The Behavioral Toolbox

Ivan Markovsky

Summary and downloads

The Behavioral Toolbox is a collection of Matlab functions for analysis and design of dynamical systems using the behavioral approach to systems theory and control. It implements newly emerged nonparameteric data-driven methods for linear time-invariant systems.

- tar file of the functions
- documentation
- paper

Tutorial

In order to make the behavioral approach computationally feasible, we consider discrete-time systems and restrict the behavior to a finite horizon. The restricted behavior of a linear time-invariant system is a shift-invariant subspace. The methods developed in the toolbox use a basis for this subspace as a nonparametric representation of the system. The system analysis, signal processing, and control methods take as an input a basis. The identification methods take as an input a trajectory of the system. The code is extracted in a script file `demo.m`.

Nonparameteric representation of the restricted behavior

This section introduces the functions of the toolbox for creating and using the nonparametric representation of the system. In order to demonstrate and test these functions on numerical examples, we create a random linear time-invariant system defined by an input/state/output representation—a Matlab's `ss` object `B`:

```
m = 2; p = 1; q = m + p; n = 3; B = drss(n, p, m);
```

The construction of the orthonormal basis `BT` for the finite-horizon behavior from an input/state/output representation of the system is done by the function `B2BT`:

```
T = 10; BT = B2BT(B, T);
```

The function `B2BT` implements a data-driven approach. Instead of using the parameters of the state-space representation of the system, it computes a random trajectory (using `lsim`), forms a Hankel matrix (using the toolbox's function `hank`), and computes an orthonormal basis for its image (using `svd`). Alternative functions to `B2BT` that implement the model-based approach are `ss2BT` and `R2BT`. As the names suggest, `ss2BT` uses a state-space representation, while `R2BT` uses a kernel representation to compute the basis.

In order to verify that the output `BT` of `B2BT` is correct, first, we verify that its dimension is correct:

```
check(size(BT, 2) == m * T + n)
```

The function `check` prints "PASS" if its argument is true and "FAIL" otherwise.

Then, we check that a random trajectory of the system is in the image of `BT`. The function `B2w` selects a random trajectory of `B`. Its output is a $T \times q$ matrix, where T is the number of samples and q is the number of variables. Checking if w is a trajectory is a common task and is implemented in a function `w_in_B`:

```
w = B2w(B, T); check(w_in_B(w, B))
```

The columns of `BT` are finite trajectories of the system. The function `BT2W` extracts them into a cell array `W`:

```
W = BT2W(BT, q); check(all(w_in_B(W, B)))
```

The function `w_in_B` as well as many other functions of the toolbox allow specification of multiple trajectories by a cell array as well as a single trajectory.

A q-variate, T-samples long signal $w := \big(w(1),\dots,w(T)\big)$ is represented in the toolbox by:

- $T \times q$ matrix `w`, where `w(t, :)` $= w^\top(t)$ or
- $qT \times 1$ vector `w` with the stacked consequitive samples.

The matrix format has the advantage that the number of variables q can be deduced from the size. However, it is incompatible with the data-driven representation, which uses the vector format. The conversion from the matrix to the vector format is done by `vec(w')`, where `vec` is column-wise vectorization, and the conversion from the vector to the matrix format is done by `reshape(w, q, T)'`. Functions accepting B_T as an input require q to be passed as an extra input argument.

When the $T \times q$ matrix representation is used, multiple trajectories are stored in a cell array with N elements, the ith element of which is a $T_i \times q$ matrix representing the ith trajectory w^i. When the $Tq \times 1$ vector representation is used and $T_1 = \cdots = T_N =: T$, multiple trajectories are stored in a $qT \times N$ matrix, the ith column of which represents the ith trajectory w^i.

