Adv Comput Math (2018) 44:1751–1768
https://doi.org/10.1007/s10444-018-9592-x

On reduced input-output dynamic mode decomposition

Peter Benner[1] · **Christian Himpe**[1] · **Tim Mitchell**[1]

Received: 15 July 2017 / Accepted: 25 January 2018 /
Published online: 27 February 2018

Abstract The identification of reduced-order models from high-dimensional data is a challenging task, and even more so if the identified system should not only be suitable for a certain data set, but generally approximate the input-output behavior of the data source. In this work, we consider the input-output dynamic mode decomposition method for system identification. We compare excitation approaches for the data-driven identification process and describe an optimization-based stabilization strategy for the identified systems.

Keywords Dynamic mode decomposition · Model reduction · System identification · Cross Gramian · Optimization

Mathematics Subject Classification (2010) 93B30 · 90C99

Communicated by: Karsten Urban

Supported by the German Federal Ministry for Economic Affairs and Energy (BMWi), in the joint project: "MathEnergy – Mathematical Key Technologies for Evolving Energy Grids", sub-project: Model Order Reduction (Grant number: 0324019**B**).

✉ Christian Himpe
himpe@mpi-magdeburg.mpg.de

Peter Benner
benner@mpi-magdeburg.mpg.de

Tim Mitchell
mitchell@mpi-magdeburg.mpg.de

[1] Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstr. 1, 39106, Magdeburg, Germany

1 Introduction

In various applications, it can be difficult (sometimes even impossible) to derive models from first principles. However, the input-output response of a system and data of the inner state may still be available; we refer to such a setup as a *graybox*. For example, a natural process whose underlying action is not well understood can be considered a graybox since we may only be able to measure its behavior. In applications, such as manufacturing and design, it may be necessary to model a third-party provided subcomponent whose exact or full specifications may not be obtainable due to it containing proprietary information. In order to gain insight into such natural or technical processes, and derive models that simulate and/or predict their behaviors, it is often beneficial and perhaps necessary to create models using measured or generated data. The discipline of system identification investigates methods for the task of obtaining such models from data. One class of methods for data-driven identification is dynamic mode decomposition (DMD), which also provides a modal analysis of the resulting systems. In this work, we investigate variants of DMD for the class of input-output systems and compare data sampling strategies.

DMD has its roots in the modal decomposition of the Koopman operator[1] [19], which has been used, for example, in the spectral analysis of fluid dynamics [26]. The basic DMD method was introduced in [27], and various extensions have been added, such as Exact DMD [28] or Extended DMD [9]. Furthermore, DMD can also be used as a tool for model order reduction: [25] proposed using DMD for flow analysis and control while DMD has also been combined with Galerkin-projection techniques to model nonlinear systems [1]. For a comprehensive survey of DMD and its variants, see [16].

Our work here builds upon two specific variants of DMD. The first is Dynamic Mode Decomposition with Control (DMDc) [23], and thus by relation, also Koopman with Inputs and Control (KIC) [24]. The second is Input-Output Dynamic Mode Decomposition (ioDMD) [3, 4], which itself is closely related to the Direct Numerical Algorithm for Sub-Space State System Identification (N4SID) [31]. DMDc extends DMD to scenarios where the input of the discrete system approximation is given by a functional while ioDMD additionally handles the case when the system's output is specified and also a functional.

To generically identify a system without prior knowledge on the relevant input functions, techniques such as persistent excitation [6] have been well known for some time now. We propose an extension to the ioDMD method of [3] with a new excitation variant related to the cross Gramian matrix [11]. Additionally, since DMD-based identification does not guarantee that its resulting models are stable, we propose a post-processing procedure to stabilize ioDMD-derived models, by employing the nonsmooth constrained optimization method of [10] to solve a corresponding stabilization problem.

This document is structured as follows. In Section 2, an overview of the prerequisite dynamic mode decomposition method and its relevant variants is given. Section 3

[1] The Koopman operator [15] itself is an infinite-dimensional linear operator that describes discrete dynamics of nonlinear evolution equations.

describes the new excitation strategy while our optimization-based stabilization procedure is discussed in Section 4. Finally, two numerical experiments are conducted in Section 5.

2 Dynamic mode decomposition

Consider the time-invariant ordinary differential equation (ODE):

$$\dot{x}(t) = f(x(t)), \tag{1}$$

with state $x(t) \in \mathbb{R}^N$ and vector field $f : \mathbb{R}^N \to \mathbb{R}^N$. Furthermore, consider for now that (1) is sampled at uniform intervals for times $t_0, \ldots, t_K$. The most basic version of DMD [16, 27] aims to approximate (1) by constructing a discrete-time linear system

$$x_{k+1} = Ax_k, \tag{2}$$

with a linear operator $A \in \mathbb{R}^{N\times N}$, such that if $x_k = x(t_k)$, then $x_{k+1} \approx x(t_{k+1})$ should also hold, for all $k = 0, \ldots, K-1$.

Starting at an initial state x_0, the sequence defined by (2) corresponds to a trajectory of the state vectors x_k. The corresponding data matrix $X \in \mathbb{R}^{N\times K}$ is simply the in-order concatenation of these state vectors, that is,

$$X = \begin{bmatrix} x_0 & x_1 & \ldots & x_K \end{bmatrix}.$$

By forming the following two partial trajectories:

$$X_0 = \begin{bmatrix} x_0 & x_1 & \ldots & x_{K-1} \end{bmatrix} \quad \text{and}$$
$$X_1 = \begin{bmatrix} x_1 & x_2 & \ldots & x_K \end{bmatrix},$$

where $X_0 \in \mathbb{R}^{N\times K-1}$ is just the data matrix X with its last data point removed while $X_1 \in \mathbb{R}^{N\times K-1}$ is just X with its first data point removed, the idea behind (plain) DMD [27] is to then solve the linear system of equations:

$$X_1 = AX_0,$$

which is in fact just (2), where x_{k+1} and x_k have been replaced by X_1 and X_0, respectively. A best-fit solution to this problem is given by the pseudoinverse:

$$A \approx X_1 X_0^+,$$

which is also the solution to minimizing the least-squares error in the Frobenius norm ($\|\cdot\|_{\mathrm{F}}$):

$$A = \underset{\tilde{A}\in\mathbb{R}^{N\times N}}{\arg\min}(\|X_1 - \tilde{A}X_0\|_{\mathrm{F}}).$$

The DMD modes of (1) are given by the eigenvectors of the matrix A:

$$A\Lambda = \Lambda V.$$

Beyond just using a single uniformly-sampled time series, DMD has also been generalized to a method called Exact DMD [28], which can additionally support the concatenation of multiple different time series and/or non-uniform time steppings. This generalization of DMD is made possible by reducing the requirements of X_0

and X_1 to the lesser condition that their columns need only be composed in pairs of data such that $X_1(:,k) = f(X_0(:,k))$ is fulfilled.

