A Comparison of Some Approaches to Time-delay Estimation

Mats Isaksson

Department of Automatic Control
Lund Institute of Technology
June 1997
A Comparison of Some Approaches to Time-delay Estimation (Jämförelse av några metoder för skattning av tidstördräjning).

Abstract

The use of time-delay in modeling input-output systems is quite common in practise and knowing a close estimate of the system time-delay is usually a very important feature for control. A number of attempts of dealing with systems with time-delays are reported in the literature. In this report some of these methods are implemented and studied. To be able to test and compare different approaches for time-delay estimation a piece of software has been implemented. The software creates a graphical, mouse-controlled, user-friendly environment where it is easy to test a new algorithm with a lot of different systems, sampling periods and disturbances. It also gives the user an opportunity to get an idea about how a new algorithm works compared to other existing algorithms. This report also includes the development of a new fault detection approach to time-delay estimation. The new algorithm is designed to make a first experiment on an unknown system. The main advantages of the algorithm is that it only uses a step for input signal and that no prior knowledge of the system is needed. The algorithm has shown to be very robust and usually gives accurate estimates of the time-delay.

Key words

time-delay estimation

Classification system and/or index terms (if any)

Supplementary bibliographical information

The report may be ordered from the Department of Automatic Control or borrowed through:

University Library 2, Box 3, S-221 00 Lund, Sweden
Fax +46 46 2224422  E-mail ub2@uub2.lu.se
Acknowledgments

This report was mainly written in Newcastle, Australia, during a student exchange program between Lund Institute of Technology and the University of Newcastle. I would like to thank my two supervisors involved in this project: Professor Björn Wittenmark at the department of Automatic Control in Lund, Sweden, and Associate Professor Rick Middleton at the Department of Electrical and Computer Engineering in Newcastle, Australia. Special thanks to Rick Middleton, whose enthusiasm and help made my stay in Newcastle a great experience. Thanks also to Leif Andersson, LaTex-Guru at the department of Automatic Control in Lund.
Contents

1. Introduction ........................................... 3
   1.1 Time-delay estimation ............................. 3
   1.2 Some methods of time-delay estimation .......... 3
2. Extended Numerator .................................... 5
   2.1 Introduction ...................................... 5
   2.2 The algorithm ..................................... 5
   2.3 Implementation ................................... 7
   2.4 Simulations ....................................... 7
   2.5 Conclusions ...................................... 9
3. Relaxation Algorithms ................................ 11
   3.1 Introduction ...................................... 11
   3.2 The algorithm ..................................... 11
   3.3 Implementation ................................... 12
   3.4 Simulations ....................................... 14
   3.5 Conclusions ...................................... 15
4. Time-Delay Estimation Using General Orthonormal Bases 17
   4.1 Introduction ...................................... 17
   4.2 General orthonormal bases ......................... 17
   4.3 The Padé approximation ............................ 17
   4.4 The algorithm ..................................... 18
   4.5 Implementation ................................... 19
   4.6 Simulations ....................................... 19
   4.7 Conclusions ...................................... 20
5. A Fault Detection Approach To Time Delay Estimation ... 22
   5.1 Introduction ...................................... 22
   5.2 Review of the Kalman filter ....................... 23
   5.3 Kalman filters used in this algorithm .......... 24
   5.4 The algorithm ..................................... 27
   5.5 Starting new Kalman filters ...................... 27
   5.6 Finding an upper limit of the time-delay ...... 28
   5.7 Pruning ............................................ 31
   5.8 Finding the time-delay ............................ 33
   5.9 Simulations ....................................... 35
   5.10 Conclusions ...................................... 37
6. Software For Time-delay Estimation .................... 39
   6.1 Introduction ...................................... 39
   6.2 Background ........................................ 39
   6.3 Software .......................................... 39
   6.4 Getting started ................................... 39
A. Example systems ..................................... 45
B. Bibliography ......................................... 46
1. Introduction

1.1 Time-delay estimation

This report is mainly a survey of different approaches to time-delay estimation. To be able to test and compare some of these different approaches a piece of software has been implemented. The software is described in Chapter 6.

The use of time-delay in modeling input-output systems is quite common in practise, and knowing an accurate estimate of the system time-delay is usually a very important feature for control. Due to the close relationship between the phase lag introduced by the delay and the stability of the system, most control algorithms are very sensitive to knowledge of the time-delay. In [10] it is shown that the closed loop may be unstable if the model time-delay does not coincide with the process time-delay.

In the context of system identification, having methods of making a crude first stage identification of a system is often essential to choose a relevant model structure and input signal for more thorough identification. What is usually needed for these more advanced identification techniques is some preliminary knowledge of system time-delay and dominating time-constant. Knowing an estimate of these quantities makes it possible to choose an input with its energy concentrated in the frequencies of interest (period of input signal = dominating time constant+time-delay). In some identification methods, for example methods using orthonormal bases, the preliminary information gained is also important when choosing the model structure for identification.

Like most non-linear problems time-delay estimation is not trivial. Many identification algorithms use a rational model and approximate the time-delay with a rational function. To be able to estimate the time-delay, which is one parameter, a number of additional parameters have to be decided. Most algorithms also need some a priori knowledge about the size of the time-delay and give a significant drop of performance if that information is not available. In many cases the time-delay is also time-varying which complicates the task even more. Finding a fast and accurate on-line algorithm for time-delay estimation is therefore a very interesting and important problem.

1.2 Some methods of time-delay estimation

A number of attempts of dealing with systems with time-delays are reported in the literature. A frequently used method for dealing with unknown delays has been to use a shift-operator model and expand the numerator polynomial. An example of this approach, found in [6], is discussed in Chapter 2. The algorithm is also implemented in the software package.

Another common approach is to use some kind of relaxation algorithm. That is, the process is divided into two parts. First the process parameters are estimated, assuming the time-delay is known and then the time-delay is estimated, assuming the last estimated process parameters are correct. An example of this method is presented in [4] and discussed in Chapter 3. A variant of that algorithm is implemented in the software package. Another
relaxation approach for discrete-time systems is described in [5], where the algorithm changes the estimated time-delay by up to one sampling period at each step. This is based on a parallel computation of three error terms, corresponding to a step of +1, 0 or −1 sampling period. In Chapter 4 an attempt to estimate the system time-delay by using the general orthonormal bases, found in [8], is made. For continuous-time systems the most common approach is to expand the Laplace transform expression for the time-delay, \( e^{-sT} \), to a rational approximation. In [9] Padé approximations are used.

In Chapter 5 a new fault detection approach to time-delay estimation is presented. The algorithm is designed to make a first step experiment on an unknown system. The main advantages of the algorithm are that it only uses a step for input signal and that no prior knowledge of the system is needed. The algorithm has shown to be very robust and usually gives accurate estimates of the time-delay. This algorithm is also implemented in the software package.

A good survey of time-delay estimation methods is found in [3], where a rational approximation is proposed.
2. Extended Numerator

2.1 Introduction

A common approach to time-delay estimation in the discrete-time case is based on knowledge of an upper bound on the time-delay. The numerator of the system transfer-function is expanded by the given upper bound and the resulting larger system model is then used for identification. The time-delay is decided from the number of parameters in the estimated numerator, which can be considered small in comparison to the succeeding parameters of this polynomial. The following is a variant of this method, which is presented in [6].

2.2 The algorithm

It is assumed that the process and noise signal can be described by a linear difference equation of order $n_a$ with constant parameters.

$$y(k) = -a_1 y(k-1) - \ldots - a_{n_a} y(k-n_a)$$
$$+ b_1 u(k-d-1) + \ldots + b_{n_b} u(k-d-n_b)$$
$$+ c_1 v(k-1) + \ldots + c_{n_c} v(k-n_c) \quad (2.1)$$

$u(k)$ and $y(k)$ are the sampled process input and output signal and $v(k)$ is discrete-time white noise. The sampling period is $h$ and $k = t/h$ is the discrete-time. The continuous process time-delay is $T$ which gives the discrete process time-delay $d = \text{int}(T/h)$, where $d$ is a non-negative integer. The $z$-transformation of (2.1) results in

$$y(k) = \frac{B(z^{-1})}{A(z^{-1})} z^{-d} u(k) + \frac{C(z^{-1})}{A(z^{-1})} v(k) \quad (2.2)$$

where

$$A(z^{-1}) = 1 + a_1 z^{-1} + \ldots + a_{n_a} z^{-n_a} \quad (2.3)$$
$$B(z^{-1}) = b_1 z^{-1} + \ldots + b_{n_b} z^{-n_b} \quad (2.4)$$
$$C(z^{-1}) = 1 + c_1 z^{-1} + \ldots + c_{n_c} z^{-n_c} \quad (2.5)$$

Eq. (2.2) can be written in the modified structure

$$y(k) = \frac{B^*(z^{-1})}{A(z^{-1})} u(k) + \frac{C(z^{-1})}{A(z^{-1})} v(k) \quad (2.6)$$

where

$$B^*(z^{-1}) = b_1^* z^{-1} + \ldots + b_{n_b+d_{max}}^* z^{-(n_b+d_{max})} \quad (2.7)$$
The value $d_{\text{max}}$ is the upper limit of the process time-delay and has to be assumed a priori. Comparison of $B(z^{-1})z^{-d}$ with $B^*(z^{-1})$ shows that

\begin{align}
    b^*_i &= 0 \quad i = 1, \ldots, d \\
    b^*_j &= b_{j-d} \quad j = d+1, \ldots, d+n_b \\
    b^*_l &= 0 \quad l = d+n_b+1, \ldots, d_{\text{max}}+n_b
\end{align}

Using a standard Recursive Least Squares algorithm with the model (2.6) gives the regressor and parameter-vector

\begin{align}
    \Phi(k) &= [-y(k-1) \ldots -y(k-\hat{n}_a) u(k-1) \ldots u(k-\hat{n}_b - d_{\text{max}})]^T \\
    \hat{\Theta} &= [\hat{a}_1 \ldots \hat{a}_{\hat{n}_a} \hat{b}^*_1 \ldots \hat{b}^*_{\hat{n}_b + d_{\text{max}}}]^T
\end{align}

It is well known that $\hat{n}_a = n_a$, $\hat{n}_b = n_b$, $\hat{d} = d$ and $C(z^{-1}) = 1$ are necessary conditions for consistent parameter estimation for the Least Squares method. If these conditions are fulfilled, the parameter estimates

$$
\hat{b}^*_i, \quad i = 1, \ldots, d
$$

and

$$
\hat{b}^*_l, \quad l = n_b + d + 1, \ldots, n_b + d_{\text{max}}
$$

converge to zero when $k$ goes to infinity. For finite $k$ these parameters will be small in comparison to the parameters

$$
\hat{b}^*_j, \quad j = 1 + d, \ldots, n_b + d
$$

The idea is then to estimate the model time-delay, $\hat{d}$, by looking at the parameters of the polynomial $B^*(z^{-1})$. The time-delay estimate is the number of parameters of $B^*(z^{-1})$, which can be considered small in comparison to the succeeding parameters of this polynomial. This procedure is not always easy, therefore the authors of [6] propose the following method of finding $\hat{d}$ from $B^*(z^{-1})$.

