mathematics

MDPI

Article

Identification of Nonlinear Systems Using the Infinitesimal Generator of the Koopman Semigroup—A Numerical Implementation of the Mauroy–Goncalves Method

Zlatko Drmač [1,*], Igor Mezić [2] and Ryan Mohr [3]

[1] Department of Mathematics, Faculty of Science, University of Zagreb, 10000 Zagreb, Croatia
[2] Department of Mechanical Engineering and Mathematics, University of California, Santa Barbara, CA 93106, USA; mezic@ucsb.edu
[3] AIMdyn, Inc., Santa Barbara, CA 93101, USA; mohrr@aimdyn.com
* Correspondence: drmac@math.hr

Abstract: Inferring the latent structure of complex nonlinear dynamical systems in a data driven setting is a challenging mathematical problem with an ever increasing spectrum of applications in sciences and engineering. Koopman operator-based linearization provides a powerful framework that is suitable for identification of nonlinear systems in various scenarios. A recently proposed method by Mauroy and Goncalves is based on lifting the data snapshots into a suitable finite dimensional function space and identification of the infinitesimal generator of the Koopman semigroup. This elegant and mathematically appealing approach has good analytical (convergence) properties, but numerical experiments show that software implementation of the method has certain limitations. More precisely, with the increased dimension that guarantees theoretically better approximation and ultimate convergence, the numerical implementation may become unstable and it may even break down. The main sources of numerical difficulties are the computations of the matrix representation of the compressed Koopman operator and its logarithm. This paper addresses the subtle numerical details and proposes a new implementation algorithm that alleviates these problems.

Keywords: infinitesimal generator; Koopman operator; matrix logarithm; nonlinear system identification; preconditioning; Rayleigh quotient

Citation: Drmač, Z.; Mezić, I.; Mohr, R. Identification of Nonlinear Systems Using the Infinitesimal Generator of the Koopman Semigroup—A Numerical Implementation of the Mauroy–Goncalves Method. *Mathematics* **2021**, *9*, 2075. https://doi.org/10.3390/math9172075

Academic Editor: Rami Ahmad El-Nabulsi

Received: 30 June 2021
Accepted: 17 August 2021
Published: 27 August 2021

1. Introduction

Suppose that we have an autonomous dynamical system

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t)) \equiv \begin{pmatrix} F_1(\mathbf{x}(t)) \\ \vdots \\ F_n(\mathbf{x}(t)) \end{pmatrix}, \quad \mathbf{x}(t) \in \mathbb{R}^n, \tag{1}$$

that is accessible only through snapshots from a sequence of trajectories with different (possibly unknown) initial conditions. More precisely,

$$(\mathbf{x}_k, \mathbf{y}_k) \in \mathbb{R}^n \times \mathbb{R}^n, \quad k = 1, \ldots, K, \quad \text{where} \quad \mathbf{y}_k = \varphi^t(\mathbf{x}_k) \tag{2}$$

is the flow associated with (1). In a real application, t is a fixed time step, and it is possible that the time resolution precludes any approach based on estimating the derivatives; the dataset could also be scarce, sparsely collected from several trajectories/short bursts of the dynamics under study. The task is to identify $\mathbf{F}$ and express it analytically, using a suitably chosen class of functions. This is the essence of data-driven system identification, which is a powerful modeling technique in applied sciences and engineering—for a review see e.g., [1]. Approaches like that of SINDy [2] rely on numerical differentiation, which is a formidable task in cases of scarce and/or noisy data, and it requires special techniques such as e.g.,

total-variation regularization [3] or e.g., weak formulation [4]. With an appropriate ansatz (e.g., physics-informed) on the structure of the right hand side in (1), the identification process is computationally executed as sparse regression, see e.g., [2,5,6]. An alternative approach is machine learning techniques such as physics-informed neural networks, which proved a powerful tool for learning nonlinear partial differential equations [7].

Recently, Mauroy and Goncalves [8] proposed an elegant method for learning $\mathbf{F}$ from the data, based on the semigroup $\mathbb{U}^t f = f \circ \varphi^t$ of Koopman operators acting on a space of suitably chosen scalar observables $f \in \mathcal{F}$. In the case of the main method proposed in [8], $\mathcal{F}$ is the space $L^2(\mathbf{X})$, where $\mathbf{X} \subset \mathbb{R}^n$ is compact, forward invariant, and big enough to contain all data snapshots. The method has two main steps. First, a compression of $\mathbb{U}^t$ onto a suitable finite dimensional but rich enough subspace $\mathcal{F}_N$ of $\mathcal{F}$ is computed. On a convenient basis $\mathcal{B} = \{\wp_1, \ldots, \wp_N\}$ of $\mathcal{F}_N$, having only a limited number of snapshots (2), this compression is executed in the algebraic (discrete) least squares framework, yielding the matrix representation $U_N \in \mathbb{R}^{N \times N}$ of $\mathbb{U}^t$.

It can be shown that U_N is an approximation of the matrix exponential $U_N \approx e^{L_N t}$, or equivalently $L_N \approx (1/t) \log U_N$, where L_N is a finite dimensional compression of the infinitesimal generator $\mathbb{L}$ defined by

$$\mathbb{L}f = \lim_{t \to 0^+} \frac{U^t f - f}{t}, \qquad f \in \mathcal{D}(\mathbb{L}). \tag{3}$$

Note that the infinitesimal generator is well-defined (on its domain $\mathcal{D}(\mathbb{L}) \subset L^2(X)$) since the Koopman semigroup of operators is strongly continuous in $L^2(\mathbf{X})$ (i.e., $\lim_{t \to 0^+} \|U^t f - f\| = 0$, where $\|\cdot\|$ denotes the L^2 norm on X). We refer to [9] for more details on semigroups of operators and their properties, and to [10] for the theory and applications of the Koopman operator.

In the second step, the vector field is recovered by using the fact that $\mathbb{L}$ can also be expressed as (see e.g., [11])

$$\mathbb{L}f = \mathbf{F} \cdot \nabla f = \sum_{i=1}^{n} F_i \frac{\partial f}{\partial x_i}, \qquad f \in \mathcal{D}(\mathbb{L}). \tag{4}$$

If $F_i = \sum_k \phi_{ki} \wp_k$, then the action of $\mathbb{L}$ to the basis's vectors $\wp_k$ can be computed, using (4), by straightforward calculus, and its matrix representation will, by comparison with $(1/t) \log U_N$, reveal the coefficients ϕ_{ki}. Of course, in real applications, the quality of the computed approximations will heavily depend on the information content in the supplied data snapshots. Finer sampling (smaller time resolution) and more trajectories with different initial data are certainly desirable. If $N > K$, then a modification, called a dual method, is based on the logarithm of a $K \times K$ matrix U_K. Mauroy and Goncalves [8] proved the convergence (with probability one as $t \to 0$, $N \to \infty$, $K \to \infty$) and illustrated the performances of the method (including the dual formulation) on a series of numerical examples.

In this paper, we consider numerical aspects of the method and study its potential as a robust software tool. Our main thesis is that a seemingly simple implementation based on the off-the-shelf software components has certain limitations. For instance, with the increased dimension that guarantees theoretically better approximation and ultimate convergence, the numerical implementation, due to severe ill-conditioning, may become unstable and eventually it may break down. In other words, it can happen that with more data the potentially better approximation does not materialize in the computation. This is an undesirable chasm between analytical properties and numerical finite precision realization of the method. Hence, more numerical analysis work is needed before the method becomes mature enough to be implemented as a robust and reliable software tool.

Contributions and Organisation of the Paper

We identify, analyze and resolve the two main sources of numerical instability of the original method [8]. First, for a given time lag t, numerical computation of the matrix rep-

resentation U_N of a compression of the Koopman operator $\mathbb{U}^t$ on a (finite) N-dimensional subspace may not be computed accurately enough. Secondly, even if computed accurately, the matrix U_N may be so ill-conditioned as to preclude stable computation of the logarithm. Both issues are analyzed in detail, and we propose a new numerical algorithm that implements the method.

This material can be considered a numerical supplement to [8], and as an instructive case study for numerical software development. Moreover, the techniques developed here are used in the computational part of the recently developed framework [12]. The infinitesimal generator approach has also been successfully used for learning stochastic models from aggregated trajectory data [13,14], as well as for inverse modelling of Markov jump processes [15]. The stochastic framework is not considered in this paper, but its results apply to systems defined by stochastic differential equations as described in [8]. The stochastic setting contains many challenging problems and requires sophisticated tools, based e.g., on the Kolmogorov backward equation [8], the forward and adjoint Fokker–Planck equations [16–18], or the Koopman operator fitting [19].

The rest of paper is organized as follows. In Section 2 we set the stage and setup a numerical linear algebra framework for the analysis of subtle details related to numerical implementation of the method. This is standard, well known material and it is included for the reader's convenience and to introduce necessary notation. In particular, we give detailed description of a finite dimensional compression (in a discrete least squares sense) of the Koopman operator (Section 2.1), and we review the basic properties of the matrix logarithm (Section 2.3). Then, in Section 3, we review the details of the Mauroy–Goncalves method, including a particular choice of the monomial basis $\mathcal{B}$ and in Section 3.2 the details on the corresponding matrix representation of the generator (4). A case study example that illustrates the problem of numerical ill-conditioning is provided in Section 4. In Section 5, we propose a preconditioning step that allows for a more accurate computation of the logarithm $\log U_N$ in the case $K \geq N$. In Section 6 we consider the dual method for the case $N > K$, and formulate it as a compression of $\mathbb{U}$ onto a particular K dimensional subspace of $\mathcal{F}_N$. This formulation is then generalized in Section 7, where we introduce a new algorithm that (out of a given N) selects a prescribed number of (at most K) basis functions that are most linearly independent, as seen on the discrete set of data snapshots. The proposed algorithm, designated as *basis pruning*, can be used in both the dual and the main method, and it can be combined with the preconditioning introduced in Section 5.

2. Preliminaries: Finite Dimensional Compression of $\mathbb{U}$ and Its Logarithm

To set up the stage, in Section 2.1 we first describe matrix representation U_N of the Koopman operator compressed to a finite dimensional subspace $\mathcal{F}_N \subset \mathcal{F}$. Some details of the numerical computation of U_N are discussed in Section 2.2. In Section 2.3, we briefly review the matrix logarithm from the numerical linear algebra perspective.

2.1. Compression of $\mathbb{U}^t$ and the Anatomy of Its Matrix Representation

Given $\mathbb{U}^t : \mathcal{F} \longrightarrow \mathcal{F}$ and an N-dimensional subspace $\mathcal{F}_N \subset \mathcal{F}$, we want to compress $\mathbb{U}^t$ to $\mathcal{F}_N$ and to work with a finite dimensional approximation $\Phi_N \mathbb{U}^t_{|\mathcal{F}_N} : \mathcal{F}_N \longrightarrow \mathcal{F}_N$, where $\Phi_N : \mathcal{F} \longrightarrow \mathcal{F}_N$ is an appropriate projection. The subspace $\mathcal{F}_N$ contains functions that are simple for computation, but it is rich enough to provide good approximations for the functions in $\mathcal{F}$; it will be materialized through an ordered basis $\mathcal{B} = \{\wp_1, \ldots, \wp_N\}$. If an $f \in \mathcal{F}_N$ is expressed as $f = \sum_{i=1}^N \mathsf{f}_i \wp_i$, then the coordinates of f in the basis $\mathcal{B}$ are written as $[f]_\mathcal{B} = \left(\mathsf{f}_1, \ldots, \mathsf{f}_N\right)^T$. The ambient space $\mathcal{F}$ is equipped with the Hilbert space structure.

2.1.1. Discrete Least Squares Projection $\Phi_N : \mathcal{F} \longrightarrow \mathcal{F}_N$

Since in a data-driven setting the function is known only at the points $\mathbf{x}_k$, an operator compression will be defined using discrete least squares projection. For $g \in \mathcal{F}$, the projection $\Phi_N g = \sum_{i=1}^N \widehat{\mathrm{g}}_i \wp_i \in \mathcal{F}_N$ of g is defined so that the $\widehat{g}_i$'s solve the problem

$$\frac{1}{K}\sum_{k=1}^{K} \omega_k^2 \|(\Phi_N g)(\mathbf{x}_k) - g(\mathbf{x}_k)\|_2^2 = \frac{1}{K}\sum_{k=1}^{K} \omega_k^2 \| \sum_{i=1}^{N} \widehat{\mathrm{g}}_i \wp_i(\mathbf{x}_k) - g(\mathbf{x}_k)\|_2^2 \longrightarrow \min_{\widehat{g}_i}, \tag{5}$$

where $\omega_k \geq 0$ is a weight attached to each sample $\mathbf{x}_k$, and $\|\cdot\|_2$ is the Euclidean norm. This is a L^2 residual with respect to the empirical measure δ_K defined as the sum of the Dirac measures concentrated at the $\mathbf{x}_k$'s, $\delta_K = (1/K)\sum_{k=1}^K \delta_{\mathbf{x}_k}$. The weighting will be important in the case of noisy data; it can also be used in connection with a quadrature formula so that (5) mimics a continuous norm (defined by an integral) approximation in $\mathcal{F}$. In the unweighted case $\omega_k = 1$ for all k. The objective function (5) can be written as

$$\left\| W^{\frac{1}{2}} \left[\begin{pmatrix} \wp_1(\mathbf{x}_1) & \dots & \wp_N(\mathbf{x}_1) \\ \vdots & \dots & \vdots \\ \wp_1(\mathbf{x}_K) & \dots & \wp_N(\mathbf{x}_K) \end{pmatrix} \begin{pmatrix} \widehat{\mathrm{g}}_1 \\ \vdots \\ \widehat{\mathrm{g}}_N \end{pmatrix} - \begin{pmatrix} g(\mathbf{x}_1) \\ \vdots \\ g(\mathbf{x}_K) \end{pmatrix} \right] \right\|_2^2 \equiv \|W^{\frac{1}{2}}[O_X(\widehat{\mathrm{g}}_i)_{i=1}^N - (g(\mathbf{x}_k))_{k=1}^K]\|_2^2,$$

where $W = \mathrm{diag}(\omega_k^2)_{k=1}^K$, $(O_X)_{ij} = \wp_j(\mathbf{x}_i)$. More generally, W can be a suitable positive definite (e.g., inverse of the noise covariance) matrix. In a numerical computation the positive definite square root $W^{1/2}$ is replaced, equivalently, with the Cholesky factor: if $W = LL^T$ is the Cholesky factorization with the unique lower triangular factor L, then $\Phi = H^{1/2}L^{-T}$ must be orthogonal and when we replace $W^{1/2}$ with ΦL^T, we can omit Φ because the norm in the above objective function is invariant under orthogonal transformations.

At this point we assume that the $K \times N$ matrix O_X is of full column rank N; this requires $K \geq N$. (The rank deficient case will be discussed later.) This full column rank assumption yields the unique least squares solution

$$\begin{pmatrix} \widehat{\mathrm{g}}_1 \\ \vdots \\ \widehat{\mathrm{g}}_N \end{pmatrix} = (O_X^T W O_X)^{-1} O_X^T W \begin{pmatrix} g(\mathbf{x}_1) \\ \vdots \\ g(\mathbf{x}_K) \end{pmatrix} = [\Phi_N g]_{\mathcal{B}}, \tag{6}$$

which defines the projection Φ_N. If $g \in \mathcal{F}_N$, then $[\Phi_N g]_{\mathcal{B}} = [g]_{\mathcal{B}}$. In the unweighted case, $W = \mathbb{I}_K$, we have $[\Phi_N g]_{\mathcal{B}} = O_X^{\dagger}(g(\mathbf{x}_i))_{i=1}^K$.

