
    {'y'      }    {[-0.6041]}    {[-0.2349]}
    {'xx'     }    {[      0]}    {[      0]}
    {'xy'     }    {[      0]}    {[      0]}
    {'yy'     }    {[      0]}    {[      0]}
    {'xxx'    }    {[      0]}    {[      0]}
    {'xxy'    }    {[      0]}    {[      0]}
    {'xyy'    }    {[      0]}    {[      0]}
    {'yyy'    }    {[      0]}    {[      0]}
```

(a) SINDy result of Q1.4



(b) Interpolation of original data



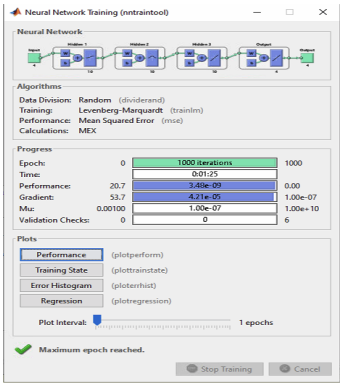
(c) SINDy model of Q1.4

Figure 5: Sparse regression model of part 1

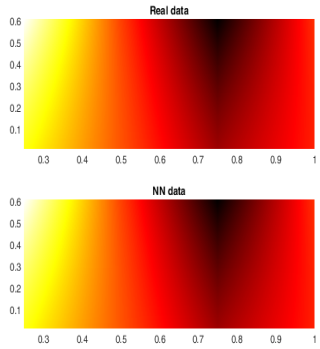
4.2 Part2: NN for reaction-diffusion system and Kuramoto-Sivashinsky system

4.2.1 Q2.1 and 2.2: NN for reaction-diffusion system

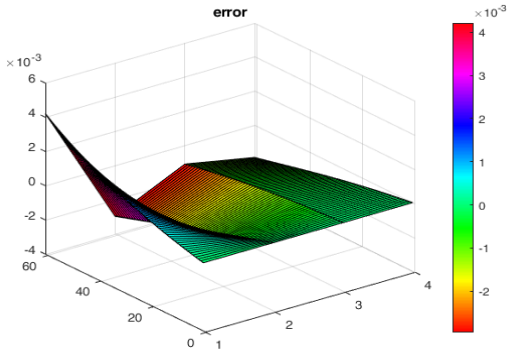
In this subsection, a NN is trained with only 4 states rather than 2048 states, the issues is that it can not run with 2048 states. I don't know where is the problem. When I tried to project 2048 states to lower space with SVD, I could not train a NN with V, U and Ξ in this lower space, since V, U and Ξ, only V shows the time characteristics, it is not possible to do the time advance from t to $t + t$ in the SVD coordinate. The Figure 6b shows the comparison between NN and ODE for 4 states. and Figure 6c shows the error in space between NN and ODE.



(a) NN tool result



(b) Comparison of NN and ODE

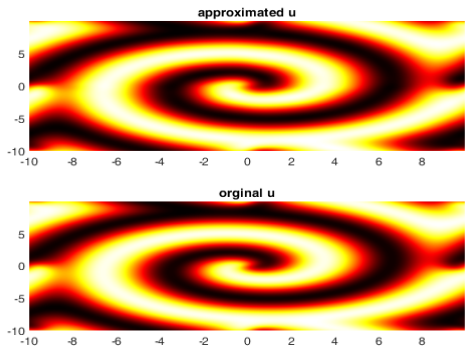


(c) Error between NN and ODE

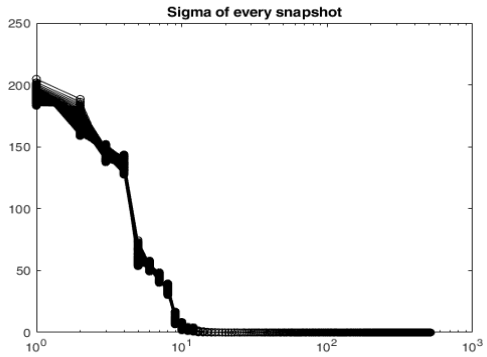
Figure 6: NN for reaction-diffusion system of Q2.1, 2.2

4.2.2 Q2.3: NN for Kuramoto-Sivashinsky system

In this section, the Kuramoto-Sivashinsky system is projected into a low-rank space. In Figure 7, each snapshot was taken SVD, and the 10 modes are used to construct the original data u. In Figure 8, all data used to find the low-rank space, and the dominant rank is 4 when using all data. Here I have the same problem when training NN. It doesn't work when I have such big data, even I reduce it using SVD, and I reshape U, V and Ξ together as one matrix, and use it as input and output. But MATLAB didn't pop up the nntools.

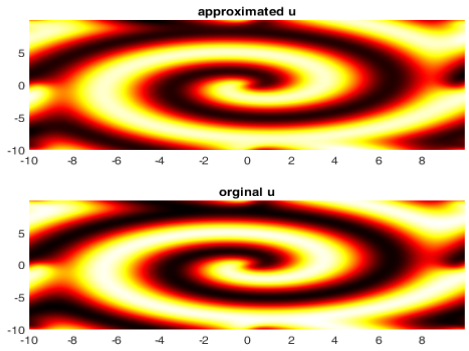


(a) Low-rank reconstruction and original u of Q2.3

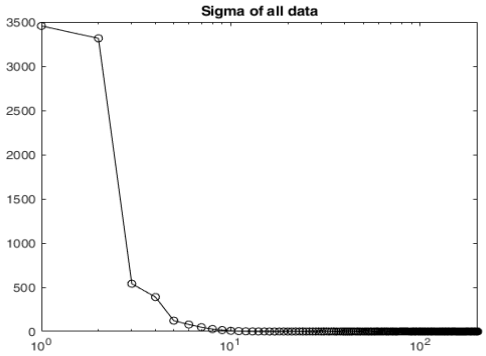


(b) Sigma of each snapshot Q2.3

