Data-Driven Encoding: A New Numerical Method for Computation of the Koopman Operator

Jerry Ng[1], H. Harry Asada[2], *Fellow, IEEE*

***Abstract*— This paper presents a data-driven method for constructing a Koopman linear model based on the Direct Encoding (DE) formula. The prevailing methods, Dynamic Mode Decomposition (DMD) and its extensions are based on least squares estimates that can be shown to be biased towards data that are densely populated. The DE formula consisting of inner products of a nonlinear state transition function with observable functions does not incur this biased estimation problem and thus serves as a desirable alternative to DMD. However, the original DE formula requires knowledge of the nonlinear state equation, which is not available in many practical applications. In this paper, the DE formula is extended to a data-driven method, Data-Driven Encoding (DDE) of Koopman operator, in which the inner products are calculated from data taken from a nonlinear dynamic system. An effective algorithm is presented for the computation of the inner products, and their convergence to true values is proven. Numerical experiments verify the effectiveness of DDE compared to Extended DMD. The experiments demonstrate robustness to data distribution and the convergent properties of DDE, guaranteeing accuracy improvements with additional sample points. Furthermore, DDE is applied to deep learning of the Koopman operator to further improve prediction accuracy.**

I. Introduction

Dynamic Mode Decomposition (DMD) was presented as a method to produce linear models from data generated through nonlinear dynamical processes by using Singular Value Decomposition (SVD) [1]. Later, this method was developed further to create Extended Dynamic Mode Decomposition (EDMD), which introduced the concept of using observable functions, nonlinear functions of state variables, as a method of augmenting the state space [2]. EDMD referenced the Koopman Operator as justification and a theoretical underpinning for lifting the state space. Decades prior, Bernard Koopman showed the existence of this operator that transforms nonlinear systems into linear systems [3]. Another extension of DMD has shown the viability of using DMD for control on non-autonomous systems [4]. This enabled complex nonlinear Model Predictive Control (MPC) to be converted to linear MPC [5], leading to numerous studies utilizing the methodology to real systems [6]–[9].

To improve the accuracy of the models based on the Koopman Operator, two avenues of research have formed. The first avenue regards the selection of the observable functions used for constructing a lifted state space, as these functions are a key ingredient in creating an accurate linear model. Various methods have been developed, including deep neural networks for learning effective observable functions [10]–[12] and optimization [13]. However, an efficient selection of observables does not solve all the issues that arise when attempting to construct an accurate linear model. The second avenue of research addressed the proper formulation of the linear transition matrix. It is known that unstable modes are involved in Koopman-based DMD models and their extensions although the underlying nonlinear systems are known to be stable [14]. Extensive studies have been done to create stable linear models to remedy the situations where an outright use of DMD would lead to the creation of an unstable linear model [14]–[17]. Recently, an extension of DMD, called Robust Dynamic Mode Decomposition (RDMD), utilizes statistical measures to suppress the effect of outliers on modeling the linear Koopman matrix [18].

A fundamental difficulty in constructing a proper linear model is data dependency. Least Squares Estimation (LSE), involved in all DMD based methods, often produces a significant bias, an over weighting of samples. This bias is due to the distribution, how prevalent similar data points are, of the dataset. To eliminate this dependency on distribution, the current work takes an alternative approach to LSE.

Recently, a new formulation of the Koopman Operator, termed Koopman Direct Encoding (DE), was produced [19]. This method directly encodes the nonlinear dynamics into the lifted linear model. Inner products of observable functions in composition with the nonlinear state transition function are used to construct the state transition matrix without use of LSE. While DE theoretically guarantees the exact linear model, it requires access to the nonlinear state equations, which are often not available in practical applications. The current work aims to fill the gap between DE and data-driven approaches.

There are four significant contributions presented in this work. The first is the conversion of the DE formula of the Koopman Operator to a data-driven formula. The second is a computational algorithm and proof of its convergence to the true inner products that constitute the DE formula. The third is numerical experiments that provide evidence that the proposed method, unlike EDMD, does not exhibit biases to data distribution, but can produce consistently higher accuracy compared to EDMD. Finally, the DDE algorithm is utilized in modeling a high order nonlinear system in combination with deep learning.

This material is based upon work supported by National Science Foundation Grant NSF-CMMI 2021625.

[1]Jerry Ng is with Department of Mechanical Engineering at the Massachusetts Institute of Technology jerryng@mit.edu

[2]H. Harry Asada is with Department of Mechanical Engineering at the Massachusetts Institute of Technology asada@mit.edu

Code and additional explanations of experiments for this work can be found on https://github.com/jerrying123/DDE

II. Koopman Operator and the Direct Encoding Method

In this section, we give a brief overview of the Koopman Operator and dynamic mode decomposition [2], and introduce the direct encoding method for obtaining a Koopman Operator directly from nonlinear dynamics [19].

A. Least Squares Estimation of the Koopman Operator

Consider a discrete-time dynamical system, given by

$$x_{t+1} = f(x_t) \tag{1}$$

where $x \in X \subset \mathbb{R}^n$ is the independent state variable vector representing the dynamic state of the system, f is a self-map, nonlinear function $f : X \to X$, and t is the current time step. Also consider a real-valued observable function of the state variables $g : X \to \mathbb{R}$. The Koopman Operator $\mathcal{K}$ is an infinite-dimensional linear operator acting on the observable function g :

$$\mathcal{K}g = g \circ f \tag{2}$$

where $g \circ f$ is the composition of function g with function f: $(g \circ f)(x) = g(f(x))$.