2.1.2. Matrix Representation of $\Phi_N \mathbb{U}^t_{|\mathcal{F}_N} : \mathcal{F}_N \longrightarrow \mathcal{F}_N$

To describe the action of $\mathbb{U}^t$ in $\mathcal{F}_N$, we first consider how it changes the basis vectors: $\mathbb{U}^t \wp_i$ can be split as a sum of a component belonging to $\mathcal{F}_N$, and a residual,

$$(\mathbb{U}^t \wp_i)(\mathbf{x}) = \wp_i(\varphi^t(\mathbf{x})) = \sum_{j=1}^{N} \mathrm{u}_{ji} \wp_j(\mathbf{x}) + \rho_i(\mathbf{x}).$$

Given the limited information (only the snapshot pairs $(\mathbf{x}_k, \mathbf{y}_k)$), the coefficients u_{ji} are determined so that the residual ρ_i is small over the data $\mathbf{x}_k$. Since $\varphi^t(\mathbf{x}_k) = \mathbf{y}_k$, we have

$$\rho_i(\mathbf{x}_k) = \wp_i(\mathbf{y}_k) - \sum_{j=1}^{N} \mathrm{u}_{ji} \wp_j(\mathbf{x}_k), \quad k = 1, \dots, K,$$

and we can select the u_{ji}'s to minimize

$$\sum_{k=1}^{K}\omega_k^2\|\sum_{j=1}^{N}\mathsf{u}_{ji}\wp_j(\mathbf{x}_k)-\wp_i(\mathbf{y}_k)\|_2^2=\left\|W^{\frac{1}{2}}\left[\begin{pmatrix}\wp_1(\mathbf{x}_1) & \dots & \wp_N(\mathbf{x}_1)\\ \vdots & \dots & \vdots\\ \wp_1(\mathbf{x}_K) & \dots & \wp_N(\mathbf{x}_K)\end{pmatrix}\begin{pmatrix}\mathsf{u}_{1i}\\ \vdots\\ \mathsf{u}_{Ni}\end{pmatrix}-\begin{pmatrix}\wp_i(\mathbf{y}_1)\\ \vdots\\ \wp_i(\mathbf{y}_K)\end{pmatrix}\right]\right\|_2^2. \tag{7}$$

The solution is the projection (see e.g., ([20], §2.1.3, §2.7.4))

$$\begin{pmatrix}\mathsf{u}_{1i}\\ \vdots\\ \mathsf{u}_{Ni}\end{pmatrix}=(O_X^TWO_X)^{-1}O_X^TW\begin{pmatrix}\wp_i(\mathbf{y}_1)\\ \vdots\\ \wp_i(\mathbf{y}_K)\end{pmatrix}\equiv(O_X^TWO_X)^{-1}O_X^TW\begin{pmatrix}(\mathbb{U}^t\wp_i)(\mathbf{x}_1)\\ \vdots\\ (\mathbb{U}^t\wp_i)(\mathbf{x}_K)\end{pmatrix}=[\Phi_N\mathbb{U}^t\wp_i]_{\mathcal{B}}.$$

Then, for any $f=\sum_{i=1}^{N}\mathsf{f}_i\wp_i\in\mathcal{F}_N$, we have

$$\begin{aligned}g(\mathbf{x}) &= \mathbb{U}^tf(\mathbf{x})=\sum_{i=1}^{N}\mathsf{f}_i\left[\sum_{j=1}^{N}\mathsf{u}_{ji}\wp_j(\mathbf{x})+\rho_i(\mathbf{x})\right]=\sum_{j=1}^{N}\wp_j(\mathbf{x})\sum_{i=1}^{N}\mathsf{u}_{ji}\mathsf{f}_i+\sum_{i=1}^{N}\mathsf{f}_i\rho_i(\mathbf{x})\\ &= \sum_{j=1}^{N}\wp_j(\mathbf{x})\mathsf{g}_j+\sum_{i=1}^{N}\mathsf{f}_i\rho_i(\mathbf{x}),\ \text{ where }\ \mathsf{g}_j=\sum_{i=1}^{N}\mathsf{u}_{ji}\mathsf{f}_i,\ \text{ i.e., }\ \begin{pmatrix}\mathsf{g}_1\\ \vdots\\ \mathsf{g}_N\end{pmatrix}=U_N\begin{pmatrix}\mathsf{f}_1\\ \vdots\\ \mathsf{f}_N\end{pmatrix}.\end{aligned}$$

Hence, $\Phi_N\mathbb{U}^t_{|\mathcal{F}_N}:\mathcal{F}_N\longrightarrow\mathcal{F}_N$ is on the basis $\mathcal{B}$ represented by the matrix

$$[\Phi_N\mathbb{U}^t_{|\mathcal{F}_N}]_{\mathcal{B}}=O_X^{\dagger}O_Y\equiv U_N, \tag{8}$$

$$\Phi_N\mathbb{U}^t_{|\mathcal{F}_N}\begin{pmatrix}\wp_1(x) & \dots & \wp_N(x)\end{pmatrix}\begin{pmatrix}\mathsf{f}_1\\ \vdots\\ \mathsf{f}_N\end{pmatrix}=\begin{pmatrix}\wp_1(x) & \dots & \wp_N(x)\end{pmatrix}(U_N\begin{pmatrix}\mathsf{f}_1\\ \vdots\\ \mathsf{f}_N\end{pmatrix}), \tag{9}$$

where we use $W=\mathbb{I}_K$ for the sake of technical simplicity, and $O_X^{\dagger}$ is the Moore–Penrose generalized inverse. If $W\neq\mathbb{I}_K$, then $[\Phi_N\mathbb{U}^t_{|\mathcal{F}_N}]_{\mathcal{B}}=(O_X^TWO_X)^{-1}O_X^TWO_Y\equiv U_N$.

2.1.3. When Is U_N Nonsingular?

If both O_X and O_Y are of full column rank N, then the rank of U_N depends on the canonical angles between the ranges of O_X and O_Y. Indeed, if $O_X=Q_XR_X$, $O_Y=Q_YR_Y$ are the thin QR factorizations, then $U_N=R_X^{-1}Q_X^TQ_YR_Y$, where the singular values of $Q_X^TQ_Y$ can be written in terms of the canonical angles $0\leq\theta_1\leq\cdots\leq\theta_N\leq\pi/2$ as $\cos\theta_1\geq\cdots\geq\cos\theta_N$. Hence, in the case of full column rank of O_X and O_Y, nonsingularity of U_N is equivalent to $\cos\theta_N>0$. To visualize this condition, U_N will be nonsigular if none of the ranges of O_X and O_Y contain a direction that is orthogonal to the other one. If the basis function and the flow map are well behaved and the sampling time is reasonable, it is reasonable to expect $\theta_N<\pi/2$.

2.1.4. Relations with the DMD

In the DMD framework, $\wp_1,\dots,\wp_N$ are the scalar components of a $N\times1$ vector valued observable evaluated at the sequence of snapshots $\mathbf{x}_1,\dots,\mathbf{x}_K$, and it is customary to arrange them in a $N\times K$ matrix, i.e., O_X^T. Similarly, the values of the observables at the $\mathbf{y}_k$'s are in the matrix O_Y^T. The Exact DMD matrix is then $A=O_Y^T(O_X^T)^{\dagger}=U_N^T$. For more details on this connection we refer to [21,22].

2.2. On the Numerical Solution of $\|O_XU_N-O_Y\|_F\to min$

In general, the least squares projection $O_XU_N=O_XO_X^{\dagger}O_Y$ is uniquely determined, but, unless O_X is of full column rank N, the solution U_N of the problem $\|O_XU_N-O_Y\|_F\to\min$ is not unique—each of its columns is from a linear manifold determined by the null-space of O_X and we can vary them independently by adding to them arbitrary vectors from the null space of O_X. Furthermore, even when O_X is of rank N but ill-conditioned, a

typical numerical least squares solver will detect the ill-conditioning by revealing that the matrix is close to singular matrices and it will treat it as numerically rank deficient. Then, the computed solution U_N becomes non-unique, the concrete result depends on the least squares solution algorithm, and it may be rank deficient. This calls for caution when computing U_N and $\log U_N$ numerically.

We discuss this in Section 2.2.1, and illustrate the problems in practice using a case study example in Section 4.

2.2.1. Least Squares Solution in Case of Numerical Rank Deficiency

If O_X is not of full column rank N, then $O_X^\dagger O_Y$ is one of infinitely many solutions to the least squares problem for the matrix U_N that is used to represent the operator compression. Furthermore, since it is necessarily singular, its logarithm does not exist and identifying a matrix approximation of the infinitesimal generator is not feasible. This is certainly the case when $K < N$ (recall that in this case a dual form of the method is used; see Section 6), and in the case $K \geq N$, considered here, the matrix O_X can be numerically rank deficient and the software solution will return a solution that depends on a particular algorithm for solving the least squares problem.

Let $O_X = \Phi\Sigma\Psi^T$ be the SVD of O_X and let r be the rank of O_X such that $r < \min(K, N)$. Let $\Sigma_r = \mathrm{diag}(\sigma_i)_{i=1}^r$, where $\sigma_1 \geq \cdots \geq \sigma_r > 0$ are the nonzero singular values. Partition the singular vector matrices as $\Phi = (\Phi_r, \Phi_0)$, $\Psi = (\Psi_r, \Psi_0)$, where Φ_r and Ψ_r have r columns, so that $O_X = \Phi_r\Sigma_r\Psi_r^T$, $O_X^\dagger = \Psi_r\Sigma_r^{-1}\Phi_r^T$. Recall that the columns of the $N \times (N - r)$ matrix Ψ_0 are an orthonormal basis for the null space of O_X.

In the rank deficient case, the solution set for the least squares problem $\|O_X U_N - O_Y\|_F \to \min_{U_N}$ is a linear manifold—in this case of the form

$$\mathcal{N} = \{O_X^\dagger O_Y + \Psi_0 \Xi, \;\; \Xi \in \mathbb{R}^{(N-r)\times N}\}. \tag{10}$$

Clearly $O_X(O_X^\dagger O_Y + \Psi_0\Xi) = O_X O_X^\dagger O_Y = \Phi_r\Phi_r^T O_Y$.

The particular choice $U_N = O_X^\dagger O_Y = \Psi_r\Sigma_r^{-1}\Phi_r^T O_Y$, as (8), is distinguished by being of minimal Frobenius norm, because $\|O_X^\dagger O_Y + \Psi_0\Xi\|_F^2 = \|O_X^\dagger O_Y\|_F^2 + \|\Xi\|_F^2$. The minimality of the Euclidean norm is one criterion to pinpoint the unique solution and uniquely define the pseudo-inverse. Such minimal norm solution, which is of rank at most r, may indeed be desirable in some applications, but here it is useless if $r < N$, because we cannot proceed with computing $\log U_N$.

It remains an interesting question whether in the rank deficient cases we can explore the solution set (10) and with some additional constraints define a meaningful matrix representation. In general, a matrix representation should as much as possible reproduce the behaviour of the operator in that subspace. For example, if O_X is of full column rank N (i.e., we have sufficiently large K and well selected observables) and the first basis function is constant, $\wp_1(x) \equiv 1$, then the first columns of O_X and O_Y are $e = (1, \ldots, 1)^T$ and simple argument implies that in (8) the first column of U_N is the first canonical vector $e_1 = (1, \ldots, 0)^T$ and

$$[\Phi_N \mathbb{U}^t_{|\mathcal{F}_N}]_{\mathcal{B}}[\wp_1]_{\mathcal{B}} \equiv U_N e_1 = e_1 = 1 \cdot [\wp_1]_{\mathcal{B}}, \tag{11}$$

which corresponds to $\mathbb{U}^t\wp_1 = 1 \cdot \wp_1$. If $r < N$ (so that O_X has a nontrivial null space), then we do not expect $U_N e_1 = e_1$, but we can show that $\mathcal{N}$ contains a matrix $\widehat{U}_N$ such that $\widehat{U}_N e_1 = e_1$.

Proposition 1. *If $\wp_1(x) \equiv 1$, then we can choose an $\widehat{U}_N \in \mathcal{N}$ such that $\widehat{U}_N e_1 = e_1$.*

Proof. To satisfy $(O_X^\dagger O_Y + \Psi_0\Xi)e_1 = e_1$, Ξ must be such that $\Psi_0\Xi e_1 = e_1 - U_N e_1$. Note that $e_1 - U_N e_1$ is in the null-space of O_X:

$$O_X(e_1 - O_X^\dagger O_Y e_1) = O_X e_1 - O_X O_X^\dagger O_Y e_1 = e - O_X O_X^\dagger e = e - e = \mathbf{0}.$$

Hence, $\Xi(:,1) = \Psi_0^T(e_1 - U_N e_1) = \Psi_0^T e_1$, and $\Xi(:,2:n)$ can be set e.g., to zero to obtain Ξ of minimal Frobenius norm. Here also $\Xi = \Psi_0^T$ is an interesting choice, but we omit the details because, in this paper, following [8], we treat the rank deficiency by a form of dual method as described in Sections 6 and 7. □

Remark 1. *The global non-uniqueness in form of the additive term $\Psi_0\Xi$ is non-essential when instead of U_N we use its compression in a certain subspace. For instance, if $r = K < N$ the Rayleigh quotient of U_N with respect to the range of O_X^T remains unchanged, see Section 6.*

In practice, the least squares solution using the SVD and the formula for the pseudo-inverse are often replaced by a more efficient method based on the column pivoted (rank revealing) QR factorization. For instance, the factorization [23] uses a greedy matrix volume maximizing scheme to determine a permutation Π such that in the QR factorization

$$O_X\Pi = QR, \quad Q \in \mathbb{R}^{K\times N}, \quad Q^TQ = \mathbb{I}_N, \quad R \in \mathbb{R}^{N\times N} \text{ upper triangular}, \tag{12}$$

the triangular factor R has a strong diagonal dominance of the form

$$|R_{ii}| \geq \sqrt{\sum_{k=i}^{j} |R_{kj}|^2}, \quad 1 \leq i \leq j \leq N. \tag{13}$$

If O_X is of rank $r < N$, then $R(1:r,1:r)$ is nonsingular and $R(r+1:N, r+1:N) = \mathbf{0}$, and the least squares solution of $\|O_X z - b\|_2 \to \min_z$ is computed as

$$z = \Pi \begin{pmatrix} R(1:r,1:r)^{-1} Q(1:K,1:r)^T b \\ \mathbf{0}_{N-r,1} \end{pmatrix}.$$

In a nearly rank deficient (i.e., ill-conditioned) case, the index r is determined so that setting $R(r+1:N, r+1:N)$ to zero introduces error below a tolerance that is of order of machine precision. This is the solution returned by the backslash operator in Matlab. Hence, in the numerical rank deficient cases, the solution will have at least $N - r$ zero components, and it will be in general different from the solution obtained using the pseudo-inverse. Here too, some additional constraints can be satisfied under some conditions, as shown in the following:

Proposition 2. *Let $O_X\Pi = QR$ be the column pivoted QR factorization in the first step of the backslash operator when solving the least squares problem $\|O_X U_N - O_Y\|_F \to \min$, where $r = \operatorname{rank}(O_X) < N$ and $O_X(:,1) = O_Y(:,1)$. If the first column of O_X is selected among the leading k pivotal columns in the permutation matrix Π, then $U_N(:,1) = e_1$.*

2.3. Computing the Logarithm $\log O_X^\dagger O_Y$

The key element of the Mauroy–Goncalves method is that $U_N \approx e^{L_N t}$, where L_N is a matrix representation of a compression of the infinitesimal generator (3), that is computed as $L_N = (1/t)\log U_N$. Hence, to use the matrix L_N as an ingredient of the identification method, it is necessary that $U_N = O_X^\dagger O_Y$ is nonsingular; otherwise $\log U_N$ does not exist. Furthermore, to achieve the primary value of the logarithm (as a primary matrix function, i.e., the same branch of the logarithm used in all Jordan blocks), the matrix must not have any real negative eigenvalues. Only under those conditions we can obtain a real (assuming real U_N) logarithm as the primary function.

For the reader's convenience, we summarize the key properties of the matrix logarithm and refer to [24] for proofs and more details.

Theorem 1. *Let A be real nonsingular matrix. Then A has real logarithm if and only if A has an even number of Jordan blocks of each size for every negative eigenvalue.*

Theorem 2. *Let A be real nonsingular matrix. Then A has a unique real logarithm if and only if all eigenvalues of A are positive real and no eigenvalue has more than one Jordan block in the Jordan normal form of A.*

Theorem 3. *Suppose that the $n \times n$ complex A has no eigenvalue on $(-\infty, 0]$. Then a unique logarithm of A can be defined with eigenvalues in the strip $\{z \in \mathbb{C} : -\pi < \Im(z) < \pi\}$. It is called the principal logarithm and denoted by $\log A$. If A is real, then its principal logarithm is real as well.*

In an application, the matrix $U_N = O_X^{\dagger} O_Y$ may be difficult to compute accurately and it may be so severely ill-conditioned that in the finite precision computations it could appear numerically rank deficient (recall the discussion in Section 2.2). Thus, from the numerical point of view, the most critical part of the method is computing U_N and its logarithm. For a detailed analysis of numerical methods for computing the matrix logarithm we refer the reader to ([25], Chapter 11).

3. Identification Method

To introduce the new numerical implementation, we need more detailed description of the method [8] and its concrete realization. In Section 3.1, we select the subspace $\mathcal{F}_N$ as the span of the monomials in n variables. The key idea of the method, to explore the connection $\mathbb{U}^t = e^{\mathbb{L}t}$ in the framework of finite dimensional compressions of $\mathbb{U}^t$ and $\mathbb{L}$, is reviewed in detail in Section 3.2. This is a rather challenging step, both in terms of theoretical justification of the approximation (including convergence when $N, K \to \infty$, $t \to 0$) and the numerical realization. As an interesting detail, we point out in Section 3.3.1 that a structure aware reconstruction/identification can be naturally formulated for e.g., the important class of quadratic systems. This generates an interesting structured least squares problem.

