

Compression of the Koopman matrix for nonlinear physical models via hierarchical clustering

Tomoya Nishikata and Jun Ohkubo

Graduate School of Science and Engineering, Saitama University, Sakura, Saitama, 338–8570 Japan

Machine learning methods allow the prediction of nonlinear dynamical systems from data alone. The Koopman operator is one of them, which enables us to employ linear analysis for nonlinear dynamical systems. The linear characteristics of the Koopman operator are hopeful to understand the nonlinear dynamics and perform rapid predictions. The extended dynamic mode decomposition (EDMD) is one of the methods to approximate the Koopman operator as a finite-dimensional matrix. In this work, we propose a method to compress the Koopman matrix using hierarchical clustering. Numerical demonstrations for the cart-pole model and comparisons with the conventional singular value decomposition (SVD) are shown; the results indicate that the hierarchical clustering performs better than the naive SVD compressions.

I. INTRODUCTION

Physical simulation is used in many research fields, for example, to predict natural phenomena, design industrial robots, and simulate the behavior of objects in games. While we perform numerical simulations based on system equations, machine learning methods are also available for the physical simulations. The simulations based on machine learning do not require the system equations; only a training data set is enough to predict the system behavior. Many systems have nonlinearity, and methods based on neural networks are hopeful candidates for practical cases. Of course, the neural networks are also nonlinear, and the computational cost is high.

Recently, several works focused on the analysis based on the Koopman operator theory [1], in which we employ linear analysis even if the underlying system is nonlinear. Instead of the nonlinear computation for the original system equations for the state, we consider the observable space in the Koopman operator theory. The Koopman operator acts on the observable function linearly, and we can employ various methods in linear algebra. However, the observable space is a function space and infinite-dimensional. Hence, we must approximate the infinite-dimensional Koopman operator as a finite-dimensional Koopman matrix in practical computation. One of the methods to construct the Koopman matrix from data is the dynamic mode decomposition (DMD) [2]. An extension of the DMD is the extended dynamic mode decomposition (EDMD) [3], in which we introduce a dictionary and achieve more accurate predictions. There are some extensions of DMD and EDMD; DMDc (DMD with control) and EDMC (EDMD with control) [4] can deal with cases with external control inputs. Recently, the Koopman operators using the EDMD have been extensively studied for the prediction and control of physical models [5, 6]. Furthermore, various applications and studies of Koopman matrices have been discussed, for example, to analyze power systems [7, 8], reduce computational time in deriving Koopman matrices [9], add robustness against noise [10], and construct Koopman matrices using prior knowledge of the underlying model [11, 12]. For the Koopman operator theory, see the recent review [13].

While the Koopman operators are available for various purposes, there is a problem with practical usage; the size of the dictionary becomes large for higher-dimensional systems. Hence, the computational complexity is enormous. Of course,

one could construct the dictionary with the aid of neural networks; in [12], the authors employed not only the conventional basis functions but also the dictionary based on the neural networks. In [14], the dictionary learning with neural ordinary differential equations was discussed. However, it will be beneficial to avoid the usage of neural networks because of their high computational costs. In addition, robotics applications would avoid the black-box natures of neural networks. Hence, it is preferable to employ simple dictionary functions if possible.

Once one fixes the dictionary, we obtain the Koopman matrix explicitly. Then, it is also crucial to reduce the computational costs related to the Koopman matrix. In [12], the authors discussed the reduction of computational cost in the construction stage of the Koopman matrix. One can combine the prior knowledge of the underlying model with the Koopman matrix; in [12], it is possible to reduce the learning cost by making some elements of the Koopman matrix common. Although this approach reduces the computational costs for the learning process, the use of the Koopman matrix in the prediction stage was the same as the conventional ones in [12]. Then, how about the methods to reduce computational cost in the prediction steps?

In the present paper, we propose a method to compress the constructed Koopman matrix. The proposed method employs hierarchical clustering to extract similar rows and columns. Based on the clustering results, we perform the grouping of similar rows and columns, which compresses the Koopman matrix and the dictionary. Since the compressed Koopman matrix is not square, it is unavailable for calculating the time evolution repeatedly. Then, we introduce an additional matrix to recover the size. As a demonstration, we perform numerical experiments on a data set generated from a cart-pole model; we also compare the numerical results with a conventional matrix compression with the singular value decomposition (SVD).

The outline of the present paper is as follows. In Sec. 2, we briefly review the Koopman operator theory and the EDMD algorithm. The main proposal is yielded in Sec. 3; the compression method based on hierarchical clustering is explained. In Sec. 4, we numerically demonstrate the proposed method. As an example, a data set generated from the cart-pole model is used. Section 5 gives some concluding remarks.

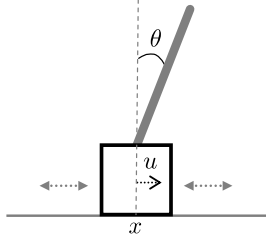


FIG. 1. Cart-pole model. The external force u only acts on the cart horizontally.

II. PRELIMINARIES

A. A concrete example of physical model

In this section, we briefly review the Koopman operator theory, the EDMD algorithm, and hierarchical clustering methods. It would be helpful to use a concrete example for readers' understanding, and then we sometimes use the cart-pole model as the concrete example.

Figure 1 shows the cart-pole model. The horizontal coordinate of the cart is x , and the cart can move only horizontally. The angle between the pole and the cart is θ . The corresponding velocities are denoted as $(\dot{x}, \dot{\theta})$. The state equation for the cart-pole model is nonlinear [15]; since only a data set is enough to apply the Koopman operator theory, we do not need to know the state equation here. There is an external force u that acts on the cart horizontally to control the pole to stand. However, the force does not appear in the following explanation because only a data set for the system states under proper control situations is employed. As explained in Sec. 4, we use the time-series data for the cart-pole model controlled by a reinforcement learning model.

B. Koopman operator and the EDMD algorithm

For the details of the EDMD algorithm, see [3]. We here only review the necessary parts of the algorithm.

