
Katholieke Universiteit Leuven

System Identification & Modeling

# EXERCISE REPORT (PART 1)

*Student:* Henri De Plaen (r0681349)

*Professors:* Johan Schoukens
Philippe Dreesen

H05R4a
2016 - 2017

January 7, 2017

# Chapter 1

# Noise on Input and Output

## 1.1 The Instrumental Variables Method

The Least Square Estimator isn't biased if the only noise is on the output, and this noise has zero mean. In this case, there is also noise on the input, although both noises have zero mean. Thus, the estimator is thus biased.

*Proof.* The bias of an estimator is defined as the difference between its expectation and its true value: $\mathbb{B}\left[\hat{\theta}\right] = \mathbb{E}\left[\hat{\theta}\right] - \theta$. Furthermore, an estimator is constitent if and only if it is convergent and asymptotically unbiased. We will consider here the constistency, i.e. the convergence in probability of the estimator when the sample size becomes infinite, hence we have here $N = 5000$. With the LSE with $\theta = R_0$ and all signals being decorrelated, we have,

$$\begin{aligned}
\lim_{N\to\infty} \hat{R}_{LS} &= \lim_{N\to\infty} \frac{\sum u(k)i(k)}{\sum i(k)i(k)} \\
&= \lim_{N\to\infty} \frac{\sum (R_0 i_0(k)^2 + R_0 i_0(k) n_i(k) + n_u(k) n_i(k))}{\sum (i_0(k)^2 + 2 n_i(k) i_0(k) + n_i(k)^2)} \\
&= R_0 \frac{\sigma_{i_0}^2}{\sigma_{i_0}^2 + \sigma_{n_i}^2}
\end{aligned}$$

The estimator converges, but not to the correct value. It must then be asymptotically biased:

$$\lim_{N\to\infty} \mathbb{B}\left[\hat{R}_{LS}\right] = -R_0 \frac{1}{1 + \sigma_{n_i}^2/\sigma_{i_0}^2} \tag{1.1}$$

As the asymptotic bias is a stronger property than the bias, our estimator is then also biased. □

It is here clear that the estimator is unbiased if there is no noise on the input (i.e. $\sigma_{n_i}^2 = 0$). Furthermore, the more the variance of the input noise is little compared the true signal, the more the estimator is unbiased. In our case, we have $\sigma_{n_i}^2 = \sigma_{i_0}^2$, thus $\hat{R}_{LS} = \frac{1}{2} R_0$.

For the IV-methods, as we can notice on figure 1.1, there is no bias for the white output noise case and the more it is filtered, the greater the bias gets. This can be seen mathematically if we compute the bias of those methods.

*Proof.* Applying the same definition an asumptions as above to the IV-estimator, we obtain,

$$\begin{aligned}\lim_{N\to\infty}\hat{R}_{IV} &= \lim_{N\to\infty}\frac{\sum u(k)i(k+s)}{\sum i(k)i(k+s)}\\ &= \lim_{N\to\infty}\frac{\sum(R_0 i_0(k)i_0(k+s)+R_0 i_0(k)n_i(k+s)+n_u(k)i_0(k+s)+n_u(k)n_i(k+s))}{\sum(i_0(k)i_0(k+s)+i_0(k)n_i(k+s)+n_i(k)i_0(k+s)+n_i(k)n_i(k+s))}\\ &= R_0\frac{R_{i_0 i_0}(s)}{R_{n_i n_i}(s)+R_{i_0 i_0}(s)}\end{aligned}$$

Thus, we get for the asymptotic bias,

$$\lim_{N\to\infty}\mathbb{B}\left[\hat{R}_{IV}\right] = -R_0\frac{1}{1+R_{i_0 i_0}(s)/R_{n_i n_i}(s)} \tag{1.2}$$

Similarly as before, our estimator is biased. □

The noise being almost left uncorrelated for $f_{noise} = 0.999$, we observe a practically unbiased estimator, but the more the noise is decorrelated (i.e. filtered with a lesser $f_{noise}$), the greater the bias gets, as observed before. $R_{n_i n_i}(s)$ and $R_{i_0 i_0}(s)$ for the correspond shift to the one needed in the bias, are marked with a circle on the auto-correlation plots.

If we increase the shift, as it can be observed in figure 1.2, the bias gets more little. This is due to the decreasing auto-correlation of $n_i$ with the lag. For $s = 5$, its practically zero, the estimator being then also practically unbiased.

The reason for the bias is the existence of an input noise. This can be mathematically seen in the expressions above. I will still try to give an interpretation of it. If there is no shift, both correlation functions will be positive ($R_{n_i n_i}(0) = \sigma^2_{n_i}$ and $R_{i_0 i_0}(0) = \sigma^2_{i_0}$) what will result in a negative bias, an underestimate. This is the case for LS. For a one shift, both correlations functions are also positive, what will have the same effect, as in $IV_1$. For $IV_2$, $R_{n_i n_i}(2) < 0$ and $R_{i_0 i_0}(2) > 0$ are of different sign, but the first one being also much more smaller in magnitude than the latter. This will result in a negative denominator and thus a positive bias, an overestimate. Finally, for $IV_3$, $R_{n_i n_i}(3) \simeq 0$, what an almost infinite denominator give, thus a practically zero bias.