3.1. The Choice of the Basis $\mathcal{B}$—Monomials

For a concrete application, the choice of a suitable basis depends on the assumption on the structure of $\mathbf{F}$. The monomial basis is convenient if $\mathbf{F}$ is a polynomial field, or if it can be well approximated by polynomials. We assume that

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} \sum_{k=1}^{N_F} \phi_{k1} x_1^{s_1^{(k)}} x_2^{s_2^{(k)}} \cdots x_n^{s_n^{(k)}} \\ \vdots \\ \sum_{k=1}^{N_F} \phi_{kn} x_1^{s_1^{(k)}} x_2^{s_2^{(k)}} \cdots x_n^{s_n^{(k)}} \end{pmatrix} = \begin{pmatrix} F_1(\mathbf{x}) \\ \vdots \\ F_n(\mathbf{x}) \end{pmatrix}, \quad F_j(\mathbf{x}) = \sum_{k=1}^{N_F} \phi_{kj} \mathbf{x}^{\mathbf{s}^{(k)}}, \tag{14}$$

where $\mathbf{x}^{\mathbf{s}^{(k)}} = x_1^{s_1^{(k)}} x_2^{s_2^{(k)}} \cdots x_n^{s_n^{(k)}}$ are monomials written in multi-index notation and have a total degree of at most m_F. In that case, $\mathcal{F}_N$ is chosen as the space of polynomials up to some total degree m, $m \geq m_F$ i.e., $\mathcal{F}_N = \mathrm{span}(\mathcal{B})$, where

$$\mathcal{B} = \{x_1^{s_1} x_2^{s_2} \cdots x_n^{s_n} \ : \ s_i \in \mathbb{N}_0, \ s_1 + s_2 + \ldots + s_n \leq m\}, \quad N = \binom{n+m}{n} \geq N_F. \tag{15}$$

To facilitate automatic and relatively simple matrix representation of linear operators acting on $\mathcal{F}_N$, we choose graded lexicographic ordering (*grlex*) of $\mathcal{B}$, which is one of the standard procedures in the multivariate polynomial framework. *Grlex* orders the basis so that it first divides the monomials in groups with same total degree; the groups are listed with increasing total degree and inside each group the monomials are ordered so that their exponents $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{N}_0^n$ are lexicographically ordered. For example, if $n = 3$,

$m = 2$, we have the order as follows (read the tables in (16) column-wise; each column corresponds to the monomials of the same total degree, ordered lexicographically):

$$\left[\begin{array}{c|c|c} 1(000) & x_3(001) & x_3^2(002) \\ & x_2(010) & x_2x_3(011) \\ & x_1(100) & x_2^2(020) \\ & & x_1x_3(101) \\ & & x_1x_2(110) \\ & & x_1^2(200) \end{array}\right] \quad (s_1,\ldots,s_n) \rightleftarrows k \quad \left[\begin{array}{c|c|c} 1 & 2 & 5 \\ & 3 & 6 \\ & 4 & 7 \\ & & 8 \\ & & 9 \\ & & 10 \end{array}\right]. \tag{16}$$

If we want to emphasize that $\mathbf{s} = (s_1,\ldots,s_n)$ is at the kth position in this ordering, we write $\mathbf{s}^{(k)} = (s_1^{(k)},\ldots,s_n^{(k)})$, and the corresponding monomial is written as $\mathbf{x}^{\mathbf{s}^{(k)}} \equiv x_1^{s_1^{(k)}} x_2^{s_2^{(k)}} \cdots x_n^{s_n^{(k)}}$. An advantage of *grlex* in our setting is that it allows simple extraction of the operator compression to a subspace spanned by monomials of lower total degree.

It should be noted that the dimension N grows extremely fast with increased n and m, which is the source of many computational difficulties, in particular when combined with the requirement $K \geq N$ which is a necessary condition for the non-singularity of U_N. (This difficulty is alleviated by the dual method.)

Even though polynomial basis is not always the best choice, it serves well for the purposes of this paper because, with increased total degree, it generates highly ill-conditioned numerical examples that are good stress test cases for development, testing and analysis of numerical implementation.

3.2. Compression of $\mathbb{L}$ in the Monomial Basis

Consider now the action of $\mathbb{L}$ to the vectors of the monomial basis $\mathcal{B}$. It is a straightforward and technically tedious task to identify the columns of the corresponding matrix $[L_N]_{\mathcal{B}}$, whose approximation can also be computed as $\frac{1}{t}\log U_N$. Since we are interested only in the coefficients ϕ_{kj} in (14), it is enough to compute only some selected columns of $[L_N]_{\mathcal{B}}$.

Let ℓ be the index of x_j in the *grlex* ordering, i.e., $\wp_\ell(\mathbf{x}) = x_j$; $\ell = n+2-j$ (see the scheme (16)). Then the application of (4) to $\wp_\ell$ reads

$$(\mathbb{L}\wp_\ell)(\mathbf{x}) = (\mathbf{F}\cdot\nabla\wp_\ell)(\mathbf{x}) = \sum_{i=1}^{n} F_i(\mathbf{x})\frac{\partial}{\partial x_i}\wp_\ell(\mathbf{x}) = F_j(\mathbf{x}) \equiv F_{n+2-\ell}(\mathbf{x}), \text{ i.e., } \mathbb{L}\wp_\ell = F_{n+2-\ell}.$$

If $L_N = \Phi_N\mathbb{L}_{|\mathcal{F}_N}$, then also $L_N\wp_\ell = F_j$ (because of the assumption (14) $F_j \in \mathcal{F}_N$). Hence, in the basis $\mathcal{B}$ we have

$$[L_N]_{\mathcal{B}}(:,\ell) = [\Phi_N\mathbb{L}\wp_\ell]_{\mathcal{B}} = [F_j]_{\mathcal{B}} = \begin{pmatrix} \phi_{1j} \\ \phi_{2j} \\ \vdots \\ \phi_{N_F j} \\ \mathbf{0}_{N-N_F} \end{pmatrix}, \text{ where } j = n+2-\ell, \ \ell = 2,\ldots,n+1. \tag{17}$$

In other words, the coordinates of F_j are encoded in $[L_N]_{\mathcal{B}}(:,n+2-j)$. Finally the entries of $[L_N]_{\mathcal{B}}$ can be obtained with the compression U_N computed from data. Indeed, it is shown in [8] that

$$\lim_{t\to 0^+} \|\Phi_N\mathbb{L}f - (1/t)\log\Phi_N\mathbb{U}^t f\| = 0, \qquad f \in \mathcal{F}_N.$$

Hence, provided that $\mathcal{B}$ contains independent basis functions, we have

$$[L_N]_{\mathcal{B}} = \lim_{t\to 0^+}\frac{1}{t}[\log\Phi_N\mathbb{U}^t_{|\mathcal{F}_N}]_{\mathcal{B}} = \lim_{t\to 0^+}\frac{1}{t}\log[\Phi_N\mathbb{U}^t_{|\mathcal{F}_N}]_{\mathcal{B}},$$

and it follows that, for t small enough,

$$[L_N]_{\mathcal{B}} \approx \frac{1}{t}\log U_N \equiv \frac{1}{t}\log O_X^{\dagger} O_Y. \tag{18}$$

For each F_j, its coefficients in the expansion (14) are simply read off from the corresponding column of $[L_N]_{\mathcal{B}}$. Alternatively, we can identify additional columns and determine the coefficients by solving a least squares problem. For more details we refer to [8].

3.3. Imposing the Structure in the Reconstruction of F

Using (17), the values of F_j at the $\mathbf{x}_k$'s can be approximated using the values

$$\begin{pmatrix} \widetilde{F_j(\mathbf{x}_1)} \\ \vdots \\ \widetilde{F_j(\mathbf{x}_K)} \end{pmatrix} = \begin{pmatrix} \wp_1(\mathbf{x}_1) & \dots & \wp_N(\mathbf{x}_1) \\ \vdots & & \vdots \\ \wp_1(\mathbf{x}_K) & \dots & \wp_N(\mathbf{x}_K) \end{pmatrix} \begin{pmatrix} \phi_{1j} \\ \phi_{2j} \\ \vdots \\ \phi_{N_F j} \\ \mathbf{0}_{N-N_F} \end{pmatrix} = O_X \cdot [L_N]_{\mathcal{B}}(:, n+2-j), \quad j = 1, \dots, n. \tag{19}$$

These can be used for expressing F_j using another suitable dictionary of functions q_{ij} (e.g., rational) by decoupled least squares fitting for the functions F_j: with an ansatz $F_j = \sum_{i=1}^{N_{F_j}} \varphi_{ij} q_{ij}$, the coefficients φ_{ij} are determined to minimize

$$\left\| \begin{pmatrix} \widetilde{F_j(\mathbf{x}_1)} \\ \vdots \\ \widetilde{F_j(\mathbf{x}_K)} \end{pmatrix} - \begin{pmatrix} q_{1j}(\mathbf{x}_1) & q_{2j}(\mathbf{x}_1) & \dots & q_{N_{F_j} j}(\mathbf{x}_1) \\ \vdots & \vdots & \dots & \vdots \\ q_{1j}(\mathbf{x}_K) & q_{2j}(\mathbf{x}_K) & \dots & q_{N_{F_j} j}(\mathbf{x}_K) \end{pmatrix} \begin{pmatrix} \varphi_{1j} \\ \vdots \\ \varphi_{N_{F_j} j} \end{pmatrix} \right\|, \tag{20}$$

where $\|\cdot\|$ is an appropriate (possibly weighted) norm.

Remark 2. *Note that (20) is slightly more general than in [8]—we can use separate dictionaries for each coordinate function F_j, which allows fitting variables of different (physical, thus mathematical) nature separately, with most appropriate classes of basis functions.*

In [8], it is recommended to solve the regression problem (20) with a sparsity promoting method, thus revealing the underlying structure. In many cases, the sparsity is known to be specially structured, and we can exploit that information. We illustrate our proposed approach using the quadratic systems.

3.3.1. Quadratic Systems

Suppose we know that the system under study is quadratic, i.e., $F(\mathbf{x}) = \mathsf{A}\mathbf{x} + \mathsf{G}(\mathbf{x} \otimes \mathbf{x})$. Quadratic systems are important class of dynamical systems, with many applications and interesting theoretical properties, see e.g., [26].

With the approximate field values $\widetilde{F(\mathbf{x}_1)}, \dots, \widetilde{F(\mathbf{x}_K)}$, we can seek $\mathsf{A} \in \mathbb{R}^{n \times n}$ and $\mathsf{G} \in \mathbb{R}^{n \times n^2}$ to achieve

$$\begin{pmatrix} \widetilde{F(\mathbf{x}_1)} & \dots & \widetilde{F(\mathbf{x}_K)} \end{pmatrix} \approx \mathsf{A}\begin{pmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_K \end{pmatrix} + \mathsf{G}\begin{pmatrix} \mathbf{x}_1 \otimes \mathbf{x}_1 & \dots & \mathbf{x}_K \otimes \mathbf{x}_K \end{pmatrix}. \tag{21}$$

If we set $\widetilde{F} = \begin{pmatrix} \widetilde{F(\mathbf{x}_1)} & \dots & \widetilde{F(\mathbf{x}_K)} \end{pmatrix}$, $X = \begin{pmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_K \end{pmatrix}$, then the identification of the coefficient matrices A and G reduces to solve the matrix least squares problem

$$\left\| \begin{pmatrix} X^T & (X \odot X)^T \end{pmatrix} \begin{pmatrix} \mathsf{A}^T \\ \mathsf{G}^T \end{pmatrix} - \widetilde{F}^T \right\|_F \longrightarrow \min_{\mathsf{A},\mathsf{G}}, \tag{22}$$

where $X \odot X = \begin{pmatrix} \mathbf{x}_1 \otimes \mathbf{x}_1 & \dots & \mathbf{x}_K \otimes \mathbf{x}_K \end{pmatrix} \in \mathbb{R}^{n^2 \times K}$ is the Khatri–Rao product. Here too, one can add a sparsity promoting regularization, with the implicitly defined underlying quadratic structure.

4. Numerical Implementation—A Case Study Analysis

When it comes to turning a numerical algorithm into software, it is often straightforward to write a few lines in Matlab, Python, Octave or some other software package and have a running implementation of a sophisticated procedure obtained by composition of building blocks (subroutines). However, one should keep in mind that the final numerical computation is in finite precision (machine) arithmetic, and that in some cases development of robust numerical software requires a more careful approach. In this section, we use a case study example that reveals difficulties from the numerical software point of view, and that motivates modifications to alleviate them.

4.1. An Example: Lorenz System

A good way to test robustness of a numerical algorithm is to push it to its limits. In this case, we choose a difficult test case and let the dimensions of the data matrices grow by increasing the total degree m of the polynomial basis (and thus the dimension N) and matching that with increased K so that $K > N$. The main goal is to provide a case study example.

Example 1. *Consider the Lorenz system*

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} -10 & 10 & 0 \\ 28 & -1 & 0 \\ 0 & 0 & -8/3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0 \\ -x_1 x_3 \\ x_1 x_2 \end{pmatrix}. \tag{23}$$

The exact coefficients, ordered to match the grlex *ordering of the monomial basis are*

	1	x_3	x_2	x_1	x_3^2	x_2x_3	x_2^2	x_1x_3	x_1x_2	x_1^2
F_1:	0	0	$\mathbf{1.0000\times10^1}$	$\mathbf{-1.0000\times10^1}$	0	0	0	0	0	0
F_2:	0	0	$\mathbf{-1.0000\times10^0}$	$\mathbf{2.8000\times10^1}$	0	0	0	$\mathbf{-1.0000\times10^0}$	0	0
F_3:	0	$\mathbf{-2.6667\times10^0}$	0	0	0	0	0	0	$\mathbf{1.0000\times10^0}$	0

To collect data, we ran simulations with 55 *random initial conditions and from each trajectory we randomly (independently) selected* 55 *points, giving a total of* $K = 3025$ *pairs* $(\mathbf{x}_k, \mathbf{y}_k)$. *The simulations were performed in Matlab, using the* `ode45()` *solver in the time interval* $[0, 0.2]$ *with the time step* $\delta t = 10^{-3}$. *In the key Formula (18), we computed the logarithm in Matlab in two ways, as* `logm(pinv`$(O_X) * O_Y$`)` *and as* `logm`$(O_X \backslash O_Y)$ *and obtained nearly the same matrix. Of course, using the pseudoinverse explicitly is not recommended. We will use it in this section for illustrative purposes; recall the discussion in Section 2.2. The computed approximations of the coefficients of (23), with* $m = 3$, $N = 20$ *and* $m_F = 2$, *are*

	1	x_3	x_2	x_1	x_3^2	x_2x_3	x_2^2	x_1x_3	x_1x_2	x_1^2
F_1:	3.2×10^{-5}	-2.5×10^{-6}	$\mathbf{1.0000\times10^1}$	$\mathbf{-1.0000\times10^1}$	-7.5×10^{-7}	-5.2×10^{-6}	1.5×10^{-7}	7.4×10^{-6}	-1.6×10^{-6}	8.0×10^{-7}
F_2:	-3.4×10^{-5}	-8.7×10^{-6}	$\mathbf{-1.0000\times10^0}$	$\mathbf{2.8000\times10^1}$	4.7×10^{-6}	1.3×10^{-5}	8.7×10^{-9}	$\mathbf{-1.0001\times10^0}$	-6.1×10^{-6}	6.1×10^{-6}
F_3:	2.8×10^{-4}	$\mathbf{-2.6667\times10^0}$	-5.5×10^{-6}	-6.9×10^{-6}	-8.3×10^{-6}	2.9×10^{-6}	1.6×10^{-8}	-1.0×10^{-5}	$\mathbf{1.0000\times10^0}$	5.7×10^{-6}

The nonzero coefficients are matched to five digits of accuracy, and the remaining coefficients are below $O(10^{-5})$. *Nearly the same result is obtained if the samples from all trajectories have the same time stamps. Given the difficulty of the Lorenz example and the fact that the results are obtained by a low dimensional approximation of a nontrivial (numerically simulated) dynamics, the results are good. Analytical properties of the method indicate that increasing dimensions provides increased accuracy, ultimately yielding convergence.*

Example 2. *Next, we illustrate the reconstruction of the function* $\mathbf{F}$. *We simulated the system in the interval* $[0, 4]$ *with* $\delta t = 0.01$, *and used the monomials with the total degree up to* $m = 3$. *In the discrete time grid, we randomly selected* 9 *positions at which we took three consecutive values*

from each trajectory; in the second experiment 27 *randomly selected values of* $\mathbf{x}_k$ *are taken from each trajectory. The total number of trajectories (with random initial conditions) was set to* 150.