A common data-driven method for constructing the operator is Extended Dynamic Mode Decomposition (EDMD) [2], where observables that are experimentally obtained or simulated from the governing equation of the system are augmented by including real-valued observable functions of the independent state vector x_t. This collection of observables, z_t is

$$z_t = \begin{bmatrix} g_1(x_t) \\ g_2(x_t) \\ \vdots \\ g_m(x_t) \end{bmatrix} \tag{3}$$

where m is the order of the lifted state corresponding to the number of observable functions. Underpinned by the Koopman Operator theory, EDMD assumes the existence of a linear state transition matrix A relating z_{t+1} to z_t, and determine A by solving a least squares regression that minimizes the Sum of Squared Error (SSE) through

$$A^o = \arg\min_A \sum_t ||z_{t+1} - Az_t||^2 \tag{4}$$

Singular Value Decomposition (SVD) is used for the least squares optimization.

B. Direct Encoding of the Koopman Operator

An alternative to the least squares estimate and EDMD is to obtain the exact A matrix by directly encoding the self-map, nonlinear state transition function $f(x)$ with an independent and complete set of observable functions through inner product computations. This Direct Encoding method is introduced next, while the full proof can be found in [19].

Let us first consider the case where $g_1, g_2, g_3, ...$ are orthonormal basis functions spanning a Hilbert space $\mathcal{H}$. We assume that the self-map nonlinear function $f(x)$ is continuous and that the composition of g_j with f is also involved in the Hilbert space.

$$g_j \circ f \in \mathcal{H} \quad \forall j \tag{5}$$

This implies that the function $g_j \circ f$ can be expanded in $[g_1, g_2, g_3, ...]$.

$$g_j \circ f = \sum_{k=1}^{\infty} \langle g_j \circ f, g_k \rangle g_k \tag{6}$$

Concatenating $g_1, g_2, g_3, ...$ and $g_1 \circ f, g_2 \circ f, g_3 \circ f...$ in infinite dimensional column vectors, respectively,

$$\bar{z}_t = \begin{bmatrix} g_1(x_t) \\ g_2(x_t) \\ \vdots \end{bmatrix}, \bar{z}_{t+1} = \begin{bmatrix} g_1[f(x_t)] \\ g_2[f(x_t)] \\ \vdots \end{bmatrix} \tag{7}$$

eq. (6) can be written in matrix and vector form.

$$\bar{z}_{t+1} = \bar{A}\bar{z}_t \tag{8}$$

where $\bar{A}$ is an infinite dimensional matrix consisting of the inner products involved in eq. (6),

$$\bar{A} = \begin{bmatrix} \langle g_1 \circ f, g_1 \rangle & \langle g_1 \circ f, g_2 \rangle & \cdots \\ \langle g_2 \circ f, g_1 \rangle & \langle g_2 \circ f, g_2 \rangle & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{9}$$

Eq. (8) manifests that the state lifted to the infinite dimensional space $\bar{z}_t$ makes linear state transition with matrix $\bar{A}$.

The observables $g_1, g_2, g_3, ...$ were assumed to be orthonormal basis functions in the above derivation. This assumption can be relaxed to an independent and complete set of basis functions spanning the Hilbert space. Hereafter, let $[g_1, g_2, g_3, ...]$ be an independent and complete set of basis functions spanning the Hilbert space.

It can be shown that the time evolution of lifted state z_t is given by a constant matrix A_f for the independent and complete set of basis functions $[g_1, g_2, g_3, ...]$.

$$z_{t+1} = A_f z_t \tag{10}$$

The matrix A_f can be computed directly from the self-map, state transition function $f(x)$ and an independent and complete set of observables $[g_1, g_2, g_3, ...]$ through inner product computations. Post-multiplying the transpose of z_t to both sides of eq. (10) and integrating them over X yield:

$$\int_X z(f(x))z^T(x)dx = A_f \int_X z(x)z^T(x)dx \tag{11}$$

which can be written as

$$Q = A_f R \tag{12}$$

where

$$Q = \begin{bmatrix} \langle g_1 \circ f, g_1 \rangle & \langle g_1 \circ f, g_2 \rangle & \cdots \\ \langle g_2 \circ f, g_1 \rangle & \langle g_2 \circ f, g_2 \rangle & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{13}$$

$$R = \begin{bmatrix} \langle g_1, g_1 \rangle & \langle g_1, g_2 \rangle & \cdots \\ \langle g_2, g_1 \rangle & \langle g_2, g_2 \rangle & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{14}$$

Because the observables are independent, the matrix R is non-singular. Therefore, the matrix A_f is given by

$$A_f = QR^{-1} \tag{15}$$

This formula for obtaining the matrix A_f directly from the governing nonlinear state equation with the function $f(x)$ and the independent observables through inner products, which are guaranteed to exist in Hilbert space $\mathcal{H}$, is the Direct Encoding method.