Figure 7: Low-rank model of part 2.3



(a) Low-rank reconstruction and original u of Q2.3



(b) Sigma of All data Q2.3

Figure 8: Low-rank model with all data of part 2.3

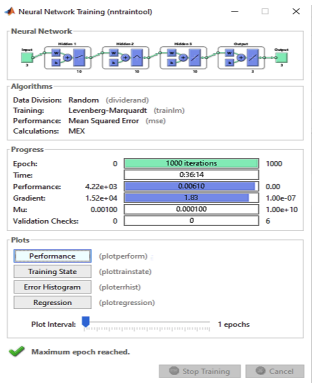
4.3 Part3: NN for Lorenz equation

4.3.1 Q3.1: one NN for three different Lorenz equation

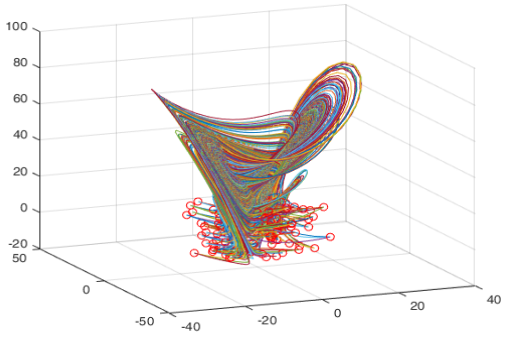
One NN is trained with 3 sets of data, each sets contains 100 trajectories. These three sets have ρ are 10,28 and 40 respectively. It turns out that the NN can not predict the new data from equation with new ρ. If more number of ρ could cover the desired ρ 17 and 35, maybe the results will be better.

4.3.2 Q3.2: NN for transition classification

Firstly the label of the training data should be found. The sign of each state times next state of x is computed. The input training data is the small window (20 times t), and the label is chosen 1, 5, 15 and 20 times t as the prediction time. And the prediction is in Figure 11, 12 and 13. Overall, all NN models can predict very well. However, if some transition happens at the very beginning time, some NN models may not predict it. For example, If we have smaller prediction time, like 0.01 and 0.05, it can predict well. If the prediction time is longer, it fails to prediction the very beginning transition, because the transition falls in the window.

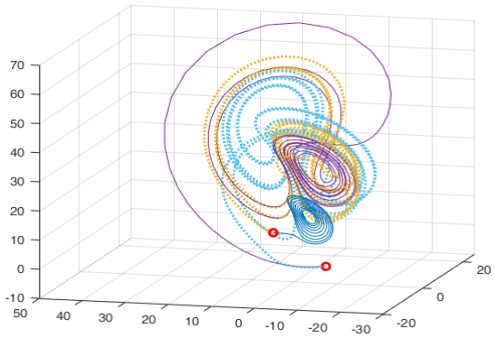


(a) NN train tool of Q3.1

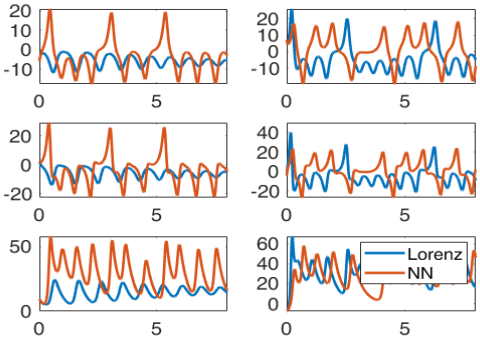


(b) Training data of Q3.1

Figure 9: Taining data and NN tool of part 3.1



(a) Trajectories of NN and ODE of Q3.1



(b) Comparison of NN and ODE of Q3.1

Figure 10: NN and ODE comparison of part 3.1

4.4 Part4: Exploration of Belousov-Zhabotinsky

4.4.1 DMD

DMD is applied on this BZ data first. It can be found that some ω are distributed on the left of zero x axis, most of them are close to zero x axis, some are distributed symmetrically on the left.

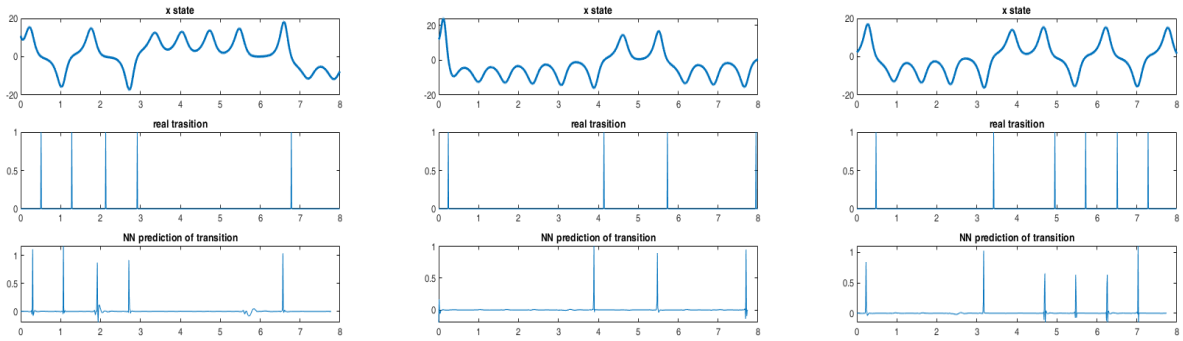
4.4.2 Time delay DMD

Time delay DMD is applied on this BZ data. Compared with only basic DMD, almost all ω are close to zero x axis. It turns out that the time delay DMD is better to represent the BZ data in a smaller space.

5 Summary and Conclusions

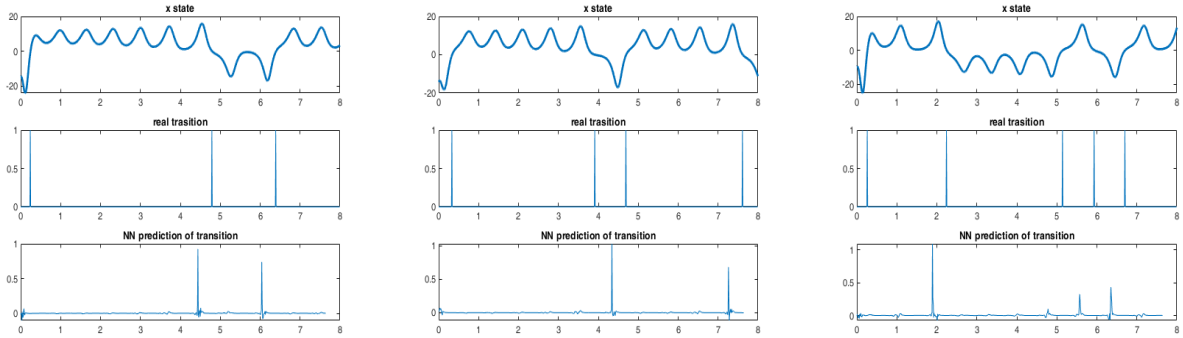
