Exercises for the SOCN course "Behavioral approach to systems theory"

Ivan Markovsky

This document illustrates and extends the material presented in the lectures of the course by providing computational examples and 44 exercises for independent work. As Richard Hamming said "the purpose of computing is insight no numbers" [5]. The value of the exercises is not the obtained solutions but the thought process of coming up with them: the ideas and attempted ideas that one has when doing the exercises. Thus, although sample solutions are provided, it is important to attempt the exercises independently before consulting with others or reading the solutions.

The exercises are organized thematically and gradually build knowledge and tools. Therefore it is recommended to do them sequentially. Section 1 gives examples of alternative direct data-driven solutions to model-based ones. The examples—representation of the restricted behavior $\mathscr{B}|_T$, computing the lag of the system, and finding a kernel representation of a given system $\mathscr{B}$—are meaningful problems on their own. Section 2 is about obtained a model from data. The data is assumed exact and the model is required to be exact for the data, however, as simple as possible. This is the concept of *most powerful unfalsified model* introduced and used in the lectures. Section 3 develops a fully featured implementation of the data-driven interpolation and approximation algorithm presented in the lectures. Sections 4–8 show applications of to algorithm to simulation, missing data estimation, errors-in-variables smoothing, classical Kalman smoothing, state transition, and least-squares optimal tracking control. Appendix A introduces the Hankel, mosaic-Hankel, and polynomial multiplication matrices, which are used in the exercises.

1 A first glimpse of data-driven methods

The problems considered involve a true data-generating system (sometimes called "plant" in the context of control). This system may be explicitly given by a parametric representation or it may be implicitly specified by data. The data consists of one or more trajectories of the system and prior knowledge, such as linear time-invariant dynamics, the number of inputs, and order. Problems and corresponding solution methods starting from a representation of the system are called *model-based*. Problems and corresponding solution methods starting from data are called *data-driven*. A data-driven problem can be reduced to a corresponding model-based problem by model identification. Methods that solve the data-driven problem by first identifying a model and then using a model-based method are called *indirect*. Methods that solve the data-driven problem directly without parametric model identification are called *direct*. Before diving into data-driven problems, this section shows that some model-based problems have simple data-driven solution, *i.e.*, even though a parametric model is given, it may be easier to solve the model-based problem by simulating data and using a direct data-driven method. "Easier" means requiring less human effort (less thinking and coding). The approach of using simulation and direct data-driven method is computationally inefficient.

Representation of the restricted behavior $\mathscr{B}|_T$

Exercise 1 (From (A,B,C,D,Π) to a basis of $\mathscr{B}|_T$). Given an input/state/output representation $\mathscr{B} = \mathscr{B}_{\rm ss}(A,B,C,D,\Pi)$ of a linear time-invariant system $\mathscr{B}$ and a natural number T, find a basis for the restricted behavior $\mathscr{B}|_T$ and implement the solution in a function `BT = ss2BT(B)`. Do the exercise in two different ways:

1. *model-based* — using the parameters A,B,C,D,Π to obtain an explicit formula for the basis, and

2. *data-driven* — using a simulated trajectory $w_{\rm d} \in \mathscr{B}|_{T_{\rm d}}$.

Computing the lag of a given system

Exercise 2 (Finding $\boldsymbol{\ell}(\mathscr{B})$). Given an input/state/output representation $\mathscr{B} = \mathscr{B}_{\mathrm{ss}}(A,B,C,D,\Pi)$, find the lag $\boldsymbol{\ell}(\mathscr{B})$ and implement the solution in a function `ell = lag(B)`. Again, do the exercise in two different ways:

1. *model-based* — using the parameters A,B,C,D,Π, and
2. *data-driven* — using a simulated trajectory $w_{\mathrm{d}} \in \mathscr{B}|_{T_{\mathrm{d}}}$.

Finding a kernel representation

Exercise 3 (From an input/state/output to a kernel representation `ss2r`). Given a linear time-invariant system $\mathscr{B}$, specified by an input/state/output representation $\mathscr{B}_{\mathrm{ss}}(A,B,C,D,\Pi)$, find a kernel representation $\mathscr{B} = \ker R(\sigma)$ and implement the solution in a function `R = ss2r(B)`. Again, do the exercise in two different ways:

1. *model-based* — using the parameters A,B,C,D,Π to obtain an explicit formula for R, and
2. *data-driven* — using a simulated trajectory $w_{\mathrm{d}} \in \mathscr{B}|_{T_{\mathrm{d}}}$.

Summary and discussion

In the data driven solutions of the problems considered, the main tool is the Hankel matrix. In the representation of the finite-horizon behavior $\mathscr{B}|_T$, we used the image of $\mathscr{H}_T(w_{\mathrm{d}})$. In finding a kernel representation, we used the left kernel of $\mathscr{H}_{\ell+1}(w_{\mathrm{d}})$, in the lag computation problem we used the rank of $\mathscr{H}_{\ell+1}(w_{\mathrm{d}})$. Basis vectors of image $\mathscr{H}_T(w_{\mathrm{d}})$ are called *generators* of the finite-horizon behavior $\mathscr{B}|_T$ because they generate it. Basis vectors of the left kernel of $\mathscr{H}_{\ell+1}(w_{\mathrm{d}})$ are called *annihilators* of $\mathscr{B}|_{\ell+1}$ because applied to it the result is zero. The "application" of a basis vector r to $\mathscr{B}|_{\ell+1}$ is left-multiplication $r\mathscr{H}_{\ell+1}(w)$, where $w \in \mathscr{B}$. The multiplication $r\mathscr{H}_{\ell+1}(w)$ can be viewed alternatively as the action of the polynomial operator $r(\sigma)$ defined by r on w. Indeed, a basis R for the left kernel of $\mathscr{H}_{\ell+1}(w_{\mathrm{d}})$ corresponds to the parameter R of a kernel representation $\ker R(\sigma)$ of the system $\mathscr{B}$.

In the model-based solutions we used an input/state/output representation of the system. These problems can be posed however for any representation. The following extra exercise aims at a representation of the restricted behavior $\mathscr{B}|_T$ from a given kernel representation.

Exercise 4 (From a kernel representation to a basis of $\mathscr{B}|_T$). Given a kernel representation $\mathscr{B} = \ker R(\sigma)$ of a linear time-invariant system $\mathscr{B}$ and a natural number T, find a basis for the restricted behavior $\mathscr{B}|_T$ and implement the solution in a function `B = r2BT(ss)`.

2 Most powerful unfalsified model

The most powerful unfalsified model $\mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}})$ of the data w_{d} is the least complex linear time-invariant system that fits the the data exactly, *i.e.*,

$$\mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}}) := \arg\min_{\widehat{\mathscr{B}} \in \mathscr{L}^q} \boldsymbol{c}(\widehat{\mathscr{B}}) \quad \text{subject to} \quad w_{\mathrm{d}} \in \widehat{\mathscr{B}}|_{T_{\mathrm{d}}}. \tag{1}$$

It was originally defined for infinite data w_{d} in which case the solution of (1) is unique [13].

