# Parallel Learning of Koopman Eigenfunctions and Invariant Subspaces For Accurate Long-Term Prediction

Masih Haseli and Jorge Cortés

*Abstract*—**We present a parallel data-driven strategy to identify finite-dimensional functional spaces invariant under the Koopman operator associated to an unknown dynamical system. We build on the Symmetric Subspace Decomposition (SSD) algorithm, a centralized method that under mild conditions on data sampling provably finds the maximal Koopman-invariant subspace and all Koopman eigenfunctions in an arbitrary finite-dimensional functional space. A network of processors, each aware of a common dictionary of functions and equipped with a local set of data snapshots, repeatedly interact over a directed communication graph. Each processor receives its neighbors' estimates of the invariant dictionary and refines its estimate by applying SSD with its local data on the intersection of the subspaces spanned by its own dictionary and the neighbors' dictionaries. We identify conditions on the network topology to ensure the algorithm identifies the maximal Koopman-invariant subspace in the span of the original dictionary, characterize its time, computational, and communication complexity, and establish its robustness against communication failures.**

## I. Introduction

Advances in processing and computation have led to numerous opportunities to enhance our understanding of complex dynamic behaviors. These opportunities, in turn, have challenged researchers to explore alternative representations of dynamical systems that take advantage of novel technologies. The Koopman operator is one such representation: rather than reasoning over trajectories, the operator describes the effect of the dynamics on functions defined over the state space. The operator is appealing for data-driven identification of dynamical phenomena since it is linear, independently of the structure of the underlying dynamics. This explains the significant insights into the physical laws governing the system provided by its eigenfunctions. Linearity is an advantage over other data-driven learning methods such as neural networks and statistical methods, which often lead to highly nonlinear models. However, the infinite-dimensional nature of the Koopman operator prevents the use of efficient algorithms developed for digital computers. One can circumvent this by finding finite-dimensional functional spaces that are invariant under the Koopman operator. The action of the operator on such subspaces is exact, enabling the use of fast linear algebraic methods with key implications for model reduction, stability, identification of stable/unstable manifolds, and control design.

*Literature Review:* The Koopman operator is a linear operator that represents the effect of a potentially nonlinear dynamics on a linear functional space [2], [3]. The eigendecomposition of the operator is particularly well suited to provide a comprehensive description of dynamical behavior since the Koopman eigenfunctions have linear temporal evolutions [4], [5]. As an example, [6] provides several criteria for global asymptotic stability of attractors based on Koopman eigenfunctions and their corresponding eigenvalues. Approximations of the Koopman operator and its eigendecomposition enable identification of nonlinear dynamical systems [7] and model reduction for nonlinear systems [8], [9], which leads to easier analysis of complex systems such as fluid flows [10]. The works [11]–[18] provide data-driven control schemes based on the identification of finite-dimensional linear representations for the Koopman operator associated with nonlinear systems. The works [19], [20] rely on symmetry to improve the accuracy of the Koopman approximation and characterize its observability properties.

To circumvent the challenges posed by the infinite-dimensional nature of the Koopman operator, significant research efforts have been devoted to find finite-dimensional approximations. Dynamic Mode Decomposition (DMD) [21], [22] uses linear algebraic methods to form a linear model based on the observed data. Extended Dynamic Mode Decomposition (EDMD) is a generalization of DMD that lifts the states of the systems to a space spanned by a dictionary of functions and finds a linear model by solving a data-driven least-squares problem [23]. Unlike DMD, EDMD provides a way to analyze the effect of the dynamics on an arbitrary functional space. EDMD converges [24] to the projection of the Koopman operator on the subspace spanned by the dictionary as the number of sampled data goes to infinity. This indicates the importance of the chosen dictionary in capturing important information about the operator. If the dictionary spans a Koopman-invariant subspace, then the EDMD approximation is exact; otherwise, EDMD loses information about the system and might not suitable for long-term prediction due to errors in the approximation. This observation motivates the search for methods to identify Koopman-invariant subspaces.

Since Koopman eigenfunctions automatically generate invariant subspaces, a number of works have focused the attention on finding them [14], [25], including the use of multi-step trajectory predictions [26] and sparsity promoting methods [27] that prune the EDMD eigendecomposition. The latter relies on the fact that EDMD captures all the Koopman eigenfunctions in the span of the dictionary, cf. [28], but not all identified eigenfunctions using EDMD are actually Koopman eigenfunctions. The literature contains several methods based on neural networks that can approximate Koopman-invariant subspaces [29]–[32]. None of the aforementioned methods provide guarantees for the identified subspaces to be Koopman invariant. Our recent work [28] provides a necessary and sufficient condition for the identification of functions that evolve

This work was supported by ONR Award N00014-18-1-2828.

A preliminary version appeared as [1] at the American Control Conference.

Masih Haseli and Jorge Cortés are with Department of Mechanical and Aerospace Engineering, University of California, San Diego, CA 92093, USA, {mhaseli,cortes}@ucsd.edu

linearly according to the dynamics based on the application of EDMD forward and backward in time, and establishes conditions on the density of data sampling to ensure that the identified functions are Koopman eigenfunctions almost surely. The work also introduces SSD algorithms which, should all data be centrally available and under mild conditions on data sampling, provably find the maximal Koopman-invariant subspace in the span of an arbitrary finite-dimensional functional space. By contrast, here we profit from parallelization to speed up computation and enable fast processing for large datasets and real-time applications.

*Statement of Contributions:* We present a parallel data-driven method to identify Koopman-invariant subspaces and eigenfunctions associated with a (potentially nonlinear) unknown discrete-time dynamical system. The proposed algorithm is compatible with parallel processing hardware. Our starting point is a group of processors communicating through a directed graph, with each processor aware of a common dictionary of functions and equipped with a local set of data snapshots acquired from the dynamics. We introduce the Parallel Symmetric Subspace Decomposition (P-SSD) algorithm to find the maximal Koopman-invariant subspace and all the Koopman eigenfunctions in the finite-dimensional linear functional space spanned by the dictionary. The proposed strategy has each processor refine its estimate of the invariant dictionary by iteratively employing the information received from its neighbors to prune it. We show that the P-SSD algorithm reaches an equilibrium in a finite number of time steps for any (possibly time-varying) network topology and carefully characterize the properties of the agents' dictionary iterates along its execution, particularly in what concerns monotonicity of the associated subspaces. We also establish that the globally reachable processors in the communication digraph find the same solution as the SSD algorithm would if all data was centrally available at a single processor. This allows us to conclude that the P-SSD algorithm finds the maximal Koopman-invariant subspace in the space spanned by the original dictionary if the network topology is strongly connected. Finally, we conclude by characterizing the algorithm's time, computational, and communication complexity, demonstrating its computational advantage over SSD, and showing its robustness against communication failures and packet drops. Simulations illustrate the superior performance of the proposed strategy over other methods in the literature[1].

[1]Throughout the paper, we use the following notation. We use $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{R}$, and $\mathbb{C}$, to denote the sets of natural, nonnegative integer, real, and complex numbers, resp. Given integers $a, b$ we use $a \bmod b$ to represent the remainder of division of $a$ by $b$. Given a matrix $A \in \mathbb{C}^{m\times n}$, we represent its set of rows, set of columns, number of rows, and number of columns by $\operatorname{rows}(A)$, $\operatorname{cols}(A)$, $\sharp\operatorname{rows}(A)$, and $\sharp\operatorname{cols}(A)$, resp. In addition, we denote its transpose, conjugate transpose, pseudo-inverse, and range space by $A^T$, $A^H$, $A^\dagger$, and $\mathcal{R}(A)$. If $A$ is square, we denote its inverse by $A^{-1}$. Given $A \in \mathbb{C}^{m\times n}$ and $B \in \mathbb{C}^{m\times d}$, we use $[A, B] \in \mathbb{C}^{m\times(n+d)}$ to represent the matrix created by concatenating $A$ and $B$. Given $v \in \mathbb{C}^n$, we denote its 2-norm by $\|v\|_2 := \sqrt{v^H v}$. Given vectors $v_1, \dots, v_k \in \mathbb{C}^n$, we denote by $\operatorname{span}\{v_1, \dots, v_k\}$, the set comprised of all vectors in the form of $c_1v_1 + \cdots + c_nv_n$ with $c_1, \dots, c_n \in \mathbb{C}$. $\angle(v, w)$ represents the angle between $v, w \in \mathbb{R}^n$. For sets $S_1$ and $S_2$, $S_1 \subseteq S_2$ means that $S_1$ is a subset of $S_2$. In addition, we denote the intersection and union of $S_1$ and $S_2$ by $S_1 \cap S_2$ and $S_1 \cup S_2$, resp. Given functions $f : S_2 \to S_1$ and $g : S_3 \to S_2$, we denote their composition by $f \circ g : S_3 \to S_1$.

## II. Preliminaries

In this section, we present a brief account of Koopman operator theory and basic definitions from graph theory.

*Koopman Operator:* Our exposition here follows [5]. Consider the discrete-time dynamical system

$$x^+ = T(x), \tag{1}$$

where $T : \mathcal{M} \to \mathcal{M}$ is defined over the state space $\mathcal{M} \subseteq \mathbb{R}^n$. The Koopman operator associated with (1) characterizes the effect of the dynamics on functions (also known as observables) in a linear functional space $\mathcal{F}$ defined from $\mathcal{M}$ to $\mathbb{C}$. If $\mathcal{F}$ is closed under composition with $T$ (i.e., $f \circ T \in \mathcal{F}$, for all $f \in \mathcal{F}$), one can define the Koopman operator $\mathcal{K} : \mathcal{F} \to \mathcal{F}$ associated with (1) as

$$\mathcal{K}(f) = f \circ T.$$

Since $\mathcal{F}$ is a linear space, the Koopman operator is *spatially linear*, i.e.,

$$\mathcal{K}(c_1 f_1 + c_2 f_2) = c_1\mathcal{K}(f_1) + c_2\mathcal{K}(f_2), \tag{2}$$

for every $f_1, f_2 \in \mathcal{F}$ and $c_1, c_2 \in \mathbb{C}$. Unlike (1), which describes the effect of the dynamics on points in the state space, the Koopman operator describes the effect of the dynamics on functions in $\mathcal{F}$. If $\mathcal{F}$ contains functions that can describe the states of the system, the Koopman operator fully describes the evolution of the states in time. One way to enforce this richness criteria is to include the state observables $f_i(x) := x_i$ in the space $\mathcal{F}$ for all $i \in \{1, \dots, n\}$. This condition, together with its closedness under $T$, might force $\mathcal{F}$ to be infinite dimensional.