2.1 Practical computation

We now give a high-level algorithmic description of DMD identification. The pseudoinverse of the data matrix can be computed using the (rank-revealing) singular value decomposition (SVD), $X_0 = \mathcal{U}\Sigma\mathcal{V}^*$, as follows:

$$X_0^+ = \mathcal{V}\Sigma^{-1}\mathcal{U}^*.$$

However, inverting tiny but nonzero singular values in the *computed* diagonal matrix Σ poses a numerical problem, as these small singular values may be inaccurate. Applying Σ^{-1} could *overamplify* components of X_0, in particular, the less important ones. To counteract this effect, computing the pseudoinverse via the SVD is done in practice by truncating any singular values that are smaller than some fixed $\varepsilon \in \mathbb{R}^+$, which is equivalent to adding a regularization term to the least-squares solution for A:

$$A = \operatorname*{arg\,min}_{\tilde{A}\in\mathbb{R}^{N\times N}} \|X_1 - \tilde{A}X_0\|_{\mathrm{F}}^2 + \beta\|\tilde{A}\|_{\mathrm{F}}^2,$$

for some value of the regularization parameter $\beta \in \mathbb{R}^+$. Following [28], the algorithm for the DMD-based computation of the system matrix A, given a state-space trajectory X and a lower bound $\varepsilon > 0$ for discarding tiny singular values, is as follows:

1. Sample the state-space trajectory and form data matrix $X = \begin{bmatrix} x_0 & \dots & x_K \end{bmatrix}$.
2. Assemble the shifted partitions X_0 and X_1.
3. Compute the SVD of $X_0 = \mathcal{U}\Sigma\mathcal{V}^*$.
4. Truncate Σ: $\widetilde{\Sigma}_{ii} = \Sigma_{ii}$ if $\Sigma_{ii} \geq \varepsilon$; 0 otherwise.
5. Identify the approximate discrete system matrix: $A := \mathcal{U}^* X_1 \mathcal{V}\widetilde{\Sigma}^{-1}$.

In this DMD variant, the order (dimension) of the computed matrix A is equal to the number of retained (nontruncated) singular values, but this truncation is done solely for numerical accuracy; the intention is to keep as many components of the dynamics as possible. In contrast, model order reduction typically aims to *significantly* reduce the system down to just a small set of *essential* dynamics, and accomplishing this goal will be the focus of Section 2.4.

2.2 Dynamic mode decomposition with control

Considering systems whose vector field f depends not just on the state but also on an input function $u : \mathbb{R} \to \mathbb{R}^M$:

$$\dot{x}(t) = f(x(t), u(t)), \tag{3}$$

has led to a DMD variant called Dynamic Mode Decomposition with Control (DMDc) [23]. We focus on a specific DMDc variant [23, Sec. 3.3] that aims to approximate (3) by a linear discrete-time control system:

$$x_{k+1} = Ax_k + Bu_k, \tag{4}$$

where $B \in \mathbb{R}^{N\times M}$ (called the input operator) must also be identified in addition to A and input $u_k = u(t_k)$ is a discretely-sampled version of the continuous input function $u(t)$ for some sampling times given by t_k. In addition to forming X and X_0 as in plain DMD (Section 2), an analogous construction of matrices for input data is also done. An in-order concatenation of the series of input data u_k vectors is done to obtain matrix $U \in \mathbb{R}^{M\times K}$ while the partial data matrix $U_0 \in \mathbb{R}^{M\times K-1}$ is simply U without its last column (the last input sample):

$$\begin{aligned} U &= \begin{bmatrix} u_0 & u_1 & \dots & u_{K-1} & u_k \end{bmatrix} \quad \text{and} \\ U_0 &= \begin{bmatrix} u_0 & u_1 & \dots & u_{K-1} \end{bmatrix}. \end{aligned}$$

Then, A and B can be obtained by computing the approximate solutions to the linear system of equations given by:

$$X_1 = AX_0 + BU_0 = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} X_0 \\ U_0 \end{bmatrix},$$

which is (4) with u_k replaced by U, x_k by X_0, and x_{k+1} by X_1, and has solution:

$$\begin{bmatrix} A & B \end{bmatrix} = X_1 \begin{bmatrix} X_0 \\ U_0 \end{bmatrix}^{+}.$$

As mentioned in Section 1, DMDc is actually a special case of the KIC method [24]. For KIC, the state of the system is also augmented with the discretized input u_k, which leads the resulting augmented system to have additional operators:

$$\begin{bmatrix} x_{k+1} \\ u_{k+1} \end{bmatrix} = \begin{bmatrix} A & B \\ B_u & A_u \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix},$$

where $B_u \in \mathbb{R}^{M\times N}$ and $A_u \in \mathbb{R}^{M\times M}$. Of course, these two additional operators must now also be identified along with matrices A and B. However, if one assumes that the input is known and no identification of the associated operators is required, then B_u and A_u are just zero matrices, and it is thus clear that KIC is a generalization of DMDc.

2.3 Input-Output dynamic mode decomposition

Extending the underlying system once more to now also include an output function $y : \mathbb{R} \to \mathbb{R}^Q$, with an associated output functional $g : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^Q$ that also depends on the state x and the input u, yields the following system:

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)), \\ y(t) &= g(x(t), u(t)). \end{aligned} \tag{5}$$

Data-based modeling of systems of the form given by (5) has given rise to a class of DMD methods called Input-Output Dynamic Mode Decomposition (ioDMD) [3]. Similar to previously discussed DMD variants, the ioDMD method also approximates the original system by a discrete-time linear system, but now with input and output:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k, \\ y_k &= Cx_k + Du_k, \end{aligned} \tag{6}$$

where $C \in \mathbb{R}^{Q\times N}$ and $D \in \mathbb{R}^{Q\times M}$ are the output and feed-through operators, respectively. Since this approximation includes output data, the discrete output instances $y_k = y(t_k) \in \mathbb{R}^Q$ are also correspondingly arranged into a matrix $Y \in \mathbb{R}^{Q\times K}$ by in-order concatenation while $Y_0 \in \mathbb{R}^{Q\times K-1}$ simply omits the last column (output sample) of Y:

$$Y = \begin{bmatrix} y_0 & y_1 & \dots & y_{K-1} & y_K \end{bmatrix} \quad \text{and}$$
$$Y_0 = \begin{bmatrix} y_0 & y_1 & \dots & y_{K-1} \end{bmatrix}.$$

Matrices A, B, C, and D are then approximated by solving:

$$\begin{cases} X_1 = AX_0 + BU_0 \\ Y_0 = CX_0 + DU_0 \end{cases} \quad \text{or equivalently} \quad \begin{bmatrix} X_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} X_0 \\ U_0 \end{bmatrix}, \tag{7}$$

which is (6) but where u_k, x_k, x_{k+1}, and y_{k+1} have been replaced by U_0, X_0, X_1 and Y_0, respectively, and has the solution:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} X_1 \\ Y_0 \end{bmatrix} \begin{bmatrix} X_0 \\ U_0 \end{bmatrix}^+. \tag{8}$$

This procedure is equivalent to the Direct N4SID algorithm [14; 30, Ch. 6.6] mentioned in Section 1.