In case of finite data $w_{\mathrm{d}} \in (\mathbb{R}^q)^T_{\mathrm{d}}$, however, the solution of (1) is a (trivial) autonomous system (see Exercise 5). One way to avoid the trivial solution is to impose an upper bound on the lag, *i.e.*, $\boldsymbol{\ell}(\widehat{\mathscr{B}}) \leq \ell_{\max}$. The upper bound

$$\ell_{\max} := \left\lfloor \frac{T_{\mathrm{d}}+1}{q+1} \right\rfloor, \tag{$\ell_{\max}$}$$

```
Lmax = @(wd) floor((size(wd, 1) + 1) / (size(wd, 2) + 1)); % <define-Lmax>
```

does not require hyper-parameters (it is determined only by the data size) and is justified by the fact that for $\ell > \ell_{\max}$ the Hankel matrix $\mathscr{H}_{\ell+1}(w_{\mathrm{d}})$ has more rows than columns and the generalized persistency of excitation condition can not be satisfied, so that the data-generating system is not identifiable from the data.

For finite data there is a (trivial) exact autonomous model

Exercise 5. Show that for finite length data, $\mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}})$ is autonomous, *i.e.*, $\boldsymbol{m}\big(\mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}})\big) = 0$.

Most powerful unfalsified model's complexity

Exercise 6. Given a trajectory $w_{\mathrm{d}} \in (\mathbb{R}^q)^{T_{\mathrm{d}}}$, find the complexity (m, ℓ, n) of $\mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}})$ and implement the solution in a function `c = c_mpum(wd)`.

Kernel representation of $\mathscr{B}_{\mathbf{MPUM}}(w_{\mathbf{d}})$

Exercise 7 (Finding a kernel representation of $\mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}})$). Given a trajectory $w_{\mathrm{d}} \in (\mathbb{R}^q)^{T_{\mathrm{d}}}$, find a kernel representation $\ker R(\sigma) = \mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}})$ of the most powerful unfalsified model and implement the solution in a function `R = w2R(wd)`.

Summary and discussion

The problem of computing $\mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}})$ is an exact/deterministic identification problem. The rationale for computing an exact model can be questioned from a practical engineering perspective: Why studying such a problem when the data is never exact in practice? There good reasons for doing this:

1. Before one understands the more complicated problem of identification of approximate model, one should understand the simpler problem of exact identification. (Indeed, exact identification is a special case of approximate identification when the data happens to be exact.)

2. Exact models and therefore exact identification appears as a subproblem of approximate identification problems as well as in data-driven signal processing and control. For example, one may ask the questions:

   "What is the smallest perturbation δw of the given data w_{d} that renders the perturbed data $\widehat{w} = w_{\mathrm{d}}$ exact?" and

   "What is the smallest auxiliary signal e that makes the extended data $w_{\mathrm{ext}} = \left[\begin{smallmatrix} w_{\mathrm{d}} \\ e \end{smallmatrix}\right]$ exact?"

   This is a sound and insightful approach to system identification and data-driven control. In a stochastic setting, the first question leads to maximum likelihood identification in the errors-in-variables setting, and the second question to maximum likelihood identification in the Auto-Regressive Moving-Average eXogenous (ARMAX) setting.

3. Minor modification of exact identification methods result in approximate identification methods. This is how the sub-field of subspace identification came about.

In subspace identification the goal is to find an input/state/output representation of $\mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}})$ (see, Exercise 7 where the goal is to find a kernel representation of $\mathscr{B}_{\mathrm{MPUM}}(w_{\mathrm{d}})$.)

The most powerful unfalsified model $\mathscr{B}_{\mathrm{MPUM}}(w)$ is defined for any discrete-time signal $w_{\mathrm{d}} \in (\mathbb{R}^q)^{T_{\mathrm{d}}}$. Assuming that w_{d} is a trajectory of a system $\mathscr{B} \in \mathscr{L}^q$, however, the question occurs "What is the relation of $\mathscr{B}_{\mathrm{MPUM}}(w)$ and $\mathscr{B}$?" In general, $\mathscr{B}_{\mathrm{MPUM}}(w) \subseteq \mathscr{B}$. Under the *identifiability condition*

$$\operatorname{rank} \mathscr{H}_{\boldsymbol{\ell}(\mathscr{B})+1}(w_{\mathrm{d}}) = \boldsymbol{m}(\mathscr{B})T + \boldsymbol{n}(\mathscr{B}). \tag{2}$$

powerful unfalsified model coincides with the data-generating system, *i.e.*, $\mathscr{B}_{\mathrm{MPUM}}(w) = \mathscr{B}$. In this case the data-generating system can be recovered back from the data by computing the $\mathscr{B}_{\mathrm{MPUM}}(w)$. The fundamental lemma gives alternative identifiability conditions in terms of the input and the system only (not on the whole w_{d}). These conditions are useful for input design, *i.e.*, for collecting data w_{d} that satisfies the identifiability condition (2).

3 Implementation of the algorithm for data-driven interpolation/approximation

The exercises in this section lead to a fully featured implementation of the method for interpolation/approximation of trajectories of [10]. The interpolation/approximation of trajectories problem is defined as follows:

$$\text{minimize} \quad \text{over } \widehat{w} \quad \|w-\widehat{w}\|_S \quad \text{subject to} \quad \widehat{w} \in \mathscr{B}|_T, \tag{3}$$

where the data-generating system $\mathscr{B}$ is implicitly specified by a *data trajectory* $w_{\mathrm{d}} \in \mathscr{B}|_{T_{\mathrm{d}}}$ and the cost function

$$\|w-\widehat{w}\|_S := \|S \odot (w-\widehat{w})\| \tag{4}$$

is the 2-norm of the element-wise product $\odot$ between the weights $S_i(t) \geq 0$ and the approximation errors $w_i(t)-\widehat{w}_i(t)$. Zero weights $S_i(t)=0$ specify missing data, while infinite weights $S_i(t)=\infty$ specify exact interpolation points.

The method for solving (3) uses the data-driven representation

$$\mathscr{B}|_T = \text{image } \mathscr{H}_T(w_{\mathrm{d}}), \tag{5}$$

of the system $\mathscr{B}$, where $\mathscr{H}_T(w_{\mathrm{d}})$ is the Hankel matrix with T-block-rows, constructed from the data w_{d}. A necessary and sufficient condition for (5) is that w_{d} is a trajectory of a linear time-invariant system $\mathscr{B}$ and

$$\operatorname{rank} \mathscr{H}_T(w_{\mathrm{d}}) = \boldsymbol{m}(\mathscr{B})T + \boldsymbol{n}(\mathscr{B}). \tag{6}$$

The result of doing the exercises is a Matlab function `ddint` with the following interface:

```
function [wh, N, g] = ddint(wd, w, S, m, n, l, L, ell)
```

The input argument `wd` corresponds to the data trajectory w_{d} and `w` corresponds to the to-be-approximated/interpolated trajectory w. Specifying `wd` and `w` is compulsory. The other input arguments are optional. The output argument `wh` corresponds to a solution $\widehat{w}$ of (3). In addition to solving (3), `ddint` implements the following functionality:

- model-based data-driven interpolation/approximation,
- data w_{d} consisting of multiple trajectories $w_{\mathrm{d}}^1, \ldots, w_{\mathrm{d}}^N$,
- specification of the set of all solutions in case of a nonunique solution $\widehat{w}$,
- preprocessing of the data w_{d} via low-rank approximation, and
- ℓ_1-norm regularization.

The extra features are accessible via the optional arguments as described in the exercises.