The computation of a basis B_T from a trajectory of the system requires finding the dimension of of the finite-horizon behavior $\mathscr{B}|_T$. Under standard conditions on w_d that are almost certainly satisfied for a sufficiently long random trajectory $w_\mathrm{d} \in \mathscr{B}|_T$,

$$\dim \mathscr{B}|_T = \operatorname{rank} \mathscr{H}_T(w_\mathrm{d}).$$

Thus, $\dim \mathscr{B}|_T$ can be found by rank computation. A robust way of computing $\operatorname{rank} \mathscr{H}_T(w_\mathrm{d})$ is thresholding the singular values of $\mathscr{H}_T(w_\mathrm{d})$. The functions of the toolbox that construct B_T from data therefore have an optional threshold parameter `tol` with default value:

```
if ~exist('tol', 'var') || isempty(tol), tol = 1e-8; end % <default-tol>
```

Alternatively, the user can specify the model's complexity, in which case the dimension is computed as $mT + n$.

Input/output partitionings

A partitioning of the variables $w(t)$ into inputs $u(t)$ and outputs $y(t)$ is defined by a permutation matrix $\Pi \in \mathbb{R}^{q\times q}$ as follows:

$$w \mapsto (u,y) : \begin{bmatrix} u \\ y \end{bmatrix} := \Pi w \qquad \text{and} \qquad (u,y) \mapsto w : \; w = \Pi^\top \begin{bmatrix} u \\ y \end{bmatrix}.$$

Πw reorders the variables w, so that the first m variables are the inputs and the remaining $p := q - m$ variables are the outputs. The permutation matrix Π is specified in the functions of the toolbox by a vector `io`, such that `w(io)` $\mapsto$ `[u; y]`. The restricted behavior with permuted variables is created with the function `BT2BT`:

```
io = flip(1:q); BTp = BT2BT(BT, q, io);
```

Indeed, the trajectory $w' := \Pi w$ with permuted variables belongs to the image of `BTp`:

```
check(w_in_B(w(:, io), BTp))
```

The function `BT2UYT` extracts from the basis of the restricted behavior $\mathscr{B}|_T$ its input and output components:

```
[UT, YT] = BT2UYT(BT, m, p);
```

`BT2UYT` and other functions of the toolbox that require an input/output partitioning but do not accept a specification for it assume $\Pi = I_q$:

```
if ~exist('io', 'var') || isempty(io), io = 1:q; end % <default-io>
```

For a general input/output partitioning, defined by `io`, call the functions with input `BT2BT(BT, q, io)` instead of `BT`.

The inverse transformation `UYT2BT` reconstructs $\mathscr{B}|_T$ from its input and output components:

```
check(norm(BT - UYT2BT(UT, YT, m, p)) == 0)
```

The number of inputs m is uniquely defined by the behavior. However, an input/output partitioning of the variables is in general not unique. The currently available methods in the literature for finding an input/output partitioning of a system $\mathscr{B}$ are based on parametric representations of the system (e.g., the kernel representation). The toolbox implements a data-driven method for finding an input/output partitioning directly from the finite-horizon behavior $\mathscr{B}|_T$. The function `is_io` checks if a given partitioning of the variables is a possible input/output partitioning and the function `BT2IO` finds all possible input/output partitionings of the system. In order to test `is_io` and `BT2IO`, consider the following single-input single-output system

```
Bp = ss(tf([0 1], [1 1], 1));
```

that has an input/output partitioning $w = \left[\begin{smallmatrix} u \\ y \end{smallmatrix}\right]$ but not $w = \left[\begin{smallmatrix} y \\ u \end{smallmatrix}\right]$.

```
BpT = B2BT(Bp, T);
check(is_io(BpT, 2, [1 2]) == true)
check(is_io(BpT, 2, [2 1]) == false)
check(all(BT2IO(BpT, 2) == [1 2]))
```

Subbehaviors

The following subbehaviors of $\mathscr{B}$ are of special interest:

- $\mathscr{Y}_0$ — *zero-input subbehavior*, i.e., the set of transient responses,
- $\mathscr{U}_0$ — *zero-output subbehavior*, i.e., the set of inputs blocked by the system,
- $\mathscr{B}_0$ — *zero initial conditions subbehavior*, i.e., the set of zero initial conditions trajectories,
- $\mathscr{B}_\text{c}$ — *controllable subbehavior*, i.e., the set of trajectories that are patchable with zero past trajectory, and
- $\mathscr{B}_\text{p}$ — *periodic subbehavior*, i.e., the set of periodic trajectories.