In this course, we learnt many method to build a model from data of dynamic system. Some of them are quite useful when dealing with data without any governing equations. Some methods like SVD, DMD, SINDy, Koopman operator and time-delay coordinate are good tools to explore the data from measurement or simulation.

Besides, these tools for dynamic system. The architecture of NN and deep learning were explained very well. These method are also quite useful. We can also get many resources like the book, website and videos to learn more. I hope I could use some of ideas for my research.



(a) prediction time is 0.01 (b) prediction time is 0.05 (c) prediction time is 0.05

Figure 11: Window 0.2, prediction time 0.01 and 0.05



(a) prediction time is 0.15 (b) prediction time is 0.15 (c) prediction time is 0.15

Figure 12: Window 0.2, prediction time 0.15

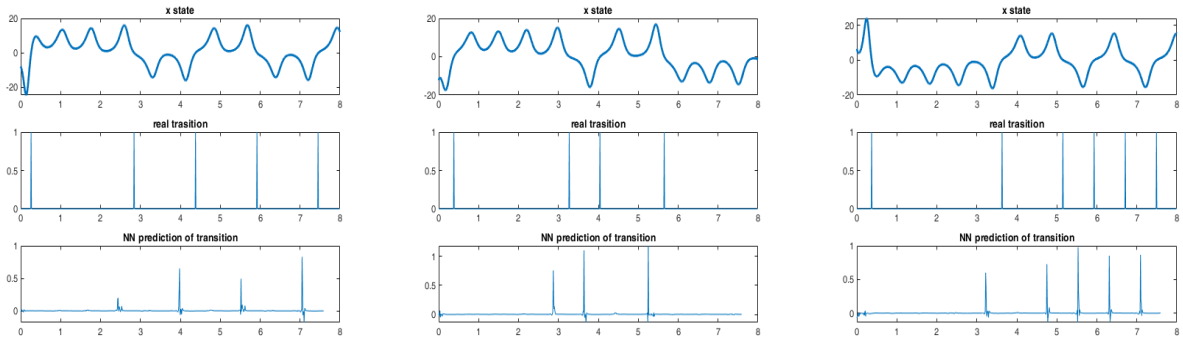
References

[1] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

Appendix A MATLAB Functions

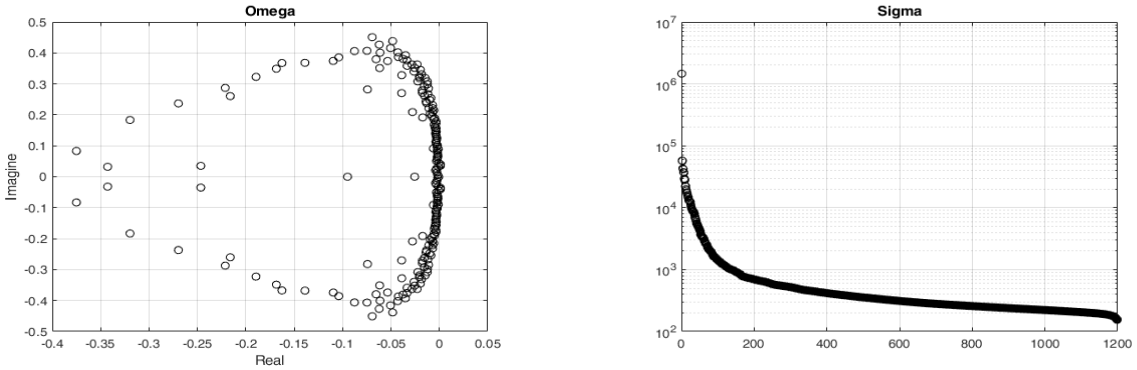
The important MATLAB functions are listed below.

- `[Phi,omega,lambda,b,Xdmd,S] = DMD2(X1,X2,r,t)` returns the output results of DMD, including the reconstructed model `Xdmd`.
- `dydt = LVequation( ,y,r)` returns the left part of the differential equations.
- `solpts = RtoODE(r,tspan,y0)` returns the ODE solution using parameters r.
- `[tsave, xsave, usave] = KSequation(u,N)` returns the KS equation with initial input.
- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[X,Y] = meshgrid(x,y)` returns 2-D grid coordinates based on the coordinates contained in the vectors `x` and `y`. X is a matrix where each row is a copy of `x`, and `Y` is a matrix where each column is a copy of `y`. The grid represented by the coordinates `X` and `Y` has `length(y)` rows and `length(x)` columns.



(a) prediction time is 0.2 (b) prediction time is 0.2 (c) prediction time is 0.2

Figure 13: Window 0.2, prediction time 0.2

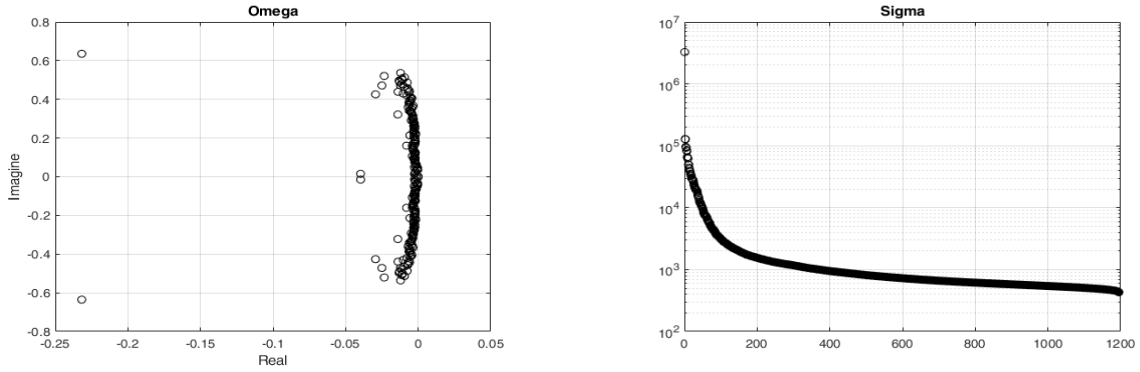


(a) Omega of Q4 (b) Sigma of Q4

Figure 14: DMD model of part 4

Appendix B MATLAB Code

Add your MATLAB code here. This section will not be included in your page limit of six pages.



(a) Omega of Q4

(b) Sigma of Q4

Figure 15: Time-delay DMD model of part 4

