System Identification – Practical Assignment 10: Recursive ARX identification

Logistics

This practical assignment should very preferably be carried out by each student separately. Otherwise, if there are more students than computers, *with the explicit agreement of the lab teacher for each group*, students may team up in groups of 2.

The assignment solution consists of Matlab code. Develop this code in a single Matlab script. If you need to create functions, they can be local to the script, see local functions in scripts.

The overall rules for labs are described on the website. For each particular lab, your attendance will only be registered if you have a working, original solution. The teacher will check that your code **works** during the lab class. Only after this has been done, for the **originality** check, upload your solution here:

`https://www.dropbox.com/request/ZcBECRE71pJ9nfH6QBNd`

Upload *only once, a single m-file, named exactly according to the following pattern*:

`L10_ENgh_LastnameFirstname.m`

where `g` is the group, `h` the halfgroup, and the last and first names follow. E.g., `L10_EN32_PopAlex.m`. *If you worked in a group per the above, only upload one file with both student names included*. Any files that are duplicate, nonstandard, inappropriately named, or corresponding to unchecked solutions will not be considered. Files will be automatically tested for plagiarism, and any solution that fails this test will be marked copied; only solutions that pass both the working and the originality test are definitively validated. Therefore, while you are encouraged to discuss ideas and algorithms amongst colleagues, sharing and borrowing pieces of code is forbidden.

Assignment description

In this assignment we will study the recursive variant of the ARX method, see the lecture: *Recursive identification methods*. Each student is assigned an index number by the lecturer. Then, the student downloads the data file that forms the basis of the assignment from the course webpage. The file contains the identification data in variable `id`, and separately the validation data in variable `val`.

From prior knowledge, it is known that the system has order n, given in variable `n` in the data file; that it is of the output error (OE) type; and that it has no time delay. To account for the model type mismatch we will take larger orders for all the ARX models to be found. The recommendation is to take $na = nb = 3 \cdot n$.

- Implement the general recursive ARX algorithm in a Matlab function; see the pseudocode below with additional hints compared to the lecture. This function should take at the input the identification dataset, the model orders na and nb, the initial parameter vector $\theta(0)$, and the initial inverse matrix $P^{-1}(0)$. The function should produce at the output a matrix $\Theta \in \mathbb{R}^{N \times (na+nb)}$ containing on each row 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 (this is compatible with the output of the Matlab function, so it will be easier to compare later).
- Run recursive ARX identification with the function you implemented, on the identification data, using an initial matrix $P^{-1}(0) = \frac{1}{\delta} I_{na+nb} = 100 I_{na+nb}$ (so $\delta = 0.01$). 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 10% of the data. Which model is better? Think about the reasons.

- Repeat the experiment with $P^{-1}(0) = \frac{1}{\delta} I_{na+nb} = 0.01 I_{na+nb}$ (so $\delta = 100$). Consider the results. For which value of δ is the early model worse, and why?

- Optionally, if you still have time, repeat the initial experiment, but now with the already available Matlab function `rarx`. Compare the results it gives (e.g., the 10% and 100% models) with those given by your own function, to verify whether they are the same or similar.

1: initialize $\widehat{\theta}(0)$ (an $na + nb$ column vector), $P^{-1}(0)$ (a $(na + nb) \times (na + nb)$ matrix)
2: **loop** at every step $k = 1, 2, \ldots$
3: retrieve $u(k)$, $y(k)$
4: form ARX regressor vector: $\varphi(k) = [-y(k-1), \cdots, -y(k-na), u(k-1), \cdots, u(k-nb)]^\top$
5: find prediction error: $\varepsilon(k) = y(k) - \varphi^\top(k)\widehat{\theta}(k-1)$ (a scalar)
6: update inverse: $P^{-1}(k) = P^{-1}(k-1) - \frac{P^{-1}(k-1)\varphi(k)\varphi^\top(k)P^{-1}(k-1)}{1+\varphi^\top(k)P^{-1}(k-1)\varphi(k)}$
7: compute weights: $W(k) = P^{-1}(k)\varphi(k)$ (an $(na + nb)$ column vector)
8: update parameters: $\widehat{\theta}(k) = \widehat{\theta}(k-1) + W(k)\varepsilon(k)$
9: **end loop**

Relevant functions from the System Identification toolbox: `rarx`, `idpoly`, `compare`. Additional hints:

- Once you have your polynomials A and B as vectors of coefficients in increasing powers of q^{-1}, use `idpoly(A, B, [], [], [], 0, Ts)` to generate the ARX model, where `Ts` is the sampling period. Do not forget that all vectors of polynomial coefficients must always contain the leading constant coefficients (power 0 of the argument q^{-1}), which must be 1 in A and 0 in B. Keep in mind that the matrix of parameters returned by the algorithm does *not* contain these leading coefficients.

- Predefined function `rarx` takes at the input the identification dataset, the model orders na and nb and the delay nk as a vector, the `'ff', 1` arguments to configure the algorithm as in the lecture, the initial parameter vector $\theta(0)$, and the initial inverse matrix $P^{-1}(0)$. The quantity denoted `P` by documentation of the `rarx` Matlab function is actually the *inverse* matrix P^{-1} from the lecture, so be careful when setting it. The function produces at the output a matrix Θ with the same structure as explained above.