Orthonormal bases for their restrictions to $[1, T]$ are computed from $\mathscr{B}|_T$ by the following functions:

```
Y0 = BT2Y0(BT, q);
U0 = BT2U0(BT, q);
B0 = BT2B0(BT, q);
BC = BT2BC(BT, q);
```

Let's verify that a free response of $\mathscr{B}$ is in the image of `Y0`:

```
y0 = initial(B, rand(n, 1), T-1); check(w_in_B(y0, Y0))
```

Next, we verify that $w = \left[\begin{smallmatrix} u \\ 0 \end{smallmatrix}\right]$ with u in the image of `U0` is a trajectory of the system:

```
u0 = reshape(U0 * rand(size(U0, 2), 1), m, T)';
w0 = [u0 zeros(T, p)]; check(w_in_B(w0, BT))
```

Next, we verify that the initial conditions of a random trajectory in the zero initial conditions subbehavior are zero

```
T0 = size(B0, 1) / q;
w0 = reshape(B0 * rand(size(B0, 2), 1), q, T0)';
xini = w2xini(w0, B); check(norm(xini) < tol)
```

Using the zero initial conditions subbehavior $\mathscr{B}_0 \subset \mathscr{B}|_T$, we can find the zero initial conditions input-to-output map $H_T : u|_T \mapsto y|_T$. Let B_0 be the matrix of the basis vectors for $\mathscr{B}_0$ and let U_0, Y_0 be the submatrices of B_0 corresponding to the inputs and the outputs. Then, $H_T = Y_0 U_0^{-1}$. The method is implemented in the function `BT2HT`:

```
[HT, T] = BT2HT(BT, q);
```

The $pT \times mT$ matrix H_T is the finite-horizon representation of the transfer function $H(z)$ of the system $\mathscr{B}$ corresponding to the input/output partitioning. In particular, it has lower-triangular block-Toeplitz structure:

```
check(norm(HT - convm(B, T)) < tol)
```

Analysis

In this section, we find properties of the system directly from its restricted behavior, rather than from parametric representations as done by classical analysis methods. The properties considered are: complexity, controllability, H_∞-norm, and distance between systems.

System's complexity

The complexity $\mathbf{c}(\mathscr{B})$ of a linear time-invariant system $\mathscr{B}$ is defined as the triple: number of inputs $\mathbf{m}(\mathscr{B})$, lag $\ell(\mathscr{B})$, and order $\mathbf{n}(\mathscr{B})$. Although in the classical setting the order $\mathbf{n}(\mathscr{B})$ is defined via a minimal state-space representation, it is a property of the system $\mathscr{B}$ and can be computed directly from the restricted behavior $\mathscr{B}|_T$ (provided $T \geq \ell(\mathscr{B}) + 1$). Also, the number of inputs $\mathbf{m}(\mathscr{B})$ can be computed from $\mathscr{B}|_T$ without reference to a particular input/output representation. The method for finding $\mathbf{m}(\mathscr{B})$ and $\mathbf{n}(\mathscr{B})$ from $\mathscr{B}|_T$ is based on the theoretical dimension $mT + n$. Evaluating $\dim \mathscr{B}|_{t_i}$ for $t_1 \neq t_2 \geq \ell(\mathscr{B})$, e.g., $t_1 = T$ and $t_2 = T - 1$, we obtain the system of equations

$$\begin{bmatrix} T & 1 \\ T-1 & 1 \end{bmatrix} \begin{bmatrix} m \\ n \end{bmatrix} = \begin{bmatrix} \dim \mathscr{B}|_T \\ \dim \mathscr{B}|_{T-1} \end{bmatrix},$$

from which $m = \mathbf{m}(\mathscr{B})$ and $n = \mathbf{n}(\mathscr{B})$ can be found. The resulting method is implemented in the function `BT2c`:

```
[ch, mh, ellh, nh] = BT2c(BT, q); check(all([mh nh] == [m n]))
```