Having a linear operator enables us to define the eigendecomposition of the Koopman operator. Formally, the function $\phi \in \mathcal{F}$ is an eigenfunction of the Koopman operator associated with eigenvalue $\lambda \in \mathbb{C}$ if

$$\mathcal{K}(\phi) = \lambda\phi. \tag{3}$$

The eigenfunctions of the Koopman operator evolve linearly in time, i.e., $\phi(x^+) = (\phi \circ T)(x) = \mathcal{K}(\phi)(x) = \lambda\phi(x)$. This *temporal linearity* in conjunction with the *spatial linearity* in (2) render the spectral properties of the Koopman operator a powerful tool in analyzing nonlinear dynamical systems. Formally, given a function

$$f = \sum_{i=1}^{N_k} c_i \phi_i, \tag{4}$$

where $\{\phi_i\}_{i=1}^{N_k}$ are Koopman eigenfunctions with eigenvalues $\{\lambda_i\}_{i=1}^{N_k}$ and $\{c_i\}_{i=1}^{N_k} \subset \mathbb{C}$, one can use the spectral properties of the Koopman operator and describe the temporal evolution of $f$ according to the dynamics as

$$f(x(k)) = \sum_{i=1}^{N_k} c_i \lambda_i^k \, \phi_i(x(0)), \quad \forall k \in \mathbb{N}. \tag{5}$$

It is important to note that due to the infinite-dimensional nature of the Koopman operator, one might need infinitely many eigenfunctions to describe an arbitrary function in a linear manner. One way to avoid this infinite-dimensionality problem is to analyze the functions in finite-dimensional subspaces that are invariant under the application of the Koopman operator. A subspace $\mathcal{S} \subseteq \mathcal{F}$ is *Koopman invariant* if $\mathcal{K}(f) \in \mathcal{S}$ for all $f \in \mathcal{S}$. Moreover, $\mathcal{S} \subseteq \mathcal{P}$ is the *maximal Koopman-invariant* subspace of $\mathcal{P}$ if $\mathcal{S}$ is Koopman invariant and, for every Koopman-invariant subspace $\mathcal{L} \subseteq \mathcal{P}$, we have $\mathcal{L} \subseteq \mathcal{S}$.

*Graph Theory:* Our exposition here mainly follows [33], [34]. Given a set $V$ comprised of $m$ *nodes* and a set $E \subseteq V \times V$ comprised of ordered pairs of nodes called *edges*, the pair $G = (V, E)$ defines a *directed graph (digraph)* on nodes $V$ and edges $E$. We say node $i$ is an *in-neighbor* of node $j$ and node $j$ is an *out-neighbor* of node $i$ if $(i, j) \in E$. We denote by $\mathcal{N}_{\text{in}}(i)$ and $\mathcal{N}_{\text{out}}(i)$ the sets of in- and out-neighbors of $i$, resp. A *directed path* with *length* $l$ is an ordered sequence of $l+1$ nodes such that the ordered pair of every two consecutive nodes is an edge of the digraph. A path is *closed* if its first and last nodes are the same. A node is called *globally reachable* if there is a directed path from every other node to it. A digraph is *strongly connected* if all of its nodes are globally reachable. Given two nodes $i, j$ in a digraph, the *distance* from $i$ to $j$, denoted $\text{dist}(i, j)$, is the length of the shortest directed path from $i$ to $j$. If there is no such directed path, the distance is $\infty$. Given two digraphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, we define their *composition* as $G_2 \circ G_1 = (V, E_\circ)$ such that $E_\circ = \{(i, j) \mid \exists k \in V \text{ with } (i, k) \in E_1 \wedge (k, j) \in E_2\}$. A finite sequence of digraphs $\{G_i = (V, E_i)\}_{i=1}^{k}$ is *jointly strongly connected* if $G_k \circ \cdots \circ G_1$ is strongly connected. An infinite sequence of digraphs (or a time-varying digraph) $\{G_i = (V, E_i)\}_{i=1}^{\infty}$ is *repeatedly jointly strongly connected* if there exists $l, \tau \in \mathbb{N}$ such that for every $k \in \mathbb{N}_0$, the sequence of digraphs $\{G_{\tau+kl+j}\}_{j=0}^{l-1}$ is jointly strongly connected.

## III. Problem Setup

Given an arbitrary finite-dimensional functional space $\mathcal{P}$ defined on a state space $\mathcal{M}$, our goal is to design efficient data-driven algorithms that are able to identify the maximal Koopman-invariant subspace in $\mathcal{P}$ associated with a dynamical system whose explicit model is unknown. To do so, we rely on data collected about the dynamical behavior of the system and aim to ensure that the proposed methods are compatible with parallel processing hardware.

We start by defining the concepts of data snapshots and dictionary. We represent by $X, Y \in \mathbb{R}^{N \times n}$ matrices comprised of $N$ data snapshots acquired from system (1) such that $x_i^T$ and $y_i^T$, the $i$th rows of $X$ and $Y$, satisfy

$$y_i = T(x_i), \quad \forall i \in \{1, \ldots, N\}.$$

Moreover, we define a dictionary $D : \mathcal{M} \to \mathbb{R}^{1 \times N_d}$

$$D(x) = [d_1(x), \ldots, d_{N_d}(x)],$$

comprised of $N_d$ real-valued functions $d_1, \ldots, d_{N_d}$ defined on $\mathcal{M}$ with $\text{span}\{d_1, \ldots, d_{N_d}\} = \mathcal{P}$. Note that, one can completely characterize any dictionary $\tilde{D}$ with elements in $\text{span}\{d_1, \ldots, d_{N_d}\}$ by a matrix $C$ with $N_d$ rows as $\tilde{D}(x) = D(x)C$. The effect of the dictionary on data matrices is

$$D(X) = [D(x_1)^T, \ldots, D(x_N)^T]^T \in \mathbb{R}^{N \times N_d}.$$

Throughout the paper, we operate under the following standard assumption on dictionary matrices.

*Assumption 3.1: (Full Column Rank Dictionary Matrices):* The matrices $D(X)$ and $D(Y)$ have full column rank. □

Assumption 3.1 requires the dictionary functions to be linearly independent and the data snapshots to be diverse enough to capture the important features of the dynamics.

We consider a group of $M$ processors or agents that communicate according to a digraph $G$. Every agent is aware of the dictionary. We assume the data snapshots $X$, $Y$ are distributed among the processors, forming the dictionary snapshots $D(X_i), D(Y_i)$ for each agent $i \in \{1, \ldots, M\}$ such that

$$\bigcup_{i=1}^{M} \text{rows}([D(X_i), D(Y_i)]) = \text{rows}([D(X), D(Y)]).$$

Local data snapshots might be available at the agent as a result of distributed acquisition or because the global data has been partitioned among the processors. The latter can be done in different ways, including partitioning in a way that minimizes the maximum time taken for each agent to finish one iteration of calculation. For homogeneous processors, this can be done by uploading the signature data sets to agents and evenly distributing the rest of the data among them.

The processors share a small portion of dictionary snapshots, $D(X_s)$ and $D(Y_s)$ with full column rank, called the *signature dictionary matrices* (note that one can build signature dictionary snapshots with no more than $2N_d$ rows according to Assumption 3.1). Hence, we have

$$\text{rows}([D(X_s), D(Y_s)]) \subseteq \text{rows}([D(X_i), D(Y_i)]),$$

for $i \in \{1, \ldots, M\}$. Interestingly, for our forthcoming results, the processors do not need to be aware of which data points correspond to the signature snapshots.

*Problem Statement:* Given a group of $M$ processors communicating according to a network, the dictionary $D = [d_1, \ldots, d_{N_d}]$, and the data snapshots $X, Y \in \mathbb{R}^{N \times n}$, we aim to design a parallel algorithm that is able to identify the dictionary $\tilde{D}$ spanning the maximal Koopman-invariant subspace in $\text{span}\{d_1, \ldots, d_{N_d}\}$. The processors must only rely on the their local data and communication with their neighbors to achieve this goal.

In general, the action of Koopman operator on the functional space $\mathcal{P} = \text{span}\{d_1, \ldots, d_{N_d}\}$ is not invariant, meaning that one needs to project back to $\mathcal{P}$ the image under $\mathcal{K}$ of functions in $\mathcal{P}$, thereby introducing error. Instead, the image of functions belonging to the identified invariant subspace under the Koopman operator remains in the subspace, avoiding the need for the projection and therefore eliminating the source of approximation error. We note that efficient solutions to this problem could be employed in the construction of better dictionaries. Although beyond the scope of this paper, one could envision, for instance, augmenting the dictionary to increase its linear span of functions and then pruning it to eliminate the approximation error. Throughout the paper, we use data-driven methods that are not specifically designed to work with data corrupted with measurement noise. Hence, one might need to pre-process the data before applying the proposed algorithms.

## IV. Parallel Symmetric Subspace Decomposition

This section presents a parallel algorithm to identify the dictionary $\tilde{D}$ spanning the maximal Koopman-invariant subspace in $\text{span}\{d_1, \dots, d_{N_d}\}$, as stated in Section III. We start by noting that the invariance of $\tilde{D}$ implies that $\mathcal{R}(\tilde{D}(x^+)) = \mathcal{R}(\tilde{D}(x))$, for all $x \in \mathcal{M}$. This gets reflected in the data as $\mathcal{R}(\tilde{D}(Y)) = \mathcal{R}(\tilde{D}(X))$. Using the fact that $\tilde{D}$ is fully characterized by a matrix $C$ with $\tilde{D}(x) = D(x)C$, this equation can be rewritten as

$$\mathcal{R}(D(Y)C) = \mathcal{R}(D(X)C). \tag{6}$$

Hence, under appropriate conditions on the data sampling, the problem of finding the maximal Koopman-invariant subspace can be formulated in algebraic terms by looking for the full column rank matrix $C$ with maximum number of columns such that (6) holds.

**Algorithm 1** Symmetric Subspace Decomposition

```
    Inputs: $D(X), D(Y) \in \mathbb{R}^{N \times N_d}$        Output: $C_{\text{SSD}}$
 1: Initialization
 2: $i \leftarrow 1$, $A_1 \leftarrow D(X)$, $B_1 \leftarrow D(Y)$, $C_{\text{SSD}} \leftarrow I_{N_d}$
 3: while 1 do
 4:     $\begin{bmatrix} Z_i^A \\ Z_i^B \end{bmatrix} \leftarrow \text{null}([A_i, B_i])$    ▷ Basis for the null space
 5:     if $\text{null}([A_i, B_i]) = \emptyset$ then
 6:         return 0                         ▷ The basis does not exist
 7:         break
 8:     end if
 9:     if $\sharp\text{rows}(Z_i^A) \leq \sharp\text{cols}(Z_i^A)$ then
10:         return $C_{\text{SSD}}$               ▷ The procedure is complete
11:         break
12:     end if
13:     $C_{\text{SSD}} \leftarrow C_{\text{SSD}} Z_i^A$                  ▷ Reduce the subspace
14:     $A_{i+1} \leftarrow A_i Z_i^A$, $B_{i+1} \leftarrow B_i Z_i^A$, $i \leftarrow i + 1$
15: end while
```

When all the data is available at a single processor, the Symmetric Subspace Decomposition (SSD) algorithm proposed in [28], cf. Algorithm 1, is a centralized procedure that identifies $\tilde{D}$. Note that (6) implies

$$\mathcal{R}(D(X)C) = \mathcal{R}(D(Y)C) \subseteq \mathcal{R}(D(X)) \cap \mathcal{R}(D(Y)),$$

which in turn leads to rank deficiency in the concatenated matrix $[D(X), D(Y)]$. SSD prunes at each iteration the dictionary using this rank deficiency until (6) holds for the final dictionary $\tilde{D}$. For reference, Appendix A summarizes the main properties of SSD. Here, we just point out that, under appropriate conditions on the data sampling, $\tilde{D}$ is a basis for the maximal Koopman-invariant subspace in the span of the original dictionary almost surely. The computational cost of SSD grows as $N$ and $N_d$ grow, mainly due to Step 4, which requires calculation of the null space of a $N \times (2N_d)$ matrix.

