A software package for system identification in the behavioral setting

Ivan Markovsky

Department ELEC, Vrije Universiteit Brussel
Pleinlaan 2, Building K, B-1050 Brussels, Belgium
Fax: +32–2–629.28.50 , Email: imarkovs@vub.ac.be

Abstract

An identification problem with no a priori separation of the variables into inputs and outputs and representation invariant approximation criterion is considered. The model class consists of linear time-invariant systems of bounded complexity and the approximation criterion is the minimum of a weighted 2-norm distance between the given time series and a time series that is consistent with the model. The problem is equivalent to and is solved as a mosaic-Hankel structured low-rank approximation problem. Software implementing the approach is developed and tested on benchmark problems. Additional nonstandard features of the software are specification of exact and missing variables and identification from multiple experiments.

Keywords: system identification; model reduction; behavioral approach; missing data; low-rank approximation; total least squares; reproducible research; DAISY.

1 Introduction

One of the main criticisms to the behavioral approach in system theory and control [25, 17] is that it is not supported by numerical algorithms and software. Despite its conceptual and theoretic advantages, the behavioral approach has produced only a few engineering tools; most notably the identification methods of [19] and [10]. Compared to the abundance of algorithms and software in the classical setting, this scarcity of tools accounts to a large extent for the slow acceptance of the behavioral approach by engineers.

This paper presents a flexible method for system identification in the behavioral setting, which is an outgrow of the method in [10]. The method is implemented in a publicly available software package which is fast, easy to use, and has the following nonstandard features.

1. *Representation free problem formulation*

   Classical identification problems are invariably defined in terms of representations of the system—transfer function, input/state/output, convolution kernel, *etc*. A problem can be equivalently posed in frequency domain as well as time domain, using input/state/output, convolution, or any other representation. Different representations have different advantages, so that all of them should be considered as options in solving the problem. The model representation should be viewed as an implementation detail of the particular solution method. A problem formulation that is representation free is defined in terms of the desired system behavior—set of trajectories of the system.

   Related to the use of a representation invariant problem formulation, is the assumption in classical system identification problems that an input/output partitioning of the variables is given and fixed. The choice of inputs and outputs is not unique [26, Section VIII]. Different input/output partitions, generically, lead to the same optimal model's behavior [11]. In nongeneric cases, however, the optimal model is not compatible with a specified input/output partition. If such a partition is chosen, the identification problem is ill-posed. In practice, even a well-posed problem with fixed input/output partition can be ill-conditioned. Using the representation free setting and thus leaving the choice of the input-output partition to the identification method has the potential advantage of leading to methods that are more general conceptually (the input/output partition is identified together with the model), easier to analyze theoretically (solution always exists), and are computationally more robust (ill-conditioning is avoided).

2. *Multiple experiments*

   Another nonstandard feature of the method presented in the paper is the use data of multiple measurement experiments. In static estimation problems, a measurement experiment yields a q-dimensional observation vector, so that the only way to increase the amount of data is to repeat the experiment. In dynamic problems, an experiment yields a q-dimensional vector time series. More data can be collected in this case also by increasing the length of the time series (number of samples in time). Our framework unifies the two approaches of data collection—repeated experiments and longer experiment. The former approach is of interest, for example, in the case of autonomous dynamical systems, where asymptotic analysis in time is not applicable.

   The problem of using data of multiple experiments for identification of a single model is often addressed by heuristic methods. The rigorous method of [20] is based on concatenation of the separate time series into one time series, adding transient responses at the times of transition from one trajectory to another. This reduces the problem of identification from N time series to the problem of identification from a single time series *and* estimation of N transient responses—both being classical problems. This approach is implemented in all methods available in the System Identification Toolbox of Matlab [5].

   The method presented in this paper uses a different approach. The time series parameterize separate block-Hankel matrices, with fixed row dimension but possibly different column dimensions. The block-Hankel matrices are appended to each other, forming a $1 \times N$ block matrix with block-Hankel elements. Such a matrix is called a mosaic block-Hankel matrix. The identification problem of using data from multiple experiments is then equivalent to low-rank approximation of the mosaic block-Hankel matrix. Our approach is a natural generalization of the classical approach for solving static estimation problems where the blocks are the individual observations and the mosaic block-Hankel matrix is the unstructured $q \times N$ matrix of the stacked next to each other observations.

3. *Exact and missing data*

   Due to malfunctioning of measurement devices, communication channels, or storing devices, the available data may have missing measurements at some time instances. When the missing data span over ℓ or more sequential time instances, where ℓ is the lag of the system, the problem reduces to the problem of identification from two independent time series. Also for problems with a given input/output partitioning, missing output variables can be estimated by existing methods. More general patterns of missing data, *e.g.*, non sequential missing variables and missing variables in arbitrary combination of variables, require new methods. The approach used in the paper, treats missing data by minimizing a weighted approximation criterion with zero weights associated to the missing values.

   The opposite of a missing data value is an exact data value. An example of how exact data occur in practice is data collection experiment for a system starting "at rest", *i.e.*, zero initial conditions. This prior knowledge can be used in the identification problem by imposing equality constraints, setting the approximation of the exact values to the given data values. The approach of imposing exact values used in this paper, is by minimizing a weighted approximation criterion with infinite weights associated to the exact values. Thus, exact and missing values are treated uniformly as extreme cases of noise corrupted observations when the noise has zero and infinite variance, respectively.

4. *Software implementation in a literature programming style*

   Scientific publications rarely include full implementation details of the methods described. A software implementation of the method (when available as open source) necessarily contains the full details about the method and is therefore the ultimate source of information. Even well documented code, however, may be difficult to read and may not correspond to a description in related scientific publication. The literature programming style [2] is a possible solution to this issue. A literate program includes the source code as an integral part of the document describing it. A literate program is designed to be read by a human being rather than a computer.

   An extended version of this paper is a literate program for the described software package. The code is split into chunks and the chunks are presented in a logical sequence, which happens to be different from the sequence in which the chunks appear in the computer executable programs. The code chunks provide the implementation details for the algorithmic steps and, vice verse, the presentation in the paper serves as documentation of the

code. The particular literate programming tool used in this paper is `noweb` [18]. The computer executable code as well as the human readable text are automatically extracted from the literate program by `noweb`, so that the code and the paper coexist in common source files.

The approach for solving the identification problem, used in the paper, is weighted structured low-rank approximation [6]. Rank deficient mosaic-Hankel matrices are in a one-to-one correspondence with trajectories of linear time-invariant systems of bounded complexity. Therefore, data approximation by a linear time-invariant system of bounded complexity is a mosaic-Hankel low-rank approximation problem. The behavioral approach and the low-rank approximation setting are two sides of the same coin: they use representation free problem formulations, where the fundamental object of interest is a relation among variables rather than a map. The system theoretic notion of a linear time-invariant model of bounded complexity corresponds to the linear algebra notion of a rank deficient Hankel matrix. The linear algebra setting provides algorithms and software for addressing problems in the system theoretic setting.

Element-wise weights allow approximation with an emphasis on some variables and/or some time instances. Zero and infinite weights are allowed. A zero weight ignores the corresponding data point in the approximation and an infinite weight forces the approximation of the corresponding data point to be equal to the given one. This allows us to take into account missing and exact data.

Once the identification problem is cast as a mosaic-Hankel structured low-rank approximation problem, it is solved by the efficient methods of [23]. A practical implementation of the methods in C++, using optimization algorithms from the GNU scientific library [1], is available [8]. The identification function presented in this paper is a wrapper function to the structured low-rank approximation solver of [8]. The software is available from:

`http://homepages.vub.ac.be/~imarkovs/slra/software.html`

The paper is organized as follows: Section 2 sets the notation and states the considered identification problem. Section 3 presents the solution approach and gives details about the implementation of the software. Specific identification problems—L_2-optimal model reduction, identification from data consisting of multiple trajectories of different lengths, and identification with missing data—are presented in Sections 5–7, respectively. In Section 8, the performance of the identification package is tested on real-life and simulated data from the database for system identification DAISY.

2 Problem formulation

2.1 Model class: linear time-invariant systems of bounded complexity

A discrete-time dynamical system $\mathscr{B}$ is a collection of trajectories — q-variables time-series $w : \mathbb{Z} \to \mathbb{R}^q$. The class of finite dimensional linear time-invariant systems with q variables and at most $\mathtt{m}$ inputs is denoted by $\mathscr{L}_{\mathtt{m}}^q$. A linear time-invariant system $\mathscr{B} \in \mathscr{L}_{\mathtt{m}}^q$ admits a representation by constant coefficients difference equation

$$\mathscr{B} = \mathscr{B}(R) := \{ w \mid R_0 w + R_1 \sigma w + \cdots + R_\ell \sigma^\ell w = 0 \}, \tag{DE}$$

where σ is the shift operator

$$(\sigma w)(t) = w(t+1).$$

Note 1 (Kernel representation). In the literature on the behavioral approach to systems and control, (DE) is called a kernel representation, because it can be written more compactly as the kernel

$$\ker\big(R(\sigma)\big) := \{ w \mid R(\sigma) w = 0 \}$$

of the operator $R(\sigma)$, where

$$R(z) := R_0 w + R_1 z + \cdots + R_\ell z^\ell$$

is a polynomial matrix.

The minimal natural number ℓ, for which there exists an ℓth order difference equation representation for $\mathscr{B}$ is an important invariant of the system, called the *lag*. The number of inputs and the lag specify the complexity of the system in the sense that the dimension of the restriction of $\mathscr{B}$ to the interval $[1,T]$, where $T \geq \ell$, is bounded by $T\mathtt{m}+\ell(q-\mathtt{m})$. The subset of $\mathscr{L}^q_{\mathtt{m}}$ with lag at most ℓ is denoted by $\mathscr{L}^q_{\mathtt{m},\ell}$.

No a priori separation of the variables $w_1,\ldots,w_q$ into inputs and output is made. However, the variables can always be partitioned into inputs u (free variables) and outputs y (dependent variables). A convenient way to represent an input/output partitioning is $w = \Pi\left[\begin{smallmatrix} u \\ y \end{smallmatrix}\right]$, where Π is a $q \times q$ permutation matrix.

An input/output representation

$$\mathscr{B}(P,Q,\Pi) = \{\, w = \Pi(u,y) \mid Q(\sigma)u = P(\sigma)y \,\},$$

of the system $\mathscr{B}(R)$ is obtained by partitioning the polynomial matrix $R\Pi$ as

$$R\Pi = \begin{bmatrix} \overset{\mathtt{m}}{Q} & \overset{\mathtt{p}}{-P} \end{bmatrix}, \quad \text{with } P \text{ nonsingular.}$$

In addition, the system can be represented in the input/state/output form

$$\mathscr{B} = \mathscr{B}(A,B,C,D,\Pi) := \{\, w = \Pi(u,y) \mid \text{there is } x, \text{ such that } \sigma x = Ax+Bu,\ y = Cx+Du \,\}. \tag{I/S/O}$$

If the permutation matrix Π is the identity matrix, then $w = (u,y)$, *i.e.*, the first $\mathtt{m}$ variables of w are inputs and the remaining $\mathtt{p}$ variables are outputs. In this case, Π is skipped in (I/S/O), *i.e.*, $\mathscr{B}(A,B,C,D) = \mathscr{B}(A,B,C,D,I)$.

The number of inputs $\mathtt{m}$, the number of outputs $\mathtt{p} = q-\mathtt{m}$, and the minimal state dimension $\mathtt{n}$ of an input/state/output representation of $\mathscr{B}$ are invariant of the representation and in particular of the input/output partitioning. The order $\mathtt{n}$ of a state-space representation of a linear time-invariant system $\mathscr{B} = \mathscr{B}(R)$ with lag ℓ and $\mathtt{p}$ outputs is $\mathtt{n} \leq \ell\mathtt{p}$. In the case when the block $P_\ell \in \mathbb{R}^{\mathtt{p}\times\mathtt{p}}$ of $R_\ell = \begin{bmatrix} Q_\ell & -P_\ell \end{bmatrix}$ is nonsingular, $\mathtt{n} = \ell\mathtt{p}$ and $w = (u,y)$ is a possible input/output partition, *i.e.*, Π can be chosen equal to I. This simplifying assumption is made in the rest of the paper. The class of systems with q variables and inputs, order, and lag bounded by, respectively $\mathtt{m}$, $\mathtt{n}$, and ℓ is denoted by $\mathscr{L}^{q,\mathtt{n}}_{\mathtt{m},\ell}$.

⟨*define* p *and* n⟩≡
```
p = q - m; n = p * ell;
```

2.2 Approximation criterion: distance between data and model

The *misfit* (lack of fit) between the data w_{d} and the model $\mathscr{B}$ is measured by the orthogonal distance from w_{d} to $\mathscr{B}$

$$M(w_{\mathrm{d}},\mathscr{B}) := \min_{\widehat{w}^1,\ldots,\widehat{w}^N \in \mathscr{B}} \sqrt{\sum_{k=1}^{N} \|w^k_{\mathrm{d}} - \widehat{w}^k\|_2^2}. \tag{M}$$

Intuitively, the misfit shows how much the model $\mathscr{B}$ fails to "explain" the data w_{d}.

Missing elements are marked by the symbol `NaN` and are excluded from the approximation criterion, *i.e.*,

$$\mathtt{NaN} - \widehat{w}_i(t) = 0, \qquad \text{for all} \quad \widehat{w}_i(t) \in \mathbb{R}.$$

The optimal approximate modeling problem considered aims to find a system $\widehat{\mathscr{B}}$ in the model class $\mathscr{L}^q_{\mathtt{m},\ell}$ that best fits the data according to the misfit criterion.