Exercise 8 (Basic algorithm `wh = ddint(wd, w)`). Implement and test on simulation examples the basic algorithm for data-driven interpolation and approximation. The first dimension `Td` of `wd` is the number of samples and the second dimension `q` is the number of variables. The number of samples `Td` must be bigger than `q`. The argument `w` specifies the to-be-interpolated/approximated trajectory w. Again, the first dimension represents time and the second dimension corresponds to variables. The missing values of w are specified by `NaN`'s. If an exact interpolant does not exist, an approximate one is computed using the unweighted least-squares cost function (4), *i.e.*, $S_{ij}=1$, for all i, j.

Exercise 9 (Nonuniqueness of the solution). The interpolation condition is that the interpolant $\widehat{w}$ of w is a trajectory of $\mathscr{B}$. If the interpolant is not unique the set of all interpolants is given by $\widehat{\mathscr{W}} = \widehat{w} + \text{image } N$, where $\widehat{w}$ (returned in the output argument `wh`) is a particular solution and the matrix N specifies the nonuniqueness via its image. Modify `ddint` to compute N and return in the output argument `N`.

Exercise 10 (Model-based interpolation/approximating). In the basic algorithm, the argument `wd` implicitly specifies the data-generating system $\mathscr{B}$ by a trajectory. Implement and test on simulation examples an option of `ddint` to accept as `wd` the model $\mathscr{B}$ specified by an `ss` or `tf` object instead of a trajectory.

Exercise 11 (Data consisting of multiple trajectories). Implement and test on simulation examples an option of `ddint` to accept as `wd` a cell array containing multiple trajectories $w_{\mathrm{d}}^1, \ldots, w_{\mathrm{d}}^N$ of the system $\mathscr{B}$.

Exercise 12 (Element-wise weighted cost function). Implement and test on simulation examples an option of `ddint` to accept an element-wise positive matrix $S \in \mathbb{R}^{T\times q}$ that specifies a weighted cost function (4). The weight matrix is passed to `ddint` via the optional argument `S`. The default value of `S` is unit weights (`ones(T, q)`), which corresponds to the ordinary least-squares approximation.

Exercise 13 (Zeros and infinite weights). Modify `ddint` so that

1. interpolation conditions can specified by infinite weights `S(t, i) = inf`, and
2. missing values can be specified by zero weights `S(t, i) = 0`.

Exercise 14 (Noise filtering via low-rank approximation). Include in `ddint` the optional input parameters `m` and `n` that specify the complexity of the data-generating system $\mathscr{B}$. Use the parameters `m` and `n` are, when passed to `ddint`, for checking the condition (6) for the data-driven representation. If the condition is not satisfied, issue a warning message. In case of inexact data `wd` (*i.e.*, $\operatorname{rank}\mathscr{H}_T(w_\mathrm{d}) > mT+n$) use the parameters `m` and `n` for preprocessing of the data matrix $\mathscr{H}_T(w_\mathrm{d})$ by unstructured rank-$mT+n$ approximation.

Exercise 15 (ℓ_1-norm regularization). By default, $\widehat{w}$ is computed by solving the weighted least-squares problem $\|w - Dg\|_S$ with the pseudo-inverse. Modify `ddint` to allow for specification of the optional parameter `l` that adds the regularization term $\mathtt{l}\cdot\|g\|_1$ in the cost function.

Exercise 16 (Recursive computation). Modify the code of `ddint` to allow for the optional parameters `L` (block size) and `ell` (lag of the system) that trigger a recursive computation of $\widehat{w}$ in blocks of `L` samples, where $1 \leq \mathtt{L} \leq T$. The recursive algorithm matches the final conditions of a block with the initial conditions of the following block.

Summary and discussion

In this section, we've build the foundation for the experiments in the following sections. Next, we will apply `ddint` for solving well-known problems, such as simulation, smoothing, and tracking control, as well as less well-known problems, such as input estimation and missing data estimation. The mini-projects compare `ddint` with alternative model-based methods in case of noisy data, disturbances, and nonlinear system dynamics.

A quick and dirty way of dealing with equality constraints is to replace the infinite weights in S by a large value (*e.g.*, 10^8). As an optional exercise, compare the solutions obtain by infinite and large values (but finite) values for weights corresponding to exact interpolation points in test examples.

In the stochastic setting of *errors-in-variables* estimation $w = \overline{w} + \widetilde{w}$, where $\overline{w} \in \mathscr{B}|_T$ is the true value of w and the measurement noise $\widetilde{w}$ is zero mean, uncorrelated, white, Gaussian, with element-wise standard deviations `1 / S(t, i)`, finite positive weights `S` specify the noise standard deviation of to-be-approximated "noisy" elements of w. The weighted least-squares cost function corresponds then to the maximum-likelihood estimation criterion, *i.e.*, the solution $\widehat{w}$ of (3) is the maximum-likelihood estimator of w.

4 Simulation

A basic operation a model is used for is simulation. It is defined for a system $\mathscr{B}$ with an input/output partitioning of the variables $w = \left[\begin{smallmatrix} u \\ y \end{smallmatrix}\right]$ as follows: given the system $\mathscr{B}$, the input u, and the "initial condition", find the output y. It turns out that for a linear time-invariant system with lag ℓ, the "initial condition" can be specified by a "prefix" trajectory $w_\mathrm{ini} \in \mathscr{B}|_{T_\mathrm{ini}}$ of w with length $T_\mathrm{ini} \geq \ell$. The result is formally stated in Lemma 1 and visualized in Figure 1.

Lemma 1 (Initial condition specification [11])**.** *Let $\mathscr{B} \in \mathscr{L}^q$ admit an input/output partition $w = (u,y)$. Then, for any given $w_{ini} \in \mathscr{B}_{T_{ini}}$ with $T_{ini} \geq \boldsymbol{\ell}(\mathscr{B})$ and $u \in (\mathbb{R}^m)^L$, there is a unique $y \in (\mathbb{R}^p)^L$, such that $w_{ini} \wedge (u,y) \in \mathscr{B}|_{T_{ini}+L}$.*

From the behavioral point of view, simulation is a way of parametrizing the elements of $\mathscr{B}$. Equivalently, simulation is the problem of selecting an element $w \in \mathscr{B}$ of the behavior. The prefix trajectory w_ini plays the role of the initial state in the state-space setting. Estimation of w_ini for a given "future" trajectory w corresponds to state estimation.

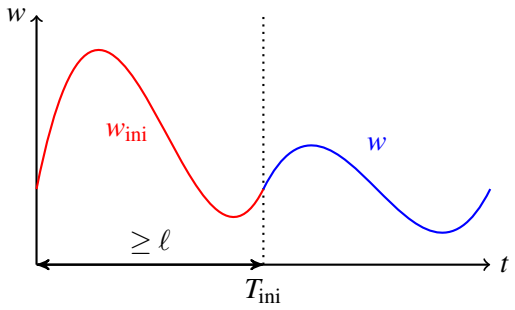

Figure 1: Initial condition for a trajectory $w \in \mathscr{B}$ are specified in the behavioral setting by a prefix trajectory w_{ini} of length $T_{\text{ini}} \geq \boldsymbol{\ell}(\mathscr{B})$. The condition that w is generated from the initial condition specified by w_{ini} is then $w_{\text{ini}} \wedge w \in \mathscr{B}$.

