The Behavioral Toolbox

Ivan Markovsky

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. In order to install it, download and unpack the archive file. A tutorial is given in Section 3 of the documentation. In order to cite it, please use:

> I. Markovsky. The behavioral toolbox. In Proc. of Machine Learning Research, volume 242, pages 130–141, 2024.

```
@InProceedings{bt-l4dc,
  Author =    {I. Markovsky},
  Title =     {The Behavioral Toolbox},
  booktitle = {Proc. of Machine Learning Research},
  volume =    {242},
  pages =     {130--141},
  Year =      {2024}
}
```

Examples

In the examples shown below, a trajectory of a discrete-time linear time-invariant system — *the data* — is given and the goal is to find some property of the data-generating system or solve a problem with the data-generating. It is assumed that the data fully specifies the system (see Section 3.7 in the documentation).

Finding the number of inputs

The data is a vector time series containing the observed variables of the system. Since these variables are not partitioned into inputs and outputs, which variables are inputs is not known. Also, how many variables are inputs is not known. It is possible however to fining the number of inputs from the given data. The function of the toolbox that does this is `w2c`. In addition to finding the number of inputs m, it finds also the order n and the lag ℓ of the underlying data-generating system.

Here is an example:

- specify the simulation parameters—number of variables `q`, number of inputs `m`, lag `ell`, order `n`, and number of data samples `Td`

```
q = 3; m = 1; ell = 2; n = 3; Td = 20;
```

- generate a random linear time-invariant system with the specified parameters

```
B = randB(q, [m, ell, n]);
```

- generate a random trajectory of the system with the specified number of samples

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

- call the `w2c` function

```
[~, mh, ellh, nh] = w2c(wd);
```

- check if the computed result is correct

```
[m == mh, ell == ellh, n == nh] % -> [true true true]
```

But what about the assumption? A "sufficiently long" random trajectory almost surely fully specifies the system. Quantifying what "sufficiently long" means, however, requires knowledge of m and n. There is no way out of the dilemma, either we have to assume that the data fully specifies the system, or we have to know m and n, in which case we can check the condition.

Exercise: Change the simulation parameters and find out empirically when the data is "sufficiently long".

Finding an input/output partitioning of the variables

We've seen that we can find the number of inputs m from the data. Can we find also *the* input variables u? No, in general, we can not do this because different sets of variables can act as inputs. The function of the toolbox that finds all possibilities of partitioning the variables into inputs and outputs is `BT2IO`.

`BT2IO` (as well as other functions of the toolbox) takes as an input argument a matrix B_T that can be obtained from the data using `w2BT`. B_T defines a non-parametric representation of the finite-horizon behavior $\mathscr{B}|_T$. The horizon T is a problem dependent. For `BT2IO` the required horizon is $T = 2\ell + 1$.

Continuing the example above,

- first, we obtain the matrix B_T:

```
T = 2 * ell + 1; BT = w2BT(wd, T, [m ell n]);
```

- then, we call `BT2IO`:

```
IO = BT2IO(BT, q)
```

Its output

```
IO =

     1     2     3
     1     3     2
     2     1     3
     2     3     1
     3     1     2
     3     2     1
```

is a matrix, the rows of which indicate the possible input/output partitionings, e.g.,

```
ud = wd(:, IO(1, 1:m)); yd = ud = wd(:, IO(:, m+1:end));
```

is the first input/output partitioning.

Exercise: In the example above, all partitionings of the variables are valid input/output partitionings. Create an example where this is not the case.

The data-generating system may be unstable (but it's OK)

In the examples above, we've used the function `randB` for obtaining a random data-generating system `B` and the function `B2w` for obtaining a random trajectory of that system.

- `randB` constructs a kernel representation `R` corresponding to the specified structure (number of variables, number of inputs, lag, and order) with nonzero elements generated as random uniformly distributed in the interval $[0, 1]$. The kernel representation `R` is then converted to an input/state/output representation `B`.

- `B2w` uses the function `lsim` from the Control System Toolbox of Matlab in order to simulate a trajectory of `B` under random initial conditions and a random input.

The system produced by `randB` may be unstable:

```
[B, R] = randB(q, [m, ell, n]); any(abs(eig(B)) > 1) % -> true
```

When `B` is unstable, the trajectory obtained from random input and initial conditions is almost surely exponentially diverging

```
T = 300; w = B2w(B, T); max(w(:)) % -> large
```

Generating a "sufficiently long well behaved" random trajectory from unstable system is a problem.

An alternative way to generate a random trajectory of the system is by using the nonparameteric representation B_T of the finite-horizon behavior: $w = B_T g$, where g is a random vector

```
BT = B2BT(B, T); w = BT * rand(size(BT, 2), 1); max(w) % -> small
```

Since by construction the matrix B_T is orthonormal and $||g||$ is bounded, the resulting trajectory w is also bounded.

Distance of a signal to a system

The distance of a signal w to a system $\mathscr{B}$ has the familiar definition and geometrical interpretation: $\min_{\widehat{w}\in\mathscr{B}} \|w-\widehat{w}\|$. It's computation is implemented in the function `dist`. As a specification of $\mathscr{B}$, `dist` accepts an `lti` object as well as the finite-horizon representation B_T with horizon T the length of w.

Finding the distance of a signal to a system is useful in particular for checking if the signal is a valid trajectory of the system:

```
dist(wd, B) % -> 0
```

The function `dist` returns also the projection $\widehat{w}$ of w in $\mathscr{B}$. The signal $\widehat{w}$ is useful because it the best approximation of w by a trajectory of the system. It is called the smoothed signal. Here is an example.

- We add a perturbation on the trajectory and compute its distance to the system:

```
wd_noisy = wd + randn(size(wd)); [d, wdh] = dist(wd_noisy, B);
```

- The distance of the best approximation to the original trajectory is smaller than the distance of the perturbed trajectory to the original trajectory:

```
[norm(wd - wdh) norm(wd - wd_noisy)] / norm(wd)
```

Distance between systems

How to check if two systems are equal? By construction the system

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

is equal to `B`. However, the Control Systems Toolbox of Matlab does not allow us to infer it by

```
B == Bp % -> Operator '==' is not supported for operands of type 'ss'.
```

Instead, we can check the norm of difference:

```
norm(Bp - B) % -> 0
```

The norm computation, however, is restricted to stable systems.

The function `Bdist` computes a distance between systems defined as the angle between the corresponding behaviors:

```
Bdist(Bp, B) % -> 0
```

This distance measure is applicable also to stable systems.