Step 1:

Determine the maximal parameter $\hat{b}^*_{d_{\text{max}}}$ of $B^*(z^{-1})$, i.e.

$$
\hat{b}^*_{d_{\text{max}}} = \max \{ |\hat{b}^*_i|, \quad i = 1, \ldots, n_b + d_{\text{max}} \}
$$

Then $0 \leq \hat{d} \leq d'_{\text{max}} - 1$

Step 2:

Calculate the error functions

$$
P(\tilde{d}) = \sum_{k=1}^{N} \Delta y^2_d(k), \quad \tilde{d} = 0, \ldots, d'_{\text{max}} - 1
$$
where

\[ \Delta g_d(k) = \hat{g}^*(k) - \hat{g}_d(k) \]  

(2.15)

\( \hat{g}^*(k) \) is the pulse response of the model \( \hat{G}^*(z^{-1}) \) and \( \hat{g}_d(k) \) is the pulse response of the model

\[ \hat{G}_d(z^{-1}) = \frac{\hat{B}_d(z^{-1})}{\hat{A}(z^{-1})}, z^{-\bar{d}} \]  

(2.16)

The integer \( N \) should be larger than \( T_{95}/h \), where \( T_{95} \) is the 95 percent settling time of the process transient function. \( \hat{G}^*(z^{-1}) \) and \( \hat{G}_d(z^{-1}) \) have the same denominator polynomial. The numerator polynomial \( \hat{B}_d(z^{-1}) \) only influences the values \( \hat{g}_d(k), k = 1 + \bar{d}, \ldots, n_b + \bar{d} \). Therefore \( \hat{B}_d(z^{-1}) \) is assumed such that \( \hat{g}_d(k), k = 1 + \bar{d}, \ldots, n_b + \bar{d} \) is equal to \( \hat{g}^*(k), k = 1 + \bar{d}, \ldots, n_b + \bar{d} \). It is shown in [1] that the errors \( \Delta g_d(k) \) can be calculated recursively without explicit calculation of \( \hat{g}^*(k) \) and \( \hat{g}_d(k) \).

Step 3:

Determine the minimal value \( F(\bar{d}) \) of \( F(\bar{d}) \), i.e.

\[ F(\bar{d}) = \min\{F(\bar{d}) \mid \bar{d} = 0, \ldots, \bar{d}_{max} - 1\} \]  

(2.17)

Then, \( \bar{d} \) is the estimated discrete model time-delay and \( \bar{T} = \bar{d}h \) the estimated delay for the continuous-time model.

2.3 Implementation

In [6] this algorithm is implemented using the Recursive Least Squares method for the estimation of the parameters. This implementation is a non-recursive version of the algorithm which uses the Maximum Likelihood method for the parameter estimation. Thereafter the steps (2.13), (2.14) and (2.17) are carried out. The upper limit of the time-delay and the size of \( n_a, n_b \) and \( n_c \) in (2.1) are chosen interactively by the user. The value \( N \) in (2.14) is important. It should be larger than \( T_{95}/h \), where \( T_{95} \) is the 95 percent settling time of the process transient function and \( h \) the sampling period. It is very important that the value of \( N \) is not chosen to low, in which case the algorithm works badly. In this implementation the value of \( N \) is set to 25/h, but it should be remembered that if \( T_{95} \) is very large this value should be increased.

2.4 Simulations

In the simulations the values of \( n_a \) and \( n_b \) were set to two. The value of \( n_c \) was set to zero, that is only white noise was expected. The chosen model order is relatively high, so to get a persistently exciting input signal white noise with
spectral density 0.1 was used. In all simulations the eleven example systems in Appendix A, all with a time-delay of 5 seconds, were used. The upper limit of the time-delay was set to 10 seconds. The system disturbances were white noise with spectral densities in the range 0.001 to 0.07. The sampling period was 0.1 seconds and 20 runs were made for each noise-level. The length of each simulation was 40 seconds.

Table 2.1 shows the absolute values of the errors (mean value over 20 runs) between estimated time-delay and real time-delay and Table 2.2 shows how often, in percentage, the algorithm manages to estimate a time-delay that is less than five samples (0.5 seconds) too big or too small.

The algorithm works very well for the systems 1, 2 and 3. For noise spectral densities below 0.01 the algorithm finds the exact time-delay in all the simulations. For low noise-levels it is often enough to use the first step, i.e. (2.13), to find the time-delay. For higher noise-levels the steps (2.14) and (2.17) improves the results from the upper limit given by (2.13). The best results are for system 3, where the algorithm finds the exact time-delay for all noise-levels.

The results for system 5 are also very good. System 5 is of second order with two poles in $-10$. System 4 is also of second order. It has two poles in $-1$ and is therefore much slower than system 5. As can be seen, the results for this system are really bad. The upper limit, found by the Maximum Likelihood method, is too far from the true time-delay, and the extra steps, (2.14) and (2.17), do not improve the result. To improve the estimates for this system, the spectral density of the input signal has to be increased. If it is increased with a factor ten, the results become almost as good as for system 5.

The results for system 6 are also poor. System 6 is a slow second order system ($\omega = 0.2, \xi = 0.5$). It does not help to increase the length of the simulation or to change the model order. To get good estimates of the time-delay, the spectral density of the input signal has to be very high. System 7 is also a second order system, with the same relative damping as system 6, but about five times faster ($\omega = 1, \xi = 0.5$). The results are slightly better than for system 6, but still very poor. For the badly damped system 8, the results for the lowest noise level are good, but for higher noise-levels the performance drops. The only way to improve the results for systems 7 and 8 is to increase the spectral density of the input signal.

System 9 is a non-minimum-phase system with one zero in 1.4 and three poles in $-1$. Also here the results are poor. If the noise spectral density of the input signal is increased to one the estimated upper limit, given by (2.13), gets really close to the true value, but then the extra steps (2.14) and (2.17), destroy the good results. In this case it clearly would be preferable not to use the extra steps in the algorithm. System 10 is also a non-minimum-phase system, but here the results are very good.

System 11 is of order eight and the algorithm works poorly for this system. The results in Table 2.1 can be misinterpreted. This system behaves approximately like a first order system with an extra time-delay, so if the results showed a large constant overestimation of the delay it would indicate that the algorithm worked reasonably well. A quick look in Table 2.1 might also give that impression. However, that is not the case. What the table shows is a mean value of 20 absolute values between true and estimated time-delay. The time-delay is actually mostly underestimated, mixed with some very large overestimations. To see that things are not working well the values in the parenthesis can be studied. Those values show the standard deviation of the twenty estimates for each noise level, and the very high values show that the
algorithm is not working well.

The results from the simulation show that the time-delay estimates are completely dependent on how well the parameter-estimation algorithm works. The two extra steps, (2.14) and (2.17), can improve the result. But only if the parameter estimation algorithm works reasonably well. Even if the extra steps do not give major improvements of the results, they most often improve the time-delay estimate with a few samples. Another important thing is that they have never made the results worse, with exception for the non-minimum-phase system 9, where it clearly would be preferable not to use the extra steps.

2.5 Conclusions

A lot of different implementations of the algorithm are possible. Any identification method capable of identifying the parameters of the model (2.1) can be used. It is also possible with other modifications of the algorithm. For example, to include a lower limit, \( d_{\text{min}} > 0 \), of the process time-delay in the model. Below are some pluses and minuses about the algorithm:

+ The algorithm can be implemented in recursive form and can therefore be used for processes with time-varying time-delay and in adaptive algorithms.

+ If the input signal is persistently exciting this algorithm gives very good results.

− The number of estimated parameters and hence the computational effort increases rapidly with shorter sampling period.

− An upper bound of the time-delay has to be known a priori.

− The convergence of parameter estimation algorithms is usually poor.
Table 2.1  Simulations showing the absolute value of the error between estimated time-delay and real time-delay. The value is a mean value over 20 runs. The value in the parenthesis is the standard-deviation of 20 estimates.

<table>
<thead>
<tr>
<th>System</th>
<th>Sp. den. =0.001</th>
<th>Sp. den. =0.005</th>
<th>Sp. den. =0.01</th>
<th>Sp. den. =0.03</th>
<th>Sp. den. =0.05</th>
<th>Sp. den. =0.07</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.27(0.80)</td>
<td>0.98(1.58)</td>
<td>2.16(2.37)</td>
</tr>
<tr>
<td>2</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.08(0.11)</td>
<td>0.60(1.41)</td>
<td>1.41(2.14)</td>
</tr>
<tr>
<td>3</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
</tr>
<tr>
<td>4</td>
<td>0.63(0.20)</td>
<td>0.91(0.19)</td>
<td>0.95(0.34)</td>
<td>1.74(1.78)</td>
<td>2.05(2.47)</td>
<td>2.22(2.64)</td>
</tr>
<tr>
<td>5</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.01(0.02)</td>
<td>0.01(0.03)</td>
<td>0.02(0.04)</td>
</tr>
<tr>
<td>6</td>
<td>1.55(2.01)</td>
<td>2.13(2.53)</td>
<td>2.89(3.20)</td>
<td>2.63(3.02)</td>
<td>2.69(3.13)</td>
<td>2.18(2.66)</td>
</tr>
<tr>
<td>7</td>
<td>0.36(0.07)</td>
<td>1.01(0.24)</td>
<td>1.06(0.27)</td>
<td>1.34(0.62)</td>
<td>1.48(1.81)</td>
<td>2.11(2.53)</td>
</tr>
<tr>
<td>8</td>
<td>0.07(0.08)</td>
<td>1.14(1.56)</td>
<td>1.63(1.92)</td>
<td>1.45(1.40)</td>
<td>2.13(2.50)</td>
<td>1.67(1.78)</td>
</tr>
<tr>
<td>9</td>
<td>1.42(1.12)</td>
<td>2.38(0.35)</td>
<td>2.39(0.49)</td>
<td>2.29(1.99)</td>
<td>2.76(2.84)</td>
<td>2.13(2.66)</td>
</tr>
<tr>
<td>10</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.00(0.00)</td>
<td>0.03(0.09)</td>
<td>0.00(0.00)</td>
<td>0.05(0.12)</td>
</tr>
<tr>
<td>11</td>
<td>2.75(2.63)</td>
<td>2.26(2.73)</td>
<td>2.37(2.75)</td>
<td>2.98(3.34)</td>
<td>3.07(3.45)</td>
<td>2.34(2.87)</td>
</tr>
</tbody>
</table>

Table 2.2  Simulations showing how often, in percentage, the algorithm manages to estimate a time-delay that is less than five samples (0.5 seconds) too big or too small.

<table>
<thead>
<tr>
<th>System</th>
<th>Sp. den. =0.001</th>
<th>Sp. den. =0.005</th>
<th>Sp. den. =0.01</th>
<th>Sp. den. =0.03</th>
<th>Sp. den. =0.05</th>
<th>Sp. den. =0.07</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>90</td>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>80</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>15</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>90</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>35</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>
3. Relaxation Algorithms

3.1 Introduction

In [2] a method for recursive estimation of both process parameters and time-delay is presented. The algorithm is a relaxation algorithm, that is the process model is divided into two parts, which are estimated separately. At every sample two steps are carried out. First the model parameters are estimated, assuming that the last estimated time-delay was correct, and then the time-delay is estimated, assuming the latest estimated parameters were correct. For the algorithm to work an upper limit of the time-delay, \( d_{\text{max}} \), has to be known in advance.