Example of using `ddint` for simulation

In order to illustrate the idea on a numerical example, we choose a data-generating system $\mathscr{B}$

```
n = 5; p = 1; m = 1;
B = drss(n, p, m);
```

input `us`, and initial conditions `xs_ini` for the simulation problem

```
Ts = 10; us = rand(Ts, m); xs_ini = zeros(n, 1);
```

In the example, the simulation horizon is $T_{\text{s}} = 10$ samples, the input is a uniform random process, and the system is initially at rest (zero initial conditions). Once the system, the input, and the initial conditions are chosen, we obtain the corresponding response y_{s} by

```
ys = lsim(B, us, [], xs_ini); ws = [us ys];
```

In order to do data-driven simulation, we need a "data" trajectory `wd` of the system $\mathscr{B}$. In the example, `wd` is a random trajectories of $\mathscr{B}$ with length $T_{\text{d}} = 100$ samples

```
Td = 100; wd = B2wd(Td)
```

The length $T_{\text{d}} = 100$ is chosen arbitrarily, however, it must satisfy the lower bound

$$T_{\text{d}} \geq T_{\text{min}} := (m+1)T + n - 1.$$

The trajectory w represents the to-be-simulation trajectory w_{s} of the system. In order to account for the initial conditions, according to Lemma 1, w is split into two parts, called "past" and "future". The past is of length n and specifies the initial conditions. The future is of length T_{s} and contains the simulated response. In the example, the initial conditions are zero, so that the past is set to zero. The future is the given input u_{s} and `NaN`'s for the to-be-computed output y_{s}.

```
w = [zeros(n, m + p); [us NaN(Ts, p)]];
```

With `wd` and `w` specified, we are ready to solve the data-driven simulation problem by calling the function `ddint`

```
wh = ddint(wd, w);
```

The simulation output $\widehat{y}_{\text{s}}$ is obtained from the interpolated samples in `wh`

```
ysh = wh(n+1:end, m+1:end);
```

Finally, we verify that the model-based and the data-driven simulation methods yield the same result

```
norm(ys - ysh) % -> 1e-15
```

Exercises

Exercise 17 (Impulse response computation). An important special case of simulation is impulse response computation. Apply the data-driven simulation method implemented in `ddint` to compute the first $T_\mathrm{s} = 10$ samples of the impulse response of the system directly from the data w_d. Verify that the result obtained by `ddint` is exact by comparing it with the true impulse response computed by model-based simulation.

The System Identification Toolbox of Matlab provides a function `impulseest` for estimation of impulse response from data. Apply it to the problem at hand using the data w_d. Compare the estimation accuracy and computational efficiency of `ddint` and `impulseest`.

Exercise 18 (Step response computation). Another important special case of simulation is step response computation. Apply `ddint` to compute the first $T_\mathrm{s} = 10$ samples of the step response of the system directly from the data w_d. Verify that the result obtained by `ddint` is exact by comparing it with the true step response computed by model-based simulation.

Exercise 19 (Minimum number of data samples T_d). Verify that the minimum number of samples T_d for which `ddint` yields correct (*i.e.*, exact up to the numerical precision) result is $T_\mathrm{min} := (m+1)T + n - 1$.

Exercise 20 (Multivariable systems). Compare the model-based (`lsim`) and data-driven (`ddint`) simulation methods on multivariable systems.

Exercise 21 (Multiple data trajectories $w_\mathrm{d}^1, \ldots, w_\mathrm{d}^N$). Try out the feature of `ddint` to use as data `wd` multiple trajectories. This allows one to reduce the length T_d of the experiments. How small can T_d be made by using multiple experiments? How many experiments are needed then?

Exercise 22 (Computational time as a function of the simulation horizon T). Increase the number of samples T_s (using the minimum number of data samples T_d, see Exercise 19) and observe how the computational time grows. Compare the result with the one of model-based methods.

Exercise 23 (Nonzero initial conditions). Explain how to deal with nonzero initial conditions. Compare the model-based (`lsim`) and the data-driven (`ddint`) simulation methods in case of nonzero initial conditions.

Exercise 24 (Autonomous systems). Compare the model-based and data-driven simulation methods in case of autonomous systems.

Exercise 25 (Recursive data-driven simulation). Experiment with the option of `ddint` to do recursive computation in blocks of L samples, where $\ell < L \leq T$. Show that the recursive computation allows to reduce the data limit $T_\mathrm{min} := (m+1)T + n - 1$. What is the minimum number of data samples T_d needed in case of recursive computation of $\widehat{w}$?

Summary and discussion

This section shows that `ddint` is applicable to the simulation problem and results in a general and practical data-driven simulation method. Exercises 17, 18, and 24 demonstrate the data-driven simulation method based on `ddint` in the special cases of impulse response computation, step response computation, and simulation of an autonomous system. For impulse response computation of a finite impulse response system, the System Identification Toolbox of Matlab provides an alternative direct data-driven method `impulseest`. For all other cases, the available data-driven simulation methods are indirect, *i.e.*, they involve an explicit model identification. Exercise 20 demonstrates that `ddint` works correctly in case of general multivariable systems. Exercise 23 shows how initial conditions can be obtained also by using `ddint` for solving a data-driven version of the classical observer problem. Exercise 21 demonstrates the possibility of using data from multiple experiments. Exercise 19 verifies formula $(m+1)T + n - 1$ for the minimal amount of data samples T_d and Exercise 22 shows empirically the computation time of `ddint` as a function of the simulation horizon T. In comparison with model-based methods `ddint` is less efficient, however, on a modern computer the difference is noticeable only at relatively large simulation horizons.

The idea of data-driven interpolation and approximation as well as the method of [10] are inspired by [11], where data-driven methods for simulation and control are presented. Apart from generalization of the problem, an important new development compared to [11] is the use of low-rank approximation [7] and regularization for dealing with inexact data [2]. These developments allowed practical application of the methods.

5 Missing data estimation

Simulation is a special interpolation problem. In this section, we consider two other interpolation problems: input estimation and estimation of periodically missing values. The latter example demonstrates the full flexibility of `ddint` to treat missing values among inputs as well as outputs at arbitrary moments of time.

Input estimation

Consider a system $\mathscr{B}$ with two inputs—u_1 and u_2—and one output:

```
n = 5; p = 2; m = 1;
<<random-data-generating-system-B>>
```

implicitly specified by a trajectory w_d

```
Td = 100;
<<random-data-trajectory-wd>>
```

The goal is to estimate u_1, given u_2 and y.

```
T = 20; u0 = rand(T, m); x_ini = zeros(n, 1);
y0 = lsim(B, u0, [], x_ini); w0 = [u0 y0];
```

In order to solve the data-driven input estimation with `ddint`, we pose it as a missing data estimation problem

```
w  = [NaN(T, 1) u0(:, 2:end) y0];
wh = ddint(wd, w); u1h = wh(:, 1);
```

Finally, we verify that the estimate `u1h` matches the true input u_1

```
norm(u0(:, 1) - u1h) % -> 1e-14
```

Periodic missing values

The given values `w` are sampled from `w0` with a period of $n+1$ samples:

```
w = w0; w(3:n+1:end, :) = NaN;
```

The interpolated trajectory by `ddint`

```
wh = ddint(wd, w);
```

matches exactly the true trajectory `ws`

```
norm(w0 - wh) % -> 1e-12
```

A model-based method for missing data estimation is implemented in the function `misdata` of the System Identification toolbox of Matlab. Applying it to the data in the example

```
wh_mb = misdata(iddata(w(:, 2:3), w(:, 1)), idss(B));
wh_mb = [wh_mb.InputData wh_mb.OutputData];
norm(w0 - wh_mb) % -> 1e-15
```

it also recovers exactly the missing data.