Consider a nonlinear time evolution function $F : \mathbb{R}^{D_s} \rightarrow \mathbb{R}^{D_s}$ in a D_s -dimensional state space. The state at discrete time t is denoted as $\mathbf{x}_t$, and then $F(\mathbf{x}_t) = \mathbf{x}_{t+1}$. In the cart-pole model, $D_s = 4$ and the state is given as $\mathbf{x}_t = [x_t, \theta_t, \dot{x}_t, \dot{\theta}_t]$.

In the Koopman operator theory, we introduce an observable vector $\mathbf{g}(\mathbf{x})$ for the state $\mathbf{x}$. An observable, $g_i(\mathbf{x})$, is a function; for example, $g_1(\mathbf{x}) = \theta$ is a function to observe only the angle θ . As depicted in Fig. 2, a Koopman operator $\mathcal{K}$ is a linear operator that performs discrete-time evolution for the observable vector by linear computation. The discrete-time evolution equation using the Koopman operator is defined as follows:

$$\mathcal{K}\mathbf{g}(\mathbf{x}_t) = \mathbf{g} \circ F(\mathbf{x}_t) = \mathbf{g}(\mathbf{x}_{t+1}), \quad (1)$$

where $\circ$ means the composition of functions. The Koopman operator allows linear computation for the nonlinear time evolution of the original system.

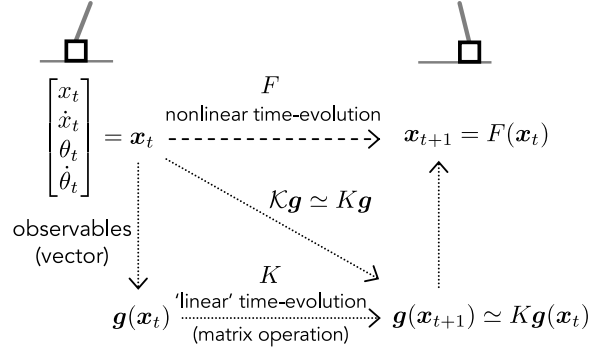


FIG. 2. Linear computation via the Koopman operator. Instead of the original nonlinear dynamics, we employ the linear operator $\mathcal{K}$ in the observable space. In practice, we estimate a finite-dimensional matrix to approximate the linear operator.

lution of the original system. Note that the observables are functions; hence, the Koopman operator acts on the corresponding function space. Since the function space is infinite-dimensional, we practically approximate the Koopman operator $\mathcal{K}$ as a finite-dimensional matrix, called a Koopman matrix K .

The EDMD algorithm is one of the algorithms to derive the finite-dimensional approximation of the Koopman operator from data [3]. The EDMD algorithm requires snapshot pairs of time series data and a dictionary. The snapshot pairs are s pairs of data before and after time evolution, i.e., $(X_1 = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_s])$ and $X_2 = [\mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_{s+1}]$; we here simply make the snapshot pairs from a single time-series data. The dictionary is a vector of functions. Let D be the size of the dictionary, i.e., the number of dictionary functions. Then, the dictionary $\boldsymbol{\psi}(\mathbf{x})$ is expressed as follows:

$$\boldsymbol{\psi}(\mathbf{x}) = [\psi_1(\mathbf{x}), \psi_2(\mathbf{x}), \dots, \psi_D(\mathbf{x})]^\top. \quad (2)$$

For example, monomial dictionary functions for the cart-pole model is written as

$$\begin{aligned} \psi_1(\mathbf{x}) &= 1, \psi_2(\mathbf{x}) = x, \psi_3(\mathbf{x}) = \theta, \psi_4(\mathbf{x}) = \dot{x}, \psi_5(\mathbf{x}) = \dot{\theta}, \\ \psi_6(\mathbf{x}) &= x^2, \psi_7(\mathbf{x}) = x\theta, \psi_8(\mathbf{x}) = x\dot{x}, \psi_9(\mathbf{x}) = x\dot{\theta}, \dots \end{aligned}$$

It is easy to see that the dictionary size D becomes large for high-dimensional systems. There are other dictionary functions, such as radial basis functions and neural network-based functions. However, the monomial dictionary functions are sometimes beneficial because of their simplicity and understandability. In the present paper, we employ the monomial dictionary functions.

An observable function $g_i(\mathbf{x})$ is expressed in terms of the dictionary $\boldsymbol{\psi}(\mathbf{x})$:

$$g_i(\mathbf{x}) = \sum_{d=1}^D a_{i,d} \psi_d(\mathbf{x}), \quad (3)$$

where $\{a_{i,d}\}$ are the expansion coefficients. Then, the action of

the Koopman operator on the observable function yields

$$\mathcal{K} \circ g_i(\mathbf{x}) = \sum_{d=1}^D a_{i,d} (\mathcal{K} \circ \psi_d(\mathbf{x})). \quad (4)$$

The action of $\mathcal{K}$ on the observable vector $\mathbf{g}$ is defined as the element-by-element action. Then, Eq. (4) allows us to derive the time evolution of the observable vector $\mathbf{g}(\mathbf{x})$ by applying the Koopman operator to the dictionary $\psi(\mathbf{x})$. Note that Eqs. (3) and (4) yield only approximations if the number of dictionary function is not enough.

The action of the Koopman operator $\mathcal{K}$ is approximated via the corresponding Koopman matrix K as follows:

$$\mathcal{K} \circ \psi(\mathbf{x}) \simeq K\psi(\mathbf{x}). \quad (5)$$

To obtain the Koopman matrix, the conventional least-squares method is available. From the snapshot pairs $\mathbf{X}_1, \mathbf{X}_2$ and the dictionary $\psi(\mathbf{x})$, we construct the data matrix $\Psi(\mathbf{X}_1) \in \mathbb{R}^{D \times s}$ and $\Psi(\mathbf{X}_2) \in \mathbb{R}^{D \times s}$. Then, the Koopman matrix is estimated via

$$K = \arg \min_{\tilde{K}} \|\Psi(\mathbf{X}_2) - \tilde{K}\Psi(\mathbf{X}_1)\|^2. \quad (6)$$

As derived in [3], the solution is given as

$$K = AG^+, \quad (7)$$

where

$$A = \frac{1}{s} \sum_{i=1}^s \psi(\mathbf{x}_i) \psi(\mathbf{x}_{i+1})^\top, \quad G = \frac{1}{s} \sum_{i=1}^s \psi(\mathbf{x}_i) \psi(\mathbf{x}_i)^\top, \quad (8)$$

and G^+ is the pseudo-inverse matrix of G .

