Evaluation of (unstable) non-causal systems applied to iterative learning control

TU/e

Technische Universiteit Eindhoven
Faculteit Werktuigbouwkunde
Sectie Systems & Control

Stagerapport 2001.08

Evaluation of (unstable) non-causal systems applied to iterative learning control

door Maurice Schneiders

Rapport van een interne stage
uitgevoerd van 10 april 2000 tot 29 september 2000

Begeleider: **M.J.G. van de Molengraft**
Afstudeerhoogleraar: **M. Steinbuch**

Evaluation of (unstable) non-causal systems applied to iterative learning control

M.G.E. Schneiders,
M.J.G. van de Molengraft,
M. Steinbuch

February 14, 2001

Abstract

This paper presents a new approach towards the design of iterative learning control. In linear motion control systems the design is often complicated by the inverse plant sensitivity being non-causal and unstable. To overcome these problems we apply high-performance differentiating filters together with a numerical collocation technique to compute the control signal. This allows for an accurate approximation of the theoretical solution. Moreover, the new approach offers the advantage of control over the boundary conditions of the learnt signal. It will be demonstrated with an example, i.e. an industrial H-drive.

1 Introduction

In this paper an algorithm for *ILC* (Iterative Learning Control) design is proposed that can deal with both non-causal and unstable inverse plant sensitivities. Because the presented algorithm is

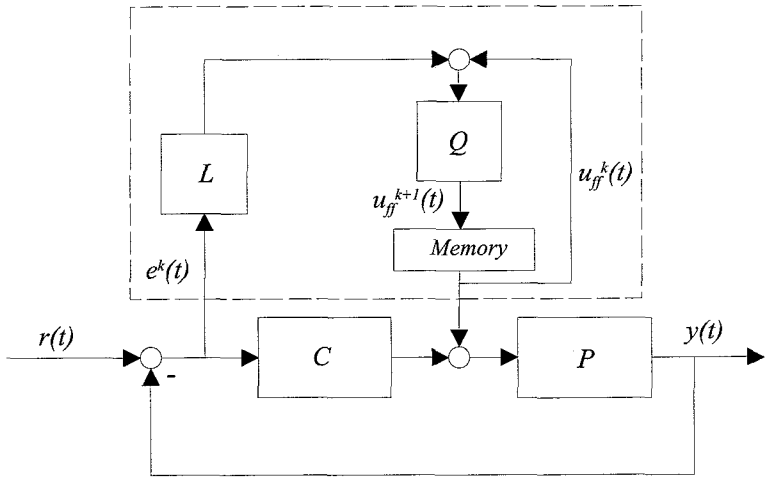


Figure 1: iterative learning setup

designed for use in learning control, some comment on *ILC* is made (for more information i.e. see [1]). Iterative learning control improves the tracking accuracy of a (closed-loop) control system by learning from previous experience prescribing the same trajectory. This is done by updating the feedforward signal u_{ff} in an iterative way according to a learning law. If we have a SISO loop with

plant P and feedback-controller C, we can measure the frequency response function belonging to the process (or plant) sensitivity:

$$PS = P\frac{1}{1+PC} = \frac{P}{1+PC} \tag{1}$$

where P is the plant transfer function, C the controller transfer function and S is the sensitivity of the closed loop. The learning filter $L(s)$ translates the tracking error $e(t)$ into the necessary feedforward action, which is then added to the existing feedforward signal $u_{ff}(t)$ (see [1] for more details). Q is a so-called robustness filter.

$$u_{ff}^{k+1}(t) = Q\{u_{ff}^{k}(t) + Le^{k}(t)\} \tag{2}$$

The best choice for $L(s)$ is therefore the inverse system $(PS)^{-1}$. To guarantee convergence of the learning process a robustification filter $Q(s)$ is added to cope with model imperfections and measurement noise. Usually $Q(s)$ is a lowpass filter. For the learning filter L the inverse model of the process sensitivity PS must be determined. Because closed-loop control systems are (at least designed to be) causal stable system, the inverse system cannot be evaluated directly. There are two reasons why this evaluation is not straightforward:

- inversion of a causal system leads to a non-causal system
- if the system is non-minimum phase, the inverse system will have unstable poles

A commonly used method is ZPETC [2]. ZPETC needs a system description in discrete time (transfer function) form. In discrete time, the non-causality is canceled by a simple time shift. For unstable zeros, which would become unstable poles of the inverse system, the method cancels the phase shift induced by them. This phase cancellation assures that the frequency response of the exact inverse system and the calculated one exhibits zero phase shift for all frequencies. The method is fast, but not always accurate because we do not get the exact amplitude behaviour of the inverse system. Especially for non-minimum phase systems a bad (but stable) approximation of the inverse model using ZPETC is obtained (see results in section 5). The next sections will discuss another approach towards evaluating an inverse dynamic system H^{-1}, not yet encountered in ILC practice.

2 $H^{-1}(s)$-systems

We assume that we have a reliable (linear) model of system H in transfer function form $H(s)$. To avoid numerical problems with higher order systems we prefer the use of *zero-pole-gain* notation instead of the *polynomial transfer function* form. For non-causal systems, as $H^{-1}(s)$ in general will be, the number of zeros is higher than the number of poles ($M + L \geq K$ Eq. 3). It is also assumed that $H(s)$ is a minimal realization of the system H.

$$\begin{aligned} H^{-1}(s) = \frac{Num(s)}{Den(s)} &= K\prod_{m=1}^{M}(s+c_m)\frac{\prod_{l=1}^{L}(s+b_l)}{\prod_{k=1}^{K}(s+a_k)} \\ &= H_{nc}(s) \quad H_c(s) \end{aligned} \tag{3}$$

We can split up $Num(s)$ in two (real) parts, and if we satisfy the inequality $L \leq K$, the system is split up in a causal (but probably unstable) part $H_c(s)$ and a true non-causal part $H_{nc}(s)$:

$$H_{nc}(s) = K\prod_{m=1}^{M}(s+c_m) = \sum_{m=0}^{M} c_m s^m = c_M s^M + c_{M-1}s^{M-1} + \ldots + c_1 s + c_0 \tag{4}$$

As we see (Eq. 4, our non-causal system $H_{nc}(s)$ can be written as a linear combination of differentiators of different order. Since the whole discrete time input series $e(t)$ is known, we must

design decent non-causal discrete time differentiating filters, as we will see in the next section. The output of this evaluation $y_{nc}(t)$ serves as an input for solving the causal part (section 4). In spite of the fact that $H_c(s)$ can be unstable we are able to evaluate such system in a finite time interval. We only need an accurate ODE solver to reduce numerical errors to the minimum. Solvers using a shooting or collocation method seemed to be most appropriate. By choosing a solver for mixed boundary value problems we can put extra constraints on begin and end values of the output signal. Section 5 will show the advantages of this new approach over the existing techniques in an example. Conclusions are given in section 6.

