Exercise Session 5: A Practical System Identification Problem

January 2024 examination session
version 2

In this exercise session, you will carry out a practical identification problem by collecting input-output data from an unknown system. You will need to

- design input-signals,
- do an experiment (software),
- pre-process the data,
- identify different models,
- validate the models.

1 Generating the output

When you have designed an input u, e.g.

```
>> u = randn(1000,1);
```

then you can generate the output y by typing

```
>> y = course_system_identification('black_box',u);
```

The output of your system will be returned in the vector `y`.

2 Questions

The question for this exercise is simple. The answer is harder. The basic question is: try to generate a 'good' model for the black box system. Everything is left to your creativity and inventiveness. You can use everything from the course slides and previous exercise sessions and much more if you want to try some things out.
As stated before, the basic steps are

1. Input design: step, white noise, colored noise, random binary, impulse, sine waves.
   Important: The input will saturate at a certain level (just like in reality the actuators will saturate). The saturation level is at 3 and -3. So, if you put an input in the simulator that has an amplitude bigger than 3, it will be clipped to 3. No warning will be given (this is your own responsibility).

2. Preprocessing:

    - missing values: Replace missing values (zeros) by using linear interpolation. You can easily program this yourself or use the MATLAB command `interp1` (see https://nl.mathworks.com/help/matlab/ref/interp1.html).
    - trend: If there is a trend in your output signals, you can use `detrend` and/or `filtfilt` to remove it.
    - delay correction: try `cra` or `delayest` or the following three commands in combination: `struc`, `arxstruc`, `selstruc`. More information: https://nl.mathworks.com/help/ident/ug/preliminary-step-estimating-model-orders-and-input-delays.html

3. Identification.

4. Validation.

You are allowed to do as many experiments as you want on the system (using the command `course_system_identification`). Nevertheless, keep in mind that in industry, experiments are costly. So, by taking such limitations into account when designing an experiment, you can think/argue about practical benefits and/or downsides.
Note this is an open question. You are encouraged to apply the knowledge you gathered from the previous exercise sessions and the lectures as much as possible to this question.

Hints:

- The function[1] `cra` allows you to get an idea of the time delay in the system, in the same way as described in the course slides of Chapter 5, see also https://nl.mathworks.com/help/ident/ref/cra.html.
  The `cra` function will compute an estimate of the impulse response and plot this estimate together with the confidence bounds.

- To simulate real-world data and temporary sensor malfunctions, the signal contains missing (zero) values. Interpolation can help in filling the gaps with estimates, but it may introduce artifacts too. Simple versions like linear interpolation usually offer good enough approximation accuracy, calculation efficiency and very few artifacts.

- You could estimate a reference transfer function (see Exercise 2 of Session 3)

    - by making an empirical Bode diagram:
        * Apply a sine wave input at a certain frequency.
        * Measure and pre-process the output. Check (through an `fft` for instance) if the output still has an energy peak at the frequency used for the input (if not, the signal to noise ratio is too bad).
    - by using `spa` or `etfe`

- You can get an idea of the noise spectrum by applying a zero input to the system. The output then measured is purely due to the disturbances.

- You can try different model classes and the corresponding system identification functions: `arx`, `oe`, `bj`, `armax`, `arma`, ...

[1] You need to be able to explain how the MATLAB functions that you have applied, work. Make sure you understand what is happening behind the scenes.

Evaluation 2.1:
Your report should contain (at least) the following sections:

- input design,
- preprocessing,
- identification,
- validation,
- conclusion (including choice of best models).

A Appendix: The System Identification Toolbox

You will be using a collection of 'identification functions' in MATLAB. This collection is called the *System Identification Toolbox*. The toolbox comes with a graphical user interface, which can be called from the MATLAB prompt by

```
systemIdentification
```

You are free to use and experiment with the graphical user interface, however, we will use the command line instructions (functions) from the toolbox instead. The System Identification Toolbox manual describes all functions in this toolbox.
It is important to note that the toolbox uses models in the so-called `idpoly` format. As long as you keep on using functions of the toolbox, you can use this format. To extract the coefficients of a model, `sys`, you use the commands

```
sys.A
sys.B
sys.C
sys.IODelay
```

where in this case we've extracted the coefficients of $A(z), B(z), C(z)$ and the input-output delay n_k. You can find out what other properties you can obtain from a model by typing `properties(sys)`. Or, to see the properties and their values: `get(sys)`. Do not forget to consult the help files to brush up on conventions!