$\mathbf{F}$ *is approximated using (19). For each* $\mathbf{x}_k$*, we compute the approximation error as*

$$\epsilon_k = \max_{i=1,2,3} \frac{|\widetilde{F_i(\mathbf{x}_k)} - F_i(\mathbf{x}_k)|}{\|\mathbf{F}(\mathbf{x}_k)\|_\infty}. \tag{24}$$

The sampling points and the values of ϵ_k *are shown in Figure 1.*

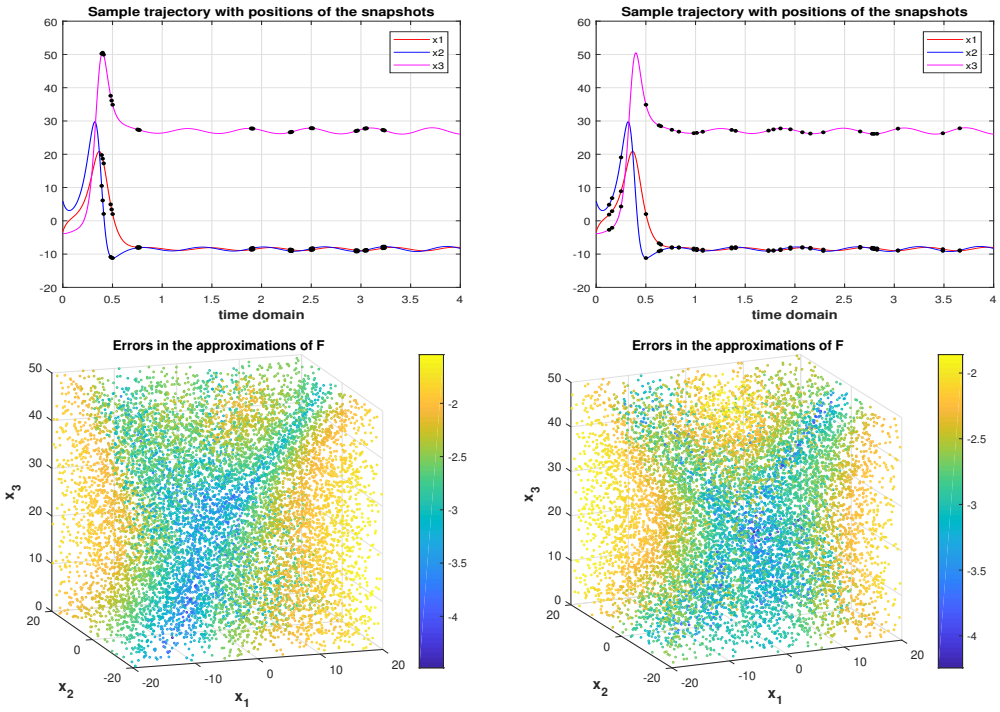

Figure 1. (Example 2, $m = 3$.) **First row**: samples of $\mathbf{x}_k$ from the first three trajectories, using two different sampling schemes. **Second row**: the corresponding values of $\log_{10} \epsilon_k$ defined in (24) for 12,000 randomly selected points in the box $[-20, 20] \times [-20, 20] \times [0, 50]$.

Example 3. *Now we use the data snapshots from Example 1, and increase the total degree to* $m = 9$*, thus increasing* N *from* $N = 20$ *to* $N = 220$*. Recall that* $K = 3025$*. Surprisingly, the computed coefficients are all complex, and are completely off; their absolute values are (the euclidean norms of the real and the imaginary parts of the vector of the computed coefficients are* $O(10^6)$*.*

	1	x_3	x_2	x_1	x_3^2	x_2x_3	x_2^2	x_1x_3	x_1x_2	x_1^2
F_1:	4.6×10^3	1.6×10^6	$\mathbf{3.6\times10^5}$	$\mathbf{3.0\times10^4}$	1.5×10^4	1.2×10^4	4.3×10^3	1.0×10^4	9.1×10^3	1.0×10^4
F_2:	3.0×10^4	1.2×10^6	$\mathbf{2.8\times10^5}$	$\mathbf{1.0\times10^4}$	9.6×10^3	1.2×10^4	7.6×10^3	$\mathbf{2.1\times10^4}$	9.5×10^3	5.5×10^3
F_3:	6.8×10^3	$\mathbf{2.4\times10^5}$	5.5×10^4	2.2×10^3	1.9×10^3	2.6×10^3	1.6×10^3	4.5×10^3	$\mathbf{2.0\times10^3}$	1.1×10^3

The approximation of $\mathbf{F}$ *is also bad, see Figure 2. Increasing the number of trajectories and samples per trajectory to obtain* $K = 24{,}025$ *did not bring any improvement.*

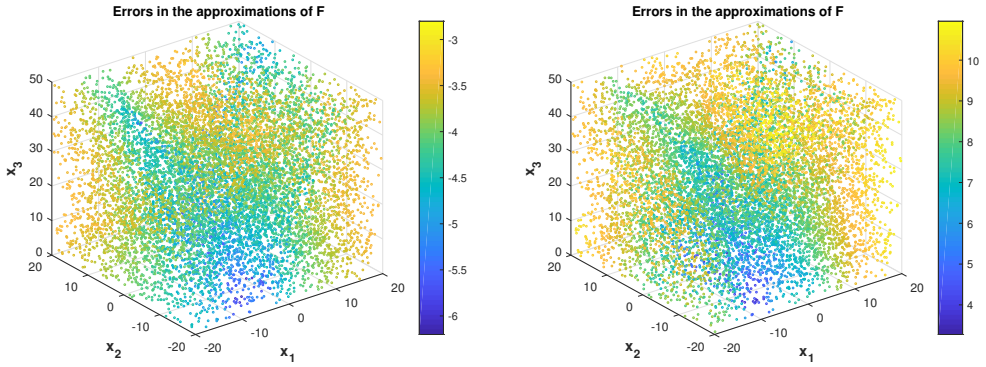

Figure 2. (Example 3, $m = 9$.) The values of $\log_{10} \epsilon_k$ defined in (24) for 12,000 randomly selected points in the box $[-20, 20] \times [-20, 20] \times [0, 50]$. **Left panel**: with $m = 3$ as in Example 1. **Right panel**: with $m = 9$.

Although increasing N, with correspondingly large K, should be the step toward better approximation, the numerical implementation breaks down. In situation like this one, it is important to find out and understand the sources of the problem.

4.2. What Went Wrong

It is clear that in this computation the most sensitive part is computation of the matrix logarithm, and that this is the most likely source of problems. Indeed, in the last run in Example 3 a warning was issued by the `logm()` function:

```
Warning:  the principal matrix logarithm is not defined for A with nonpositive
real eigenvalues.  A non-principal matrix logarithm is returned.
```

A closer inspection of the eigenvalues of U_N, shown in Figure 3 confirms that U_N indeed has problematic (real negative) eigenvalues.

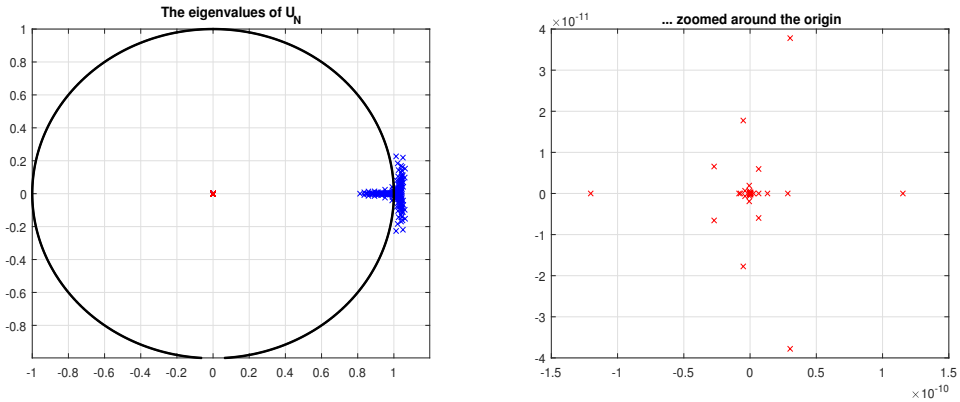

Figure 3. (Example 3, $m = 9$.) **Left panel**: the (computed) eigenvalues of the matrix representation of the computed compression U_N of $\mathbb{U}^t$. The red cross at the origin indicates a cluster of eigenvalues. **Right panel**: zoomed neighborhood of the origin, showing many (in this case 44) absolutely small eigenvalues, quite a few of which are negative real. The matrix U_N is computed as $(\texttt{pinv}(O_X) * O_Y)$. If it is computed as $(O_X \backslash O_Y)$, instead of the cluster around zero, the eigenvalue zero appears with multiplicity 45.

Although Figures 3 and 4 show numerically computed eigenvalues, by a backward stability analysis argument one can argue that U_N is certainly close to matrices with eigenvalues that preclude computing the logarithm in finite precision arithmetic. Hence, computation of the logarithm must be ill-conditioned as the ill-conditioning is essentially measured as the inverse distance to singularity [27].

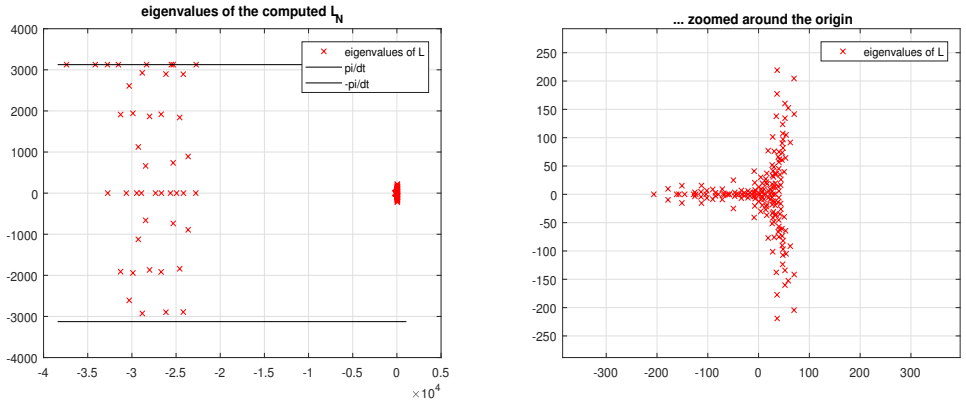

Figure 4. (Example 3, $m = 9$.) The (computed) eigenvalues of the matrix L_N. Note that some of them are at the boundary of the strip $\{z \in \mathbb{C} \ : \ -\pi/\delta t < \Im(z) < \pi/\delta t\}$, i.e., the eigenvalues of $\log U_N$ are at the boundary of $\{z \in \mathbb{C} \ : \ -\pi < \Im(z) < \pi\}$, cf. Theorem 3. The right panel shows the distribution of the eigenvalues closer to the origin. Compare with Figure 3.

Another look at U_N reveals that its first column is not what it should be. In this example we use monomials and the first basis vector is the constant. Since $\mathbb{U}^t \wp_1 = 1 \cdot \wp_1$, we expect $[\Phi_N \mathbb{U}^t_{|\mathcal{F}_N}]_{\mathcal{B}} [\wp_1]_{\mathcal{B}} \equiv U_N e_1 = e_1 = 1 \cdot [\wp_1]_{\mathcal{B}}$, which clearly follows from the solution of the least squares problem (7), if the coefficient matrix is of full column rank. On the other hand, the first column of U_N is computed as shown in Figure 5.

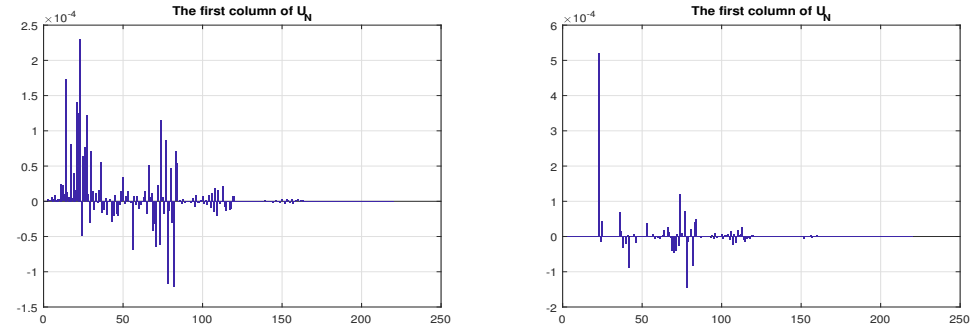

Figure 5. (Example 3, $m = 9$.) **Left** panel: the first column of U_N, computed in Matlab as $\texttt{pinv}(O_X) * O_Y$. Its norm is $\|U_N(:,1)\|_2 \approx 5.0350 \times 10^{-4}$. **Right panel**: the first column of $U_N = O_X \backslash O_Y$, with norm $\|U_N(:,1)\|_2 \approx 5.9611 \times 10^{-4}$. The true value of $U_N(:,1)$ should be $e_1 = (1, 0, \ldots, 0)^T$.

Two conclusions are immediate. First, O_X is considered (by the software) numerically rank deficient. Secondly, two different software solutions of the same problem return two different solutions. For the data shown in Figures 3 and 4, we computed U_N as $\texttt{pinv}(O_X) * O_Y$. This is in general not a good idea for solving least squares problems; nevertheless, it can be often seen in the literature and software implementations as a textbook-style numerical method for least squares problems, albeit a wrong one. We included it here for instructive purposes. If we repeat the experiment with the backslash operator, $O_X \backslash O_Y$, the results are, unfortunately, not better. One conspicuous difference is that, instead of the cluster of absolutely small eigenvalues of $\texttt{pinv}(O_X) * O_Y$ (see Figure 4), $O_X \backslash O_Y$ has a zero eigenvalue of multiplicity 45. This multiple zero eigenvalue is a consequence of the sparsity structure of $O_X \backslash O_Y$, see Figure 6.

In the course of solving the least squares problem, in both methods (explicit use of the pseudoinverse or the backslash operator), the coefficient matrix O_X is truncated (small singular values are set to zero if $\texttt{pinv}(O_X)$ is used; the upper triangular factor is truncated in the case of backslash, which uses a rank revealing QR factorization) and the computed U_N, due to rounding errors, is small a perturbation of a rank-deficient matrix

with a number of eigenvalues in the vicinity of zero. As a result, the matrix logarithm function fails.

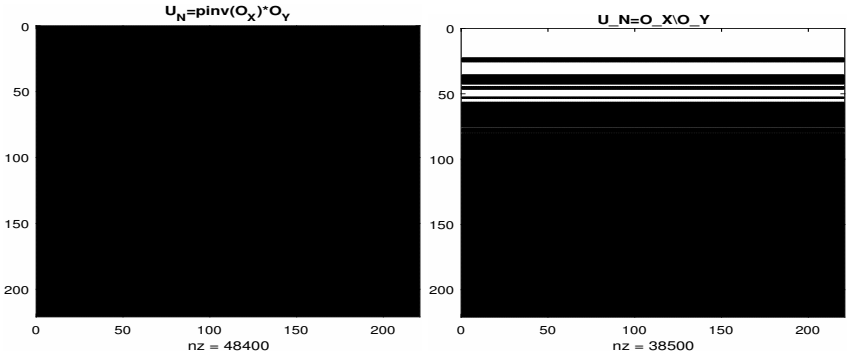

Figure 6. (Example 3, $m = 9$.) The sparsity structure of $\texttt{pinv}(O_X) * O_Y$ (numerical rank 176) and $O_X \backslash O_Y$ (numerical rank 175). The backslash operator uses the rank revealing (column pivoted) QR factorization and, by truncation, returns a sparse solution.

The data shown in Figure 7 are instructive. The condition number of O_X and the distribution of its singular values are nicely revealed by the column norms of O_X. The LS solver automatically truncates small singular values that originate from small columns (small basis function values over the snapshots $\mathbf{x}_k$) and not necessarily from collinearity in the sense of small angles between basis functions.

Look at the singular values $\sigma_1(U_N) \geq \cdots \geq \sigma_N(U_N)$ in the right panel, in particular the gap $\sigma_{176}(U_N) \approx 1.4058 \times 10^{-6} \gg \sigma_{177}(U_N) \approx 3.9261 \times 10^{-10}$. The numerical rank of U_N is determined as 176 because $\sigma_{176}(U_N)$ is the smallest singular value that is above the threshold $N \cdot \texttt{eps} \cdot \sigma_1(U_N) \approx 4.6543 \times 10^{-8}$. The index 176 originates in the numerical rank of O_X which is determined in Matlab ($\texttt{rank}(O_X)$ returns 176) by the default tolerance. Note that there is no such visible gap (cliff) in the ordered singular values of O_X.