Here, we build on the SSD algorithm to address the problem laid out in Section III, i.e., identifying $\tilde{D}$ by means of a parallel strategy, either because the data is not centrally available at a single processor or, even if it is, because its implementation over multiple processors speeds up the identification significantly. Our algorithm, termed Parallel Symmetric Subspace Decomposition[2] (P-SSD), cf. Algorithm 2, has each processor use SSD on its local data and prune its subspace using the estimates of the invariant dictionary communicated by its neighbors. The following is an informal description of the algorithm steps:

> *Informal description of P-SSD:* Given dictionary snapshots $D(X)$, $D(Y)$ distributed over $M$ processors communicating over a network, each agent iteratively: (i) receives its neighbors' estimates of the invariant dictionary, (ii) uses the SSD algorithm to identify the largest invariant subspace (according to the local data) in the intersection of its dictionary and its neighbors' dictionaries, (iii) chooses a basis for the identified subspace as its own dictionary, and (iv) transmits that dictionary to its neighbors.

The proposed P-SSD algorithm builds on the observation that the subspaces identified by SSD are monotone non-increasing with respect to the addition of data (Lemma A.2), i.e., the dimension of the identified subspace does not increase when we add more data.

**Algorithm 2** Parallel Symmetric Subspace Decomposition

```
    Inputs: $D(X_i), D(Y_i)$, $i \in \{1, \dots, M\}$
    Each agent i executes:
 1: $k \leftarrow 0$, $C_0^i \leftarrow I_{N_d}$, $\text{flag}_0^i \leftarrow 0$
 2: while 1 do
 3:     $k \leftarrow k + 1$
 4:     Receive $C_{k-1}^j$, $\forall j \in \mathcal{N}_{\text{in}}^k(i)$
 5:     $D_k^i \leftarrow \text{basis}\Big(\bigcap_{j \in \{i\} \cup \mathcal{N}_{\text{in}}^k(i)} \mathcal{R}(C_{k-1}^j)\Big)$
 6:     $E_k^i \leftarrow \text{SSD}(D(X_i)D_k^i, D(Y_i)D_k^i)$
 7:     if $\sharp\text{cols}(D_k^i E_k^i) < \sharp\text{cols}(C_{k-1}^i)$ then
 8:         $C_k^i \leftarrow D_k^i E_k^i$            ▷ The subspace gets smaller
 9:         $\text{flag}_k^i \leftarrow 0$
10:     else
11:         $C_k^i \leftarrow C_{k-1}^i$         ▷ The subspace does not change
12:         $\text{flag}_k^i \leftarrow 1$
13:     end if
14:     Transmit $C_k^i$ to out-neighbors
15: end while
```

### A. Equilibria and Termination of P-SSD

Here, we define the concept of equilibria of the P-SSD algorithm and discuss how to detect whether the agents have attained one (we refer to this as termination). Viewing the algorithm as a discrete-time dynamical system, we start by defining the concept of equilibrium.

*Definition 4.1: (Equilibrium):* The agent matrices $C_k^i$, $i \in \{1, \dots, M\}$ in Algorithm 2 are an equilibrium of P-SSD at time $k \in \mathbb{N}$ if $C_p^i = C_k^i$ for all $p > k$ and all $i \in \{1, \dots, M\}$.

Note that, after reaching an equilibrium, the subspaces identified by the agents do not change anymore. The next result

[2]The function basis($\mathcal{A}$) returns a matrix whose columns provide a basis for $\mathcal{A}$. If $\mathcal{A} = \{0\}$, then basis($\mathcal{A}$) returns 0. Moreover, $\sharp\text{cols}(0) = 0$.

shows that the P-SSD algorithm always reaches an equilibrium in a finite number of iterations.

*Proposition 4.2: (Reaching Equilibrium in a Finite Number of Iterations):* The P-SSD algorithm executed over any (possibly time-varying) digraph reaches an equilibrium after a finite number of iterations.

For space reasons, the proof is presented in the online version [35]. Since the algorithm runs in parallel, we need a mechanism to detect if the P-SSD algorithm has reached an equilibrium. The flag variables carry out this task.

*Proposition 4.3: (Equilibria is Detected by Flag Variables in Time Invariant Digraphs):* Given a time-invariant network and under the P-SSD algorithm, $\text{flag}_l^i = 1$ for some $l \in \mathbb{N}$ and for every $i \in \{1, \dots, M\}$ if and only if $C_k^i = C_{l-1}^i$ and $\text{flag}_k^i = 1$ for every $i \in \{1, \dots, M\}$ and every $k \geq l$.

For space reasons, the proof is presented in the online version [35]. Note that the flags detect an equilibrium with one time-step delay. We say that the P-SSD algorithm has *terminated at step* $k$ if $\text{flag}_k^i = 1$ for all $i \in \{1, \dots, M\}$.

*Remark 4.4: (Termination of the P-SSD Algorithm):* We point out that one may consider different notions of termination for P-SSD: (i) the algorithm reaches an equilibrium, i.e., the agents continue their calculations but do not change their outputs; (ii) based on Proposition 4.3, a user external to the parallel processing hardware terminates the algorithm when $\prod_{i=1}^{M} \text{flag}_k^i = 1$ for some $k \in \mathbb{N}$; (iii) agents use distributed halting algorithms [36], [37] to decide when to terminate based on their output and flags. Here, we only employ (i) and (ii) as appropriate and do not implement (iii) for space reasons. □

## B. Properties of Agents' Matrix Iterates along P-SSD

Here we characterize important properties of the matrices computed by the agents at each iteration of the P-SSD algorithm. The results provided in this section hold for any (including time-varying) communication network topology. The next result characterizes basic properties of the agents' matrix iterates.

*Theorem 4.5: (Properties of Agents' Matrix Iterates):* Under the P-SSD algorithm and for any (possibly time-varying) digraph, for each $i \in \{1, \dots, M\}$,

(a) for all $k \in \mathbb{N}_0$, $C_k^i$ is zero or has full column rank;
(b) for all $k \in \mathbb{N}$, $\mathcal{R}(D_k^i E_k^i) = \mathcal{R}(C_k^i)$;
(c) for all $k \in \mathbb{N}$, $\mathcal{R}(D(X_i) C_k^i) = \mathcal{R}(D(Y_i) C_k^i)$.

*Proof:* Consider an arbitrary $i \in \{1, \dots, M\}$. We prove (a) by induction. For $k = 0$, $C_0^i = I_{N_d}$. Now, suppose that $C_k^i$ is zero or has full column rank, and let us show the same fact for $k+1$. Note that, if $D_{k+1}^i = 0$ or $E_{k+1}^i = 0$, then $C_{k+1}^i = 0$ and the result follows. Now, suppose that $D_{k+1}^i \neq 0$ and $E_{k+1}^i \neq 0$. Based on definition of basis and Theorem A.1(a), $D_{k+1}^i$ and $E_{k+1}^i$ have full column rank. Note that the algorithm either executes Step 8 or executes Step 11. Hence, $C_{k+1}^i = D_{k+1}^i E_{k+1}^i$ or $C_{k+1}^i = C_k^i$. Consequently, one can deduce that $C_{k+1}^i$ has full column rank ($C_k^i \neq 0$ since $D_{k+1}^i \neq 0$, hence $C_k^i$ has full column rank), establishing (a).

Next, we show (b). If the algorithm executes Step 8 at iteration $k$, then $C_k^i = D_k^i E_k^i$ and consequently $\mathcal{R}(D_k^i E_k^i) = \mathcal{R}(C_k^i)$. Suppose instead that the algorithm executes Step 11, and consequently $C_k^i = C_{k-1}^i$. For the case $C_k^i = 0$, using Step 5, we deduce $D_k^i E_k^i = 0$ and the statement follows. Assume then that $C_k^i \neq 0$ and hence it has full column rank as a result of (a). Consequently, since the condition in Step 7 does not hold and using the definition of $\sharp\text{cols}$, we have $D_k^i \neq 0$ and $E_k^i \neq 0$. Also, they have full column rank based on the definition of basis and Theorem A.1(a). Moreover, $\mathcal{R}(D_k^i E_k^i) \subseteq \mathcal{R}(D_k^i) \subseteq \mathcal{R}(C_{k-1}^i)$ and since the matrices have full column rank, $\sharp\text{cols}(D_k^i E_k^i) \leq \sharp\text{cols}(D_k^i) \leq \sharp\text{cols}(C_{k-1}^i)$. In addition, and based on Step 7, $\sharp\text{cols}(D_k^i E_k^i) \geq \sharp\text{cols}(C_{k-1}^i)$. Hence,

$$\sharp\text{cols}(D_k^i E_k^i) = \sharp\text{cols}(D_k^i) = \sharp\text{cols}(C_{k-1}^i). \tag{7}$$

We can use $\sharp\text{rows}(E_k^i) = \sharp\text{cols}(D_k^i)$ and $\sharp\text{cols}(D_k^i E_k^i) = \sharp\text{cols}(E_k^i)$ in conjunction with (7) to deduce that $E_k^i$ is square and nonsingular (since it has full column rank). Consequently, $\mathcal{R}(D_k^i) = \mathcal{R}(D_k^i E_k^i)$. This fact, in combination with $\mathcal{R}(D_k^i) \subseteq \mathcal{R}(C_{k-1}^i)$, yields

$$\mathcal{R}(D_k^i E_k^i) \subseteq \mathcal{R}(C_{k-1}^i). \tag{8}$$

Moreover, since $D_k^i$, $E_k^i$, and $C_{k-1}^i$ have full column rank, one can use (7) and (8) to deduce that $\mathcal{R}(D_k^i E_k^i) = \mathcal{R}(C_{k-1}^i)$. Finally, since $C_k^i = C_{k-1}^i$, we have $\mathcal{R}(D_k^i E_k^i) = \mathcal{R}(C_k^i)$, which shows (b).