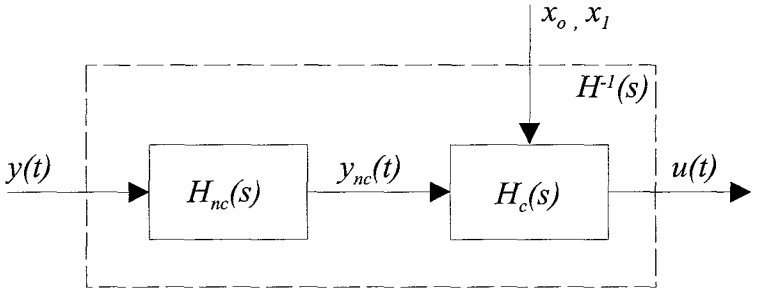


Figure 2: schematic representation of the used method

3 Differentiating filters

We are looking for digital differentiating filters, as just mentioned. Important work in this field is done in [3] and [4], where a number of first order differentiating filters are presented. A Rth-order differentiator in Laplace-domain, $H_R(s) = s^R$, has $\frac{\pi}{2}R$ phase-lead and is non-causal. The ideal frequency response for a discrete-time differentiator is:

$$H_R^i(q = e^{i\omega T}) = (i\omega)^R \qquad \omega \leq \frac{\pi}{T} \tag{5}$$

3.1 A reconstruction problem

In most cases we don't have the pure continuous signal available to calculate the derivative. All measurements y_k are noisy and in discrete time.

$$y_k = s_k + v_k = s(kT) + v(kT) \tag{6}$$

With only y_k given, we try to reconstruct the Rth derivative d_k^R of the signal $s(kT)$:

$$\hat{d}_k^R = H_R(q = e^{i\omega T})y_k \tag{7}$$

The design of a differentiating filter is nontrivial for al least two reasons:

- the relation between the sampled-time series s_k and the continuous-time signal $s(t)$ is not known in general.
- in general the measurements y_k will be corrupted with noise v_k.

Because in learning control a preceding filter Q will bother about the noise-pollution of the signal in a decent way, the differentiating filter doesn't have to cope with noise-reduction in the first place. The reconstruction problem mentioned is a much bigger problem. There are two main methods to reconstruct a continuous signal out of discrete-time data:

- Shannon reconstruction
- polynomial interpolation/fitting

Differentiating filters based on both methods will be briefly discussed in the next subsection.

3.2 Filter selection

The most accurate differentiating filters presented in [4] use one of these two reconstruction methods and are both non-causal FIR filters. Digital filters with Finite-duration Impulse Response (all-zero, or FIR filters) have both advantages and disadvantages compared to Infinite-duration Impulse Response (IIR) filters. FIR filters have the following primary advantages:

- can have exactly linear phase
- always stable
- filter startup transients have finite duration

The primary disadvantage of FIR filters is that they often require a much higher filter order than IIR filters to achieve a given level of performance. Because linear phase-behaviour of IIR filters cannot be guaranteed (except using (forward-backward) non-causal zero(!)-phase filtering), they are no good way to design accurate differentiating filters. The basic digital FIR filter looks like:

$$H(q) = \sum_{n=-N_1}^{N_2} h_n q^{-n} \tag{8}$$

Where q is the shift operator: $x_{k+1} = qx_k$. Since all filtering occur off-line, computation time is no restriction and we can use a non-causal filter ($N_1 > 0$). A general layout for a N^{th} order differentiating FIR filter to calculate the R^{th} derivative (see appendix A) looks like:

$$H_R(q) = \frac{1}{2T^R}\left(\sum_{n=1}^{N} c_n(q^n + (-1)^R q^{-n}) + c_0\right) \tag{9}$$

With this layout we guarantee the right phase behaviour (as in Eq. 5), and we calculate filter coefficients which are independent of the sample time T. In [4], a design criterion is suggested which covers both reconstruction methods. We calculate our filter coefficients minimizing the following cost function:

$$E = \int_0^{\frac{\pi}{T}} |H_R^i(i\omega) - H_R(q = e^{i\omega T})|^2 d\omega \tag{10}$$

subject to the constraint

$$\mathbf{W}\tilde{c} = \tilde{e} \tag{11}$$

with $\tilde{c} = [c_0 \; c_1 \ldots c_N]^T$. The constraints in Eq. 11 (which can be zero up to N in number) will be further specified in the next subsections. Mostly they represent constraints on the filters derivatives for $\omega = 0$.

3.3 Shannon-based differentiators

We can reconstruct our continuous-time signal using Shannon reconstruction (see [3]). Differentiating filters based on this method let $N \to \infty$ (as the Shannon reconstruction does), so they are

unrealizable. We can proof that using this method is the same as constructing a filter by only optimizing the cost function from Eq. 10. We can truncate the filter order to make a realizable filter. We will get a wide-band (up to $\omega \rightarrow \frac{\pi}{T}$) differentiator. The disadvantage of truncating the filter to order N are oscillations in the amplitude response of the filter, due to the discontinuity around $\omega = \frac{\pi}{T}$ (known as the Gibb's phenomenon). Another approach based on the Shannon reconstruction using the cost function (but now optimizing the integral up to $\alpha\frac{\pi}{T}$ instead of $\frac{\pi}{T}$) and one extra constraint $\frac{\partial}{\partial\omega}H_R(q = e^{i\omega T})|_{\omega=0} = 0$ is called the Usui & Amidror approach (according to [4]). Other methods that reduce these oscillations use modified forms of the ideal differentiator (Eq. 5) in the cost function. For first order derivative filters all these methods have been discussed earlier in [3] and [4]. These Shannon-based methods all retain some oscillatory behaviour in their amplitude response (e.g. see fig. 3). Some of the above mentioned methods were expanded for

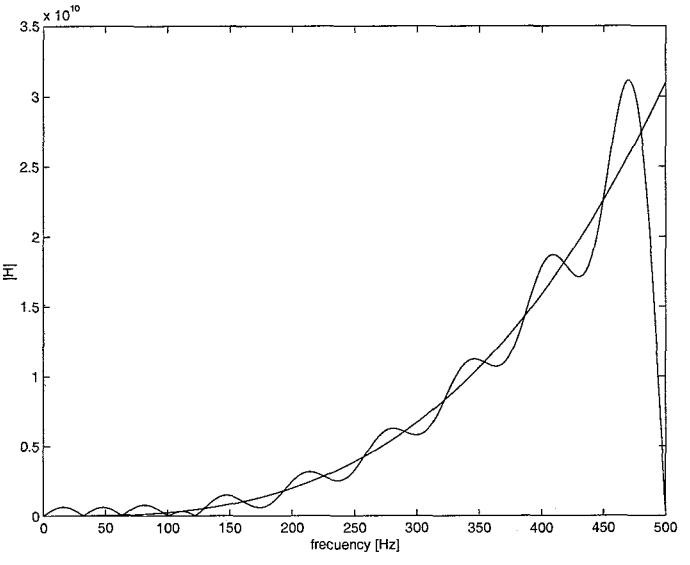


Figure 3: Amplitude response for a truncated ideal Shannon-based differentiator (R=3 N=15 T=10^{-3} s)

higher order derivatives. All these methods bring wide band differentiators but the price we pay are the Gibb's oscillations. These oscillations give big relative errors, especially for low frequencies in higher order derivative filters. Therefore these methods are not very suited for our goal; we need very accurate higher order differentiators, especially for the low frequency range.