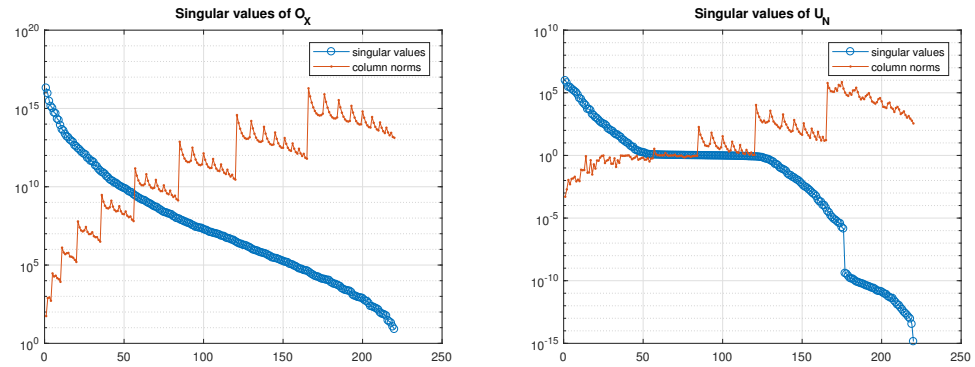

Figure 7. The (computed) singular values of $O_X \in \mathbb{R}^{3025\times 220}$ and $U_N \in \mathbb{R}^{220\times 220}$, and the column norms. The numerical rank of O_X and U_N is 176, which is considerably below the full rank 220. If U_N is computed as $O_X \backslash O_Y$, the numerical rank is 175. The results are similar (with the numerical rank 163) if the number of snapshots is increased to 24,025 as explained in Example 3.

Remark 3. *It should be also mentioned here that the cosines of the canonical angles between the ranges of O_X and O_Y are between 0.988 and one (cf. Section 2.1.3), and that we expect the spectrum of U_N not to be too far away from the unit circle. The intuition is that (in the process of computation of $O_X^{\dagger} O_Y$) the truncated part of O_X did not (could not) cancel the corresponding part in O_Y, which resulted in the problematic cluster of eigenvalues near zero. (In Figure 3, the number of eigenvalues in the cluster around zero is 44, which is the numerical rank deficiency, because the numerical rank is determined (from the singular values) as 176.)*

Remark 4. *In the computations* $\texttt{pinv}(O_X) * O_Y$ *and* $O_X \backslash O_Y$*, the truncation is conducted based on the SVD and the rank revealing QR factorization, respectively, of* O_X*, independent of* O_Y*. It is more appropriate to solve each LS problem (7) separately (for the corresponding column of* U_N*), and use the truncation strategy following Rust [28]. Note that this is a more general issue because the matrix* $O_X^\dagger O_Y$ *is often used in the Koopman/DMD setting. (See [29] for a detailed numerical analysis of the DMD and its variations.) Furthermore, the problem can become even more difficult if we have weighting matrix W with scaling factors that spread over several orders of magnitude. (In this paper, we work with* $W = \mathbb{I}_K$ *for the sake of brevity.)*

Remark 5. *In the case of noisy data, it might be advantageous to replace the least squares fit with the total least squares ([20], §2.7.6). This is an important issue and we leave it for our future work.*

5. Computing $[L_N]_{\mathcal{B}}$ with Preconditioning

Numerical examples in Section 4 show that implementation of the method in state of the art packages such as Matlab which is straightforward and effortless: the key computation is coded as $\texttt{logm}(O_X \backslash O_Y)$ or as $\texttt{logm}(\texttt{pinv}(O_X) * O_Y)$. However, the results are not always satisfactory, and the problems are both in the computation of U_N (solving the least squares problem) and in computing the matrix logarithm. In this section, we use the structure of the least squares problem solver (reviewed in Section 2.2.1) to introduce a simple modification and then to construct preconditioned computations of $\log U_N$. These techniques can be combined with the dual method that is analyzed in detail in Sections 6 and 7.

5.1. A Simple Modification

The first attempt to avoid some of the problems illustrated in Section 4 is to prevent truncation when computing $O_X^\dagger O_Y$. So we simply run the procedure used in the backslash solver (see Section 2.2.1), but without truncation. More precisely, using the column pivoted QR factorization $O_X \Pi = QR$, U_N is computed as $U_N = \Pi R^{-1} Q^T O_Y$.

With $m = 9$ and same setting as in Example 1, the computation of the logarithm of U_N was successful. (Matlab function `logm()` computed a real valued logarithm, without an error/warning message) and the coefficients of (23) are recovered to four digits of accuracy, which is satisfactory. The remaining coefficients are below $O(10^{-4})$ and are discarded in the Formula (19) for reconstructing **F**. The first column of U_N was e_1 up to an error of the order of the roundoff.

The results shown in Figure 8 are encouraging, because they show that the data contain information that can be turned into more accurate output, provided that the algorithm successfully curbs the ill-conditioning. Furthermore, test with $m = 10$ ($\kappa_2(U_N) \approx 2.9 \times 10^{20}$) showed nearly the same four digit accuracy; with $m = 11$ ($\kappa_2(U_N) \approx 2.2 \times 10^{24}$) the accuracy dropped to two digits (but the logarithm was successfully computed); with $m = 12$ ($\kappa_2(U_N) \approx 7.7 \times 10^{26}$) the logarithm is still computed as real matrix, but the accuracy of the computed coefficients drops to $O(10^{-1})$ and the reconstruction of **F** fails. However, if we increase K to 24,025 (by sampling more points from more trajectories) the accuracy rebounds to at least for digits both in the coefficients and the reconstruction of **F**. At $m = 16$ ($N = 969$ and $K = 24{,}025$, or $K = 3025$) the accuracy is lost. The condition number of U_N is greater than 10^{37}.

Our adversarial testing is used to expose potential weaknesses of the computational steps in a software implementation of the algorithm. Of course, all these outcomes may vary, depending on the number of trajectories, initial conditions, sampling resolution; in some examples it is possible that the computation of the matrix logarithm breaks down even for smaller values of m. (Moreover, numerical accuracy, i.e., the conditioning of the problem, heavily depends on the underlying dynamics). In any case, at this point we have to conclude that using the $\log O_X^\dagger O_Y$ is a source of nontrivial numerical difficulties that preclude efficient and robust deployment of the Mauroy–Goncalves method. The

theoretical convergence (as $N, K \to \infty$, $t \to 0$) is not matched by numerical convergence of a straightforward implementation of the method in finite precision computation.

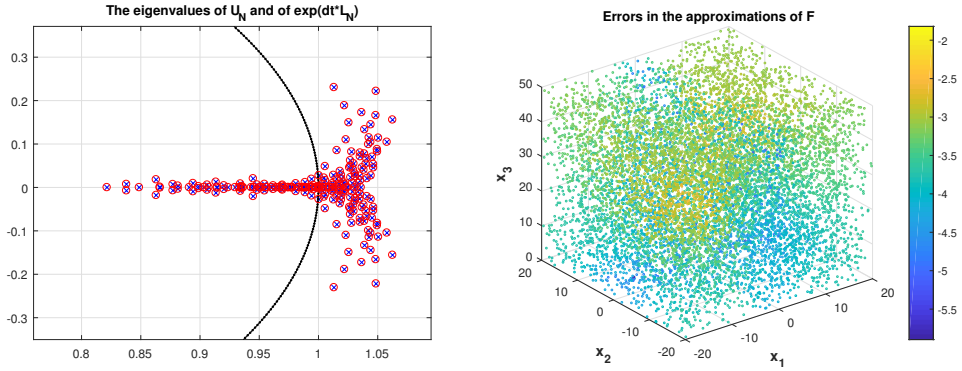

Figure 8. Left panel: The (computed) eigenvalues of $U_N \in \mathbb{R}^{220\times 220}$ ($\times$), and the eigenvalues of $exp(\delta t L_N)$ ($\circ$). The maximal relative difference between the matching eigenvalues is computed as 8.1×10^{-9}. Here, U_N is computed as $U_N = \Pi R^{-1} Q^T O_Y$ without any truncation of R. The diagonal entries of R span, in absolute value, the range between 1.9×10^{16} and 1.0×10^{1}. This reveals the condition number of U_N which is computed as 7.4×10^{16} by the Matlab function `cond()`. **Right panel**: The values of $\log_{10} \epsilon_k$ are defined in (24) for 12,000 randomly selected points in the box $[-20, 20] \times [-20, 20] \times [0, 50]$. Compare with Figures 3, 4 and 7.

5.2. Preconditioning the Logarithm of $O_X^\dagger O_Y$

The discussion in Sections 4.2 and 5.1 is only one step toward more robust computation—it merely identifies the problem and shows how small changes in an implementation substantially change the final result. Clearly, sufficiently accurate computation of U_N is among the necessary conditions for successful computation of the matrix algorithms. However, this is not sufficient; even if we compute U_N accurately, computation of the logarithm may fail if the matrix is ill-conditioned.

We now introduce a new scheme that uses functional calculus-based preconditioning. If S is any nonsingular matrix, then

$$\log(O_X^\dagger O_Y) = S \log(S^{-1}(O_X^\dagger O_Y)S)S^{-1} \equiv S \log((O_X S)^\dagger O_Y S)S^{-1}. \tag{25}$$

Note that replacing $O_X^\dagger O_Y$ with the similar matrix $S^{-1}(O_X^\dagger O_Y)S$ corresponds to changing the basis for matrix representation of the compressed Koopman operator. With the experience from Section 4.2, it is clear that the key is to compute the preconditioned matrix $S^{-1}(O_X^\dagger O_Y)S$ without first computing $O_X^\dagger O_Y$. (Once we compute $O_X^\dagger O_Y$ explicitly in floating point arithmetic and store it in the machine memory, it may be then too late even for exact computation.)

The conditions on S are:

(i) Tt should facilitate more accurate computation of the argument $S^{-1}(O_X^\dagger O_Y)S = (O_X S)^\dagger O_Y S$ for the matrix logarithm;
(ii) It should have preconditioning effect for computing the logarithm of $S^{-1}(O_X^\dagger O_Y)S$;
(iii) The application of S and S^{-1} should be efficient and numerically stable.

Example 4. To test the concept, we use the same data as in Example 1 with $m = 9$, and for the matrix S we take $S = \mathrm{diag}(1/\|O_X(:,i)\|_2)_{i=1}^N$ and compute

$$L_N = \frac{1}{t} S \log((O_X S)^\dagger (O_Y S))S^{-1}. \tag{26}$$

Contrary to the failure of the formula $L_N = (1/t)\log(O_X^\dagger O_Y)$, (26) computes the real logarithm of the explicitly computed $(O_X S)^\dagger (O_Y S)$, and recovers the coefficients with an

$O(10^{-5})$ relative error. That is, we scale O_X and O_Y by $\mathrm{diag}(1/\|O_X(:,i)\|_2)_{i=1}^N$, and then proceed by solving the least squares problem with the thus scaled matrices. To understand the positive result, we first note that the condition number of $O_X S$ is $\kappa_2(O_X S) \approx 1.3 \times 10^9$, so the least squares solution is computed without truncation. The condition number of $(O_X S)^\dagger (O_Y S)$ was $\kappa_2((O_X S)^\dagger (O_Y S)) \approx 6.4 \times 10^5$. (The condition number of $O_X^\dagger O_Y$ is $O(10^{20})$ so that the matrix is considered numerically rank deficient.) With $m = 12$ we had $\kappa(U_N) \approx 6.9 \times 10^{61}$ and $\kappa_2((O_X S)^\dagger (O_Y S)) \approx 1.4 \times 10^{13}$; the coefficients are recovered to three accurate digits and approximation of **F** is with error slightly larger than the one in the right panel of Figure 8.

However, as m increases, the diagonal scaling cannot cope with the increased condition number; already at $m = 15$, a complex non-principal value of the logarithm is computed, with some eigenvalues whose imaginary parts are equal $\pi/\delta t$. The computed U_N has a cluster of eigenvalues around zero. For the record, $\kappa_2(U_N) \approx 2.2 \times 10^{220}$, $\kappa_2((O_X S)^\dagger (O_Y S)) \approx 6.7 \times 10^{78}$. Surprisingly, the approximate coefficients, although complex, have small imaginary parts (of the order of the roundoff) and their real parts still provide reasonably good approximations of the true coefficients.

5.2.1. Scaled QR Factorization Based Preconditioner

To develop a stronger preconditioner, we start with the following observation: No matter how ill-conditioned O_X and O_Y may be (in the sense of badly scaled columns), the distance between the ranges of O_X and O_Y, as measured by the canonical angles between the subspaces, should not be too big. (Intuitively, O_Y contains the observables evaluated at the states $\mathbf{y}_k$ downstream in time δt from the $\mathbf{x}_k$'s used in O_X. Recall Remark 3.)

Hence, if we compute the QR factorization of O_X, the inverse of its triangular factor will have a preconditioning effect on O_Y by postmultiplication. This leads to Algorithm 1.

Algorithm 1 $[L_N] = \mathrm{Inf\_Generator\_QRSC}(O_X, O_Y, T)$

Input: $O_X \in \mathbb{C}^{K\times N}$, $O_Y \in \mathbb{C}^{K\times N}$, $T > 0$
1: $S = \mathrm{diag}(1/\|O_X(:,i)\|_2)_{i=1}^N$
2: $[Q_X, \widehat{R}_X] = \mathrm{qr}(O_X S)$ {*QR factorization*}
3: $\widehat{U}_N = Q_X^T (O_Y S) \widehat{R}_X^{-1}$ {$\widehat{U}_N$ *is similar to* $O_X^\dagger O_Y$.}
4: $\widehat{L}_N = \log(\widehat{U}_N)$
5: $L_N = (1/T) S (\widehat{R}_X^{-1} \widehat{L}_N \widehat{R}_X) S^{-1}$
Output: L_N. $\{L_N = (1/T)\log(O_X^\dagger O_Y)\}$

Example 5. *To test this algorithm, we use the data from Example 1, take* $m = 15$ *and increase the number of snapshots to K = 24,025. The matrix* $\widehat{U}_N$ *is well conditioned,* $\kappa_2(\widehat{U}_N) \approx 1.2 \times 10^2$, *and computing the matrix logarithm is successful. The coefficients are recovered to four digits of accuracy, and the reconstruction of* **F** *is slightly better than the one shown in the right panel in Figure 8.*

This example, as a test of the proposed approach, is encouraging. Our next task is to further develop the method along the lines of Algorithm 1, and to provide a robust method with accompanying numerical analysis, and finally to implement it as a reliable software toolbox.

5.2.2. Pivoted QR Factorization Based Preconditioner

Since in this approach the matrix logarithm is the most critical and numerically most difficult computational task, the preprocessing/preconditioning aims to ensure successful completion of that particular step in the method. The back application of the similarity is also an important step. In Algorithm 1, the main preconditioning is performed by an upper triangular factor from the QR factorization, and by its inverse. For a numerically robust computation, it is important that the QR factorization is computed accurately even

in the case of wildly scaled data, and, moreover, that the resulting triangular factor is rank revealing and well structured. These goals can be accomplished by pivoting. In this subsection we outline the main principles along which the idea of QR factorization-based preconditioned computation of the matrix L_N (matrix representation of a compression of the infinitesimal generator) can be further pursued.

The column pivoting has the rank revealing property and the triangular factor is diagonally dominant in a very strong sense, see e.g., [23,30] and (13).

In the case of Businger–Golub pivoting, we know that $R_X = \Delta_X T_X$, where $\Delta_X = \mathrm{diag}(|(R_X)_{ii}|)_{i=1}^N$ and T_X is well conditioned. Hence, $R_X^{-1} = T_X^{-1}\Delta_X^{-1}$, and after Line 3 in Algorithm 2 one can insert another preconditioning with Δ_X. We omit the details for the sake of brevity. Instead, we conclude this theme with few remarks that should be useful for further study and implementation.

Algorithm 2 $[L_N] = \mathrm{Inf\_Generator\_QRCP}(O_X, O_Y, T)$

Input: $O_X \in \mathbb{C}^{K\times N}$, $O_Y \in \mathbb{C}^{K\times N}$, $T > 0$

1: Reorder the snapshots by simultaneous row permutation of O_X and O_Y; see Remark 6.
2: $[Q_X, R_X, \Pi_X] = \mathrm{qr}(O_X)$ {*Rank revealing QR factorization with column pivoting*}
3: $\widehat{U}_N = Q_X^T(O_Y\Pi_X)R_X^{-1}$ {$\widehat{U}_N$ *is similar to* $O_X^\dagger O_Y$.}
4: $\widehat{L}_N = \log(\widehat{U}_N)$
5: $L_N = (1/T)\Pi_X(R_X^{-1}\widehat{L}_N R_X)\Pi_X^T$

Output: L_N. $\{L_N = (1/T)\log(O_X^\dagger O_Y)\}$

Remark 6. *For the numerical accuracy of the QR factorization, an additional row pivoting may be needed to obtain that the rows are ordered so that their ℓ_∞ norms are decreasing, see [31,32]. If Ψ is a permutation matrix that encodes the row pivoting, then $(\Psi O_X)^\dagger = O_X^\dagger \Psi^T$, so that $(\Psi O_X)^\dagger(\Psi O_Y) = O_X^\dagger O_Y$. This means that using the additional row pivoting in the QR factorization in Algorithm 2 is equivalent to a particular ordering of the data snapshots. The column pivoting corresponds to reordering the basis' functions. Both reorders of the data are allowed operations and can thus be used to enhance numerical robustness of the computation.*

Remark 7. *If $K \gg 2N$ then it pays off to change the coordinates by computing the QR factorization*

$$\begin{pmatrix} Q_X & Q_Y \end{pmatrix} = \mathcal{Q}\begin{pmatrix} \mathcal{R} \\ \mathbf{0} \end{pmatrix},$$

and use the corresponding columns of $\mathcal{R}$ instead of O_X and O_Y. This follows the idea of the QR-compressed DMD [29].