To show (c), from Step 6 of the algorithm and Theorem A.1(a), for each $i \in \{1, \dots, M\}$ and each $k \in \mathbb{N}$,

$$\mathcal{R}(D(X_i) D_k^i E_k^i) = \mathcal{R}(D(Y_i) D_k^i E_k^i).$$

Based on part (b) and the fact that $D(X_i), D(Y_i)$ have full column rank, one can deduce $\mathcal{R}(D(X_i) C_k^i) = \mathcal{R}(D(Y_i) C_k^i)$, concluding the proof. ■

*Remark 4.6: (Computation of the* basis *Function with Finite-Precision Machines):* Based on Theorem 4.5(a), we provide here a practical way of implementing the basis function. Since the matrices $C_k^i$ are either full rank or zero, instead of working with their range space, one can work directly with the matrices themselves. Here, we compute iteratively the output of the basis function for the input matrices. Given the full column-rank matrices $A_1, A_2$, we find $\text{basis}(\mathcal{R}(A_1) \cap \mathcal{R}(A_2))$ using [28, Lemma A.1] by finding the null space of $[A_1, A_2]$ (if one of the matrices is equal to zero, then $\text{basis}(\mathcal{R}(A_1) \cap \mathcal{R}(A_2)) = 0$). This procedure can be implemented iteratively for $i \geq 2$ by noting

$$\text{basis}\Big(\bigcap_{j=1}^{i+1} \mathcal{R}(A_i)\Big) = \text{basis}\bigg(\mathcal{R}\Big(\text{basis}\Big(\bigcap_{j=1}^{i} \mathcal{R}(A_i)\Big)\Big) \bigcap \mathcal{R}(A_{i+1})\bigg).$$

When implemented on digital computers, this might lead to small errors affecting the null space of $[A_1, A_2]$. To avoid this problem, given the singular values $\sigma_1 \geq \cdots \geq \sigma_l$ of $[A_1, A_2]$ and a tuning parameter $\epsilon_\cap > 0$, we compute $k$ as the minimum integer satisfying $\sum_{j=k}^{l} \sigma_j^2 \leq \epsilon_\cap (\sum_{j=1}^{l} \sigma_j^2)$ and then set $\sigma_k = \cdots = \sigma_l = 0$. The parameter $\epsilon_\cap$ tunes the sensitivity of the implementation. □

Next, we show that the range space identified by an agent at any time step is contained into the range space identified by that agent and its neighbors in the previous time step.

*Lemma 4.7: (Subspace Monotonicity of Agents' Matrix Iterates):* Under the P-SSD algorithm and for any (possibly time-varying) digraph, at each iteration $k \in \mathbb{N}$, it holds that $\mathcal{R}(C_k^i) \subseteq \mathcal{R}(C_{k-1}^j)$ for all $i \in \{1, \dots, M\}$ and all $j \in \{i\} \cup \mathcal{N}_{\text{in}}^k(i)$.

For space reasons, the proof is presented in the online version [35]. We conclude this section by showing that the range space of the agents' matrix iterates contain the *SSD subspace*, i.e., the subspace identified by the SSD algorithm when all the data is available at once at a single processor.

*Proposition 4.8: (Inclusion of SSD Subspace by Agents' Matrix Iterates):* Let $C_{\text{SSD}} = \text{SSD}(D(X), D(Y))$ be the output of the SSD algorithm applied on $D(X), D(Y)$. Under the P-SSD algorithm and for any (possibly time-varying) digraph, at each iteration $k \in \mathbb{N}_0$, it holds that $\mathcal{R}(C_{\text{SSD}}) \subseteq \mathcal{R}(C_k^i)$ for all $i \in \{1, \dots, M\}$.

For space reasons, the proof is presented in the online version [35].

## V. Equivalence of P-SSD and SSD

In this section, we study under what conditions on the digraph's connectivity, the P-SSD algorithm is equivalent to the SSD algorithm, i.e., it finds the maximal Koopman-invariant subspace contained in the span of the original dictionary. We start by studying the relationship between the matrix iterates and local data of the agents at the beginning and end of a directed path in the digraph.

*Proposition 5.1: (Relationship of Agents' Matrices and Local Data Along Directed Paths):* Given an arbitrary constant digraph, let $P$ be a directed path of length $l$ from node $j$ to node $i$. Then, under the P-SSD algorithm, for each iteration $k \in \mathbb{N}$ and all $q \geq k + l$,

$$\mathcal{R}(C_q^i) \subseteq \mathcal{R}(C_k^j), \tag{9a}$$

$$\mathcal{R}(D(X_j)C_q^i) = \mathcal{R}(D(Y_j)C_q^i). \tag{9b}$$

*Proof:* To prove (9a), we label each node on the path $P$ by $p_1$ through $p_{l+1}$, with $p_1$ corresponding to node $j$ and $p_{l+1}$ corresponding to node $i$ (note that a node will have more than one label if it appears more than once in the path, which does not affect the argument). Based on Lemma 4.7, for all $k \in \mathbb{N}$, one can write

$$\mathcal{R}(C_{k+m}^{p_{m+1}}) \subseteq \mathcal{R}(C_{k+m-1}^{p_m}), \, \forall m \in \{1, \dots, l\}.$$

Using this equation $l$ times yields $\mathcal{R}(C_{k+l}^i) \subseteq \mathcal{R}(C_k^j)$. Moreover, using Lemma 4.7 once again for node $i$, we deduce $\mathcal{R}(C_q^i) \subseteq \mathcal{R}(C_{k+l}^i)$ for every $q \geq k+l$, and the proof follows.

Regarding (9b), the case $C_q^i = 0$ is trivial. Suppose then that $C_q^i \neq 0$, with full column rank (cf. Theorem 4.5(a)). Using (9a) and Theorem 4.5(a), we deduce that $C_k^j$ also has full column rank. Therefore, there exists a full column rank matrix $F$ such that

$$C_q^i = C_k^j F. \tag{10}$$

From Theorem 4.5(c), we have $\mathcal{R}(D(X_i)C_q^i) = \mathcal{R}(D(Y_i)C_q^i)$. Looking at this equation in a row-wise manner and considering only the signature matrices, one can write

$$\mathcal{R}(D(X_s)C_k^j F) = \mathcal{R}(D(Y_s)C_k^j F), \tag{11}$$

where we have used (10). From Theorem 4.5(c) for agent $j$,

$$\mathcal{R}(D(X_j)C_k^j) = \mathcal{R}(D(Y_j)C_k^j).$$

Using this equation and given the fact that $D(X_j)$, $D(Y_j)$, and $C_k^j$ have full column rank, there must exist a square nonsingular matrix $K$ such that

$$D(Y_j)C_k^j = D(X_j)C_k^j K. \tag{12}$$

Looking at this equation in a row-wise manner and considering only the signature data matrices,

$$D(Y_s)C_k^j = D(X_s)C_k^j K. \tag{13}$$

Using a similar argument for (11), one can deduce that there exists a nonsingular square matrix $K^*$ such that

$$D(Y_s)C_k^j F = D(X_s)C_k^j F K^*. \tag{14}$$

Multiplying both sides of (13) from the right by $F$ and subtracting from (14) results in

$$D(X_s)C_k^j(FK^* - KF) = 0.$$

Since $D(X_s)C_k^j$ has full column rank, we deduce $FK^* = KF$. Now, by multiplying both sides of (12) from the right by $F$ and replacing $KF$ by $FK^*$, we can write

$$D(Y_j)C_q^i = D(Y_j)C_k^j F = D(X_j)C_k^j F K^* = D(X_j)C_q^i K^*,$$

where we have used (10) twice, which shows the result. ■

Next, we show that globally reachable nodes in the digraph determine the SSD subspace in a finite number of iterations.

*Theorem 5.2: (Globally Reachable Nodes Find the SSD Subspace):* Given an arbitrary constant digraph, let $i$ be a globally reachable node and define $l = \max_{j \in \{1, \dots, M\}} \text{dist}(j, i)$. Then, under the P-SSD algorithm, $\mathcal{R}(C_k^i) = \mathcal{R}(C_{\text{SSD}})$ for all $k \geq l + 1$.

*Proof:* If $C_k^i = 0$ for some $k \in \mathbb{N}$, then based on Proposition 4.8 and Theorem A.1(a), we have $C_{\text{SSD}} = 0$ and consequently, $\mathcal{R}(C_k^i) = \mathcal{R}(C_{\text{SSD}})$. Now, suppose that for all $k \in \mathbb{N}$, $C_k^i \neq 0$ and has full column rank (cf. Theorem 4.5(a)). Since $i$ is a globally reachable node, for each node $j \in \{1, \dots, M\} \setminus \{i\}$, there exists a directed path with length $l_j = \text{dist}(j, i) < \infty$ to node $i$. Using Proposition 5.1 for $j \in \{1, \dots, M\} \setminus \{i\}$ and Theorem 4.5(c) for agent $i$ , we can write

$$\mathcal{R}(D(X_j)C_{l+1}^i) = \mathcal{R}(D(Y_j)C_{l+1}^i), \, \forall j \in \{1, \dots, M\}.$$

Moreover, since for all $j \in \{1, \dots, M\}$, $D(X_j)$, $D(Y_j)$, and $C_{l+1}^i$ have full column rank, there exist nonsingular square matrices $\{K_j\}_{j=1}^M$ such that

$$D(Y_j)C_{l+1}^i = D(X_j)C_{l+1}^i K_j, \, \forall j \in \{1, \dots, M\}. \tag{15}$$

Looking at (15) in a row-wise manner and only considering the signature data sets, one can write

$$D(Y_s)C_{l+1}^i = D(X_s)C_{l+1}^i K_j, \, \forall j \in \{1, \dots, M\}.$$

For any $p \neq q \in \{1, \dots, M\}$, we subtract (15) evaluated at $j = p$ and at $j = q$ to obtain $D(X_s)C_{l+1}^i(K_q - K_p) = 0$. Since $D(X_s)$ and $C_{l+1}^i$ have full column rank, this implies

$K := K_1 = \cdots = K_M$. Looking at (15) in a row-wise manner and considering the fact that

$$\bigcup_{j=1}^{M} \text{rows}([D(X_j), D(Y_j)]) = \text{rows}([D(X), D(Y)]),$$

we deduce $D(X)C^i_{l+1}K = D(Y)C^i_{l+1}$ and consequently

$$\mathcal{R}(D(X)C^i_{l+1}) = \mathcal{R}(D(Y)C^i_{l+1}).$$

This, together with Theorem A.1(b), implies $\mathcal{R}(C^i_{l+1}) \subseteq \mathcal{R}(C_{\text{SSD}})$. This inclusion along with Proposition 4.8 yields $\mathcal{R}(C^i_{l+1}) = \mathcal{R}(C_{\text{SSD}})$. Finally, for every $k \geq l+1$, Lemma 4.7 implies $\mathcal{R}(C^i_k) \subseteq \mathcal{R}(C^i_{l+1}) = \mathcal{R}(C_{\text{SSD}})$, which along with Proposition 4.8 yields $\mathcal{R}(C^i_k) = \mathcal{R}(C_{\text{SSD}})$. ∎

Theorem 5.2 shows that the globally reachable nodes identify the SSD subspace. Next, we use this fact to derive guarantees for agreement of the range space of all agents' matrices on the SSD subspace over strongly connected networks (we refer to this as P-SSD reaching consensus).