If the shift is great, this results in multiplying values that are more decorrelated with each other, as the auto-correlation functions of both the noise and the true signal is decreasing with the lag, especially the true signal. This is due to filter response not attenuating values for small frequencies, or greater wavelengths, thus value far apart from each other. The consequence is a small denominator and much more sensible to value changes. This is the cause for the observed variance increase of the estimator as the shift increases.

We could conclude that an IV-estimator works well if $f_{gen} \ll f_{noise}$, resulting in a small bias and a small variance.

## 1.2 The Errors-In-Variable Method

The LS estimator is biased, this is due, similarly as before to the input noise. Using the formula's derived for the previous question, we get, with $\sigma^2_{n_i} = 0.001$, $\sigma^2_{i_0} = 0.01$ and $R_0 = 1000$, a bias of -9.900990099... This can be observed on figure 1.3. We also see that the EIV-estimator is not biased.

An interpretation of the LS-estimator here is a linear regression. It minimize for $R_0$ the sum of the

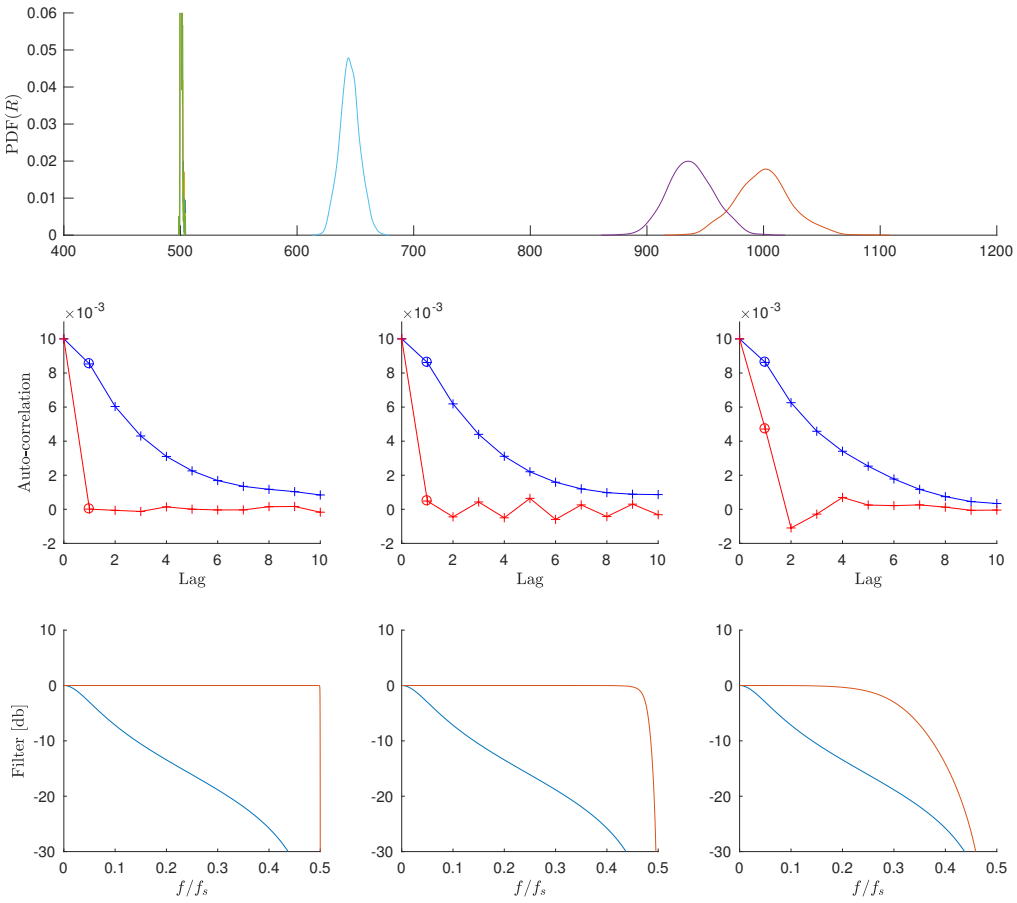


Figure 1.1: Study of the LS and IV estimates for a varying noise filter bandwidth and fixed shift parameter s = 1. Top: the LS (green) and IV estimates. $IV_1$ (red), $IV_2$ (purple) and $IV_3$ (blue) correspond to the noise filters with cut-off frequencies 0.999, 0.95 and 0.6 respectively. Middle: the auto-correlation of $i_0$ (blue) and $n_i$ (red) for the different noise filters. Bottom: filter characteristics of $i_0$ (blue) and of $n_i$ (red).

distances between the model and the output, called residue $r(k) = u(k) - R_0 i(k)$. In other words $\hat{R}_{LS} = \arg\min_R \sum_{\forall k} r(k;R)^2$. By doing this, it creates a bias. A way to avoid it is to take the bias into account, as we know the variances on which it depends. This is what the EIV-estimator tries to do. Another way of interpreting it is to consider it as the minizer of the orthogonal distance between the points and the linear regression.