3.2 The algorithm

The idea is to use the standard model

\[
y(k) = \Phi(k)^T \Theta + e(k)
\]

(3.1)

where the parameter vector, \( \Theta \), is a function of the parameters while the regressor, \( \Phi \), is a function of the time-delay. That is

\[
\Phi(k, d) = [-y(k-1) \ldots -y(k-n) \ u(k-d-1) \ldots u(k-d-m)]^T
\]

(3.2)

\[
\Theta = [a_1 \ a_2 \ldots a_n \ b_1 \ b_2 \ldots b_m]^T
\]

(3.3)

\( d \)=system time-delay
\( n \)=number of poles
\( m \)=number of zeros

The estimated parameters and time-delay are \( \hat{\Theta} \) and \( \hat{d} \) respectively and the predicted output is

\[
\hat{y} = \hat{\Phi}^T \hat{\Theta}
\]

(3.4)

where

\[
\hat{\Phi} = [-y(k-1) \ldots -y(k-n) \ u(k-\hat{d}-1) \ldots u(k-\hat{d}-m)]^T
\]

(3.5)

\[
\hat{\Theta} = [\hat{a}_1 \ \hat{a}_2 \ldots \hat{a}_n \ \hat{b}_1 \ \hat{b}_2 \ldots \hat{b}_m]^T
\]

(3.6)

The performance index is chosen to be

\[
J = \sum_{i=0}^{t} e_i^2
\]

(3.7)
where $\epsilon_i$ is the prediction error, defined as

$$
\epsilon_i = y_i - \hat{y}_i = y_i - \hat{\Phi}(i)^T \hat{\Theta}
$$

(3.8)

Minimizing with respect to the parameters and the time-delay gives

$$
\frac{\partial J}{\partial \hat{\Theta}} = 0
$$

(3.9)

$$
J = \min[ J(\hat{d}) ] \quad \forall \hat{d} \in [0, d_{max}]
$$

(3.10)

The delay-estimation must be obtained by simultaneous solution to (3.9) and (3.10).

### 3.3 Implementation

The algorithm has been implemented using the Recursive Least Squares algorithm, but can as well be implemented using any other recursive estimation algorithm (Instrumental Variable, Maximum Likelihood, … etc.). At every sample two steps have to be carried out. The first step (Eqs. 3.11 – 3.14) is used to estimate the parameters, assuming the last estimated time-delay is correct. The next step (Eqs. 3.15 – 3.16) is to estimate the time-delay, assuming the last estimated parameters are correct.

$$
\epsilon(k) = y(k) - \Phi(k, \hat{d})^T \hat{\Theta}(k - 1)
$$

(3.11)

$$
K(k) = \frac{P(k - 1) \Phi(k, \hat{d})}{1 + \Phi(k, \hat{d})^T P(k - 1) \Phi(k, \hat{d})}
$$

(3.12)

$$
\hat{\Theta}(k) = \hat{\Theta}(k - 1) + K(k)\epsilon(k)
$$

(3.13)

$$
P(k) = P(k - 1) - \frac{P(k - 1) \Phi(k, \hat{d}) \Phi(k, \hat{d})^T P(k - 1)}{1 + \Phi(k, \hat{d})^T P(k - 1) \Phi(k, \hat{d})}
$$

(3.14)

$$
J(k, d) = J(k - 1, \hat{d}) + (y(k) - \Phi(k, d)^T \hat{\Theta}(k))^2 \quad \forall d \in [0, d_{max}]
$$

(3.15)

$$
J(k, \hat{d}) = \min[ J(k, d) ] \quad \forall d \in [0, d_{max}]
$$

(3.16)

In the implementation a first order model (3.17) has been used. The parameter vector then becomes (3.18) and the variable regressor (3.19). The correction factor becomes (3.22), the estimated parameters (3.21) and the covariance matrix (3.24).
\[ y(k) = -a_1 y(k - 1) + b_1 u(k - 1 - d) \quad (3.17) \]

\[ \Theta = \begin{bmatrix} a_1 & b_1 \end{bmatrix}^T \quad (3.18) \]

\[ \Phi(k, \hat{d}) = [-y(k - 1) \ u(k - 1 - \hat{d})]^T \quad (3.19) \]

\[ \epsilon(k) = y(k) - (-\hat{a}_1(k - 1)y(k - 1) + \hat{b}_1(k - 1)u(k - 1 - \hat{d})) \quad (3.20) \]

\[ \hat{\Theta}(k) = \begin{pmatrix} \hat{a}_1(k) \\ \hat{b}_1(k) \end{pmatrix} = \begin{pmatrix} \hat{a}_1(k - 1) \\ \hat{b}_1(k - 1) \end{pmatrix} + \begin{pmatrix} k_{11}(k) \\ k_{21}(k) \end{pmatrix} \epsilon(k) \quad (3.21) \]

\[ K(k) = \begin{pmatrix} k_{11}(k) \\ k_{21}(k) \end{pmatrix} = \frac{P(k - 1)\Phi(k, \hat{d})}{1 + \Phi(k, \hat{d})^T P(k - 1)\Phi(k, \hat{d})} \quad (3.22) \]

where

\[ k_{11}(k) = \frac{-p_{11}(k - 1)y(k - 1) + p_{12}(k - 1)u(k - 1 - \hat{d})}{1 + p_{11}(k - 1)y(k - 1)^2 - (p_{12}(k - 1) + p_{21}(k - 1))u(k - 1 - \hat{d}) + p_{22}(k - 1)u(k - 1 - \hat{d})^2} \]

\[ k_{21}(k) = \frac{-p_{21}(k - 1)y(k - 1) + p_{22}(k - 1)u(k - 1 - \hat{d})}{1 + p_{11}(k - 1)y(k - 1)^2 - (p_{12}(k - 1) + p_{21}(k - 1))u(k - 1 - \hat{d}) + p_{22}(k - 1)u(k - 1 - \hat{d})^2} \]

\[ (3.23) \]

\[ P(k) = \begin{pmatrix} p_{11}(k) & p_{12}(k) \\ p_{21}(k) & p_{22}(k) \end{pmatrix} = P(k - 1) - \frac{P(k - 1)\Phi(k, \hat{d})\Phi(k, \hat{d})^T P(k - 1)}{1 + \Phi(k, \hat{d})^T P(k - 1)\Phi(k, \hat{d})} \quad (3.24) \]

where

\[ p_{11}(k) = \frac{p_{11}(k - 1) + (p_{11}(k - 1)p_{22}(k - 1) - p_{12}(k - 1)p_{21}(k - 1))u(k - 1 - \hat{d})^2}{1 + p_{11}(k - 1)y(k - 1)^2 - (p_{12}(k - 1) + p_{21}(k - 1))u(k - 1 - \hat{d}) + p_{22}(k - 1)u(k - 1 - \hat{d})^2} \]

\[ p_{12}(k) = \frac{p_{12}(k - 1) + (p_{11}(k - 1)p_{22}(k - 1) - p_{12}(k - 1)p_{21}(k - 1))u(k - 1 - \hat{d})y(k - 1)}{1 + p_{11}(k - 1)y(k - 1)^2 - (p_{12}(k - 1) + p_{21}(k - 1))u(k - 1 - \hat{d}) + p_{22}(k - 1)u(k - 1 - \hat{d})^2} \]

\[ p_{21}(k) = \frac{p_{21}(k - 1) + (p_{11}(k - 1)p_{22}(k - 1) - p_{12}(k - 1)p_{21}(k - 1))u(k - 1 - \hat{d})}{1 + p_{11}(k - 1)y(k - 1)^2 - (p_{12}(k - 1) + p_{21}(k - 1))u(k - 1 - \hat{d}) + p_{22}(k - 1)u(k - 1 - \hat{d})^2} \]

\[ p_{22}(k) = \frac{p_{22}(k - 1) + (p_{11}(k - 1)p_{22}(k - 1) - p_{12}(k - 1)p_{21}(k - 1))y(k - 1)^2}{1 + p_{11}(k - 1)y(k - 1)^2 - (p_{12}(k - 1) + p_{21}(k - 1))u(k - 1 - \hat{d}) + p_{22}(k - 1)u(k - 1 - \hat{d})^2} \]

\[ (3.25) \]

The initial values of the model parameters and the time-delay are set to zero.
3.4 Simulations

This algorithm has shown to be very sensitive to the choice of input signal. For the method to work well the input signal should be one that is constant at least as long as the sum of the real system time-constant plus time-delay. Choosing an input that changes more frequently than that gives really bad results. Using for example white noise for input signal does not work.

In the simulations the eleven example systems in Appendix A, all with a time-delay of 1 second, were used. The a priori known upper limit of the time-delay was set to five seconds. The input signal was a square-wave with period 20 seconds and the length of each simulation was 40 seconds. The disturbances acting on the system were white noise with spectral densities in the range 0.001 to 0.07 and 10 runs were made for each noise-level. The sampling period was 0.1 seconds.

The results from the simulations can be studied in Tables 3.1 and 3.2. Table 3.1 shows the absolute values of the errors (mean value over 10 runs) between estimated time-delay and real time-delay and Table 3.2 shows how often, in percentage, the algorithm manages to estimate a time-delay that is less than five samples (0.5 seconds) too big or too small.

The results show that if good a priori knowledge of the system time-delay is available, this algorithm can work very well. System 1 is a pure integrator with time-delay and the algorithm gives good results for that case. System 2 and 3 are two first order systems and the results are very good. The results are better for system 3 than for system 2, depending on that system 3 is much faster, which means that the system has longer time to settle to the input before it changes again. The same thing can be seen if the results for systems 4 and 5 are compared. The results for system 2 and system 4 improve significantly if a larger period of the input signal is chosen, which shows that the choice of input signal is very important. The results for the second order systems 4 and 5 are reasonably good, even though the model is of first order only. Even though the results probably can be improved with a second order model, the algorithm still works with a first order model. If the true system has more pronounced second order characteristic, a first order model is not enough. The systems 6 and 7 are two second order systems. The only difference between them is that system 6 is much slower. The results for system 6 are really bad and do not improve with a larger period of the input signal. The results for system 7 are not as bad, but also here the results are not improved if the period of the input signal is increased. To get good results for these systems, a higher order model is needed. System 8 is a very badly damped second order system and the results show that also in that case a first order model is not enough. Systems 9 and 10 are non-minimum-phase systems of third order. The algorithm has problems with system 9 but handles system 10 reasonably well. System 8 is of order eight and, not very surprisingly, the algorithm does not work on that system.

If the a priori known upper limit of the time-delay is increased to 20 seconds the performance drops for all the systems and the algorithm also gets a lot slower. To know a reasonable upper limit of the time-delay is therefore very important. A summation of the results from the simulations is that the characteristics of the algorithm are essentially those of the parameter estimation algorithm used. When good parameters can be estimated, so can the time-delay. To be able to understand the algorithm better a higher order model should be tried. This might show other problems with the algorithm, because
a higher model order would need a more exciting input signal and already with a first order model the choice of input signal is hard. What is clear from the simulations is that the choice of the input signal is very important, something the authors of [2] did not mention.