Given a set of time series

$$w_{\mathrm{d}} = \{\, w^1_{\mathrm{d}},\ldots,w^N_{\mathrm{d}} \,\}, \quad \text{where} \quad w^k_{\mathrm{d}} = \big(w^k_{\mathrm{d}}(1),\ldots,w^k_{\mathrm{d}}(T_k)\big), \quad w^k_{\mathrm{d}}(t) \in \mathbb{R}^q,$$

and a complexity specification $(\mathtt{m},\ell)$, find the system

$$\widehat{\mathscr{B}} := \arg\min_{\mathscr{B} \in \mathscr{L}^q_{\mathtt{m},\ell}} M(w_{\mathrm{d}},\mathscr{B}). \tag{SYSID}$$

Special cases of (SYSID) are static data modeling ($\ell = 0$) and output-only or autonomous system identification ($\mathtt{m} = 0$). The solution approach, described next, leads to an algorithm that covers these special cases. In addition,

1. elements of the given time series w_{d} can be specified as "missing" by passing the symbol `NaN` for their value;
2. elements of the given time series w_{d} can be specified as "exact", in which case they appear unmodified in the approximation $\widehat{w}$;
3. the approximation $\widehat{w}$ can be constrained to be a trajectory of the model $\mathscr{B}$, generated under a priori fixed initial conditions w_{ini}, see [7], *i.e.*,
$$\begin{bmatrix} w_{\mathrm{ini}} \\ \widehat{w} \end{bmatrix} \in \mathscr{B}.$$
(Note that problem (SYSID) identifies the model without prior knowledge about the initial conditions, under which the data w_{d} is generated, *i.e.*, w_{ini} is a free variable.)

2.3 Stochastic interpretation

The system identification problem considered in the paper is defined as a deterministic approximation problem. As shown in [6, Section 3.1], however, it yields the maximum likelihood estimator in the errors-in-variables setting [21].

Proposition 2 ([6, Proposition 6])**.** *Assume that the data w_{d} is generated in the errors-in-variables setting*

$$w_{\mathrm{d}} = \bar{w} + \widetilde{w},$$

where the true data $\bar{w}$ is a trajectory of a true model $\bar{\mathscr{B}} \in \mathscr{L}^{q}_{\mathtt{m},\ell}$ and the measurement noise $\widetilde{w}$ is zero mean normally distributed with covariance matrix that is a multiple of the identity. Then the solution $\widehat{\mathscr{B}}^*$ of (SYSID) is a maximum likelihood estimator for the true model $\bar{\mathscr{B}}$.

In [15, 3], it is proven that under additional mild assumptions the estimator $\widehat{\mathscr{B}}^*$ is consistent and the estimated parameters have asymptotically normal joint distribution. Therefore, asymptotic confidence regions can be obtained as a byproduct of the optimization algorithm (see [8, Section 5.1]). The statistical setting justifies the choice of the deterministic approximation criterion and provides a testbed for the method: the method works "well" (consistency) and is optimal (asymptotic efficiency) under specified conditions.

Note 3 (Exact and missing data correspond to noisy data with, respectively, zero and infinite noise variance). In the classical regression model, an input-output partitioning of the variables is a priori given and the input variables (regressors) are assumed to be noise free. Classical regression is a special case of the errors-in-variables model when the input noise variance is zero. The opposite of a noise free variable is a missing variable, or equivalently a variable with infinite variance in the errors-in-variables setting.

3 Solution approach

For a given set of trajectories w_{d} and a natural number ℓ, the mosaic-block-Hankel matrix (a $1 \times N$ block matrix with block-Hankel blocks) is defined as

$$\mathscr{H}_{\ell+1}(w_{\mathrm{d}}) := \begin{bmatrix} \mathscr{H}_{\ell+1}(w_{\mathrm{d}}^1) & \cdots & \mathscr{H}_{\ell+1}(w_{\mathrm{d}}^N) \end{bmatrix}, \quad \text{where} \quad \mathscr{H}_{\ell+1}(w_{\mathrm{d}}^k) := \begin{bmatrix} w_{\mathrm{d}}^k(1) & w_{\mathrm{d}}^k(2) & \cdots & w_{\mathrm{d}}^k(T-\ell) \\ w_{\mathrm{d}}^k(2) & w_{\mathrm{d}}^k(3) & \cdots & w_{\mathrm{d}}^k(T-\ell+1) \\ \vdots & \vdots & & \vdots \\ w_{\mathrm{d}}^k(\ell+1) & w_{\mathrm{d}}^k(\ell+2) & \cdots & w_{\mathrm{d}}^k(T) \end{bmatrix}.$$

The solution method is based on the following equivalence

$$\big(w_{\mathrm{d}}(1), \ldots, w_{\mathrm{d}}(T_k - \ell)\big) \in \mathscr{B} \in \mathscr{L}^{q,\mathtt{n}}_{\mathtt{m},\ell}, \quad \text{for } k = 1, \ldots, N \qquad \Longleftrightarrow \qquad \operatorname{rank}\big(\mathscr{H}_{\ell+1}(w_{\mathrm{d}})\big) \leq (\ell+1)\mathtt{m} + \mathtt{n},$$

i.e., the time series w_{d}, possibly except for the last ℓ samples, are exact trajectories of a linear time-invariant model $\mathscr{B}$ with complexity bounded by $(\mathtt{m},\mathtt{n})$ if and only if the mosaic-block-Hankel matrix $\mathscr{H}_{\ell+1}(w_{\mathrm{d}})$ has rank bounded by

$$r = (\ell+1)\mathtt{m}+\mathtt{n}.$$

Therefore, the identification problem (SYSID) is equivalent to the structured low-rank approximation problem

$$\text{minimize} \quad \text{over } \widehat{w} \quad \|w_{\mathrm{d}}-\widehat{w}\|_{\ell_2}^2 \quad \text{subject to} \quad \operatorname{rank}\left(\mathscr{H}_{\ell+1}(\widehat{w})\right) \leq r. \tag{SLRA}$$

Note 4. Under the assumption that P_ℓ is full rank, the whole trajectories $w_{\mathrm{d}}^1,\ldots,w_{\mathrm{d}}^N$ are in the behavior of the model.

Problem (SLRA) is a classic problem that can be solved in different ways. The approach used in the paper is based on the kernel representation of the rank constraint

$$\operatorname{rank}\left(\mathscr{H}_{\ell+1}(\widehat{w})\right) \leq r \qquad \Longleftrightarrow \qquad R\mathscr{H}_{\ell+1}(\widehat{w}) = 0 \quad \text{and} \quad R^\top R = I_{(\ell+1)q-r}. \tag{KER}$$

The matrix R in the right-hand-side of (KER) is related to the parameters $R_0,R_1,\ldots,R_\ell$ of the difference equation representation (DE) of the exact model for $\widehat{w}$ as follows:

$$R = \begin{bmatrix} R_0 & R_1 & \cdots & R_\ell \end{bmatrix}, \qquad \text{where} \quad R_i \in \mathbb{R}^{\mathtt{p}\times q}.$$

Algorithms and software for mosaic-Hankel low-rank approximation are developed in [9, 23]. The software of [8] is used. The software presented in this paper is just an interface to the structured low-rank approximation solver for the purpose of linear time-invariant system identification, see Figure 1. On its own, the structured low-rank approximation solver computes a parameter $\widehat{R}$ of the difference equation representation of the optimal approximating system $\widehat{\mathscr{B}}$. The system identification function converts the parameter $\widehat{R}$ to the parameters $(\widehat{A},\widehat{B},\widehat{C},\widehat{D})$ of an input/state/output representation of $\widehat{\mathscr{B}}$ in order to facilitate the usage of the model by other analysis and synthesis tools. In addition, a system $\mathscr{B}_{\mathrm{ini}} \in \mathscr{L}^q_{\mathtt{m},\ell}$, specified by parameters $(A_{\mathrm{ini}},B_{\mathrm{ini}},C_{\mathrm{ini}},D_{\mathrm{ini}})$ can be used as an initial approximation for the optimization algorithm. The parameters $(A_{\mathrm{ini}},B_{\mathrm{ini}},C_{\mathrm{ini}},D_{\mathrm{ini}})$ are converted to the parameter R_{ini}, used by the structured low-rank approximation solver. Details about these conversions are given in Appendix B. Although the identification function has as external interface the (I/S/O) representation of the model, the internal computations are done via the parameter R of the difference equation representation.

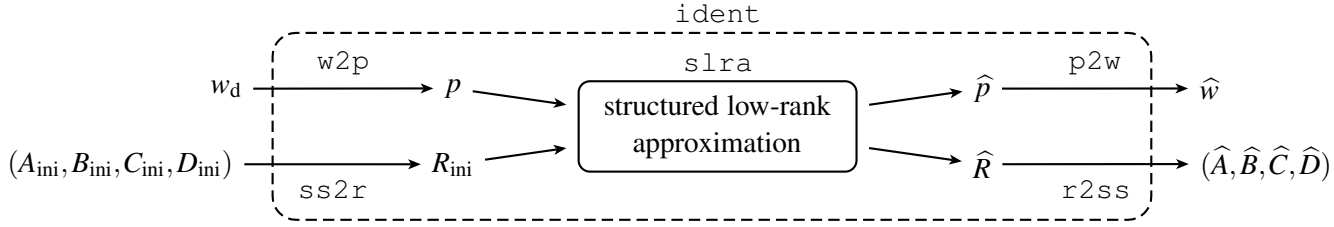

Figure 1: Implementation of the system identification function `ident`.

The software package [8] naturally deals with the mosaic-Hankel matrix

$$\mathscr{H}'_{\ell+1}(w) = \begin{bmatrix} \mathscr{H}_{\ell+1}(w_1^1) & \cdots & \mathscr{H}_{\ell+1}(w_1^N) \\ \vdots & & \vdots \\ \mathscr{H}_{\ell+1}(w_q^1) & \cdots & \mathscr{H}_{\ell+1}(w_q^N) \end{bmatrix}$$

rather than the mosaic-block-Hankel matrix $\mathscr{H}_{\ell+1}(w)$. The two matrices $\mathscr{H}_{\ell+1}(w)$ and $\mathscr{H}'_{\ell+1}(w)$ are related by a row permutation that does not affect the optimal solution $\widehat{p}$ of (SLRA). For this reason, we use in the software implementation the mosaic-Hankel structure and take into account the permutation in the pre- and post-processing operations, as explained in Appendix B. For example, the parameter vector R' corresponding to $\mathscr{H}'_{\ell+1}(w)$ is

$$R' = \begin{bmatrix} R'_0 & R'_1 & \cdots & R'_q \end{bmatrix}, \qquad \text{where} \quad R'_i \in \mathbb{R}^{\mathtt{p}\times(\ell+1)},$$

is link to the kernel representation via (using Matlab notation for indexing of a matrix) $R'_i(:,j) = R_j(:,i)$.

⟨*structure specification*⟩≡

```
s.m = (ell + 1) * ones(q, 1); s.n = T - ell; r = (ell + 1) * m + n;
```

The function `ident` solves the approximate identification problem (SYSID), and `misfit` computes the misfit $M(w_{\mathrm{d}}, \mathscr{B})$. They implement the following mappings (the input/output parameters are defined in Appendix A):

`ident`: $(w_{\mathrm{d}}, \mathtt{m}, \ell) \mapsto (\widehat{A}, \widehat{B}, \widehat{C}, \widehat{D})$, where $(\widehat{A}, \widehat{B}, \widehat{C}, \widehat{D})$ define a (locally) optimal solution of (SYSID)

⟨`ident` *function definition*⟩≡
```
function [sysh, info, wh, xini] = ident(w, m, ell, opt)
```

In the simulation examples the method is called by the sequence

⟨*use* `ident`⟩≡
```
try, [sysh, info, wh, xini] = ident(w, m, ell, opt);
catch, sysh = NaN; wh = NaN; end
```

`misfit`: $\big(w_{\mathrm{d}}, (A,B,C,D)\big) \mapsto (M, \widehat{w})$, where M is the misfit between $\mathscr{B}(A,B,C,D)$ and w_{d}, and $\widehat{w}$ is the optimal approximation of w_{d} within $\mathscr{B}(A,B,C,D)$ (the smoothed trajectory).

⟨`misfit` *function definition*⟩≡
```
function [M, wh, xini] = misfit(w, sysh, opt)
```

4 Identification without input/output partitioning

In this section, we show a simulation example that illustrates the independence of the representation free problem formulation (SYSID) of an a priori given input output partitioning of the variables. Noisy data $(u_{\mathrm{d}}, y_{\mathrm{d}})$ is generated from a single-input single-output second order system

⟨*permutation of the variables*⟩≡
```
ell = 2; m = 1; p = 1; q = m + p; n = ell * p; T = 30; s = 0.1;
sys0 = drss(n, p, m); u0 = rand(T, 1); y0 = lsim(sys0, u0);
w0 = [u0 y0]; w = w0 + s * randn(T, q);
```

and the identification method `ident` is applied, first on the data $w_{\mathrm{d}} = (u_{\mathrm{d}}, y_{\mathrm{d}})$:

⟨*permutation of the variables*⟩+≡
```
opt.solver = 'm';
[sysh1, info1] = ident(w, m, ell, opt);
```

and, second on the data $w'_{\mathrm{d}} = (y_{\mathrm{d}}, u_{\mathrm{d}})$ with permuted variables:

⟨*permutation of the variables*⟩+≡
```
[sysh2, info2] = ident(fliplr(w), m, ell, opt);
```

The achieved misfit $M(w_{\mathrm{d}}, \widehat{\mathscr{B}}) =$ `info1.M` and $M(w'_{\mathrm{d}}, \widehat{\mathscr{B}}') =$ `info2.M` in the two cases is equal and the identified models $\widehat{\mathscr{B}} =$ `sys1` and $\widehat{\mathscr{B}}' =$ `sys2` are closely related. (If $\mathscr{B}_1$ is invertible $\mathscr{B}_2$ is equal to the inverse of $\mathscr{B}_1$.)

