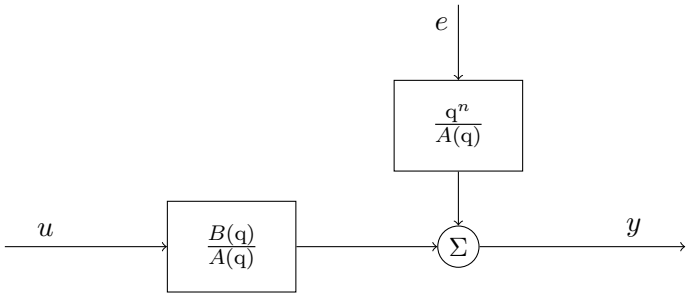
System identification using matlab

Kjartan Halvorsen

October 2, 2019

1 Least-squares parameter estimation

The model structure used is the so-called ARX (Auto-Regressive with eXogenous input) model:



where $u(k)$ is a known input signal, $y(k)$ is the output signal and $e(k)$ is a disturbance (an unknown input signal) in the form of a zero-mean white noise sequence.

The model can be written

$$A(\mathrm{q})y(k) = B(\mathrm{q})u(k) + \mathrm{q}^n\, e(k)$$

$$(\mathrm{q}^n + a_1\,\mathrm{q}^{n-1} + \cdots + a_n)y(k) = (b_0\,\mathrm{q}^m + b_1\,\mathrm{q}^{m-1} + \cdots + b_m)u(k) + \mathrm{q}^n\, e(k)$$

$$y(k+n)+a_1y(k+n-1)+\cdots+a_ny(k) = b_0u(k+m)+b_1u(k+m-1)+\cdots+b_mu(k)+e(k+n)$$

$$y(k+1)+a_1y(k)+\cdots+a_ny(k-n+1) = b_0u(k+m-n+1)+b_1u(k+m-n)+\cdots+b_mu(k-n+1))+e(k+1)$$

The one-step-ahead predictor for this model becomes

$$\begin{aligned}\hat{y}(k+1) &= -a_1 y(k) - a_2 y(k-1) - \cdots - a_n y(k-n+1) \\ &\quad + b_0 u(k+m-n+1) + b_1 u(k+m-n) + \cdots + b_m u(k-n+1) \\ &= \underbrace{\begin{bmatrix} -y(k) & \cdots & -y(k-n+1) & u(k+m-n+1) & \cdots & u(k-n+1) \end{bmatrix}}_{\varphi^{\mathrm{T}}(k+1)} \underbrace{\begin{bmatrix} a_1 \\ \vdots \\ a_n \\ b_0 \\ \vdots \\ b_m \end{bmatrix}}_{\theta} \\ &= \varphi^{\mathrm{T}}(k+1)\theta.\end{aligned}$$

Note that the white noise term $e(k+1)$ by definition cannot be predicted from knowledge of previous values in the sequence (which we don't know) nor from previous output values $y(t)$, $t \leq k$ (which could have been used to estimate $\hat{e}(k)$). Therefore $e(k+1)$ is predicted by its mean value which is zero. Note also that if our model with $\theta = \theta^*$ is perfect (θ^* contains the true parameters for the system which generated the data), then the prediction error equals the white noise disturbance: $\epsilon(k+1) = y(k+1) - \varphi^{\mathrm{T}}(k+1)\theta^* = e(k+1)$. Therefore, we can check how good a models is by testing how close the prediction errors resembles a white noise sequence.

The system of equations in the unknown system parameters θ is

$$\Phi\theta = y,$$

where

$$\Phi = \begin{bmatrix} \varphi^{\mathrm{T}}(n+1) \\ \varphi^{\mathrm{T}}(n+2) \\ \vdots \\ \varphi^{\mathrm{T}}(N) \end{bmatrix},$$

$$y = \begin{bmatrix} y(n+1) \\ y(n+2) \\ \vdots \\ y(N) \end{bmatrix}.$$

The least-squares solution to this system of equations is, by definition, the solution $\hat{\theta}$ which minimizes the sum of squares of the residuals $\epsilon = y - \Phi\theta$, i.e. the solution that minimizes the criterion

$$J(\theta) = \epsilon^{\mathrm{T}}\epsilon = \sum_i \epsilon_i^2.$$

It is given by

$$\hat{\theta}_{LS} = \underbrace{(\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}}_{\Phi^{+}} y,$$

where Φ^{+} is called the *Moore-Penrose invers* of the (typically) non-square, tall matrix Φ.

1.1 Instructions

1. Download the data: `http://alfkjartan.github.io/files/sysid_exercise_data.mat` The data consist of 12 vectors `u1`, `y1`, `u1_val`, `y1_val`, `u2`, `y2`, `u2_val`, `y2_val`, `u3`, `y3`, `u3_val`, `y3_val` corresponding to three different LTI systems.

   `ui` input data for system i

   `yi` output data for system i

   `ui_val` validation input data for system i

   `yi_val` validation output data for system i

   The systems are

   a) $H_1(z) = \frac{b}{z+a} = \frac{0.2}{z-0.8}$

   b) $H_2(z) = \frac{b_0 z + b_1}{z(z+a_1)} = \frac{0.0625z+0.0375}{z(z-0.9)}$

   c) $H_3(z) = \frac{b}{z^2+a_1 z+a_2} = \frac{0.18}{z^2-1.4z+0.58}$

2. For each case estimate a model using the input-output data. Formulate the one-step ahead predictor $\hat{y}(k) = \varphi^{\mathrm{T}}(k)\theta$ and the system of equations $\Phi\theta = y$, and find the least-squares solution to the system. Compare with the true parameters.

3. Validate your model using the validation data. Let's say you have obtained estimates of the numerator $\hat{B}(z)$ and denominator $\hat{A}(z)$ of your model. You can then simulate the model using the validation data (a data set which is **not** used for parameter estimation) like this

```
H = tf(B, A, 1); % Create the discrete-time LTI model

% Simulate the model.
ysim = lsim(H, u_val);
% The above is also known as infinite-step ahead prediction
% and works only for stable models. It is often better to
% validate the model using a k-step ahead predictor.

e = y_val - ysim; % The error
```

```
% Calculate the root mean square of the error
RMSerror =      %Your code here

% Plot
figure()
subplot(121)
stairs(y_val)
hold on
stairs(ysim)
legend('Validation output', 'Simulated output')
subplot(122)
stairs(e)
title(sprintf('Simulation error. RMS=%f', RMSerror))
```

1.2 About the validation simulation

In general it is preferred to compare the validation output to the output from a k-step ahead predictor, where k is chosen to correspond to a typical time interval of the data. For control models where we have used the rule-of-thumb of 4-10 sampling periods per rise time, a choice of k=10 is reasonable. A choice of k=1 (one-step ahead predictor) gives unreliable validation, since the trivial predictor $\hat{y}(k+1) = y(k)$ can give good predictions if the sampling period is short ("the weather in one minute from now will be equal to the weather now"). Choosing k extremely large corresponds to pure simulation of the model (the predicted output sequence depends only on the input sequence) and will not work for unstable models. Also, for models with integration a small bias in the input sequence will be accumulated and give poor validation result, even for a good model.

The system identification toolbox provides such a function that computes the k-step ahead prediction. See `https://www.mathworks.com/help/ident/ref/predict.html`. An implementation that uses only basic matlab functions can be downloaded here `http://alfkjartan.github.io/files/predictlti.m`.