Data-driven learning of dynamical systems

Hugues Garnier

January 2025

Contents

Tutorials 1

Identification of linear transfer function models with the CONTSID toolbox

Some general comments

- For this tutorial session, you will mainly use the CONTinuous-Time System IDentification (CONTSID) toolbox which is a collection of m-files written in Matlab.
- The toolbox can be run in command-line mode or in a graphical user interface (GUI) mode or in any combination of these. It is however recommended to run/solve the exercices with the command-line mode.
- The exercices are intended for the CONTSID toolbox 7.5 to be run with Matlab version R2024a equipped with the System Identification and the Control toolboxes.

Downloading of the data needed for the tutorial

- Download the zipped file **Tutorial1_SYSID.zip** from the course website and save it in your Matlab working directory.
- Start Matlab.
- Change the current folder of Matlab so that it becomes your Tutorial1_SYSID folder that contains the files needed for this lab.

Exercise 1.1 - Tutorial introduction to the CONTSID toolbox
This first exercise is meant as an introduction to computer-based system identification with the CONTSID toolbox. You will run an identification demonstration program available in the toolbox. The chosen demonstration includes several steps that will be discussed thoroughly later on in the other exercices, so do not expect to understand all steps taken in the demonstration.

Start Matlab and run the CONTSID main demonstration program through the command:

```
>> contsid_demo
```

Select *Tutorials* in the menu window.
In the new menu window, select *Getting started* and follow the demo displayed in the command window.
When the demo is finished, select *Quit* to come back to the main menu window.

Run one or more other demos to get a feel for the different options available in the CONTSID toolbox.

Exercise 1.2 - Parameter estimation of a simulated transfer function model
The basic problem in data-driven learning for dynamical systems is to find a mathematical model from input/output measurements. Of course, in practice the true system is unknown (otherwise there would be no need for system identification) and only the measurements are available.

Data-generating system

We will consider estimation from data given by the following system:

$$\mathcal{S}\begin{cases} x(t) = \dfrac{B_o(s)}{F_o(s)}u(t-\tau) \\ y(t_k) = x(t_k) + e(t_k) \end{cases} \tag{1.1}$$

where s represents here the differentiation operator $s = \frac{d}{dt}$ which is the notation used in the Matlab System Identification and CONTSID toolboxes.
τ is the time-delay (in sec), u is the system input, x is the noise-free output and y the noisy output.
The disturbance $e(t_k)$ is a discrete-time white Gaussian noise, independent of the input u, with zero mean and variance σ_e^2.
The polynomials $B_o(s)$ and $F_o(s)$ are defined by:

$$\begin{cases} B_o(s) = 3 \\ F_o(s) = s^2 + 2s + 3 \end{cases} \tag{1.2}$$

1. Determine the main features of the true system: steady-state gain, time-constants and time-delay (in s)

For different inputs, we simulate the system responses contaminated by measurement noise with σ_e.

We assume that the "correct" model order is known and will try to estimate the parameters from the different sets of data by the different direct continuous-time SRIVC-based optimization methods.
Note that the input `u` is a deterministic signal. The noise signal `e` on the other hand is stochastic, so each time you run the commands above, you will obtain a different realization of the noise signal and hence also of the output. Your model estimate should therefore slightly differ for each noise realization every time you execute the code.

Polynomial model learning by the SRIVC routine

We first assume a COE polynomial model structure for the system:

$$\mathcal{M}_{\text{COE}} : y(t_k) = \frac{B(s)}{F(s)}u(t_k - \tau) + e(t_k) \tag{1.3}$$

The model polynomials $B(s)$ and $F(s)$ are given by:

$$\begin{cases} B(s) = b_0 \\ F(s) = s^2 + f_1 s + f_2 \end{cases} \tag{1.4}$$

while the time-delay τ is assumed to be a multiple integer of the sampling time: $\tau = nk * T_s$.

The CONTSID `srivc` routine can be used estimates the parameters of a COE polynomial model by the SRIVC.

```
Gsrivc=srivcf(data,[nb nf nk]);
```

where `nb` et `nf` represent the number of parameters to be estimated for the $B(s)$ and $F(s)$ polynomials respectively and `nk` denotes the number of samples for the time-delay.

1. Type the following commands:

   ```
   Gsrivc=srivc(data,[1 2 0]);
   present(Gsrivc)
   ```

   The estimated parameters can be obtained from the model objet from the following command:

   ```
   Gsrivc.param’
   ```

Process model learning by the PROCSRIVC routine

We now consider the identification of a low-order process model by the PROCSRIVC.
In industrial practice, identification for control really is the determination of simple process models of order limited at most to 3, adjusted from transient or possibly relay experiments, followed by the tuning of PI(D)-regulator based on the identified simple process model parameters. The family of simple process models is characterized by steady-state gain, dominating

time-constants and possible time-delay and pure-integrator. Examples of this type of model structures includes the following second-order model of the form:

$$\mathcal{M}_{\text{procsrivc}} : y(t_k) = \frac{K}{\frac{s^2}{\omega_o^2} + 2\frac{z}{\omega_o}s + 1} e^{-\tau s} u(t_k) + e(t_k) \tag{1.5}$$

where τ is the time-delay and the $e(t_k)$ is assumed to be white.
The CONTSID `procsrivc` routine can be used as follows:

```
type = idproc("P2");
Gprocsrivc=procsrivc(data,type);
```

where `type` defines the type of low-order process model (of order 2 without any delay (D) or pure Integrator (I).
The estimated model parameters along with their standard deviations can be displayed by using the following command:

```
present(Gprocsrivc);
```