C. Hierarchical clustering

Hierarchical clustering [16, 17] is one of the well-known clustering methods for unsupervised learning. To seek a hierarchy of clusters, we employ a bottom-up approach in this work. We merge the clusters with the smallest distance from each other and repeat this process. The stepwise classification yields a dichotomous tree called a dendrogram. An example of the dendrogram is shown in Fig. 3; there are five elements (or clusters) $[A, B, C, D, E]$ initially. The vertical axis in Fig. 3 represents the distance between clusters.

In hierarchical clustering, we first define a distance function, and all distances between clusters are evaluated. Then, we merge two clusters with the smallest distance. In this paper, we employ the Euclidean distance between a vector $\mathbf{y}$ and $\mathbf{z}$:

$$L_E = \sqrt{\sum_{i=1}^n (y_i - z_i)^2}. \quad (9)$$

After merging the clusters with the smallest distance, we update the distance between the merged cluster and the other

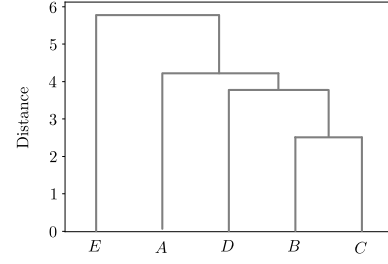


FIG. 3. An example of a dendrogram constructed from hierarchical clustering. Here, the five elements are clustered hierarchically according to distance.

clusters. There are several updating methods, and in this work, we employ the shortest distance method. For example, after merging clusters A and B , the updated formula for the distance to cluster C is defined as follows:

$$d_{[AB]C} = \min(d_{AC}, d_{BC}), \quad (10)$$

where d_{AC} and d_{BC} are the distances between clusters A and C , and that between clusters B and C , respectively. We update the distance between the merged cluster and other clusters for each merging process.

III. PROPOSED METHOD

As mentioned in Sec. 1, since the dictionary size is generally large, the Koopman matrix will also be huge. Therefore, it is desirable to compress the Koopman matrix. Here, we propose a compression method based on hierarchical clustering, which consists of four steps.

A. Step 1: Compress the Koopman matrix via hierarchical clustering

We consider the reduction of the Koopman matrix of Fig. 4 as an example. Hierarchical clustering of this matrix using the shortest distance method with Euclidean distance yields the dendrogram depicted in Fig. 5. We denote the N clusters in the row direction as $\mathbf{C}^r = [C_1^r, \dots, C_N^r]$ and M clusters in the column direction as $\mathbf{C}^c = [C_1^c, \dots, C_M^c]$. A cluster consists of some rows or columns; C_{ik}^r and C_{jk}^c are the k -th components of clusters in C_i^r and C_j^c , respectively. The clustering results in Fig. 5 are used to compress the Koopman matrix.

Figure 6 shows an example of a matrix in which we sort the rows and columns so that elements in the same clusters are adjacent to each other. The green block in the rows and the red one in the columns in Fig. 6 are examples of rows and columns belonging to the same cluster, respectively. The results of hierarchical clustering yield similar rows or columns. Hence, we assume that the rows or columns in the same clus-

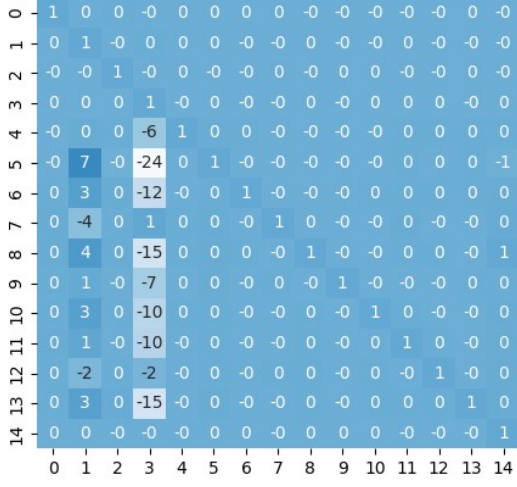


FIG. 4. An example of the original matrix for the clustering. Since the Koopman matrix is square, we start from this square matrix in the following explanation.

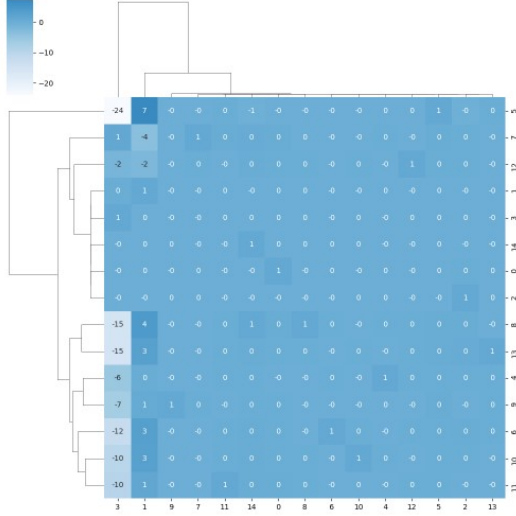


FIG. 5. Hierarchical clustering for the matrix. Rows and columns are hierarchically clustered separately.

ter have the same values. From this assumption, we set

$$K_{C_{i1}^r, j} = K_{C_{i2}^r, j} = \dots = K_{C_{i|C_i^r|}^r, j}, \quad (11)$$

$$K_{j, C_{i1}^c} = K_{j, C_{i2}^c} = \dots = K_{j, C_{i|C_i^c|}^c}. \quad (12)$$