```
%%%% Q1.1 Develop a DMD model to forecast the future population states
clear all; close all; clc;

%%%%data x1--Snowshoe Hare   x2----Canada Lynx
x1 = [20,20,52,83,64,68,83,12,36,150,110,60,7,10,70,...
    100,92,70,10,11,137,137,18,22,52,83,18,10,9,65];
x2 = [32,50,12,10,13,36,15,12,6,6,65,70,40,9,20,...
    34,45,40,15,15,60,80,26,18,37,50,35,12,12,25];

slices = 30;
t = linspace(0,58,slices);
dt = t(2) - t(1);
X =[x1;x2];
X1 = X(:,1:end-1);
X2 = X(:,2:end);
r = 2;

[Phi_q1,omega_q1,lambda_q1,b_q1,Xdmd_q1,S_q1] = DMD2(X1,X2,r,t);

fq1 = figure;
subplot(2,2,1)
plot(t,x1,'ro-',t,abs(Xdmd_q1(1,:)),'b*--')
legend('Hare','DMD Hare');
title('Q1 Hare Comparison')
grid on

subplot(2,2,2)
plot(t,x2,'ro-',t,abs(Xdmd_q1(2,:)),'b*--')
legend('Lynx','DMD Lynx');
title('Q1 Lynx Comparison')
grid on

subplot(2,2,3);
semilogy(abs(diag(S_q1)),'ko');
title('Sigma')
grid on

subplot(2,2,4)
plot(omega_q1,'ko')
title('Omega')
xlabel('Real')
ylabel('Imagine')
grid on
```

Listing 1: Code1.1 Basic DMD model.