With this parameters, the IV-estimator won't be much interesting, as its strength is to use the correlation of the true input signal to mask its noise. Here, because of a lack of filter, there is no correlation at all.

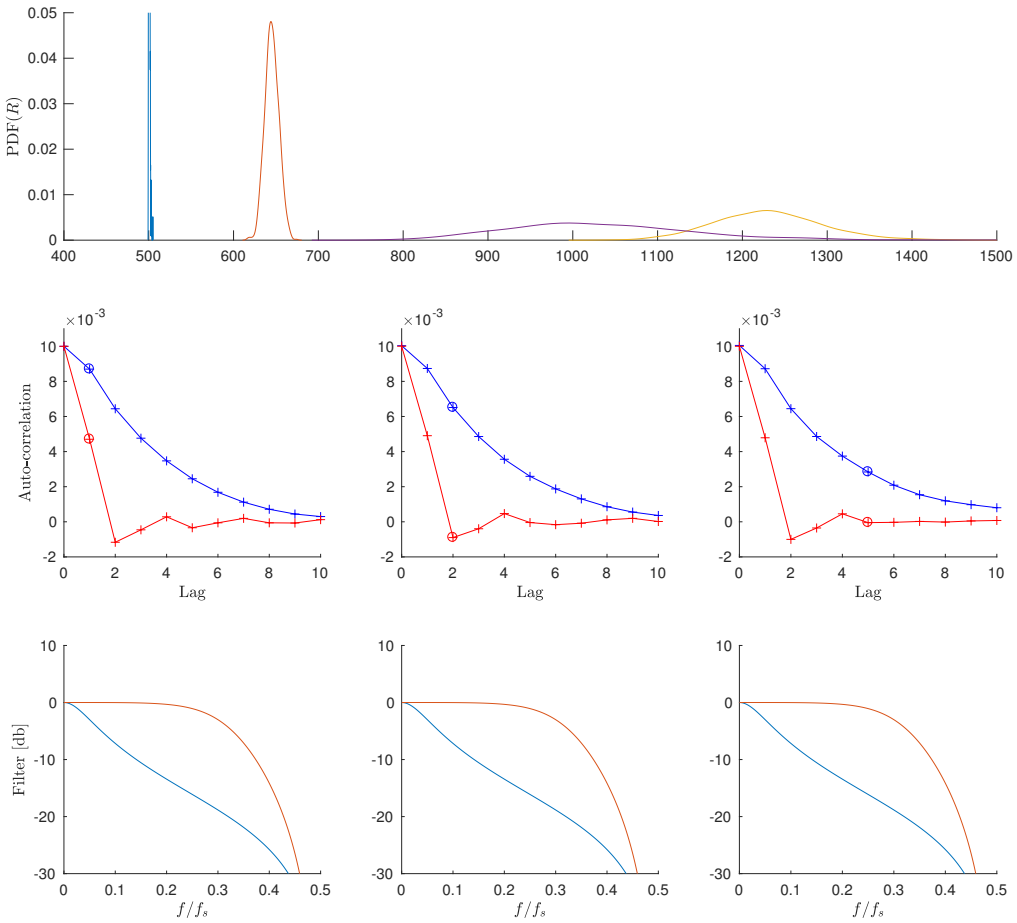


Figure 1.2: Study of the LS and IV estimates for a fixed noise filter bandwidth and varying shift parameter s = 1, 2, 5. Top: the LS (blue) and IV estimates. $IV_1$ (red), $IV_2$ (yellow) and $IV_3$ (purple) correspond to a shift of 1, 2 and 5 respectively. Middle: the auto-correlation of $i_0$ (blue) and $n_i$ (red). Bottom: filter characteristics of $i_0$ (blue) and of $n_i$ (red).