From this assumption, we construct a compressed Koopman matrix by grouping the row-by-row and column-by-column elements in the same cluster. After the grouping procedure, we take the average of elements in the same cluster and set it as the compressed element. That is, the ij element of the

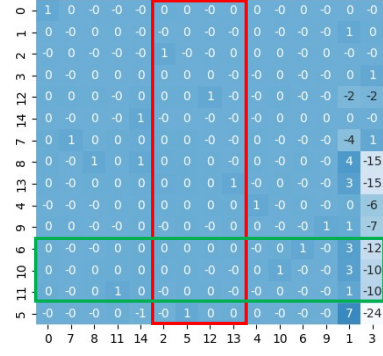


FIG. 6. An example of rearranged Koopman matrix by the hierarchical clustering results. The columns surrounded by the red block and the rows surrounded by the green block belong to the same cluster, respectively.

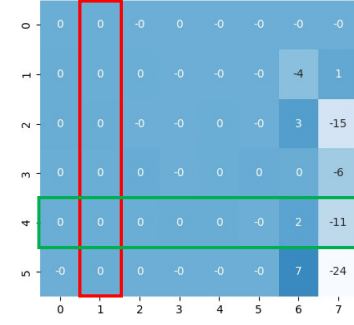


FIG. 7. An example of the compressed Koopman matrix based on the proposed method.

compressed Koopman matrix $K' \in \mathbb{R}^{N \times M}$ is evaluated as

$$K'_{ij} = \frac{\sum_{k=1}^{|C_i^r|} \sum_{l=1}^{|C_j^c|} K_{C_{ik}^r, C_{jl}^c}}{|C_i^r| |C_j^c|}. \quad (13)$$

As an example, let us consider the case with $N = 6$ clusters in the row direction and $M = 8$ clusters in the column direction. Then, the original Koopman matrix $K \in \mathbb{R}^{15 \times 15}$ is compressed as a matrix $K' \in \mathbb{R}^{6 \times 8}$. Figure 7 shows the compressed Koopman matrix. The green and red blocks in Fig. 6 correspond to them in Fig. 7, respectively.

Although the compressed Koopman matrix is derived, we should pay attention to the matrix size. The original Koopman matrix is square. Hence, it is possible to apply the Koopman matrix repeatedly:

$$\psi(x_{t+2}) = K\psi(x_{t+1}) = K^2\psi(x_t) \quad (14)$$

However, the compressed Koopman matrix K' is not square. Hence, we need different dictionaries for the compressed

Koopman matrix. To construct them, we here distinguish dictionaries before and after the matrix acts on them. Let $\psi_B(\mathbf{x}_t)$ and $\psi_A(\mathbf{x}_t)$ be the dictionaries before and after the action, respectively. Then,

$$K\psi_B(\mathbf{x}_t) = \psi_A(\mathbf{x}_t). \quad (15)$$

Note that $\psi(\mathbf{x}_{t+1}) = \psi_B(\mathbf{x}_{t+1}) = \psi_A(\mathbf{x}_t)$. In the following two steps, we construct two compressed dictionaries, respectively.

B. Step 2: Construct the dictionary after the action

As an example, we consider a 3×3 Koopman matrix. Then,

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} \psi_{B1}(\mathbf{x}_t) \\ \psi_{B2}(\mathbf{x}_t) \\ \psi_{B3}(\mathbf{x}_t) \end{bmatrix} = \begin{bmatrix} K_{11}\psi_{B1}(\mathbf{x}_t) + K_{12}\psi_{B2}(\mathbf{x}_t) + K_{13}\psi_{B3}(\mathbf{x}_t) \\ K_{21}\psi_{B1}(\mathbf{x}_t) + K_{22}\psi_{B2}(\mathbf{x}_t) + K_{23}\psi_{B3}(\mathbf{x}_t) \\ K_{31}\psi_{B1}(\mathbf{x}_t) + K_{32}\psi_{B2}(\mathbf{x}_t) + K_{33}\psi_{B3}(\mathbf{x}_t) \end{bmatrix} \quad (16)$$

$$= \begin{bmatrix} \psi_{A1}(\mathbf{x}_t) \\ \psi_{A2}(\mathbf{x}_t) \\ \psi_{A3}(\mathbf{x}_t) \end{bmatrix}. \quad (17)$$

Let us consider the case where the second and third rows are within the same group, i.e. $C^r = [[1], [2, 3]]$. Then, we construct the matrix elements K'_{2i} from K_{2i} and K_{3i} . Hence, we have

$$\begin{bmatrix} \psi'_{A1}(\mathbf{x}_t) \\ \psi'_{A2}(\mathbf{x}_t) \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K'_{21} & K'_{22} & K'_{23} \end{bmatrix} \begin{bmatrix} \psi_{B1}(\mathbf{x}_t) \\ \psi_{B2}(\mathbf{x}_t) \\ \psi_{B3}(\mathbf{x}_t) \end{bmatrix}. \quad (18)$$

We want to construct a compressed dictionary $\psi'_A(\mathbf{x})$ from $\psi_A(\mathbf{x})$. Note that the second row in Eq. (16) is a linear combination of $\{K_{2i}\}$ and $\{\psi_{B2}\}$. From the relation in Eq. (11), we have $K_{2i} = K_{3i}$. Hence, the second and third rows in Eq. (16) should also be equal. This fact leads to

$$\psi'_{A2}(\mathbf{x}_t) = \psi_{A2}(\mathbf{x}_t) = \psi_{A3}(\mathbf{x}_t). \quad (19)$$

Of course, the grouping procedure based on the clustering results does not give an exact equality. Hence, we approximate $\psi'_{A2}(\mathbf{x}_t)$ with the average:

$$\psi'_{A2}(\mathbf{x}_t) = \frac{1}{2} (\psi_{A2}(\mathbf{x}_t) + \psi_{A3}(\mathbf{x}_t)). \quad (20)$$