3.5 Conclusions

A lot of variations in the implementation of the algorithm are possible. For example, in this implementation a standard Recursive Least Squares algorithm is used for the identification of the parameters in the rational model, but any other recursive estimation algorithm (Instrumental Variable, Maximum Likelihood, ... etc.) could be used. In this implementation a first order rational model is used, but if a priori information of a good model size is known it is of course possible to use a model of higher order. It is also possible with other modifications. For example to put some restrictions on how much the estimated time-delay is allowed to change every sample, or to use a known lower limit of the delay, \( d_{\text{min}} \), in the model.

This algorithm can be very good if the user has a reasonably good preliminary estimate of the time-delay. And for the algorithm to work really well it is useful to have some idea of the system time-constant and a reasonable size of the rational model. The method is definitely not good for doing a first stage experiment on a system. Below are some pluses and minuses about the algorithm.

+ The time-delay estimation is recursive and the code can easily be added to any recursive identification algorithm.

+ Because of the recursive character of the method it can be used for identification of systems where the delay is time-varying, and in adaptive algorithms.

– In the present implementation the algorithm does not stop when the time-delay is found. The user has to decide the length of the experiment in advance.

– An upper limit of the system time-delay has to be decided before the identification experiment is started. If that upper limit is chosen too low the real time-delay can not be found and if it is chosen too high the performance of the algorithm decreases dramatically.

– The algorithm is very sensitive to the choice of input signal.

– Compared to other implemented algorithms this one is relatively slow. Especially if no good estimate of the time-delay is known a priori.
Table 3.1  Simulations showing the absolute value of the error between estimated time-delay and real time-delay. The value is a mean value over 10 runs. The value in the parenthesis is the standard-deviation of 10 estimates.

<table>
<thead>
<tr>
<th>System</th>
<th>Sp. den. =0.001</th>
<th>Sp. den. =0.005</th>
<th>Sp. den. =0.01</th>
<th>Sp. den. =0.03</th>
<th>Sp. den. =0.05</th>
<th>Sp. den. =0.07</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00(0.00)</td>
<td>0.10(0.03)</td>
<td>0.06(0.07)</td>
<td>0.56(0.40)</td>
<td>0.77(0.20)</td>
<td>0.72(0.41)</td>
</tr>
<tr>
<td>2</td>
<td>0.00(0.00)</td>
<td>0.06(0.07)</td>
<td>0.04(0.07)</td>
<td>0.38(0.18)</td>
<td>0.61(0.28)</td>
<td>0.59(0.32)</td>
</tr>
<tr>
<td>3</td>
<td>0.00(0.00)</td>
<td>0.01(0.03)</td>
<td>0.02(0.04)</td>
<td>0.01(0.03)</td>
<td>0.06(0.12)</td>
<td>0.04(0.07)</td>
</tr>
<tr>
<td>4</td>
<td>0.16(0.08)</td>
<td>0.19(0.22)</td>
<td>0.40(0.80)</td>
<td>1.34(0.29)</td>
<td>1.59(0.23)</td>
<td>1.75(0.37)</td>
</tr>
<tr>
<td>5</td>
<td>0.00(0.00)</td>
<td>0.01(0.03)</td>
<td>0.04(0.05)</td>
<td>0.06(0.05)</td>
<td>0.10(0.11)</td>
<td>0.15(0.10)</td>
</tr>
<tr>
<td>6</td>
<td>0.76(0.14)</td>
<td>0.84(0.98)</td>
<td>1.56(1.43)</td>
<td>3.94(0.16)</td>
<td>4.07(0.07)</td>
<td>4.04(0.11)</td>
</tr>
<tr>
<td>7</td>
<td>0.16(0.08)</td>
<td>0.19(0.22)</td>
<td>0.12(0.15)</td>
<td>0.94(0.19)</td>
<td>1.15(0.29)</td>
<td>1.15(0.35)</td>
</tr>
<tr>
<td>8</td>
<td>2.99(0.07)</td>
<td>0.37(0.13)</td>
<td>0.42(0.35)</td>
<td>0.44(0.21)</td>
<td>0.72(0.21)</td>
<td>0.86(0.08)</td>
</tr>
<tr>
<td>9</td>
<td>0.78(0.23)</td>
<td>0.65(0.66)</td>
<td>1.07(1.02)</td>
<td>2.71(0.34)</td>
<td>3.22(0.36)</td>
<td>3.27(0.50)</td>
</tr>
<tr>
<td>10</td>
<td>0.30(0.00)</td>
<td>0.30(0.00)</td>
<td>0.31(0.03)</td>
<td>0.35(0.35)</td>
<td>1.45(0.23)</td>
<td>1.38(0.33)</td>
</tr>
<tr>
<td>11</td>
<td>1.50(0.39)</td>
<td>1.25(2.50)</td>
<td>3.26(2.04)</td>
<td>4.00(0.00)</td>
<td>4.00(0.00)</td>
<td>3.72(1.59)</td>
</tr>
</tbody>
</table>

Table 3.2  Simulations showing how often, in percentage, the algorithm manages to estimate a time-delay that is less than five samples (0.5 seconds) too big or too small.

<table>
<thead>
<tr>
<th>System</th>
<th>Sp. den. =0.001</th>
<th>Sp. den. =0.005</th>
<th>Sp. den. =0.01</th>
<th>Sp. den. =0.03</th>
<th>Sp. den. =0.05</th>
<th>Sp. den. =0.07</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>40</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>50</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>100</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>20</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>80</td>
<td>60</td>
<td>40</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
4. Time-Delay Estimation
Using General Orthonormal Bases

4.1 Introduction

In this chapter an attempt of using general orthonormal bases for time-delay estimation is described. The time-delay is approximated using a Padé approximation. Knowing that all zeros in a Padé approximation are non-minimum-phase, and supposing that the system is minimum-phase, all identified non-minimum-phase zeros should correspond to the time-delay. The time-delay is found in four distinct steps. First, the orthonormal bases for the experiment are chosen by the user. Second, those bases are used for identification of the system. Thereafter the non-minimum-phase zeros of the identified system are decided, and finally those zeros are used to calculate the time-delay.

4.2 General orthonormal bases

In [8], the unifying construction of orthonormal bases for system identification (4.1) is presented.

$$B_n(q) = \left( \frac{\sqrt{1 - |\xi_n|^2}}{q - \xi_n} \right) \prod_{k=0}^{n-1} \frac{1 - \xi_k q}{q - \xi_k}$$  (4.1)

This construction preserves orthonormality and allows prior information about a variety of modes at \{\xi_0, \xi_1, \ldots, \xi_{p-1}\} to be incorporated in the model. Furthermore, the construction provides a unifying formulation of all known system identification orthonormal systems, including the well known FIR, Laguerre and Kautz models.

4.3 The Padé approximation

The so called Padé approximation is one method of expressing a time-delay as a rational transfer-function. It is found by choosing the parameters of the expression

$$H_n(s) = \frac{b_0 + b_1 s + \ldots + b_n s^n}{a_0 + a_1 s + \ldots + a_n s^n}$$  (4.2)

so that the first $2n$ derivatives of the expression $H_n(s)$ match those of $e^{-sT}$ at
\(s = 0\). The corresponding approximations for \(n = 1, 2, 3\) and 4 are

\[
H_1(s) = \frac{1 - \frac{sT}{2}}{1 + \frac{sT}{2}}
\]

\[
H_2(s) = \frac{1 - \frac{sT}{2} + \frac{s^2T^2}{12}}{1 + \frac{sT}{2} + \frac{s^2T^2}{12}}
\]

\[
H_3(s) = \frac{1 - \frac{sT}{2} + \frac{s^2T^2}{10} - \frac{s^3T^3}{120}}{1 + \frac{sT}{2} + \frac{s^2T^2}{10} + \frac{s^3T^3}{120}}
\]

\[
H_4(s) = \frac{1 - \frac{sT}{2} + \frac{3s^2T^2}{28} - \frac{s^3T^3}{84} + \frac{s^4T^4}{1680}}{1 + \frac{sT}{2} + \frac{3s^2T^2}{28} + \frac{s^3T^3}{84} + \frac{s^4T^4}{1680}}
\]

(4.3)

All zeros in a Padé approximation are non-minimum-phase.

### 4.4 The algorithm

Assume that the real system has the transfer-function

\[
G_{system}(s) = G(s) \cdot e^{-sT} = \frac{B(s)}{A(s)} \cdot e^{-sT}
\]

(4.4)

and that \(G(s)\) is minimum-phase. The time-delay part, \(e^{-sT}\), can be approximated using a Padé model,

\[
e^{-sT} \approx H_n(s) = \frac{B_n(s)}{A_n(s)} = \frac{1 - \frac{sT}{2} + \ldots}{1 + \frac{sT}{2} + \ldots}
\]

(4.5)

And the system can be approximated with

\[
G_{system}(s) \approx G(s)H_n(s) = \frac{B(s)B_n(s)}{A(s)A_n(s)} = \frac{\prod_{i=1}^{n} \left(1 - \frac{s}{zi}\right)}{\prod_{i=1}^{n} \left(1 - \frac{s}{zi}\right)} = \frac{1 - s \cdot \sum_{i=1}^{n} \frac{1}{zi} + \ldots}{1 - s \cdot \sum_{i=1}^{n} \frac{1}{zi} + \ldots}
\]

(4.6)

If \(G(s)\) is minimum-phase then all non-minimum-phase zeros in \(G_{system}(s)\) correspond to the Padé approximation of \(e^{-sT}\) and it should be possible to get an idea of the time-delay by looking at the non-minimum-phase zeros of the identified system. That is, the time-delay is decided from:

\[
T = \sum_{NMP} \frac{2}{z_i}
\]

(4.7)
4.5 Implementation

The general orthonormal bases (4.1) are used to identify the system. In this implementation the number of different poles in the orthonormal bases and their position and also the number of orthonormal bases used in the experiment are chosen interactively by the user. The identified system model is

\[ H(q) = \sum_{n=0}^{p-1} \Theta_n B_n(q) = \]

\[ \sum_{n=0}^{p-1} \Theta_n \sqrt{1 - \xi_n^2} \cdot \prod_{k=0}^{n-1} (1 - \xi_k q) \cdot \prod_{k=n}^{p-1} \left( \frac{1}{q - \xi_k} \right) = \]

\[ \frac{\Theta_0 \sqrt{1 - \xi_0^2}}{q - \xi_0} + \ldots + \frac{\Theta_{p-1} \sqrt{1 - \xi_{p-1}^2}}{(q - \xi_0)(q - \xi_1) \ldots (q - \xi_{p-2})(q - \xi_{p-1})} = \]

\[ \frac{\Theta_0 \sqrt{1 - \xi_0^2}}{(q - \xi_0)(q - \xi_1) \ldots (q - \xi_{p-1})} + \ldots + \frac{\Theta_{p-1} \sqrt{1 - \xi_{p-1}^2}}{(q - \xi_0)(q - \xi_1) \ldots (q - \xi_{p-2})} = \]

\[ \frac{1}{\prod_{k=0}^{p-1} (q - \xi_k)} \cdot \sum_{k=0}^{p-1} \left( \Theta_k \sqrt{1 - \xi_k^2} \prod_{i=0}^{k-1} (1 - \xi_i q) \prod_{i=k+1}^{p-1} (q - \xi_i) \right) \]  

(4.8)

That is, the discrete-time zeros are the roots to

\[ B(q) = \sum_{k=0}^{p-1} \left( \Theta_k \sqrt{1 - \xi_k^2} \prod_{i=0}^{k-1} (1 - \xi_i q) \prod_{i=k+1}^{p-1} (q - \xi_i) \right) \]  

(4.9)

The pulse-transfer operator is calculated from (4.8). Thereafter the corresponding continuous-time transfer-function is decided. The time-delay can then be calculated by using the non-minimum-phase zeros of this transfer-function, according to (4.7).