III. Data-Driven Koopman Encoding

The prevailing method for construction of the Koopman Operator, EDMD, is based on LSE and SVD. This method, however, cannot provide an unbiased estimate; the result is biased - dependent on the distribution of data within a dataset- as the Koopman Operator is being approximated [20]. This dependency on distribution of a dataset occurs because a core assumption of LSE is that the model structure is correct; when this assumption is violated, LSE is unable to create an unbiased estimator [21]. As the Koopman Operator is truncated in practical use, this assumption does not hold.

Non-uniform data distributions, that is datasets where distances between data points in the state space are not equidistant from their nearest neighbors, inevitably occur in practical applications. For a nonlinear dynamical system with a stable equilibrium, for example, data collected from experiments and/or simulation of the system tend to be dense in the vicinity of the equilibrium, as all trajectories that begin within a region of attraction converge to the equilibrium. Because LSE applies equal weighting to all data points, the model is heavily tuned to the behavior of the densely populated region.

The Direct Encoding method described previously enables us to obtain the exact linear state transition matrix A through inner product computations. As the formulation is based on integration over the entire state space, there is no bias towards particular parts of the domain.

However, the original form of the Direct Encoding method utilizes the nonlinear state equation, i.e. the self-map $f(x)$, to compute the inner products. In practical applications, such a nonlinear function is not always available; only data are available. The objective of this section is to establish a computational algorithm to obtain the A matrix by numerically computing the inner products, $\langle g_i, g_j \rangle, \langle g_i \circ f, g_j \rangle$, from a given set of data.

The method presented consists of three operations.

- The integral of the inner products is reduced in range from the entire state space to the dynamic range encapsulated by the data.
- The dynamic range is discretized with data points.
- The inner product integral is reduced to a weighted summation of the integrand evaluated at each data point multiplied by the volume Δv associated to each point.

Naturally, if data are densely populated in a small region, the discretized integral interval is small and thereby the volume also becomes small. Similarly, the volume tends to be larger where the data are sparse. In the summation, the integrand evaluated at individual data points are "weighted" by the size of the volume. This numerical inner product calculation prevents overemphasis of clustered data.

A. Inner Product Computation

We present the data-driven encoding method (DDE) as an alternative data-driven method to DMD for calculating a finite order approximation of the Koopman Operator. The objective of this method is to compute the matrices R and Q in (13) and (14) from data. This entails the computation of inner products:

$$\langle g_i, g_j \rangle = \int_X G_{ij}(\xi) d\xi \tag{16}$$

$$\langle g_i \circ f, g_j \rangle = \int_X F_{ij}(\xi) d\xi \tag{17}$$

where

$$G_{ij}(x) = g_i(x)\bar{g}_j(x), F_{ij}(x) = g_i[f(x)]\bar{g}_j(x) \tag{18}$$

are assumed to be Riemann Integrable; the functions are bounded and continuous [22].

There are two data sets used for the inner product computation. The first data set is

$$D_N = \{x_i \mid i = 1, \cdots, N; x_i \in X\} \tag{19}$$

Note that all the data values are finite, $|x_i| < \infty$. As such, the integral interval of the inner products is finite in computing them from the data. To define the integral interval, we consider the dynamic range of the system, X_D, determined from the data set D_N. See Fig.1. The dynamic range X_D is defined to be the minimum domain in the space X that includes all the data points in D_N, $X_D \supset D_N$, and that is convex. Namely, for any two states in X_D, $x, x' \in X_D$,

$$\xi = \alpha x + (1 - \alpha)x' \in X_D \tag{20}$$

where $0 \leq \alpha \leq 1$. Each data point x_i is mapped to $f(x_i)$,

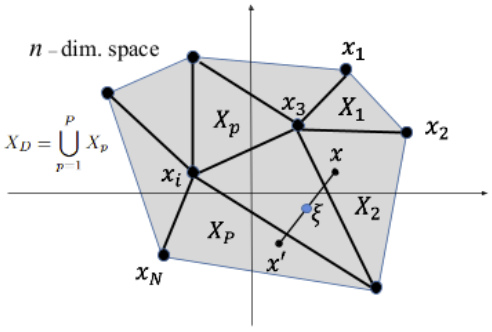

Fig. 1: Illustration of the dynamic range of a dataset defined by the convex hull containing all points in the set, partitioned using a triangulation method. The data points are in black and the convex hull that encapsulates all data points is in grey.

following the state transition law in eq.(1). We assume that the transferred state, too, stays within the same dynamic

range X_D. Collecting all the transferred states yields the second data set.

$$D_N^f = \{f(x_i) \mid i = 1, \cdots, N; x_i \in D_N\} \tag{21}$$

$$D_N^f \subset X_D \tag{22}$$

This implies that the state space of the nonlinear system under consideration is closed within the dynamic range X_D.