⟨*permutation of the variables*⟩+≡
```
[qh1, ph1] = tfdata(tf(sysh1), 'v');
[qh2, ph2] = tfdata(tf(sysh2), 'v');

[qh1 / qh1(1); ph2]
[ph1; qh2 / qh2(1)]

format long, [info1.M info2.M], format
```

Although, identification from data with permuted variables leads to equivalent optimization problems, the behavior of optimization methods in these problems may differ significantly. This is illustrated on the plots of Figures 2 and 3. Figure 2 shows the convergence of the parameters R (identification using w_{d}) and R' (identification using w'_{d}) in the course of the optimization algorithm. Figure 3 shows the approximation errors $M_k := M(w_{\mathrm{d}}, \widehat{\mathscr{B}}_k)$ and $M'_k := M(w'_{\mathrm{d}}, \widehat{\mathscr{B}}'_k)$ as a function of the iteration step k.

⟨*permutation of the variables*⟩+≡
```
figure, plot(info1.Th'), print_fig('ident-conv-th-1')
figure, plot(info2.Th'), print_fig('ident-conv-th-2')
figure, plot(info1.F, 'b-'), hold on, plot(info2.F, 'r-')
legend('identification from (u, y)    .', 'identification from (y, u)    .'), print_fig('id
```

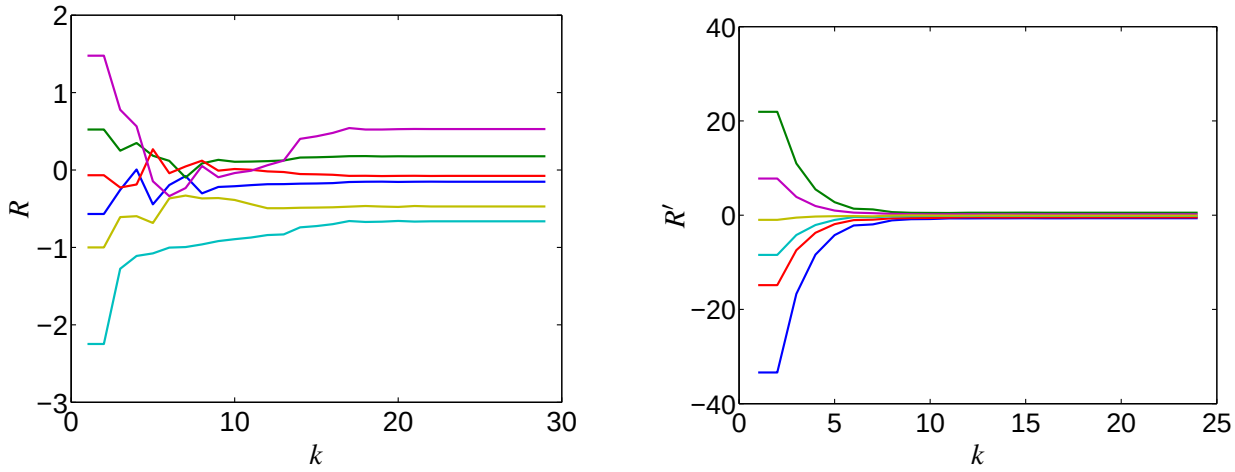

Figure 2: Convergence of the model parameters R and R' in the course of the optimization method.

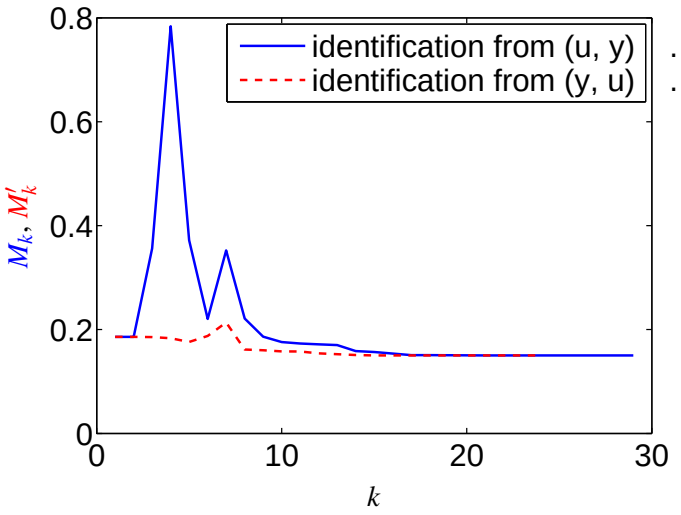

Figure 3: Misfits $M_k := M(w_{\mathrm{d}}, \widehat{\mathscr{B}}_k)$ and $M'_k := M(w'_{\mathrm{d}}, \widehat{\mathscr{B}}'_k)$ as a function of the iteration step k.

5 L_2-optimal model reduction

In this section, the `ident` function is used for solution of a (finite horizon) L_2-optimal model reduction problem [6, Section 3.3]. Consider a discrete-time linear time-invariant system $\mathscr{B}$ with an input/output partition $w = (u, y)$ and let $H(t) \in \mathbb{R}^{\mathtt{p}\times\mathtt{m}}$ be the tth sample of the impulse response of $\mathscr{B}$. The finite horizon-T L_2 norm of $\mathscr{B}$ (w.r.t. the given input/output partition $w = (u, y)$) is defined as

$$\|\mathscr{B}\|_{2,T} := \|H\|_{2,T} := \sqrt{\sum_{t=0}^{T} \|H(t)\|_{\mathrm{F}}^2},$$

where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm.

Problem 5 (L_2-optimal model reduction problem [6])**.** Given a linear time-invariant system $\mathscr{B}_{\mathrm{d}} \in \mathscr{L}^{q}_{\mathtt{m},\ell}$ and a complexity specification $\ell_{\mathrm{red}} < \ell$, find an optimal approximation of $\mathscr{B}_{\mathrm{d}}$ with bounded complexity $(\mathtt{m}, \ell_{\mathrm{red}})$

$$\widehat{\mathscr{B}}^* := \arg\min_{\widehat{\mathscr{B}}} \|\mathscr{B}_{\mathrm{d}} - \widehat{\mathscr{B}}\|_{2,T} \quad \text{subject to} \quad \widehat{\mathscr{B}} \in \mathscr{L}^{q}_{\mathtt{m},\ell_{\mathrm{red}}}.$$

The link between L_2-optimal model reduction and system identification is due to equivalence of an impulse response of $\mathscr{B}$ and a set of responses of a related autonomous system $\mathscr{B}_{\mathrm{aut}}$.

Proposition 6. *The shifted impulse response*

$$\sigma H = \big(H(1), H(2), \dots\big)$$

of $\mathscr{B} = \mathscr{B}(A,B,C,D)$ *is equal to the matrix* $\begin{bmatrix} y_1 & \cdots & y_{\mathtt{m}} \end{bmatrix}$ *of responses* $y_1, \dots, y_{\mathtt{m}}$ *of the autonomous system* $\mathscr{B}_{\text{aut}} = \mathscr{B}(A,C)$ *to initial conditions* $b_1, \dots, b_{\mathtt{m}}$, *where* $B = \begin{bmatrix} b_1 & \cdots & b_{\mathtt{m}} \end{bmatrix}$.

Proposition 6 and the fact that $H(0)$ is equal to the feed through parameter D of the input/state/output representation $\mathscr{B}(A,B,C,D)$ imply that the L_2-optimal model reduction problem is equivalent to an L_2-optimal identification problem for an autonomous linear time-invariant system from the set of $\mathtt{m}$ responses $h_1, \dots, h_{\mathtt{m}}$.

⟨*model reduction by* `ident`⟩≡
```
w = y(2:end, :, :); m = 0; ell = ell_red; ⟨use ident⟩, m = size(y, 3);
```
$\langle (\mathscr{B}(\widehat{A},\widehat{C}), \widehat{X}_{\text{ini}}) \mapsto \mathscr{B}(\widehat{A},\widehat{B},\widehat{C},\widehat{D}) \rangle$

A input/state/output representation of the reduced order model is obtained from the identified autonomous system $\mathscr{B}(\widehat{A},\widehat{C})$, initial conditions $X_{\text{ini}} = \begin{bmatrix} \widehat{x}_{\text{ini},1} & \cdots & \widehat{x}_{\text{ini},\mathtt{m}} \end{bmatrix}$, and $H(0)$ as follows:

$$\mathscr{B}\big(\widehat{A}, X_{\text{ini}}, \widehat{C}, H(0)\big).$$

$\langle (\mathscr{B}(\widehat{A},\widehat{C}), \widehat{X}_{\text{ini}}) \mapsto \mathscr{B}(\widehat{A},\widehat{B},\widehat{C},\widehat{D}) \rangle \equiv$
```
if isa(sysh, 'lti'), sysh.b = xini; sysh.d = reshape(y(1, :, :), p, m); end
```

A benchmark model reduction problem

As a test example, consider a mechanical system consisting of N point masses connected in a chain by ideal springs and ideal dampers. The first and the last masses are also connected to walls. Friction force, proportional to the speed, acts on all masses. The parameters of the system are the masses $m_1, \dots, m_N$, the spring and damper coefficients $k_0, k_1, \dots, k_N$ and $d_0, d_1, \dots, d_N$, the length δ of the springs, and the friction coefficient f, which are all nonnegative numbers. Setting the coefficients of the most left and most right springs and dampers to zero has the effect of detaching the chain of masses from the left and right walls.

Denoting by p_i the position of the ith point mass with respect to the left wall, the system dynamics is described by the system of differential equations

$$\begin{aligned}
m_1\ddot{p}_1 &= -(k_0+k_1)p_1 + k_1 p_2 - (d_0+d_1+f)\dot{p}_1 + d_1\dot{p}_2 + (k_0-k_1)\delta,\\
m_2\ddot{p}_2 &= k_1 p_1 - (k_1+k_2)p_2 + k_2 p_3 + d_1\dot{p}_1 - (d_1+d_2+f)\dot{p}_2 + d_2\dot{p}_3 + (k_1-k_2)\delta,\\
&\vdots\\
m_i\ddot{p}_i &= k_{i-1}p_{i-1} - (k_{i-1}+k_i)p_i + k_i p_{i+1} + d_{i-1}\dot{p}_{i-1} - (d_{i-1}+d_i+f)\dot{p}_i + d_i\dot{p}_{i+1} + (k_{i-1}-k_i)\delta,\\
&\vdots\\
m_N\ddot{p}_N &= k_{N-1}p_{N-1} - (k_{N-1}+k_N)p_N + k_{N-1}\dot{p}_{N-1} - (d_{N-1}+d_N+f)\dot{p}_N + (k_{N-1}+k_N N)\delta.
\end{aligned} \tag{SYS}$$

The positions y of the masses with indexes in the set $\mathscr{Y}$ are observed and external forces u are applied to the masses with indexes in the set $\mathscr{U}$. An input/state/output representation of the resulting model is given in Appendix C.

Numerical results

The input data for the model reduction methods is the first $T+1$ samples of the impulse response H of the full order system and the lag of the reduced order model. The value of T is selected so that, after T steps, the impulse response

has converged sufficiently close to zero. The L_2-optimal model reduction problem is solved by the `ident` function and the function `arl2` from the RARL2 package [12].

⟨*model reduction by* `arl2`⟩≡
```
addpath ~/mfiles/rarl2/arl2lib ~/mfiles/rarl2/boplib
arl2_options = arl2Options('real');
optim_options = optimset( ...
    'MaxFunEval', 1e4, 'MaxIter', 1e3, ...
    'TolFun', 1e-12, 'TolX', 1e-12, ...
    'GradObj', 'on', 'DerivativeCheck', 'off', 'Display', 'off');
try
  report = arl2(struct('type', 'coeff', ...
                       'value', permute(y(2:end, :, :), [2, 3, 1]), ...
                       'valinf', reshape(y(1, :, :), p, m)), ...
                        sys_ini, arl2_options, optim_options);
  sysh = report.sys;
catch, sysh = NaN; end
```

Note 7 (Using H vs $\mathscr{B}$ as input data). The `arl2` function accepts as an input either an input/state/output representation of the model or impulse response coefficients of the system. The latter option is used in the simulation examples.

Note 8 (Initial approximation). The optimization based methods `ident` and `arl2` are initialized with the suboptimal approximation computed by Kung's method [4], implemented in the function `fc2ss` of the RARL2 package.