4.6 Simulations

The pole excess of the identified system is one, according to (4.8). Most of the identified discrete-time zeros are gathered around (1, 0) in the z-space. If the sampling rate is increased these zeros move closer and closer to (1, 0).

The original idea was to decide the continuous-time transfer-function from the estimated pulse-transfer operator, and then use the continuous-time non-minimum-phase zeros to decide the time-delay. Simulations have shown that this approach does not work. Due to numerical problems it is hard to decide the continuous-time equivalent of (4.8). If standard MATLAB routines are used, warnings for low numerical stability are displayed, and the resulting continuous-time system is often no longer non-minimum-phase. Most or all non-minimum-phase zeros are lost.

To decide the continuous-time zeros some approximation has to be used. One approximation that has shown to work reasonably well is to do a logarithmic mapping of the discrete-time zeros. If the sampling period is short the continuous-time system will have zeros in

\[ s_i \approx \ln(z_i)/h \]  

(4.10)
where the $x_i$'s are the zeros of the discrete-time system and $h$ the sampling period. If this approximation is used, the time-delay estimate is found by first deciding the discrete-time zeros from (4.9). Thereafter the corresponding continuous-time zeros are decided according to (4.10), and finally the time-delay estimate is found from (4.7).

Figures 4.1 and 4.2 show how the algorithm works on the two example systems $G(s) = \frac{1}{(s+1)^2}$ and $G(s) = \frac{1}{s^2 + s + 1}$. For both systems, the time-delay was set to 5 seconds. In both figures, the left plot shows the step-response for the true and estimated system and the right plot shows the identified discrete-time zeros. In these two simulations the number of orthonormal bases used was 15. Six different poles of the orthonormal bases were used. They were placed in $-0.5$, $-1.0$, $-1.5$, $-2.0$, $-2.5$, $-3.0$. The system input was a square-wave with period 20 seconds and amplitude one. The sampling period was 0.1 seconds and the length of each simulation 50 seconds. The disturbance acting on the systems was white noise with spectral density 0.002. The estimated time-delay was 5.0 for the system $G(s) = \frac{1}{(s+1)^2}$ and 4.92 for the system $G(s) = \frac{1}{s^2 + s + 1}$.

Even though the choice of the poles in the orthonormal bases was not made to optimize the performance for these two systems, the results are good. But to make the algorithm work well on a wider range of systems a more clever way to place the poles of the orthonormal bases must be found.

The approximation (4.10) is more accurate if a short sampling period is used, but one problem with using a very short sampling period is that the estimated discrete-time zeros move closer and closer to $(1, 0)$ in the $z$-plane, which can give numerical problem when trying to find the zeros of (4.9).

### 4.7 Conclusions

The simulations have shown that using general orthonormal bases to estimate the time-delay of a system can work very well. To find an algorithm that gives good results for a wide range of systems, a more clever way to place the poles of the orthonormal bases must be found.

This algorithm uses the estimated continuous-time zeros to decide the time-delay. Maybe a method that calculated the time-delay straight from the estimated discrete-time zeros, without first having to decide the corresponding continuous-time zeros, would work even better.
Figure 4.1 An example of how the algorithm work. The example system was $G(s) = \frac{1}{(s+1)^2}$. The left plot shows the step-response for the true and estimated system and the right plot shows the estimated discrete-time zeros.

Figure 4.2 An example of how the algorithm work. The example system was $G(s) = \frac{1}{s^2+4}$. The left plot shows the step-response for the true and estimated system and the right plot shows the estimated discrete-time zeros.
5. A Fault Detection Approach To Time Delay Estimation

5.1 Introduction

In this chapter a fault detection approach to time-delay estimation using Kalman filters is presented. Two types of Kalman filters are used. They will both be thoroughly described in Section 5.3. For now on, let us call them type one and type two.

The top plot in Fig. 5.1 shows the input signal used in the algorithm. The input is a step that is zero the first few samples of the experiment. During that time, the noise spectral density is estimated. After some time, $T_1$, the input shifts to the higher level. At the same time the Kalman filter of type one is started. This filter models the system output as a constant plus white noise. Only one filter of this type is used and its purpose is to decide an upper limit of the system time-delay and to give initial values to the filters of type two. The upper limit of the system time-delay is found by looking at the residuals from this filter with the system output. As long as the system input has not begun to influence the output these residuals are small. After a time equal to the system time-delay the output is no longer noise only and the residuals start to grow. After some additional time, $T_3$, the residuals are so large that a statistical test can be used to decide that an upper limit of the time-delay has been found.

The bottom plot in Fig. 5.1 shows an example of a real system output. $T_2$ is the system time-delay and $T_3$ the additional time, beyond the time-delay, for the filter of type one to decide that the output has changed. $T_2 + T_3$ is the upper limit of the time-delay found by the filter of type one. The filters of type two have two states. They are designed to adapt to a linear output plus white noise. If the real system was a pure integrator with time-delay $T_2$, a Kalman filter of this type started at a time $T_1 + T_2$ after the experiment was started would be the optimal filter to track the output. A filter started too early or too late would usually give larger residuals with the system output.

What is done in the algorithm is to start a filter of this type at every sample, starting with the sample after the input changed to the higher level. The output from the filter started at a time exactly corresponding to the system time-delay should fit best to the real system output, thereby deciding the time-delay. New filters are started every sample until an upper limit of the system time-delay has been found. After that no more new filters are started. Instead the best one of the filters of this type, still running, is decided. This then indicates, via its starting time, the system time-delay.
5.2 Review of the Kalman filter

Before starting to describe the algorithm in more detail, the Kalman filter is reviewed.

Consider the state-space model

\[
\begin{align*}
    x_{k+1} &= \Phi x_k + \Gamma u_k + v_k \\
    y_k &= C x_k + D u_k + e_k
\end{align*}
\]  

(5.1)

with

\[
\begin{align*}
    E\{v_k\} &= 0 \\
    E\{e_k\} &= 0 \\
    E\{v_k v_t^T\} &= \delta_{kt} R_1 \\
    E\{e_k e_t^T\} &= \delta_{kt} R_2 \\
    P_k &= E\{x_k x_k^T\} \\
    P_0 &= E\{x_0 x_0^T\}
\end{align*}
\]  

(5.2)

Assume the noisy input-output data \(\{y_k\}\) and \(\{u_k\}\) to be the only data available, and that the state \(e_k\) is not available for measurement. The Kalman filter for prediction of \(x\) at time \(k+1\) based on present data at time \(k\) is
\[ \hat{x}_{k+1|k} = \Phi \hat{x}_{k|k-1} + \Gamma u_k + K_k(y_k - C \hat{x}_{k|k-1}) \]
\[ K_k = \Phi P_k \Phi^T (R_2 + C P_k C^T)^{-1} \]
\[ P_{k+1} = \Phi P_k \Phi^T + R_1 - \Phi P_k \Phi^T (R_2 + C P_k C^T)^{-1} C P_k \Phi^T \] (5.3)

which is a recursive equation where the estimates are updated as soon as new input-output data are available. Appropriate choices of initial conditions are
\[ \hat{x}_{1|0} = E\{x_1\} = m_0 \]
\[ P_{1|0} = V\{x_1\} = P_0 \] (5.4)

that is, the mean value and the variance of \( z_1 \) taken over many realizations. In reality it is often not known how \( z_1 \) is distributed over many realizations and an initial value has to be guessed. With a large or small value of \( P_0 \) the uncertainty/certainty about how close the guess is to the real value of \( z_1 \) is marked. If the system (5.1) is time invariant, the covariance matrix, \( P_k \), and the correction factor, \( K_k \), converge towards constant matrices.

### 5.3 Kalman filters used in this algorithm

This section describes the two Kalman filters used in the algorithm.

**Type 1**

This type of Kalman filter only has one state. It models the output of the real system as a constant plus white noise, and the filter output gives small residuals with the system output as long as that is the case. Only one filter of this type is used and it is started at the same time as the input-step changes to the higher level. For this filter the notation \( P_0 \) is used.

The state-space model is
\[ \begin{align*}
  x_{k+1} &= x_k \\
  y_k &= x_k + e_k
\end{align*} \] (5.6)

With the same notations as in the previous page

\[ \begin{align*}
  \Phi &= 1 \\
  \Gamma &= 0 \\
  C &= 1 \\
  D &= 0 \\
  R_1 &= 0 \\
  R_2 &= \sigma_e^2
\end{align*} \] (5.6)

The covariance is
\[ P_{k+1} = \frac{P_k \sigma_e^2}{\sigma_e^2 + P_k} \] (5.7)
That is

\[ P_1 = \frac{P_0 \sigma_e^2}{\sigma_e^2 + P_0} \]

\[ P_2 = \frac{P_1 \sigma_e^2}{\sigma_e^2 + P_1} = \frac{P_0 \sigma_e^2}{\sigma_e^2 + 2P_0} \]

\[ \vdots \]

\[ P_k = \frac{P_0 \sigma_e^2}{\sigma_e^2 + kP_0} \] (5.8)

The correction-factor is

\[ K_k = \frac{P_k}{\sigma_e^2 + P_k} = \frac{P_0}{(k+1)P_0 + \sigma_e^2} \] (5.9)

The Kalman filter of type one becomes

\[ \hat{x}_{k+1|k} = \hat{x}_{k|k-1} + \frac{P_0}{(k+1)P_0 + \sigma_e^2}(y_k - \hat{x}_{k|k-1}) \]

\[ = \frac{kP_0 + \sigma_e^2}{(k+1)P_0 + \sigma_e^2} \hat{x}_{k|k-1} + \frac{P_0}{(k+1)P_0 + \sigma_e^2} y_k \] (5.10)

**Type 2**

This type of Kalman filter has two states. State \( x_1 \) can be seen as the position and state \( x_2 \) as the slope per sample. The filter is designed to adapt to a linear output plus white noise. A filter of this type is started at every sample, beginning with the sample after the input-step changed to the higher level. For a filter of this type, started at sample \( k \) after the input changed, the notation \( P_k \) is used.