Note that all the DMD variants discussed so far identify the original continuous-time systems by *linear* discrete-time models. However, one can create a continuous-time model given by $\{\widehat{A}, \widehat{B}, \widehat{C}, \widehat{D}\}$, that is a first-order approximation to the discrete-time model obtained by ioDMD, using for example, the standard first-order Euler method:

$$\begin{aligned} x_{k+1} &= x_k + h(\widehat{A}x_k + \widehat{B}u_k) \\ \Rightarrow Ax_k + Bu_k &= x_k + h\widehat{A}x_k + h\widehat{B}u_k \\ \Rightarrow \begin{bmatrix} \widehat{A} & \widehat{B} \end{bmatrix} &= \begin{bmatrix} h(I - A) & hB \end{bmatrix}. \end{aligned}$$

The output and feed-through operators for the continuous-time model remain the same as in the discrete-time model produced by ioDMD, that is, $\widehat{C} = C$, $\widehat{D} = D$. Lastly, we stress that (io)DMD derived models are generally only valid for the time horizon over which the data has been gathered.

2.4 Reduced order DMD

To accelerate the computation of ioDMD-derived models, we follow [4, 7] and reduce the order of the possibly high-dimensional state trajectories using a projection-based approach. The data matrices X_0 and X_1 are compressed using a truncated (Galerkin) projection $Q \in \mathbb{R}^{N\times n}$, $n \ll N$, $Q^*Q = I$:

$$\begin{bmatrix} Q^*X_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} A_r & B_r \\ C_r & D_r \end{bmatrix} \begin{bmatrix} Q^*X_0 \\ U_0 \end{bmatrix}$$
$$\begin{bmatrix} Q^*X_1 \\ Y_0 \end{bmatrix} \begin{bmatrix} Q^*X_0 \\ U_0 \end{bmatrix}^+ = \begin{bmatrix} A_r & B_r \\ C_r & D_r \end{bmatrix}.$$

The order of the identified system is thus determined by the rank of the projection.

A popular method to compute such a truncating projection Q is proper orthogonal decomposition (POD) [12], which is practically obtained as the left singular vectors (POD modes) of the data matrix X:

$$X \overset{\text{SVD}}{=} UDV^* \to X_r = U_1^\intercal X \approx X.$$

The number of resulting computed POD modes n depends on the desired projection error $\|X - UU^*X\|_{\text{F}} \leq (\sum_{i=1}^{n} D_{ii}^2)^{1/2}$, which is a consequence of the Schmidt-Eckhard-Young-Mirsky Lemma (see for example [5]).

Note again that this data compression scheme has a very different motivation compared to that of the aforementioned dimension-reduction done when computing the pseudoinverse via the truncated SVD (discussed in Section 2.1). The latter truncation, based on the magnitude of the singular values, is done for reasons of numerical accuracy when computing the pseudoinverse (and applying it in subsequent computations). The projection-based truncation discussed here, using the sum of the singular values squared, allows for the possibility of a much less onerous computational burden, as the state space of the models can often be greatly reduced by discarding all but a handful of essential modes in order to obtain a desired approximation error. Other projection-based data-driven model reduction techniques for the compression of the state trajectory are similarly applicable, for example empirical balanced truncation [17].

3 Training data and generic identification

DMD is a data-driven method, hence, the source of the data used for the system identification needs to be considered. Usually it is possible to identify an input-output system (for provided discrete input, state, and output functions) so that the identified system produces similar outputs given the input used for the identification. To identify a model from data, the associated system needs to produce outputs approximating the data source, not only for specific data sets, but for a whole class of relevant input functions and perhaps even arbitrarily admissible ones. For such generic linear system identification, the matrices A, B, C, D have to be computed in such a manner that (a) does not require a specific data set and (b) allows behaviors of the system being modeled to be predicted for a given initial condition and input function. This can be accomplished, for example, by persistent excitation (PE) [6, 23], which utilizes signals such as step functions or Gaussian noise as training input data. Here, we propose a related approach that also exploits random (Gaussian) sampling, yet is based on the cross operator.

3.1 Cross excitation

The cross operator [13] $W_X : \mathbb{R}^N \to \mathbb{R}^N$ is a tool for balancing-type model reduction and encodes the input-output coherence of an associated system, which for linear time-invariant systems is the so-called cross Gramian matrix [11]. This operator is

defined as the composition of the controllability operator $\mathcal{C} : L_2 \to \mathbb{R}^N$ with the observability operator $\mathcal{O} : \mathbb{R}^N \to L_2$:

$$W_X = \mathcal{C} \circ \mathcal{O}.$$

Thus, for a square system with the same input and output space dimension, W_X maps a given initial state x_0 via the observability operator to an output function, and is in turn passed to the controllability operator as an input function resulting in a final state[2]:

$$x_0 \overset{\mathcal{O}}{\mapsto} y \overset{\mathcal{C}}{\mapsto} x(T).$$

To generate trajectory data, we modify the cross operator by replacing the controllability operator with the input-to-state map $\xi : L_2 \to L_2$ [5, Ch. 4.3]. This yields an operator $\mathcal{W}_X : \mathbb{R}^N \to L_2$:

$$x_0 \overset{\mathcal{O}}{\mapsto} y \overset{\xi}{\mapsto} x,$$

which maps, as before, an initial state to an output function, but then maps this output (as an input) function to a state trajectory (instead of to a final state). Compared to PE, the cross(-operator-based) excitation (CE) is a two-stage procedure using perturbations of the initial state to generate the excitation, as opposed to perturbing the input directly.