⟨*model reduction by* `fc2ss`⟩≡
```
addpath ~/mfiles/rarl2/arl2lib;
sysh = fc2ss(permute(y(2:end, :, :), [2, 3, 1]), reshape(y(1, :, :), p, m), n_red);
```

The approximate model $\widehat{\mathscr{B}}$ is evaluated by computing the finite horizon-T relative L_2 approximation error

$$e = \frac{\|\mathscr{B} - \widehat{\mathscr{B}}\|_{2,T}}{\|\mathscr{B}\|_{2,T}}.$$

⟨*evaluate*⟩≡
```
if isa(sysh, 'ss')
  yh = impulse(sysh, T); e = norm(y(:) - yh(:)) / norm(y(:));
  plot(yh(:, 1, 1), ls)
else, e = NaN; end
```

⟨`test_mod_red`⟩≡
```
sys0 = sys(N, opt); [p, m] = size(sys0); n = size(sys0, 'order'); n_red = ell_red * p;
if exist('T')
  [y, t] = impulse(sys0); y = y(1:T + 1, :, :);
else
  [y, t] = impulse(sys0); T = length(y) - 1;
end
ts = t(2) - t(1);
figure(1), plot(y(:, 1, 1), '-k'), hold on
tic, ⟨model reduction by fc2ss⟩, t_kung = toc;  ls = '-.r'; ⟨evaluate⟩, e_kung = e; sys_ini = sysh;
opt.sys0 = ss(sys_ini.a, [], sys_ini.c, [], -1);
tic, ⟨model reduction by arl2⟩,  t_arl2 = toc;  ls = ':r'; ⟨evaluate⟩, e_arl2 = e; sys_arl2 = sysh;
tic, ⟨model reduction by ident⟩, t_ident = toc; ls = '-b'; ⟨evaluate⟩, e_ident = e; sys_slra = sysh;
res = [{'method'; 'e'; 't'} [{'kung' 'arl2' 'ident'};
              num2cell([e_kung e_arl2 e_ident])
              num2cell([t_kung t_arl2 t_ident])]];
⟨print results⟩
ax = axis; axis([0 T ax(3:4)]), legend('data    .', 'kung', 'arl2', 'ident'),
print_fig(ex)
figure,
Y0 = abs(fft(y)); W0 = (0:T)' * (1 / ts);
semilogx(W0, db(Y0(:)), '-k'), hold on
```

```
Yini = abs(fft(impulse(sys_ini, T))); Wini = (0:T)' * (1 / ts);
semilogx(Wini, db(Yini(:)), '-.r'), hold on

Yarl2 = abs(fft(impulse(sys_arl2, T))); Warl2 = (0:T)' * (1 / ts);
semilogx(Warl2, db(Yarl2(:)), ':r'), hold on

Yslra = abs(fft(impulse(sys_slra, T))); Wslra = (0:T)' * (1 / ts);
semilogx(Wslra, db(Yslra(:)), '-b'), hold on

axis([2 10^2 -20 22])
print_fig([ex '-frf'])
```

⟨*print results*⟩≡
```
eval(['!rm ' ex]), diary(ex), disp(res), pause(.1), diary off,
```

Example 9. In an example with model parameters

$$\begin{aligned} N = 10, \quad f = 0.1, \quad \delta = 1, \quad \mathscr{U} = 1, \quad \mathscr{Y} = 5, \quad \ell = 3, \\ \begin{bmatrix} m_1 & m_2 & \dots & m_{N-1} & m_N \end{bmatrix} = \begin{bmatrix} 10 & 9 & \cdots & 2 & 1 \end{bmatrix}, \\ \begin{bmatrix} k_0 & k_1 & \dots & k_{N-1} & k_N \end{bmatrix} = \begin{bmatrix} 0.5 & 9 & \cdots & 1 & 0.1 \end{bmatrix}, \\ \begin{bmatrix} d_0 & d_1 & \dots & d_{N-1} & d_N \end{bmatrix} = 0.2 \begin{bmatrix} 0 & 9 & \cdots & 1 & 1 \end{bmatrix}, \end{aligned}$$

⟨*Model reduction example 1*⟩≡
```
clear all, close all
ex = 'ex-mod-red1'; N = 10; opt.f = 0.1; ell_red = 3;
opt.M = (N:-1:1)';
opt.K = [0.5; ones(N - 1, 1); .1];
opt.D = 0.2 * [0; ones(N - 1, 1); 1]; test_mod_red
```

the obtained results are:

```
'method'    'kung'      'arl2'      'ident'
'e'         [0.4380]    [0.4263]    [0.4263]
't'         [0.1035]    [2.3251]    [0.0579]
```

The high-order and reduced order systems' impulse and frequency responses are shown in Figure 4. The `arl2` and `ident` functions compute the same approximation, however, the computation time required by the `ident` function is much smaller. This is due to the efficient implementation of the structured low-rank approximation solver.

Example 10. In the setup of Example 9, detaching the system from the right wall ($k_N = d_N = 0$) gives qualitatively similar results:

```
'method'    'kung'      'arl2'      'ident'
'e'         [0.8368]    [0.5950]    [0.5950]
't'         [0.0804]    [3.2495]    [0.0789]
```

The impulse and frequency responses are shown in Figure 5, right.

⟨*Model reduction example 2*⟩≡
```
clear all, close all
ex = 'ex-mod-red2'; N = 15; opt.f = 0.1; ell_red = 3;
opt.M = (N:-1:1)';
opt.K = [0.5; ones(N - 1, 1); 0];
opt.D = 0.2 * [0; ones(N - 1, 1); 0]; test_mod_red
```

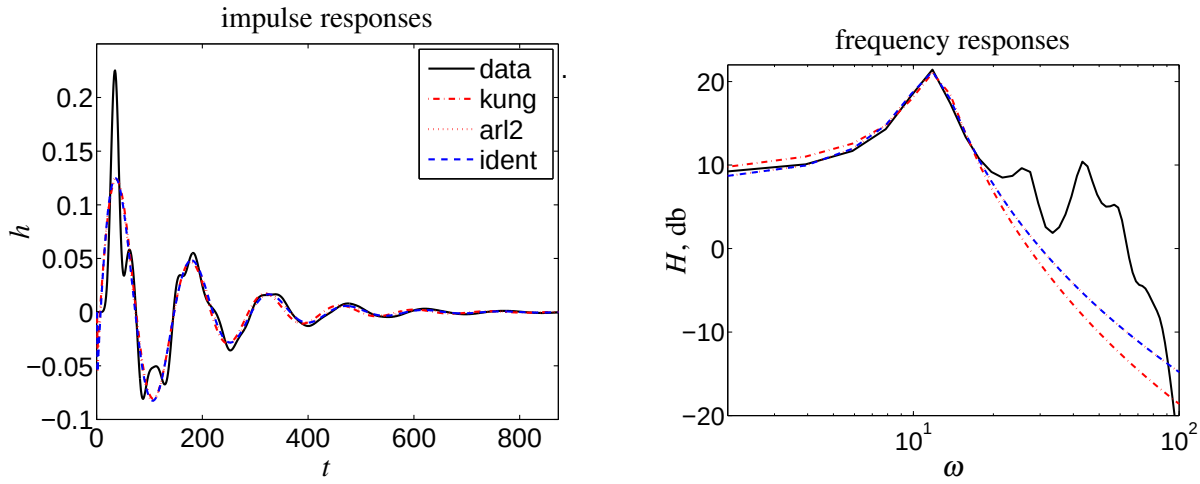

Figure 4: Comparison of the high order and reduced order models in example 9.

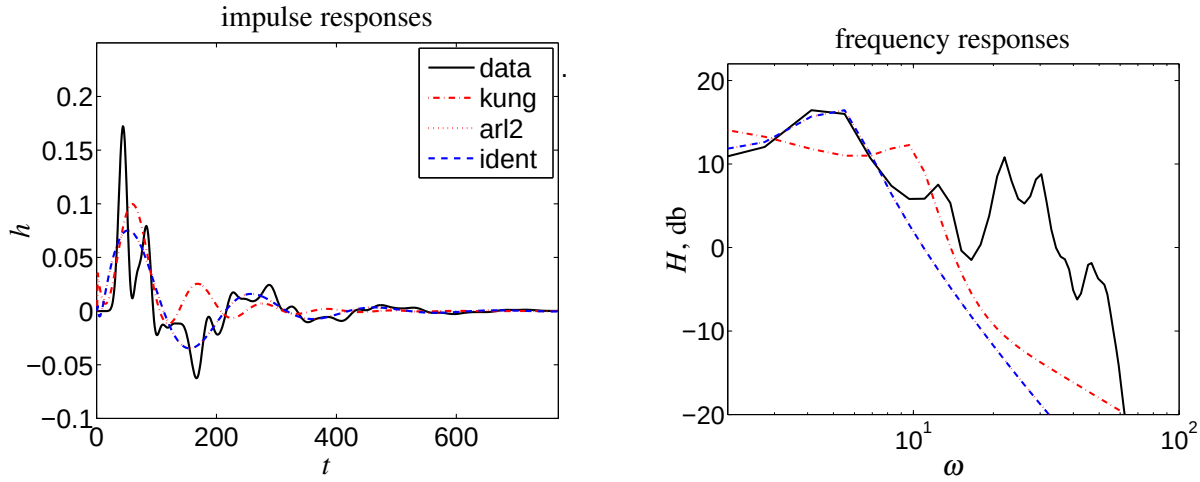

Figure 5: Comparison of the high order and reduced order models in example 10.

Example 11. In the setup of Example 9, detaching the system from both right and left walls ($k_N = d_N = k_0 = d_0 = 0$) introduced a free motion, *i.e.*, an unstable mode. This gives qualitatively different result in the approximations (see Figure 6):

```
'method'    'kung'      'arl2'      'ident'
'e'         [0.6830]    [0.2244]    [0.0337]
't'         [0.0561]    [1.9533]    [0.0564]
```

The reason for the poor approximation of `arl2` is that this method imposes stability of the approximation.

⟨*Model reduction example 3*⟩≡

```
clear all, close all
ex = 'ex-mod-red3'; N = 15; opt.f = 0.1; ell_red = 3;
opt.M = (N:-1:1)';
opt.K = [0; ones(N - 1, 1); 0];
opt.D = 0.2 * [0; ones(N - 1, 1); 0]; test_mod_red
```

6 Identification from multiple trajectories of different experiments

In this section, the multiple experiments data feature of the identification method is tested. An alternative method to `ident`, used for comparison, is the function `pem` from the System Identification Toolbox of Matlab. The data is generated in the output-error setup and the methods are called with the corresponding options:

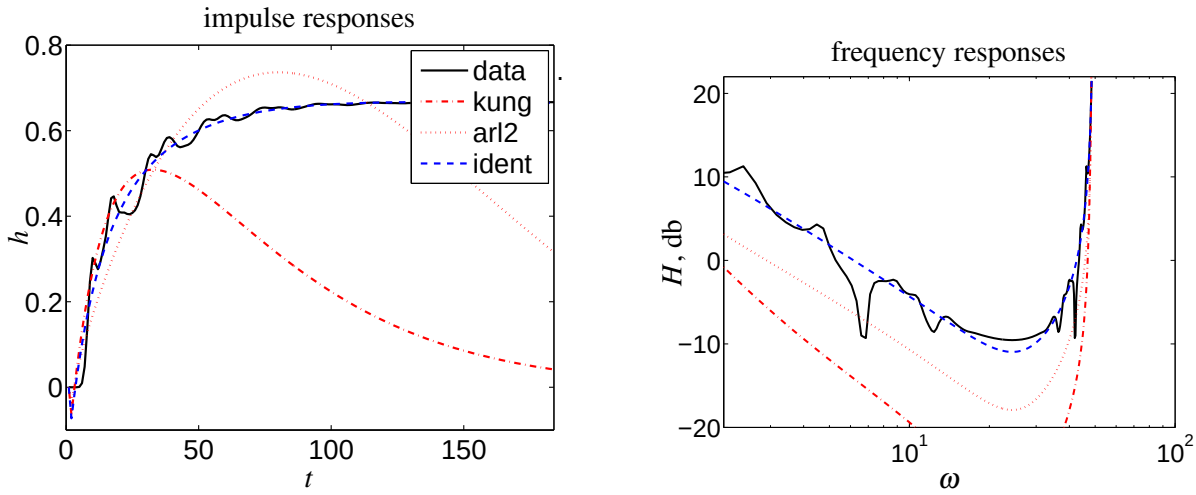

Figure 6: Comparison of the high order and reduced order models in example 10.

- `opt.exct = 1:m` for `ident` and
- `'dist', 'none'` for `pem`.

⟨*use* `pem`⟩≡
```
try,   [sysh, wh] = pem_(iddata(y, u), p * ell, 'dist', 'none', 'nk', zeros(1, m));
catch, sysh = NaN; end
```

⟨`pem_`⟩≡
```
function [sys, wh, data] = pem_(data, varargin)
data = misdata(data, 10);
sys  = pem(data, varargin{:});
if nargout > 1
    out = compare(data, sys);
    if iscell(data.u)
      for k = 1:length(data.u)
        wh{k} = [data.u{k} out{k}.y];
      end
    else, wh = [data.u out.y]; end
end
sys = ss(sys);
```

The true system is randomly generated with `drss` and the true data consists of random trajectories of the true system. The input is normally distributed with zero mean and identity covariance matrix. The data used for identification is a noise corrupted version of the true trajectory, where the output noise is zero mean white Gaussian. The simulation parameters are: number of inputs, specified by the variable `m`; number of outputs, specified by the variable `p`; lag of the system, specified by the variable `ell`; number of time series, specified by the variable `N`; number of samples of the time series, specified by the variable `T`; and a noise standard deviation, specified by the variable `NL`.