*Theorem 5.3: (Consensus on SSD Subspace and Time Complexity of P-SSD over Strongly Connected Digraphs):* Given a strongly connected digraph with diameter $d$,

(a) P-SSD reaches consensus in at most $d+1$ iterations: specifically, for all $k \geq d+1$ and all $i \in \{1, \ldots, M\}$,

$$\mathcal{R}(C^i_k) = \mathcal{R}(C_{\text{SSD}}),$$
$$\mathcal{R}(D(X)C^i_k) = \mathcal{R}(D(Y)C^i_k);$$

(b) the P-SSD algorithm terminates after at most $d+2$ iterations, i.e, $\text{flag}^i_{d+2} = 1$ for each $i \in \{1, \ldots, M\}$.

*Proof:* Regarding (a), the proof of consensus on the SSD subspace readily follows from Theorem 5.2 by noting that any node is globally reachable from any other node through a path with at most $d$ edges. The rest of the statement is a corollary of this fact together with Theorem A.1(a).

Regarding (b), one needs to establish that not only the range space of the agents' matrices remains the same but also that the P-SSD algorithm executes Steps 11-12, which means the matrices themselves also remain the same and flags become 1. Note that using (a) we deduce that $\mathcal{R}(C^i_{d+1}) = \mathcal{R}(C^i_{d+2})$ for all $i \in \{1, \ldots, M\}$. Hence, if $C^i_{d+1} = 0$, then $C^i_{d+2} = 0$, and the P-SSD algorithm does not execute Steps 8-9 but executes Steps 11-12 instead. Consequently, $\text{flag}^i_{d+2} = 1$ for every agent $i \in \{1, \ldots, M\}$. Suppose then that $C^i_{d+1} \neq 0$, with full column rank based on Theorem 4.5(a), and consequently $C_{\text{SSD}} \neq 0$ (also with full column rank, cf. Theorem A.1(a)). Using (a), we then deduce that in Step 5 of the P-SSD algorithm for agent $i$ at iteration $d+2$, we have $\mathcal{R}(D^i_{d+2}) = \mathcal{R}(C_{\text{SSD}})$. Since $D^i_{d+2}$ has full column rank by definition,

$$\mathcal{R}(D(X)D^i_{d+2}) = \mathcal{R}(D(Y)D^i_{d+2}).$$

Looking at this equation in a row-wise manner and considering the fact that $\text{rows}([D(X_i), D(Y_i)]) \subseteq \text{rows}([D(X), D(Y)])$,

$$\mathcal{R}(D(X_i)D^i_{d+2}I) = \mathcal{R}(D(Y_i)D^i_{d+2}I),$$

where $I$ is the identity matrix with appropriate size. This fact, together with definition of $E^i_{d+2}$ in Step 6 of the P-SSD algorithm, implies that $\mathcal{R}(I) \subseteq \mathcal{R}(E^i_{d+2})$ (cf. Theorem A.1(b)). Consequently, $E^i_{d+2}$ must be a nonsingular square matrix (cf. Theorem A.1(a)). Hence,

$$\sharp\text{cols}(D^i_{d+2}E^i_{d+2}) = \sharp\text{cols}(D^i_{d+2}) = \sharp\text{cols}(C_{\text{SSD}}) = \sharp\text{cols}(C^i_{d+1}).$$

Consequently, the P-SSD algorithm executes Steps 11-12 and we have $C^i_{d+2} = C^i_{d+1}$ and $\text{flag}^i_{d+2} = 1$ for every agent $i \in \{1, \ldots, M\}$, concluding the proof. ∎

In general, the P-SSD algorithm might terminate faster than the explicit upper bound given in Theorem 5.3. The main reason for this is that each agent depends on performing SSD on its own data and *usually* SSD can identify the maximal Koopman-invariant subspace in the span of dictionary with a moderate amount of data. The next remark shows that the floating point operation (FLOPs) complexity of the P-SSD algorithm is much lower for each processor that the SSD algorithm and hence the P-SSD subspace search runs much faster than SSD.

*Remark 5.4: (Floating Point Operation (FLOP) Complexity of P-SSD):* For the (standard) case that $N \gg N_d$, let $N_i$ be the number of data snapshots available to agent $i \in \{1, \ldots, M\}$. Considering the fact that the most time consuming operation in Algorithm 2 is Step 6, one can use [28, Remark 5.2] and deduce that each iteration of the P-SSD algorithm takes $O(N_i N_d^3)$ FLOPs for agent $i$. Based on Theorem 5.3(b), agent $i$ performs at most $O(N_i N_d^3 d)$ to find the SSD subspace. In case the data is uniformly distributed among the agents, for each $i \in \{1, \ldots, M\}$,

$$N_i = O\Big(\sharp\text{rows}(X_s) + \frac{N - \sharp\text{rows}(X_s)}{M}\Big) = O(N/M),$$

where in the last equality we have used $\sharp\text{rows}(X_s) = O(N_d)$ (in fact one can find signature data with $\sharp\text{rows}(X_s) \leq 2N_d$ as a consequence of Assumption 3.1). Hence, the FLOPs complexity of each agent with data uniformly distributed across the agents is $O(dNN_d^3/M)$. This gives a factor of $d/M$ reduction when compared to the FLOP complexity $O(NN_d^3)$ of SSD. If the processors are not homogeneous, then the uniform distribution of the data is not optimal. In general, the optimal distribution minimizes the maximum time taken for processors to complete one algorithm iteration, which means that faster processors should get more data. In practice, our simulations show that P-SSD runs drastically faster than the worst-case bound. □

*Remark 5.5: (Communication Complexity of P-SSD):* Given a strongly connected digraph with $M$ nodes and diameter $d$, agent $i$ transmits $|\mathcal{N}_{\text{out}}(i)|$ matrices with maximum size of $N_d \times N_d$. Thus, considering the real numbers as basic messages, and using the fact that the algorithm terminates after at most $d+1$ iterations, the communication complexity of the P-SSD algorithm is

$$O(dN_d^2 \sum_{i=1}^{M} |\mathcal{N}_{\text{out}}(i)|) = O(dN_d^2 E),$$

where $E$ is the number of edges in the digraph. In most conventional parallel computing units, the processors communicate through a shared bus. Hence, at iteration $k$ of the P-SSD algorithm, the $i$th agent only sends one message comprised of the matrix $C_k^i$ to the bus. Therefore, over such networks, the communication complexity reduces to $O(dN_d^2M)$. □

*Remark 5.6: (Approximated P-SSD):* The original dictionary might not contain nontrivial Koopman-invariant subspaces. For such cases, one can look for *approximate* informative invariant subspaces by replacing Step 6 of P-SSD with the Approximated-SSD algorithm [28, Section VII], which approximates Koopman-invariant subspaces given a tuning parameter $\epsilon$. This modification gives rise to the Approximated P-SSD algorithm for which a similar set of results to the ones stated here for the P-SSD algorithm regarding equilibria, finite termination and eventual agreement of the matrix iterates can be established (we omit them for space reasons). The agents' identified subspaces satisfy the accuracy bounds provided in [28, Section VII] based on their local data. □

## VI. Robustness Against Packet Drops and Time-Varying Networks

The parallel nature of P-SSD enables the use of parallel and distributed processing hardware, such as GPUs and clusters of processors. The communication between processors required by such hardware is subject to packet drops or communication failures. Moreover, there are instances when some processors are needed for other tasks or go offline temporarily, resulting in time-varying networks. To address these issues, we investigate the robustness of the P-SSD algorithm against packet drops in the communication network. To tackle this problem, we model the network by a time-varying digraph where a dropped packet from agent $i$ to agent $j$ at time $k \in \mathbb{N}$ corresponds to the absence of edge $(i, j)$ at time $k$ from the digraph at that time. The next result shows that if the digraph remains repeatedly jointly strongly connected, the agents executing the P-SSD algorithm reach a consensus on the SSD subspace in finite time.

*Theorem 6.1: (Consensus on SSD Subspace and Finite-Time Convergence of P-SSD over Repeatedly Jointly Strongly Connected Digraphs):* Given a time-varying repeatedly jointly strongly connected digraph $\{G_k = (\{1, \dots, M\}, E_k)\}_{k=1}^{\infty}$, the P-SSD algorithm reaches a consensus equilibrium on the SSD subspace in finite time, i.e., there exists $l \in \mathbb{N}$ such that for every iteration $k \geq l$, $C_k^i = C_l^i$ for all $i \in \{1, \dots, M\}$ with

$$\mathcal{R}(C_l^1) = \mathcal{R}(C_l^2) = \cdots = \mathcal{R}(C_l^M) = \mathcal{R}(C_{\text{SSD}}).$$

*Proof:* The fact that the P-SSD algorithm reaches an equilibrium is a direct consequence of Proposition 4.2, which states that there exists $l \in \mathbb{N}$ such that

$$C_k^i = C_l^i, \; \forall k \geq l, \; \forall i \in \{1, \dots, M\}. \tag{16}$$

Hence, we only need to show that the range spanned by these matrices corresponds to the SSD subspace. Since the digraph is repeatedly jointly strongly connected, there exists a closed "temporal" path after time $l$ that goes through every node of the digraph. By this we mean that there exist $L$, time instants $l < k_1 < k_2 < \cdots < k_L$ and node labels $p_1, \dots, p_L$ covering all of $\{1, \dots, M\}$ (note that some nodes might correspond to more that one label) such that

$$(p_i, p_{(i \bmod L)+1}) \in E_{k_i}, \; \forall i \in \{1, \dots, L\}.$$

Using (16), Lemma 4.7 at times $\{k_i\}_{i=1}^L$, and the fact that the path is closed, we deduce

$$\mathcal{R}(C_l^{p_1}) \subseteq \mathcal{R}(C_l^{p_L}) \subseteq \cdots \subseteq \mathcal{R}(C_l^{p_2}) \subseteq \mathcal{R}(C_l^{p_1}).$$

Hence, $\mathcal{R}(C_l^{p_1}) = \cdots = \mathcal{R}(C_l^{p_L})$. Since the path goes through every node and (16) again, we arrive at the consensus

$$\mathcal{R}(C_k^1) = \cdots = \mathcal{R}(C_k^M), \; \forall k \geq l. \tag{17}$$

It remains to show that this consensus is achieved on the SSD subspace. The inclusion $\mathcal{R}(C_{\text{SSD}}) \subseteq \mathcal{R}(C_k^i)$ for all $i \in \{1, \dots, M\}$ and $k \geq l$ follows from Proposition 4.8. To show the other inclusion, first note that if $C_k^i = 0$ for some $i \in \{1, \dots, M\}$ and $k \geq l$, then $C_{\text{SSD}} = 0$ and the proof follows. Suppose then that $C_k^i$'s are nonzero with full column rank (cf. Theorem 4.5(a)) for $k \geq l$. Based on Theorem 4.5(c), for every $i \in \{1, \dots, M\}$ and $k \geq l$, we have $\mathcal{R}(D(X_i)C_k^i) = \mathcal{R}(D(Y_i)C_k^i)$. Moreover, using (17) and the fact that all matrices $C_k^i$'s have full column rank, we have

$$\mathcal{R}(D(X_i)C_k^j) = \mathcal{R}(D(Y_i)C_k^j), \; \forall i, j \in \{1, \dots, M\},$$

for every $k \geq l$. Using this equality for $j = 1$ and $k = l$,

$$\mathcal{R}(D(X_i)C_l^1) = \mathcal{R}(D(Y_i)C_l^1), \; \forall i \in \{1, \dots, M\}.$$