The state-space model is

\[ x_1(k + 1) = x_1(k) + hx_2(k) \]
\[ x_2(k + 1) = x_2(k) \]
\[ y(k) = x_1(k) + e(k) \] (5.11)
With the same notation as before

\[
\Phi = \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix}
\]

\[\Gamma = 0\]

\[C = \begin{pmatrix} 1 \\ 0 \end{pmatrix}\]

\[D = 0\]

\[R_1 = 0\]

\[R_2 = \sigma_e^2\]  \hspace{1cm} (5.12)

The covariance-matrix is

\[
P_{k+1} = \begin{pmatrix} p_{11}(k+1) & p_{12}(k+1) \\ p_{21}(k+1) & p_{22}(k+1) \end{pmatrix}
\]  \hspace{1cm} (5.13)

where

\[
p_{11}(k+1) = \frac{\sigma_e^2 (p_{11}(k) + h(p_{12}(k) + p_{21}(k)) + h^2 p_{22}(k)) + h^2 \text{det}(P(k))}{p_{11}(k) + \sigma_e^2}
\]

\[
p_{12}(k+1) = \frac{\sigma_e^2 (p_{12}(k) + h p_{21}(k)) + h \text{det}(P(k))}{p_{11}(k) + \sigma_e^2}
\]

\[
p_{21}(k+1) = \frac{\sigma_e^2 (p_{21}(k) + h p_{22}(k)) + h \text{det}(P(k))}{p_{11}(k) + \sigma_e^2}
\]

\[
p_{22}(k+1) = \frac{\sigma_e^2 p_{22}(k) + \text{det}(P(k))}{p_{11}(k) + \sigma_e^2}
\]  \hspace{1cm} (5.14)

The correction factor is

\[
K_k = \begin{pmatrix} k_{11}(k) \\ k_{21}(k) \end{pmatrix}
\]  \hspace{1cm} (5.15)

where

\[
k_{11}(k) = \frac{p_{11}(k) + h p_{21}(k)}{p_{11}(k) + \sigma_e^2}
\]

\[
k_{21}(k) = \frac{p_{21}(k) + h p_{21}(k)}{p_{11}(k) + \sigma_e^2}
\]  \hspace{1cm} (5.16)

The Kalman filter of type two becomes

\[
\hat{x}_1(k+1|k) = \hat{x}_1(k|k-1) + h \hat{a}_2(k|k-1) + k_{11}(k)(y(k) - \hat{x}_1(k|k-1))
\]

\[
\hat{x}_2(k+1|k) = \hat{x}_2(k|k-1) + k_{21}(k)(y(k) - \hat{x}_1(k|k-1))
\]  \hspace{1cm} (5.17)
5.4 The algorithm

The algorithm for finding the system time-delay is a branch-and-prune algorithm. The branching is done when new filters, \( F_k \), are started and the pruning when obviously incorrect filters, \( F_k \), are stopped. The pruning is necessary to keep the computation time down. The algorithm works on-line. That is, no prior information is needed and new output from the system is dealt with immediately. That means that the algorithm only needs to run about the same time as the time-delay itself. When the time-delay is found the algorithm stops.

In the pseudo-code below \( \mathcal{F} \) is the set of running Kalman filters.

begin algorithm
    Zero input
    Estimate noise spectral density
    Shift input to the higher level
    \( k = 0 \)
    \( \mathcal{F} = \{ F_0 \} \)
    repeat
        \( k = k + 1 \)
        \( \mathcal{F} = \mathcal{F} \cup \{ F_k \} \) (* Branch *)
        for all \( F \in \mathcal{F} - F_0 \)
            if Prob( \( F \) correct ) \( \leq \epsilon \) then
                \( \mathcal{F} = \mathcal{F} - \{ F \} \) (* Prune *)
            end{if}
        end{for}
    until ( Prob( \( F_0 \) correct ) \( \leq \epsilon \) )
    Find the “best” \( F \) from \( \mathcal{F} \) (and thereby the time-delay)
end algorithm

To complete the algorithm it has to be decided:

1.) How to choose initial values when starting a new Kalman filter.
2.) How to find an upper limit of the time-delay.
3.) How to do the pruning, i.e. how to stop filters, \( F_k \), that obviously does not correspond to the real time-delay.
4.) When the repetition has stopped, how to find the “best” \( F_k \), and thereby the system time-delay.

In the following sections these subjects are dealt with one by one.

5.5 Starting new Kalman filters

To start a new Kalman filter the initial values of the filter states and the covariance matrix have to be chosen. It is also necessary to estimate the variance of the noise, \( \sigma_e^2 \), which is needed in the Kalman filters. When filter \( F_0 \)
is started the \( N \) first samples, the samples before the input-step shifts to the higher level, are used to choose

\[
\hat{x}_{1|0} = E\{x_0\} = \bar{y} \\
P_0 = \frac{\sigma_y^2}{N} \\
\sigma_c^2 = \sigma_y^2
\]  

(5.18)

where

\[
\bar{y} = \frac{1}{N} \sum_{k=1}^{N} y_k \\
\sigma_y^2 = \frac{1}{N} \sum_{k=1}^{N} (y_k - \bar{y})^2
\]  

(5.19)

With these choices of initial values the Kalman filter \( F_0 \) becomes

\[
\hat{x}_{k+1|k} = k + N \frac{\hat{x}_{k+1|k-1} + 1}{k + N + 1} y_k \\
P_k = \frac{\sigma_c^2}{k + N}
\]  

(5.20)

(5.21)

When starting a Kalman filter, \( F_k \), the initial value of the position state is taken from \( F_0 \) and the velocity (slope per sample) state is set to zero. That is

\[
\hat{x}_1(1|0) = \hat{x}(k) \\
\hat{x}_2(1|0) = 0
\]  

(5.22)

The initial value of the covariance matrix is chosen as

\[
P_0^{F_1} = \begin{pmatrix} p_{11}(1|0) & p_{12}(1|0) \\ p_{21}(1|0) & p_{22}(1|0) \end{pmatrix} = \begin{pmatrix} P_{k} & 0 \\ 0 & \sigma_y \end{pmatrix}
\]  

(5.23)

The initial value, \( p_{22}(1|0) \), is important but hard to decide. Simulations have shown that choosing a very low value works well for low noise-levels but gives a bad performance for higher noise-levels. Choosing a relatively high value works better all over but does not give optimal performance for low noise-levels. The optimal value of \( p_{22}(1|0) \) also depends on which method that is used to make the final decision of the time-delay. For the method that I have found works best (see Section 5.8) a value of \( p_{22}(1|0) \) that is dependent of the estimated noise standard-deviation works best.

5.6 Finding an upper limit of the time-delay

When it is sure that the model \( F_0 \) is false it is clear that an upper limit of the time-delay has been found and that a Kalman filter, \( F_k \), corresponding to
the real time-delay has already been started. After that no more new filters should be started. To decide the upper limit of the time-delay some statistical theory has to be used. Let the hypothesis that the model $F_0$ still is correct at sample $k$ be the null hypothesis, $H_0$. A relevant test statistic is then

$$
\tau_N(k) = \sum_{i=k-n+1}^{k} \frac{\varepsilon_i}{\sqrt{n}} \epsilon N(0,1)
$$

(5.24)

where $\varepsilon_i$ is the normalized residuals between the output from filter $F_0$ and the system output.

$$
\varepsilon_i = \frac{y_i - \hat{y}_i}{\sigma_{\varepsilon_i}} = \frac{y_i - \hat{y}_i}{\sqrt{\sigma^2_{yi} + \sigma^2_{\hat{y}_i}}} = \frac{y_i - \hat{y}_i}{\sqrt{\sigma^2_{yi} + R_2 + CP_1C^T}} = \frac{y_i - \hat{y}_i}{\sqrt{2\sigma^2 + P_i}}
$$

(5.25)

and $n$ is the number of residuals in the sum. Any $n$-value in the interval 1 to $k$ can be used. The extreme cases are to use $n = 1$ or $n = k$ which gives

$$
\tau_N(k) = \bar{\varepsilon}_k
$$

(5.26)

and

$$
\tau_N(k) = \sum_{i=1}^{k} \frac{\varepsilon_i}{\sqrt{k}}
$$

(5.27)

A natural choice of criterion for rejecting the null hypothesis is

$$
|\tau_N| > v
$$

(5.28)

The value $v$ must be chosen so that the chance that $H_0$ is rejected too early (that is, a false alarm) is very small. If $H_0$ is true then

$$
P(|\tau_N| \leq v) = P(\tau_N \leq v) - P(\tau_N \leq -v) = \text{erf}(v) = \lambda_v
$$

(5.29)

The chance, $f$, of false alarm (i.e. premature rejection of $H_0$) equals

$$
f(v) = 1 - \lambda_v^T
$$

(5.30)

where $T$ is the time-delay and $h$ is the sampling period. If the probability of false alarm is to be less than some value, $f_{\text{lim}}$, then $v$ should be chosen as

$$
v = \text{erf}^{-1}(1 - f_{\text{lim}})\frac{h}{T}
$$

(5.31)

Of course, in reality the value of $T$ is not known, but if high values of the ratio time-delay to sampling period is expected it is reasonable to choose a higher value of $v$. Fig. 5.2, which is a plot of (5.31) gives an idea about how to choose $v$. In the implementation the $v$-value is decided as a function of the sampling period. That is

$$
v = \text{erf}^{-1}(0.998^{0.1h})
$$

(5.32)
Figure 5.2  Choice of detection threshold for $F_0$. The optimal $n$-value is plotted against the quote $\frac{T}{h}$ for the case that we want the probability of false alarm to be 0.002, 0.01 or 0.05. $T$ is the system time-delay and $h$ the sampling period.

The function (5.32) is found from (5.31) by putting $T = 10$s and $f_{\text{min}} = 0.002$.

Next consider the choice of $n$ in (5.24). Figures 5.3 and 5.4 show the results from extensive simulations on the example systems $G(s) = \frac{1}{s+1}$ and $G(s) = \frac{7}{s^2 + 5s + 7}$. In Fig. 5.3 the noise spectral density was varied between 0 and 0.2 and the $n$-values investigated were 1, 10 and $k$. The $y$-axis shows how often the test manages to find an upper limit of the time-delay that is less than 10 seconds larger than the real time-delay. In Fig. 5.4 the noise spectral density was varied between 0 and 0.03 and the $n$-values investigated were 1, 3 and 5. The $y$-axis shows the mean time (100 runs), beyond the delay time, to determine that the output has changed. Hundred simulations with different noise realizations were done for each $n$-value. It is obvious from Fig. 5.3a and Fig. 5.3b that in cases with high noise spectral density the $n$-value should be chosen as high as possible, i.e. $n = k$. Fig. 5.4a and Fig. 5.4b show that in the case of low noise spectral density it also seems to be best to choose $n$ as high as possible. All simulations so far point to that if a single $n$-value has to be decided the best choice is $n = k$. Possibly, a few test with different $n$-value could be run in parallel, rejecting $H_0$ when any of the corresponding test statistics say so. This gives a minor improvement in the length of the upper time-delay estimate. The price to be paid is a small increment in the probability of false alarm.