With this dynamic range, we redefine our objective to compute the inner products over X_D.

$$R_{ij} = \int_{X_D} G_{ij}(x)dx \tag{23}$$

$$Q_{ij} = \int_{X_D} F_{ij}(x)dx \tag{24}$$

The integrals can be computed by partitioning the domain X_D into many segments $X_1, \cdots X_P$, as shown in Fig. 1.

$$X_D = \bigcup_{p=1}^{P} X_p \tag{25}$$

We generate these segments by applying a meshing technique to the data set D_N, where the n-dimensional coordinates of individual data points are treated as nodes of a mesh. Delaunay Triangulation, for example, generates a triangular mesh structure with desirable properties [23]. As illustrated in Fig. 1, each triangular element is convex and has no internal node. The volume of the dynamic range $V(X_D)$ is the sum of the volumes of all the elements.

$$V(X_D) = \sum_{p=1}^{P} \Delta v_p \tag{26}$$

Accordingly, the integral R_{ij} in eq.(23) can be segmented to

$$R_{ij} = \sum_{p=1}^{P} \int_{X_p} G_{ij}(x)dx \tag{27}$$

Suppose that the p-th element has K_p nodes, as shown in Fig.2. Renumbering these nodes 1 through K_p,

$$\{x[k_p] \Big| x[k_p] \in X_p; k_p = 1, \cdots, K_p\}, p = 1, \cdots, P \tag{28}$$

The integrand G_{ij} within the p-th element can be approximated to the mean of the K_p nodes involved in the p-th element.

$$G_{ij}(x; p) \approx \bar{G}_{ij,p} = \frac{1}{K_p} \sum_{k_p=1}^{K_p} G_{ij}(x[k_p]), x[k_p] \in X_p \tag{29}$$

If Delaunay Triangulation is used, $K_p = n + 1$. See Fig. 2. Substituting this into (23) yields the approximate value of R_{ij}.

$$\hat{R}_{ij} = \sum_{p=1}^{P} \frac{1}{K_p} \sum_{k_p=1}^{K_p} G_{ij}(x[k_p])\Delta v_p \tag{30}$$

where

$$\Delta v_p = \int_{X_p} 1 \cdot dx \tag{31}$$

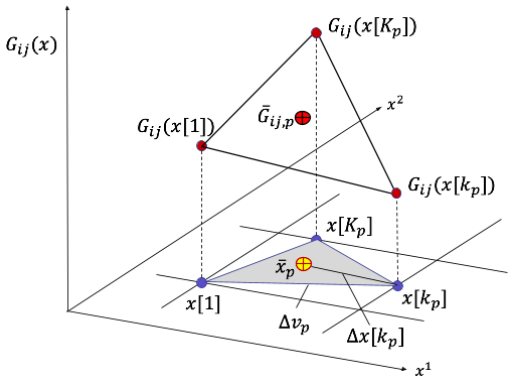

Fig. 2: Visualization of the integrand calculation process. The volume of the partition is encapsulated by the data points is denoted as Δv_p. With this grouping, the value of G_{ij} is calculated for each point and the average among this group is computed, $\bar{G}_{ij,p}$. In turn, this value, weighted by the volume of this partition, is summed across other partitions (not shown) to approximate the value of the element R_{ij}.

Similarly, each component of the matrix Q can be computed by using the same meshing.

$$\hat{Q}_{ij} = \sum_{p=1}^{P} \frac{1}{K_p} \sum_{k_p=1}^{K_p} F_{ij}(x[k_p])\Delta v_p \tag{32}$$

Note that F_{ij} is evaluated by using the data points in both D_N^f and D_N,

$$F_{ij}(x[k_p]) = g_i[f(x[k_p])]\bar{g}_j(x[k_p]) \tag{33}$$

where $f(x[k_p]) \in D_N^f$, thus not requiring evaluation of the nonlinear function f.

B. Convergence

Consider the center of each partition, $\bar{x}_p = \int_{X_p} xdx/\Delta v_p$, and the distance between $\bar{x}_p$ and each point, $x[k_p]$:

$$\Delta x[k_p] = \bar{x}_p - x[k_p] \tag{34}$$

See Fig.2. The maximum distance from the center of the partition to each point that makes up the partition is

$$|\Delta x_p| = \max\{|\Delta x[1]|, \ldots |\Delta x[k_p - 1]|, |\Delta x[k_p]|\} \tag{35}$$

Consider a sequence of refining the approximate inner product integral $\hat{R}_{ij}$ by increasing data points N. We can show that, as the number of partition P tends infinity and the maximum subintervals $|\Delta x_p|$ approach zero, the approximate inner product integral $\hat{R}_{ij}$ converges to its true integral.

$$R_{ij} = \lim_{\substack{P\to\infty \\ |\Delta x_p|\to 0}} \sum_{p=1}^{P} \frac{1}{K_p} \sum_{k_p=1}^{K_p} G_{ij}(x[k_p])\Delta v_p \tag{36}$$

This formulation takes the form of weighted sums, specifically Riemann sums. Given functions that are bounded and continuous over the subdomain of interest, sequences of this form are known to have a common limit and thus converge upon refinement to the Riemann integral value over that subdomain, according to Numerical Integration theory [22, Section 1.5].