Transfer function model learning by the TFSRIVC routine

We now consider the identification of a transfer function model of any order by the TFSRIVC routine.

$$\mathcal{M}_{\text{COE}} : y(t_k) = \frac{B(s)}{F(s)} e^{-\tau s} u(t_k) + e(t_k) \tag{1.6}$$

where τ is the time-delay and the $e(t_k)$ is assumed to be white.
The CONTSID `tfsrivc` routine can be used as follows:

```
Gtfsrivc=tfsrivc(data,np,nz);
```

where `np` et `nz` represent the number of poles and zeros of the transfer function to be estimated.
The estimated model parameters along with their standard deviations can be displayed by using the following command:

```
present(Gtfsrivc);
```

Type the following commands:

```
IODelay=0;
 Gtfsrivc=tfsrivc(data,2,0,'TdMax',IODelay);
 present(Gtfsrivc);
```

A time-delay can also be estimated along with the transfer function coefficients but this is not exploited here. To do so, the upper time-delay boundary 'TdMax' is set to 0.
Open the file `Tutorial1.mlx` in the live editor and execute the different sections that will :

1. generate different excitation signals: square wave, prbs, sine chirp, square chirp.
2. generate input/data data for a given signal-to-noise ratio.
3. use the SRIVC, PROCSRIVC, TFSRIVC routines to estimate the parameters of polynomial model, low-order process model and transfer function model respectively.

4. evaluate the quality of the estimated model by basic validation test comparing the .

Duplicate the file and modify the true system so that it becomes a first-order plus pure-integrator model.

$$\mathcal{S} : y(t_k) = \frac{K}{s(Ts+1)} e^{-\tau s} u(t_k) + e(t_k) \tag{1.7}$$

Exercise 1.3 - Real-life system data - The Blue Robotics T200 Thruster

The T200 Thruster from Blue Robotics, shown in Figure 1.1, is one of the world's most popular underwater thruster for ROVs, AUVs or surface vessels. Its flooded motor design makes it compact but efficient and powerful.

Figure 1.1: The T200 Thruster from Blue Robotics.

The T200 thruster is in use on thousands of marine robotic vehicles including the BlueROV2 as shown in Figure 1.2.

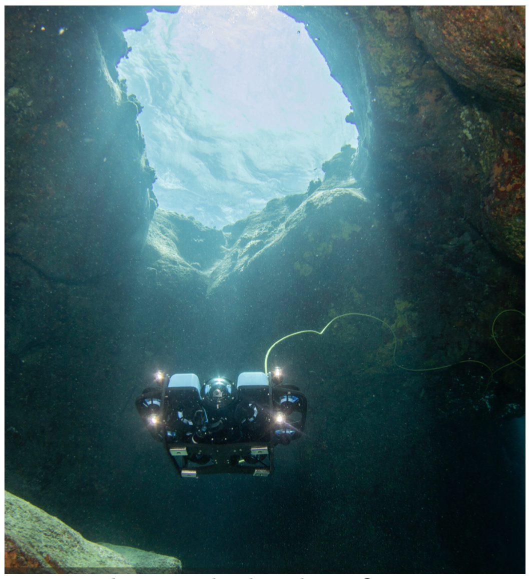

Figure 1.2: The marine robotic vehicle BlueROV2 - `www.bluerobotics.com`.

1. Run the file `import_thruster_data.mlx` in Matlab that loads dataset[1] from two different inputs (sine and square chirp signals) applied to a Blue Robotics T200 Thruster. You can watch a short video showing one of the two experimental tests at:

[1]MathWorks Student Competitions Team (2022). MATLAB and Simulink Robotics Arena : From Data to Model (https://www.mathworks.com/matlabcentral/fileexchange/65919-matlab-and-simulink-robotics-arena-from-data-to-model), MATLAB Central File Exchange

`www.youtube.com/watch?v=RU_L0trE0Bo` (go to the 17mn25). The input is the motor controller command, which is the duty cycle of a PWM signal given in microseconds. The whole range is [1100, 1900], where 1500 is neutral (zero thrust), 1900 is max. thrust, and 1100 is max. reverse thrust. The output is the generated thrust force.

2. Apply the data-driven learning methodology to estimate and validate a transfer function model of the T200 thruster.
3. Pay attention, in particular, to the filtering stage to be applied to the raw measured output. Use the SignalAnalyzer App to display the spectrum and apply a low-pass filter to the noisy output. Use the filtered output to fit a continuous-time model.