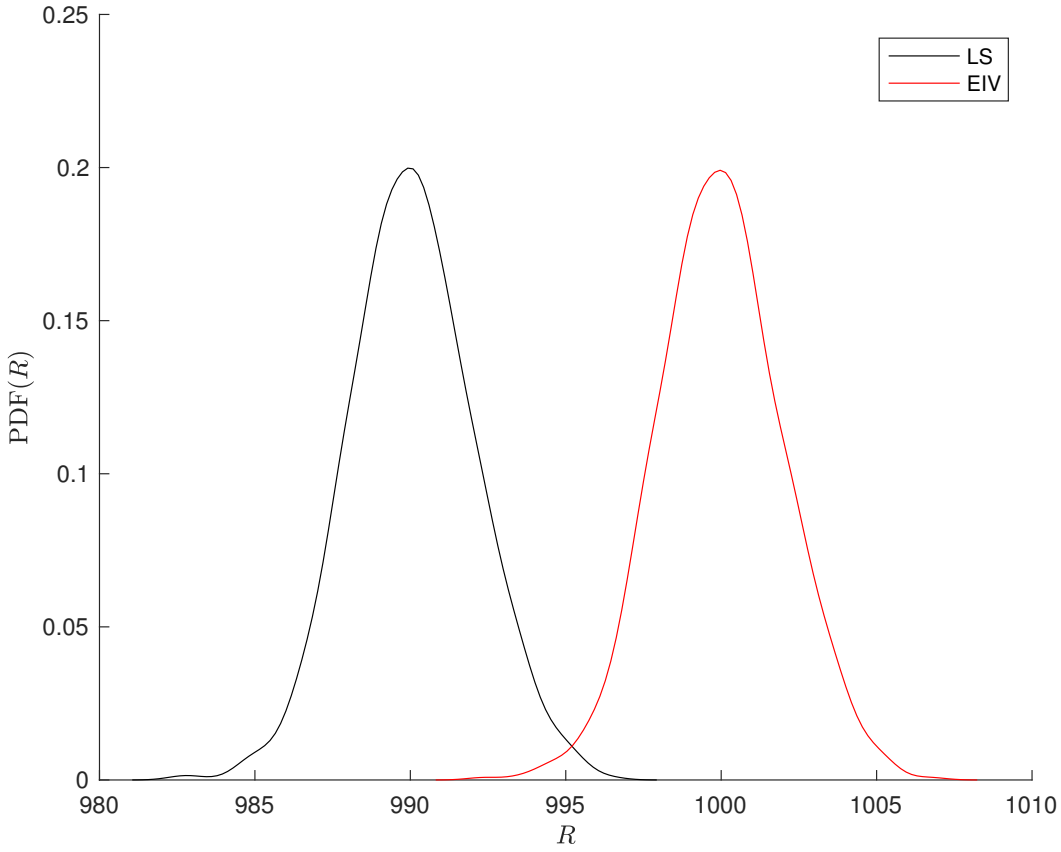


Figure 1.3: Comparison of the PDF of the LS (black) and EIV (red) estimate, calculated with prior known variances.

# Chapter 2

# Model selection using AIC

The model is being optimised for the estimation set. The more we increase the order, or in other words, the more we give degrees of freedom to the estimator, the more it can optimise itself on the estimation set. This explains the decrease of $V_{est}$ with the increase of the order, as we can see on figure 2.1.

The estimator of certain order is the one corresponding the model which gives the lowest value of the cost function between all models of the same order. The reason for the decrease is that all the models of an order $n$ are always a subspace of all the models of order $n+1$. There must thus always be one model of order $n+1$ that is at least as good (in the sense of the cost function) as the best of order $n$.

If the order is as big as the size of the estimation size, it will have as much degrees of freedom, thus it can model the estimation perfectly. The cost function would then be 0. For all bigger orders, there will even be an infinite amount of models without any error, but as we will see further, this doesn't makes a lot of sense, as we are not interested in extreme values of the order.

In comparison to $V_{est}$, $V_{val}$ does not decrease continually as the order increases, but starts to increase again until a certain order. I will try to give an intuitive explanation. The model used for the validation set is the one that has been optimised on the estimation set.

When the order is relatively little, the transfer function isn't complex enough to model the estimation general idea of the data, in this case the true function. Modeling the noise asks much more complexity of the model, so it doesn't have a real influence. The only differences between the estimation set and the validation set is the size and the noise. The model is independent of the input size and isn't influenced by the noise for relative little orders. Therefore, it should perform as well on both sets, explaining the decrease on the validation set as we increase the order, as well as the decrease on the estimation set. This is called *underfitting*.

At a certain order, we see a minimum on $V_{val}$: the model is perfectly optimized. It has a sufficient order to model the complexity of the transfer function, but a sufficiently little order to not take the noise into account, thus optimizing itself on the specificities of the estimation set.

After this optimal point, $V_{val}$ starts to increase again. The model is too complex and has also optimized itself on the specificities of the estimation set, i.e. its noise. It doesn't model the general scheme anymore, but the particular case on which it has learnt (been optimized). This is called *overfitting* [1].

[1] *Wikipedia gives a good description of overfitting (underfitting is then the opposite)*:
In statistics and machine learning, one of the most common tasks is to fit a "model" to a set of training data, so as to be able to make reliable predictions on general untrained data. In overfitting, a statistical model describes random error or noise instead of the underlying relationship. Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfit has poor predictive performance, as it overreacts to minor fluctuations in the training data.

Always on figure 2.1, we see that using the Akaike Information Criterion (AIC), we don't need a validation set, the first one having the same minimum as the latter.

If we increase variance of the disturbing noise, we notice that the optimal order gets smaller. The reason for it is the noise being proportionally more present on the data, having more influence in the optimisation. The model has to be less complex to avoid overfitting. We can extrapolate and conclude that for an undisturbed signal, the optimal order is as big as possible. However, if the order is as big as the estimation sample size, the optimal model of that order will already give a zero cost function (the model will already be "perfect"). Using a greater model will thus be a waste of computation power and time.

Looking at the normalized cost function using the output data without disturbing noise gives a better idea of the performance of the model, as it gives the error between the model and the ideal value it should be giving, not a noisy one. $V_0$ is thus a more fare indicator of the performance of the model than $V_{val}$. Normalizing and taking the root of it is interesting as gives a better comparison with other estimators and models, as the RMS is a generalized mean with exponent 2, thus a more general indicator, measuring the imperfection of the fit of a model.