Depending on the value of `opt.ini`, the initial conditions are normally distributed with zero mean and identity covariance matrix or zeros.

⟨*simulate true trajectory*⟩≡
```
sys0 = drss(n, p, m); N = length(T);
if isfield(opt, 'wini') && opt.wini ~= 0;
  xini = randn(p * ell, 1);
else
  xini = zeros(p * ell, 1);
end
for k = 1:N,
  u0{k} = randn(T(k), m); % [eye(m); zeros(T(k) - m, m)]; % randn, ones
  y0{k} = lsim(sys0, u0{k}, [], xini); w0{k} = [u0{k} y0{k}];
end
```

⟨*add noise*⟩≡
```
if eiv, opt.exct = []; else opt.exct = 1:m; end
v = ones(m + p, 1); v(opt.exct) = 0;
for k = 1:N
  wn = randn(size(w0{k})) * diag(v);
  w{k} = w0{k} + (nl * norm(w0{k}) / norm(wn)) * wn;
  u{k} = w{k}(:, 1:m); y{k} = w{k}(:, (m + 1):end);
end
if N == 1, w0 = w0{1}; w = w{1}; u = u{1}; y = y{1}; end
```

The identified models by the `ident` and `pem` functions are compared with respect to their relative output error

$$e = \sum_{i=1}^{N} \frac{\|y^i - \hat{y}^i\|_2^2}{\|y^i\|_2^2}$$

⟨*evaluate multiple experiments*⟩≡
```
if isa(sysh, 'lti') || isa(sysh, 'idss')
  ny2 = 0; for k = 1:N, ny2 = ny2 + norm(w{k}(:, (m + 1):end)) ^ 2; end
  M = misfit(w, sysh, opt); e = M / ny2;
else, e = NaN; end
```

Example 12. In an example with parameters

⟨*Multiple experiments example 1*⟩≡
```
ex = 'mult-traj-ex1'; ell = 3; p = 1; m = 1; T = [500 1000];
K = 5; NL = linspace(0, 0.1, K); test_mult_traj
```

the results obtained

```
    'nl'          [          0]    [0.0250]    [0.0500]    [0.0750]    [0.1000]
    'e ident'     [5.5929e-32]    [0.1137]    [0.3360]    [0.5445]    [0.6760]
    'e pem'       [3.4563e-32]    [0.1137]    [0.3360]    [0.5445]    [0.6760]
```

show that the `ident` and `pem` methods achieve the same approximation error. (Difference of less than 10^{-4} in the relative approximation errors can be attributed to exiting the optimization with the default convergence tolerance of 10^{-5}.) Although, in this example `ident` and `pem` are functionally equivalent (compute the same answer), `ident` is on the average 28 times faster than `pem`.

Example 13. In the case of $N = 50$ data sets of short ($T = 20$) trajectories

⟨*Multiple experiments example 2*⟩≡
```
ex = 'mult-traj-ex2'; ell = 2; p = 1; m = 1; T = 20 * ones(1, 50);
K = 5; NL = linspace(0, 0.1, K); test_mult_traj
```

```
    'nl'          [          0]    [0.0250]    [0.0500]    [0.0750]    [0.1000]
    'e ident'     [2.1716e-32]    [0.0015]    [0.0057]    [0.0130]    [0.0225]
    'e pem'       [1.9819e-32]    [0.0015]    [0.0057]    [0.0127]    [0.0225]
```

the obtained results are again identical. In this case the `ident` function is on the average 59 times faster than `pem`. Note that the theoretical computational complexity of the structured low-rank approximation method, used by `ident`, scales linearly in N, see [23]. Empirical results suggest that the algorithms used in the implementation of the `pem` function scale worse with the number of time series.

⟨`test_mult_traj`⟩≡
```
eiv = 0; n = ell * p; opt = struct; ⟨simulate true trajectory⟩
for i = 1:K
  nl = NL(i); ⟨add noise⟩
  tic, ⟨use pem⟩, t_pem(i) = toc; ⟨evaluate multiple experiments⟩, e_pem(i) = e;
  tic, ⟨use ident⟩, t_ident(i) = toc; ⟨evaluate multiple experiments⟩, e_ident(i) = e;
end
res = [{'nl'; 'e ident'; 'e pem'} num2cell([NL; e_ident; e_pem])];
t_ident = mean(t_ident, 2), t_pem = mean(t_pem, 2), ⟨print results⟩
bode(sys0, '-k', sysh, '-b')
%bode(sys0d - sys_ini, '-.r', sys0d - sys_arl2, ':r', sys0d - sys_slra, '-b', {1e-2 1})
```

7 System identification with missing data

This section shows examples of identification problems with missing data. The data w is a noisy T samples long random trajectory of a single-input single-output linear time-invariant system $\bar{\mathscr{B}} = \mathscr{B}(\bar{R})$ with lag $\ell = 2$. Input, output, or both samples are missing at moments of time $t \in \mathscr{T}_{\mathrm{m}}$. The true model parameters are

$$\bar{R}_0 = \begin{bmatrix} 0.3 & 0.7 \end{bmatrix}, \quad \bar{R}_1 = \begin{bmatrix} 1 & -1.4 \end{bmatrix}, \quad \bar{R}_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}. \tag{$*$}$$

The approximation accuracy is measured by the angle

$$e = \angle(\bar{R}, \widehat{R}) = \cos^{-1}\left(\frac{\bar{R}^\top \widehat{R}}{\|\bar{R}\|\|\widehat{R}\|}\right),$$

between the true $\bar{R}$ and estimated $\widehat{R}$ parameter vectors.

⟨*evaluate identified model*⟩≡
```
if isa(sysh, 'lti') % ~isnan(wh),
  [num, den] = tfdata(sysh, 'vec');
  Rh = zeros(1, 6); Rh(1:ell + 1) = - fliplr(num); Rh(ell + 2:end) = fliplr(den);
  e = real(acos( dot(R0_vec, Rh) / norm(R0_vec) / norm(Rh) ));
else e = NaN; end
```

Assuming single-input single output system:

⟨*true system*⟩≡
```
R0(1, :, :) = [0.3 0.7]; R0(2, :, :) = [1 -1.4]; R0(3, :, :) = [0 1];
[ell1, p, q] = size(R0); ell = ell1 - 1; m = q - p;
Q0 = - R0(:, :, 1:m); P0 = R0(:, :, m + 1:q);
sys0 = tf(fliplr(Q0(:)'), fliplr(P0(:)'), 1);
R0_vec = shiftdim(R0, 3); R0_vec = R0_vec(:)';
```

⟨*simulate* $\bar{w}$⟩≡
```
u0 = rand(T, m); y0 = lsim(sys0, u0); w0 = [u0 y0];
```

In the Control Toolbox, the coefficients of the numerator and denominator polynomials are ordered in decreasing degrees, while in `ident` the parameters of the kernel representation are ordered in increasing degrees. This requires interchanging the order when passing parameters to and receiving parameters from the Control Toolbox functions.

⟨*add noise and missing data*⟩≡
```
yt = randn(T, q - m);
w = w0 + nl * [zeros(T, m) yt] / norm(yt, 'fro') * norm(y0, 'fro'); w(Tm, m_io) = NaN;
u = w(:, 1:m); y = w(:, m + 1:end);
```

An alternative method for solving the identification problem considered in this section is proposed in [14]. This method uses a frequency domain approach [16]. A Matlab implementation of the algorithm (called below `sysid`) was kindly provided by the authors and is used below for verification of the results obtained with `ident`.

⟨*use* `sysid`⟩≡
```
addpath ~/mfiles/missing-data-dynamic/rik/
try    [sysh, wh_m] = sysid_missing_data_rik(w(:, 1), w(:, 2), ell, [0 std(yt)]); % wh = ze
catch, sysh = NaN; end
```

CAPTAIN toolbox [22]

⟨*use* `captain`⟩≡
```
addpath ~/mfiles/captain/
try    [th, stats, e, var, Ps, Pc, yh] = riv([w(:, 2), w(:, 1)], [ell ell 1 0]);
       [Ph, Qh] = getpar(th); sysh = tf(Qh, Ph, -1); % wh = zeros(size(w)); wh(Tm, 1) = yh(
catch, wh = NaN; sysh = NaN; end
```

The simulation parameters in the experiments are the number of samples `T`; the set of missing values $\mathcal{T}_{\mathrm{m}}$, specified by a variable `Tm`; and a noise standard deviation interval, specified by a vector `NL`. The reported results show the error e for the compared methods and for the different noise levels specified in `NL`. Three experiments are done for different distribution of the missing values: sequential, periodic, and random. Both input and output values are missing. A `NaN` value in the table of results indicates that the corresponding method fails in (or does not apply to) this case.

Example 14 (Sequential missing data samples). In an example with sequential missing data,

⟨*Missing data example 1*⟩≡
```
ex = 'missing-data-ex1';
T = 100; K = 5; NL = linspace(0, 0.01, K); Tm = [30:70]; test_missing_data
```

the `ident` and `sysid` functions achieve comparable accuracy

```
'nl'       [    0]    [0.0025]    [0.0050]    [0.0075]    [0.0100]
'ident'    [2e-08]    [0.0012]    [0.0078]    [0.0017]    [0.0067]
'sysid'    [    0]    [0.0012]    [0.0078]    [0.0014]    [0.0065]
```

The `ident` function is 9 time faster than `sysid`.

Example 15 (Periodic missing data samples). Similar results are obtained in the case of periodic missing data for small noise levels

⟨*Missing data example 2*⟩≡
```
ex = 'missing-data-ex2';
T = 100; K = 5; NL = linspace(0, 0.01, K); Tm = [30:3:70]; test_missing_data
```

```
'nl'       [    0]    [0.0025]    [0.0050]    [0.0075]    [0.0100]
'ident'    [4e-07]    [0.0014]    [0.0058]    [0.0077]    [0.0068]
'sysid'    [    0]    [0.0014]    [0.0073]    [0.0082]    [0.0077]
```

In this example, the `ident` function is 5 time faster than `sysid`.

Example 16 (Randomly distributed missing data samples). Finally, results for a simulation example with $T = 1000$ data points from which 600 are randomly missing are shown:

⟨*Missing data example 3*⟩≡
```
ex = 'missing-data-ex3';
T = 1000; K = 5; NL = linspace(0, 0.01, K); Tm = sort(randperm(T, 600));
test_missing_data
```

```
'nl'       [    0]    [0.0025]    [0.0050]    [0.0075]    [0.0100]
'ident'    [9e-05]    [0.0029]    [0.0087]    [0.0028]    [0.0123]
'sysid'    [  NaN]    [   NaN]    [   NaN]    [   NaN]    [   NaN]
```

⟨test_missing_data⟩≡
```
m_io = [1 2]; randn('seed', 0); rand('seed', 0); warning off
⟨true system⟩, ⟨simulate w̄⟩
opt.exct = 1:m; opt.sys0 = ss(sys0); % figure(1), plot(w0(:, 2), 'r'), hold on
for i = 1:K
  nl = NL(i); ⟨add noise and missing data⟩,              %plot(w(:, 2),  ':k')
  tic, ⟨use ident⟩, t_ident(i) = toc; ⟨evaluate identified model⟩, e_ident(i) = e; %plot(wh(:, 2), '-b
  tic, ⟨use sysid⟩, t_sysid(i) = toc; ⟨evaluate identified model⟩, e_sysid(i) = e; %plot(wh(:, 2), '-.
end
labels = {'nl'; 'ident'; 'sysid'};
res = [labels, num2cell([NL; e_ident; e_sysid])];
t_ident = mean(t_ident), t_sysid = mean(t_sysid), ⟨print results⟩,
bode(sys0, '-k', sysh, '-b')
%bode(sys0d - sys_ini, '-.r', sys0d - sys_arl2, ':r', sys0d - sys_slra, '-b', {1e-2 1})
```

Discussion of the system identification results with missing data: The problems solved by the `ident` and `sysid` functions are equivalent. In the stochastic setting of Section 2.3, they are errors-in-variables identification problems with missing data. Examples 14 and 15 numerically confirm that the results computed by the two methods coincide for small noise levels and a small number of missing values. Since the problems are nonconvex, however, for high noise levels the methods may converge to different locally optimal solutions.

In addition, there is an important algorithmic differences between `ident` and `sysid`, which explains the results of Example 16. In `ident`, the initial conditions are eliminated analytically at the level of the misfit computation (see Section 3). In `sysid`, the initial conditions are included in the parameter vector of the nonlinear least squares problem (see [14]). This results in a larger optimization problem, solved by `sysid` than the one solved by `ident`. Since in `sysid` a transient response is added for every missing data point, the number of initial conditions to-be-estimated is growing with the increase of number of missing data points. In contrast, the dimension of the nonlinear least squares problem solved by `ident` is fixed. The difference in performance between `ident` and `sysid` due to the different way of dealing with the initial conditions becomes pronounced in problems with more missing values. This is illustrated in Example 16, where the problem is no longer solvable by `sysid`. Detailed statistical and numerical analysis of the identification methods in the case of missing data will be presented elsewhere.

8 Performance on real-life data

In this section, the performance of the `ident` function is tested on benchmark problems from the data base for system identification DAISY [13]. The data come from a number of applications in process industry, electrical, mechanical, and environmental engineering. A validation criterion is chosen that measures the predictive power of the model: how accurate the model can fit a part of the data that is not used for identification. Values for the identification methods' parameters that correspond to this validation criterion are chosen and fixed for all data sets. Although "better" results may be obtained by preprocessing of the data and tuning "hyper parameters" (model structure and identification criterion) this is not done. Our tests reflect the view that the identification process should be as automatic as possible, *i.e.*, it should be done with as little human interaction as possible. The methods are applied on all data sets choosing only the model class (specified by a bound on the model complexity).

