System Identification – Practical Assignment 10: Recursive ARX identification

Logistics

Please reread the logistics part of lab 2, the same rules will apply to this lab. The only thing that changes is the dropbox link, which for this lab is:

`https://www.dropbox.com/request/OLmCCSzHHXpvWqvO1xuS`

Assignment description

In this assignment we will study the recursive variant of the ARX method, see the lecture: *Recursive identification methods*. The Matlab code to interact with the DC motor changes a bit, since we need to work online. Namely, we will use an object for this, created at the beginning of the experiment with `obj = DCMRun.start()` (this constructor figures out the COM port and sets the sampling rate automatically). Then, we will use the following methods of this object:

- `yk = obj.step(uk)` to apply $u(k)$ and read $y(k)$ at each step k.
- `obj.wait()` at the end of the computation for each time step, so as to wait the remainder of the sampling period and therefore synchronize with the next discrete time step.
- `obj.stop()` to stop the experiment (called once, at the end).

The requirements for the lab are the following:

- To start with, we will applying some zeros, a step input (which together with the corresponding output we will also keep for validation), followed by another sequence of zeros to bring the motor back to zero conditions in preparation for the online experiment. The sampling rate is $0.01\,\mathrm{s}$ ($10\,\mathrm{ms}$). The step signal has magnitude around 0.3 and around 70 samples in length.
- To prepare for identification, create (but do not apply yet!) a PRBS signal with amplitudes in the interval $[-0.8, 0.8]$ and a length N of about 200 samples. You can either use `idinput` or your own PRBS code from the previous labs, but in the latter case use a sufficiently large number of bits so that the signal does not repeat.
- Implement the general recursive ARX algorithm which applies the input designed above step by step to the DC motor, and updates the ARX model at each step, see the pseudocode below with additional hints compared to the lecture. The code should produce a matrix $\Theta \in \mathbb{R}^{(na+nb)\times N}$ containing on each column k the parameter vector $\theta(k)$: first the coefficients $a_1, \ldots, a_{na}$ of A, and then the coefficients $b_1, \ldots, b_{nb}$ of B. Use an initial inverse $P^{-1}(0) = 1000 I_{na+nb}$ and a zero initial parameter vector $\widehat{\theta}(0)$. Take $na = nb = 2$, this will work better despite the fact that the system is single-order.
- Compare *on the validation data* the quality of two models: one with the final parameters found after processing the whole dataset; and another after only 5% of the data. Which model is better? Think about the reasons. Hint: You can use `idpoly(A,B,[],[],[],0,Ts)` followed by

`compare` on an `iddata` object for the validation dataset (which you need to create yourself). Do not forget that for `idpoly`, the vectors of polynomial coefficients must be rows and must contain the leading constant coefficients (power 0 of the argument q^{-1}), which must be 1 in A and 0 in B); these leading coefficients are not stored in Θ.

---

initialize $\widehat{\theta}(0)$, an $na + nb$ column vector
initialize $P^{-1}(0)$, an $(na + nb) \times (na + nb)$ matrix
for each step $k = 1, 2, \dots, N$ **do**
    send $u(k)$ to the system, read $y(k)$ from the system
    form ARX regressor vector: $\varphi(k) = [-y(k-1), \cdots, -y(k-na), u(k-1), \cdots, u(k-nb)]^\top$
    find prediction error: $\varepsilon(k) = y(k) - \varphi^\top(k)\widehat{\theta}(k-1)$ (a scalar)
    update inverse: $P^{-1}(k) = P^{-1}(k-1) - \frac{P^{-1}(k-1)\varphi(k)\varphi^\top(k)P^{-1}(k-1)}{1+\varphi^\top(k)P^{-1}(k-1)\varphi(k)}$
    compute weights: $W(k) = P^{-1}(k)\varphi(k)$ (an $(na + nb)$ column vector)
    update parameters: $\widehat{\theta}(k) = \widehat{\theta}(k-1) + W(k)\varepsilon(k)$
    wait for the sampling period to elapse
end for