C. Algorithm

In the prior section, integrals (30) and (32) are presented as summations over partitions. This computation can be streamlined by converting the summations over partitions to the one over nodes. Consider node 3 associated to data point x_3 in Fig.1, for example. This node is an apex of the 5 surrounding triangles. This implies that integrand $G_{ij}(x)$ is calculated or recalled 5 times in computing (30) and (32). This repetition can be eliminated by computing volume Δv_k associated to node k, rather than partition $p : \Delta v_p$. Namely, we compute

$$\Delta v_k = \sum_{p=1}^{P} \frac{\Delta v_p}{K_p} I(k,p) \tag{37}$$

where $I(k,p)$ is a membership function that takes value 1 when node k is an apex of triangle p, that is, node k is involved in partition p. Using this volume as a new weight we can rewrite (30) and (32) to be

$$\hat{R}_{ij} = \sum_{k=1}^{K} G_{ij}(x[k]) \Delta v_k \tag{38}$$

$$\hat{Q}_{ij} = \sum_{k=1}^{K} F_{ij}(x[k]) \Delta v_k \tag{39}$$

Using this conversion, the computation can be streamlined and cleanly separated into three steps, as shown by pseudocode in Algorithm 1. The steps are: (1) *Graph Creation:* data are connected to create partitions of the domain using a mesh generator: lines 3 to 8, (2) *Weighting Calculation:* calculation of the weights for each data point: lines 10 to 17, and (3) *Matrix Calculation:* the calculation of the R and Q matrices to find the matrix A, lines 19 to 21.

In comparison to EDMD, this algorithm is notably slower. In terms of time complexity, EDMD uses Singular Value Decomposition (SVD) which is $\mathcal{O}(mn^2)$, while the current method of graph creation for DDE, Delaunay Triangulation, is $\mathcal{O}(m^{n/2})$, where n is the dimensions of the space and m is the number of points [24] [25]. In addition, as DDE is based on numerical integration, issues arise when the underlying nonlinear system becomes significantly high order.

IV. Experiments

In this section, the DDE algorithm is implemented for the sake of evaluating its validity and comparing its modeling accuracy to EDMD. Consider a 2nd order nonlinear system consisting of a pendulum with a nonlinear damper. See Fig. 3. The pendulum also bounces against walls with nonlinear compliance. The state variables for this system are $x = [\theta, \dot{\theta}]^T$, and the equation of motion can be written as:

$$\ddot{\theta} = -sin(\theta) + F_k + F_c \tag{40}$$

where F_k and F_c are wall reaction moment and damping moment, respectively,

$$F_k = \begin{cases} -\text{sign}(\theta)\, k(|\theta| - \frac{\pi}{4})^2 & \text{if } |\theta| \geq \frac{\pi}{4} \\ 0 & \text{if } |\theta| < \frac{\pi}{4} \end{cases} \tag{41}$$

$$F_c = -\text{sign}(\dot{\theta})\, c\, \dot{\theta}^2 \tag{42}$$

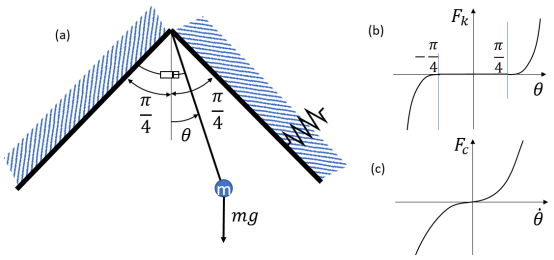

Fig. 3: Diagram of pendulum with with walls. (a) depicts the range of the pendulum where the walls are equally angularly displaced from the vertical. (b) depicts the forces exerted on the pendulum due to contact with walls. (c) depicts the damping force exerted onto the pendulum.

where $k = 200$ and $c = 1$. We present a two part numerical experiment for this system. The first experiment regards variations in dataset size and distribution, and the second experiment varies the usage of observable functions.

A. Dataset Variations

The datasets tested are of three types:

1) **Uniform**: These datasets are composed of a rectangular dynamic range which is sampled uniformly, like an evenly divided grid. The range varies from $\theta = [-0.8, 0.8]$ and $\dot{\theta} = [-2, 2]$, where the mass can hit the

Algorithm 1: Algorithm for DDE in pseudocode

Input:
1 D_N, D_N^f
Output:
2 P: p, K: k, R, Q A
3 *Graph Creation:*
4 **for** x_i *in* D_N *and corresponding* $f(x_i)$ *in* D_N^f **do**
5 Create node, k
6 Assign node attributes for current data point $k.x_t = x_t$ and $k.x_{t+1} = x_{t+1}$
7 Append node to node list K.
8 **end**
9 **then**
10 *Weighting Calculation:*
11 Create list of triangles, P using Delaunay Triangulation on node list K using $k.x_t$
12 **for** p *in* P **do**
13 Calculate volume, V, of p
14 **for** k *corresponding to* K_p **do**
15 Update volume of each node $k.\Delta v_k = k.\Delta v_k + \frac{V}{n+1}$
16 **end**
17 **end**
18 **then**
19 *Matrix Calculation:*
20 Find R and Q via (38) and (39)
21 Find $A = QR^{-1}$

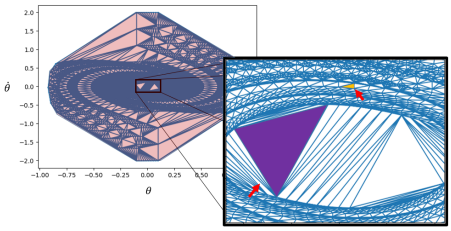

Fig. 4: Graph of connections for a trajectory dataset composed of 10000 points formed when utilizing the data-driven direct encoding method. The lightly red shaded region denotes the dynamic range. In the zoomed-in image, the difference in volumes associated to different data points can be observed. Noting the difference in size of the purple triangle versus the yellow triangle.