The cross excitation is related to indirect identification of closed-loop systems [22, 29], which is also a two-stage process. First, an intermediary system (open-loop) system is identified, which then is used in the second step to generate a signal that acts as an excitation for the identification of the actual closed-loop system. A distinct difference of CE compared to the indirect closed-loop identification approach is that the latter exclusively uses input-output data while the former also uses state trajectory data in addition to the input-output data.

4 Stabilization

As DMD and its variants are time-domain methods (ioDMD included), it is typically desired to preserve stability in the (reduced) identified discrete-time systems. However, models derived by ioDMD are not guaranteed to be stable. To enforce stability, an additional post-processing step is required. For example, [2] proposed stabilizing models derived using Petrov-Galerkin projections by solving a sequence of semidefinite programs. In this paper, we take a much more direct approach.

A square matrix A is discrete-time stable if its spectral radius is less than one, that is, $\rho(A) < 1$, where

$$\rho(A) \coloneqq \max\{|\lambda| : \lambda \in \Lambda(A)\}.$$

Although the spectral radius is nonconvex, it is a continuous function with respect to the matrix A and furthermore, it is continuously differentiable *almost everywhere* (in the mathematical sense). In other words, the set of points where the spectral radius is

[2]This is also illustrated in [13, Fig. 1]

nonsmooth only has measure 0 and so it holds that points chosen randomly will, with probability 1, be outside of this set. Hence, despite the nonsmoothness of the spectral radius, it is still possible to attain a wealth of information from its gradient, since it is defined on all but a subset of measure 0 in the full space. Thus if the matrix A from the ioDMD-derived model, that is, from (8), is not stable, we could consider employing a gradient-based optimization method to stabilize it, while hopefully ensuring that the resulting stabilized version of the ioDMD solution still models the original large-scale system. In order to meet these two objectives, we consider solving the following constrained optimization problem:

$$\begin{bmatrix} A_\mathrm{s} & B_\mathrm{s} \\ C_\mathrm{s} & D_\mathrm{s} \end{bmatrix} = \underset{\tilde{A},\tilde{B},\tilde{C},\tilde{D}}{\arg\min} \left\| \begin{bmatrix} X_1 \\ Y_0 \end{bmatrix} - \begin{bmatrix} \tilde{A} & \tilde{B} \\ \tilde{C} & \tilde{D} \end{bmatrix} \begin{bmatrix} X_0 \\ U_0 \end{bmatrix} \right\|_\mathrm{F}^2 \quad \text{s.t.} \quad |\lambda(\tilde{A})| < 1 - \tau, \quad (9)$$

where $\tilde{A} \in \mathbb{R}^{r\times r}$, $\tilde{B} \in \mathbb{R}^{r\times m}$, $\tilde{C} \in \mathbb{R}^{p\times r}$, $\tilde{D} \in \mathbb{R}^{p\times m}$, and $\tau \geq 0$ is a margin for the stability tolerance. As the unstabilized ioDMD model is already a solution to (7), solving (9) should promote solutions that are still close to the original ioDMD model while simultaneously enforcing the requirement that these models must now be stable, due to the presence of the stability radius in the inequality constraint. Furthermore, the unstabilized ioDMD model should make a good point to start the optimization method at.

There are however some difficulties with solving (9) iteratively via optimization techniques. The first is that the objective function of (9) is typically underdetermined in DMD settings, which can adversely impact a method's usual rate of convergence, as minimizers are no longer locally unique. However, as our goal is mainly to stabilize the ioDMD model, without changing its other properties too much, we do not *necessarily* need to solve (9) exactly. A few iterations may be all that is needed to accomplish this task.

As an alternative, one could consider solving

$$\min_{\tilde{A},\tilde{B},\tilde{C},\tilde{D}} \left\| \begin{bmatrix} A_\mathrm{DMD} & B_\mathrm{DMD} \\ C_\mathrm{DMD} & D_\mathrm{DMD} \end{bmatrix} - \begin{bmatrix} \tilde{A} & \tilde{B} \\ \tilde{C} & \tilde{D} \end{bmatrix} \right\|_\mathrm{F}^2 \quad \text{s.t.} \quad |\lambda(\tilde{A})| < 1 - \tau$$

in lieu of (9), where A_DMD, B_DMD, C_DMD, and D_DMD are the matrices produced by ioDMD. On the upside, this helps to avoid the problem of underdeterminedness arising in (9) and encourages that a stable solution close to the original ioDMD-derived system is found. However, this modified objective no longer takes any observed data into account. We did evaluate this alternate optimization problem in our experiments, but the performance of the models it produced was sometimes worse. As such, we will only report results for our experiments done using (9) in Section 5.

The second issue for trying to solve (9) is that the spectral radius can be a rather difficult function to optimize, relatively speaking. First, despite being continuously differentiable almost everywhere, the spectral radius is still a nonsmooth function, specifically at matrices which have multiple eigenvalues that attain the maximum modulus, that is, the value of the spectral radius. Typically, minimizers of the spectral radius will be at such matrices (for example, see the plots of spectral configurations post optimization in [10, Section 4.1 and Appendix 9.1]), so optimizing the spectral radius often means that an optimization method must try to converge to a nonsmooth

minimizer, a difficult prospect. Worse still is that the spectral radius is also non-locally Lipschitz at matrices where these multiple eigenvalues attaining the value of the spectral radius coincide (see [8]). Many of the available continuous optimization methods are designed under the assumption that functions they optimize are smooth or if not, at least locally Lipschitz. If the functions being optimized do not meet these criteria, these methods typically break down. Furthermore, the nonconvexity of the spectral radius means that optimization codes may get stuck in local minima that may or may not provide sufficiently acceptable solutions.

Although the inclusion of the spectral radius constraint makes (9) a nonsmooth, nonconvex optimization problem, with a non-locally-Lipschitz constraint function, again we do not necessarily need to solve it exactly (though this will remain to be seen in our experiments). Furthermore, while much of the literature on nonsmooth optimization has historically focused on unconstrained problems, there has also been recent interest in addressing problems with nonsmooth constraints. For example, a new algorithm combining quasi-Newton BFGS (Broyden-Fletcher-Goldfarb-Shanno) updating and SQP (sequential quadratic programming)[3] was recently proposed for general nonsmooth, nonconvex, constrained optimization problems [10], where no special knowledge or structure is assumed about the objective or constraint functions. Particularly relevant to our approach here, this BFGS-SQP method was evaluated on 100 different spectral radius constrained optimization problems, with promising results relative to other solvers [10, Section 6]. This indicates that it may also be a good solver for our nonsmooth constrained optimization problem and thus we propose using it to solve (9). Specifically, we use GRANSO: GRadient-based Algorithm Non-Smooth Optimization [20], an open-source MATLAB code implementing the aforementioned BFGS-SQP method of [10].