Hence, for every $i \in \{1, \dots, M\}$ there exists a nonsingular square matrix $K_i$ such that

$$D(X_i)C_l^1 K_i = D(Y_i)C_l^1. \tag{18}$$

Now, looking at (18) in a row-wise manner, one obtains for the signature dictionary snapshots, $D(X_s)C_l^1 K_i = D(Y_s)C_l^1$, for $i \in \{1, \dots, M\}$, and hence

$$D(X_s)C_l^1(K_i - K_j) = 0, \; \forall i, j \in \{1, \dots, M\}.$$

Since $D(X_s)$ and $C_l^1$ have full column rank, we have $K := K_1 = \cdots = K_M$. Replacing $K_i$ by $K$ in (18) gives

$$D(X_i)C_l^1 K = D(Y_i)C_l^1, \; \forall i \in \{1, \dots, M\}.$$

Looking at this equation in a row-wise manner and since

$$\bigcup_{i=1}^{M} \text{rows}([D(X_i), D(Y_i)]) = \text{rows}([D(X), D(Y)]),$$

one can write $D(X)C_l^1 K = D(Y)C_l^1$ and consequently

$$\mathcal{R}(D(X)C_l^1) = \mathcal{R}(D(Y)C_l^1).$$

Using now Theorem A.1(b), we deduce $\mathcal{R}(C_l^1) \subseteq \mathcal{R}(C_{\text{SSD}})$. Using (16) and (17), we obtain that $\mathcal{R}(C_k^i) = \mathcal{R}(C_k^1) = \mathcal{R}(C_l^1) \subseteq \mathcal{R}(C_{\text{SSD}})$, for all $i \in \{1, \dots, M\}$ and $k \geq l$, and this concludes the proof. ■

An interesting interpretation of the result in Theorem 6.1 is that, for time-invariant networks subject to failures in the communication links, as long as the agents re-connect at least once within some uniform time period, the P-SSD algorithm identifies the SSD subspace in a finite number of iterations.

*Remark 6.2: (Using the Agents' Matrix Iterates to Find Maximal Koopman-Invariant Subspaces and Eigenfunctions):* After the agents reach consensus on the SSD subspace, they can use their computed matrix $C$ instead of $C_{\text{SSD}}$ to find the dictionary $\tilde{D}(x) = \tilde{D}(x)C$, for $x \in \mathcal{M}$. Any agent $i \in \{1, \dots, M\}$ can use its P-SSD matrix to find

$$K_{\text{P-SSD}} = \tilde{D}(X_i)^\dagger \tilde{D}(Y_i) = \tilde{D}(X_s)^\dagger \tilde{D}(Y_s).$$

Note that this is also equal to the square matrix $K_{\text{SSD}}$ found by SSD, cf. (A.3). Importantly, all the results for SSD subspaces in [28] are also valid for the new dictionary $\tilde{D}$. For space reasons, we omit those results here and only mention that under some mild conditions on data sampling, $\tilde{D}(x)$ spans the maximal Koopman-invariant subspace in $\operatorname{span}(D(x))$ and $\tilde{D}(x)w$ is an eigenfunction of the Koopman operator almost surely, given $K_{\text{P-SSD}}w = \lambda w$ with $\lambda \in \mathbb{C}$ and $w \in \mathbb{C}^{\sharp\text{cols}(K_{\text{SSD}})} \setminus \{0\}$ (see [28, Theorems 5.7-5.8] for more information). □

## VII. Simulation Results

Here, we provide three examples[3] to demonstrate the properties and effectiveness of the P-SSD algorithm[4].

*Example 7.1: (Unstable Nonlinear System):* Consider the discrete-time system with state $x = [x_1, x_2]$,

$$x_1^+ = 1.2\, x_1 \tag{19a}$$

$$x_2^+ = \sqrt[3]{0.8\, x_2^3 + 8\, x_1^2 + 0.1}\,. \tag{19b}$$

The system can be transformed into a discrete-time polyflow [38] by the nonlinear transformation $[x_1, x_2] \mapsto [x_1, x_2^3]$. We aim to find the informative Koopman eigenfunctions and invariant subspaces. we sample $N = 10^6$ data snapshots with initial conditions in the state space $\mathcal{M} = [-3, 3] \times [-3, 3]$ (dense sampling ensures that the properties identified here hold almost surely over the whole state space [28]). We use the dictionary $D$ with $N_d = 15$ comprised of all distinct monomials up to degree 4 of the form $\prod_{i=1}^4 y_i$, with $y_i \in \{1, x_1, x_2\}$ for $i \in \{1, \dots, 4\}$.

To identify the maximal Koopman-invariant subspace in $\operatorname{span}(D(x))$, we implement the SSD algorithm, and the P-SSD algorithm with $M \in \{5, 20, 100\}$ agents communicating according to a directed ring graph with diameter $d = M - 1$. For the P-SSD strategy, we use the first 15 data snapshots in our database as signature data snapshots and distribute the rest of the data evenly among the agents. Both strategies are implemented on a single computer using MATLAB®. We calculate the time elapsed for each iteration of P-SSD as the maximum time taken by agents to execute the algorithm in that iteration using the `tic` and `toc` commands. Since we use finite precision, we apply the approximation provided in Remark 4.6 with $\epsilon_\cap = 10^{-12}$.

For all $M \in \{5, 20, 100\}$, the P-SSD algorithm reaches the consensus equilibrium after 1 iteration and terminates (based on Remark 4.4) after 2 iterations, which is significantly faster than the bounds provided in Theorem 5.3. We have also observed in simulations that packet drops do not delay consensus. Both P-SSD and SSD algorithms correctly identify the 7-dimensional subspace spanned by $\{1, x_1, x_1^2, x_1^3, x_2^3, x_1^4, x_1 x_2^3\}$ as the maximal Koopman-invariant subspace in $\operatorname{span}(D(x))$. Table I shows the time employed by the algorithms to find it. The P-SSD strategy with $M = 5$, $M = 20$, and $M = 100$, is 80%, 96%, and 99% faster than SSD, resp.

TABLE I: Time elapsed to identify the maximal Koopman-invariant subspace in $\operatorname{span}(D)$ associated with the dynamics (19).

| **Method** | SSD | P-SSD (5) | P-SSD (20) | P-SSD (100) |
|---|---|---|---|---|
| **Time (ms)** | 2175 | 439 | 88 | 17 |

Using the output matrix of any of the agents, cf. Remark 6.2, we build the invariant dictionary $\tilde{D}$ and matrix $K_{\text{P-SSD}}$. Table II shows the Koopman eigenfunctions and their corresponding eigenvalues calculated using the eigendecomposition of $K_{\text{P-SSD}}$. One can verify analytically using (19) that the eigenfunctions in Table II evolve linearly in time. Even though the function $x_2$ does not belong to the span of the Koopman-invariant subspace, the Koopman eigenfunctions fully capture the behavior of the system since the functions $x_1$ and $x_2^3$ do belong. Hence, one can use (5) to predict the evolution of any function in form of (4) or one simply can use

$$\tilde{D}(x^+) = \tilde{D}(x) K_{\text{P-SSD}},\ \forall x \in \mathcal{M},$$

to describe the behavior of (19) in a linear way.

TABLE II: Identified eigenfunctions and eigenvalues of the Koopman operator associated with the dynamics (19).

| **Eigenfunction** | **Eigenvalue** |
|---|---|
| $\phi_1(x) = 1$ | $\lambda_1 = 1$ |
| $\phi_2(x) = x_1$ | $\lambda_2 = 1.2$ |
| $\phi_3(x) = x_1^2$ | $\lambda_3 = 1.44$ |
| $\phi_4(x) = x_1^3$ | $\lambda_4 = 1.728$ |
| $\phi_5(x) = 2\, x_2^3 - 25\, x_1^2 - 1$ | $\lambda_5 = 0.8$ |
| $\phi_6(x) = x_1^4$ | $\lambda_6 = 2.0736$ |
| $\phi_7(x) = 2\, x_1 x_2^3 - 25\, x_1^3 - x_1$ | $\lambda_7 = 0.96$ |

To demonstrate the effectiveness of our method in long-term predictions, following the Extended Dynamic Mode Decomposition (EDMD) method [23] and Remark 6.2, and given an arbitrary dictionary $\mathcal{D}$, we define its linear prediction matrix as $K = \mathcal{D}(X)^\dagger \mathcal{D}(Y)$. We also define the following relative and angle error functions on a trajectory $\{x(i)\}_{i=0}^L$

$$\begin{aligned} E_{\text{relative}}(k) &= \frac{\left\| \mathcal{D}(x(k)) - \mathcal{D}(x_0)K^k \right\|_2}{\|\mathcal{D}(x(k))\|_2} \times 100, \\ E_{\text{angle}}(k) &= \angle\big(\mathcal{D}(x(k)), \mathcal{D}(x_0)K^k\big). \end{aligned} \tag{20}$$

We compare the prediction accuracy on the original dictionary $D$ with the dictionary $\tilde{D}$ identified by P-SSD on 1000 trajectories with length $L = 15$. In order to make sure the trajectories remain in the space on which we have trained our methods, we sample the initial conditions from $[-0.1, 0.1] \times [-3, 3]$. Figure 1 shows the median and range between the first and third quartiles of the relative and angle prediction errors for $D$ and $\tilde{D}$. The P-SSD method has zero prediction error since it identifies and predicts the evolutions on Koopman-invariant subspaces. In contrast, the prediction errors on the original dictionary are significantly large, even for short time steps. □

[3]We refer the interested reader to the online version [35] for an additional example where we show the performance of P-SSD under packet drops.

[4]We intentionally use low-dimensional systems with sparse eigenfunctions to facilitate a complete in-depth presentation of the results. However, we should point out that the results are valid for high-dimensional systems with no conditions on the sparsity of invariant subspaces.

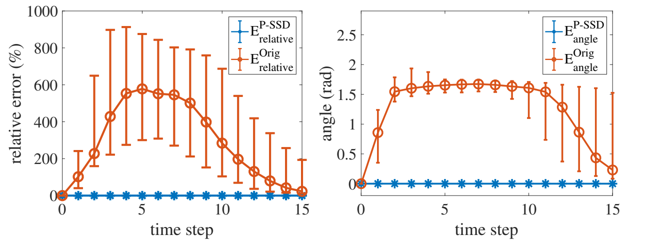

Fig. 1: Median and range between first and third quartiles of relative (left) and angle (right) prediction errors for the original dictionary and the dictionary identified by P-SSD for system (19) on 1000 trajectories of length $L = 15$ with initial conditions randomly selected from $[-0.1, 0.1] \times [-3, 3]$.