3.4 Polynomial interpolation/fitting

Another approach only uses the constraints from Eq.11.

$$\begin{bmatrix} \frac{\partial^p}{\partial\omega^p}H_R(q = e^{i\omega T})|_{w=0} \\ \frac{\partial^{p+1}}{\partial\omega^{p+1}}H_R(q = e^{i\omega T})|_{w=0} \\ \vdots \\ \frac{\partial^P}{\partial\omega^P}H_R(q = e^{i\omega T})|_{w=0} \end{bmatrix} = \begin{bmatrix} \frac{\partial^p}{\partial\omega^p}(i\omega)^R|_{w=0} \\ \frac{\partial^{p+1}}{\partial\omega^{p+1}}(i\omega)^R|_{w=0} \\ \vdots \\ \frac{\partial^P}{\partial\omega^P}(i\omega)^R|_{w=0} \end{bmatrix} \tag{12}$$

If we take the derivative of the basic filter (Eq. 9), for the lefthand side we get:

$$\begin{aligned} \frac{\partial^p}{\partial\omega^p}H_R(q)|_{w=0} &= \frac{\partial^p}{\partial\omega^p}\frac{1}{2T^R}\Big(\sum_{n=1}^{N} c_n(e^{ni\omega T} + (-1)^R e^{-ni\omega T}) + c_0\Big)|_{w=0} \\ &= \frac{1}{2T^R}\sum_{n=1}^{N} c_n((niT)^p + (-1)^R(-niT)^p) \end{aligned}$$

$$
\begin{aligned}
&= \frac{i^p T^{p-R}}{2} \sum_{n=1}^{N} c_n n^p (1 - (-1)^{R+p}) \\
&= \begin{cases} 0 & \text{if } R+p \text{ is even} \\ i^p T^{p-R} \sum_{n=1}^{N} c_n n^p & \text{if } R+p \text{ is odd} \end{cases}
\end{aligned} \tag{13}
$$

The righthand side prescribes the values for the derivatives using the values of the ideal differentiator of Eq. 5:

$$
\begin{aligned}
\frac{\partial^p}{\partial \omega^p} (i\omega)^R|_{w=0} &= i^R \frac{\partial^p}{\partial \omega^p} \omega^R|_{w=0} = i^R \frac{R!}{(R-p)!} \omega^{R-p}|_{w=0} \\
&= \begin{cases} 0 & \text{if } R \neq p \\ i^R R! & \text{if } R = p \end{cases}
\end{aligned} \tag{14}
$$

Combining the results of Eq.13 and Eq.14 in the set of constraints (Eq.12, we get:

$$
\begin{bmatrix} \vdots \\ \sum_{n=1}^{N} c_n n^p \\ 0 \\ \sum_{n=1}^{N} c_n n^{p+2} \\ \vdots \\ 0 \\ \sum_{n=1}^{N} c_n n^P \end{bmatrix} = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ R! \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{15}
$$

Because of the zeros in Eq.13 if $R+p$ is even, we need $p = 1, \ldots, 2N$ to get N usable equations to compute the N coefficients $c_1, \ldots, c_N$ (solving a determined set of equations). For even derivative filters, we optionally use $H_R(q = e^{i\omega T})|_{w=0} = 0$ to calculate c_0 (for odd derivatives $c_0 = 0$). It can be proved [4] that this approach gives the same filter as one based on polynomial interpolation. A filter based on polynomial interpolation:

- determines the (unique) interpolating polynomial of degree $2N$ from $2N+1$ data points
- estimates the Rth derivative by differentiating the polynomial

As expected from the set of constraints, this method has extremely low error in the low-frequency range and a little low-pass behaviour for higher frequencies. Note that $2N \geq R$ to obtain a usable filter.

Besides polynomial interpolation, we can use polynomial fitting to design a differentiating filter. This fitting technique determines a polynomial fit of degree $2N$ from $2P+1$ data points ($P > N$). We can calculate the filter coefficients by using equation 15 but now using P (usable) equations instead of N. We obtain an overdetermined set of equations, which we can solve by a least squares minimalisation. Such filter smoothes out some high frequencies so this method has more lowpass behaviour than the interpolation technique. The low frequency accuracy is almost the same for both methods. Since the filter does not have to deal with (high frequency) noise, we choose polynomial interpolation as the best method to design differentiating filter for this particular purpose.

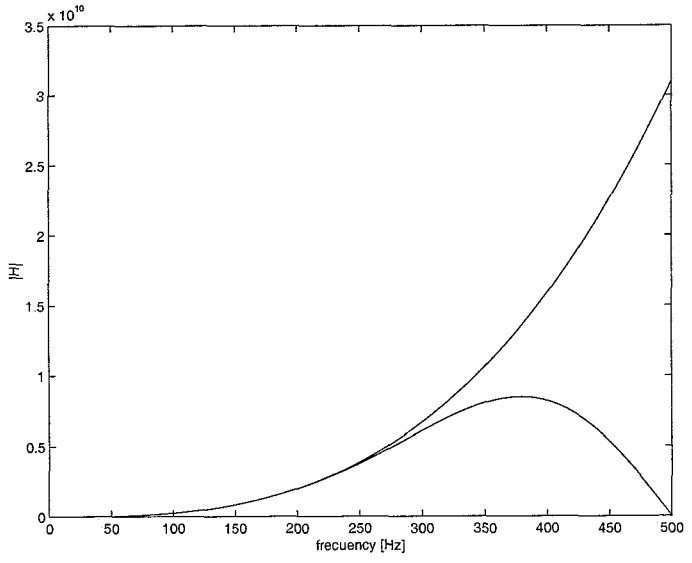


Figure 4: Amplitude response for a differentiator based on polynomial interpolation (R=3 N=6 T=10^{-3} s)

3.5 Cancelling border distortions

Since we use finite time series, we have to take special care to the borders of our signal. The chosen filter is non-causal so we have to do some data-expansion on begin and the end of our data to minimize border distortions. Looking at the good characteristics of polynomials in the filter design, we chose to expand our data using polynomial extrapolation. If we want to determine the R^{th} derivative of a data series, the polynomial order of the expansion must be $\geq R$. We get the best overall results if we use a polynomial of order R, fitted on $R+1$ border points.

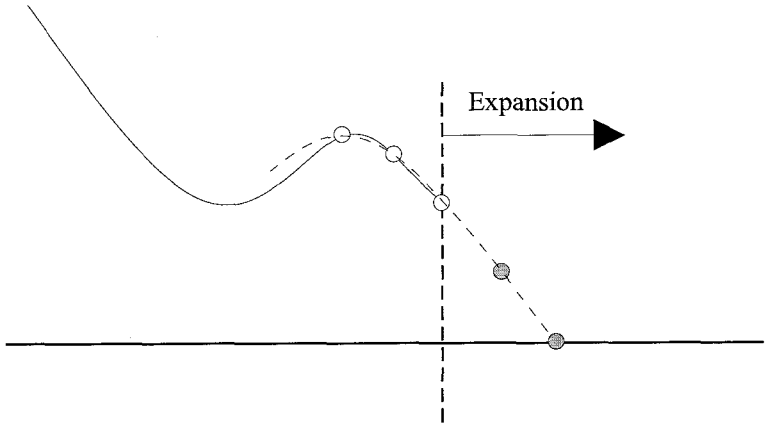


Figure 5: example of data expansion: 2^{nd} order expansion based on 3 data points

4 An accurate ODE solver

Now that we are able to calculate the non-causal system $H_{nc}(s)$ (from Eq. 3, we denote the discrete-time output data as $y_{nc}(k)$, we can use this as an input to evaluate the causal system $H_c(s)$ of order K (Eq. 3). To evaluate this system, we use a ODE-solver from the NAG foundation. This routine calculates the solution of a two-point boundary value problem for a regular linear

n^{th} order system of first order ordinary differential equations in Chebyshev-series in a specified range. The n boundary conditions can by specified on begin $\tilde{x}(t_0)$ or end $\tilde{x}(t_1)$ points (no mixed problems can be solved).

$$\dot{\tilde{x}}(t) = \mathbf{A}(t)\tilde{x}(t) + u(t) \tag{16}$$

The boundary conditions are solved exactly, and the remaining equations resulting from the system of Chebyshev polynomials are then solved by a least-squares method. The algorithm is fast but cannot handle (very) stiff problems. To ensure accuracy we used scaling: the time-constants of the system $H_c(s)$ are scaled back and the time-scale is elongated.