The considered data sets are listed in Table 1. References and details about the nature and origin of the data is given in [13]. The data is preprocessed by centering it. The model's lag ℓ is chosen manually for each data set from the complexity–accuracy trade-off curve (misfit as a function of the lag). For comparison, also the automatically selected lags ℓ' by the `pem` function are shown in the table. (In the first two data sets, automatic order section by `pem` is not possible due to a small number of samples.)

The data $w_{\mathrm{d}} = (u_{\mathrm{d}}, y_{\mathrm{d}})$ in all examples is split into identification and validation parts. The first 70% of the data, denoted w_{idt}, are used for identification, and the remaining 30% of the data, denoted w_{val}, are used for validation. A model $\widehat{\mathscr{B}}$ is identified from w_{idt} by an identification method and is validated on w_{val} by the validation criterion defined next. The model class is linear time-invariant systems with a bound ℓ on the lag.

The validation criterion is the "simulation fit" computed by the function `compare` of the System Identification Toolbox. This choice is motivated by the fact that in this case the results obtained by the `ident` function can be validated, using the `pem` function. Correspondingly, the `ident` and `pem` functions are applied setting options for output error identification:

- `opt.exct = 1:m` for `ident` and
- `'dist', 'none'` for `pem`.

For a given time series $w_{\mathrm{d}} = (u_{\mathrm{d}}, y_{\mathrm{d}})$ and a model $\mathscr{B}$, the approximation $\widehat{y}$ of y in $\mathscr{B}$ is defined as follows:

$$\widehat{y}\big((u_{\mathrm{d}}, y_{\mathrm{d}}), \mathscr{B}\big) := \min_{\widehat{y}} \|y_{\mathrm{d}} - \widehat{y}\| \quad \text{subject to} \quad \mathrm{col}(u_{\mathrm{d}}, \widehat{y}) \in \mathscr{B}.$$

(The optimization is carried over the initial conditions that generate $\widehat{y}$ from the given input u_{d}.) With this notation, the fit of w_{d} by $\mathscr{B}$ is defined as

$$F(w_{\mathrm{d}}, \mathscr{B}) := 100 \frac{\max\big(0, 1 - \|y_{\mathrm{d}} - \widehat{y}(w_{\mathrm{d}}, \mathscr{B})\|\big)}{\|y_{\mathrm{d}} - \sum_{t=1}^{T} y_{\mathrm{d}}(t)/T\|}.$$

#	Data set name	T	m	p	ℓ	ℓ'
1	Data of a simulation of the western basin of Lake Erie	57	5	2	1	—
2	Data of ethane-ethylene distillation column	90	5	3	1	—
3	Heating system	801	1	1	2	1
4	Data from an industrial dryer (Cambridge Control Ltd)	867	3	3	1	2
5	Data of a laboratory setup acting like a hair dryer	1000	1	1	5	1
6	Data of the ball-and-beam setup in SISTA	1000	1	1	2	3
7	Wing flutter data	1024	1	1	5	2
8	Data from a flexible robot arm	1024	1	1	4	6
9	Data of a glass furnace (Philips)	1247	3	6	1	4
10	Heat flow density through a two layer wall	1680	2	1	2	1
11	Simulation data of a pH neutralization process	2001	2	1	6	2
12	Data of a CD-player arm	2048	2	2	1	1
13	Data from a test setup of an industrial winding process	2500	5	2	2	1
14	Liquid-saturated steam heat exchanger	4000	1	1	2	3
15	Data from an industrial evaporator	6305	3	3	1	1
16	Continuous stirred tank reactor	7500	1	2	1	2
17	Model of a steam generator at Abbott Power Plant	9600	4	4	1	1

Table 1: Examples from DAISY. T—number of data points, m—number of inputs, p—number of outputs, ℓ—lag of the identified model.

The fitting criterion $F(w_{\mathrm{val}}, \widehat{\mathscr{B}})$ is compared for the models produced by `ident` and `pem`. The results are shown in Figure 7. In terms of accuracy, the `ident` and `pem` function show comparable performance. The differences in the results can be attributed to the nonconvexity of the optimization problem and the usage of different initial approximations (unstructured low-rank approximation for the `ident` function and a subspace identification method for the `pem` function). In terms of execution time, except for example 17, the `ident` function is faster than `pem`. It should be noted that in example 17, the accuracy achieved by `ident` is significantly higher on both identification and validation data than the one achieved by `pem`.

9 Conclusions

The paper presented a method for system identification in the behavioral setting, with the following salient features:

1. representation free problem formulation,
2. identification from multiple experiments,
3. specification of exact variables,
4. missing values in arbitrary variables and moments of time,
5. implementation in a literature programming style.

Application of the method for L_2 optimal model reduction, identification from multiple data sets of different length, identification from data with missing values, and benchmark problems from the DAISY dataset was considered and illustrated on numerical examples. The developed software package was compared with state-of-the-art system identification packages, with respect to accuracy and computational efficiency. Despite of its generality and flexibility, the developed software is functionally equivalent to and computationally faster than existing alternatives.

Acknowledgements

The structured low-rank approximation package [8], used in the implementation of the identification method, is a joint work with Konstantin Usevich. During the preparation of the manuscript, I had discussions with Peter Young,

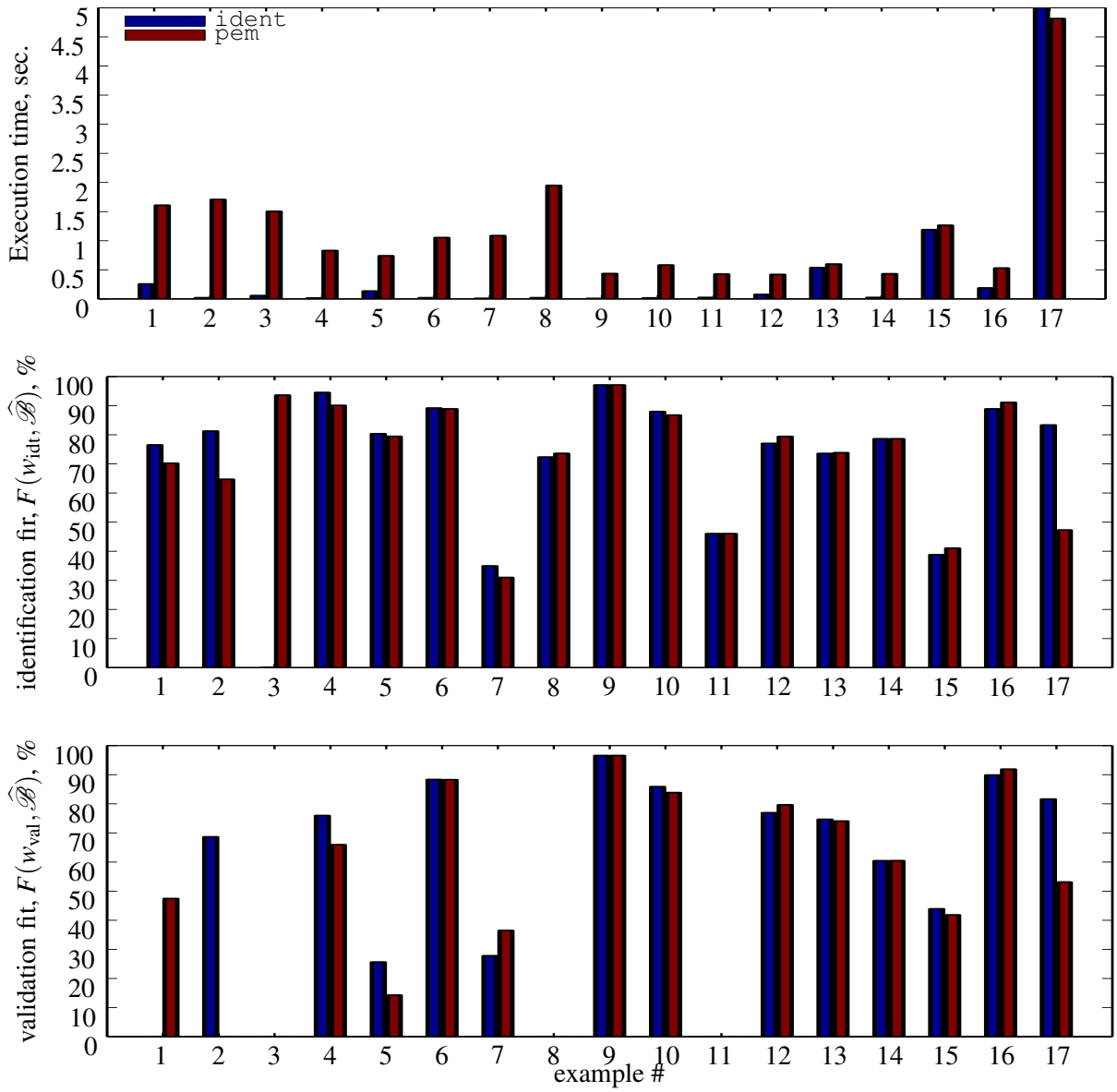

Figure 7: Results on all data sets by splitting of the data into first 70% for identification and remaining 30% for validation.

Rik Pintelon, Johan Schoukens, Lennart Ljung, and Martine Olivi on various aspects of the results presented in Sections 5–8.

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement number 258581 "Structured low-rank approximation: Theory, algorithms, and applications".

References

[1] M. Galassi et al. GNU scientific library reference manual. `http://www.gnu.org/software/gsl/`.

[2] D. Knuth. *Literate programming*. Cambridge University Press, 1992.

[3] A. Kukush, I. Markovsky, and S. Van Huffel. Consistency of the structured total least squares estimator in a multivariate errors-in-variables model. *J. Statist. Plann. Inference*, 133(2):315–358, 2005.

[4] S. Kung. A new identification method and model reduction algorithm via singular value decomposition. In *Proc. 12th Asilomar Conf. Circuits, Systems, Computers*, pages 705–714, Pacific Grove, 1978.

[5] L. Ljung. *System Identification Toolbox: User's guide*. The MathWorks.

[6] I. Markovsky. Structured low-rank approximation and its applications. *Automatica*, 44(4):891–909, 2008.

[7] I. Markovsky and P. Rapisarda. Data-driven simulation and control. *Int. J. Control*, 81(12):1946–1959, 2008.

[8] I. Markovsky and K. Usevich. Software for weighted structured low-rank approximation. Technical Report 339974, Univ. of Southampton, `http://eprints.soton.ac.uk/339974`, 2012.

[9] I. Markovsky, S. Van Huffel, and R. Pintelon. Block-Toeplitz/Hankel structured total least squares. *SIAM J. Matrix Anal. Appl.*, 26(4):1083–1099, 2005.

[10] I. Markovsky, J. C. Willems, S. Van Huffel, and B. De Moor. Software for approximate linear system identification. In *Proc. 44th Conf. on Decision and Control*, pages 1559–1564, Seville, Spain, 2005.

[11] I. Markovsky, J. C. Willems, S. Van Huffel, B. De Moor, and R. Pintelon. Application of structured total least squares for system identification and model reduction. *IEEE Trans. Automat. Control*, 50(10):1490–1500, 2005.

[12] Jean-Paul Marmorat and Martine Olivi. RARL2 software (realizations and rational approximation in L_2 norm). `http://www-sop.inria.fr/apics/RARL2`.

[13] B. De Moor, P. De Gersem, B. De Schutter, and W. Favoreel. DAISY: A database for identification of systems. *Journal A*, 38(3):4–5, 1997. Available from `http://homes.esat.kuleuven.be/~smc/daisy/`.

[14] R. Pintelon and J. Schoukens. Frequency domain system identification with missing data. *IEEE Trans. Automat. Control*, 45(2):364–369, February 2000.

[15] R. Pintelon and J. Schoukens. *System Identification: A Frequency Domain Approach*. IEEE Press, Piscataway, NJ, 2001.

[16] R. Pintelon and J. Schoukens. *System Identification: A Frequency Domain Approach*. IEEE Press, Piscataway, NJ, second edition, 2012.

[17] J. Polderman and J. C. Willems. *Introduction to mathematical systems theory*. Springer-Verlag, New York, 1998.

[18] N. Ramsey. Literate programming simplified. *IEEE Software*, 11:97–105, 1994.

[19] B. Roorda. *Global Total Least Squares—a method for the construction of open approximate models from vector time series*. PhD thesis, Tinbergen Institute, 1995.

[20] J. Schoukens, G. Vandersteen, Y. Rolain, and R. Pintelon. Frequency response function measurements using concatenated subrecords with arbitrary length. *IEEE Trans. on Instr. and Measurement*, 61(10):2682–2688, 2012.

[21] T. Söderström. Errors-in-variables methods in system identification. *Automatica*, 43:939–958, 2007.

[22] C. J. Taylor, D. J. Pedregal, P. C. Young, and W. Tych. Environmental time series analysis and forecasting with the CAPTAIN toolbox. *Environmental Modelling & Software*, 22:797—-814, 2007.

[23] K. Usevich and I. Markovsky. Variable projection for affinely structured low-rank approximation in weighted 2-norm. *J. Comput. Appl. Math.*, 2013. Available from `http://arxiv.org/abs/1211.3938`.