TABLE I: Sum of Squared Errors over dynamic range with varying dataset sizes.

Dataset Size	Total SSE EDMD / DDE	SSE Variance EDMD / DDE
Uniform Datasets		
900	19.470 / **17.167**	**0.0094** / 0.0097
2500	17.995 / **16.471**	**0.0095** / 0.0103
10000	17.010 / **16.184**	**0.0099** / 0.0105
22500	16.698 / **16.133**	**0.0101** / 0.0105
Trajectory Datasets		
1000	56.532 / **33.392**	0.0349 / **0.0193**
2500	33.330 / **25.064**	0.0200 / **0.0130**
5000	31.690 / **25.099**	0.0195 / **0.0129**
10000	30.184 / **25.101**	0.0186 / **0.0129**
25000	29.380 / **25.106**	0.0181 / **0.0129**

walls and the damping can vary from 0 to a significant value. See Fig. 3-(b), (c).

2) **Gaussian**: Data points are sampled with a finite-support Gaussian distribution. The data are distributed non-uniformly with their highest density at the peak of the Gaussian placed at diverse locations. In addition, each dataset contains 100 data points uniformly distributed along the border of the dynamic range to guarantee the same dynamic range as the uniform datasets. Samples outside the dynamic range are excluded.
3) **Trajectories**: These datasets are composed of trajectories, beginning from 100 initial conditions that are simulated forward the same number of time steps. The dynamic range of this dataset differs from the two other dataset types.

The models constructed for DDE and EDMD use the same observable functions. The observable functions chosen are two dimensional radial basis functions (RBFs), uniformly distributed between the maximum and minimum values of each state variable in their respective dataset, and the state variables. The total order of the system is 27th order with 25 RBFs and 2 state variables.

A trajectory dataset graph is generated using Delaunay Triangles in DDE, shown in Fig. 4.

The accuracy of the models is tested through calculating sum of squared errors (SSE) for one-step ahead predictions

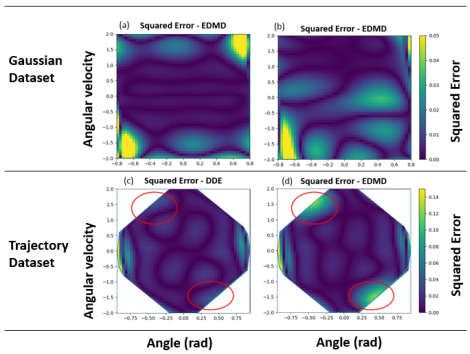

Fig. 5: Sum of Squared Error plots for various datasets. (a) and (b) are EDMD models for Gaussian datasets; (a) uses a dataset that is centered at $[0, 0]$ and (b) uses a dataset that is centered at $[0, 2]$. (c) DDE and (d) EDMD models using the trajectory dataset composed of 2500 data points and using 25 RBFs. Circled in red are the regions with greatest variation in SSE between models. Only the dynamic range is shown in all plots.

TABLE II: Sum of Squared Errors over dynamic range for Gaussian distributions with different centers. Centers of the distribution are noted above each column.

Dataset Size	Total SSE Center: $[0, 0]$ EDMD / DDE	Center: $[0.8, 0]$ EDMD / DDE	Center: $[0, 2]$ EDMD / DDE
Gaussian Datasets			
1000	25.565 / **24.377**	25.161 / **22.98**	24.338 / **22.445**
2500	24.991 / **23.739**	25.421 / **22.747**	24.221 / **21.590**
5000	24.662 / **23.518**	26.805 / **22.415**	23.914 / **21.195**
10000	24.395 / **23.085**	28.424 / **21.825**	25.770 / **21.052**
25000	25.219 / **22.167**	30.788 / **21.687**	26.941 / **20.704**

over the dynamic range of the datasets. These error values are calculated for a uniform grid of points, similar to that used in the uniform datasets. A visualization of the SSE is plotted in Fig. 5. The results of these calculations are shown in Table I and II. In the computation, the dynamic range is discretized, and the SSE value of each point is summed. For the Gaussian datasets, the test is run for eight iterations of each dataset to account for randomness and the average result is noted.

TABLE III: Sum of Squared Errors over dynamic range with varying order of lifted linear models.

# Observables	Total SSE EDMD	DDE
Trajectory Dataset, 5000 points		
27	31.690	**25.099**
51	36.657	**21.637**
83	28.437	**13.613**

B. Observable Function Variation

The second experiment varies the number of observable functions selected, thus increasing the order. In this experiment, the number of RBFs is varied through uniformly increasing the density of the centers of the function over

the range of the dataset. The results are noted in Table III. In the table, the number detailing number of observables is the number of functions including the state variables.

C. Discussion

From these results we can make the following observations.