```
%%%% Q1.2 Do a time-delay DMD model to produce a forecast and compare with
%%%%regular DMD. Determine if it is likely that there are latent variables.
clear all; close all; clc;

%%%%data x1--Snowshoe Hare   x2----Canada Lynx
x1 = [20,20,52,83,64,68,83,12,36,150,110,60,7,10,70,...
    100,92,70,10,11,137,137,18,22,52,83,18,10,9,65];
x2 = [32,50,12,10,13,36,15,12,6,6,65,70,40,9,20,...
    34,45,40,15,15,60,80,26,18,37,50,35,12,12,25];

slices = 30;
t = linspace(0,58,slices);
dt = t(2) - t(1);

X1_q2 =[];
X2_q2 =[];
kk = 10;
for j=1:kk
    X1_q2 = [X1_q2;x1(j:29-kk+j);x2(j:29-kk+j)];
    X2_q2 = [X2_q2;x1(j+1:29-kk+j+1);x2(j+1:29-kk+j+1)];

end
r_q2 = 18;

[Phi_q2,omega_q2,lambda_q2,b_q2,Xdmd_q2,S_q2] = DMD2(X1_q2,X2_q2,r_q2,t);

fq2 = figure;
subplot(2,2,1)
plot(t,x1,'ro-',t,abs(Xdmd_q2(1,:)),'b*--')
legend('Hare','DMD Hare');
title('Q2 Hare Comparison')
grid on

subplot(2,2,2)
plot(t,x2,'ro-',t,abs(Xdmd_q2(2,:)),'b*--')
legend('Lynx','DMD Lynx');
title('Q2 Lynx Comparison')
grid on

subplot(2,2,3);
semilogy(abs(diag(S_q2)),'ko');
title('Sigma')
grid on

subplot(2,2,4)
plot(omega_q2,'ko')
title('Omega')
xlabel('Real')
ylabel('Imagine')
grid on
```

Listing 2: Code1.2 Time-delay DMD model.

```
%%%% Q1.3 Empirical Predator-Prey models such as Lotka-Volterra are commonly
%%%%used to models such phenomenon. Consider the model
%%%%Use the data to fit values of b, p, r and d.
clear all; close all; clc;

%%%%data x1--Snowshoe Hare   x2----Canada Lynx
x1 = [20,20,52,83,64,68,83,12,36,150,110,60,7,10,70,...
    100,92,70,10,11,137,137,18,22,52,83,18,10,9,65];
x2 = [32,50,12,10,13,36,15,12,6,6,65,70,40,9,20,...
    34,45,40,15,15,60,80,26,18,37,50,35,12,12,25];
slices = 30;
t = linspace(0,58,slices);
dt = t(2) - t(1);
X =[x1;x2];
y0 = [20 30];
%%%% define variable for the solution b, p, r and d.
r = optimvar('r',4,'LowerBound',0.1,"UpperBound",15);
%%%%convert the function to an optimization expression
myfcn = fcn2optimexpr(@RtoODE,r,t,y0);
%%%%Express the objective function as the sum of squared differences
%%%%between the ODE solution and the solution with true parameters
obj = sum(sum((myfcn-X).^2));
%%%%Create an optimization problem with the objective function
prob = optimproblem("Objective",obj);
%%%%View the problem by calling show.
show(prob)
%%%%Give the initial guess £r0£ for the solver
%r0.r = [0.3 0.3 0.1 0.3];
r0.r = [7 0.3 0.1 0.3];
[rsol,sumsq] = solve(prob,r0);
disp(rsol.r)
%disp(true)
%%%%Use the solution to reconstruct the data and compare with original data
b = rsol.r(1);
p = rsol.r(2);
r = rsol.r(3);
d = rsol.r(4);
LV = @(t,x)([ b * x(1)-p * x(1)*x(2)       ; ...
                    r * x(1)*x(2)- d *x(2); ]);
[t,y] = ode45(LV,t,[20,30]);
%%
figure;
subplot(2,1,1)
plot(t,x1,'ro-',t,y(:,1),'b*--')
legend('Hare','DMD Hare');
title('Q3 Hare Comparison')
grid on
subplot(2,1,2)
plot(t,x2,'ro-',t,y(:,2),'b*--')
legend('Lynx','DMD Lynx');
title('Q3 Lynx Comparison')
grid on
```

Listing 3: Code1.3 Least square curve fitting.