Note that for this practical case, there is thus "no correct solution (no true model)" to it, but you might use some physical insight to estimate/validate your model.

Tutorials 2

Identification of linear continuous-time canonical state-space models

Exercise 2.1 - State-space model identification and LQR control for a SIMO flexible link
The rotary flexible link, shown in Figure 2.1, is a two-degree of freedom system with an actuated rotary servo and a flexible link/arm. This emulates many applications such as lightweight robot manipulators (e.g., in space applications) and cantilever beams.

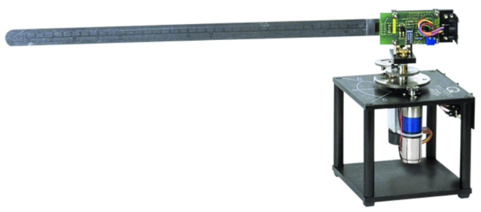

Figure 2.1: The rotary flexible link

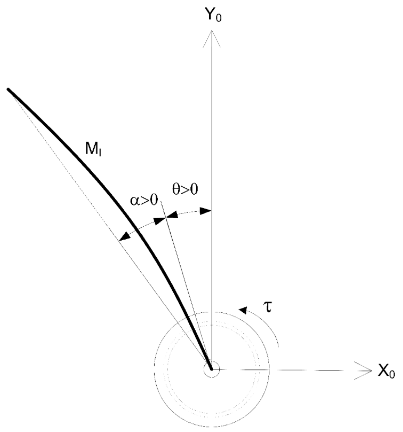

Figure 2.2: Schematic of the rotary flexible link

The rotary flexible link is a one input, two output systems. It is therefore a SIMO system where, with reference to Figure 2.2:

- $\tau(t)$: the torque proportional to the voltage $u(t)$ sent to the DC motor;
- $\theta(t)$: the angular position of the servo base;
- $\alpha(t)$: the angle defection of the flexible link.

The rotary servo angle $\theta(t)$ is measured using an incremental encoder while the angle deflection (relative to the servo base) $\alpha(t)$ of the flexible link is measured using a strain gage.

1. Open and run the file `Flexlink_SYSID_and_LQR_control.mlx`
2. Examine the file and make sure you understand its contents. Note that the model order selection and model validation are incomplete in the given .mlx file (more tests are usually carried out but they are not presented in this file).
3. The identified model is then used to design a LQR control. Examine the part of the .mlx file dedicated to this control design and make sure you understand it.

Exercise 2.2 - State-space model identification and LQR control for a SIMO rotary tower crane

The motions of a rotary pendulum, shown in Figure 2.3, are similar to that of a tower crane. The rotary arm emulates the horizontal jib, the slewing unit is the DC motor, and the pendulum represents the cable carrying a payload.

The goal of the control is to minimize the deflection of the pendulum, i.e. the crane carrying a load, as the rotary arm, the jib, rotates to a desired angle.

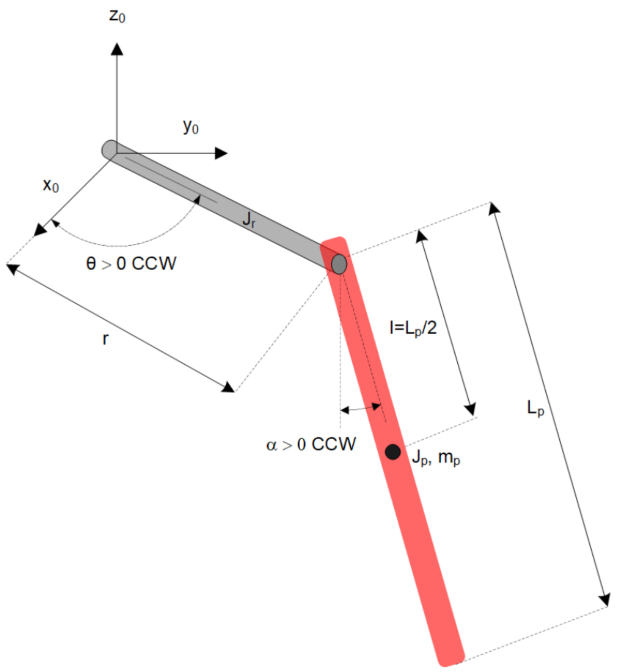

Figure 2.3: Rotary pendulum motions similar to tower crane.

1. Run the file `import_tower_crane_data.mlx` in Matlab that loads the data from a square or simple sine wave test applied to the tower crane system.
2. Choose one dataset as the estimation or training dataset and the other as the validation dataset.
3. Use the data to determine a continuous-time linear state-space model of the tower crane system.

4. Based on the initial analysis of the dataset available, suggest a new experiment design (a totally new excitation input). Carry out your proposed experiment to record the new dataset to be exploited for further identification. USe the file `Tower_crane_OL_your_test.slx`

5. Design and implement a LQR control to move the base arm from an initial position to a final angle with a minimum of pendulum oscillation. Test your LQR control by using the slx file provided.