As we can still see on figure 2.1, the choice a an order is not very critical, the minimum being really flat. Choosing an order slightly bigger or smaller than the ideal value won't change the performance a lot.

If we look at the maximal model order of the ones we've tested ($I = 100$), we will notice of course overfitting, as we discussed before. Nevertheless, we also notice that $V_{val}$ is greater than 1 and normalized $V_0$ is more in the range of 0.3. The cause of this is normalisation. The latter can be interpreted as the distance between the correct point and the modelled point in the data set hyperspace. A normalized RMS of 1 would mean that the model is as noisy as the output with the noise, but doesn't model the true signal at all (greater than one says that its even noisier); and 0 being that it models the true signal perfectly. Without normalization, or being a more general indicator, $V_{val}$ doesn't give a good interpretation.

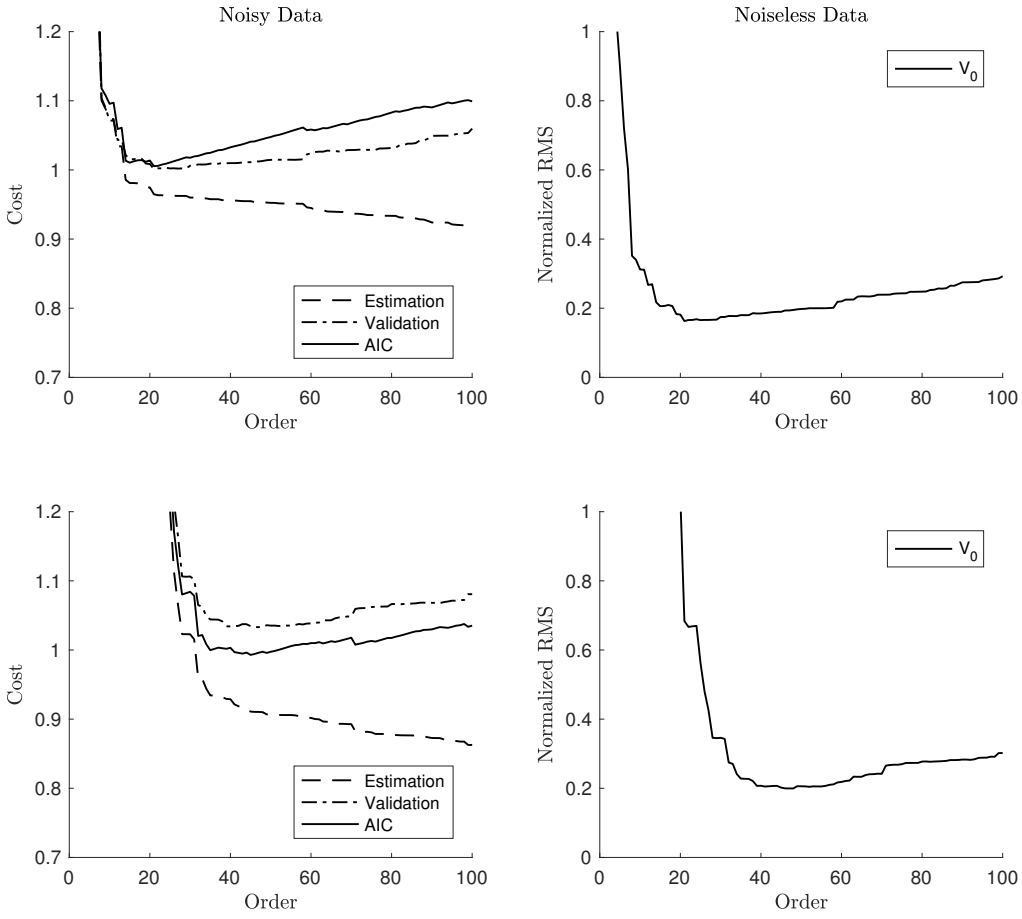


Figure 2.1: Comparison of the normalised cost functions for estimation data, validation data and AIC-criterion (left) and plot of $V_0$ (right) for $\sigma_{n_y} = 0.5$ (top) and $\sigma_{n_y} = 0.05$ (bottom).

# Appendix

# Appendix A

# Matlab source code

Running the code just necessitates to copy all files in a directory. Each question or experiments has a corresponding script, which just has to be runned. All the files are provided in the zip-file with this report. Nevertheless, a copy is given here under.