5 Numerical results

We implemented our new ioDMD variant using both PE and CE sampling strategies (presented in Section 3.1) to collect observations of the original system's behaviors. Furthermore, our software can also optionally stabilize the resulting ioDMD-derived models by using GRANSO [10] on our associated nonsmooth constrained optimization problem.

As discussed in Section 4, (9) is an underdetermined optimization problem in DMD settings. Since such problems are in a sense quite flat, the norm of the gradient merely being small can be a poor measure of when to terminate; correspondingly, we tightened GRANSO's default termination tolerance of 10^{-6} to 10^{-8} (i.e. `opts.opt_tol = 1e-8`). Relatedly, convergence can also be slow so the choice of a starting point can also be critical. As our goal, specified by (9), is to stabilize a model while minimizing the tradeoff in any increased approximation error (that may occur due to stabilization), we simply used the unstable ioDMD-derived models as starting points for GRANSO and used GRANSO's custom termination feature

[3] For a good introductory reference on many of the optimization techniques referred to in this paper, see [21].

to halt optimization once a model had been found that was both stable (for (9) we used $\tau := 0$) and had an objective value that was less than 1000 times the objective function evaluated at the original ioDMD-derived unstable model. We found that this loose limit on how much the objective function was allowed to increase was more than adequate to retain good output errors. We ran GRANSO using its full-memory BFGS mode (its default behavior and the recommended choice for nonsmooth problems) and kept all other GRANSO options at their default values as well.

The numerical experiments were implemented in the Matlab language and were run under MATLAB R2017a on a workstation computer with an Intel Core i7-6700 (4 Cores @ 3.4 GHz) and 8 GB memory.

5.1 Excitation and stabilization evaluation

We demonstrate the effects of different types of excitation used for the ioDMD-based system identification by a numerical example. Specifically, for a given target data set, we identify a discrete-time linear system first using the target data itself, second by persistent excitation (PE) and third by utilizing the herein proposed cross excitation (CE) from Section 3.1.

The considered input-output system is based on the (one-dimensional) transport equation, with the left boundary of the domain selected as input and the right boundary as output:

$$\begin{aligned} \frac{\partial}{\partial t} z(x,t) &= a \frac{\partial}{\partial x} z(x,t), \quad x \in [0,1], \\ z(0,t) &= u(t), \\ z(x,0) &= 0, \\ y(t) &= z(1,t). \end{aligned}$$

The partial differential equation is discretized in space using the first-order finite-difference upwind scheme, with a spatial resolution of $\Delta x = \frac{1}{1000}$ and $a = 1.3$. The resulting ODE input-output system is then given by:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t). \end{aligned} \tag{10}$$

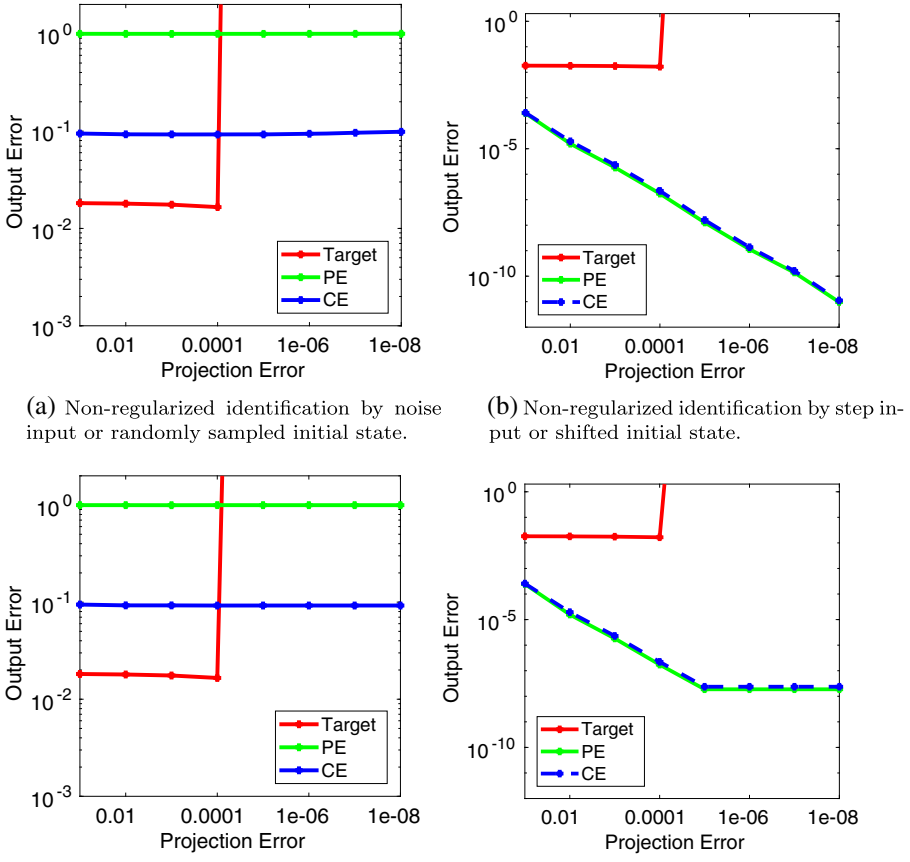
The target data is given by discrete input, state and output functions from a simulation, which is performed by a first order implicit Runge-Kutta method. We used a time-step width of $\Delta t = \frac{1}{1000}$ and a time horizon of $T = 1$, with a Gaussian-bell type input function $\hat{u}(t) = e^{-\frac{1}{1000}(t-\frac{1}{10})^2}$.

For the comparison, a discrete-time linear system was first identified from the 1000 snapshots generated by a simulation of (10) excited by input $\hat{u}$, to obtain a baseline for the modeling performance of ioDMD. The PE variant was sampled using zero-mean, unit-covariance Gaussian noise and a unit step function input was used for the ioDMD-based identification. Finally, our CE variant had the initial state

sampled from a unit Gaussian distribution and a (component-wise) shifted initial state $x_{0,i} = 1$ was tested. All methods were tested with ioDMD only and then also with stabilization.

5.1.1 *ioDMD without stabilization*

Figure 1 depicts the relative output error for simulations of systems identified using data associated to the target input $\hat{u}$, PE, and CE for increasingly accurate state-space data compression (that is, for increasingly smaller amounts of compression). The state-space data dimensionality was reduced using the POD method with prescribed projection errors of $10^{-1}, \ldots, 10^{-8}$. In this set of experiments, we did not use stabilization.