To solve system $H_c(s)$ we use this NAG routine because of two specific properties:

- it can solve unstable ODE's in an accurate way
- we can evaluate the system as a two-point boundary value problem (BVP)

The first property is strictly needed since system $H_c(s)$ can have unstable poles (because the total system can be non-minimum phase). The second property let's us prescribe K boundary conditions like:

$$\begin{bmatrix} \tilde{x}(t_0) \\ \tilde{x}(t_1) \end{bmatrix} = \tilde{v}_{bc} \tag{17}$$

Most ODE-solvers just ask to prescribe initial conditions: $\tilde{x}(t_0) = \tilde{x}_0$. Eq. 17 gives much more freedom in choosing boundary conditions on begin (t_0) and end (t_1) time. Especially in this case where $y(t)$ is a feedforward signal in a motion system we can prescribe the signal (and his derivatives) to be zero at the starting and the end points, which is a desirable property in controlling motion systems. To use the algorithm we have to transcribe the system $H_c(s)$ into state-space notation:

$$\begin{cases} \dot{\tilde{x}}(t) &= \mathbf{A}\tilde{x}(t) &+ \mathbf{B}u(t) \\ y(t) &= \mathbf{C}\tilde{x}(t) &+ Du(t) \end{cases} \tag{18}$$

Note that the system has order K and is still SISO. Since the order of $H_{nc}(s)$ (in Eq. 3) is restricted due to numerical problems, the numerator of the causal system $H_c(s)$ will not be a constant in general. This means that the states of system in Eq. 18 are not just linear combinations of the output $y(t)$ and its derivatives. These states are now combinations of input $u(t)$ and the output $y(t)$ and their derivatives. We can write the system in the observable canonical form:

$$\mathbf{A} = \begin{bmatrix} -a_{K-1} & 1 & & \emptyset \\ \vdots & & \ddots & \\ -a_1 & \emptyset & & 1 \\ -a_0 & 0 & \cdots & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} b_{K-1} \\ \vdots \\ b_1 \\ b_0 \end{bmatrix} - b_K \begin{bmatrix} a_{K-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}, \mathbf{D} = b_K \tag{19}$$

We can state the state-conditions at a certain moment t^* in terms of the input and output values and their derivatives at moment t^* as:

$$\mathcal{O}\tilde{x}(t^*) = \begin{bmatrix} y(t^*) \\ \dot{y}(t^*) \\ \vdots \\ y^{K-1}(t^*) \end{bmatrix} - \mathcal{R} \begin{bmatrix} u(t^*) \\ \dot{u}(t^*) \\ \vdots \\ u^{K-1}(t^*) \end{bmatrix} \tag{20}$$

in which $y^r(t^*)$ is defined as the r^{th} time-derivative of y at moment t^* ($\frac{\partial^r x}{\partial t^r}\big|_{t=t^*}$), and

$$\mathcal{R} = \begin{bmatrix} D & 0 & 0 & 0 & 0 & \cdots & 0 \\ \mathbf{CB} & D & 0 & 0 & 0 & \cdots & 0 \\ \mathbf{CAB} & \mathbf{CB} & D & 0 & 0 & \cdots & 0 \\ \mathbf{CA^2B} & \mathbf{CAB} & \mathbf{CB} & D & 0 & \cdots & 0 \\ & & & \vdots & & & \\ \mathbf{CA^{K-2}B} & \mathbf{CA^{K-3}B} & \cdots & \cdots & \mathbf{CAB} & \mathbf{CB} & D \end{bmatrix}, \mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA^2} \\ \vdots \\ \mathbf{CA}^{K-1} \end{bmatrix} \tag{21}$$

Note that $\mathcal{O}$ is the lower triangular observability matrix. If we want to prescribe the output and its derivatives up to a certain order at t^*, we don't need to know the whole state-vector at t^*. Since both $\mathcal{R}$ and $\mathcal{O}$ are lower triangular matrices, for $1 < p < K$, the first p states will prescribe $y(t^*), \dot{y}(t^*)$ up to $y^{p-1}(t^*)$ if $u(t^*), \dot{u}(t^*)$ up to $u^{p-1}(t^*)$ are given. Since the input signal is given $(y_{tnc}(t))$, we can calculate the derivatives at every moment using a filter based on polynomial expansion as presented in section 3. Now we can convert the boundary conditions given in Eq. 17 to initial and end conditions on the output $y(t)$:

$$\begin{bmatrix} y(t_0) \\ \dot{y}(t_0) \\ \vdots \\ y^{p-1}(t_0) \\ y(t_1) \\ \dot{y}(t_1) \\ \vdots \\ y^{K-p}(t_1) \end{bmatrix} = \tilde{y}_{bc} \tag{22}$$

5 Applications

First this new approach is applied on an academic non-minimum phase system. Some systems in process industry as boiler systems consist of combinations of opposing effects between first- or second-order subsystems (see Fig. 6 and Eq. 23). If we choose $\frac{k_2-k_1}{k_1\tau_2-k_2\tau_1} > 0$ the system becomes non-minimum phase. The system is controlled with a very mild tuned PD-controller. We want to track the trajectory from Fig. 7.

$$H(s) = \frac{(k_1\tau_2 - k_2\tau_1)s - (k_2 - k_1)}{(\tau_1 s + 1)(\tau_2 s + 1)} \tag{23}$$

To increase performance *ILC* is applied in 5 iterations. The results are clear: ZPETC cannot deal

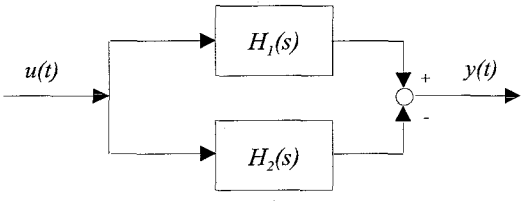


Figure 6: block diagram of a non-minimum phase system ($H_1 = \frac{k_1}{\tau_1 s+1}$ and $H_2 = \frac{k_2}{\tau_2 s+1}$)

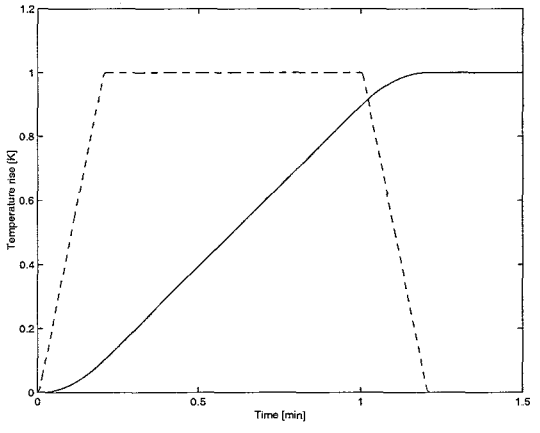
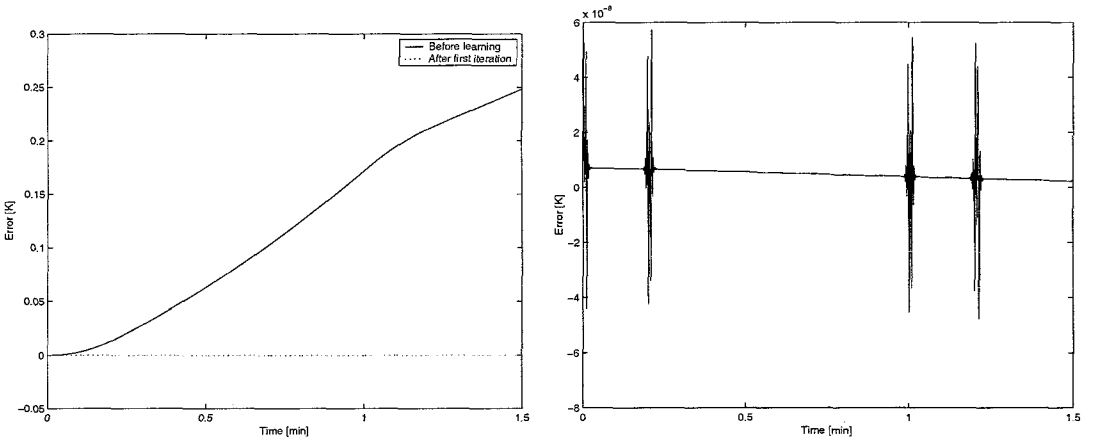