A model-based method for computing the lag of the system is implemented in the function `lag`:

```
ell = lag(B); check(ellh == ell)
```

As discussed in the next section, the complexity connects nonparameteric and parametric representations and is a critical first step in finding a parametric representation of the system.

`BT2c` finds $\dim \mathscr{B}|_{T-1}$ and $\dim \mathscr{B}|_T$ by rank computation, i.e., thresholding of the singular values using a user defined threshold `tol`.

Distance between systems

The function `Bdist` computes distance between systems. The distance is defined as the principal angle between the finite-horizon behaviors of the systems with default horizon $T = 100$. Here is an example:

```
Bp = B; Bp.a = Bp.a + 0.01 * randn(n); d = Bdist(B, Bp);
```

The distance measure `Bdist` is used by the function `equal` for checking if two systems are equal:

```
Bp = ss2ss(B, rand(n)); check(equal(B, Bp))
```

Controllability

The controllable subbehavior $\mathscr{B}_c \subset \mathscr{B}|_T$ for $T \geq \ell(\mathscr{B})$ is $mT+n$-dimensional if and only if $\mathscr{B}$ is controllable. This leads to a data-driven controllability test that is implemented in the function `isunctr`:

```
Bp = ss([0.5 1; 0 0.25], [1; 0], [1 1], 1, -1);
n_unctr = isunctr(Bp); check(n_unctr == 1)
```

Moreover, $mT+n-\dim \mathscr{B}_c|_T$ is the number of uncontrollable modes of $\mathscr{B}$. Based on $\mathscr{B}_c$, a quantitative test for controllability—a distance to uncontrollability—is implemented in the function `distunctr`. Here is an example:

```
Bp = ss([0.5 1; 0 0.25], [1; 1e-5], [1 1], 1, -1);
d = distunctr(Bp); % -> 1e-7
```

H_∞-norm

Using the zero initial conditions input-to-output map H_T of $\mathscr{B}$ for a given input/output partitioning allows us to compute the finite-horizon H_∞-norm of $\mathscr{B}$. The method is implemented in the function `BT2Hinf`:

```
[Hinf, ~, uinf] = BT2Hinf(BT, q); C = convm(B, T);
check(abs(Hinf - norm(C)) / norm(C) < tol)
```

Parametric representations

So far, the tutorial reviewed functions related to and based on the nonparametric representation of the restricted behavior $\mathscr{B}|_T$. This section shows an application of the approach for computing parametric representations.

Computing a kernel representation from $\mathscr{B}|_T$ is essentially applying the `null` function on $B_{\ell+1}^\top$. The method is implemented in the functions `B2R` and `BT2R` (`B2R` constructs `BT` from `B` and calls `BT2R`):

```
R = B2R(B); R = BT2R(BT, q);
```

The inverse operation—finding the restricted behavior from a kernel or a state-space representation—is done by the functions `R2BT` and `ss2BT`, which implement the model-based approach, i.e., they construct B_T from the model parameters R and (A,B,C,D), respectively.

```
BT_ = R2BT(B2R(B), q, T); check(equal(B, BT_))
BT_ = ss2BT(B, T);        check(equal(B, BT_))
```

The basis computed by `ss2BT` is not orthonormal. It consists of observability and convolution matrices.