These simulations do not only give information about how to choose $n$ but they also say something about the limitations of this algorithm. In general, if the system has stationary gain equal to one, then to be able to give a "limited" upper limit of the time-delay the ratio amplitude of input-step to noise spectral density has to be larger than ten.
Figure 5.3  Simulations to give an idea about how to choose $n$. The $y$-axis shows how often (mean value of 100 runs) an upper limit of the delay, that is less than 10 seconds larger than the real delay, is decided. The system was $G(s) = \frac{1}{s+1}$ in figure a and $G(s) = \frac{1}{s+\frac{1}{2}}$ in figure b. The sampling period was 0.1 and the amplitude of the input-step one. The values corresponding to $n = 1$ is indicated by $+$, the ones corresponding to $n = 10$ are indicated with $o$ and the ones corresponding to $n = k$ are indicated with $\ast$.

Figure 5.4  Simulations to give an idea about how to choose $n$. The $y$-axis shows the mean time (100 runs), beyond the time-delay, to determine that the output has changed. The system was $G(s) = \frac{1}{s+1}$ in figure a and $G(s) = \frac{1}{s+\frac{1}{2}}$ in figure b. The sampling period was 0.1 and the amplitude of the input-step one. The values corresponding to $n = 1$ is indicated by $+$, the ones corresponding to $n = 3$ are indicated with $o$ and the ones corresponding to $n = 5$ are indicated with $\ast$.

5.7 Pruning

If it is absolutely sure that one of the filters, $F_k$, started earlier does not correspond to the real time-delay there is no reason to keep it running. Stopping the filter both saves computation time and removes one wrong alternative when the filter corresponding to the time-delay is to be chosen. To decide if a filter, $F_k$, is incorrect it is not possible, as in the previous section, to use any test on the residuals. The filters of type 2 have two states, position and velocity (slope per sample), and can adapt equally well to an output consisting of a constant plus noise and an output consisting of a linear signal plus noise. What is needed is to test that the value of the velocity state is not too low, which corresponds to an adaption to a constant output plus noise and means that the filter is started to early. It seems reasonable to let the test-quantity depend on both the estimate of the velocity state, $\hat{\sigma}_2$, and the variance of this
estimate, \( p_{22} \). That is, to use the test-quantity, \( T_k \), where

\[
T_k(i) = f( \hat{z}_2(i), p_{22}(i) ) \tag{5.33}
\]

and to terminate a filter, \( F_k \), if its \( T_k \)-value goes below some limit, \( t \). The function, \( f \), must be chosen so that the \( T_k \)-value for filters started too early is significantly lower than the \( T_k \)-value for filters started at a time corresponding to the system time-delay or later. The first attempt was to use the test-quantity

\[
T^1_k(i) = \hat{z}_2(i)^2 + p_{22}(i) \tag{5.34}
\]

However, simulations showed that this test-quantity puts too much weight on the estimate \( \hat{z}_2(i) \) and too little on its variance, \( p_{22}(i) \). A better test-quantity is

\[
T^2_k(i) = |\hat{z}_2(i) + 3 \cdot \text{sign}(\hat{z}_2(i)) \cdot \sqrt{p_{22}(i)}| \tag{5.35}
\]

This test-quantity calculates the absolute value of the maximum possible slope of filter \( F_k \) at every sample. If this value gets too low, then the corresponding filter \( F_k \) is not significantly different from filter \( F_0 \) and can as well be stopped.

Fig. 5.5 shows the results from simulations were the characteristics of this test-quantity were investigated. It is of interest to see if the test-quantity (5.35) is significantly lower for filters, \( F_k \), that should be stopped than for filters that should be kept running. Two different example systems were used. In plots a, c, and e the example system was \( G(s) = \frac{1}{s+1} \) and in plots b, d and f it was \( G(s) = \frac{1}{s^2 + s + 1} \). The sampling period was 0.1 seconds and the noise spectral density 0.001 in plots a, b, and 0.01 in plots c, d, and 0.1 in plots e, f. Each plot shows 20 realizations of the test-quantity (5.35) for the first started filter of type 2. Ten realizations were made for the case that the real time-delay was zero. In that case the first started filter is correct and should not be stopped. That is, high values of (5.35) are expected. The other ten realizations were made for the case that the real time-delay was ten seconds. In that case the first started filter is not correct and should be stopped. That is, low values of (5.35) are expected. Those realizations are also the ten lower ones in each plot.

As can be seen in plots a and b in Fig. 5.5, when the noise spectral density is 0.001 the quantity (5.35) decrease fast to zero in the case the time-delay is ten seconds. That is, the method would work well to stop incorrect filters. In plot c and d, the noise spectral density was 0.01 and still it works reasonably well. In plot e and f the noise spectral density was 0.1, a very high value, and as can be seen it is hard to distinguish the ten realizations with a time-delay of zero from the ten with a time-delay of ten seconds. If the noise-level is that high it is hard not to stop a filter that should not be stopped. In those cases it is preferable not to stop any filter.

The plots b, d and f in Fig. 5.5 show one potential problem. The realizations with time-delay zero and the realizations with time-delay ten seconds both decrease very fast the first few samples. The reasons for this behavior is not so strange. A system with a large time-constant is very easily interpreted as a system with longer time-delay. That is, the time-delay is overestimated, which would make it easy to stop a filter corresponding to the real time-delay. This is not really a big problem. Because the whole algorithm is constructed
Figure 5.5 Simulations testing the the quantity (5.35). The example system is $G(s) = \frac{1}{s+1}$ in plots a, c, e and $G(s) = \frac{1}{s^2 + s + 1}$ in plots b, d, f. The sampling period was 0.1 and the amplitude of the input-step one. The ten upper realizations in each plot correspond to a system time-delay of zero and the ten lower realizations correspond to a time-delay of ten seconds.

to estimate only the time-delay of a system it would overestimate the time-delay for slow systems anyway so stopping the correct filter does not change anything.

It is probably possible to find a better test-quantity than (5.35), but since this one has shown to work reasonably well it is used in the algorithm. The limit, $t_0$, is set to 0.1.

5.8 Finding the time-delay

To find the system time-delay all the Kalman filters of type 2 that are still running have to be searched and it has to be decided which one that best fits the
data. To be able to compare the performance of these filters some sort of cost-function, which should give a measure of how much the filter output differs from the real system output, is needed. Then it is just to choose the filter that minimizes that cost-function as the one that best correspond to the time-delay.

First some terminology:

\( k \): The number of the samples, starting from the sample the input-step changed to the higher level.

\( F_k \): As before, a Kalman filter of type 2 started at sample \( k \).

\( T \): The sample an upper limit of the time-delay was found and all filters, \( F_k \), stopped running.

\( N_k \): The number of samples a filter, \( F_k \), has been running, \( N_k = T - k + 1 \).

\( J^x_k \): The \( x \):th cost-function for filter \( F_k \). The \( x \) is used to differ between different types of cost-functions tried.

\( \epsilon_{k,i,p} \): The residual at sample \( i \) between a filter \( F_k \), started at sample \( k \), and the system output. The \( p \) shows that the residual is calculated using the progressive parameter estimates. That is

\[
\epsilon_{k,i,p} = y_i - \hat{y}_{k,i,p}
\]

\[
\hat{y}_{k,i,p} = C\hat{x}_{k,i|i-1}
\]  

(5.36)

\( \epsilon_{k,i,f} \): The residual at sample \( i \) between a Kalman filter, \( F_k \), started at sample \( k \), and the system output. The \( f \) shows that the residual is calculated using the final parameter estimates. That is

\[
\epsilon_{k,i,f} = y_i - \hat{y}_{k,i,f}
\]

\[
\hat{y}_{k,i,f} = C\hat{x}_{k,i|T}
\]  

(5.37)

\( e_{i,p} \): The residual at sample \( i \) between the filter of type 1 and the system output, using the progressive parameter estimates.

\( e_{i,f} \): The residual at sample \( i \) between the filter of type 1 and the system output, using the final parameter estimates.

One straightforward choice of cost-function is to use the sum of the squared residuals between the system output and the output from the filter \( F_k \), normalized by the number of samples the filter has been running. The residuals

34
are calculated using the progressive parameter estimates.

\[ J_k^1 = \frac{1}{N_k} \sum_{i=k}^{T} \varepsilon_{k,i,p}^2 \]  

(5.38)

Another way is to use the cost-function.

\[ J_k^2 = \sum_{i=k}^{T} \varepsilon_{k,i,p}^2 - \sigma_{\varepsilon_{k,i,p}}^2 \]  

(5.39)

That is, also here the sum of the squared residuals is calculated, but instead of normalizing by the number of samples the filter has been running, a term that equals the estimated variance of each residual is subtracted. Yet another method is to use the cost-function

\[ J_k^3 = \sum_{i=1}^{k-1} \varepsilon_{i,p}^2 + \sum_{i=k}^{T} \varepsilon_{k,i,p}^2 \]  

(5.40)

That is, to let the cost-function also include the residuals from the filter \( F_0 \) with the system output until the time when the filter, \( F_k \), was started. If instead the final parameter estimates are used the three new cost-functions become

\[ J_k^4 = \frac{1}{N_k} \sum_{i=k}^{T} \varepsilon_{k,i,j}^2 \]  

(5.41)

\[ J_k^5 = \sum_{i=k}^{T} \varepsilon_{k,i,j}^2 - \sigma_{\varepsilon_{k,i,j}}^2 \]  

(5.42)

\[ J_k^6 = \sum_{i=1}^{k-1} \varepsilon_{i,j}^2 + \sum_{i=k}^{T} \varepsilon_{k,i,j}^2 \]  

(5.43)

Simulations have shown that the results for cost-functions (5.38), (5.39), (5.41) and (5.42) are very similar. The cost-functions (5.41) and (5.42), which use the final parameter estimates are more robust and therefore better. The cost-function (5.40) works really badly for low noise-levels but very well for high noise-levels. The cost-function that works best all over and is most robust is (5.43) and it is therefore used in the algorithm.

5.9 Simulations

In the simulations eleven different systems, all with a time-delay of ten seconds, were used. The input signal was a step that was zero the first ten seconds and then one for 30 seconds. The sampling period was 0.1 seconds. The system disturbances were white noise with spectral densities in the range 0.001 to 0.07 and twenty runs were made for each noise-level. The eleven example systems are found in Appendix A, where the time-delay is not included in the transfer-functions.
Table 5.1 shows the absolute values of the errors (mean value over 20 runs) between estimated time-delay and real time-delay and Table 5.2 shows how often, in percentage, the algorithm manages to estimate a time-delay that is less than five samples (0.5 seconds) too big or too small.

The algorithm works best for system 1, a pure integrator with time-delay, which is the ideal case. But the results for system 2 and system 3 are not much worse, which shows that the idea behind the algorithm seems to work.

The systems 4 and 5 are two simple second order systems with two poles in $-1$ and $-10$ respectively. When the input signal starts affecting the output, the output from system 5 immediately increases very fast, while the output from system 4 increases much slower. That is why the estimated time-delays for system 5 are very close to the true time-delay, while the time-delays for system 4 are slightly overestimated.

System 6 is a very slow second order system, which gives large overestimations of the time-delay. Systems 7 and 8 are both second order systems. The only difference between them is that system 7 has higher relative damping. The algorithm has no problems with badly damped systems. The small overestimation of the delay depends on that also the output from these two system increase slowly in the beginning. Systems 9 and 10 are non-minimum-phase systems. In those cases the time-delay is also overestimated, depending on that the non-minimum-phase behavior is interpreted as an additional delay.