Remark 8. *The key assumption in the above described method is that $K \gg N$, i.e., that both O_X and O_Y are tall matrices; their columns are in a high dimensional space and with suitable transformation S in the column spaces ($O_X \mapsto O_X S$, $O_Y \mapsto O_Y S$) we can improve the condition numbers. By the (variational) monotonicity principle, supplying more snapshots (increasing K) moves the singular values of O_X and O_Y to the right, thus improving the condition numbers of both matrices. Since, by the underlying continuity, the canonical angles between the ranges of O_X and O_Y are expected to be away from $\pi/2$, and $U_N = O_X^\dagger O_Y$ is going to be nonsingular. Moreover, the overall condition number of the computation can be controlled using our proposed modifications that are designed to ensure stable computation of the matrix logarithm.*

6. Dual Method

It is clear that the values of N linearly independent functions $\wp_1, \ldots, \wp_N$ over the discrete set $\mathbf{x}_1, \ldots, \mathbf{x}_K$ of $K < N$ snapshots (that is, with only the tabulated values in the $K \times N$ matrices O_X, O_Y) contain redundancy. On the other hand, increasing the space dimension N is a way to lift the data to higher dimensional space; more observables improve both the DMD and KMD analyses. Note that, in the case $N > K$, both the

compression $U_N = [\Phi_N \mathbb{U}^t_{|\mathcal{F}_N}]_{\mathcal{B}} = O_X^\dagger O_Y$ and the EDMD matrix (U_N^T) are rank deficient; unfortunately, the infinitesimal generator identification framework cannot work in that setting because the matrix logarithm is not defined. It should be stressed here that, e.g., in the DMD setting, the action of the operator given by the data is restricted to an at most K-dimensional subspace in the N dimensional space, and that the approximations of the Koopman modes are obtained by a Rayleigh–Ritz extraction. Hence, any operator (matrix) function of U_N only makes sense and has a practical usability in the context of an approximation from the subspace defined by the data.

In [8], a dual method is proposed, which instead of $U_N = O_X^\dagger O_Y$ works with the logarithm of $U_K = O_Y O_X^\dagger$. In this section we first provide, in Section 6.1, a detailed linear algebra description of the dual method that will facilitate a more general formulation that allows for modifications which may lead to better numerical algorithms. In fact, we show in Section 7 that the dual method of [8] is but a special case of subspace projection methods, and we show how to exploit this for design of numerically better schemes.

6.1. A Rayleigh Quotient Formulation

In the dual formulation, the transition from U_N to U_K can be formulated as another compression of $\mathbb{U}^t$ onto a particular K-dimensional subspace of $\mathcal{F}_N$.

Proposition 3. *If we define* $\begin{pmatrix}\psi_1(x) & \dots & \psi_K(x)\end{pmatrix} = \begin{pmatrix}\wp_1(x) & \dots & \wp_N(x)\end{pmatrix} O_X^\dagger$, *then (assuming* O_X *is of full row rank K),* $\mathcal{B}_K = \{\psi_1, \dots, \psi_K\}$ *is a basis of the K dimensional subspace* $\mathcal{F}_K \subset \mathcal{F}_N$, *and* $U_K = O_Y O_X^\dagger$ *is the matrix representation of the Rayleigh quotient* $\Phi_K \mathbb{U}^t_{|\mathcal{F}_K}$ *in which* $\Phi_K : \mathcal{F} \to \mathcal{F}_K$ *is the least squares projection as in Section 2.1.1. The matrix* U_K *is the matrix Rayleigh quotient of* U_N *with respect to the range of* $O_X^\dagger$, *i.e.,* $U_K = O_X U_N O_X^\dagger$ *and* $U_N O_X^\dagger = O_X^\dagger U_K$. *Furthermore, the Rayleigh quotient with respect to* $O_X^\dagger$ *is the same for all matrices from the set* $\mathcal{N}$ *(see (10)):*

$$O_X(O_X^\dagger O_Y + \Psi_0 \Xi) O_X^\dagger = O_Y O_X^\dagger. \quad (\textit{Note here that} \quad (O_X^\dagger)^\dagger = O_X.) \tag{27}$$

Proof. First, note that the basis functions $\psi_1, \dots, \psi_K$ evaluated at $\mathbf{x}_1, \dots, \mathbf{x}_K$ can be tabulated as the matrix

$$O_{X,K} = \begin{pmatrix} \psi_1(\mathbf{x}_1) & \psi_2(\mathbf{x}_1) & \dots & \psi_K(\mathbf{x}_1) \\ \vdots & \vdots & \dots & \vdots \\ \psi_1(\mathbf{x}_K) & \psi_2(\mathbf{x}_K) & \dots & \psi_K(\mathbf{x}_K) \end{pmatrix} = O_X O_X^\dagger = \mathbb{I}_K,$$

and that for a $g \in \mathcal{F}_K$, its representation in the basis $\mathcal{B}_K$ is $[g]_{\mathcal{B}_K} = \big(g(\mathbf{x}_1), \dots, g(\mathbf{x}_K)\big)^T$; see (6), where we take $W = \mathbb{I}$ for the sake of simplicity. Similarly, $(\mathbb{U}^t \psi_1, \dots, \mathbb{U}^t \psi_K)$ evaluated at $\mathbf{x}_1, \dots, \mathbf{x}_K$ yields the matrix $O_{Y,K} = O_Y O_X^\dagger$. Now, as in Section 2.1.2, $U_K = O_{X,K}^\dagger O_{Y,K} = O_Y O_X^\dagger$. Relation (27) is easily checked. □

If U_K is nonsingular then the identification scheme should be recast in terms of $\log U_K$. In the basis $\mathcal{B}_K$ we have then the following representations of the Rayleigh quotient $\Phi_K \mathbb{U}^t_{|\mathcal{F}_K}$ and its logarithm.

Corollary 1. *The matrix representations of the compressed* $\mathbb{U}^t$ *and its logarithm (if defined) are as follows: For any* $(\mathrm{g}_1, \dots, \mathrm{g}_K)^T \in \mathbb{C}^K$,

$$\Phi_K \mathbb{U}^t_{|\mathcal{F}_K}\Big(\begin{pmatrix}\psi_1(x) & \dots & \psi_K(x)\end{pmatrix}\begin{pmatrix}\mathrm{g}_1 \\ \vdots \\ \mathrm{g}_K\end{pmatrix}\Big) = \begin{pmatrix}\psi_1(x) & \dots & \psi_K(x)\end{pmatrix}\Big(U_K \begin{pmatrix}\mathrm{g}_1 \\ \vdots \\ \mathrm{g}_K\end{pmatrix}\Big),$$

$$\log(\Phi_K \mathbb{U}^t_{|\mathcal{F}_K})\Big(\begin{pmatrix}\psi_1(x) & \dots & \psi_K(x)\end{pmatrix}\begin{pmatrix}\mathrm{g}_1 \\ \vdots \\ \mathrm{g}_K\end{pmatrix}\Big) = \begin{pmatrix}\psi_1(x) & \dots & \psi_K(x)\end{pmatrix}\Big(\log(U_K) \begin{pmatrix}\mathrm{g}_1 \\ \vdots \\ \mathrm{g}_K\end{pmatrix}\Big).$$

Next, we consider function evaluation at $\mathbf{y}_k = \varphi^t(\mathbf{x}_k)$, $k = 1, \ldots, K$.

Proposition 4. *Let $f = \sum_{i=1}^{N} \mathsf{f}_i \wp_i \in \mathcal{F}_N$. Then*

$$\begin{pmatrix} \mathbb{U}^t f(\mathbf{x}_1) \\ \vdots \\ \mathbb{U}^t f(\mathbf{x}_K) \end{pmatrix} = \begin{pmatrix} f(\mathbf{y}_1) \\ \vdots \\ f(\mathbf{y}_K) \end{pmatrix} = O_Y O_X^{\dagger} \begin{pmatrix} f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_K) \end{pmatrix} + O_Y(\mathbb{I}_N - O_X^{\dagger} O_X) \begin{pmatrix} \mathsf{f}_1 \\ \vdots \\ \mathsf{f}_N \end{pmatrix}. \tag{28}$$

If $\mathbb{I}_N \neq O_X^{\dagger} O_X$, then the second term on the right hand side in (28) is zero if and only if the f_i's are of the form $(\mathsf{f}_i)_{i=1}^{N} = O_X^{\dagger}(\mathsf{g}_k)_{k=1}^{K}$ with arbitrary g_k's. Furthermore, if $g = \sum_{i=1}^{K} \mathsf{g}_i \boldsymbol{\psi}_i \in \mathcal{F}_K \subset \mathcal{F}_N$, then $\mathsf{g}_i = g(\mathbf{x}_i)$ and

$$\begin{pmatrix} \mathbb{U}^t g(\mathbf{x}_1) \\ \vdots \\ \mathbb{U}^t g(\mathbf{x}_K) \end{pmatrix} = O_Y O_X^{\dagger} \begin{pmatrix} g(\mathbf{x}_1) \\ \vdots \\ g(\mathbf{x}_K) \end{pmatrix}, \quad \begin{pmatrix} \log(\Phi_K \mathbb{U}^t_{|\mathcal{F}_K}) g(\mathbf{x}_1) \\ \vdots \\ \log(\Phi_K \mathbb{U}^t_{|\mathcal{F}_K}) g(\mathbf{x}_K) \end{pmatrix} = \log(O_Y O_X^{\dagger}) \begin{pmatrix} g(\mathbf{x}_1) \\ \vdots \\ g(\mathbf{x}_K) \end{pmatrix}. \tag{29}$$

Proof. For (28), it suffices to write

$$\begin{pmatrix} f(\mathbf{y}_1) \\ \vdots \\ f(\mathbf{y}_K) \end{pmatrix} = O_Y \begin{pmatrix} \mathsf{f}_1 \\ \vdots \\ \mathsf{f}_N \end{pmatrix} = O_Y(O_X^{\dagger} O_X + \mathbb{I}_N - O_X^{\dagger} O_X) \begin{pmatrix} \mathsf{f}_1 \\ \vdots \\ \mathsf{f}_N \end{pmatrix}.$$

For the first relation in (29), note that $g = \sum_{i=1}^{K} \mathsf{g}_i \boldsymbol{\psi}_i$ evaluated at $\mathbf{y}_1, \ldots, \mathbf{y}_K$ reads

$$\begin{pmatrix} g(\mathbf{y}_1) \\ \vdots \\ g(\mathbf{y}_K) \end{pmatrix} = O_Y O_X^{\dagger} \begin{pmatrix} \mathsf{g}_1 \\ \vdots \\ \mathsf{g}_K \end{pmatrix}, \quad \text{where} \quad \mathsf{g}_i = g(\mathbf{x}_i) \quad \text{because } O_{X,K} = \mathbb{I}_K.$$

□

Hence, using the last relation and $\mathbb{L} = \mathbf{F} \cdot \nabla \approx (1/t) \log(\Phi_K \mathbb{U}^t_{|\mathcal{F}_K})$ in $\mathcal{F}_K$ (evaluated at the snapshots $\mathbf{x}_i$), we have

$$\begin{pmatrix} \mathbf{F} \cdot \nabla g(\mathbf{x}_1) \\ \vdots \\ \mathbf{F} \cdot \nabla g(\mathbf{x}_K) \end{pmatrix} \approx \frac{1}{t} \log(O_Y O_X^{\dagger}) \begin{pmatrix} g(\mathbf{x}_1) \\ \vdots \\ g(\mathbf{x}_K) \end{pmatrix}. \tag{30}$$

If we choose $g(\mathbf{x}) = x_j$, $j = 1, \ldots, n$, respectively, (where $\mathbf{x} = (x_1, \ldots, x_n)^T$) then $\mathbf{F} \cdot \nabla g(\mathbf{x}) = F_j(\mathbf{x})$ and we obtain approximate filed values $\widetilde{\mathbf{F}(\mathbf{x}_i)}$ defined as

$$\begin{pmatrix} \widetilde{\mathbf{F}(\mathbf{x}_1)}^T \\ \vdots \\ \widetilde{\mathbf{F}(\mathbf{x}_K)}^T \end{pmatrix} = \frac{1}{t} \log(O_Y O_X^{\dagger}) \begin{pmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_K^T \end{pmatrix}. \tag{31}$$

Note however that (30), (31) require that $g \in \mathcal{F}_K$. Furthermore, the approximations of $\mathbf{F}$ are given only at the sequence $\mathbf{x}_1, \ldots, \mathbf{x}_K$.

Global Identification of **F**

After identifying the values $\mathbf{F}(\mathbf{x}_k)$, the idea is to use the ansatz

$$\mathbf{F}(x) = \begin{pmatrix} F_1(x) \\ \vdots \\ F_n(x) \end{pmatrix}, \quad F_j(x) = \sum_{k=1}^{N_F} \varphi_{kj} h_j(x), \tag{32}$$

where $h_1, \ldots, h_{N_F}$ are chosen from a dictionary of functions, possibly different from the basis used to lift the data and identify the compression of the infinitesimal generator. Then we obtain the sequence of least squares problems

$$\left\| \begin{pmatrix} h_1(\mathbf{x}_1) & \ldots & h_{N_F}(\mathbf{x}_1) \\ h_1(\mathbf{x}_2) & \ldots & h_{N_F}(\mathbf{x}_2) \\ \vdots & \ldots & \vdots \\ h_1(\mathbf{x}_K) & \ldots & h_{N_F}(\mathbf{x}_K) \end{pmatrix} \begin{pmatrix} \varphi_{1j} \\ \varphi_{2j} \\ \vdots \\ \varphi_{N_F j} \end{pmatrix} - \begin{pmatrix} \widetilde{F}_j(\mathbf{x}_1) \\ \widetilde{F}_j(\mathbf{x}_2) \\ \vdots \\ \widetilde{F}_j(\mathbf{x}_K) \end{pmatrix} \right\|_2^2 \longrightarrow \min_{\varphi_{1j} \cdots \varphi_{N_F j}}, \quad j = 1, \ldots, n, \tag{33}$$

that can also be equipped with a regularization factor that promotes sparse solution.

6.2. A Numerical Example

As in Section 4, a simple but difficult example can be used to explore the numerical feasibility of the derived scheme. It has been shown in [8] that the method works well for the Lorenz system on small time intervals. In the following example, we increase the time domain and test the accuracy of the reconstruction of **F**. An ill-conditioned problem is obtained by taking the total degree of the polynomials to be 12. Although this may seem artificial, it is useful because it provides numerically difficult cases that expose the weaknesses of the computational scheme and excellent case studies for better understanding and further development.

Example 6. *In this example we run simulation of the Lorenz system with time resolution* $\delta t = 10^{-3}$ *on the intervals* $[0, 0.1]$, $[0, 0.18]$ *and* $[0, 0.19]$. *The basis functions* $\wp_i$ *are the monomials with total degree up to* $m = 12$. *We generate* 12 *trajectories with random initial conditions, and from each trajectory we sample three consecutive snapshots at ten randomly selected positions; the matrices* O_X, O_Y *are thus* 360×455. *In the reconstruction Formula (31), the matrix logarithm is computed in Matlab as* `logm(OY/OX)`; *in all three runs,* `logm()` *issued a warning that a non-principal value of the algorithm was computed. The reconstruction error is measured component-wise and snapshot-wise as*

$$\epsilon_k^{(i)} = \frac{|\widetilde{F_i(\mathbf{x}_k)} - F_i(\mathbf{x}_k)|}{\|\mathbf{F}(\mathbf{x}_k)\|_\infty}, \quad i = 1, \ldots n; \quad k = 1, \ldots, K. \tag{34}$$

In addition, we measure the total error of the tabulated values in the Frobenius norm as

$$\tau = \|(\widetilde{F_i(\mathbf{x}_k)})_{k,i=1}^{K,n} - (F_i(\mathbf{x}_k))_{k,i=1}^{K,N}\|_F \,/\, \|(F_i(\mathbf{x}_k))_{k,i=1}^{K,N}\|_F. \tag{35}$$

In Figure 9, we show the errors $\epsilon_k^{(i)}$ *for the intervals* $[0, 0.1]$ *and* $[0, 0.18]$; *in the case of the time interval* $[0, 0.19]$, *the method broke down and the reconstructed values were computed as* `NaN`*'s.*

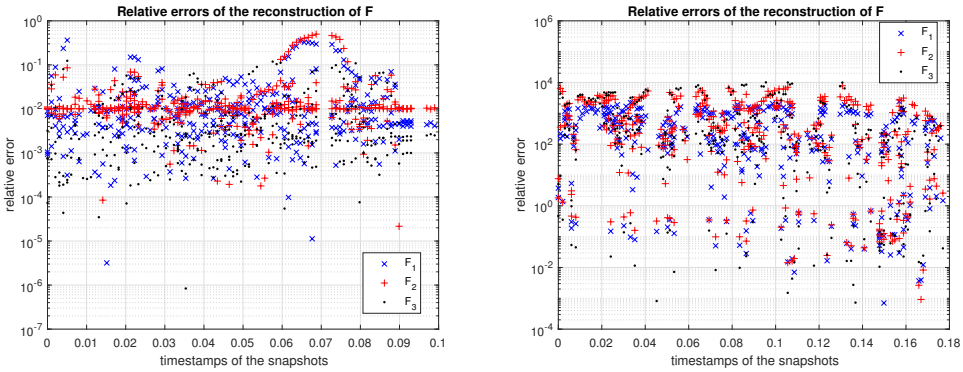

Figure 9. The errors $\epsilon_k^{(i)}$ (34). **Left panel:** time interval $[0, 0.1]$. The total error is $\tau = 2.2259 \times 10^{-2}$. **Right panel:** time interval $[0, 0.18]$. The total error is $\tau = 4.6174 \times 10^{2}$. Similar results are obtained using `logm(Y*pinv(X))`. On the other hand, while on the interval $[0, 0.19]$ `logm(Y/X)` fails producing `NaN`'s, `logm(Y*pinv(X))` completed without `NaN` exceptions, alas with the error $\tau = 5.1718 \times 10^{2}$ and with $\epsilon_k^{(i)}$ similar to that in the right panel above.