Contrary to `BT2R` (which is essentially Matlab's `null` function), computing a state-space representation from the restricted behavior is nontrivial. Indeed, it requires to do 1) state construction and 2) detect a possible input/output partitioning of the variables. The data-driven approach for these operations is implemented in the function `BT2ss`:

```
check(equal(B, BT2ss(BT, q)))
```

Similarly, the transformation from a kernel representation to a state-space representation is nontrivial. A data-driven method for this operation is implemented in the function `R2ss`:

```
check(equal(B, R2ss(R, q)))
```

For multi-output systems, the function `B2R` computes a *nonminimal* kernel representation. Computing a minimal kernel representation or converting a nonminimal kernel representation to a minimal one are also nontrivial operations. Data-driven methods for them are implemented in the functions `BT2Rmin` and `R2Rmin`:

```
Rmin = BT2Rmin(BT, q); check(equal(BT, R2BT(Rmin, q, T), q))
Rmin = R2Rmin(R, q);   check(equal(BT, R2BT(Rmin, q, T), q))
```

The dichotomy of parametric vs nonparametric representations is misleading—there is a range of nonminimal parametric representations that cover the gap between minimal parametric and nonparametric representations. The problem of detecting when a parametric representation is minimal is equivalent to the one of finding the model's complexity. It is also essential in the case of identification from noisy data where the key issue is the one of achieving an accuracy–complexity trade-off.

All functions of the toolbox computing model's complexity accept as an optional input argument a tolerance for thresholding the singular values. Alternatively, the user may specify the model's complexity `[m, ell, n]` by `ctol`. If `ctol` is a scalar, it is used as a threshold `tol` for rank estimation. Otherwise, `ctol` should be the vector of integers specifying the model's complexity.

Signal processing and open-loop control

This section collects operations on signals by a dynamical system. The operations implemented are: projection of a signal on a system, computing initial conditions for a signal, simulation, and inference of one variable from another. They are special cases of a signal processing problem, called *interpolation and approximation of trajectories*.

Projection

The basic operation of projection of a signal $w \in (\mathbb{R}^q)^T$ on the behavior $\mathscr{B}|_T$ of a system $\mathscr{B} \in \mathscr{L}^q$ is equivalent to computing the *distance from w to $\mathscr{B}$*. The same operation is equivalent to (errors-in-variables) Kalman smoothing: the projection $\widehat{w}$ is the *smoothed version* of w. The problem is solved by the function `dist`:

```
[d, wh] = dist(w, B);
```

The distance `d` from w to $\mathscr{B}|_T$ is zero if and only if $w \in \mathscr{B}|_T$, so that `dist` is a robust way of checking if w is an exact trajectory of $\mathscr{B}$. (It is used in the implementation of `w_in_B`.)

Initial conditions estimation

For $w_\text{d} \in \mathscr{B}|_T$, there is a corresponding initial conditions. If $\mathscr{B}$ is defined by an input/state/output representation, the initial conditions can be specified by the initial state $x_\text{ini} = x(1)$. The initial state x_ini corresponding to a trajectory w can be found with the function `w2xini`:

```
[xini, e] = w2xini(w, B); check(norm(w - B2w(B, T, xini, w(:, 1:m))) < tol)
```

The output argument `e` is zero when $w \in \mathscr{B}|_T$ and nonzero otherwise.

The initial conditions can be specified in a representation free way by an initial trajectory w_ini of length $T_\text{ini} \geq \ell(\mathscr{B})$. An initial trajectory w_ini of corresponding to a given trajectory w can be computed by the function `w2wini`:

```
BT = B2BT(B, ell + size(w,1));
[wini, e] = w2wini(w, BT, ell); check(w_in_B([wini; w], B))
```

Simulation

The classical simulation problem: given initial conditions and input, find the corresponding output is solved by `u2y`:

```
yf = u2y(BT, q, w(:, 1:m), wini); check(w(:, m+1:end) - yf)
```

As another example, next, we use `u2y` in order to find the first `Tf` samples of the impulse response of the system:

```
Tf = 3; u1 = [[1; zeros(q * Tf - 1, 1)] zeros(q * Tf, m - 1)];
h1 = u2y(BT, q, u1); h = lsim(B, u1); check(h1 - h < tol)
```