```
clear all; close all; clc;

%%%%data x1--Snowshoe Hare   x2----Canada Lynx
x1 = [20,20,52,83,64,68,83,12,36,150,110,60,7,10,70,...
    100,92,70,10,11,137,137,18,22,52,83,18,10,9,65];
x2 = [32,50,12,10,13,36,15,12,6,6,65,70,40,9,20,...
    34,45,40,15,15,60,80,26,18,37,50,35,12,12,25];

slices = 30;
t = linspace(0,58,slices);
dt = t(2) - t(1);
X =[x1;x2];
X1 = X(:,1:end-1);
X2 = X(:,2:end);
r = 2;
X_q3 = [x1;x2;x1.*x2;x2.*x1.^2;x1.*x2.^2;x1.^3.*x2;x2.*x1.^3;...
    x1.^2.*x2.^2;x1.*x2.^4;x1.^2.*x2.^3;x1.^3.*x2.^2;x1.^4.*x2.^1;];
%X_q3 = [x1;x2;x1.*x2;x2.*x1.^2;x1.*x2.^2;x1.^3.*x2;x2.*x1.^3;x1.^2.*x2.^2];
X1_q3 = X_q3(:,1:end-1);
X2_q3 = X_q3(:,2:end);

r_q3 = 11;

[Phi_q3,omega_q3,lambda_q3,b_q3,Xdmd_q3,S_q3] = DMD2(X1_q3,X2_q3,r_q3,t);

fq3 = figure;
subplot(2,2,1)
plot(t,x1,'ro-',t,abs(Xdmd_q3(1,:)),'b*--')
legend('Hare','DMD Hare');
title('Q3 Hare Comparison')
grid on

subplot(2,2,2)
plot(t,x2,'ro-',t,abs(Xdmd_q3(2,:)),'b*--')
legend('Lynx','DMD Lynx');
title('Q3 Lynx Comparison')
grid on

subplot(2,2,3);
semilogy(abs(diag(S_q3)),'ko');
title('Sigma')
grid on

subplot(2,2,4)
plot(omega_q3,'ko')
title('Omega')
xlabel('Real')
ylabel('Imagine')
grid on
```

Listing 4: Code1.3 eDMD.

```
%%%% Q1.4 Find sparse regression using SINDy and interpolation
clear all; close all; clc;
x1 = [20,20,52,83,64,68,83,12,36,150,110,60,7,10,70,...
    100,92,70,10,11,137,137,18,22,52,83,18,10,9,65];
x2 = [32,50,12,10,13,36,15,12,6,6,65,70,40,9,20,...
    34,45,40,15,15,60,80,26,18,37,50,35,12,12,25];
slices = 30;
t = linspace(0,58,slices);
dt = t(2) - t(1);
r = 2;
dt_new = 0.5;
t_new = 0:dt_new:58;
n = length(t_new);
%%
x1_interp = interp1(t,x1,t_new,'spline').';
x2_interp = interp1(t,x2,t_new,'spline').';
X =[x1_interp(1:end-1) x2_interp(1:end-1)];
fq4 = figure();
subplot(2,1,1)
plot(t,x1,'ro-',t_new,x1_interp,'b*')
title('Interpolation of Lynx')
subplot(2,1,2)
plot(t,x2,'ro-',t_new,x2_interp,'b*')
title('Interpolation of hare')
for j=1:n-1
    x1dot(j) = (x1_interp(j+1)-x1_interp(j))/(dt_new);
    x2dot(j) = (x2_interp(j+1)-x2_interp(j))/(dt_new);
end
dx = [x1dot.' x2dot.'];
%% SINDy
polyorder = 3;
Theta = poolData(X,r,polyorder);
m =size(Theta,2);
lambda = 0.025;
Xi = sparsifyDynamics(Theta,dx,lambda,r)
poolDataLIST({'x','y'},Xi,r,polyorder);
%% reconstraction of data with sparse regression
options = odeset('RelTol',1e-12,'AbsTol',1e-12*ones(1,2));
 Beta(1)= Xi(1,1);
 Beta(2)= Xi(2,1);
 Beta(3)= Xi(1,2);
 Beta(4)= Xi(2,2);
Beta(5)= Xi(2,2);
Beta(6)= Xi(3,2);
[tt,xx]=ode45(@(tt,xx) LV2(tt,xx,Beta),t(1:end-1),[20 32],options);
fq5 = figure();
subplot(2,1,1)
plot(t,x1,'ro-',tt,xx(:,1),'b*-')
title('Sparse regression model of Lynx')
subplot(2,1,2)
plot(t,x2,'ro-',tt,xx(:,2),'b*-')
title('Sparse regression model of hare')
```

Listing 5: Code1.4 Sparse regression.