Figure 7: Designed trajectory. This can be a prescribed step in the temperature of a boiler system

with such non-minimum phase system, where the new approach shows convergence and succeeds in suppressing the tracking error.



(a) before and after first iteration

(b) after fourth iteration

Figure 8: Tracking error after several iterations using the new method

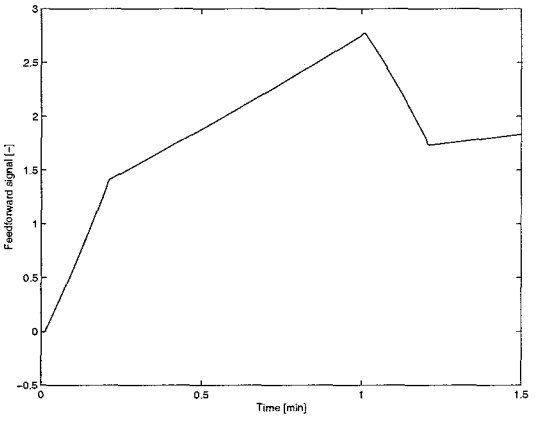


Figure 9: feedforward signal after four iterations

Also the new approach and standard ZPETC are both applied to an industrial motion system, i.e. an H-drive. This robot uses 3 linear motion motor systems (LIMMs) to drive the 2 main sliders. Only the X-slide is considered in the learning process (the most horizontal slider in fig. 5). The control loop is implemented with a dSPACE system. We used an 10^{th} order model of the process sensitivity. A PID with lowpass D controller is used, resulting in a bandwidth of 30 Hz. *ILC* is used with a lowpass robustification filter with a cutoff frequency at 250 Hz. After 7 iterations both methods converge. The resulting tracking error is of the same order.

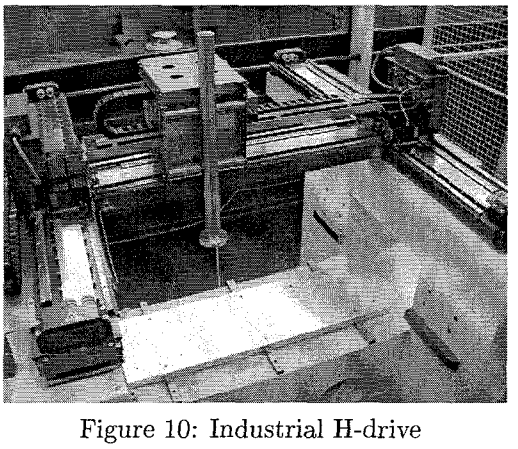
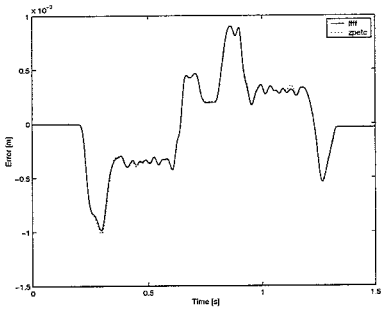
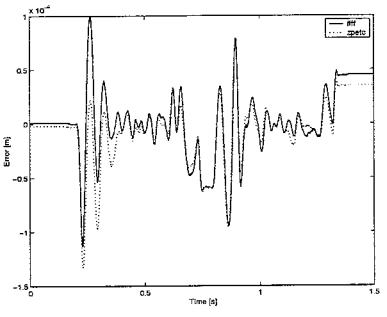


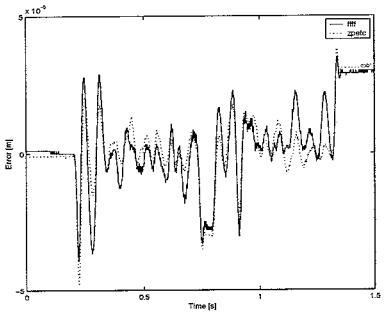
Figure 10: Industrial H-drive



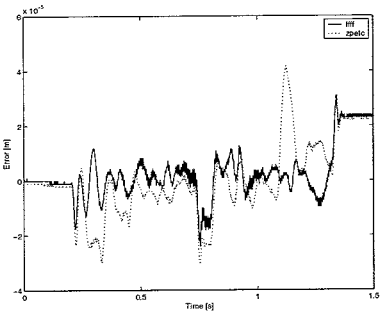
(a) Before ILC; no feedforward



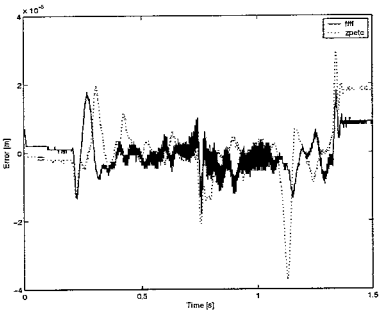
(b) after one iteration



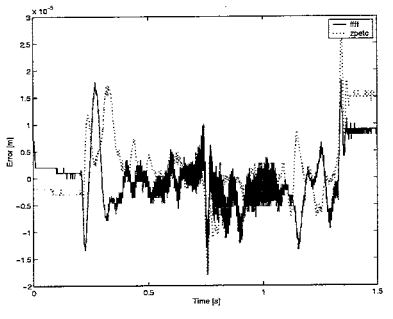
(c) after two iterations



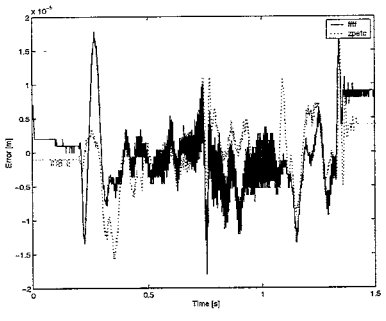
(d) after three iterations



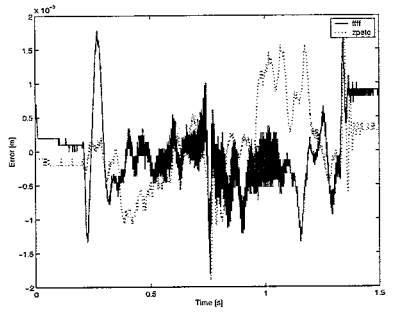
(e) after four iterations



(f) after five iterations



(g) after six iterations



(h) after seven iterations

Figure 11: Resulting error after ILC iterations using ZPETC and the new method

6 Conclusions

At this point, contrary to ZPETC, the new approach succeeds in calculating inverse responses of non-minimum phase systems. For minimum phase systems of higher order the implementation of this new method is not competitive to ZPETC-method [2] in sense of computation time. One major advantage is that we can put restrictions on begin and end values of the feedforward signal, which is very admirable in motion control. Furhter optimizing the numerical implementation can make this method more efficient and probably competing with ZPETC for higher-order (minimum phase) real world systems.