Observer

Consider a linear time-invariant system $\mathscr{B} \in \mathscr{L}^q$. The variables w of $\mathscr{B}$ are partitioned into given/observed variables w_g with dimension q_g and missing/to-be-estimated variables w_m with dimension $q_\mathrm{m} = q - q_\mathrm{g}$. Without loss of generality, we assume that $w = \left[\begin{smallmatrix} w_\mathrm{g} \\ w_m \end{smallmatrix}\right]$. The problem considered is: Given a system $\mathscr{B} \in \mathscr{L}^q$ and observed part w_g of a trajectory $w \in \mathscr{B}|_T$, find the missing part w_m of w.

The solution is based on the finite-horizon nonparameteric representation of $\mathscr{B}$. Let B_g be the submatrix of B_T corresponding to w_g and B_m be the submatrix corresponding to w_m. Then, there is a g, such that

$$w_\mathrm{g} = B_\mathrm{g} g \quad \text{and} \quad w_\mathrm{m} = B_\mathrm{m} g.$$

Solving for g the first equation and substituting into the second one, we have the following estimate of w_m:

$$\widehat{w}_\mathrm{m} := B_\mathrm{m} B_\mathrm{g}^+ w_\mathrm{g}.$$

It can be shown that, when $\operatorname{rank} B_\mathrm{g} = \mathbf{m}(\mathscr{B})T + \mathbf{n}(\mathscr{B})$, $\widehat{w}_\mathrm{m}$ is exact, i.e., of $\widehat{w}_\mathrm{m} = w_\mathrm{m}$.

Special cases of the observer problem are estimation of the input given the output (which is also a system inversion problem) and estimation of a second input, e.g., disturbance, given the input and the output.

The solution of the observer problem is implemented in the function `wgiven2wmissing`. Here is an example verifying the method for disturbance estimation:

```
qg = 2; qm = 2; q = qm + qg; m = 1; p = q - m; n = 5; B = drss(n, p, m);
T = 20; w = B2w(B, T); wg = w(:, 1:qg); wm = w(:, qg+1:q);
wmh = wgiven2wmissing(wg, B); check(norm(wm - wmh) / norm(wm) < tol)
```

Open-loop control

The control problem considered is open-loop linear quadratic tracking: Given a system $\mathscr{B} \in \mathscr{L}^q$, to-be-tracked trajectory $w_\mathrm{r} \in (\mathbb{R}^q)^{T_\mathrm{r}}$, and tracking criterion $\|e\|_v := \|v \otimes e\|$, where $v \in (\mathbb{R}_+^q)^{T_\mathrm{r}}$ and $\otimes$ is the element product,

$$\text{minimize} \quad \text{over } \widehat{w} \in (\mathbb{R}^q)^{T_\mathrm{r}} \quad \text{norm} w_\mathrm{r} - \widehat{w}_v \quad \text{subject to} \quad \widehat{w} \in \mathscr{B}|_{T_\mathrm{r}}.$$

A variation of the problem is to specify initial and final conditions for the control trajectory $\widehat{w}$ via "past" and "future" trajectories $w_\mathrm{p} \in (\mathbb{R}^q)^{T_\mathrm{p}}$ and $w_\mathrm{f} \in (\mathbb{R}^q)^{T_\mathrm{f}}$:

$$\text{minimize} \quad \text{over } \widehat{w} \in (\mathbb{R}^q)^{T_\mathrm{r}} \quad \text{norm} w_\mathrm{r} - \widehat{w}_v \quad \text{subject to} \quad w_\mathrm{p} \wedge \widehat{w} \wedge w_\mathrm{f} \in \mathscr{B}|_{T_\mathrm{p}+T_\mathrm{r}+T_\mathrm{f}}.$$

This allows us to solve optimal state transfer problems, i.e., find a trajectory of the system that achieves optimal transition from the given initial state to the given final state. The solution exists for any initial and final states if and only if the system $\mathscr{B}$ is controllable and the state transfer is at least $\ell(\mathscr{B})$-samples long.