```
%Q1: Train a NN for KS equation
% a NN can advance from t to t+dt
clear all, close all
input=[];
output=[];
N = 4;
for j=1:100
    u0 = randn(N,1);
    [t,x,u] = KSequation(u0,N);
     input = [input; u(1:end-1,:)];
    output = [output; u(2:end,:)];

end
%%
net = feedforwardnet([10 10 10]);
net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'radbas';
net.layers{3}.transferFcn = 'purelin';

net = train(net,input.',output.');
%%
%Q2: Compare with different initial conditions
u_kk = randn(N,1);
[t_real,x_real,u_real] = KSequation(u_kk,N);
u_test1 = u_real(1,:).';
unn(1,:)=u_test1;
for jj=2:length(t_real)
    unext = net(u_test1);
    unn(jj,:)=unext.';
    u_test1=unext;
end
 %%
figure(1)
subplot(2,1,1)
pcolor(x_real,t_real,u_real),shading interp, colormap(hot)
title('Real data')
subplot(2,1,2)
pcolor(x_real,t_real,unn),shading interp, colormap(hot)
title('NN data')
figure(2)
surf(u_real-unn)
title('error')
colormap hsv
colorbar
```

Listing 6: Code2.1 NN for a reaction-diffusion system.

```
%%%%Q3 Reaction-diffusion system SVD
clear all, close all
%%% load data from original provided code
load('reaction_diffusion_big.mat')
%%% apply SVD on u, for every snapshot
dt = t(2) - t(1);
numt = length(t);
U = zeros(length(x),length(y),length(t));
V = zeros(length(x),length(y),length(t));
S = zeros(length(x),length(y),length(t));
for j=1:numt
    [U(:,:,j),S(:,:,j),V(:,:,j)] = svd(u(:,:,j),'econ');
end
%%% It could be found that the previous 10 space could be a good approx.
for jj=1:numt
   semilogx(abs(diag(S(:,:,jj))),'ko-'), hold on
   title('Sigma of every snapshot')
end
%%% truncate to rank-r, in order to represent the orginal model in low rank space
r = 10;
U_r = U(:,1:r,:);
S_r = S(1:r,1:r,:);
V_r = V(:,1:r,:);
%%% Convert approximated u with low rank space
u_approx = zeros(length(x),length(y),numt);
for jj=1:numt
    u_approx(:,:,jj) = U_r(:,:,jj)*S_r(:,:,jj)*V_r(:,:,jj)';
end
%%% compare the orginal u and the approximated u
figure;
subplot(2,1,1)
pcolor(x,y,u_approx(:,:,end)); shading interp; colormap(hot)
title('approximated u')
subplot(2,1,2)
pcolor(x,y,u(:,:,end)); shading interp; colormap(hot)
title('orginal u')
%%%%%%%%%%%%% compress all data together and find low-rank space
[k1,k2,k3] = size(u)
uu = reshape(u,k1*k2,k3);
[UU SS VV] = svd(uu,'econ');
figure;
semilogx(abs(diag(SS)),'ko-')
title('Sigma of all data')
rr = 4;
UU_r = UU(:,1:rr);
SS_r = SS(1:rr,1:rr);
VV_r = VV(:,1:rr);
uu_approx = UU_r*SS_r*VV_r';
u_approx2 = reshape(uu_approx,k1,k2,k3);
figure;
subplot(2,1,1)
pcolor(x,y,u_approx2(:,:,end)); shading interp; colormap(hot)
title('approximated u')
subplot(2,1,2)
pcolor(x,y,u(:,:,end)); shading interp; colormap(hot)
title('orginal u')
```


Listing 7: Code2.2 NN for a RS system.

```
%%%% Q3.1 Train a NN to advance the solution from t to t + {t
dt=0.01; T=8; t=0:dt:T;
b=8/3; sig=10;
%r_rain=[10, 28, 40];
r1 = 10;
r2 = 28;
r3 = 40;
Lorenz1 = @(t,x)([ sig * (x(2) - x(1)) ; r1 * x(1)-x(1) * x(3) - x(2) ; x(1) * x(2) - b*x(3)]);
Lorenz2 = @(t,x)([ sig * (x(2) - x(1));r2 * x(1)-x(1) * x(3) - x(2) ; x(1) * x(2) - b*x(3)]);
Lorenz3 = @(t,x)([ sig * (x(2) - x(1)); r3 * x(1)-x(1) * x(3) - x(2) ; x(1) * x(2) - b*x(3)]);
r4 = 17;
r5 = 35;
Lorenz4 = @(t,x)([ sig * (x(2) - x(1)) ; r4 * x(1)-x(1) * x(3) - x(2) ; x(1) * x(2) - b*x(3)]);
Lorenz5 = @(t,x)([ sig * (x(2) - x(1));  r5* x(1)-x(1) * x(3) - x(2) ;  x(1) * x(2) - b*x(3)]);
ode_options = odeset('RelTol',1e-10, 'AbsTol',1e-11);
input=[]; output=[];
for j=1:100  % training trajectories
    x0=30*(rand(3,1)-0.5);
    [t,y] = ode45(Lorenz1,t,x0);
    input=[input; y(1:end-1,:)];
    output=[output; y(2:end,:)];
    plot3(y(:,1),y(:,2),y(:,3)), hold on
    plot3(x0(1),x0(2),x0(3),'ro')