Exact recovery of the missing data, however, is not guaranteed. General necessary and sufficient conditions are not available. Necessary conditions and special cases are explored in the exercises.

Exercises

Exercise 26 (Necessary condition for uniqueness). Argue that a necessary condition for unique solution of the interpolation problem is that at least $mT+n$ elements in w are given. Is it also sufficient?

Exercise 27 (Condition for uniqueness of the input estimation problem). When does the input estimation problem have a unique solution? Give separately the cases of known initial conditions and unknown initial conditions. Start with the single-input single-output case.

The following problems explore the impact of the missing elements location on the uniqueness of the completion.

Exercise 28 (Smallest number of given elements in w guaranteeing unique interpolation). Give examples of interpolation problems with as few given elements in w as possible that result in unique completion. Consider separately the cases of scalar autonomous system, single-input single-output system, and general multivariable system.

Exercise 29 (Smallest number of missing elements in w leading to nonuniqueness). Give examples of interpolation problems with as few missing values as possible that result in nonunique completion. Consider separately the cases of scalar autonomous system, single-input single-output system, and general multivariable system.

Summary and discussion

This section demonstrates the flexibility of `ddint` to estimate missing elements in arbitrary configuration of input as well as output variables. This is another viewpoint of the trajectory interpolation aspect of the basic problem (3) that `ddint` solves. The key question in missing data estimation is "under what conditions the completion is unique?" General necessary and sufficient conditions for uniqueness of the interpolant is an open problem. The exercises explored some special cases thus developing intuition about the general question.

Dealing with missing values in the data trajectory w_{d} (as well as in the to-be-interpolated trajectory w) makes the problem much harder. A subspace-type method is proposed in [8]. Other methods are based on local optimization [12] and convex relaxations [3].

6 Errors-in-variables smoothing

In the examples reviewed so far the given samples are exact and the goal is interpolation, or, equivalently, missing values recovery. Existence of an exact completion of the missing values is guaranteed by the exactness of the data. Moreover, uniqueness of the completion guarantees correct recovery of the missing values, so it is the critical property.

The new aspect in this section is that w is inexact. This can be due to

1. additive measurement noise on w (*errors-in-variables setup*),

2. unobserved disturbances acting on the system and measurement noise on the output y (*ARMAX setup*),

3. the data-generating system is not linear time-invariant,

or a combination of the three. The data trajectory w_{d} however is still assumed exact. (Inexact w_{d} is considered in the mini-projects.)

In case of inexact data w, exact solution of (3) generically does not exist. The problem involves then an approximation. In case of additive measurement noise (errors-in-variables setup) and disturbances (ARMAX setup), which are modeled as stochastic processes, the goal is to obtain the maximum-likelihood estimator of the true value of w.

The errors-in-variables smoothing problem is defined as

$$\text{minimize} \quad \text{over } \widehat{w} \quad \|w-\widehat{w}\|_S \quad \text{subject to} \quad \widehat{w} \in \mathscr{B}|_T. \tag{7}$$

The least-squares approximation $\widehat{w}$ is the maximum-likelihood estimator of the true value `w0` of w in the errors-in-variables setup assuming zero mean white Gaussian noise.

Exercises

Exercise 30 (Errors-in-variables smoothing). Solve the errors-in-variables smoothing problem (7) and implement the solution in a function `eiv_smoothing`, using the two approaches:

1. *model-based* — assuming that an input/state/output representation of the system is given, use the parameters A, B, C, D, Π to obtain an explicit formula for the smoothed trajectory $\widehat{w}$, and
2. *data-driven* — find $\widehat{w}$ using a trajectory $w_{\mathrm{d}} \in \mathscr{B}|_{T_{\mathrm{d}}}$.

Exercise 31 (Monte-Carlo simulation). In order to reduce the randomness of the result, average the relative approximation errors over $N = 100$ noise realizations.

Exercise 32 (Approximation error as a function of the noise level). Plot the average relative approximation error as a function of the noise level for noise in the interval $[0\%,\ 50\%]$. Is the result what you expected? Can you explain it?

Exercise 33 (Smoothing with exactly known initial conditions). Do the errors-in-variables smoothing experiment with known zero initial conditions. Compare the accuracy of the solutions (with respect to the true value) with and without using the prior knowledge about the initial conditions.

Exercise 34 (Errors-in-variables smoothing with exact interpolation points). Redo the errors-in-variables smoothing experiment with unknown initial conditions for an autonomous system with $1, \ldots, n-1$ interpolation points that are randomly chosen and for which the given data points are noise free. Verify that the approximation error with respect to the true trajectory is decreasing as more exact interpolation points are added.

Summary and discussion

Errors-in-variables smoothing can be viewed alternatively as computing the distance, called *misfit*, of w to $\mathscr{B}$

$$\mathrm{misfit}(w, \mathscr{B}) := \min_{\widehat{w}} \ \|w - \widehat{w}\|_S \quad \text{subject to} \quad \widehat{w} \in \mathscr{B}|_T.$$

The misfit quantifies the lack of fit between w and $\mathscr{B}$. It is the size (measured in the $\|\cdot\|_S$-norm) of the smallest perturbation on w that renders w an exact trajectory of $\mathscr{B}$. A recursive algorithm for model-based errors-in-variables smoothing is given in [9, Section 3].

The opposite of missing data is exactly known data. Exactly known data is imposed by equality constraints. They are the strongest type of prior knowledge about the data—complete confidence in the given values. Exercises 33 and 34 demonstrates how adding interpolation points in the smoothing problem (*i.e.*, adding prior knowledge) improves the estimation accuracy. The next section imposes the priori knowledge that the input is known exactly. The result is the ordinary Kalman smoother.

7 Kalman smoothing

In this section, we consider optimal smoothing in the ARMAX setup. The problem is known as Kalman smoothing because it was first solved by the celebrated Kalman smoother.

The Kalman smoothing problem differs from the errors-in-variables smoothing problem in to ways:

1. the input u is exactly known while the output y is still observed with additive noise, and
2. an unobserved input e, called *disturbance*, that is a zero-mean white Gaussian stochastic process acts on the system.

Let $\mathscr{B}_{\mathrm{ext}}$ be the behavior of the extended variable $w_{\mathrm{ext}} := \left[\begin{smallmatrix} w \\ e \end{smallmatrix}\right]$.

The disturbance e is not observed, however, the assumption that it is a zero-mean white Gaussian stochastic process makes it different from the missing input e considered before. The estimate $\widehat{e}$ of a missing input does not appear in the cost function and is determined only from the interpolation condition $(w, \widehat{e}) \in \mathscr{B}_{\mathrm{ext}}$, while the estimate $\widehat{e}$ of a stochastic input appears in the cost function with the term $\|\widehat{e}\|_{S_e}$ and is estimated from the prior knowledge that it is as small as possible in the $\|\cdot\|_{S_e}$-norm. More pragmatically, using `ddint`, a missing input is specified by `ed = NaN`, while a stochastic input is specified by `ed = 0` and the block S_e of the weight matrix S that corresponds to e.