In summary, we define the compressed dictionary after the action as follows:

$$\psi'_{Ai}(\mathbf{x}_t) = \frac{1}{|C_j^r|} \sum_{k=1}^{|C_j^r|} \psi_{Ak}(\mathbf{x}_t). \quad (21)$$

C. Step 3: Construct the dictionary before the action

Again, we consider the example in Eq. (16) and Eq. (17). When we consider the case with $C^c = [[1], [2, 3]]$, the follow-

ing equation should be treated:

$$\begin{bmatrix} K_{11} & K'_{12} \\ K_{21} & K'_{22} \\ K_{31} & K'_{32} \end{bmatrix} \begin{bmatrix} \psi_{B1}(\mathbf{x}_t) \\ \psi'_{B2}(\mathbf{x}_t) \end{bmatrix} = \begin{bmatrix} \psi_{A1}(\mathbf{x}_t) \\ \psi_{A2}(\mathbf{x}_t) \\ \psi_{A3}(\mathbf{x}_t) \end{bmatrix}. \quad (22)$$

Here, we want to construct $\psi'_{B2}(\mathbf{x}_t)$ from $\psi_{B2}(\mathbf{x}_t)$ and $\psi_{B3}(\mathbf{x}_t)$. The left-hand side of Eq. (22) is calculated as

$$\begin{bmatrix} K_{11} & K'_{12} \\ K_{21} & K'_{22} \\ K_{31} & K'_{32} \end{bmatrix} \begin{bmatrix} \psi_{B1}(\mathbf{x}_t) \\ \psi'_{B2}(\mathbf{x}_t) \end{bmatrix} = \begin{bmatrix} K_{11}\psi_{B1}(\mathbf{x}_t) + K'_{12}\psi'_{B2}(\mathbf{x}_t) \\ K_{21}\psi_{B1}(\mathbf{x}_t) + K'_{22}\psi'_{B2}(\mathbf{x}_t) \\ K_{31}\psi_{B1}(\mathbf{x}_t) + K'_{32}\psi'_{B2}(\mathbf{x}_t) \end{bmatrix}, \quad (23)$$

and the comparison of the right-hand side of Eq. (23) and Eq. (16) leads to the following equations:

$$\begin{aligned} K_{j1}\psi_{B1}(\mathbf{x}_t) + K'_{j2}\psi'_{B2}(\mathbf{x}_t) \\ = K_{j1}\psi_{B1}(\mathbf{x}_t) + K_{j2}\psi_{B2}(\mathbf{x}_t) + K_{j3}\psi_{B3}(\mathbf{x}_t). \end{aligned} \quad (24)$$

Hence, we have

$$K'_{j2}\psi'_{B2}(\mathbf{x}_t) = K_{j2}\psi_{B2}(\mathbf{x}_t) + K_{j3}\psi_{B3}(\mathbf{x}_t). \quad (25)$$

Using Eq. (12) and Eq. (13), we have $K'_{j2} = K_{j2} = K_{j3}$. Then, we derive the following relationship:

$$\psi'_{B2}(\mathbf{x}_t) = \psi_{B2}(\mathbf{x}_t) + \psi_{B3}(\mathbf{x}_t). \quad (26)$$

In summary, we define the compressed dictionary before the action as follows:

$$\psi'_{Bj}(\mathbf{x}_t) = \sum_{k=1}^{|C_j^c|} \psi_{C_{jk}}(\mathbf{x}_t). \quad (27)$$

D. Step 4: Recover the suitable dictionary for the compressed Koopman matrix

The above two steps (Step 2 and Step 3) allow us to apply the compressed Koopman matrix to the compressed dictionaries:

$$K'\psi'_B(\mathbf{x}_t) = \psi'_A(\mathbf{x}_t). \quad (28)$$

Note that the two dictionaries $\psi'_B(\mathbf{x}_t)$ and $\psi'_A(\mathbf{x}_t)$ have different sizes. Then, we cannot apply K' repeatedly. Hence, we introduce an additional matrix R , which recovers the suitable size for the dictionary. Figure 8 shows an image of the procedure. The matrix R acts on $\Psi'_A(\mathbf{x})$ as follows:

$$R\psi'_A(\mathbf{x}_t) = \psi'_B(\mathbf{x}_{t+1}). \quad (29)$$

To explain the matrix R , we here use a 5×5 matrix as an example. Assume that $C^r = [[1, 5], [3], [2, 4]]$ in the row direction and $C^c = [[1, 2], [3, 4, 5]]$ in the column direction. Hence, we have

$$\begin{bmatrix} K'_{11} & K'_{12} \\ K'_{21} & K'_{22} \\ K'_{31} & K'_{32} \end{bmatrix} \begin{bmatrix} \psi'_{B1}(\mathbf{x}_t) \\ \psi'_{B2}(\mathbf{x}_t) \end{bmatrix} = \begin{bmatrix} \psi'_{A1}(\mathbf{x}_t) \\ \psi'_{A2}(\mathbf{x}_t) \\ \psi'_{A3}(\mathbf{x}_t) \end{bmatrix}. \quad (30)$$

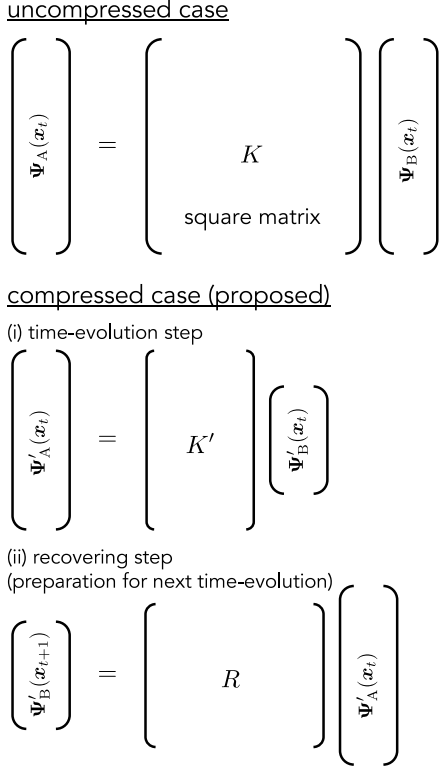


FIG. 8. Recovering step. The matrix R recovers the dictionary before the action, $\Psi_B(x)$, from the dictionary after the action, $\Psi_A(x)$.