(a) Non-regularized identification by noise input or randomly sampled initial state.

(b) Non-regularized identification by step input or shifted initial state.

(c) Regularized identification by noise input or randomly sampled initial state.

(d) Regularized identification by step input or shifted initial state.

Fig. 1 First numerical experiment (Section 5.1.1): Non-stabilized ioDMD-based system identification. The plots show the identified (reduced order) system's output error compared to the original system's output for varying accuracies of the POD projection-based model reduction

In Fig. 1a and b, the ioDMD procedure was not regularized by truncating singular values, while regularization $\Sigma_{ii} < 10^{-5}$ was used in Fig. 1c and d. In Fig. 1a and c, system identification was performed using zero-mean Gaussian noise for the PE and an initial state sampled from a zero-mean Gaussian distribution for CE, respectively. In Fig. 1b and d, respectively, the identification was driven by a step input for the PE and a shifted initial state $x_{0,i} = 1$ for CE.

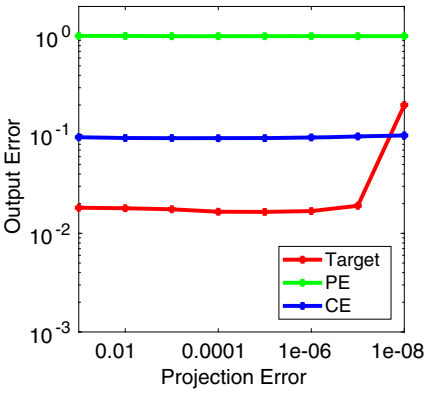
For this set of experiments, using the target data only produced stable systems for large values of acceptable projection error while the PE-derived models were always unstable (and thus had very poor performance regardless of the projection error). In contrast, the CE method produced stable systems for all levels of projection error tested. Performance-wise, when comparing to the few target-data-derived models that also happened to be stable, the CE-derived models had errors that were less than one order of magnitude higher (see Fig. 1a and c). On the other hand, when using the step input or shifted initial state, both PE and CE produced models with increasing accuracy as the level of acceptable projection error of the data was decreased, as seen in Fig. 1b and d. In Fig. 1d, we see that the regularization limited the attainable accuracy for both PE and CE. The target-data-derived system had a constant error independent from the projection error of the data.

5.1.2 ioDMD with stabilization

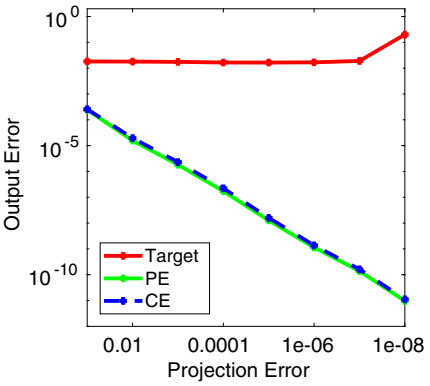
In this second set of experiments, Fig. 2 still shows the relative output error for simulations of systems identified using the target data, PE, and CE for increasingly accurate state-space data compression, but now the systems have been post-processed using our optimization-based approach to enforce stability, as necessary. The state-space data dimensionality was again reduced using the POD method, with prescribed projection errors of $10^{-1}, \ldots, 10^{-8}$. The subfigures are arranged as they were in Fig. 1.

The step function PE and shifted initial state CE are unaffected by our stabilization post-processing phase, as these systems were already stable; thus their plots are the same in Fig. 2b and d as they were in Fig. 1b and d. In the case of using Gaussian noise or randomly sampled initial state (Fig. 2a and c), which had not yielded stable systems for the target data or PE (either with or without regularization), our optimization-based post-processing procedure now enforced stability.

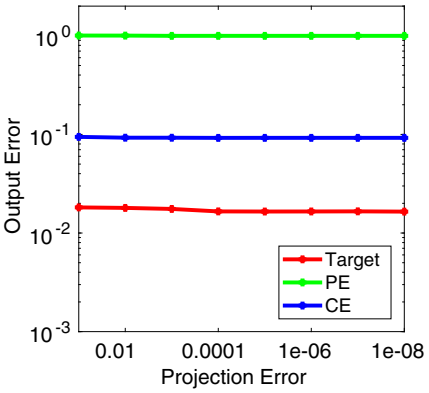
For the ioDMD-derived models that were unstable, GRANSO was able to stabilize all 24 of them. The average number of iterations needed to find the first stabilized version was 13.5 while the maximum was just 61. Furthermore, for 12 of the problems, our full termination criteria were met in less than 20 iterations while the average and maximum iteration counts over all 24 problems were respectively 84.2 and 329. This demonstrates that our optimization-based approach is indeed able to stabilize such models reliably and efficiently. Solving (9) via GRANSO also met our secondary goal, that stabilization is achieved without deviating too significantly from the original unstable models. The largest observed relative change between an initial unstable model and its corresponding GRANSO-derived stabilized version was just 1.44% while the average observed relative change was merely 0.231%; the relative differences were calculated by comparing the vectorization $\text{vec}[A_s, B_s; C_s, D_s]$, where the



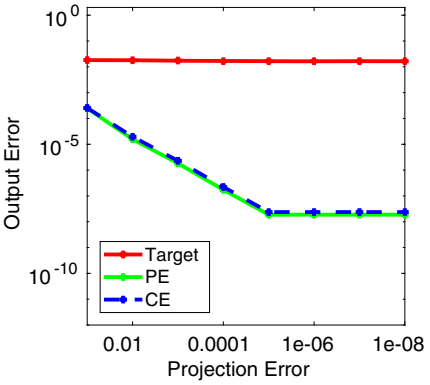
(a) Non-regularized identification by noise input or randomly sampled initial state.



(b) Non-regularized identification by step input or shifted initial state.



(c) Regularized identification by noise input or randomly sampled initial state.



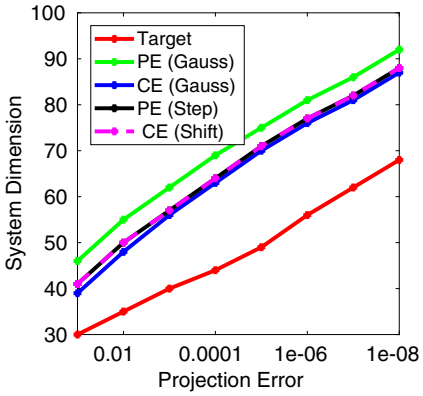
(d) Regularized identification by step input or shifted initial state.