[24] P. Van Overschee and B. De Moor. *Subspace identification for linear systems: Theory, implementation, applications*. Kluwer, Boston, 1996.

[25] J. C. Willems. From time series to linear system—Part I. Finite dimensional linear time invariant systems, Part II. Exact modelling, Part III. Approximate modelling. *Automatica*, 22, 23:561–580, 675–694, 87–115, 1986, 1987.

[26] J. C. Willems. Paradigms and puzzles in the theory of dynamical systems. *IEEE Trans. Automat. Control*, 36(3):259–294, 1991.

A Input/output parameters of `ident` and `misfit`

- `w` is the given set of time series w_{d} — a real Matlab array of dimension $T \times q \times N$, where T is the number of samples, q is the number of variables, and N is the number of time series. In case of multiple experiments of different duration, `w` should be specified as a cell array with N cells, each one of which is a $T_i \times q$ matrix containing the ith time series, *i.e.*,

$$\mathtt{w(t,:,k)} = \left(w^k(t)\right)^\top \qquad \text{or} \qquad \mathtt{w\{k\}(t,:)} = \left(w^k(t)\right)^\top.$$

⟨*define* `T, q, N`⟩≡
```
if ~iscell(w)
  [T, q, N] = size(w); T = ones(N, 1) * T;
else
  N = length(w); for k = 1:N, [T(k), q] = size(w{k}); end, T = T(:);
end
```

- (`m`, `ell`) is the complexity specification (input dimension and lag).

- `opt` is a optional argument specifying exact variables, exact initial conditions, and options for the optimization solver, used by the `ident` function. The options are passed to the functions `ident` and `misfit` as fields of a structure.

  - `'exct'` (default value `[]`) — q-dimensional vector or N-dimensional cell array with q-dimensional vector elements, specifying the indices of the exact variables.

    ⟨*default* `opt.exct`⟩≡
    ```
    if ~isfield(opt, 'exct'), opt.exct = []; end
    ```

    Define a weight vector `s.w` in order to take into account the exact variables.

    ⟨*exact variables*⟩≡
    ```
    if ~isempty(opt.exct)
      if iscell(opt.exct)
        for i = 1:N s.w(:, i) = ones(q, 1); s.w(opt.exct{i}) = inf; end
      else
        s.w = ones(q, 1); s.w(opt.exct) = inf;
      end
    end
    ```

  - `'wini'` — specifies exact initial conditions. If `wini = 0`, exact zero initial conditions are specified, *i.e.*, $\mathrm{col}(0, \widehat{w}^k) \in \widehat{\mathscr{B}}$, More generally, `wini` $= w_{\mathrm{ini}}$ is an ℓ samples long trajectory (specified by an $\ell \times q \times N$ array or a N-dimensional cell array of $\ell \times q$ matrices), defining initial conditions for the time series $\widehat{w}$, *i.e.*, $\mathrm{col}(w^k_{\mathrm{ini}}, \widehat{w}^k) \in \widehat{\mathscr{B}}$.

    Allow convenient specification of zero initial conditions by scalar zero.

    ⟨*specification of zero initial conditions by* `0`⟩≡
    ```
    if iscell(w)
      if isempty(opt.wini)
        opt.wini = cell(1, N);
      elseif ~iscell(opt.wini) & (opt.wini == 0)
        for k = 1:N, wini{k} = zeros(ell, q); end, opt.wini = wini;
    ```

```
  else
    for k = 1:N
      if opt.wini{k} == 0, opt.wini{k} = zeros(ell, q); end
    end
  end
elseif opt.wini == 0
  opt.wini = zeros(ell, q, N);
end
```

Augment the data w_{d} with w_{ini} and modify accordingly the structure specification and the number of samples T. Specify that w_{ini} are exact by modifying the weight vector.

⟨*include* `opt.wini` *in* `w` *and modify* `s, T`⟩≡

```
if isfield(opt, 'wini')
  ⟨specification of zero initial conditions by 0⟩
  if isfield(s, 'w'), s = rmfield(s, 'w'); end
  if ~iscell(opt.wini) && ~isempty(opt.wini)
    W = ones(T, q, N); W(:, opt.exct, :) = inf;
    s.n = s.n + ell; T = T + ell;
    s.w = [inf * ones(ell, q, N); W];
    w   = [opt.wini; w];
  elseif iscell(opt.wini) && ~isempty(cell2mat(opt.wini))
    for k = 1:N
      W = ones(T(k), q); W(:, opt.exct{k}) = inf;
      if ~isempty(opt.wini{k})
        s.n(k) = s.n(k) + ell; T(k) = T(k) + ell;
        s.w{k} = [inf * ones(ell, q); W];
        w{k}   = [opt.wini{k}; w{k}];
      else
        s.w{k} = W;
      end
    end
  end
end
```

The approximation $\widehat{w}$ computed by `slra` includes the prefix trajectory w_{ini}. It is removed for the approximation returned to the used.

⟨*remove* `opt.wini` *from* `wh`⟩≡

```
if isfield(opt, 'wini')
  if ~iscell(opt.wini) && ~isempty(opt.wini)
    wh = wh(ell + 1:end, :, :);
  elseif iscell(opt.wini) && ~isempty(cell2mat(opt.wini))
    for k = 1:N
      if ~isempty(opt.wini{k})
        wh{k} = wh{k}(ell + 1:end, :);
      end
    end
  end
end
```

– `'sys0'` — initial approximation: an input/state/output representation of a system, given as an `ss` object, with `m` inputs, `p` := $q-$`m` outputs, and order `n` := ℓ`p`. Default value is computed by the `slra` function, using unstructured low-rank approximation.

⟨*initial approximation*⟩≡

```
if isfield(opt, 'sys0') && isa(opt.sys0, 'lti'), opt.Rini = ss2r(opt.sys0); end
```

– Arguments allowing the user to specify different optimization algorithms (`'solver'` and `'method'`), control the displayed information (`'disp'`), and change the convergence criteria are described in the structured low-rank approximation user's manual [8].

- `sysh` is an input/state/output representation of the identified or validated system $\widehat{\mathscr{B}}$, given by an `ss` object.
- `info` is a structure, containing exit information from the structured low-rank approximation solver: `info.M` is the misfit $M(w_{\mathrm{d}}, \widehat{\mathscr{B}})$, `info.time` is the execution time, and `info.iter` is the number of iterations.
- `M` is the misfit $M(w_{\mathrm{d}}, \widehat{\mathscr{B}})$.
- `wh` is the optimal approximating time series.
- `xini` is a matrix whose columns are the initial condition, under which $\widehat{w}^k$, $k = 1, \ldots, N$ are obtained.

B Implementation

B.1 Main functions

Figure 1 visualizes the data processing done when the `ident` function is called. A number of auxiliary functions, explained and defined next, are used. The main one is the structured low-rank approximation solver `slra`, which computes a locally optimal solution of (SLRA). Technically, `slra` is a mex-file calling a C++ solver. For our purposes the `slra` function is a black box with inputs the structure parameter vector p, the structure specification $\mathscr{S}$, upper bound for the rank r, and (optionally) an initial value of the kernel parameter R_{ini}; and outputs the structure parameter vector $\widehat{p}$ of a locally optimal approximation, and the corresponding kernel parameter $\widehat{R}$.

The computation done in the `ident` function is

1. transformation of the user defined data w_{d}, `m`, ℓ, exact variables, and (optionally) initial model into input parameters for the `slra` function; and
2. transformation of the `slra` function's solution $(\widehat{p}, \widehat{R})$ into state-space representation of the optimal approximate model $\widehat{\mathscr{B}} = \mathscr{B}(\widehat{A}, \widehat{B}, \widehat{C}, \widehat{D})$, optimal approximation $\widehat{w}$ of the data w_{d}, and corresponding initial conditions x_{ini}.

Obtaining a input/state/output representation of the identified model is possible in two different ways: 1) using the $\widehat{R}$ parameter and 2) using the trajectory $\widehat{w}$. The first option is a realization problem (see Sections B.3) and the second one is a deterministic identification problem [24]. Both transformations are classic problems, for which solutions exists.

```
⟨ident⟩≡
  ⟨ident inline help⟩
  ⟨ident function definition⟩
  if ~exist('opt'), opt = []; end
  ⟨default opt.exct⟩
  ⟨define T, q, N⟩, ⟨define p and n⟩
  ⟨structure specification⟩
  ⟨exact variables⟩
  ⟨include opt.wini in w and modify s, T⟩
  ⟨initial approximation⟩
  par = w2p(w); if isfield(s, 'w') && isfield(opt, 'wini'), s.w = w2p(s.w); end
  [ph, info] = slra(par, s, r, opt); info.M = info.fmin;
  wh = p2w(ph, q, N, T, iscell(w)); sysh = r2ss(info.Rh, m, ell);
  ⟨remove opt.wini from wh⟩
  if nargout > 3, xini = inistate(wh, sysh); end
  ⟨define w2p⟩
  ⟨define p2w⟩
  ⟨define ss2r⟩
  ⟨define r2ss⟩
  ⟨define inistate⟩
```

The misfit $M(w_{\mathrm{d}}, \mathscr{B})$ between the data w_{d} and the model $\mathscr{B}$ is the cost function of the approximate identification problem. Its fast computation is a key element of the optimization method used by the `slra` function. A convenient

way to access the misfit computation is to call the `ident` function with the data w_{d}, initial approximation corresponding to the model $\mathscr{B}$, and with specification of zero iterations for the optimization solver. Then the misfit is returned in the field `M` of the output parameter `info`.

⟨`misfit`⟩≡
```
⟨misfit inline help⟩
⟨misfit function definition⟩
[p, m] = size(sysh); n = size(sysh, 'order'); ell = n / p;
opt.sys0 = sysh; opt.maxiter = 0; opt.disp = 'off';
[~, info, wh] = ident(w, m, ell, opt); M = info.M;
if nargout > 2, xini = inistate(wh, sysh); end
⟨define inistate⟩
```

B.2 `w2p` and `p2w`

⟨*define* `w2p`⟩≡
```
function p = w2p(w)
if ~iscell(w), p = w(:); else
  p = []; for k = 1:length(w), p = [p; w2p(w{k})]; end
end
```

⟨*define* `p2w`⟩≡
```
function w = p2w(p, q, N, T, c)
if ~c
  if N == 1, w = reshape(p, T, q, N); else
    for k = 1:N, w(:, :, k) = p2w(p(1:q * T(k)), q, 1, T(k), 0); p(1:q * T(k)) = []; end
  end
else
  for k = 1:N, w{k} = p2w(p(1:q * T(k)), q, 1, T(k), 0); p(1:q * T(k)) = []; end
end
```

B.3 $R \mapsto$ input/state/output representation

The transformation from kernel to input/state/output representation is done using the standard observer canonical form, defined in the following proposition.

Proposition 17 ([11, Section IV.A])**.** *Consider a kernel representation $\mathscr{B}(R)$ of a linear time-invariant system and let $R =: \begin{bmatrix} Q & -P \end{bmatrix}$, with*

$$R_i = \begin{bmatrix} Q_i & -P_i \end{bmatrix}, \qquad \text{where} \quad Q_i \in \mathbb{R}^{\mathtt{p}\times\mathtt{m}} \quad \text{and} \quad P_i \in \mathbb{R}^{\mathtt{p}\times\mathtt{p}}.$$

Assuming that P_ℓ is nonsingular, a minimal state-space representation $\mathscr{B}(A,B,C,D)$ of the system $\mathscr{B}(R)$, i.e.*,*

$$\mathscr{B}(A,B,C,D) = \mathscr{B}(\begin{bmatrix} Q & P \end{bmatrix}),$$

is given by

$$A = \begin{bmatrix} 0 & \cdots & 0 & -P_\ell^{-1}P_0 \\ I_{\mathtt{p}} & & & -P_\ell^{-1}P_1 \\ & \ddots & & \\ & & I_{\mathtt{p}} & -P_\ell^{-1}P_{\ell-1} \end{bmatrix}, \quad B = \begin{bmatrix} P_\ell^{-1}(Q_0 - P_0P_\ell^{-1}Q_\ell) \\ P_\ell^{-1}(Q_1 - P_1P_\ell^{-1}Q_\ell) \\ \vdots \\ P_\ell^{-1}(Q_{\ell-1} - P_{\ell-1}P_\ell^{-1}Q_\ell) \end{bmatrix}, \quad C = \begin{bmatrix} 0 & \cdots & 0 & I_{\mathtt{p}} \end{bmatrix}, \quad D = P_\ell^{-1}Q_\ell.$$

⟨*define* `r2ss`⟩≡
```
function sysh = r2ss(R, m, ell)
[p, tmp] = size(R); ell1 = ell + 1; q = tmp / ell1; n = ell * p;
R = permute(reshape(R, p, ell1, q), [1 3 2]);
Q = R(:, 1:m, :); P = - R(:, m + 1:q, :); inv_Pl = pinv(P(:, :, ell + 1));
a = zeros(n); b = zeros(n, m); c = [];
```

```
if n > 0
  a(p + 1:end, 1:n - p) = eye(n - p);
  c = [zeros(p, n - p) eye(p)];
end
d = inv_Pl * Q(:, :, ell1); ind_j = (n - p + 1):n;
for i = 1:ell
  ind_i = ((i - 1) * p + 1):(i * p); Pi = P(:, :, i);
  a(ind_i, ind_j) = - inv_Pl * Pi;
  b(ind_i, :) = inv_Pl * (Q(:, :, i) - Pi * d);
end
sysh = ss(a, b, c, d, 1);
```

B.4 Estimation of the initial state

Given an input/state/output representation $\mathscr{B}(A,B,C,D)$ and a trajectory $w = \mathrm{col}(u,y)$ of that system, the corresponding initial state x_{ini} is computed from the system of linear equations

$$y - y_{\mathrm{forced}} = \mathscr{O}_L(A,C)x_{\mathrm{ini}}, \tag{x_{ini}}$$

where y_{forced} is the output of the system to input u and zero initial conditions and

$$\mathscr{O}_L(A,C) := \mathrm{col}(C,CA,\ldots,CA^{L-1})$$

is the extended observability matrix.