    [t,y] = ode45(Lorenz2,t,x0);
    input=[input; y(1:end-1,:)];
    output=[output; y(2:end,:)];
    plot3(y(:,1),y(:,2),y(:,3)), hold on
    plot3(x0(1),x0(2),x0(3),'ro')

    [t,y] = ode45(Lorenz3,t,x0);
    input=[input; y(1:end-1,:)];
    output=[output; y(2:end,:)];
    plot3(y(:,1),y(:,2),y(:,3)), hold on
    plot3(x0(1),x0(2),x0(3),'ro')
end
grid on, view(-23,18)
net = feedforwardnet([10 10 10]);
net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'radbas';
net.layers{3}.transferFcn = 'purelin';
net = train(net,input.',output.');
figure(2)
x0=30*(rand(3,1)-0.5);
[t,y] = ode45(Lorenz4,t,x0);
plot3(y(:,1),y(:,2),y(:,3)), hold on
plot3(x0(1),x0(2),x0(3),'ro','Linewidth',[2])
grid on
ynn(1,:)=x0;
for jj=2:length(t)
    y0=net(x0);
    ynn(jj,:)=y0.'; x0=y0;
end
plot3(ynn(:,1),ynn(:,2),ynn(:,3),':','Linewidth',[2])
figure(3)
subplot(3,2,1), plot(t,y(:,1),t,ynn(:,1),'Linewidth',[2])
subplot(3,2,3), plot(t,y(:,2),t,ynn(:,2),'Linewidth',[2])
subplot(3,2,5), plot(t,y(:,3),t,ynn(:,3),'Linewidth',[2])
figure(2)
x0=20*(rand(3,1)-0.5);
```

```
%%%% Q3.2 predict the transition
clear all; close all
dt=0.01; T=8; t=0:dt:T;
b=8/3; sig=10;
r = 28;
Lorenz = @(t,x)([ sig * (x(2) - x(1))       ; ...
                  r * x(1)-x(1) * x(3) - x(2) ; ...
                  x(1) * x(2) - b*x(3)         ]);
input=[];
output=[];
k = 1;
kk = 20;
t_window = dt*kk;
t_pre = dt*k;
lent = length(t);
for j=1:100  % training trajectories
    x0=30*(rand(3,1)-0.5);
    [t,y] = ode45(Lorenz,t,x0);
    signout = 1;
    signsum = 1;
    for n=1:lent-1
        signout = sign(y(n,1)*y(n+1,1));
        if signout >0
            signout2 = 0;
        else
            signout2 = 1;
        end
        signsum(n) = signout2;
    end

    lent2 = lent-kk-k-1;
    for i=1:lent-kk-k-1
        input(:,i+(j-1)*(lent2))=[y(i:i+kk,1);y(i:i+kk,2);y(i:i+kk,3)];
        output(1,i+(j-1)*(lent2))=signsum(i+kk+k);

    end
end

net = feedforwardnet([10 10 10]);
net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'radbas';
net.layers{3}.transferFcn = 'purelin';
net = train(net,input,output);

%%
x0=30*(rand(3,1)-0.5);
[t,y] = ode45(Lorenz,t,x0);
signout = 1;
signsum = 1;
for j=1:length(t)-1
    signout = sign(y(j,1)*y(j+1,1));
    if signout >0
        signout2 = 0;
    else
        signout2 = 1;
    end
   signsum(j) = signout2;
end

figure;
```

```
%%% view the data
clear all; close all; clc;
load('BZ.mat')
[m,n,k]=size(BZ_tensor);% x vs y vs time data
%
% for j=1:k
%     A = BZ_tensor(:,:,j);
%     pcolor(A),shading interp, pause(0.2)
% end
%%%% DMD
X = zeros(m*n,k);
for j=1:k
    X(:,j) = reshape(BZ_tensor(:,:,j),m*n,1);

end
X1 = X(:,1:end-1);
X2 = X(:,2:end);

slices = 1200;
t = linspace(0,1199,slices);
r = 200;

[Phi,omega,lambda,b,Xdmd,S] = DMD2(X1,X2,r,t);

fq1 = figure;
semilogy(abs(diag(S)),'ko');
title('Sigma')
grid on

fq2 = figure;
plot(omega,'ko')
title('Omega')
xlabel('Real')
ylabel('Imagine')
grid on
%%% Time-delay DMD
X1_q2 =[];
X2_q2 =[];
r_q2 = 200;
kk = 5;
for j=1:kk
    X1_q2 = [X1_q2;X(:,j:(k-1)-kk+j)];
    X2_q2 = [X2_q2;X(:,j+1:(k-1)-kk+j+1)];

end
[Phi_q2,omega_q2,lambda_q2,b_q2,Xdmd_q2,S_q2] = DMD2(X1_q2,X2_q2,r_q2,t);
fq3 = figure;
semilogy(abs(diag(S_q2)),'ko');
title('Sigma')
grid on

fq4 = figure;
plot(omega_q2,'ko')
title('Omega')
xlabel('Real')
ylabel('Imagine')
grid on
```


Listing 10: Code4 BZ data.