Equation (27) leads to

$$\psi'_{B1}(x_t) = \psi_1(x_t) + \psi_2(x_t), \quad (31)$$

$$\psi'_{B2}(x_t) = \psi_3(x_t) + \psi_4(x_t) + \psi_5(x_t). \quad (32)$$

In addition, we here employ the simple equalities as in Eq. (19);

$$\psi'_{A1}(x_t) = \psi_1(x_{t+1}) = \psi_5(x_{t+1}), \quad (33)$$

$$\psi'_{A2}(x_t) = \psi_3(x_{t+1}), \quad (34)$$

$$\psi'_{A3}(x_t) = \psi_2(x_{t+1}) = \psi_4(x_{t+1}). \quad (35)$$

Since we want to satisfy Eq. (29), $\Psi_B(x_{t+1})$ must be connected to $\Psi_A(x_t)$. Then, we derive the following equations from the above relationships:

$$\psi'_{B1}(x_{t+1}) = \psi'_{A1}(x_t) + \psi'_{A3}(x_t), \quad (36)$$

$$\psi'_{B2}(x_{t+1}) = \psi'_{A2}(x_t) + \psi'_{A3}(x_t) + \psi'_{A1}(x_t), \quad (37)$$

which leads to

$$R = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \quad (38)$$

In summary, the matrix for the recovering step, R , is constructed as follows:

$$R_{ji} = \sum_{k=1}^{|C_i^r|} \sum_{l=1}^{|C_j^c|} \mathbb{I}[C_{ik}^r = C_{jl}^c] \quad (39)$$

where $\mathbb{I}$ is the indicator function, which returns 1 when the condition is satisfied and 0 if not.

E. Remark on the procedure

There are two ways for the usage of the matrix R ;

$$RK'\psi_B(x_t) = K'_B\psi_B(x_t) = \psi_B(x_{t+1}), \quad (40)$$

$$K'R\psi_A(x_t) = K'_A\psi_A(x_t) = \psi_A(x_{t+1}). \quad (41)$$

Equation (40) is a time evolution using the dictionary before the action, $\psi_B(x_t)$; Eq. (41) is that for $\psi_A(x_t)$. Note that these two procedures employ matrices with different size, i.e., $RK' = K'_B \in \mathbb{R}^{M \times M}$ and $K'R = K'_A \in \mathbb{R}^{N \times N}$.

IV. NUMERICAL EXPERIMENTS

In this section, we demonstrate the proposed method using the concrete physical model. We here use a little difficult model for the prediction tasks, i.e., the cart-pole model controlled by a reinforcement learning model. The cart-pole model aims to stand the pole only with force on the cart. The underlying system equations of the cart-pole model are non-linear. Of course, there is no need to know the underlying system equations; we only use a data set for the dynamics. Here, we construct the Koopman matrix by the EDMD algorithm and evaluate the features of the compressed Koopman matrix.

A. Data generation

We artificially generate the data for the cart-pole model. The cart-pole model is denoted in Sec. 2.1 and Fig. 1.

The details of the data generation is as follows. The initial state is set as $x_0 = [0, 0, 0, 0]$. The time-evolution is simulated with $\Delta t = 0.05$, and the final time is set as $T = 5$. Hence, we obtain a data set with 100 snapshot pairs from a single trajectory data. In the numerical experiments, we apply the control inputs obtained from the pre-trained Q-learning model so that the pole stands. This procedure is repeated 100 times to generate the training data set; we finally obtain a training data set with 10,000 snapshot pairs. Furthermore, we employ further 100 repeated simulations with the same settings to obtain the evaluation data set.

B. Definition of compression ratio

We define the compression ratio as the ratio of the row and column size of K' to those of K . The size of K' is $N \times M$, and that of K is $D \times D$. Hence, we denote the ratio as $[N/D, M/D]$. Note that the ratio $[1.0, 1.0]$ corresponds to the uncompressed case. We use the monomial dictionary functions with a total degree of up to 10, and then $D = 1,001$.

TABLE I. Average computation time per step [ms]. The rows and columns correspond to different row size ratios and column size ratios, respectively.

Row \ Column	1.0	0.8	0.6	0.4	0.2
1.0	0.524	0.529	0.558	0.490	0.243
0.8	0.404	0.392	0.349	0.345	0.143
0.6	0.192	0.176	0.151	0.141	0.123
0.4	0.120	0.098	0.080	0.098	0.116
0.2	0.072	0.057	0.106	0.060	0.063

In Sec. III E, we mentioned two types of formulas, i.e., Eq. (41) and Eq. (40). In the following numerical experiments, we employ Eq. (41). The dictionary before the action used in Eq. (40) is constructed from the sum of the original dictionary functions; see Eq. (26). Hence, we require accurate evaluations for all these functions. By contrast, the dictionary after the action used in Eq. (41) is based on the relation of equality; see Eq. (19). Hence, we expect the numerical evaluation of Eq. (41) to be more stable than that of Eq. (40). From some preliminary numerical experiments, we determined the use of Eq. (41).

C. Computation time

First, we compare the difference in computation time by compression of the Koopman matrix. Here, we output the average computation time per step for the evaluation data, i.e., 10,000 steps. The average computation time per step is shown in Table I. The row shows the row size ratio of the compressed Koopman matrix, and the column corresponds to the column size ratio.