- All the numerical experiments show that DDE outperforms EDMD in total SSE.
- For uniform datasets, both models are nearly equivalent, though DDE has slightly lower SSE in all cases, shown in Table I. This result is expected as all data points are weighted equally in a uniform distribution.
- EDMD models exhibit significantly different distributions of prediction error, depending on dataset distribution. In Fig. 5-(a), the EDMD model learned from a dataset with high density near the origin produces a prediction error distribution that is low in accuracy in the top-right and the bottom-left corners of the dynamic range. In contrast, Fig.5-(b) illustrates that when using EDMD to learn from data with high density at $\theta = 0, \dot{\theta} = 2$, the model results in low accuracy in the lower half of the dynamic range. DDE does not exhibit this high distribution dependency and achieves lower total SSE, as shown in Table II.
- In the trajectory datasets in Table I, DDE is not only lower in total SSE than EDMD but is also smaller in variance. Fig. 5-(c) shows that DDE has a uniformly low error distribution across the dynamic range, while EDMD in Fig.5-(d) has two regions, as circled in the figure, with significantly larger error. These regions coincide with sparsity in the dataset, providing evidence of EDMD's bias towards regions of high data density and explaining the difference in performance between models for the non-uniform datasets.
- For trajectory datasets, the total SSE converges for DDE with small dataset sizes. This result implies that the elements in the R and Q matrices of DDE, that is, the inner product integral computations, converge as the data size and the data density increase. This convergence is confirmed in Fig. 6, where several elements of the Q matrix are plotted against the data size.
- The second experiment, regarding variations in observable function numbers demonstrates the effect of DDE remains even for significant increases in observable functions, referring to Table III. In all cases tested, DDE significantly outperforms EDMD over the dynamic range, as expected for a non-uniform dataset.

V. Application to Neural Net Koopman Modeling

The use of deep neural networks for finding effective observable functions and constructing a Koopman linear model has been reported by several groups [10]–[12]. This method, sometimes referred to as Deep Koopman, is effective for approximating the Koopman Operator to a low-order model, compared to the use of locally activated functions, such as RBFs, which scale poorly for high-order nonlinear

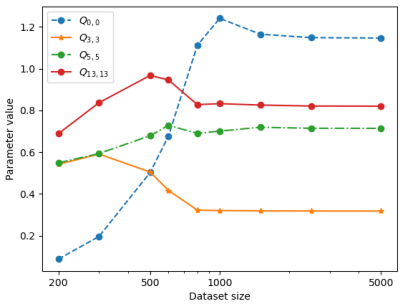

Fig. 6: Convergence of selected elements of the Q matrix. As the dataset size increases, each element $Q_{i,j}$ reaches a constant value. This implies that the A matrix converges as the dataset increases.

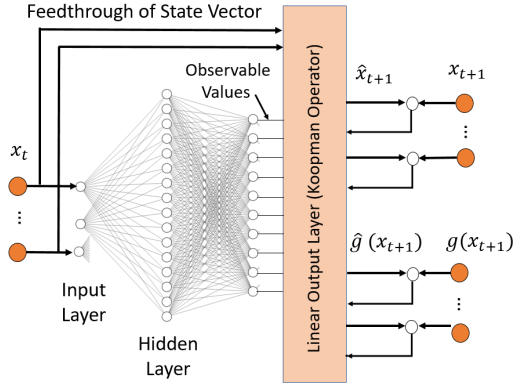

Fig. 7: Feedforward neural network model used to generate observable functions. The final layer of the neural network is a linear layer. In the standard Deep Koopman model this remains the same after the model is fully trained. However, with the DDE model, this final layer is recalculated using DDE by taking in the dataset used to train the model.

systems. The proposed DDE method can be incorporated into Deep Koopman, further improving approximation accuracy.

DDE can be applied to the neural network having an architecture similar to prior works [10]–[12]. See Fig. 7. The input layer receives training data of independent state variables. The hidden layers produce observable functions; these functions feed into the output layer consisting of linear activation units. This linear output layer corresponds to the A matrix that maps the observables of the current time to those of the next time step, i.e. the state transition in the lifted space. In the Deep Koopman approach, the output layer, that is, the A matrix, is trained together with observable functions in the hidden layers. This A matrix can be further improved by replacing it with the A matrix obtained from DDE, captioned in Fig. 7.

This Deep Koopman-DDE method is applied to a simulated cable manipulation system, similar to prior art [6]. The specific system used for the simulation experiment consists of one cable suspending a point mass, where a winch varies the length of one cable. The system is a 6-th order nonlinear, switched system where the cables go slack because of the unidirectional nature of cable tension. Because

this system is higher order, it necessitates the application of Deep Koopman for the selection of observable functions to accurately represent the system linearly with a finite order. More details can be found in [26]. The network is constructed using PyTorch with the parameters shown in Table IV. From simulated trajectories starting at diverse initial conditions, 3,000 data points are drawn. Table V compares the Deep Koopman model to the proposed model that uses DDE. Results are in terms of sum of squared error over a set of test trajectories. A significant improvement is achieved by incorporating DDE into the deep learning method.

TABLE IV: Neural network parameters and characteristics.

Parameters and Characteristics	Value
Number of Hidden layers	3
Activation Functions, Both Hidden Layers	ReLU
Width of 1st Hidden Layer	16
Width of 2nd Hidden Layer	16
Width of 3rd Hidden Layer	40
Learning Rate, α	0.01

TABLE V: Average SSE prediction error for trajectories in set of test data for single winch system.

