# 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 of the functions. A tutorial for its usage 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 $\ell$ 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 = 100;
```

- 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] % -> [1 1 1]
```

*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.

## Distance of a signal to a system

A useful operation is computing distance between data and system. In particular, it allows us to check if a given signal is a valid trajectory of the system. In case it is not, the distance computation returns the best approximation of this signal by a trajectory of the system. This operation is called smoothing.

The distance computation is implemented in the function `dist`. As a specification of the system, it accepts both `lti` object as well as ($B_T$), in which case the horizon $T$ should be at least the length of the signal.

Continuing the example above, first, we generate a random trajectory and compute its distance to the system:

```
T = 100; w0 = B2w(B, T); dist(w0, B) % -> 0
```

Then, we add a perturbation and compute its distance to the system:

```
w = w0 + randn(size(w0)); [d, wh] = dist(w, 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(w0 - wh) norm(w0 - w)]
```

### Distance between systems

How to check if two systems are equal. For example, by construction the system

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

is equal to `B`, however, the Control Toolbox of matlab does not allow us to infer this by

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

Instead, we can check the norm of difference:

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

This, however, is restricted to stable systems only.

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

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

This distance measure is not restricted to stable systems.

### Distance of a signal to being a trajectory of a bounded complexity system

In the previous sections we considered distance of a signal to a system and distance between systems. In this section, we consider another distance problem: the distance of a given signal to a system in a class of linear time-invariant systems with bounded complexity. This is actually a model identification problem. Zero distance means that there is an exact model of bounded complexity. Nonzero distance delivers as a by-product a system with bounded complexity that certifies the reported distance. Since the problem is non-convex there is no grantee that the obtained distance is globally minimal.