References

[1] Chang F (1997)
Learning Control and Setpoint Design (Application to a Wafer Stepper)
Delft University of Technology, Department of Mechanical Engineering and Marine technology, Systems and Control Group, Delft

[2] Tomizuka M (1986)
Zero Phase Error Tracking Algorithm for Digital Control
ASME Journal of Dynamic Systems, Measurement, and Control
Vol. 109, No. 3, pp. 65-68

[3] Carlsson B, Söderström T and Ahlén A (1987)
Digital differentiating filters
Teknikum, Institute of technology, Uppsala University

[4] Carlsson B (1989)
Digital differentiating filters and Model based fault detection
Acta Univ. Ups. Uppsala Dissertations from the Faculty of Science 28, 215pp.,
Uppsala

[5] Bélanger P.R (1995)
Control Engineering: a modern approach
McGill University, Saunders College Publishing

[6] Signal Processing Toolbox Users Guide (1999)
Revised for Version 4.2 (Release 11)
The MathWorks, Inc.

A Basic differentiating FIR filter

Phase-properties determine the quality of differentiating filters especially for higher order derivatives. To guarantee linear phase, filter $H(q)$ (Eq. 8) gets some constraints. First we make the filter symmetric: $N_1 = N_2 = N$. For odd derivative ($R = 1, 3, \ldots$) filters we set $h_n = -h_{-n}$. The filter becomes:

$$\begin{aligned} H_R(q) &= \sum_{n=0}^{N} h_n(q^n - q^{-n}) = \sum_{n=1}^{N} h_n(q^n - q^{-n}) \\ H_R(\omega) &= \sum_{n=1}^{N} h_n(e^{ni\omega T} - e^{-ni\omega T}) = 2i\sum_{n=1}^{N} h_n(sin(n\omega T)) \end{aligned} \tag{24}$$

This filter has an anti-symmetric impulse response. Note that we get the exact phase-shift as in Eq. 5. Filter coefficient $h_0 = 0$, so the discrete data point itself isn't responsible for its own odd derivatives.

For even derivative ($R = 2, 4, \ldots$) filters we set $h_n = h_{-n}$, so:

$$\begin{aligned} H_R(q) &= \sum_{n=0}^{N} h_n(q^n + q^{-n}) \\ H_R(\omega) &= \sum_{n=0}^{N} h_n(e^{ni\omega T} + e^{-ni\omega T}) = 2(\sum_{n=1}^{N} h_n(cos(n\omega T)) + h_0) \end{aligned} \tag{25}$$

Now we get a pure real tranfer function of the filter and again the phase properties as in Eq. 5 are guaranteed. Note this FIR filter has an symmetric impulse response. Because sample time T is not mentioned yet, the filter coefficients $h_0, \ldots, h_N$ will depend on this sample time. We can present a general differentiating FIR filter (as in [4]) with filter coefficients which don't include sample time T:

$$H_R(q) = \frac{1}{2T^R}(\sum_{n=1}^{N} c_n(q^n + (-1)^R q^{-n}) + c_0) \tag{26}$$

B Truncated Shannon-based filter

According to the sampling theorem we can reconstruct a continuous-time series $s_c(t)$ exactly from the discrete-time values $s(k)$ (also called Shannon reconstruction):

$$s_c(t) = \sum_{j=-\infty}^{\infty} \Phi_j(t)s(j) \tag{27}$$

where

$$\Phi_j(t) = \begin{cases} 1 & \text{if } t = jT \\ \frac{\sin(\pi(\frac{t-jT}{T}))}{\pi(\frac{t-jT}{T})} & \text{if } t \neq jT \end{cases} \tag{28}$$

This function is also known as the Dirichlet, *periodic sinc* or *aliased sinc* function. By differentiating Eq. 27 we get:

$$\dot{s}_c(t) = \sum_{j=-\infty}^{\infty} \dot{\Phi}_j(t)s(j) \tag{29}$$

We are only interested in values of the derivatives in the sampling points, so Eq. 29 becomes:

$$d(k) = \sum_{j=-\infty}^{\infty} \dot{\Phi}_j(kT)s(j) \tag{30}$$

Differentiating $\Phi_j(t)$ (Eq. 28)

$$\dot{\Phi}_j(t) = \begin{cases} 0 & \text{if } t = jT \\ \frac{\cos\left(\pi\left(\frac{t-jT}{T}\right)\right)}{t-jT} - \frac{\sin\left(\pi\left(\frac{t-jT}{T}\right)\right)}{\frac{\pi}{T}(t-jT)^2} & \text{if } t \neq jT \end{cases}$$

and evaluating the result in the sample-points kT gives:

$$\dot{\Phi}_j(kT) = \begin{cases} 0 & \text{if } t = jT \\ \frac{(-1)^{(k-j)}}{T(k-j)} & \text{if } t \neq jT \end{cases} \tag{31}$$

So the first-order derivative signal becomes

$$d^1(k) = \sum_{j=-\infty}^{k-1} \frac{(-1)^{k-j}}{T(k-j)} s(j) + \sum_{j=k+1}^{\infty} \frac{(-1)^{k-j}}{T(k-j)} s(j)$$

Now we substitute $n = k - j$:

$$d^1(k) = \sum_{n=1}^{\infty} \frac{(-1)^n}{nT} s(k-n) + \sum_{n=-\infty}^{-1} \frac{(-1)^n}{nT} s(k-n)$$

Because $\dot{\Phi}_j(t)$ is odd symmetric around $t = kT$ (which means $\dot{\Phi}_j(-n) = -\dot{\Phi}_j(n)$) we can write:

$$\begin{aligned} d^1(k) &= \sum_{n=1}^{\infty} \frac{(-1)^n}{nT} (s(k-n) - s(k+n)) \\ &= \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{nT} (s(k+n) - s(k-n)) \end{aligned} \tag{32}$$

Denoted as filter 8 with shift-operator q, we can denote the ideal reconstructive differentiating filter:

$$H(q) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{nT} (q^n - q^{-n}) \tag{33}$$

The filter coefficients become:

$$h_n = \begin{cases} 0 & \text{if } n = 0 \\ \frac{(-1)^n}{nT} & \text{if } n \neq 0 \end{cases} \tag{34}$$

By replacing $q = e^{i\omega T}$ we can calculate the frequency response of the filter. We can proof (found in [4]) that this frequency response equals the response of the ideal differentiator (Eq. 5 with R=1), expanding it into a Fourier series.