*Example 7.2: (Van der Pol Oscillator):* Here, we provide an example of the approximation of Koopman eigenfunctions and invariant subspaces when the original dictionary does not contain exact informative eigenfunctions. Consider the Van der Pol oscillator with state $x = [x_1, x_2]^T$,

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -x_1 + (1 - x_1^2)x_2. \end{aligned} \tag{21}$$

To gather data, we run $10^3$ simulations for $5s$ with initial conditions uniformly selected from $\mathcal{M} = [-4, 4] \times [-4, 4]$. We sample the trajectory data with time step $\Delta t = 0.05s$ resulting in $N = 10^5$ data snapshots. Moreover, we use the dictionary $D$ with $N_d = 45$ comprised of all distinct monomials up to degree 8 of the form $\prod_{i=1}^{8} y_i$, with $y_i \in \{1, x_1, x_2\}$ for $i \in \{1, \dots, 8\}$. To avoid numerical problems caused by finite-precision errors, we form the matrix $[D(X)^T, D(Y)^T]^T$ and scale each dictionary function such that the norm of columns of the aforementioned matrix become equal. Note that this scaling does not change the functional space spanned by the dictionary since each function in the dictionary is scaled by a nonzero number in $\mathbb{R}$. The communication network is modeled by a complete digraph with $M = 20$ processors resembling a GPU. In addition, we use 1000 snapshots randomly selected from our database as the signature data snapshots and distribute the rest of the data evenly among the processors.

Note that the dictionary $D$ only contains the trivial Koopman eigenfunction $\phi(x) \equiv 1$ with eigenvalue $\lambda = 1$. Consequently, the P-SSD algorithm results in a one-dimensional Koopman-invariant subspace providing exact prediction but no information about the dynamics. To address this issue, we use instead the Approximated P-SSD algorithm presented in Remark 5.6 with $\epsilon = 0.005$. Moreover, we set $\epsilon_{\cap} = 0.005$ following Remark 4.6. This algorithm reaches a consensus equilibria after 4 iterations identifying a 4-dimensional subspace. It is worth mentioning that we could not implement the Approximated SSD algorithm from [28] on the same dataset since it performs SVD on the whole dataset and requires a large memory that is beyond our computational resources (also, performing SVD on such large datasets may result in large round-off errors and inaccurate results).

Using the output matrix of any of the agents, cf. Remark 6.2, we find the dictionary $\tilde{D}$ with $\tilde{N}_d = 4$ functions and create the approximated P-SSD matrix as $K^{\text{apprx}}_{\text{P-SSD}} = \tilde{D}(X)^{\dagger}\tilde{D}(Y)$. Moreover, we find the Koopman eigenfunctions approximated by the eigendecomposition of $K^{\text{apprx}}_{\text{P-SSD}}$. Approximated P-SSD finds the only exact Koopman eigenfunction ($\phi(x) \equiv 1$ with eigenvalue $\lambda = 1$) in the span of the original dictionary correctly. Figure 2 shows the leading nontrivial approximated eigenfunction corresponding to the largest eigenvalue, which captures the behavior of the Van der Pol oscillator.

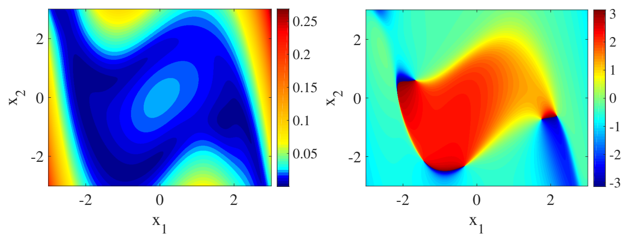

Fig. 2: Absolute value (left) and angle (right) of the approximated eigenfunction corresponding to eigenvalue $0.9647 + 0.018j$.

To show the advantage of our method in long-term prediction, we use the error functions presented in (20) on 1000 trajectories with length $L = 20$ time steps and initial conditions uniformly taken from $[-4, 4] \times [-4, 4]$. Figure 3 shows the median and the range between the first and third quartiles of the predictions errors for the dictionary $\tilde{D}$ identified by Approximated P-SSD and the original dictionary $D$ over the aforementioned trajectories. According to Figure 3, Approximated P-SSD has a clear advantage in long-term prediction. After 20 time steps the median of the relative prediction errors for the original dictionary is 60% while the same error for the dictionary identified with Approximated P-SSD is 5%. Moreover, the median of the angle errors after 20 time steps are 0.5 and 0.05 radians for $D$ and $\tilde{D}$, resp. $\square$

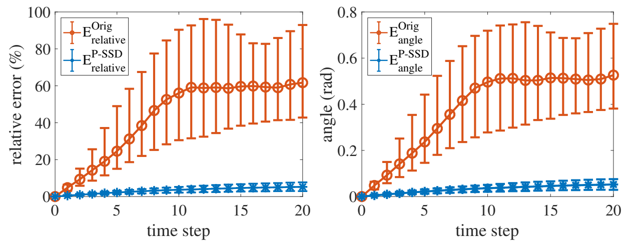

Fig. 3: Median and range between first and third quartiles of relative (left) and angle (right) prediction errors for the original dictionary and the dictionary identified with Approximated P-SSD for system (21) on 1000 trajectories of length $L = 20$ with initial conditions randomly selected from $\mathcal{M}$.

*Example 7.3: (Chaotic Lorenz System):* Consider the chaotic Lorenz system

$$\begin{aligned} \dot{x} &= 10(y - x) \\ \dot{y} &= x(28 - z) - y \\ \dot{z} &= xy - (8/3)z, \end{aligned} \tag{22}$$

with state $s = [x, y, z]^T$ belonging to state space $\mathcal{M} = [-20, 20] \times [-30, 30] \times [0, 50]$. Our strategy for sampling from $\mathcal{M}$, the number of samples and signature data, and the processor network are similar to Example 7.2. We use the dictionary $D$ with $N_d = 84$ comprised of all distinct monomials up to degree 6 of the form $\prod_{i=1}^{6} y_i$, with $y_i \in \{1, x, y, z\}$ for $i \in \{1, \dots, 6\}$. To avoid numerical problems caused by round-off errors, we scale the dictionary elements as in Example 7.2. Note that the dictionary $D$ only contains the trivial Koopman eigenfunction $\phi(x) \equiv 1$ with eigenvalue $\lambda = 1$. Consequently, the P-SSD algorithm results in a one-dimensional Koopman-invariant subspace providing exact prediction but no information about the dynamics. To address this issue, we use instead the Approximated P-SSD algorithm presented in Remark 5.6 with $\epsilon = 0.001$. Moreover, we set $\epsilon_{\cap} = 0.001$ following Remark 4.6. This algorithm reaches a consensus equilibria after 3 iterations identifying a 2-dimensional subspace. It is worth mentioning that we could not implement the Approximated SSD algorithm from [28] on the same dataset because of its large computational requirements..

Using the output matrix of any of the agents, cf. Remark 6.2, we find the dictionary $\tilde{D}$ with $\tilde{N}_d = 2$ and two approximated eigenfunctions in its span. Approximated P-SSD finds the only exact Koopman eigenfunction ($\phi(x) \equiv 1$ with eigenvalue $\lambda = 1$) in the span of the original dictionary correctly. It also approximates another real-valued eigenfunction with eigenvalue $\lambda \approx 0.46$ which predicts that the trajectories converge to an *approximated* invariant set since $|\lambda| < 1$.

To show the advantage of our method in long-term prediction, we use the error functions presented in (20) on 1000 trajectories with length $L = 20$ time steps and initial conditions uniformly taken from $\mathcal{M} = [-20, 20] \times [-3, 30] \times [0, 50]$ and compare the prediction accuracy of the dictionary $\tilde{D}$ identified by Approximated P-SSD versus the original dictionary $D$. Figure 4 shows the median and the range between the first and third quartiles of the aforementioned predictions errors, indicating the advantage of P-SSD in long-term prediction. $\square$

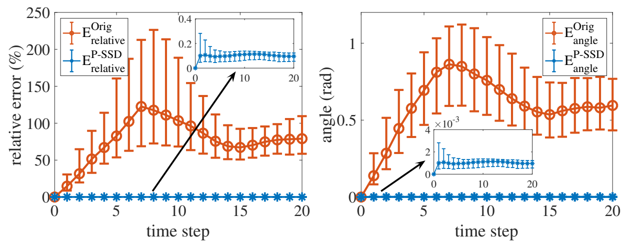

Fig. 4: Median and range between first and third quartiles of relative (left) and angle (right) prediction errors for dictionary $\tilde{D}$ identified by Approximated P-SSD and the original dictionary $D$ for system (22) on 1000 trajectories of length $L = 20$ with initial conditions randomly selected from $\mathcal{M}$.

## VIII. Conclusions

We have proposed a data-driven strategy, termed P-SSD algorithm, to identify Koopman eigenfunctions and invariant subspaces corresponding to an unknown dynamics. The strategy runs in parallel over a network of processors that communicate over a digraph. We have thoroughly characterized the algorithm convergence and complexity properties. In particular, we have identified conditions on the network connectivity that ensure that the P-SSD's output coincide with that of the SSD algorithm (a centralized method that under mild conditions on data sampling provably identifies the maximal Koopman-invariant subspace in an arbitrary finite-dimensional functional space) when run over the whole set of data snapshots. The parallel nature of P-SSD makes it run significantly faster than its centralized counterpart. We have also established the robustness of P-SSD against communication failures and packet drops. Future work will explore to what extent the data available to the individual agents and its density can be employed in lieu of the common signature data, the development of noise-resilient counterparts of the proposed algorithms, the synthesis of distributed strategies that can work with streaming data sets, and the construction of dictionaries containing informative Koopman eigenfunctions leading to the approximation of the dynamics with accuracy guarantees.