Table I indicates that we can reduce the computation time largely by compressing in the row direction. The reason is that the row size ratio determines the size of the square matrix K'_A . Hence, the small row size ratio yields a small square matrix K'_A , and we achieve a large decrease in the computation time.

D. Prediction accuracy

While the computation time is reduced, we must confirm that the compressed Koopman matrix retains accuracy even with the reduction in dimensions. Here, we compare the differences between the predictions and the true values in the evaluation data set. Figure 9 shows the mean squared errors of the cart coordinate x and the pole angle θ for the original Koopman matrix and the compressed ones of size ratios $[0.8, 0.8]$, $[0.6, 0.6]$, $[0.4, 0.4]$, and $[0.2, 0.2]$. Note that the accuracy for the cart position x is not good even in the uncompressed original Koopman matrix. This is because the cart-pole model aims to stand the poles, which causes the cart to move significantly. Actually, the cart position largely varied in the trajectory data controlled by the reinforcement learning model. Hence, it is difficult to predict the position of the cart. By contrast, the pole angle variations are smaller due to

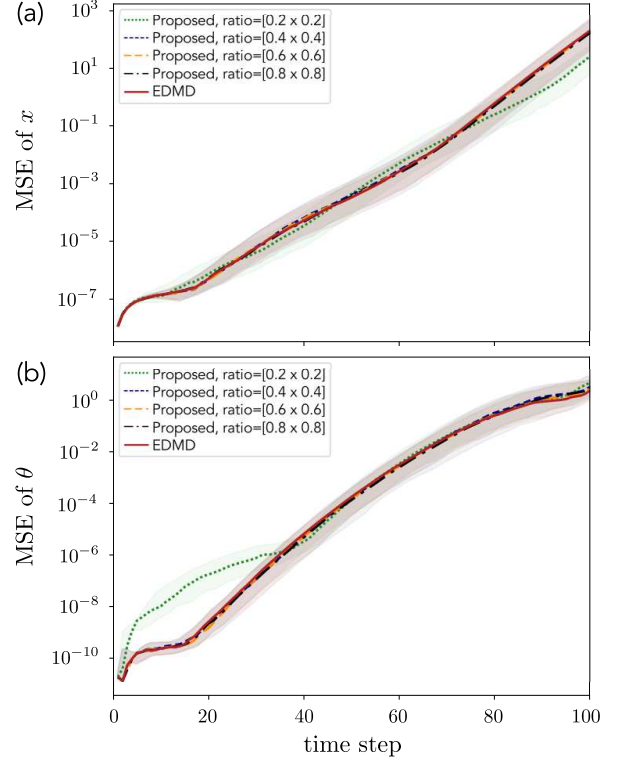


FIG. 9. Mean squared errors between the predictions and the true values in the evaluation data set. (a) Results for the cart coordinate x . (b) Results for the pole angle θ . The colored areas represent the corresponding quartiles.

the control, which leads to better predictive accuracy. From Fig. 9, one may consider that a long-term prediction is difficult. However, it is general to perform the control based on short-term prediction while observing the system state. Hence, this length of time will be sufficient. Of course, the aim here is not to make the control inputs; we investigate the effects of the compression of the Koopman matrix on the prediction accuracy. Then, it is enough to compare the accuracy of the proposed methods with the uncompressed original Koopman matrix.

The results in Fig. 9 indicate that the accuracies for the cases with $[0.8, 0.8]$, $[0.6, 0.6]$, $[0.4, 0.4]$ are comparable to that of the conventional method. For the case with $[0.2, 0.2]$, the results show different behaviors; for x , the mean squared error is smaller than that of the conventional method in the large time steps, while the result is worse for θ . Hence, the ratio $[0.2, 0.2]$ is not enough to recover the uncompressed results. From these results, we confirm that the proposed method works well even in large compression with $[0.4, 0.4]$. As we saw in Sec. 4.3, the $[0.4, 0.4]$ case yields a speedup of about 5 times.

TABLE II. The number of matrix elements in the proposed method.

Row size ratio	Number of matrix elements
0.8	640000
0.6	360000
0.4	160000
0.3	90000
0.2	40000

TABLE III. The total number of matrix elements for $K_{U\Sigma}^r$ and $(K_V^r)^\top$ in the SVD method.

Rank	Number of matrix elements
300	600600
200	400400
100	200200
50	100100
20	40040

E. Comparison with SVD

Here, we compare the proposed method with the SVD method, which leads to a low-rank approximation and the corresponding small memory size.

The usage of the SVD method for Koopman operators has already been discussed in [18]. However, the discussion focused on the reduction of computational complexity in the derivation of the Koopman matrix by the EDMD algorithm and the effect of noise in the data. Here, we focus on the SVD and the proposed method in terms of memory reduction of the Koopman matrix. Therefore, we compare the accuracy of the results of the SVD method with the results of the compressed Koopman matrix with the comparable memory size.

The SVD yields the following low-rank approximation with $r \leq D$ of the Koopman matrix:

$$K \simeq K_U^r K_\Sigma^r (K_V^r)^\top = K_{U\Sigma}^r (K_V^r)^\top, \quad (42)$$

where $K_U^r \in \mathbb{R}^{D \times r}$, $K_\Sigma^r \in \mathbb{R}^{r \times r}$, $K_V^r \in \mathbb{R}^{D \times r}$, and $K_U^r K_\Sigma^r = K_{U\Sigma}^r \in \mathbb{R}^{D \times r}$. By using K_V^r and $K_{U\Sigma}^r$, it is possible to construct the corresponding Koopman matrix with the memory-size reduction. Note that the condition $r < \frac{D}{2}$ is necessary to compress the matrix size more than the original matrix size.

The number of elements in the original Koopman matrix is 1,002,001. Table II shows the numbers of matrix elements of K_A^r for various settings. In Table III, we show the total number of matrix elements for $K_{U\Sigma}^r$ and $(K_V^r)^\top$. From these results, we compare the results for [0.2, 0.2] with the rank = 20 case and those for [0.3, 0.3] with the rank = 50 case.