A.1 Getting Acquainted with the System Identification Toolbox

These exercises are intended to make you familiar with the MATLAB System Identification Toolbox and with the identification theory of the course.

Evaluation A.1:
You do not have to write a report for the exercises in this and the following sections.

Before starting, it is advised to have a look at the documentation of the toolbox and its functions and data structures. For extra hands-on practice opportunities, type `demos` in MATLAB's command window, then choose the System Identification Toolbox.
To load your system, type at the MATLAB prompt:

```
demo_sys = course_system_identification('demo_system')
syspoly = demo_sys('syspoly')
```

This will load a system of your choice (pick a number between 1-30) into the MATLAB environment, represented by `syspoly`, which is an `idpoly` object. The system is represented as a discrete-time polynomial model with identifiable coefficients

$$A(z)y_k = \frac{B(z)}{F(z)}u_k + \frac{C(z)}{D(z)}e_k \ .$$

The coefficients in the `idpoly` are arranged in descending degrees of z^{-1} (e.g., $A(z) = 1 + a_1 z^{-1} + a_2 z^{-2}$ is represented by `A = [1 a1 a2]`). More information about the model at hand can be obtained using typing `syspoly`, `present(syspoly)` and `get(syspoly)`. While for models directly constructed from coefficients the information is minimal, it is more abundant for estimated models (e.g. variance on coefficients).

A.2 Delay estimation

First, determine the delay in the system. This can easily be done by looking at the impulse response. You can plot it directly using the model or by simulation and using input-output data. Some useful commands are

```
impulse(syspoly)
u = iddata([],randn(1000,1),1);
y = sim(syspoly,u);
cra([y,u])
```

Plot the impulse response. What is the delay of the system? In this case, the system is already known, so compare with the system characteristics.
Try adding some noise. You can accomplish this by setting the noise variance property of the system as follows

```
set(syspoly,'NoiseVariance',1e-2);
u = iddata([],randn(1000,1),1);
opt = simOptions('AddNoise',true,'NoiseData',randn(1000,1));
y = sim(syspoly,u,opt);
cra([y,u])
```

What are your observations? Try playing with the variance. Also keep in mind that the noise variance field in your system changes that of your noise input relative to the variance of the supplied noise. If you were to supply a noise sequence in `simOptions` with a variance not equal to 1, it multiplies with the noise variance of the system. Try it out.

A.3 Transfer function

The $C(z)$ and $D(z)$ polynomials are only used to model the noise. The deterministic transfer function is found by looking at the coefficients of $A(z)$, $B(z)$ and $F(z)$ (in this case $F(z)$ equals 1). For example, using one of the examples:

```
>> tf(syspoly)

ans =

    0.0804 z^-1 - 0.05523 z^-2 + 0.001418 z^-3
 ------------------------------------------------------
1 - 0.7983 z^-1 + 1.127 z^-2 - 0.7565 z^-3 + 0.3388 z^-4
```

- Plot the transfer function using the `bode` function. Note that we only work with discrete time systems and assume a sampling time of 1 s. The maximum frequency in the Bode plot is therefore equal to π rad/s.
- Look at the poles and zeros of the system using `pzmap`. How do they correspond to the roots of the $A(z)$ and $B(z)$ polynomials? Starting from the poles and zeros, explain the shape of the transfer function and impulse response. Use

  ```
  [p,z] = pzmap(syspoly)
  ```

  to get numerical values of the poles and zeros.[2]

[2]This is generally true for plotting functions in the System Identification Toolbox. Calling `bode(sys)` will give you a plot, while assigning it to a set of variables will give you the data shown in the plot. See `help` for more information.

A.4 Estimating an ARX model

As a first exercise (to become acquainted with the toolbox) we will use the function `arx` to estimate ARX models. Generate a white noise input u_k ($\sigma = 1, N = 500$) using the function `randn`. Now apply this input to the system to find the output y_k. In order to simulate the system, you must wrap your input in an `iddata` object and use

```
u = iddata([],randn(500,1),1);
y = sim(syspoly,u);
```

From this input-output pair, you can identify the system using the function `arx` and the exact model structure, which can be retrieved from the system itself:

```
na = get(syspoly,'na')
nb = get(syspoly,'nb')
nk = get(syspoly,'nk')
model = arx([y,u],[na nb nk]);
```

Use `present` to check that the system and the model identified with `arx` are identical.