In order to illustrate the application of `ddint` to Kalman smoothing, first we choose an extended data-generating system $\mathscr{B}_{\mathrm{ext}}$ with variables $w_{\mathrm{ext}} = (u, y, e)$

```
n = 5; p = 1; mu = 1; me = 1; m = mu + me; sy = 0.1; se = 0.1;
Bext = drss(n, p, m); % Bext = Bext * diag(1 ./ dcgain(Bext));
```

and generate a data trajectory w_{d}

```
Td = 100;
ud = rand(Td, mu); ed = se * randn(Td, me); xd_ini = rand(n, 1);
yd = lsim(Bext, [ud ed], [], xd_ini); wextd = [ud yd ed];
```

and a to-be-smoothed trajectory w

```
T = 10; u = rand(T, mu); e = randn(T, me); x_ini = rand(n, 1);
y0 = lsim(Bext, [u e], [], x_ini); yt = randn(T, p);
y = y0 + sy * norm(y0) * yt / norm(yt);
```

The Kalman smoothing problem is specified as a data-driven interpolation/approximation problem (3) by setting

```
wext = [u y zeros(T, me)];
S = [inf(T, mu), (1 / norm(y - y0)) * ones(T, p), (1 / norm(e)) * ones(T, me)];
```

The smoothed output `yh` is obtained then by

```
wexth = ddint(wextd, wext, S); yh = wexth(:, mu+1:mu+p);
```

In order to verify the result, we use the relative estimation error

```
error = @(yh) norm(y0 - yh) / norm(y0);
```

and compare the estimate `yh` with the estimate obtained by a model-based method implemented in the function `predict` from the System Identification toolbox of Matlab

```
yh_mb = predict(idss(Bext(1,2)), iddata(y, u), T); yh_mb = yh_mb.OutputData;
```

The results are

```
[error(yh) error(yh_mb)] % -> 0.0956    0.3544
```

Exercises

Exercise 35 (Kalman smoothing without disturbance). Revise the solution of the Kalman smoothing problem for the case when w is not exact due to measurement noise on the output only, *i.e.*, when there is no disturbance signal.

Exercise 36 (Kalman smoothing with exactly known initial conditions). In the setup of Exercise 35 (measurement noise without disturbance), add the prior knowledge that the initial conditions are given and are exact. (You can assume zero initial conditions.)

Exercise 37 (Kalman smoothing without measurement noise). Revise the solution of the Kalman smoothing problem for the case when there is no measurement noise on the output.

Summary and discussion

In the errors-in-variables setup the lack of fit between w and $\mathscr{B}$ is due to the measurement errors. The appropriate estimation criterion in this case is the misfit. When the lack of fit between w and $\mathscr{B}$ is due to disturbance and the measurement error on the output, the appropriate estimation criterion is the *latency*, defined as

$$\mathrm{latency}(w, \mathscr{B}_{\mathrm{ext}}) := \min_{(w,\widehat{e}) \in \mathscr{B}_{\mathrm{ext}}|_T} \|\widehat{e}\|.$$

The latency computation corresponds to the Kalman smoothing problem without measurement noise (Exercise 37).

In [6] a combined misfit–latency approach for system identification is proposed. The misfit–latency approach has as special cases the errors-in-variables and ARMAX setups and allows one to consider the most general problem where the uncertainty is due to both disturbance as well as measurement errors on both inputs and outputs. Data-driven smoothing in the combined misfit–latency setting can be done with `ddint`.

Exercise 38 (Mixed misfit–latency smoothing). Using `ddint`, solve the data-driven smoothing problem

$$\text{minimize} \quad \text{over } \widehat{w} \text{ and } \widehat{e} \quad \left\| \begin{bmatrix} w - \widehat{w} \\ \widehat{e} \end{bmatrix} \right\|_S \quad \text{subject to} \quad \begin{bmatrix} \widehat{w} \\ \widehat{e} \end{bmatrix} \in \mathscr{B}_{\text{ext}}|_T,$$

where the system $\mathscr{B}_{\text{ext}}$ is implicitly specified by a trajectory $w_{\text{d,ext}} \in \mathscr{B}_{\text{ext}}|_{T_\text{d}}$. Apply the solution on an example.

8 Data-driven control

The control problems considered are open-loop. The design specifications in the first two problems are constraints on the trajectory, while the design specification of the third problem is an optimality criterion.

State transition

A basic control problem is state transition: transition from a given "past" trajectory $w_\text{p} \in \mathscr{B}|_{T_\text{p}}$ to a given "future" trajectory $w_\text{f} \in \mathscr{B}|_{T_\text{f}}$ via a "control" trajectory $w_\text{c} \in \mathscr{B}|_{T_\text{c}}$, *i.e.*, find w_c, such that $w_\text{p} \wedge w_\text{c} \wedge w_\text{f} \in \mathscr{B}_{T_\text{p}+T_\text{c}+T_\text{f}}$, see Figure 2.

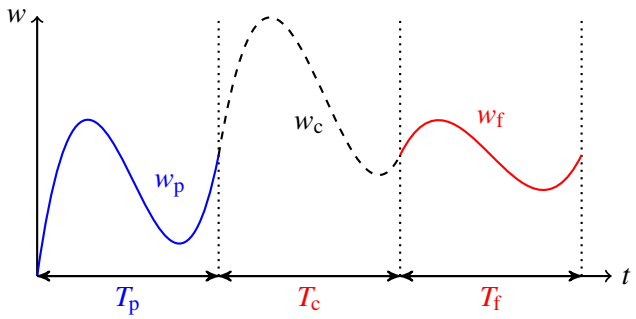

Figure 2: The state transition control problem aims to connect the given "past" w_p and "future" w_f trajectories via a "control" w_c trajectory.

In the simulation example we take a random stable system

```
n = 5; p = 1; m = 1; q = m + p; Tc = 10;
B = drss(n, p, m);
```

and a random data trajectory

```
Td = 100; ud = rand(Td, m); xd_ini = rand(n, 1);
yd = lsim(B, ud, [], xd_ini); wd = [ud yd];
```

The state transition problem is specified by the past and future trajectories

```
wp = rand(n, q); wf = zeros(n, q);
```

In order to cast the state transition problem as a missing data estimation problem, we define $w := w_\text{p} \wedge w_\text{c} \wedge w_\text{f}$ with w_p and w_f the given trajectories and w_c as missing/to-be-completed:

```
w = [wp; NaN(Tc, q); wf];
S = [inf(n, q); zeros(Tc, q); inf(n, q)];
[wh, N] = ddint(wd, w, S); wc = wh(n+1:n+Tc, :);
```

In the example there is a nonunique control trajectory w_c achieving the desired state transition. The set of possible control trajectories has dimension

```
size(N, 2) % -> 5
```

One way of utilizing the control freedom is by choosing the minimum energy control. The minimum energy control is computed by `ddint` as follows:

```
w = [wp; [zeros(Tc, m) NaN(Tc, p)]; wf];
S_me = [inf(n, q); [ones(Tc, m) zeros(Tc, p)]; inf(n, q)];
[wh_me, N_me] = ddint(wd, w, S_me); wc = wh(n+1:n+Tc, :);
```

Trajectory planning

The problem considered next is to find a trajectory that passes through given interpolation "points" $p_1, \ldots, p_K$ at given moments of time $t_1, \ldots, t_K$. By "points" we mean any subset of elements of the variables w (the whole w, the output y, or any subset of inputs and outputs). Moreover, the subset of variables need not be fixed for all moments of time.

Depending on the number of interpolation points and the number of variables involved, there may be infinitely many, one, or no feasible trajectory. This is explored in Exercise 39. In control problems, typically there are infinitely many feasible trajectories, in which case an optimal one is selected by specifying an optimality criterion.