The function in the toolbox solving the open-loop linear quadratic tracking control problem is

```
wh = lqctr(B, wr, v, wp, wf);
```

The input parameters `v`, `wp`, and `wf` are optional. The default value of `v` is `ones(Tr, q)`, i.e., uniform weights for all variables and all moments of time. If `wp` and/or `wf` is not specified, then the corresponding constraints are dropped, i.e., the corresponding initial or final condition is free.

Here is an example of linear quadratic tracking with free initial and final condition:

```
m = 1; p = 1; n = 3; q = m + p; B = drss(n, p, m);
Tr = 10; wr = [zeros(Tr, m) ones(Tr, p)]; ell = lag(B);
v = [1e-2 * ones(Tr, m), ones(Tr, p)]; wh = lqctr(B, wr, v);
```

In order to solve the optimal state transfer problem with specified initial and final states, first we check controllability of the system. The solution is verified by showing that $w_\mathrm{p} w_\mathrm{d} edge \widehat{w} w_\mathrm{d} edge w_\mathrm{f} \in \mathscr{B}|_{T_\mathrm{p}+T_\mathrm{r}+T_\mathrm{f}}$.

```
if ~isunctr(B)
  Tr = ell; wr = zeros(Tr, q);
  wp = B2w(B, ell); wf = zeros(ell, q);
  wh = lqctr(B, wr, [], wp, wf);
  check(w_in_B([wp; wh; wf], B))
end
```

Control as interconnection

In the following example the plant $\mathscr{B}$ is a 4th order single-input single-output system, defined by the transfer function

$$H(z) = \frac{0.2826z + 0.5067z^2}{1 - 1.4183z + 1.5894z^2 - 1.3161z^3 + 0.8864z^4}.$$

```
Q = [0 0 0 0.28261 0.50666]; P = [1 -1.41833 1.58939 -1.31608 0.88642];
B = ss(tf(Q, P, -1));
```

The controller $\mathscr{B}_{\mathrm{c}}$ is obtained with the function `h2syn` from the Robust Control Toolbox of Matlab

```
Bc = h2syn(B, 1, 1);
```

It leads to the closed-loop system $\mathscr{B}_{\mathrm{cl}}$

```
B_ = ss(B.a, B.b, [B.c; B.c], [B.d; B.d], 1); Bcl = lft(B_, Bc);
```

The closed-loop system $\mathscr{B}_{\mathrm{cl}}$ can be obtained alternatively by projecting on the y-variable the interconnection of the plant $\mathscr{B}$ and the control $\mathscr{B}_{\mathrm{c}}$ with flipped inputs and outputs:

$$\mathscr{B}_{\mathrm{cl}} = \Pi_y\left(\mathscr{B} \cap \left[\begin{smallmatrix}0 & 1\\ 1 & 0\end{smallmatrix}\right] \mathscr{B}_{\mathrm{c}}\right).$$

```
T = 10; BT = B2BT(B, T); BcT = B2BT(Bc, T);
Bcl_wT = BTintersect(BT, BT2BT(BcT, q, [2 1]));
[Bcl_uT, Bcl_yT] = BT2UYT(Bcl_wT, 1, 1);
equal(Bcl, Bcl_yT, q) % -> 1
```

Identification

The functions presented in the previous sections of the tutorial use the nonparameteric representation of a linear time-invariant system $\mathscr{B} \in \mathscr{L}_c^q$. In this section, the system $\mathscr{B}$ is implicitly specified by a trajectory $w_{\mathrm{d}} \in \mathscr{B}|_{T_{textd}}$

```
Td = 100; wd = B2w(B, Td);
```

A necessary and sufficient condition for w_{d} to define $\mathscr{B} \in \mathscr{L}_c^q$ is the *identifiability condition*: \] rank \, H$_{\ell(\mathscr{B})+1(\{w\{d\}\}) = m(B)\left(\ell(\mathscr{B}\right.}$