Fig. 2 Second numerical experiment (Section 5.1.2): Stabilized ioDMD-based system identification. The plots show the identified (reduced order) system's output error compared to the original system's output for varying accuracies of the POD projection-based model reduction

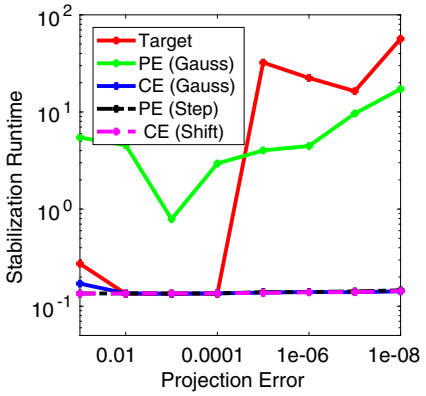
matrices are GRANSO's computed stabilized solution to (9), to a similar vec of the original unstable ioDMD-derived model.

5.1.3 Reduced orders and runtimes

We now compare the order of the identified systems. The order of the identified system is determined by the projection error selected for the state-space compression in the POD. For each data set (target, Gaussian noise PE, Gaussian sample CE, Step input PE, shifted initial state CE), the resulting reduced order is plotted for the different prescribed projection errors in Fig. 3a. As we can see, the step-input PE and shifted-initial-state CE method behave similar in terms of system dimension while for



(a) Comparison of the identified system's reduced order for different pre-selected mean L_2 projection errors used in both experiments (see Section 5.1.1).



(b) Comparison of runtimes in seconds for the stabilization procedure in the second experiment (Section 5.1.2).

Fig. 3 Comparison of reduced identified system orders and stabilization runtimes

the Gaussian-noise PE and Gaussian-sampled initial-state CE, the CE variant resulted in smaller system dimensions.

In terms of computational cost, only the target and Gaussian-noise PE variants required stabilization and as such, it is only for these that we see increased runtimes, as shown by the red and green plots (respectively) in Fig. 3b. Otherwise, the runtimes were mostly identical for the other variants in the comparison.

5.2 Limited-Memory BFGS for stabilization?

One potential downside to our optimization-based stabilization procedure is that *full-memory* BFGS inverse Hessian updating (GRANSO's recommended setting for nonsmooth problems) requires per-iteration work and storage that is quadratic in the number of optimization variables. As the number of optimization variables is $(r+m) \times (r+p)$, the running time required to solve (9) could become unacceptably long as r, the reduced order of the model, is increased. Thus, we now also consider whether or not *limited-memory* BFGS updating can also be effective for solving (9).

Using limited-memory BFGS has the benefit that the per-iteration work and storage is reduced to a linear amount (again in the number of optimization variables). However, one of the tradeoffs is that convergence can be quite slow in practice. For smooth problems, full-memory BFGS converges superlinearly while limited-memory BFGS only does so linearly; on nonsmooth problems, linear convergence is the best one can typically expect. Another potential issue is that while there has been much evidence supporting that full-memory BFGS is very reliable for nonsmooth optimization, the case for using limited-memory BFGS on nonsmooth problems is much less clear; for a good overview of the literature on this topic, see [18, Section 1].

To investigate this question, we reran the experiments from Section 5.1 a second time but where GRANSO's limited-memory BFGS mode was now enabled.

Specifically, we configured GRANSO to approximate the inverse Hessian at each iteration using only the 10 most recently computed gradients, accomplished by setting `opts.limited_mem_size = 10`. All other parameters of our experimental setup were kept as they were described earlier.

In this limited-memory configuration, GRANSO often required significantly more iterations, as one might expect. The average and max number of iterations to find the first stable version of a model were respectively 73.6 and 474, about an order of magnitude more iterations than incurred when using full-memory BFGS. On the other hand, for 19 of the 24 problems, stable models were encountered within the first 20 iterations. To meet our full termination criteria, the average and max number of iterations were respectively 222.0 and 811, roughly about two and a half times more than incurred when using full-memory BFGS. Nevertheless, 12 of the 24 problems were still satisfactorily solved in less than 20 iterations, matching the earlier result when using full-memory BFGS. Despite the large increases in iteration counts, GRANSO's overall runtime was on average 3.83 times faster when enabling limited-memory BFGS.

With respect to output error in this limited-memory evaluation, the resulting stabilized models still essentially matched the earlier results using full-memory BFGS. There was one notable exception, for the target data using the smallest projection error of 10^{-8}, where GRANSO remarkably found a better-performing model when using limited-memory BFGS. However, we did observe that the quality of the stabilized models appeared to be much more sensitive to changing GRANSO's parameters than they were when using full-memory BFGS. As a consequence, we still advocate that solving (9) with GRANSO is generally best done using its default full-memory BFGS updating. Nonetheless, if this is simply not feasible computationally, one may still be able to obtain good results using limited-memory BFGS but perhaps not as reliably or consistently.

As a final clarifying remark on this topic, we note that one cannot necessarily expect good performance on nonsmooth problems when using *just any* BFGS-based optimization code and that generally it is critical that the choice of software is one specifically designed for nonsmooth optimization. Indeed, this is highlighted in the evaluation done in [10, Section 6], where off-the-shelf quasi-Newton-based codes built for smooth optimization perform much worse on a test set of nonsmooth optimization problems compared to the quasi-Newton-based codes specifically built with nonsmooth optimization in mind.

6 Conclusion

In this work, we evaluated the approximation quality of ioDMD system identification using a novel excitation scheme and a new optimization-based, post-processing procedure to ensure stability of the identified systems. Our new cross excitation strategy, particularly when used with random sampling, often produces better results than when using persistent excitation, and our experiments indicate that both excitation schemes are useful for efficiently obtaining good models for approximating the target

data. Furthermore, we show that directly solving a nonsmooth constrained optimization problem can indeed be a viable approach for stabilizing ioDMD-derived systems while retaining the salient properties for approximating the output response.

Code Availability

The source code of the presented numerical examples can be obtained from:

http://runmycode.org/companion/view/2902

and is authored by: CHRISTIAN HIMPE and TIM MITCHELL.

Acknowledgements Open access funding provided by Max Planck Society. The authors are grateful for the helpful feedback and comments provided by the two anonymous referees.