The results depend on the initial conditions used to generate sample trajectories. In the above experiments, each initial condition is taken as a normally distributed 3×1 *vector generated in Matlab using* `randn`*. If we generate the initial conditions as uniformly distributed inside the sphere of radius* 0.1*, centered at the origin, then for all three test intervals the error* τ *was* $O(10^{-3})$*. With such initialization, the same level of accuracy is then maintained for larger time intervals, up to* $[0, 0.4]$*; for* $[0, 0.41]$ *the error increased to* $\tau \approx 0.1$ *and for* $[0, 0.42]$ *the result was* `NaN`*.*

This example shows that numerical (software) implementation of the dual method requires additional analysis and modifications, similar to the main method. In the next section we explore numerical linear algebra techniques that could facilitate a more robust implementation.

7. Subspace Selection

The approximation (31) is based on a particular K-dimensional subspace of $\mathcal{F}_N = \mathrm{span}(\wp_1, \ldots, \wp_N)$, where (in the case of monomial basis) the choice of the new basis functions precludes direct coefficient comparisons with (17), (19). Instead, the identification procedure follows the lines of (30)–(33).

We can set up a more general framework: independent of the ratio N/K (and independent of the formulation—original or dual) we can seek other suitable subspaces of $\mathcal{F}_N$, and not necessarily of dimension K. The selection criterion is numerical: both the subspace and its dimension $\widehat{N}$ should be determined with respect to the numerical conditioning of the matrix representations at the finite sequence $\mathbf{x}_1, \ldots, \mathbf{x}_K$. A basis of such a $\widehat{N}$-dimensional subspace $\mathcal{F}_{\widehat{N}}$ of $\mathcal{F}_N$ is written as $(\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{\widehat{N}}) = (\wp_1, \ldots, \wp_N)\mathcal{S}$, where $\mathcal{S}$ is $N \times \widehat{N}$ selection operator, i.e., matrix, of rank $\widehat{N}$. The tabulated values of $(\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{\widehat{N}})$ at $\mathbf{x}_1, \ldots, \mathbf{x}_K$ are easily computed as $O_{X,\mathcal{S}} = O_X\mathcal{S}$. Similarly, the tabulated values of $(\mathbb{U}^t\boldsymbol{\psi}_1, \ldots, \mathbb{U}^t\boldsymbol{\psi}_{\widehat{N}})$ are $O_{Y,\mathcal{S}} = O_Y\mathcal{S}$.

Certainly, $\widehat{N} \leq \min(K, N)$. If e.g., $K < N$, we aim at $\widehat{N} = K$, but we will have option that a numerical algorithm decides whether that choice is feasible, depending on the level of ill-conditioning.

The subspace selection operator $\mathcal{S}$ should ensure that, if needed, the constructed basis of $\mathcal{F}_{\widehat{N}}$ contains $\ell < \widehat{N}$ a priori selected functions $\wp_{i_1}, \ldots, \wp_{i_\ell}$ (recall the identification procedure outlined in Section 3.2) and that $O_{X,\mathcal{S}}$ is well conditioned. This implicitly restricts ℓ to be at most K, and in practice $\ell \ll K$. Furthermore, the remaining $\widehat{N} - \ell$ basis functions should be selected among the $\wp_j$'s. In other words, we seek $\mathcal{S}$ as a selection of $\widehat{N}$ columns of the identity $\mathbb{I}_N$. This is achieved by the following two step procedure:

1. (Optional) Define $\mathcal{S}_0$ as the $N \times N$ permutation matrix that moves the selected functions to the leading positions, i.e., such that $(\wp_1, \ldots, \wp_N)\mathcal{S}_0 = (\wp_{i_1}, \ldots, \wp_{i_\ell}, \ldots)$.

2. Add to these ℓ functions a selection of $\widehat{N}-\ell$ functions that are most linearly independent in the orthogonal complement of $\mathrm{span}(\wp_{i_1},\ldots,\wp_{i_\ell})$ as seen on the discrete points $\mathbf{x}_1,\ldots,\mathbf{x}_K$.

The second step, which can be designated as *basis pruning*, is based on the numerical rank revealing techniques.

7.1. Pruning the Basis $(\wp_1,\ldots,\wp_N)$

Removing the basis functions $\wp_j$ that carry numerically redundant information on the set $\mathbf{x}_1,\ldots,\mathbf{x}_K$ can be automated using the rank revealing QR factorization [23] as follows. First, compute the QR factorization of the selected functions, with an optional column pivoting

$$(O_X\mathcal{S}_0)(:,1:\ell)\Pi_1 = Q_1\begin{pmatrix}R_{11}\\ \mathbf{0}\end{pmatrix} = \begin{pmatrix}\cdot&\cdot\\ \cdot&\cdot\\ \cdot&\cdot\\ \cdot&\cdot\\ \cdot&\cdot\end{pmatrix}\begin{pmatrix}*&*\\ 0&*\\ 0&0\\ 0&0\\ 0&0\end{pmatrix}, \tag{36}$$

and then apply Q_1^* to the remaining $N-\ell$ columns to obtain

$$Q_1^*O_X\mathcal{S}_0(\Pi_1\oplus\mathbb{I}_{N-\ell}) = \left(\begin{array}{cc}R_{11}&\widetilde{R}_{12}\\ \hline \mathbf{0}&\widetilde{R}_{22}\end{array}\right) = \left(\begin{array}{ccccccc}*&*&\times&\times&\times&\times&\times\\ \hline 0&*&\times&\times&\times&\times&\times\\ 0&0&+&+&+&+&+\\ 0&0&+&+&+&+&+\\ 0&0&+&+&+&+&+\end{array}\right). \tag{37}$$

(The structure of the computed matrices is illustrated in (36), (37) for $K=5$, $N=8$, $\ell=2$.) Now, the columns of $\widetilde{R}_{22}$ are the coordinates of the projection of the trailing $N-\ell$ columns of $O_X\mathcal{S}_0$ onto the $K-\ell$ dimensional orthogonal complement of the span of the leading ℓ columns (of $O_X\mathcal{S}_0$, at this moment assumed linearly independent). A well conditioned selection of $\widehat{N}-\ell$ columns can be computed by another column pivoted (rank revealing) QR factorization

$$\widetilde{R}_{22}\Pi_2 = \begin{pmatrix}+&+&+&+&+&+\\ +&+&+&+&+&+\\ +&+&+&+&+&+\end{pmatrix}\Pi_2 = Q_2\begin{pmatrix}R_{22}&R_{23}\end{pmatrix} = Q_2\begin{pmatrix}\star&\star&\star&x&x&x\\ 0&\star&\star&x&x&x\\ 0&0&\star&x&x&x\end{pmatrix}. \tag{38}$$

Altogether, we have the factorization

$$\begin{pmatrix}\mathbb{I}_\ell&\mathbf{0}\\ \mathbf{0}&Q_2^*\end{pmatrix}Q_1^*O_X\mathcal{S}_0\begin{pmatrix}\Pi_1&\mathbf{0}\\ \mathbf{0}&\mathbb{I}_{N-\ell}\end{pmatrix}\begin{pmatrix}\mathbb{I}_\ell&\mathbf{0}\\ \mathbf{0}&\Pi_2\end{pmatrix} = \begin{pmatrix}R_{11}&R_{12}&R_{13}\\ \mathbf{0}&R_{22}&R_{23}\end{pmatrix} = \left(\begin{array}{ccccccc}*&*&\times&\times&\times&\times&\times\\ \hline 0&*&\times&\times&\times&\times&\times\\ 0&0&\star&\star&x&x&x\\ 0&0&0&\star&x&x&x\\ 0&0&0&0&\star&x&x\end{array}\right),$$

and the leading $\widehat{N}$ columns of $O_X\mathcal{S}_0(\mathbb{I}\oplus\Pi_2)$ are the desired selection. Note that in this formula we have not used the permutation $(\Pi_1\oplus\mathbb{I}_{N-\ell})$ of the first ℓ columns (here assumed independent so that R_{11} is nonsingular), to respect the requested ordering $\wp_{i_1},\ldots,\wp_{i_\ell}$ encoded in $\mathcal{S}_0$

7.2. Well Conditioned Selection by Basis Pruning—General Case

In Section 7.1 we assumed that it was indeed possible to select $\widehat{N}$ linearly independent columns from O_X. This may not be the case; moreover, the mere linear independence in finite precision numerical computation is not enough. We need well-conditioned selection of the columns of O_X, i.e., well conditioned matrix $O_{X,\mathcal{S}}$, but also well conditioned $O_{Y,\mathcal{S}}$. The rank revealing pivoting (materialized in the permutation matrices Π_1, Π_2) will provide relevant information.

If the initially selected ℓ functions are nearly linearly dependent on the supplied snapshots, but $\widehat{\ell}<\ell$ of them can be considered well conditioned, then on the diagonal of R_{11} we will see that $|(R_{11})_{\widehat{\ell}\widehat{\ell}}| > \mathtt{tol}|(R_{11})_{\widehat{\ell}+1,\widehat{\ell}+1}|$. In that case, we set $(R_{11})(\widehat{\ell}+1:\ell,\widehat{\ell}+1:\ell)=\mathbf{0}$, and the submatrix $\widetilde{R}_{22}$ is now defined as the subarray at the positions $(\widehat{\ell}+1:K,\widehat{\ell}+1:N)$. Since its first column is now zero, the pivoting Π_2 will eliminate it

from further selections (unless the entire $\widetilde{R}_{22}$ is zero). To illustrate, assume that in (37) the $(2,2)$ position carries a value ϵ that is smaller than a prescribed threshold. Then, we have

$$Q_1^* O_X \mathcal{S}_0 (\Pi_1 \oplus \mathbb{I}_{N-\ell}) = \begin{pmatrix} R_{11} & \widetilde{R}_{12} \\ \mathbf{0} & \widetilde{R}_{22} \end{pmatrix} = \begin{pmatrix} * & * & \times & \times & \times & \times & \times & \times \\ 0 & \epsilon & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + & + \end{pmatrix} \approx \begin{pmatrix} * & * & \times & \times & \times & \times & \times & \times \\ 0 & \mathbf{0} & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + & + \\ 0 & 0 & + & + & + & + & + & + \end{pmatrix}. \quad (39)$$

Altogether, this discussion can be summarized in Algorithm 3.

Algorithm 3 $[\mathcal{S}, \widehat{\ell}, \widehat{r}, \Pi_1, \Pi_2] = \text{Subspace\_Selection}(O_X, \mathcal{S}_0, \widehat{N}, \texttt{tol})$

Input: $O_X \in \mathbb{C}^{K \times N}$, $\mathcal{S}_0$, $\widehat{N}$, `tol`

1: (Optional) Reorder the snapshots by simultaneous row permutation of O_X and O_Y; see Remark 6.
2: Bring the selected functions forward to the leading ℓ positions: $Q_X = Q_X \mathcal{S}_0$. Implement $\mathcal{S}_0$ as a sequence of swaps to avoid excess data movement (in the case of large dimensions).
3: $[Q_1, R_1, \Pi_1] = \text{qr}(O_X(:, 1:\ell))$ {*Rank revealing QR factorization with column pivoting. Overwrite* $R_1 = (R_{11}^T, \mathbf{0})^T$ *over the leading* ℓ *columns of* O_X. *See (36).*}
4: Determine the numerical rank $\widehat{\ell}$ of R_{11} and in the case $\widehat{\ell} < \ell$ set $R_{11}(\widehat{\ell}+1:\ell, \widehat{\ell}+1:\ell) = \mathbf{0}$.
5: $O_X(:, \ell+1:N) = Q_1^* O_X(:, \ell+1:N)$.
6: $[Q_2, R_2, \Pi_2] = \texttt{qr}(O_X(\widehat{\ell}+1:K, \widehat{\ell}+1:N))$. {*Rank revealing QR factorization with column pivoting.* $R_2 = (R_{22}, R_{23})$ *overwrites* $O_X(\widehat{\ell}+1:K, \widehat{\ell}+1:N)$}
7: Determine the numerical rank $\widehat{r}$ of R_{22}. Set $\widetilde{N} = \widehat{\ell} + \widehat{r}$.
8: $\mathcal{S} = (\mathcal{S}_0(\Pi_1 \oplus \mathbb{I}_{N-\ell})(\mathbb{I}_{\widehat{\ell}} \oplus \Pi_2))(:, 1:\min(\widehat{N}, \widetilde{N}))$;

Output: $\mathcal{S}, \widehat{\ell}, \widehat{r}, \Pi_1, \Pi_2$.

7.3. Implementation Details

Remark 9. *Note that the case* $\widehat{\ell} < \ell$ *triggers an exception if the selected* ℓ *functions are essential in the overall computation, as for example in (17), (19), (30), (31). In the examples used in this paper* $\widehat{\ell} = \ell$, *so that no additional action is needed.*

Remark 10. *The columns of* O_X *can be severely ill-conditioned and the rank revealing QR factorization should be carefully implemented [33]. The thresholding strategy can vary from soft, mild, to hard, depending on the the concrete example, see [34]. Algorithm 3 can be used to determine the numerical rank but also to ensure that the condition number of the selected columns of* O_X *is the below specified value. This can be efficiently implemented using incremental condition estimators tailored for triangular matrices.*

Remark 11. *If the column dimension* $\min(\widehat{N}, \widetilde{N})$ *of* $\widehat{O}_X = O_X \mathcal{S}$ *and* $\widehat{O}_Y = O_Y \mathcal{S}$ *is at most K, then we can also apply the original approach from Section 2. Furthermore, the pruning scheme can be also directly applied to the original method.*

Remark 12. *Suppose that* $N \gg K$ *and that K is moderate or also big. Then computational complexity is an issue and the algorithm can be modified as follows:*

In Line 3, ℓ *is expected to be small or moderate compared to K, so that this step can be efficiently implemented using LAPACK (the functions* `xGEQRF` *and* `xGEQP3`*) or ScaLAPACK using* `PxGEQPF` *(but using the most recent implementation [35]). In the second part of the algorithm, we need a well-conditioned submatrix of a* $(K - \widehat{\ell}) \times (N - \widehat{\ell})$ *submatrix of* $Q_1^* O_X(:, \ell+1:N)$. *To do that, we do not need to compute the whole matrix. We can apply the scheme from [36] and sample the columns of* $Q_1^* O_X(:, \ell+1:N)$ *until we form a well-conditioned* R_{22}.