## A.1 Noise on Input and Output

Function for calculating the estimator:

```
function varargout = r_estimator(varargin)
%R_ESTIMATE Calculates an estimator for R in the eq u=R*i
%r_estimate = r_estimator(sample_u, sample_i, method_name, parameters)
%
%   INPUT:
%   sample_u = samples of the tension u
%   sample_i = samples of the current i
%   method_name = choice between 3 methods
%       'ls' Least Square
%           Parameters:
%               none
%       'iv' Instumental Variables
%           Parameters:
%               s = shift parameter
%       'eiv' Errors-In-Variable
%           Parameters:
%               sigma_nu = standard deviation of the tension noise
%               sigma_ni = standard deviation of the current noise
%   OUTPUT:
%   r_estimate = estimation of R
%
%Author: HENRI DE PLAEN (r0681349)
%Date: 01-03-2017
%Katholieke Universiteit Leuven

sample_u = varargin{1} ; sample_i = varargin{2} ;
sample_u = sample_u(:) ; sample_i = sample_i(:) ;
assert( size(sample_u,1) == size(sample_i,1), 'The samples must have the ...
    same size') ;

switch varargin{3}
    case 'iv'
        assert(nargin==4, 'Method iv: parameters not correct') ;
        varargout{1} = iv_r(sample_u, sample_i, varargin{4}) ;
```

```
    case 'eiv'
        assert(nargin==5, 'Method eiv: parameters not correct') ;
        varargout{1} = eiv_r(sample_u, sample_i, varargin{4}, varargin{5}) ;
    case 'ls'
        assert(nargin==3, 'Method ls: parameters not correct') ;
        varargout{1} = iv_r(sample_u, sample_i, 0) ;
    otherwise, error('Unkwnown method name') ;
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function r_estimate = iv_r(sample_u, sample_i, s)

assert(isscalar(s) && s>=0, 's has to be a positive scalar') ;

u_k  = sample_u(1:end-s) ;
i_ks = sample_i(1+s:end) ;
i_k  = sample_i(1:end-s) ;

num = sum(u_k.*i_ks) ; denom = sum(i_k.*i_ks) ;
r_estimate = num/denom ;

end

function r_estimate = eiv_r(sample_u, sample_i, sigma_nu, sigma_ni )

assert(isscalar(sigma_nu) && isscalar(sigma_ni), 'sigma has to be a scalar') ;

r_estimate = ( sum(sample_u.^2)/sigma_nu^2 - sum(sample_i.^2)/sigma_ni^2 + ...
    sqrt( (sum(sample_u.^2)/sigma_nu^2 - sum(sample_i.^2)/sigma_ni^2)^2 + ...
    4*sum(sample_u.*sample_i)^2/sigma_nu^2/sigma_ni^2 ) ) / ...
    ( 2*sum(sample_u.*sample_i)/sigma_nu^2 ) ;
end
```

### A.1.1 The Instrumental Variables Method

Script for the questions (two parts):

```
%% PRELIMINARIES
N = 5000;
R0 = 1000 ;
fgen = .1 ;
fnoise = [ .999, .95, .6 ] ;
sigma_i0 = .1 ;
sigma_ni = .1 ;
sigma_nu = 1 ;

expeset_size = 1000 ;
maxlag = 10 ;
%prealloc
expe_ls = zeros(expeset_size,size(fnoise,2)) ;
expe_iv = zeros(expeset_size,size(fnoise,2)) ;

%% GENERATING EXPERIMENTS
set(0,'DefaultTextInterpreter','Latex')
figure ; hold on ;

```

```
h = waitbar(0) ;
for inoise = 1:size(expe_ls,2)
    for iexpe = 1:expeset_size
        e1=randn(N,1);
        [bgen,agen] = butter(1,fgen);
        i0 = filter(bgen,agen,e1);
        i0 = sigma_i0/std(i0)*i0 ; %rescaling

        e2=randn(N,1);
        [bnoise,anoise] = butter(2,fnoise(inoise));
        ni = filter(bnoise,anoise,e2);
        ni = sigma_ni/std(ni)*ni ; %rescaling

        u0 = R0*i0 ;
        nu = sigma_nu*randn(N,1) ; %rescaling

        i = i0 + ni ;
        u = u0 + nu ;

        expe_ls(iexpe,inoise) = r_estimator(u,i,'ls') ;
        expe_iv(iexpe,inoise) = r_estimator(u,i,'iv',1) ;

        waitbar(((inoise-1)*expeset_size + iexpe)/3000, h) ;
    end

    %% PLOT
    subplot(3,3,[1 2 3]) ; axis([400 1200 0 .06]) ; hold on ;
    [f,xi] = ksdensity(expe_ls(:,inoise)) ; plot(xi,f) ;
    [f,xi] = ksdensity(expe_iv(:,inoise)) ; plot(xi,f) ;
    ylabel('PDF($R$)') ;

    subplot(3,3,3+1) ; ylabel('Auto-correlation') ; hold on ;
    subplot(3,3,3+inoise) ; hold on ; xlabel('Lag') ;
    [i0_corr, i0_lags] = xcorr(i0,maxlag,'unbiased') ;
    [ni_corr, ni_lags] = xcorr(ni,maxlag,'unbiased') ;
    plot(i0_lags(maxlag+1:end),i0_corr(maxlag+1:end), ...
        '-+b', ni_lags(maxlag+1:end), ni_corr(maxlag+1:end), '-+r') ;
    plot(i0_lags(maxlag+2),i0_corr(maxlag+2), ...
        'ob', ni_lags(maxlag+2), ni_corr(maxlag+2), 'or') ;
    axis([0 10 -.002 0.011]);

    subplot(3,3,2*3+1) ; ylabel('Filter [db]') ; hold on ;
    subplot(3,3,2*3+inoise) ; hold on ; xlabel('$f/f_s$') ;
    [hz,w] = freqz(bgen,agen,8192) ; plot(w/2/pi,20*log10(abs(hz))) ;
    [hz,w] = freqz(bnoise,anoise,8192) ; plot(w/2/pi,20*log10(abs(hz))) ;
    axis([0 .5 -30 10]) ;
end

close(h) ; hold off ;
```

```
%% PRELIMINARIES
N = 5000 ;
R0 = 1000 ;
fgen = .1 ;
fnoise = .6 ;
sigma_i0 = .1 ;
sigma_ni = .1 ;
sigma_nu = 1 ;
s = [0, 1, 2, 5] ;