References

1. Alla, A., Kutz, J.N.: Nonlinear model order reduction via dynamic mode decomposition. SIAM J. Sci. Comput. **39**(5), B778–B796 (2017). https://doi.org/10.1137/16M1059308
2. Amsallem, D., Farhat, C.: Stabilization of projection-based reduced-order models. Numer. Methods Eng. **91**(4), 358–377 (2012). https://doi.org/10.1002/nme.4274
3. Annoni, J., Gebraad, P., Seiler, P.: Wind farm flow modeling using input-output dynamic mode decomposition. In: American Control Conference (ACC), pp. 506–512 (2016). https://doi.org/10.1109/ACC.2016.7524964
4. Annoni, J., Seiler, P.: A method to construct reduced-order parameter-varying models. Int. J. Robust Nonlinear Control **27**(4), 582–597 (2017). https://doi.org/10.1002/rnc.3586
5. Antoulas, A.C.: Approximation of Large-Scale dynamical systems, Adv. Des. Control, vol. 6. Society of Industrial and Applied Mathematics Publications, Philadelphia (2005). https://doi.org/10.1137/1.9780898718713
6. Aström, K.J., Eykhoff, P.: System identification – a survey. Automatica **7**(2), 123–162 (1971). https://doi.org/10.1016/0005-1098(71)90059-8
7. Brunton, B.W., Johnson, L.A., Ojemann, J.G., Kutz, J.N.: Extracting spatial-temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition. J. Neurosci. Methods **258**, 1–15 (2016). https://doi.org/10.1016/j.jneumeth.2015.10.010
8. Burke, J.V., Overton, M.L.: Variational analysis of non-Lipschitz spectral functions. Math. Program. **90**(2, Ser. A), 317–352 (2001). https://doi.org/10.1007/s102080010008
9. Chen, K.K., Tu, J.H., Rowley, R.W.: Variants of dynamic mode decomposition: boundary condition, Koopman, and Fourier analyses. Nonlinear Sci. **22**(6), 887–915 (2012). https://doi.org/10.1007/s00332-012-9130-9
10. Curtis, F.E., Mitchell, T., Overton, M.L.: A BFGS-SQP method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles. Optim. Methods Softw. **32**(1), 148–181 (2017). https://doi.org/10.1080/10556788.2016.1208749
11. Fernando, K.V., Nicholson, H.: On the structure of balanced and other principal representations of SISO systems. IEEE Trans. Autom. Control **28**(2), 228–231 (1983). https://doi.org/10.1109/TAC.1983.1103195

12. Holmes, P., Lumley, J.L., Berkooz, G., Rowley, C.W.: Turbulence, Coherent Structures, Dynamical Systems and Symmetry. Cambridge Monographs on Mechanics. Cambridge University Press, Cambridge (2012). https://doi.org/10.1017/CBO9780511919701
13. Ionescu, T.C., Fujimoto, K., Scherpen, J.M.A.: Singular value analysis of nonlinear symmetric systems. IEEE Trans. Autom. Control **56**(9), 2073–2086 (2011). https://doi.org/10.1109/TAC.2011.2126630
14. Katayama, K.: Subspace Methods for System Identification. Communications and Control Engineering. Springer, London (2005). https://doi.org/10.1007/1-84628-158-X
15. Koopman, B.O.: Hamiltonian systems and transformation in Hilbert space. Proc. Natl. Acad. Sci. **17**(5), 315–381 (1931). http://www.pnas.org/content/17/5/315.full.pdf
16. Kutz, J.N., Brunton, S.L., Brunton, B.W., Proctor, J.L.: Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems. Society of Industrial and Applied Mathematics, Philadelphia. https://doi.org/10.1137/1.9781611974508 (2016)
17. Lall, S., Marsden, J.E., Glavaški, S.: Empirical model reduction of controlled nonlinear systems. In: Proceedings of the IFAC World Congress, vol. F, pp. 473–478 (1999). https://doi.org/10.1016/S1474-6670(17)56442-3
18. Lewis, A.S., Overton, M.L.: Nonsmooth optimization via quasi-Newton methods. Math. Program. **141**(1–2, Ser. A), 135–163 (2013). https://doi.org/10.1007/s10107-012-0514-2
19. Mezic, I.: Spectral properties of dynamical systems, model reduction and decompositions. Nonlinear Dyn. **41**(1), 309–325 (2005). https://doi.org/10.1007/s11071-005-2824-x
20. Mitchell, T.: GRANSO: GRadient-based Algorithm for Non-Smooth Optimization. http://timmitchell.com/software/GRANSO. See also [10]
21. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, New York (1999). https://doi.org/10.1007/b98874
22. Oku, H., Fujii, T.: Direct subspace model identification of LTI systems operating in closed-loop. In: 43Rd IEEE Conference on Decision and Control, pp. 2219–2224 (2004). https://doi.org/10.1109/CDC.2004.1430378
23. Proctor, J.L., Brunton, S.L., Kutz, J.N.: Dynamic mode decomposition with control. SIAM J. Appl. Dyn. Syst. **15**(1), 142–161 (2016). https://doi.org/10.1137/15M1013857
24. Proctor, J.L., Brunton, S.L., Kutz, J.N.: Generalizing Koopman Theory to Allow for Inputs and Control. arXiv:1602.07647, Cornell University. 1602.07647. Math.OC (2016)
25. Rowley, C.W., Dawson, S.T.M.: Model reduction for flow analysis and control. Annu. Rev. Fluid Mech. **49**, 387–417 (2017). https://doi.org/10.1146/annurev-fluid-010816-060042
26. Rowley, C.W., Mezic, I., Bagheri, S., Schlatter, P., Henningson, D.S.: Spectral analysis of nonlinear flows. J. Fluid Mech. **641**, 115–1127 (2009). https://doi.org/10.1017/S0022112009992059
27. Schmid, P.J.: Dynamic mode decomposition of numerical and experimental data. J. Fluid Mech. **656**, 5–28 (2010). https://doi.org/10.1017/S0022112010001217
28. Tu, J.H., Rowley, C.W., Luchtenburg, D.M., Brunton, S.L., Kutz, J.N.: On dynamic mode decomposition: Theory and applications. J. Comput. Dyn. **1**(2), 391–421 (2014). https://doi.org/10.3934/jcd.2014.1.391
29. Van Den Hof, P.M.J., Schrama, R.J.P.: An indirect method for transfer function estimation from closed loop data. Automatica **29**(6), 1523–1527 (1993). https://doi.org/10.1016/0005-1098(93)90015-L
30. Van Overschee, P., De Moor, B.: N4SID: Numerical algorithms for state space subspace system identification. In: IFAC Proceedings Volumes, vol. 26, pp. 55–58 (1993). https://doi.org/10.1016/S1474-6670(17)48221-8
31. Viberg, M.: Subspace-based methods for the identification of linear time-invariant systems. Automatica **31**(12), 1835–1851 (1995). https://doi.org/10.1016/0005-1098(95)00107-5