A.5 Estimating an ARX model corrupted with colored noise

Now we will use the function `arx` to compute different ARX models:

1. Generate u_k as a white noise sequence ($\sigma = 1, N = 1000$, use `randn`).

2. Generate the output with the model:

$$G(z) = \frac{B(z)}{A(z)}$$
$$H(z) = \frac{C(z)}{D(z)}$$

   where $C(z)$ and $D(z)$ are the numerator and denominator of a high pass Butterworth filter (order 4, cutoff 0.2). The noise e_k has a standard deviation equal to 0.01 ($\sigma = 0.01$).

```
u = iddata([],randn(1000,1),1);
[b_butter, a_butter] = butter(4,0.2,'high');
std_butter = b_butter(1);
set(syspoly,'c',b_butter/std_butter);
set(syspoly,'d',a_butter);
set(syspoly,'NoiseVariance',(0.01*std_butter)^2);
e = randn(1000,1);
opt = simOptions('AddNoise',true,'NoiseData',e);
y = sim(syspoly,u,opt);
```

3. Using the data generated in the previous point, identify an ARX model with n_a, n_b and n_k equal to the true n_a, n_b, n_k (use `get(syspoly,'na')`). Note that even though $G(z)$ of the ARX model is in the same model class as $G(z)$ of the data generating model, this is not the case for the noise model $H(z)$. Compare the transfer functions. Are they close? If not, try to explain.

Hint: `bode` allows plotting of more than one transfer function on top of each other. For instance:

```
bode(syspoly,modelpoly);
```

A.6 Estimating an ARX model corrupted with colored noise and unknown order

The previous question was easy since we started from a given model structure of order `[na nb nk]`. In principle, this structure is unknown. A more realistic way to go about this is thus by trying out different model structures and checking which one gives the 'best' model. To generate all the possible structures in the vicinity of the true order (or some initial guess) use:

```
na = get(syspoly,'na')
nb = get(syspoly,'nb')
nk = get(syspoly,'nk')
search_region = struc([na-2:na+2],[nb-2:nb+2],[nk-1:nk+1]);
```

Now, generate a new data set: $u_{\text{val}}, y_{\text{val}}$ in the same way you generated u and y at the beginning of this question:

```
e_val = randn(500,1);
opt = simOptions('AddNoise',true,'NoiseData',e_val);
set(syspoly,'NoiseVariance',0.01*0.01);
u_val = iddata([],randn(500,1),1);
y_val = sim(syspoly,u_val,opt);
```

Use this new dataset as a validation set in `arxstruc`. Afterwards `selstruc` enables you to choose the 'best' model. Does this structure correspond to the true one? If not, try to explain. Check the models that come close in terms of fitness. Compare the transfer function of the model obtained in the previous step with the true one. If there is a difference, try to explain it.

An alternative way of estimating a model with unknown order is to overestimate and apply model reduction methods. A final model with this input-output data can be obtained as follows:

1. Estimate a high order ARX model ($n_a = n_b = 25$), and the delay equal to the given delay or the one estimated from the impulse response).
2. Convert the ARX model to a state space model using `ss`.
3. Inspect the Hankel singular values using `hsvplot`. What is the order?
4. Reduce the state space model using the balanced model reduction techniques (you only need `balred`). Plot the frequency response of the reduced model using `bode`.
5. Convert the result back to a polynomial model using `idpoly`. Compare the coefficients of $A(z)$ and $F(z)$, as well as the $B(z)$ coefficients in both models. Try to list some advantages of this method. You can also go straight for the transfer function using `idtf` on the reduced state space system.

A.7 Effects of a wrong model class

Take u_k a white noise sequence ($\sigma = 1, N = 1000$) and generate an output using an OE (output error) model[3]

$$G(z) = \frac{B(z)}{A(z)}$$
$$H(z) = 1$$

[3]Note that we are using $A(z)$ from the original ARX system as $F(z)$ in our OE model.

and with the standard deviation of the noise sequence e_k equal to 0.05 ($\sigma = 0.05$):

```
b = get(syspoly,'b');
f = get(syspoly,'a');
oepoly = idpoly(1,b,1,1,f,0.05*0.05,1);
opt = simOptions('AddNoise',true,'NoiseData',randn(1000,1));
u = iddata([],randn(1000,1),1);
y = sim(oepoly,u,opt);
```

Determine an OE model in the exact model class (use `oe`). Note that the order of the model class parameters for `arx` was `[na,nb,nk]`, while for `oe` the order is `[nb,nf,nk]`). Plot the identified transfer function and the true transfer function.
Now determine (with the data you just generated) an ARX model as follows:

```
arxmodel = arx([y,u],[nf nb nk]);
```

Compare the identified transfer function with the true one. What goes wrong? Try to solve this problem by low pass filtering the input output data (use `idfilt`). Does the transfer function agree better with the true transfer function now? If you have the time, you can try other filters to solve the problem.
Tip: If you find it hard to improve the ARX model even with low pass filtering, try to notch up the model orders, then retry estimating an ARX model.