## References

[1] M. Haseli and J. Cortés, "Fast identification of Koopman-invariant subspaces: parallel symmetric subspace decomposition," in *American Control Conference*, Denver, CO, July 2020, pp. 4545–4550.
[2] 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.
[3] B. O. Koopman and J. V. Neumann, "Dynamical systems of continuous spectra," *Proceedings of the National Academy of Sciences*, vol. 18, no. 3, pp. 255–263, 1932.
[4] I. Mezić, "Spectral properties of dynamical systems, model reduction and decompositions," *Nonlinear Dynamics*, vol. 41, no. 1-3, pp. 309–325, 2005.
[5] M. Budišić, R. Mohr, and I. Mezić, "Applied Koopmanism," *Chaos*, vol. 22, no. 4, p. 047510, 2012.
[6] A. Mauroy and I. Mezić, "Global stability analysis using the eigenfunctions of the Koopman operator," *IEEE Transactions on Automatic Control*, vol. 61, no. 11, pp. 3356–3369, 2016.
[7] A. Mauroy and J. Goncalves, "Linear identification of nonlinear systems: A lifting technique based on the Koopman operator," in *IEEE Conf. on Decision and Control*, Las Vegas, NV, Dec. 2016, pp. 6500–6505.
[8] S. Klus, F. Nüske, P. Koltai, H. Wu, I. Kevrekidis, C. Schütte, and F. Noé, "Data-driven model reduction and transfer operator approximation," *Journal of Nonlinear Science*, vol. 28, no. 3, pp. 985–1010, 2018.
[9] A. Alla and J. N. Kutz, "Nonlinear model order reduction via dynamic mode decomposition," *SIAM Journal on Scientific Computing*, vol. 39, no. 5, pp. B778–B796, 2017.
[10] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. S. Henningson, "Spectral analysis of nonlinear flows," *Journal of Fluid Mechanics*, vol. 641, pp. 115–127, 2009.
[11] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, 2018.
[12] S. Peitz and S. Klus, "Koopman operator-based model reduction for switched-system control of PDEs," *Automatica*, vol. 106, pp. 184–191, 2019.
[13] B. Huang, X. Ma, and U. Vaidya, "Feedback stabilization using Koopman operator," in *IEEE Conf. on Decision and Control*, Miami Beach, FL, Dec. 2018, pp. 6434–6439.
[14] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of Koopman eigenfunctions for control," *arXiv preprint arXiv:1707.01146*, 2017.
[15] D. Goswami and D. A. Paley, "Global bilinearization and controllability of control-affine nonlinear systems: a Koopman spectral approach," in *IEEE Conf. on Decision and Control*. IEEE, 2017, pp. 6107–6112.
[16] G. Mamakoukas, M. Castano, X. Tan, and T. Murphey, "Local Koopman operators for data-driven control of robotic systems," in *Robotics: Science and Systems*, Freiburg, Germany, June 2019.
[17] A. Narasingam and J. S. Kwon, "Data-driven feedback stabilization of nonlinear systems: Koopman-based model predictive control," *arXiv preprint arXiv:2005.09741*, 2020.
[18] S. H. Son, A. Narasingam, and J. S. Kwon, "Handling plant-model mismatch in Koopman Lyapunov-based model predictive control via offset-free control framework," *arXiv preprint arXiv:2010.07239*, 2020.
[19] A. Salova, J. Emenheiser, A. Rupe, J. P. Crutchfield, and R. M. DSouza, "Koopman operator and its approximations for systems with symmetries," *Chaos*, vol. 29, no. 9, p. 093128, 2019.
[20] A. Mesbahi, J. Bu, and M. Mesbahi, "Nonlinear observability via Koopman analysis: Characterizing the role of symmetry," *Automatica*, p. 109353, 2020.
[21] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of Fluid Mechanics*, vol. 656, pp. 5–28, 2010.
[22] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, "On dynamic mode decomposition: theory and applications," *Journal of Computational Dynamics*, vol. 1, no. 2, pp. 391–421, 2014.
[23] 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.

[24] M. Korda and I. Mezić, "On convergence of extended dynamic mode decomposition to the Koopman operator," *Journal of Nonlinear Science*, vol. 28, no. 2, pp. 687–710, 2018.
[25] S. L. Brunton, B. W. Brunton, J. L. Proctor, and J. N. Kutz, "Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control," *PLOS One*, vol. 11, no. 2, pp. 1–19, 2016.
[26] M. Korda and I. Mezić, "Optimal construction of Koopman eigenfunctions for prediction and control," 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02278835
[27] S. Pan, N. Arnold-Medabalimi, and K. Duraisamy, "Sparsity-promoting algorithms for the discovery of informative Koopman invariant subspaces," *arXiv preprint arXiv:2002.10637*, 2020.
[28] M. Haseli and J. Cortés, "Learning Koopman eigenfunctions and invariant subspaces from data: Symmetric Subspace Decomposition," *https://arxiv.org/abs/1909.01419*, 2020.
[29] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis, "Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator," *Chaos*, vol. 27, no. 10, p. 103111, 2017.
[30] N. Takeishi, Y. Kawahara, and T. Yairi, "Learning Koopman invariant subspaces for dynamic mode decomposition," in *Conference on Neural Information Processing Systems*, 2017, pp. 1130–1140.
[31] E. Yeung, S. Kundu, and N. Hodas, "Learning deep neural network representations for Koopman operators of nonlinear dynamical systems," in *American Control Conference*, Philadelphia, PA, July 2019, pp. 4832–4839.
[32] S. E. Otto and C. W. Rowley, "Linearly recurrent autoencoder networks for learning dynamics," *SIAM Journal on Applied Dynamical Systems*, vol. 18, no. 1, pp. 558–593, 2019.
[33] F. Bullo, J. Cortés, and S. Martinez, *Distributed Control of Robotic Networks*, ser. Applied Mathematics Series. Princeton University Press, 2009.
[34] J. Liu, A. S. Morse, A. Nedić, and T. Başar, "Exponential convergence of a distributed algorithm for solving linear algebraic equations," *Automatica*, vol. 83, pp. 37–46, 2017.
[35] M. Haseli and J. Cortés, "Parallel learning of Koopman eigenfunctions and invariant subspaces for accurate long-term prediction," *https://arxiv.org/abs/2005.06138*, 2020.
[36] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann, 1997.
[37] D. Peleg, *Distributed Computing. A Locality-Sensitive Approach*, ser. Monographs on Discrete Mathematics and Applications. SIAM, 2000.
[38] R. M. Jungers and P. Tabuada, "Non-local linearization of nonlinear differential equations via polyflows," in *American Control Conference*, Philadelphia, PA, 2019, pp. 1906–1911.
[39] M. Haseli and J. Cortés, "Efficient identification of linear evolutions in nonlinear vector fields: Koopman invariant subspaces," in *IEEE Conf. on Decision and Control*, Nice, France, Dec. 2019, pp. 1746–1751.

## Appendix A
## Properties of the SSD Algorithm

Here, we gather some important results from [28], [39] regarding the SSD algorithm and its output.

*Theorem A.1: (Properties of the SSD's Output [28]):* Let $C_{\mathrm{SSD}} = \mathrm{SSD}(D(X), D(Y))$ be the output of the SSD algorithm applied on $D(X), D(Y)$. Given Assumption 3.1,

(a) $C_{\mathrm{SSD}}$ is either 0 or has full column rank and satisfies $\mathcal{R}(D(X)C_{\mathrm{SSD}}) = \mathcal{R}(D(Y)C_{\mathrm{SSD}})$;
(b) the subspace $\mathcal{R}(D(X)C_{\mathrm{SSD}})$ is maximal, in the sense that, for any matrix $E$ with $\mathcal{R}(D(X)E) = \mathcal{R}(D(Y)E)$, we have $\mathcal{R}(D(X)E) \subseteq \mathcal{R}(D(X)C_{\mathrm{SSD}})$ and $\mathcal{R}(E) \subseteq \mathcal{R}(C_{\mathrm{SSD}})$.

Theorem A.1 provides the symmetric range equality needed for identification of Koopman-invariant subspaces. Moreover, it ensures the maximality of such symmetric subspaces.

*Lemma A.2: (Monotonicity of SSD Output with Respect to Data Addition [28]):* Let $D(X), D(Y)$ and $D(\hat{X}), D(\hat{Y})$ be two pairs of data snapshots such that

$$\mathrm{rows}\big([D(X), D(Y)]\big) \subseteq \mathrm{rows}\big([D(\hat{X}), D(\hat{Y})]\big), \tag{A.1}$$

and for which Assumption 3.1 holds. Then

$$\mathcal{R}(\mathrm{SSD}([D(\hat{X}), D(\hat{Y})])) \subseteq \mathcal{R}(\mathrm{SSD}(D(X), D(Y))).$$

For $C_{\mathrm{SSD}} \neq 0$, we define the reduced dictionary

$$\tilde{D}(x) = D(x)C_{\mathrm{SSD}}, \quad \forall x \in \mathcal{M}. \tag{A.2}$$

Under some mild conditions on data sampling, the identified subspace $\mathrm{span}(\tilde{D}(x))$ converges to the *maximal Koopman-invariant subspace* in $\mathrm{span}(D(x))$ almost surely [28, Theorem 5.8] as the number of samples goes to infinity.

Based on Theorem A.1(a) and Assumption 3.1, we deduce there exists a nonsingular square matrix $K_{\mathrm{SSD}}$ such that

$$\tilde{D}(Y) = \tilde{D}(X)K_{\mathrm{SSD}}. \tag{A.3}$$

The eigendecomposition of $K_{\mathrm{SSD}}$ specifies the functions in $\mathrm{span}(\tilde{D})$ that evolve linearly in time, i.e., given $K_{\mathrm{SSD}}w = \lambda w$ with $\lambda \in \mathbb{C}$ and $w \in \mathbb{C}^{\sharp\mathrm{cols}(K_{\mathrm{SSD}})} \setminus \{0\}$, we have

$$\tilde{D}(Y)w = \lambda\tilde{D}(X)w. \tag{A.4}$$

The next result shows that the eigendecomposition of $K_{\mathrm{SSD}}$ captures all the functions in the span of the original dictionary that evolve linearly in time according to the available data.

*Theorem A.3: (Identification of Linear Evolutions using the SSD Algorithm [39]):* Under Assumption 3.1, $K_{\mathrm{SSD}}w = \lambda w$ for some $\lambda \in \mathbb{C}$ and $w \in \mathbb{C}^{\sharp\mathrm{cols}(K_{\mathrm{SSD}})\setminus\{0\}}$ iff there exists $v \in \mathbb{C}^{N_d}$ such that $D(Y)v = \lambda D(X)v$. In addition $v = C_{\mathrm{SSD}}w$.

Theorem A.3 shows that every function $\phi$ in the span of $D$ that satisfies $\phi(x^+) = \phi(T(x)) = \lambda\phi(x)$ for every $x \in \mathrm{rows}(X)$ is in turn in the span of $\tilde{D}$ and corresponds to an eigenvector of $K_{\mathrm{SSD}}$. This does necessarily mean that $\phi$ is an eigenfunction of the Koopman operator since its temporal linear evolution might not hold for all $x \in \mathcal{M}$. Under reasonable assumptions on data sampling, the identified functions are Koopman eigenfunctions almost surely [28, Theorem 5.7].

**Masih Haseli** was born in Kermanshah, Iran in 1991. He received the B.Sc. degree, in 2013, and M.Sc. degree, in 2015, both in Electrical Engineering from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran. In 2017, he joined the University of California, San Diego to pursue the Ph.D. degree in Mechanical and Aerospace Engineering. His research interests include system identification, nonlinear systems, network systems, data-driven modeling and control, and distributed computing. Mr. Haseli was the recipient of the bronze medal in Iran's national mathematics competition in 2014.

**Jorge Cortés** (M'02, SM'06, F'14) received the Licenciatura degree in mathematics from Universidad de Zaragoza, Zaragoza, Spain, in 1997, and the Ph.D. degree in engineering mathematics from Universidad Carlos III de Madrid, Madrid, Spain, in 2001. He held postdoctoral positions with the University of Twente, Twente, The Netherlands, and the University of Illinois at Urbana-Champaign, Urbana, IL, USA. He was an Assistant Professor with the Department of Applied Mathematics and Statistics, University of California, Santa Cruz, CA, USA, from 2004 to 2007. He is currently a Professor in the Department of Mechanical and Aerospace Engineering, University of California, San Diego, CA, USA. He is a Fellow of IEEE and SIAM. His current research interests include distributed control and optimization, network science, nonsmooth analysis, reasoning and decision making under uncertainty, network neuroscience, and multi-agent coordination in robotic, power, and transportation networks.