In order to illustrate the application of `ddint` for trajectory planning, we choose a random stable system

```
n = 5; p = 2; m = 1; q = m + p; Tc = 10;
B = drss(n, p, m);
```

and a data trajectory

```
Td = 100; ud = rand(Td, m); xd_ini = rand(n, 1);
yd = lsim(B, ud, [], xd_ini); wd = [ud yd];
```

In addition to passing through the interpolation points, the desired trajectory is required to start and to end at the zero state, thus before adding the interpolation conditions the problem is defined by trajectory `w` and weight `S`

```
w = [zeros(n, q); NaN(Tc, q); zeros(n, q)];
S = [inf(n, q); zeros(Tc, q); inf(n, q)];
```

The interpolation points are $y(n+3) = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ and $y(n+6) = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$. They are added to the problem specification as follows

```
Iy = (m+1):q
w(n + 3, Iy) = [1 0]; S(n + 3, Iy) = inf;
w(n + 6, Iy) = [0 1]; S(n + 6, Iy) = inf;
```

Now the set of all trajectories satisfying the constraints can be computed by evoking `ddint`

```
[wh, N] = ddint(wd, w, S);
```

It turns out that is in the particular example there is one dimensional solution set

```
size(N, 2) % -> 1
```

In order to find the minimum energy control that satisfies the constraints, we modify the problem as follows

```
w(n+1:n+Tc, 1:m) = 0; S(n+1:n+Tc, 1:m) = 1;
[wh_me, N_me] = ddint(wd, w, S);
```

Tracking control

The tracking control problem is mathematically equivalent to the errors-in-variables smoothing problem. The difference is that in the errors-in-variables smoothing problem `w` is a noise corrupted trajectory of the system, while in the tracking control problem `w` together with `S` specify the control objective and `w` need not be a trajectory of the system. In the example, we choose a random stable plant

```
n = 5; p = 1; m = 1; B = drss(n, p, m);
```

represented for the data-driven control by a trajectory

```
Td = 100; ud = rand(Td, m); xd_ini = rand(n, 1);
yd = lsim(B, ud, [], xd_ini); wd = [ud yd];
```

The to-be-tracked trajectory is zero input (minimum energy control) and constant output (step tracking)

```
Tc = 10; u = zeros(Tc, m); w = [u ones(Tc, p)];
```

The cost function minimized by the controller is

```
S = [ones(Tc, m) 100 * ones(Tc, p)];
e = @(wh) norm(S .* (w - wh), 'fro') / norm(w);
```

i.e., the control objective is minimum energy least-squares tracking of a constant unit output with time horizon $T_c = 10$.

Assuming zero initial conditions, the actual trajectory achieved applying the control input on the plant is

```
wha = @(wh) [wh(:, 1:m), lsim(B, wh(:, m+1:end))];
```

so that, the achieved performance is

```
ea  = @(wh) e(wha(wh));
```

Using `ddint` we solve the open-loop tracking control problem with zero initial conditions as follows:

```
wh = ddint(wd, [zeros(n, 2); w], [inf(n, 2); S]);
wh = wh(n+1:end, :);
```

The objective function and the actual achieved performance coincide:

```
[e(wh) ea(wh)] % -> 44.4719   44.4719
```

Exercises

Exercise 39 (Constraints and degrees of freedom). In the state transition and trajectory planning problems, what is the dimension of the set of solutions? How many interpolation constraints can be added in the trajectory planning problem retaining feasibility?

Exercise 40 (Uncertain initial conditions). Relax the prior knowledge of exactly known initial condition to uncertain initial conditions and observe the effect on the performance.

Exercise 41 (Unknown initial conditions). The opposite extreme of exactly known initial condition is unknown initial conditions. Compare the performance of the tracking control with exact knowledge of the initial condition with the one of unknown initial conditions.

Summary and discussion

The controllability property defined in the behavioral setting is a necessary and sufficient condition for existence of solution of the state transition problem. The trajectory planning problem occurs in robotics applications [4]. Closed-loop data-driven control can be achieved by embedding the open-loop data-driven control methods in predictive control. This led to the data enabled predictive control (DeePC) method [1]. In the mini-projects we consider the case of inexact data `wd` and compare the data-driven methods with model-based ones.

A Structured matrices

Studying the behavior of discrete-time linear dynamical systems over a finite horizon is an application of linear algebra. When the systems, in addition to linear, are also time-invariant, the matrices involved have Hankel structure. The exercises in this section introduce three types of structured matrices—Hankel, mosaic Hankel, and polynomial multiplication. These structures are used for analysis, identification, and control of linear time-invariant systems.

The term *structure* in "structured matrix" refers to a function $\mathscr{S} : \mathbb{R}^{n_p} \to \mathbb{R}^{m\times n}$ mapping a vector $p \in \mathbb{R}^{n_p}$, called *structure parameter vector*, to an $m \times n$ matrix S. We say that a matrix D has the structure $\mathscr{S}$ if it is in the image of $\mathscr{S}$, *i.e.*, there is a p, such that $D = \mathscr{S}(p)$. In the exercises, you will

- prove that the Hankel, mosaic Hankel, and polynomial multiplication structures are linear (*i.e.*, $\mathscr{S}$ is linear);
- derive the orthogonal projector $\Pi_{\mathscr{S}}$ on image $\mathscr{S}$, *i.e.*, a function $\Pi_{\mathscr{S}} : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ solving the problem

$$\text{minimize} \quad \text{over } \widehat{p} \quad \|D - \mathscr{S}(\widehat{p})\|_{\mathrm{F}}, \tag{8}$$

  where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm; and
- write functions that implement $\mathscr{S}$ and $\Pi_{\mathscr{S}}$.

Hankel structure

A Hankel matrix $\mathscr{H}_L(w)$ with L block rows, constructed from the finite vector time-series

$$w = \big(w(1),\dots,w(T)\big), \quad w(t) \in \mathbb{R}^q, \tag{9}$$

is defined as

$$\mathscr{H}_L(w) := \begin{bmatrix} w(1) & w(2) & \cdots & w(T-L+1) \\ w(2) & w(3) & \cdots & \\ \vdots & \vdots & & \vdots \\ w(L) & w(\ell+2) & \cdots & w(T) \end{bmatrix} \in \mathbb{R}^{qL\times(T-L+1)}.$$

Exercise 42 (Hankel matrix constructor `blkhank` and projector `Pblkhank`).

1. Explain how $\mathscr{H}_L$ defines a matrix structure (*i.e.*, what are $\mathscr{S}$ and p).
2. Write a function `blkhank` that takes as inputs w and L and returns as an output $\mathscr{H}_L(w)$. (w is represented by a $q\times T$ matrix `w`, where `w(:, t)` is equal to $w(t)$.)
3. Show that the Hankel structure $\mathscr{H}_L$ is linear.
4. Derive the orthogonal projector $\Pi_{\mathscr{H}_L}$ on image $\mathscr{H}_L$.
5. Write a function `Pblkhank` that takes as inputs an $qL\times(T-L+1)$ matrix D and an integer L and returns as an output $\widehat{w}$, such that $\mathscr{H}_L(\widehat{w}) = \Pi_{\mathscr{H}_L} D$.