A.8 Asymptotic properties and overmodeling

Take u_k a white noise sequence ($\sigma = 1, N = 1000$) passed through a fourth order Butterworth filter:

```
>> [bf,af] = butter(4,0.15);
>> u = filter(bf,af,randn(1000,1));
```

Now generate the output with the OE model:

$$G(z) = \frac{B(z)}{A(z)}$$
$$H(z) = 1$$

in MATLAB:

```
b = get(syspoly,'b');
f = get(syspoly,'a');
oepoly = idpoly(1,b,1,1,f,0.01*0.01,1);
opt = simOptions('AddNoise',true,'NoiseData',randn(1000,1));
u = iddata([],randn(1000,1),1);
y = sim(oepoly,u,opt);
```

Plot the input and output using `plot`. Determine an OE model in the exact model class (meaning the model class of the data-generating model). Plot the transfer function and the true transfer function. Plot on the same plot the uncertainty of the transfer function (3σ, use `bode`, check the documentation). Is the true transfer function within the uncertainty band? If not, try to explain.
Make a pole-zero plot with 3σ error bounds. Are there poles and zeros that are likely to cancel? You can plot the confidence intervals using

```
h = iopzplot(oepoly,arxmodel);
showConfidence(h,3);
```

Overmodel the transfer function by taking n_b and n_f two times the exact value. Repeat the plotting of transfer function and pole-zeros with their error bounds. What is the difference with before? Are there clear pole-zero cancellations?
For the two OE models computed in this exercise, also compute the prediction error using `compare` or `predict` (always with an independent validation set!). Which model predicts the data the best?

A.9 Model assessment

Throughout this question, you used `syspoly` as a reference model to assess the model quality. In reality, the true system is not known, and you will need to resort to other means of validation. In this question, you will explore the possibilities.
Remark: A small remark on the use of `compare` (and `predict`) is in order. These functions calculate 'predicted' outputs. These 'predictions' can be done over different horizons, though. The default horizon in `predict` is $M = 1$, the default in `compare` is $M = \infty$, which are the two extreme (and most important) cases:

- $M = \infty$ (which is the default in `compare`). This is called 'pure simulation', and the errors (the difference between the true output and the simulated output) are called simulation errors. The simulation mechanism is simply:

$$y_k^{\text{sim}} = G(z,\theta)u_k$$

  The simulation error can then be computed as:

$$\begin{aligned} e_k^{\text{sim}} &= y_k - y_k^{\text{sim}} \\ &= [G_0(z)u_k + H_0(z)e_k] - G(z,\theta)u_k \\ &= [G_0(z) - G(z,\theta)]u_k + H_0(z)e_k \end{aligned}$$

  It should thus be clear that these simulation errors are NOT white when $G(z,\theta) = G_0(z)$, unless $H_0 = 1$.

- $M = 1$. This is called 'prediction', and the errors (the difference between the true output and the predicted output) are called prediction errors. The prediction mechanism is:

$$y_k^{\text{pred}} = H^{-1}(z,\theta)G(z,\theta)u_k + [1 - H^{-1}(z,\theta)]y_k$$

  The prediction error can then be computed as:

$$\begin{aligned} e_k^{\text{pred}} &= y_k - y_k^{\text{pred}} \\ &= H(z,\theta)^{-1}[y_k - G(z,\theta)u_k] \\ &= \frac{G_0(z) - G(z,\theta)}{H(z,\theta)}u_k + \frac{H_0(z)}{H(z,\theta)}e_k \end{aligned}$$

  The prediction errors are white when $G(z,\theta) = G_0(z)$ and $H(z,\theta) = H_0(z)$. This is the reason why `resid` uses 'prediction errors' (and not simulation errors).

Take two models you computed in the previous steps (the models should be taken from the same step, since they should be comparable). Validate both models by looking at:

1. The simulation errors (use `compare`).

2. The prediction errors (use `compare` with the prediction horizon set to 1).

3. The whiteness of the prediction error (use `resid`)

4. The cross-correlation between prediction error and input (use `resid`)

Which of the two models do you prefer?