We can build higher derivative filters by differentiating $\Phi_j(t)$ (Eq. 28) several times. If we evaluate these derivatives of the Dirichlet function only in the sample points (as in Eq. 31) we get for odd derivatives:

$$c_n = \begin{cases} 0 & \text{if } n = 0 \\ \sum_{r=0}^{\frac{R-1}{2}} \frac{(-1)^{R+r+1} R!}{(2r+1)!} \frac{2\pi^{2r}(-1)^n}{n^{R-2r}} & \text{if } n \neq 0 \end{cases} \tag{35}$$

For even derivative filters:

$$c_n = \begin{cases} (-1)^{\frac{1}{2}R} \frac{\pi^R}{R+1} & \text{if } n = 0 \\ \sum_{r=0}^{\frac{R}{2}-1} \frac{(-1)^{R+r+1} R!}{(2r+1)!} \frac{2\pi^{2r}(-1)^n}{n^{R-2r}} & \text{if } n \neq 0 \end{cases} \tag{36}$$

C M-files

C.1 invsim.m

```
function u=invsim(z,p,k,y,t)
% INVSIM evaluates the inverse of the SISO system H(s), specified
% by the continuous-time zero-pole-gain (ZPK) model SYS with zeros Z,
% poles P, and gain K. The ouput series of the system H(s), which
% now is the input, is specified by Y and time series T, which must
% be equally spaced.
%
% U=INVSIM(Z,P,K,Y,T)
%
%
%                  (s - b )(s - b  )... (s - b )
%          Z             1       1-1          0
% H(s)= K --- = K  -------------------------------
%          P
%                  (s - a )(s - a  )... (s - a )
%                       k       k-1          0
%
% We can prescribe initial and end values of the input U and it's
% derivatives in columnvectors U1 and U2. Since the number of boundary
% conditions is l, the order of the numerator, U1 and U2 have length l/2.
% If l is odd, length U1 must be round(l) and length U2 round(l)-1. U1 and
% U2 may be longer but the redundant values won't be used. If U2 or U1 is an
% empty matrix, we only get contraints on respectively begin or end values of
% the input U.
%
% RELTOL and ABSTOL prescribe the relative and absolute error tolerance
% for solving the differential equations. Defaults are respectively 1e-11
% and 1e-6.
% The optional parameter RES sets the root resolution: all coefficients
% smaller than res will be neglected (set to 0). The default value of
% RES is 1e-14.
%
% First the transfer function is split up in a true non-causal (tnc)
% and a (unstable) causal part. The true non-causal part can be
% seen as a summation of several differentiators of different
% degrees. Some decent(non-causal) discrete time differentiating
% filters are used to evaluate the system. Since the causal part
% may be unstable, a shooting method (MUS) is used to evaluate the
% system in an accurate way. By using MUS, a two-point boundary value
% solver, we can put extra constrains on begin and end values of the
% output signal.

%   Author: Maurice Schneiders
%   Date:   July 2000

global A B t_y y_tnc bcv

% check input dimensions
if size(z,1)~=1 & size(z,2)~=1
   error('Z has to be a vector')
elseif size(p,1)~=1 & size(p,2)~=1
   error('P has to be a vector')
elseif size(k,1)~=1 | size(k,2) ~=1
   error('K has to be a scalar')
elseif size(t,1)~=1 & size(t,2)~=1
   error('Timeseries T has to be a vector')
```

```
elseif size(y,2)~=1
   error('Dataseries Y has to be a rowvector')
end

if size(z,2)==1
   z=z';
end

if size(p,2)==1
   p=p';
end

if size(t,1)==1
   t=t';
end

if size(y,1)==1
   y=y';
end

%   Check equidistanceness of timeseries
if max(diff(diff(t)))>10*eps
   error('Timeseries T has no equidistantial spacing')
end

T   =   t(2)-t(1);

% Check for causal system
if length(z)>length(p)
   error('Zero-pole-gain (ZPK) system is not causal')
end

% calculate inverse model
z_inv   =   p;
p_inv   =   z;

% split up causal and non-causal part (causal part (just) proper if possible)
% use smallest (absolute value) zeros for non-causal part
m=length(z_inv)-length(p_inv);
z_inv=z_inv+eps*i;
z_inv=sort(z_inv);
z_inv=z_inv-eps*i;

if imag(z_inv(m))==0
   z_tnc=z_inv(1:m);
   z_c=z_inv(m+1:end);
else
   if m==1
      z_tnc=z_inv(1:2);
      z_c=z_inv(3:end);
   elseif abs(z_inv(m))==abs(z_inv(m-1))
      z_tnc=z_inv(1:m);
      z_c=z_inv(m+1:end);
   else
      z_tnc=z_inv(1:m+1);
      z_c=z_inv(m+2:end);
   end
end

tnc=poly(z_tnc);
% evaluate true non-causal part
y_tnc=tncsim(tnc,y,T);
```

```
% evaluate cuasal part

% scaling system
scale=round(log10(mean(abs(p_inv))));
c=10^(-scale);
z_c=z_c*c;
p_inv=p_inv*c;
t_y=t/c;
% to state space notation
[A,B,C,D]=zp2ss(z_c,p_inv,1);
% instead of this we can use tf2ssic if we want to prescribe
% initial conditions out of begin and end conditions of the system

n=size(A,1);
% boundary conditions
bcv=[-[1:n]' zeros(n,1)];

% run the NAG-algorithm
% order of pol.app. is k1-1
% try different orders and compare results!
k1=44;
% number of collocation points
kp=1000;
% number of output points
npoints=length(t_y);
% initial time
x0=t_y(1);
% final time
x1=t_y(end);
nsamp=length(t_y);
save bvpdat.mat n k1 kp npoints x0 x1 A B t_y y_tnc bcv nsamp
tic
!bvp
toc
load bvpsol.mat
u   =   [C*y + D*y_tnc']';
u=u*c^(length(p_inv)-length(z_c))/k;
```

C.2 tncsim.m

```
function y=tncsim(tnc,u,T,N,method,method_parameters)
% TNCSIM simulates true non-causal system H(s).
%
% Y=TNCSIM(TNC,U,T,N,METHOD,METHOD_PARAMETERS)
%
%                        m       m-1
% H(s) = TNC(S) =   c s  + c  s  + ... + c
%                    m      m-1           0
%
% TNC = [c   c ...   c ]
%        m   m-1     0
%
% T is the sampletime of the inputseries U (a columvector).
% Optional we can set the filter order N for each differentiating
% filter. We can also set METHOD and corresponding METHOD_PARAMETERS
% for each differentiation.
% That's why N, METHOD and METHOD_PARAMETERS are all columvectors
% with size(NTC)-1.
%
```

```
%                    m                 m-1
% N = [filter_order_s  ; filter order_s ; ......; filter_order_s]
%
%
%                   m          m-1
% METHOD = [method_s ;  method_s ; ......; method_s]
%
%
%                                  m            m-1
% METHOD_PARAMETERS = [meth_para_s ; meth_para_s ; ......; meth_para_s]
%
% For information on these options see function BDIFFILT.M. Default filter
% order for all filters is 8. Default method is Polynomial Interpolation 'PInt'
% ('PInt' has no method_parameters).

%   Author: Maurice Schneiders
%   Date:   Julye 2000

if nargin<6
   if nargin<5
      if nargin<4
            N   =   8*ones(size(tnc,2)-1,1);
      end
      method = [];
      for k=1:length(tnc)-1
         method=[method; 'PInt'];
      end
   end
   method_parameters = zeros(size(tnc,2)-1,1);
end

% check inputdimensions
if size(tnc,1)~=1
   error('TNC has to be a rowvector')
elseif size(u,2)~=1
   error('U has to be a columvector')
elseif size(T,1)~=1 | size(T,2)~=1 | T(1,1)<=0
   error('Invalid value of sampletime T')
elseif size(N,2)>1
   error('N must be a columvector')
elseif size(N,1)~=length(tnc)-1
   error('Vector N must have length(TNC)-1')
elseif size(method,2)~=4
   error('METHOD must be a columvector and method must hold 4 characters')
elseif size(method,1)~=length(tnc)-1
   error('Vector METHOD must have length(TNC)-1')
elseif size(method_parameters,2)>1
   error('METHOD_PARAMETERS must be a columvector')
elseif size(method_parameters,1)~=length(tnc)-1
   error('Vector METHOD_PARAMETERS must have length(TNC)-1')
end

y   =   zeros(size(u));

for m=1:length(tnc)-1
   R    =   length(tnc)-m;
   C    =   bdiffilt(R,N(m),method(m,:),method_parameters(m,:));
   y    =   y+tnc(m)*diffilt(u,C,R,T);
end

y   =   y+tnc(end)*u;
```