Mosaic Hankel structure

The mosaic Hankel matrix with L block rows, constructed from the set of finite vector time-series

$$w := \{w^1,\dots,w^N\}, \quad w^i = \big(w^i(1),\dots,w^i(T_i)\big), \quad w^i(t) \in \mathbb{R}^q$$

is defined as

$$\mathscr{H}_L(w) := \begin{bmatrix} \mathscr{H}_L(w^1) & \cdots & \mathscr{H}_L(w^N) \end{bmatrix} \in \mathbb{R}^{qL\times\sum_{i=1}^N (T_i-L+1)}.$$

Exercise 43 (Mosaic Hankel matrix constructor `moshank` and projector `Pmoshank`).

1. Explain how $\mathscr{H}_L$ defines a matrix structure (*i.e.*, what are $\mathscr{S}$ and p).
2. Write a function `moshank` that takes as inputs w and L and returns as an output $\mathscr{H}_L(w)$. (The set of time series w is represented by a $1\times N$ cell array `w` of $q\times T_i$ matrices `w{i}`, where `w{i}(:, t)` is equal to $w^i(t)$.)
3. Show that the mosaic Hankel structure $\mathscr{H}_L$ is linear.
4. Derive the orthogonal projector $\Pi_{\mathscr{H}_L}$ on image $\mathscr{H}_L$.
5. Write a function `Pmoshank` that takes as inputs an $qL\times\sum_{k=1}^N n_i$, $n_i := T_i - L + 1$ matrix D and $n = \begin{bmatrix} n_1 & \cdots & n_N \end{bmatrix}$ and returns as an output $\widehat{w}$, such that $\mathscr{H}_L(\widehat{w}) = \Pi_{\mathscr{H}_L} D$.

Polynomial multiplication structure

The multiplication matrix $\mathscr{M}_T(R)$ with T block columns related to the polynomial

$$R(z) = R_0 + R_1 z + \cdots + R_\ell z^\ell = \begin{bmatrix} R^1(z) \\ \vdots \\ R^g(z) \end{bmatrix} = \begin{bmatrix} R_0^1 + R_1^1 z + \cdots + R_{\ell_1}^1 z^{\ell_1} \\ \vdots \\ R_0^g + R_1^g z + \cdots + R_{\ell_g}^g z^{\ell_g} \end{bmatrix} \in \mathbb{R}^{g\times q}[z] \tag{10}$$

is defined as

$$\mathscr{M}_T(R) := \begin{bmatrix} \mathscr{M}_T(R^1) \\ \vdots \\ \mathscr{M}_T(R^g) \end{bmatrix}$$

where

$$\mathscr{M}_T(r) := \begin{bmatrix} r_0 & r_1 & \cdots & r_\ell & & & \\ & r_0 & r_1 & \cdots & r_\ell & & \\ & & \ddots & \ddots & & \ddots & \\ & & & r_0 & r_1 & \cdots & r_\ell \end{bmatrix} \in \mathbb{R}^{(T-\ell)\times qT}. \tag{11}$$

In Matlab, the polynomial R is represented by a $g \times q(\ell+1)$ matrix $\begin{bmatrix} R_0 & R_1 & \cdots & R_\ell \end{bmatrix}$.

Exercise 44 (Polynomial multiplication matrix constructor `mulmat` and projector `Pmulmat`).

1. Explain how $\mathscr{M}_T$ defines a matrix structure (*i.e.*, what are $\mathscr{S}$ and p).
2. Write a function `multmat` that takes as inputs R and T and returns as an output $\mathscr{M}_T(R)$.
3. Show that the polynomial multiplication structure $\mathscr{M}_T$ is linear.
4. Derive the orthogonal projector $\Pi_{\mathscr{M}_T}$ on image $\mathscr{M}_T$.
5. Write a function `Pmultmat` that takes as inputs $g(T-\ell) \times qT$ matrix D, p, and q and returns as an output $\widehat{R}$, such that $\mathscr{M}_T(\widehat{R}) = \Pi_{\mathscr{M}_T} D$.

Summary and discussion

We focused on the "mechanics" of the matrix structures $\mathscr{H}_L$ and $\mathscr{M}_T$ without explanation why they are important. The Hankel and mosaic Hankel matrices are used in the data-driven representation of the finite horizon behavior of a linear time-invariant system. The polynomial multiplication matrix $\mathscr{M}_T$, as its name suggests, is used for computing the product of polynomials. Indeed, $c := b\mathscr{M}_T(a)$ defines the product of the polynomials defined by a and b. Check it numerically:

```
a  = 1:4; b = 1:4;
c  = conv(a, b);                % -> 1     4     10    20    25    24    16
c_ = b * multmat([1 2 3 4], 7) % -> 1     4     10    20    25    24    16
```

A polynomial (10) is represented by the matrix of its coefficients, which can represent also the matrix-valued sequence $(R_0, R_1, \cdots, R_\ell)$. With some abuse of notation, we use the same letter R to denote the polynomial, the matrix, and the corresponding sequence of coefficients. Depending on the context, either the polynomial or the sequence are the primary object of interest, while the matrix is the way of store and manipulate it analytically and computationally.

References

[1] J. Coulson, J. Lygeros, and F. Dörfler. "Distributionally robust chance constrained data-enabled predictive control". In: *IEEE Trans. Automat. Contr.* 67 (2022), pp. 3289–3304.

[2] F. Dörfler, J. Coulson, and I. Markovsky. "Bridging direct & indirect data-driven control formulations via regularizations and relaxations". In: *IEEE Trans. Automat. Contr.* (2023). DOI: 10.1109/TAC.2022.3148374.

[3] P. Dreesen and I. Markovsky. "Data-Driven Simulation Using The Nuclear Norm Heuristic". In: *In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. Brighton, UK, 2019. DOI: 10.1109/icassp.2019.8682993.

[4] E. Frazzoli, M. A. Dahleh, and E. Feron. "Real-time motion planning for agile autonomous vehicles". In: *Proc. American Control Conf.* Vol. 1. 2001, pp. 43–49.

[5] R. Hamming. *Numerical Methods for Scientists and Engineers*. Dover Publications, 1962.

[6] P. Lemmerling and B. De Moor. "Misfit versus latency". In: *Automatica* 37 (2001), pp. 2057–2067.

[7] I. Markovsky. *Low-Rank Approximation: Algorithms, Implementation, Applications*. 2nd edition. Springer, 2019. ISBN: 978-3-319-89619-9. DOI: 10.1007/978-3-319-89620-5.

[8] I. Markovsky. "On the most powerful unfalsified model for data with missing values". In: *Systems & Control Lett.* 95 (2016), pp. 53–61. DOI: 10.1016/j.sysconle.2015.12.012.

[9] I. Markovsky and B. De Moor. "Linear dynamic filtering with noisy input and output". In: *Automatica* 41.1 (2005), pp. 167–171.

[10] I. Markovsky and F. Dörfler. "Data-driven dynamic interpolation and approximation". In: *Automatica* 135 (2022), p. 110008. DOI: 10.1016/j.automatica.2021.110008.

[11] I. Markovsky and P. Rapisarda. "Data-driven simulation and control". In: *Int. J. Control* 81.12 (2008), pp. 1946–1959.

[12] I. Markovsky and K. Usevich. "Structured low-rank approximation with missing data". In: *SIAM J. Matrix Anal. Appl.* 34.2 (2013), pp. 814–830. DOI: 10.1137/120883050.

[13] J. C. Willems. "From time series to linear system—Part II. Exact modelling". In: *Automatica* 22.6 (1986), pp. 675–694.