Figure 10 shows the results. We see that the low-rank approximation via the SVD gives worse predictions than the proposed method. Hence, we conclude that the proposed method reduces the memory size while maintaining accuracy better than the low-rank approximation of the SVD.

These result suggests that the hierarchical structure could

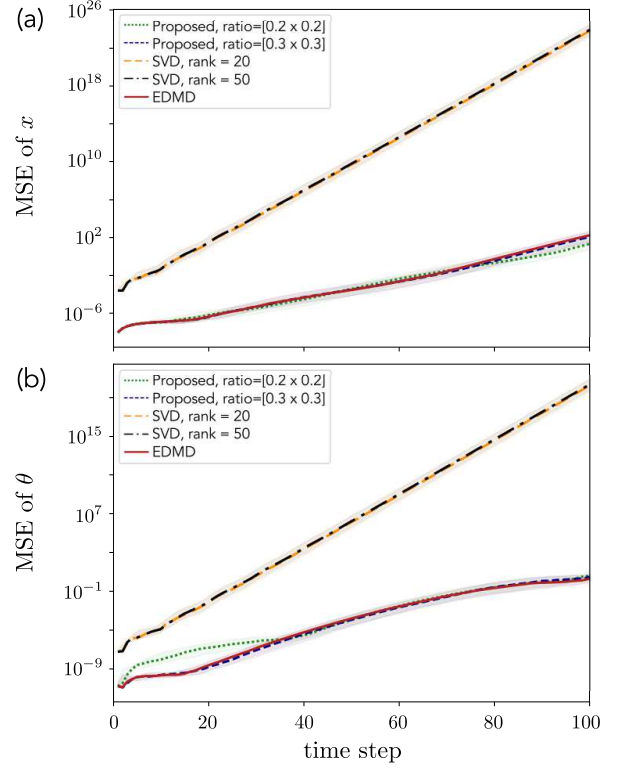


FIG. 10. Mean squared errors between the predictions and the true values in the evaluation data set. The results obtained by the SVD method are included. (a) Results for the cart coordinate x . (b) Results for the pole angle θ . The colored areas represent the corresponding quartiles.

extract and preserve some important structures for physical simulation models.

V. CONCLUSION

The large dictionary size yields the large Koopman matrix, which results in considerably high computational costs. To reduce the computational cost, we confirm that the compression method based on hierarchical clustering is more effective than the conventional SVD method. The proposed method based on hierarchical clustering could effectively employ the information embedded in the Koopman matrix.

Of course, the adequate size ratio of the compressed Koopman matrix could be different for other physical models and settings. In practice, we should consider methods to find the optimal size of the compressed Koopman matrices. In addition, we will pursue an interpretation of the results by hierarchical clustering from theoretical interests. At this stage, we cannot find a useful interpretation, and more extensive studies will be necessary in the future.

While the usage of the Koopman operator theory has been actively studied recently, there is room to discuss the features of the Koopman matrix. The present work is the first attempt

to apply hierarchical clustering to this research topic. We believe that this idea will be helpful for future research.

ACKNOWLEDGMENTS

This work was supported by JST FOREST Program (Grant Number JPMJFR216K, Japan).

-
- [1] B.O. Koopman, *Proc. Natl. Acad. Sci. USA*, vol. 17, pp. 315–318 (1931).
 - [2] P.J. Schmid, *J. Fluid Mech.*, vol. 656, pp. 5–28 (2010).
 - [3] M.O. Williams, I.G. Kevrekidis, and C.W. Rowley, *J. Nonlinear Sci.*, vol. 25, pp. 1307–1346 (2015).
 - [4] J.L. Proctor, S.L. Brunton, and J.N. Kutz, *SIAM J. Appl. Dyn. Syst.*, vol. 15, pp. 142–161 (2016).
 - [5] D. Bruder, X. Fu, R.B. Gillespie, C.D. Remy, and R. Vasudevan, *IEEE Trans. Robotics*, vol. 37, pp. 948–961 (2021).
 - [6] E. Kaiser, J.N. Kutz, and S.L. Brunton, *Machine Learning: Science and Technology*, vol. 2, article no. 035023 (2021).
 - [7] Y. Susuki, I. Mezić, F. Raak, and T. Hikihara, *NOLTA, IEICE*, vol. 7, pp. 430–459 (2016).
 - [8] K. Takamichi, Y. Susuki, M. Netto, and A. Ishigame, *NOLTA, IEICE*, vol. 13, pp. 409–414 (2022).
 - [9] M. Li and L. Jiang, arXiv:2204.13180.
 - [10] C. Dicle, H. Mansour, D. Tian, M. Benosman, and A. Vetro, *Proc. of 2016 ICME*, pp. 1–6 (2016).
 - [11] C. Folkestad, D. Pastor, I. Mezic, R. Mohr, M. Fonoberova, and J. Burdick, *Proc. of ACC 2020*, pp. 3906–3913 (2020).
 - [12] Y. Li, H. He, J. Wu, D. Katabi, and A. Torralba, *Proc. of ICLR 2020* (2019).
 - [13] S. L. Brunton, M. Budišić, E. Kaiser, and J. Nathan Kutz, *SIAM Rev.*, vol. 64, pp. 229–340 (2022).
 - [14] H. Terao, S. Shirasaka, and H. Suzuki, *NOLTA, IEICE*, vol. 12, pp. 626–638 (2021).
 - [15] A. Mills, A. Wills, and B. Ninness, *Proc. ACC 2009*, pp. 2335–2340 (2009).
 - [16] F. Murtagh and P. Contreras, *WIREs Data Mining and Knowledge Discovery*, vol. 2, pp. 86–97 (2012).
 - [17] F. Murtagh and P. Contreras, *WIREs Data Mining and Knowledge Discovery*, vol. 7, article no. e1219 (2017).
 - [18] S.T.M. Dawson, M.S. Hemati, M.O. Williams, and C.W. Rowley, *Experiments in Fluids*, vol. 57, article no. 42 (2016).