7.4. Numerical Experiments with the Dictionary Pruning Algorithm

Example 7. *(Continuation of Example 6) We use the same data as in Example 6, but instead of O_X and O_Y we use K-column submatrices $O_{X,\mathcal{S}}$, $O_{Y,\mathcal{S}}$, selected by Algorithm 3 with the requirement that $\wp_1, \ldots, \wp_{n+1}$ must be kept in the subspace $\mathcal{F}_K$. More precisely, we set $\widehat{N} = K$, and the numerical rank is determined with* `tol` $= 0$*, so that $\widetilde{N} = \widehat{N} = K$. The results shown in Figure 10 show a significant improvement.*

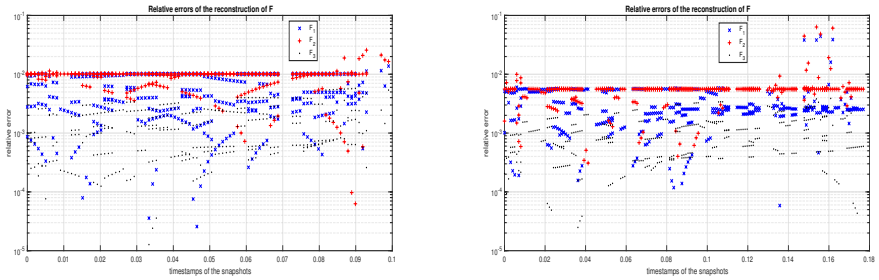

Figure 10. (Example 7.) The errors $\epsilon_k^{(i)}$ (34). **Left panel**: time interval $[0, 0.1]$. The total error is $\tau = 1.1516 \times 10^{-2}$ (1.2406×10^{-2} for `logm(Y*pinv(X))`). **Right panel**: time interval $[0, 0.18]$. The total error is $\tau = 1.7873 \times 10^{-2}$ (9.4987×10^{2} for `logm(Y*pinv(X))`). Compare this with Figure 9.

Example 8. *The purpose of this example is to illustrate the robustness of the proposed algorithm: we take the sampling interval as big as $[0, 30]$ or $[0, 50]$ with the resolution of the numerical simulation $\delta t = 0.01$ and $\delta t = 0.1$, and the samples are taken, as before, at randomly selected time instances. For illustration, the time stamps of the snapshots are marked on the first generated trajectory (with $\delta t = 0.01$) and shown in the first row in Figure 11.*

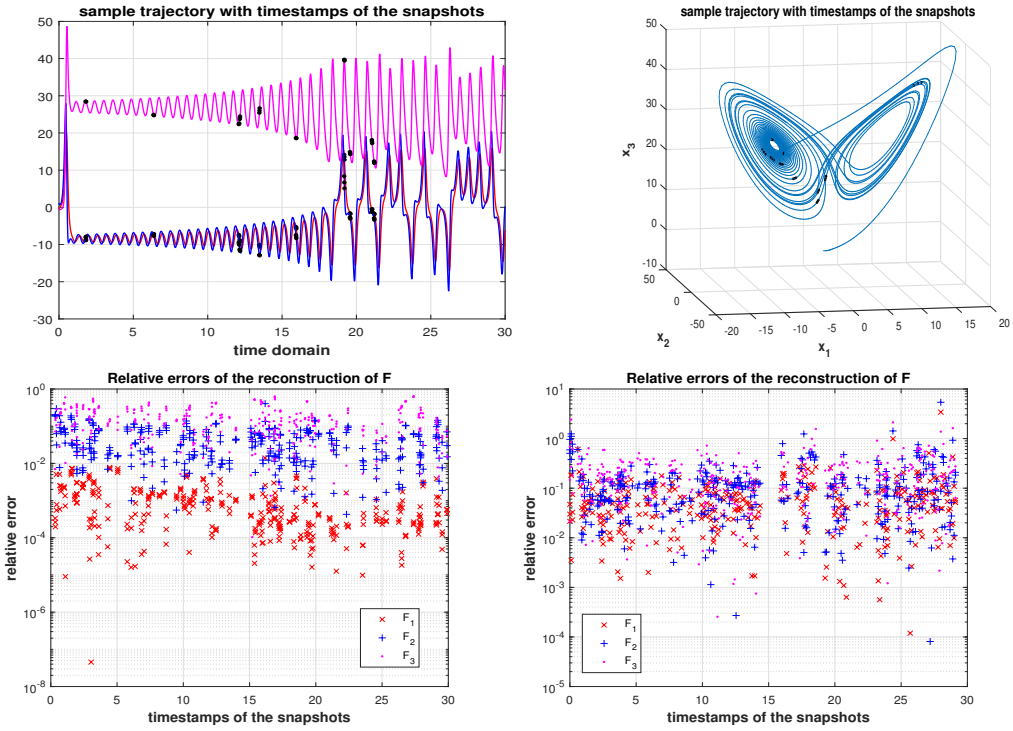

Figure 11. (Example 8.) **First row:** the time stamps of $\mathbf{x}_1, \ldots, \mathbf{x}_{360}$, illustrated on the first out of 12 generated trajectories. Three consecutive snapshots, with time lag $\delta t = 0.01$, are taken at ten randomly selected and fixed time instances. **Second row:** The first plot shows the relative errors $\epsilon_k^{(i)}$ with $\delta t = 0.01$, and the second plot for $\delta t = 0.1$ (sampled on another randomly selected times). The total errors are $\tau = 1.4893 \times 10^{-1}$ and $\tau = 8.6613 \times 10^{-1}$, respectively.

The accuracy is satisfactory, given the length of the interval ($[0,30]$), the discretization step of the simulation ($\delta t = 0.01$, $\delta t = 0.1$) and the number of samples. Next, we increase the interval to $[0,50]$; the results are in Figure 12.

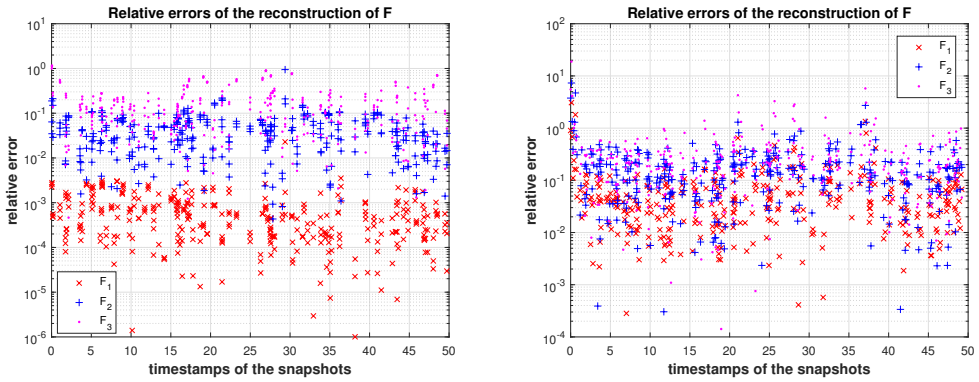

Figure 12. (Example 8.) The first plot shows the relative errors $\epsilon_k^{(i)}$ with $\delta t = 0.01$, and the second plot for $\delta t = 0.1$ (sampled on another randomly selected times in $[0,50]$). The total errors are $\tau = 1.8672 \times 10^{-1}$ and $\tau = 1.8278 \times 10^{0}$, respectively. In the second plot, we used the real parts of the computed approximations $\widetilde{F_i(\mathbf{x}_k)}$, as a non-principal (non-real) value of the logarithm was computed.

Now, we reduce the number of samples—from each trajectory we sample at five positions (instead of ten), giving the total of 180 *instead of* 360 *snapshots* $\mathbf{x}_k$*. The results are summarized in Figure 13.*

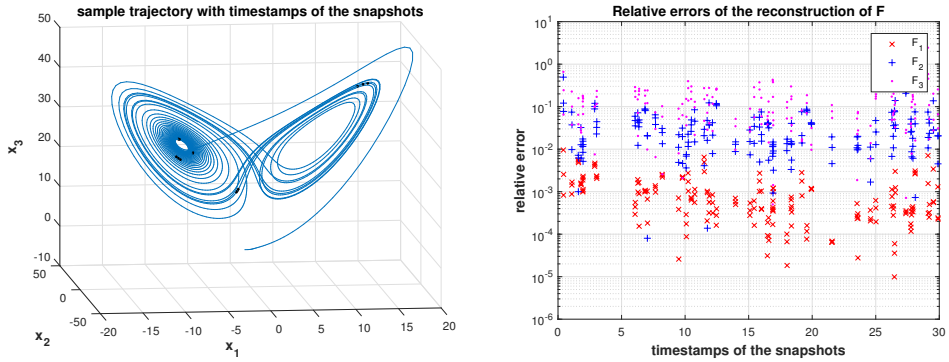

Figure 13. (Example 8.) The first plot shows the five positions at a sample trajectory where three consecutive snapshots are taken. The matrix O_X is 180×455. The reduced dimension $\widetilde{N}$ is 64. The second plot shows the relative errors $\epsilon_k^{(i)}$ with $\delta t = 0.01$. The total error is $\tau = 4.4319 \times 10^{-1}$.

In all examples, the rank revealing was conducted with a hard threshold, with no attempt in the direction of strong rank revealing, which could further improve the numerical accuracy. The details are omitted for the sake of brevity and will be available in our future work.

8. Concluding Remarks

In this work we provided the first steps towards a robust numerical software implementation of the method [8] for identification of dynamical systems using the infinitesimal generator of the Koopman semigroup. An adversarial testing using polynomial bases revealed critical numerical issues that we addressed in detail. We proposed two techniques: preconditioning and basis pruning. These are a basis for a new software implementation that will include other choices of basis functions, including data-driven/empirical constructions of the bases. This is subject of our ongoing and planned future work, including stochastic models.

Author Contributions: Conceptualization, Z.D., I.M. and R.M.; methodology, Z.D.; formal analysis, Z.D., I.M. and R.M; software, Z.D.; writing—original draft preparation, Z.D.; writing—review and editing, Z.D., I.M and R.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the DARPA contract HR0011-18-9-0033.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: The authors thank Alexandre Mauroy (University of Namur, Belgium) for his remarks and insights that helped us to improve the presentation.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Montáns, F.J.; Chinesta, F.; Gómez-Bombarelli, R.; Kutz, J.N. Data-driven modeling and learning in science and engineering. *C. R. MÉCanique* **2019**, *347*, 845–855. [CrossRef]
2. Brunton, S.L.; Proctor, J.L.; Kutz, J.N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl. Acad. Sci. USA* **2016**, *113*, 3932–3937.
Available online: https://www.pnas.org/content/113/15/3932.full.pdf (accessed on 20 June 2021). [CrossRef]
3. Chartrand, R. Numerical differentiation of noisy, nonsmooth data. *ISRN Appl. Math.* **2011**, *2011*, 164564. [CrossRef]
4. Messenger, D.A.; Bortz, D.M. Weak SINDy: Galerkin-based data-driven model selection. *arXiv* **2020**, arXiv:2005.04339.
5. Rudy, S.H.; Brunton, S.L.; Proctor, J.L.; Kutz, J.N. Data-driven discovery of partial differential equations. *Sci. Adv.* **2017**, *3*,
Available online: https://advances.sciencemag.org/content/3/4/e1602614.full.pdf (accessed on 20 June 2021). [CrossRef]
6. Goyal, P.; Benner, P. Discovery of nonlinear dynamical systems using a Runge-Kutta inspired dictionary-based sparse regression approach. *arXiv* **2021**, arXiv:2105.04869.
7. Raissi, M.; Perdikaris, P.; Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
8. Mauroy, A.; Goncalves, J. Koopman-based lifting techniques for nonlinear systems identification. *IEEE Trans. Autom. Control.* **2020**, *65*, 2550–2565. [CrossRef]
9. Engel, K.J.; Nagel, R. *One-Parameter Semigroups for Linear Evolution Equations*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 1999; Volume 194.
10. Budišić, M.; Mohr, R.; Mezi'c, I. Applied Koopmanism. *Chaos Interdiscip. J. Nonlinear Sci.* **2012**, *22*, 047510. [CrossRef]
11. Lasota, A.; Mackey, M.C. *Chaos, Fractals, and Noise: Stochastic Aspects of Dynamics*; Springer: Berlin/Heidelberg, Germany, 1994.
12. Mohr, R.; Drmač, Z.; Mezić, I.; Fonoberova, M. *Composition Operator for Static Data*; AIMDyn Tech Report; AIMDyn Inc.: Santa Barbara, CA, USA, 2019.
13. Niemann, J.H.; Klus, S.; Schütte, C. Data-driven model reduction of agent-based systems using the Koopman generator. *PLoS ONE* **2021**, *16*, e0250970. [CrossRef]
14. Klus, S.; Nüske, F.; Peitz, S.; Niemann, J.H.; Clementi, C.; Schütte, C. Data-driven approximation of the Koopman generator: Model reduction, system identification, and control. *Phys. D Nonlinear Phenom.* **2020**, *406*, 132416. [CrossRef]
15. Metzner, P.; Horenko, I.; Schütte, C. Generator estimation of Markov jump processes based on incomplete observations nonequidistant in time. *Phys. Rev. E* **2007**, *76*, 066702. [CrossRef] [PubMed]
16. Siefert, M.; Kittel, A.; Friedrich, R.; Peinke, J. On a quantitative method to analyze dynamical and measurement noise. *EPL (Europhys. Lett.)* **2007**, *61*, 466. [CrossRef]
17. Friedrich, R.; Peinke, J.; Sahimi, M.; Reza Rahimi Tabar, M. Approaching complexity by stochastic methods: From biological systems to turbulence. *Phys. Rep.* **2011**, *506*, 87–162. [CrossRef]
18. Callaham, J.L.; Loiseau, J.C.; Rigas, G.; Brunton, S.L. Nonlinear stochastic modelling with Langevin regression. *Proc. R. Soc. A Math. Phys. Eng. Sci.* **2021**, *477*, 20210092. [CrossRef]
19. Riseth, A.N.; Taylor-King, J.P. Operator fitting for parameter estimation of stochastic differential equations. *arXiv* **2017**, arXiv:1709.05153.
20. Björck, A. *Numerical Methods in Matrix Computations*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 59. [CrossRef]
21. Williams, M.O.; Kevrekidis, I.G.; Rowley, C.W. A data–driven approximation of the Koopman operator: Extending Dynamic Mode Decomposition. *J. Nonlinear Sci.* **2015**, *25*, 1307–1346. [CrossRef]
22. Mezić, I.; Drmač, Z.; Črnjarić Žic, N.; Maćešić, S.; Fonoberova, M.; Mohr, R.; Avila, A.M.; Manojlović, I.; Andrejčuk, A. A Koopman operator-based prediction algorithm and its application to COVID-19 pandemic. *Nat. Commun.* submitted.
23. Businger, P.A.; Golub, G.H. Linear least squares solutions by Householder transformations. *Numer. Math.* **1965**, *7*, 269–276. [CrossRef]
24. Culver, W.J. On the existence and uniqueness of the real logarithm of a matrix. *Proc. Am. Math. Soc.* **1966**, *17*, 1146–1151. [CrossRef]

25. Higham, N.J. *Functions of Matrices: Theory and Computation*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2008; pp. xx+425.
26. Kinyon, M.; Sagle, A. Quadratic dynamical systems and algebras. *J. Differ. Equations* **1995**, *117*, 67–126. [CrossRef]
27. Demmel, J.W. The geometry of ill-conditioning. *J. Complex.* **1987**, *3*, 201–229. doi: 10.1016/0885-064X(87)90027-6. [CrossRef]
28. Rust, B.W. *Truncating the Singular Value Decomposition for ill–Posed Problems*; Technical Report NISTIR 6131; Mathematical and Computational Sciences Division, National Institue of Standards and Technology, U.S. Department of Commerce, NIST: Gaithersburg, MD, USA, 1998.
29. Drmač, Z.; Mezić, I.; Mohr, R. Data driven modal decompositions: Analysis and enhancements. *SIAM J. Sci. Comput.* **2018**, *40*, A2253–A2285. [CrossRef]
30. Chandrasekaran, S.; Ipsen, I.C.F. On rank–revealing factorizations. *SIAM J. Matrix Anal. Appl.* **1994**, *15*, 592–622. [CrossRef]
31. Powell, M.J.D.; Reid, J.K. On applying Householder transformations to linear least squares problems. In *Information Processing 68, Proceedings of the International Federation of Information Processing Congress, Edinburgh, UK, 5–10 August 1968*; IFIP: Amsterdam, The Netherlands, 1969; pp. 122–126.
32. Cox, A.J.; Higham, N.J. Stability of Householder QR factorization for weighted least squares problems. In *Numerical Analysis 1997, Proceedings of the 17th Dundee Biennial Conference, Dundee, UK, 24–27 June 1997*; Griffiths, D.F., Higham, D.J., Watson, G.A., Eds.; Pitman Research Notes in Mathematics; Pitman: London, UK, 1998; Volume 380, pp. 57–73.
33. Drmač, Z.; Bujanović, Z. On the failure of rank revealing QR factorization software—A case study. *ACM Trans. Math. Softw.* **2008**, *35*, 1–28. [CrossRef]
34. Drmač, Z.; Šain Glibić, I. New numerical algorithm for deflation of infinite and zero eigenvalues and full solution of quadratic eigenvalue problems. *arXiv* **2019**, arXiv:1904.05418.
35. Bujanović, Z.; Drmač, Z. New robust ScaLAPACK routine for computing the QR factorization with column pivoting. *arXiv* **2019**, arXiv:1910.05623.
36. Drmač, Z.; Gugercin, S. A new selection operator for the iscrete empirical interpolation method—improved a priori error bound and extensions. *SIAM J. Sci. Comput.* **2016**, *38*, A631–A648. [CrossRef]