```

```
expeset_size = 1000 ;
%prealloc
expe = zeros(expeset_size,size(s,2)) ;

%% GENERATING EXPERIMENTS
figure ; hold on ;

h = waitbar(0) ;
for imethod = 1:size(s,2)
    for iexpe = 1:expeset_size
        e1=randn(N,1);
        [bgen,agen] = butter(1,fgen);
        i0 = filter(bgen,agen,e1);
        i0 = sigma_i0/std(i0)*i0 ; %rescaling

        e2=randn(N,1);
        [bnoise,anoise] = butter(2,fnoise);
        ni = filter(bnoise,anoise,e2);
        ni = sigma_ni/std(ni)*ni ; %rescaling

        u0 = R0*i0 ;
        nu = sigma_nu*randn(N,1) ; %rescaling

        i = i0 + ni ;
        u = u0 + nu ;

        expe(iexpe,imethod) = r_estimator(u,i,'iv',s(imethod)) ;

        waitbar(((imethod-1)*expeset_size + iexpe)/expeset_size/size(s,2), ...
            h) ;
    end
    %% PLOT
    subplot(3,3,[1 2 3]) ; axis([400 1500 0 .05]) ; hold on ;
    [f,xi] = ksdensity(expe(:,imethod)) ; plot(xi,f) ;
    ylabel('PDF($R$)') ;

    if imethod ~= 1
        subplot(3,3,3+1) ; ylabel('Auto-correlation') ; hold on ;
        subplot(3,3,3+imethod-1) ; hold on ; xlabel('Lag') ;
        [i0_corr, i0_lags] = xcorr(i0,maxlag,'unbiased') ;
        [ni_corr, ni_lags] = xcorr(ni,maxlag,'unbiased') ;
        plot(i0_lags(maxlag+1:end),i0_corr(maxlag+1:end), ...
            '-+b', ni_lags(maxlag+1:end), ni_corr(maxlag+1:end), '-+r') ;
        plot(i0_lags(maxlag+1+s(imethod)),i0_corr(maxlag+1+s(imethod)), ...
            'ob', ni_lags(maxlag+1+s(imethod)), ...
                ni_corr(maxlag+1+s(imethod)), 'or') ;
        axis([0 10 -.002 0.011]);

        subplot(3,3,2*3+1) ; ylabel('Filter [db]') ; hold on ;
        subplot(3,3,2*3+imethod-1) ; hold on ; xlabel('$f/f_s$') ;
        [hz,w] = freqz(bgen,agen,8192) ; plot(w/2/pi,20*log10(abs(hz))) ;
        [hz,w] = freqz(bnoise,anoise,8192) ; plot(w/2/pi,20*log10(abs(hz))) ;
        axis([0 .5 -30 10]) ;
    end
end
close(h) ; hold off ;
```

### A.1.2 The Errors-In-Variable Method

Script for the question:

```
%% PRELIMINARIES
N = 5000;
R0 = 1000 ;
sigma_i0 = .01 ;
sigma_ni = .001 ;
sigma_nu = 1 ;

expeset_size = 1000 ;
%prealloc
expe = zeros(expeset_size,2) ;

%% EXPERIMENTS
for iexpe = 1:expeset_size
    i0=sigma_i0*randn(N,1) ; ni=sigma_ni*randn(N,1) ;
    u0 = R0*i0 ; nu = sigma_nu*randn(N,1) ;
    i = i0 + ni ; u = u0 + nu ;

    expe(iexpe,1) = r_estimator(u,i,'ls') ;
    expe(iexpe,2) = r_estimator(u,i,'eiv',sigma_nu,sigma_ni) ;
end

%% PLOTS
figure ; hold on ;
[f1,xi1] = ksdensity(expe(:,1)) ; plot(xi1,f1,'-k') ;
[f2,xi2] = ksdensity(expe(:,2)) ; plot(xi2,f2,'-r') ;
set(0,'DefaultTextInterpreter','Latex') ;
ylabel('PDF($R$)') ; xlabel('$R$') ;
axis([980 1010 0 0.25]) ; legend('LS','EIV') ;
```