⟨$(A,C,L) \mapsto \mathscr{O}_L(A,C)$⟩≡
```
O = c; for t = 2:L, O = [O; O(end - p + 1:end, :) * a]; end
```

In order to determine uniquely x_{ini} from (x_{ini}), it is sufficient to choose the parameter L equal to the lag ℓ. For $L > \ell$, existence of solution for (x_{ini}) is ca be used as a test for "exactness" of the trajectory $\big(w(1),\ldots,w(L)\big)$ with respect to the model $\mathscr{B}(A,B,C,D)$, *i.e.*, a test for $\big(w(1),\ldots,w(L)\big) \in \mathscr{B}(A,B,C,D)$.

In the code below y_{forced} is obtained by the function `lsim`.

⟨*define* `inistate`⟩≡
```
function xini = inistate(w, sys, use_all_data)
a = sys.a; c = sys.c; [p, m] = size(sys.d); n = size(a, 1);
⟨define T, q, N⟩
if ~exist('use_all_data') || use_all_data ~= 1, T = max(n, 2) * ones(1, N); end
L = max(T); ⟨(A,C,L) ↦ O_L(A,C)⟩
sys.Ts = -1; xini = zeros(n, N);
for k = 1:N
  if ~iscell(w)
    uk = w(1:T(k), 1:m, k); yk = w(1:L, (m + 1):end, k);
  else
    uk = w{k}(1:T(k), 1:m); yk = w{k}(1:L, (m + 1):end);
  end
  if m > 0, y0k = (yk - lsim(sys, uk, 0:(T(k) - 1)))'; else, y0k = yk'; end
  xini(:, k) = O(1:(T(k) * p), :) \ y0k(:);
end
```

B.5 Input/state/output representation $\mapsto R$

The transformation from input/state/output to kernel representation is done using the following proposition

Proposition 18. *Consider an input/state/output representation $\mathscr{B}(A,B,C,D)$ of a linear time-invariant system with order that is a multiple of the number of outputs. We have,*

$$\mathscr{B}(A,B,C,D) = \mathscr{B}(\begin{bmatrix} Q & -P \end{bmatrix}),$$

where

$$P := \left(\mathscr{O}_{\ell+1}^{\perp}(A,C)\right)^{\top} \qquad \textit{and} \qquad Q := P \begin{bmatrix} D & 0 & 0 & \dots & 0 \\ CB & D & 0 & \ddots & \vdots \\ CAB & CB & D & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ CA^{\ell-2}B & \cdots & CAB & CB & D \end{bmatrix}. \qquad ((A,B,C,D) \mapsto (P,Q))$$

(For a matrix M, $M^{\perp}$ is a full column rank matrix of maximal dimension, such that $M^{\perp}M = 0$.)

Proof. The columns of $\mathscr{O}_{\ell+1}(A,C)$ are free trajectories of the system $\mathscr{B}(A,B,C,D)$, so that they are annihilated by the P parameter

$$P\mathscr{O}_{\ell+1}(A,C) = 0.$$

Moreover, by the assumption that the order is a multiple of the number of outputs, we have that

$$\dim\left(\operatorname{leftker}\left(\mathscr{O}_{\ell+1}^{\perp}(A,C)\right)\right) = \mathtt{p}.$$

Therefore, the rows of P form a basis for the left kernel of $\mathscr{O}_{\ell+1}^{\perp}(A,C)$. This proves that

$$P := \left(\mathscr{O}_{\ell+1}^{\perp}(A,C)\right)^{\top}.$$

Consider the sequence of the first $\ell+1$ samples of the impulse response of $\mathscr{B}(A,B,C,D)$, padded with ℓ zeros:

$$Y := \big(\underbrace{0,\dots,0}_{\ell},D,\underbrace{CB,CAB,\dots,CA^{\ell-1}B}_{\ell}\big),$$

and the corresponding input sequence

$$U := \big(\underbrace{0,\dots,0}_{\ell},I_{\mathtt{m}},\underbrace{0,0,\dots,0}_{\ell}\big).$$

Since (U,Y) is a matrix valued trajectory of the system $\mathscr{B}(\begin{bmatrix} Q & -P \end{bmatrix})$, we have that

$$Q\mathscr{H}_{\ell+1}(U) = P\mathscr{H}_{\ell+1}(Y) \quad \Longrightarrow \quad Q = P\mathscr{H}_{\ell+1}(Y) \begin{bmatrix} & & I_{\mathtt{p}} \\ & \iddots & \\ I_{\mathtt{p}} & & \end{bmatrix}. \qquad (*)$$

The second equation in $((A,B,C,D) \mapsto (P,Q))$ follows from $(*)$. □

⟨*define* `ss2r`⟩≡

```
function R = ss2r(sys)
sys = ss(sys); a = sys.a; b = sys.b; c = sys.c; d = sys.d;
[p, m] = size(d); n = size(a, 1);
ell1 = n / p + 1; L = ell1; ⟨(A,C,L) ↦ O_L(A,C)⟩, P = null(O')';
if (m > 0)
  F = [d; O(1:(end - p), :) * b]; TT = zeros(ell1 * p, ell1 * m);
  for i = 1:ell1
    TT((i - 1) * p + 1:end, (i - 1) * m + 1: i * m) = F(1:(ell1 + 1 - i) * p, :);
  end
  Q = P * TT;
else, Q = []; end
R = permute([reshape(Q, p, m, ell1), -reshape(P, p, p, ell1)], [1 3 2]);
R = reshape(R, p, (m + p) * ell1);
```

B.6 Inline help

⟨ident *inline help*⟩≡

```
% IDENT - LTI system identification by structured low-rank approximation
% [sysh, info, wh, xini] = ident(w, m, ell, opt)
%
% W   - T samples, Q variate time series, stored in a TxQ array
%       or N such trajectories, stored in a TxQxN array
%       or N trajectories, stored in cell array with TixQ elements
%       Missing elements are denoted by NaN's.
% M   - input dimension
% ELL - system lag (order = ELL * (Q - M))
% OPT - options for the optimization algorithm:
%   OPT.EXCT - vector of indices for exact variables (default [])
%   OPT.WINI = 0 specifies zero initial conditions (default [])
%   OPT.SYS0 - initial approximation: an SS system with M inputs,
%              P := Q - M outputs, and order N : = ELL * P
%   OPT.DISP - level of display [off|iter|notify|final] (default off)
%   OPT.MAXITER - maximum number of iterations (default 100)
% SYSH - input/state/output representation of the identified system
% INFO - information from the optimization solver:
%   INFO.M - misfit ||W - WH||_F^2
%   INFO.ITER - number of iterations
% WH    - optimal approximating time series
% XINI - initial condition under which WH is obtained by SYSH
```

⟨misfit *inline help*⟩≡

```
% MISFIT - orthogonal distance from W to an LTI system SYS
% M = minimum over Wh  ||W - Wh||_F^2 s.t. Wh is a trajectory of SYS
% [M, wh, xini] = misfit(w, sys, opt)
%
% W   - time series or a set of time series (see "help ident")
% SYS - state-space system with M inputs, P := size(W, 2) - M outputs
%       and order N := L * P, where L is an integer
% OPT - options
%   OPT.EXCT - vector of indices for exact variables (default [])
%   OPT.WINI = 0 specifies zero initial conditions (default [])
% M    - misfit ||W - WH||_F^2
% WH   - optimal approximating time series
% XINI - initial condition under which WH is obtained by SYSH
```

C Input/state/output representation of the model reduction benchmark

Let $p := \mathrm{col}(p_1, \ldots, p_N)$ and define the state vector by

$$x = \mathrm{col}(p, \frac{\mathrm{d}}{\mathrm{d}t}p, x_{N+1}),$$

where x_{N+L} is a constant, $x_{N+1}(t) = x_{N+1}(0)$, for all t. The order of the state space representation is $\mathtt{n} = 2N+1$. Using (SYS), we obtain an input/state/output $\mathscr{B}(A,B,C,D)$ representation with parameters

$$A = \begin{bmatrix} 0 & I_N & 0 \\ A_{21} & A_{22} & A_{23} \\ 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{(2N+1)\times(2N+1)}, \quad B = \begin{bmatrix} 0 \\ E_{\mathscr{U}} \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} E_{\mathscr{Y}}^\top & 0 & 0 \end{bmatrix}, \quad D = 0, \quad x_{2N+1}(0) = 1,$$

where

$$A_{21} = \begin{bmatrix} -\frac{k_0+k_1}{m_1} & \frac{k_2}{m_1} & & & \\ \frac{k_1}{m_2} & -\frac{k_1+k_2}{m_2} & \frac{k_3}{m_2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{k_{N-2}}{m_{N-2}} & -\frac{k_{N-2}+k_{N-1}}{m_{N-2}} & \frac{k_N}{m_{N-2}} \\ & & & \frac{k_{N-1}}{m_N} & -\frac{k_{N-1}+k_N}{m_N} \end{bmatrix}$$

and

$$A_{22} = \begin{bmatrix} -\frac{d_0+d_1+f}{m_1} & \frac{d_2}{m_1} & & & \\ \frac{d_1}{m_2} & -\frac{d_1+d_2+f}{m_2} & \frac{d_3}{m_2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{d_{N-2}}{m_{N-2}} & -\frac{d_{N-2}+d_{N-1}+f}{m_{N-2}} & \frac{d_N}{m_{N-2}} \\ & & & \frac{d_{N-1}}{m_N} & -\frac{d_{N-1}+d_N+f}{m_N} \end{bmatrix}, \qquad A_{23} = \begin{bmatrix} \frac{k_0-k_1}{m_1} \\ \frac{k_1-k_2}{m_2} \\ \vdots \\ \frac{k_{N-2}-k_{N-1}}{m_{N-1}} \\ \frac{k_{N-1+N}k_N}{m_N} \end{bmatrix} \delta.$$

Define the unit vector e_i, as the ith column of the $\mathtt{n} \times \mathtt{n}$ identity matrix. Let $\mathtt{p}$ be the number of elements of the set $\mathscr{Y}$ and $\mathtt{m}$ be the number of elements of the set $\mathscr{U}$. The matrix $E_{\mathscr{U}}$ is the $\mathtt{n} \times \mathtt{m}$ matrix with ith column $1/m_i \cdot e_i$ and, similarly, $E_{\mathscr{Y}}$ is the $\mathtt{n} \times \mathtt{p}$ matrix with ith column e_i.

The order of the system $\mathscr{B}(A,B,C,D)$ is $2N$, *i.e.*, the state space representation is nonminimal. (It has an uncontrollable mode $x_{N+1} = \text{const.}$)

⟨*form A*⟩≡
```
if N > 1
    a21 =   diag(K(2:end - 1) ./ M(2:end), -1) ...
          - diag((K(1:end - 1) + K(2:end)) ./ M) ...
          + diag(K(2:end - 1) ./ M(1:end - 1), 1);
    a22 =   diag(D(2:end - 1) ./ M(2:end), -1) ...
          - diag((D(1:end - 1) + D(2:end) + f) ./ M) ...
          + diag(D(2:end - 1) ./ M(1:end - 1), 1);
else
    a21 = - sum(K) / M; a22 = - (sum(D) + f) / M;
end
a = [zeros(N) eye(N); a21 a22];
```

⟨*form A*⟩+≡
```
a_ext = zeros(2 * N, 1);
if N > 1
    a_ext(N + 1:end, 1) = [(K(1:end - 2) - K(2:end - 1)) ./ M(1:end - 1);
                           (K(end - 1) + N * K(end)) / M(end)] * delta;
else
    a_ext(2) = sum(K) / M;
end
a = [a, a_ext; zeros(1, 2 * N + 1)];
```

⟨*form B, C, D*⟩≡
```
n = 2 * N + 1; m = length(U); p = length(Y);
b = zeros(n, m); for i = 1:m, b(N + U(i), i) = 1 / M(U(i)); end
c = zeros(p, n); for i = 1:p, c(i, Y(i)) = 1; end
d = zeros(p, m);
```

⟨*define* sys⟩≡
```
function s = sys(N, varargin)
⟨parse model parameters⟩
⟨form A⟩, ⟨form B, C, D⟩, s = ss(a, b, c, d);
```

⟨*parse model parameters*⟩≡

```
ip = inputParser; ip.KeepUnmatched = true;
ip.addParamValue('delta', 1)
ip.addParamValue('f', 0)
ip.addParamValue('M', ones(N, 1))
ip.addParamValue('K', ones(N + 1, 1))
ip.addParamValue('D', ones(N + 1, 1))
ip.addParamValue('U', 1)
ip.addParamValue('Y', round(N / 2))
ip.parse(varargin{:}); opt = ip.Results;
names = fieldnames(opt);
for i = 1:length(names), eval(sprintf('%s = opt.%s;', names{i}, names{i})); end
```