System 11 is of order eight with 8 poles in $-1$. This system has a long apparent time-delay, so it is not very surprising that the time-delay is overestimated. The time-delay estimates for this system depend strongly on the noise-level. When the noise spectral density increase, the time-delay estimates get much larger. This is reasonable considering the behaviour of the system output. There exist no distinct point where the output start growing more strongly. Instead, the output increases slowly for a long time, with only a small continuous increase of the slope. What the algorithm estimates is the apparent time-delay, which for this system strongly depends on the noise-level.

The simulations show that because the assumptions made when designing the algorithm, i.e. by modeling the system as an integrator with time-delay, the estimated time-delay will be a sum of the real time-delay and an apparent delay. A system has an apparent delay if it is non-minimum-phase or if it has high relative degree.

### 5.10 Conclusions

Some modifications of the algorithm are possible. For example to use only a limited number of samples after the filters of type 2 has been started in the cost-function for that filter. This modification can improve the performance of the algorithm, especially for high noise levels, but if no prior information of the system is available it is very hard to know how many samples that should be used. Below is a list of pluses and minuses about the algorithm

+ The algorithm works well without any preliminary knowledge of the system.

+ Only an input-step is needed and if the noise spectral density is not too high the amplitude of the step does not have to be very large.
- A step is the only input signal that can be used.

+ The algorithm works on-line and stops running when the time-delay is found.

+ The algorithm is not very noise sensitive. The results from simulations with high noise-levels are not much worse than the results from simulations with low noise-levels.

+ The algorithm is very robust. It works well on a wide range of systems and it is very rare with results that are far away from the real time-delay.

- The algorithm does become slower with shorter sampling period, but compared to the other algorithms studied in this report it still is relatively fast.
Table 5.1  Simulations showing the absolute value of the error between estimated time-delay and real time-delay. The value is a mean value over 20 runs. The value in the parenthesis is the standard-deviation of the 20 estimates.

<table>
<thead>
<tr>
<th>System</th>
<th>Sp. den. =0.001</th>
<th>Sp. den. =0.005</th>
<th>Sp. den. =0.01</th>
<th>Sp. den. =0.03</th>
<th>Sp. den. =0.05</th>
<th>Sp. den. =0.07</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00(0.00)</td>
<td>0.06(0.07)</td>
<td>0.09(0.12)</td>
<td>0.13(0.17)</td>
<td>0.24(0.33)</td>
<td>0.35(0.51)</td>
</tr>
<tr>
<td>2</td>
<td>0.00(0.00)</td>
<td>0.04(0.06)</td>
<td>0.12(0.15)</td>
<td>0.22(0.25)</td>
<td>0.22(0.31)</td>
<td>0.37(0.44)</td>
</tr>
<tr>
<td>3</td>
<td>0.00(0.00)</td>
<td>0.10(0.00)</td>
<td>0.18(0.06)</td>
<td>0.27(0.10)</td>
<td>0.29(0.26)</td>
<td>0.49(0.46)</td>
</tr>
<tr>
<td>4</td>
<td>0.25(0.18)</td>
<td>0.27(0.25)</td>
<td>0.34(0.41)</td>
<td>0.52(0.51)</td>
<td>0.35(0.32)</td>
<td>0.99(0.83)</td>
</tr>
<tr>
<td>5</td>
<td>0.00(0.00)</td>
<td>0.10(0.04)</td>
<td>0.12(0.04)</td>
<td>0.21(0.32)</td>
<td>0.30(0.40)</td>
<td>0.62(0.82)</td>
</tr>
<tr>
<td>6</td>
<td>0.50(0.46)</td>
<td>0.76(0.62)</td>
<td>1.21(0.65)</td>
<td>1.85(1.21)</td>
<td>3.93(0.87)</td>
<td>5.66(1.51)</td>
</tr>
<tr>
<td>7</td>
<td>0.22(0.11)</td>
<td>0.29(0.15)</td>
<td>0.26(0.16)</td>
<td>0.43(0.29)</td>
<td>0.43(0.45)</td>
<td>0.56(0.65)</td>
</tr>
<tr>
<td>8</td>
<td>0.23(0.07)</td>
<td>0.30(0.14)</td>
<td>0.30(0.18)</td>
<td>0.49(0.39)</td>
<td>0.43(0.38)</td>
<td>0.61(0.67)</td>
</tr>
<tr>
<td>9</td>
<td>1.37(0.26)</td>
<td>1.42(0.27)</td>
<td>1.51(0.21)</td>
<td>1.55(0.55)</td>
<td>1.60(0.77)</td>
<td>2.35(0.87)</td>
</tr>
<tr>
<td>10</td>
<td>0.20(0.00)</td>
<td>0.20(0.00)</td>
<td>0.66(0.45)</td>
<td>0.60(0.27)</td>
<td>0.53(0.31)</td>
<td>0.83(0.70)</td>
</tr>
<tr>
<td>11</td>
<td>2.16(0.78)</td>
<td>3.19(0.86)</td>
<td>3.47(0.66)</td>
<td>4.22(1.27)</td>
<td>5.09(1.43)</td>
<td>6.89(1.69)</td>
</tr>
</tbody>
</table>

Table 5.2  Simulations showing how often, in percentage, the algorithm manages to estimate a time-delay that is less than five samples (0.5 seconds) too big or too small.

<table>
<thead>
<tr>
<th>System</th>
<th>Sp. den. =0.001</th>
<th>Sp. den. =0.005</th>
<th>Sp. den. =0.01</th>
<th>Sp. den. =0.03</th>
<th>Sp. den. =0.05</th>
<th>Sp. den. =0.07</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>80</td>
<td>85</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>90</td>
<td>75</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>90</td>
<td>80</td>
<td>40</td>
<td>70</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>100</td>
<td>95</td>
<td>80</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>45</td>
<td>35</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>90</td>
<td>90</td>
<td>55</td>
<td>55</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>95</td>
<td>85</td>
<td>40</td>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>100</td>
<td>35</td>
<td>45</td>
<td>60</td>
<td>35</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

38
6. Software For Time-delay Estimation

6.1 Introduction

To be able to test and compare different approaches to time-delay estimation a piece of software has been implemented. The software creates a graphical, mouse-controlled, user-friendly environment where it is easy to test a new algorithm with a lot of different systems, sampling periods and disturbances. It also gives the user an opportunity to get an idea about how a new algorithm works compared to other existing algorithms.

6.2 Background

Many of the functions in the program were originally written by Dr. Brett Ninness, University of Newcastle. The simulations are carried out using Matlab functions from the Delta Toolbox, a toolbox where discrete-time systems are described using the delta-operator model instead of the commonly used shift-operator model. The delta model is described in [7]. The Delta Toolbox can be found and down-loaded at the web-site: http://www.ee.newcastle.edu.au/users/staff/rick/Middleton.html

6.3 Software

The software is implemented using the graphical user interface of Matlab (version 4.2).

6.4 Getting started

After installing the necessary files, the program is started from MATLAB by typing "idbox". Doing that displays the main-window shown in Fig. 6.1. As can be seen in the figure there are two spaces for the plots of the reference signal and the system output. There are also seven push-buttons. Starting from the top, the first button gives a new menu where the wanted system (including disturbances) can be set. The second button gives a menu for setting the reference signal, sampling period and length of simulation. Closing that menu automatically displays a plot of the reference signal in the main-window. Pressing the third button simulates the system with the chosen reference signal and displays the system output in the main-window. The fourth button gives a menu where different algorithms for time-delay estimation can be executed. This menu also has help-buttons which describes the different algorithms. The
Figure 6.1 The main-window

following two buttons are to save and restore earlier settings chosen. Finally, the last button is to exit the program.

The next few pages describe one example of how to use the program. In the example the Kalman filter approach to time-delay estimation from Chapter 5 is tried on a second order system with time-delay ten seconds. The experiment is started by pressing the button Choose Reference Signal in the main-window shown in Fig. 6.1. Doing that displays the menu shown in Fig. 6.2, where the length of the simulation, sampling period and reference-signal can be chosen. The sampling period is set to 0.1 seconds and the reference-signal to a step that is zero the first ten seconds and then one for 30 seconds. Thereafter the menu is closed by pressing Close which brings us back to the main-window.

To choose the true system we start by pressing the button Choose True System in the main-window. Doing that displays the menu shown in Fig. 6.3, where the true system transfer-function and time-delay can be set. The system time-delay is set to 10 seconds, but instead of typing in a system numerator and denominator the button Set Preset Transfer Function is pressed, which displays the menu shown in Fig 6.4. In this menu it is possible to choose a system from a wide range of preset example systems. We choose the second order system at the bottom of the list and then press Close, which takes us back to the menu in Fig. 6.3.

To set the system disturbances we start by pressing the button Set System Disturbances. Doing that displays the menu shown in Fig. 6.5. In this menu process- and measurement-noise can be set. It is also possible to choose an extra disturbance-function. We choose to have only white measurement noise with spectral density 0.02 acting on the system. Pressing Close takes us back to the menu in Fig. 6.3 and pressing Close in that menu brings us back to the main-window.

To see how the output looks with the chosen system and reference signal
the button Create Output is pressed. The result can be seen in Fig. 6.6.

To choose an identification algorithm we start by pressing the button Identify which displays the menu shown in Fig. 6.7. The method using Kalman filters is chosen and pressing the button Kalman Filter Method starts the identification and gives the results shown in Fig. 6.8. As can be seen, in this case the method overestimated the time-delay by three samples. Pressing Close takes us back to the main-window where new experiment can be conducted.
Figure 6.4 Menu for choosing a preset transfer-function.

Figure 6.5 Menu for choosing system disturbances.
Figure 6.6 The main-window after choosing the true system, disturbances and reference-signal.

Figure 6.7 Menu for choosing an algorithm for time-delay estimation.
Figure 6.8 The results from time-delay estimation using the fault detection approach from Chapter 5.
A. Example systems

\[ G_1(s) = \frac{1}{s} \]  
\[ G_2(s) = \frac{1}{s + 1} \]  
\[ G_3(s) = \frac{10}{s + 10} \]  
\[ G_4(s) = \frac{1}{(s + 1)^2} \]  
\[ G_5(s) = \frac{100}{(s + 10)^2} \]  
\[ G_6(s) = \frac{0.2^2}{s^2 + 2 \cdot 0.5 \cdot 0.2s + 0.2^2} \]  
\[ G_7(s) = \frac{1}{s^2 + 2 \cdot 0.5 \cdot 1s + 1} \]  
\[ G_8(s) = \frac{1}{s^2 + 2 \cdot 0.1 \cdot 1s + 1} \]  
\[ G_9(s) = \frac{-0.7s + 1}{(s + 1)^3} \]  
\[ G_{10}(s) = \frac{100(1 - s)}{(s + 1) \cdot (s^2 + 2 \cdot 0.2 \cdot 10s + 100)} \]  
\[ G_{11}(s) = \frac{1}{(s + 1)^8} \]  

45
B. Bibliography