## A.2 Model selection using AIC

Function for calculating the estimator:

```
function varargout = g_estimator(varargin)
%R_ESTIMATE Calculates an estimator for g in the eq y=conv(g,u)
%[r_estimate,cost] = g_estimator(sample_y, sample_u, order, method_name)
%
%   INPUT:
%   sample_y = sample of the output signal y
%   sample_u = sample of the input signal u
%   order = order size
%
%   OUTPUT:
%   g_estimate = FIR estimation of transfer function
%   cost_ls = value of the cost function for Least Squares
%   cost_aic = value of the cost function for Akaike Information Criterion
%
%Author: HENRI DE PLAEN (r0681349)
%Date: 02-03-2017
%Katholieke Universiteit Leuven

assert(nargin==3,'incorrect number of input arguments');

sample_y = varargin{1} ; sample_u = varargin{2} ;
sample_y = sample_y(:) ; sample_u = sample_u(:) ;
```

```
order = varargin{3} ;

assert( size(sample_y,1) == size(sample_u,1), ...
    'the samples must have the same size') ;
assert(isscalar(order)&&(order>=0), ...
    'order has to be a positive integer') ;

varargout{1} = g_ls(sample_y,sample_u,order) ;
varargout{2} = cost_ls(sample_y,sample_u,varargout{1}) ;
varargout{3} = (1+2*size(varargout{1},1)/size(sample_u,1))*varargout{2} ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function g_est = g_ls(y, u, order)

K = convmtx(u,order) ; K = K(1:size(u,1),:) ;
g_est = (transpose(K)*K)\(transpose(K)*y) ;

end

function cost = cost_ls(y, u, g)

N= size(u,1) ; yhat = filter(g,1,u) ;
cost = 1/N*sum((y-yhat).^2) ;
end
```

Script for the question:

```
%Author: HENRI DE PLAEN
%r0681349
%KULeuven
%
%EXERCISES - SYSTEM ID. & MOD.
%Part Schoukens - Q3 (AIC)
%
%!! Works only on Matlab R2016b or later !!


%% PRELIMINARIES
Nest = 1000 ;
Nval = 10000 ;
su0 = 1 ;
sny = [.5 .05] ;
[b,a] = cheby1(3,.5,[2*.15 2*.3]) ;
order=0:100 ;

expeestls = zeros(size(order,2),size(sny,2)) ;
expeestaic = zeros(size(order,2),size(sny,2)) ;
expevalls = zeros(size(order,2),size(sny,2)) ;
costvalls = zeros(size(order,2),size(sny,2)) ;
%% EXPERIMENT
for iexpe=1:size(sny,2)
    %EST
    u0est=su0*randn(Nest,1);
    y0est=filter(b,a,u0est);
    nyest=sny(iexpe)*randn(Nest,1);
    yest=y0est+nyest;

    %VAL
    u0val=su0*randn(Nval,1);
```

```
    y0val=filter(b,a,u0val);
    nyval=sny(iexpe)*randn(Nval,1);
    yval=y0val+nyval;

    for iorder=1:size(order,2)
        %EST
        [g,lsest,aicest] = g_estimator(yest,u0est,order(iorder)) ;
        expeestls(iorder,iexpe) = lsest/sny(iexpe)^2 ;
        expeestaic(iorder,iexpe) = aicest/sny(iexpe)^2 ;

        %VAL
        expevalls(iorder,iexpe) = cost_ls(yval, u0val, g)/sny(iexpe)^2 ;
        costvalls(iorder,iexpe) = sqrt(cost_ls(y0val, u0val, ...
            g)/sny(iexpe)^2) ;
    end
end

%% PLOTS
figure ; set(0,'DefaultTextInterpreter','Latex') ; hold on ;
subplot(2,2,1) ; hold on ;
plot(order,expeestls(:,1),'--k','LineWidth',1) ;
plot(order,expevalls(:,1),'-.k','LineWidth',1) ;
plot(order,expeestaic(:,1),'-k','LineWidth',1) ;
axis([0 100 .7 1.2]) ; xlabel('Order') ; ylabel('Cost') ;
legend('Estimation','Validation','AIC','Location','southeast') ;

subplot(2,2,3) ; hold on ;
plot(order,expeestls(:,2),'--k','LineWidth',1) ;
plot(order,expevalls(:,2),'-.k','LineWidth',1) ;
plot(order,expeestaic(:,2),'-k','LineWidth',1) ;
axis([0 100 .7 1.2]) ; xlabel('Order') ; ylabel('Cost') ;
legend('Estimation','Validation','AIC','Location','southeast') ;

subplot(2,2,2) ; hold on ;
plot(order,costvalls(:,1),'-k','LineWidth',1) ;
axis([0 100 0 1]) ; xlabel('Order') ; ylabel('Normalized RMS') ;
legend('V_0') ;

subplot(2,2,4) ; hold on ;
plot(order,costvalls(:,2),'-k','LineWidth',1) ;
axis([0 100 0 1]) ; xlabel('Order') ; ylabel('Normalized RMS') ;
legend('V_0') ;

hold off ;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function cost = cost_ls(y, u, g)

N= size(u,1) ; yhat = filter(g,1,u) ;
cost = 1/N*sum((y-yhat).^2) ;
end
```