Modeling Method	1 Time Step	20 Time Steps
Deep Koopman only	0.2471	9.8610
Deep Koopman + DDE	**0.2350**	**4.1131**

VI. Conclusion

In this work, a new data-driven approach to generating a Koopman linear model based on the direct encoding of Koopman Operator (DDE) is presented as an alternative to dynamic mode decomposition (DMD) and other related methods using least squares estimate (LSE). The major contributions include: 1) The analytical formula of Direct Encoding is converted to a numerical formula for computing the inner product integrals from given data; 2) An efficient algorithm is developed and its convergence conditions to the true results are analyzed; 3) Numerical experiments demonstrate a) greater accuracy compared to EDMD, b) lower sensitivity to data distribution, and c) rapid convergence of inner product computation. Furthermore, the DDE method is incorporated to Deep Koopman, i.e. neural network based methods for construction of the Koopman Operator, for improving prediction accuracy. The current method, however, is for autonomous systems. The extension to systems with control is a challenge for the future and must be addressed rigorously, beyond utilizing an input as an observable.

References

[1] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of fluid mechanics*, vol. 656, pp. 5–28, 2010.

[2] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data–driven approximation of the koopman operator: Extending dynamic mode decomposition," *Journal of Nonlinear Science*, vol. 25, no. 6, pp. 1307–1346, 2015.

[3] B. O. Koopman, "Hamiltonian systems and transformation in hilbert space," *Proceedings of the National Academy of Sciences*, vol. 17, no. 5, pp. 315–318, 1931.

[4] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control," *SIAM Journal on Applied Dynamical Systems*, vol. 15, no. 1, pp. 142–161, 2016.

[5] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, jul 2018. [Online]. Available: https://doi.org/10.1016%2Fj.automatica.2018.03.046

[6] J. Ng and H. H. Asada, "Model predictive control and transfer learning of hybrid systems using lifting linearization applied to cable suspension systems," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 682–689, 2021.

[7] V. Cibulka, T. Haniš, M. Korda, and M. Hromčík, "Model predictive control of a vehicle using koopman operator," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 4228–4233, 2020, 21st IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320331748

[8] H. M. Calderón, E. Schulz, T. Oehlschlägel, and H. Werner, "Koopman operator-based model predictive control with recursive online update," in *2021 European Control Conference (ECC)*, 2021, pp. 1543–1549.

[9] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Data-driven control of soft robots using koopman operator theory," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 948–961, 2021.

[10] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature Communications*, vol. 9, no. 1, Nov 2018. [Online]. Available: http://dx.doi.org/10.1038/s41467-018-07210-0

[11] E. Yeung, S. Kundu, and N. Hodas, "Learning deep neural network representations for koopman operators of nonlinear dynamical systems," 2017. [Online]. Available: https://arxiv.org/abs/1708.06850

[12] N. S. Selby and H. H. Asada, "Learning of causal observable functions for koopman-DFL lifting linearization of nonlinear controlled systems and its application to excavation automation," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6297–6304, oct 2021. [Online]. Available: https://doi.org/10.1109%2Flra.2021.3092256

[13] M. Korda and I. Mezić, "Optimal construction of koopman eigenfunctions for prediction and control," *IEEE Transactions on Automatic Control*, vol. 65, no. 12, pp. 5114–5129, 2020.

[14] G. Mamakoukas, I. Abraham, and T. D. Murphey, "Learning stable models for prediction and control," *IEEE Trans Robot*, 2020.

[15] F. Fan, B. Yi, D. Rye, G. Shi, and I. R. Manchester, "Learning stable koopman embeddings," 2021. [Online]. Available: https://arxiv.org/abs/2110.06509

[16] P. Bevanda, M. Beier, S. Kerz, A. Lederer, S. Sosnowski, and S. Hirche, "Diffeomorphically learning stable koopman operators," *IEEE Control Systems Letters*, vol. 6, pp. 3427–3432, 2022.

[17] M. Han, J. Euler-Rolle, and R. K. Katzschmann, "Desko: Stability-assured robust control with a deep stochastic koopman operator," in *International Conference on Learning Representations*, 2021.

[18] A. H. Abolmasoumi, M. Netto, and L. Mili, "Robust dynamic mode decomposition," *IEEE Access*, vol. 10, pp. 65 473–65 484, 2022. [Online]. Available: https://doi.org/10.1109%2Faccess.2022.3183760

[19] H. Asada, "Global, unified representation of heterogenous robot dynamics using composition operators," *IEEE/ASME Transactions on Mechatronics*, 2023.

[20] L. Ljung, "System identification," in *Signal analysis and prediction*. Springer, 1998, pp. 163–173.

[21] D. A. Freedman, *Statistical models: theory and practice*. Cambridge University Press, 2009.

[22] P. J. Davis and P. Rabinowitz, *Methods of numerical integration*. Courier Corporation, 2007.

[23] S.-W. Cheng, T. K. Dey, J. Shewchuk, and S. Sahni, *Delaunay mesh generation*. CRC Press Boca Raton, 2013.

[24] N. Amenta, D. Attali, and O. Devillers, "Complexity of delaunay triangulation for points on lower-dimensional˜ polyhedra," in *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 1106–1113.

[25] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.

[26] J. Ng, "Dde," https://github.com/jerrying123/DDE, 2023.