C.3 bdiffilt.m

```
function C=bdiffilt(R,N,METHOD,method_parameters)
% BDIFFILT calculates filtercoefficients C to build a non-causal
% differentiating N-th order filter. R sets the order of the derivative
% which can be calculated with the filter. Use the filtercoefficients
% with function DIFFILT.M.
%
% C=BDIFFILT(R,N,METHOD,METHOD_PARAMETERS)
%
% C=[c0 c1 c2 c3....cN]', used in filter H (z):
%
%
%  R          1      N      n     -n
% H (z) =   ------sum   cn(z  -  z   ) for odd-derivatives, and
%           2 T^R   n=1
%
%
%
%  R           1        N      n     -n
% H (z) =   ------( sum   cn(z  +  z   ) + c0) for even-derivatives.
%           2 T^R    n=1
%
%
% Phase-shift is as the ideal R-th order differentiatingfilter +90*R degrees.
% Amplitude can be optimized with the following cost-function:
%
%       pi/T  R        R     2
% E = int | Hd (iw) - H (z) | dw
%       0
%
% Hd(iw) is the 'ideal' R-th order differentiator. This function will be minimized
% respect to the constraints:
%
% W*C=e;
%
%
% METHOD 'TISD': Truncated Ideal (Shannon-reconstruction) Differentiator (no method_parameters)
%                                                R         R
%               This method minimizes E with Hd (iw)= (iw) for 0<w<pi/T. This means the same
%               as using the R-th order derivative of the Dirichlet-funtion to calculate the
%               derivative.
%               No supplemental constraints are used.
%
% METHOD 'PInt': Polynomial Interpolation (degree 2N from 2N+1 data-points)
%
%               Note that N >= 0.5R in order to get a working filter. Fitting a polynomial
%               of degree 2N is the same as using W*C=e with the following 2*N equations:
%
%                    r                  r
%                   d      R     |     d        R |
%                   -- r  H (z)  | =   -- r  (iw) |        for r=0,1,...,2*N
%                   d w          |w=0  d w        |w=0
%
%               This leads to N usable equations (or N+1 in case R is even to calculate c0)
%
%
% METHOD 'LSPF': Least Squares Polynomial Fitting (degree 2P from 2N+1 data-points)
%               (method_parameters: P)
%
%               The same as polynomial interpolations, but now P < N. So some smoothing through
%               the samplepoints uccur which minimizes noise-transmission. A good choice of P has
```

```
%              to be made to get a working filter. Note that N >= 0.5R still holds. Using the
%              following 2P equations:
%
%                  r                  r
%                  d      R     |     d         R |
%                  -- r  H (z)  | =   -- r   (iw) |        for r=0,1,...,2*P
%                  d w          |w=0  d w         |w=0
%
%                                                                   T     T -1
%              gives P usable equations with rank(W)=N, so C = W *(W*W ) * e, to calculate the
%              least squares polynomial fit of polynomial with degree 2P.
%

%   Author: Maurice Schneiders
%   Date:   July 2000

if nargin==3
   method_parameters = 0;
end

% check input dimensions
if size(R,1)~=1 | size(R,2)~=1 | R(1,1)<=0
   error('Invalid value for derivative order R')
elseif size(N,1)~=1 | size(N,2)~=1 | N(1,1)<=0
   error('Invalid value for filter order N')
elseif size(METHOD,2)~=4 | size(METHOD,1)~=1
   error('METHOD must hold 4 characters')
elseif size(method_parameters,1)~=1
   error('METHOD_PARAMETERS must be a scalar or rowvector')
end

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Truncated Ideal (Shannon-reconstruction) Differentiator                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
if METHOD=='TISD'

   if R/2==round(R/2)
      % r is even
      r_max =   R - 2;
   else
      % r is odd
      r_max =   R - 1;
   end

   C=zeros(N+1,1);
   n=[1:1:N]';

   % calculate the R-th order Dirichlet(sinc)-function in N samplepoints
   for r=0:2:r_max
      C(1:N,1)=C(1:N,1)+2*(-1)^(R+1+0.5*r)*factorial(R)/factorial(r+1)*(-1).^n./(pi^(-r)*n.^(R-r));
   end

   c0=real((-1)^(0.5*R)/(R+1)*(pi)^R);
   C=[c0;C];
   %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Polynomial Interpolation                                                                     %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %
elseif METHOD=='PInt'
```

```
   if R > 2*N
      error('Filter order is too small to calculate a R-th derivative filter (N >= 0.5R)')
   end

   e     =   [];
   W     =   [];

   if R/2==round(R/2) % r is even

      for r =   0:2:2*N
         if r==R
            e   =   [e; factorial(r)];
         else
            e   =   [e; 0];
         end
         W   =   [W; [1:N].^r];
      end
      W = [[1; zeros(size(W,1)-1,1)] W];
      C =   W\e;

   else % r is odd

      for r =   1:2:(2*N-1)
         if r==R
            e   =   [e; factorial(r)];
         else
            e   =   [e; 0];
         end
         W   =   [W; [1:N].^r];
      end
      C =   W\e;
      C =   [0; C];

   end

   %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Least Squares Polynomial Fitting                                                 %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %
elseif METHOD=='LSPF'

   P     =   method_parameters;
   e     =   [];
   W     =   [];

   if R > 2*P
      error('Filter order is too small to calculate a R-th derivative filter (P >= 0.5R)')
   end

   if R/2==round(R/2) % r is even

      for r =   0:2:2*P
         if r==R
            e   =   [e; factorial(r)];
         else
            e   =   [e; 0];
         end
         W   =   [W; [1:N].^r];
      end
      W = [[1; zeros(size(W,1)-1,1)] W];
      C =   W'*inv(W*W')*e;
```

```
   else % r is odd

      for r =   1:2:(2*P-1)
         if r==R
            e   =   [e; factorial(r)];
         else
            e   =   [e; 0];
         end
         W  =   [W; [1:N].^r];
      end
      C =   W'*inv(W*W')*e;
      C =   [0; C];

   end
else
   error('Settign for METHOD is not correct')
end
```

C.4 diffilt.m

```
function y=diffilt(x,C,R,T)
% DIFFILT calculates the R-th order derivative Y of signal X,
% using filtercoefficients C assuming that signal X was
% sampled with sampletime T
%
% Y=DIFFILT(X,C,R,T)
%
% C=[c0 c1 c2 c3....cN]', used in filter H (z):
%
%
%  R          1      N      n     -n
% H (z) =   ------sum   cn(z  -  z   ) for odd-derivatives, and
%           2 T^R   n=1
%
%
%
%  R           1        N      n     -n
% H (z) =   ------ (sum   cn(z  +  z   ) + c0) for even-derivatives.
%           2 T^R     n=1
%
% See BDIFFILT.M for more information.
%
% To cancel border distortions, polynomial expansion is used.
% N is the filter order so also the length of the expansion.
% R is the derivative to be determined, so a polynomial of order of the
% expansion must be >=R. We get the best overall results if we use a polynomial
% of order R.

%   Author: Maurice Schneiders
%   Date:   July 2000

% check input dimensions
if size(x,2)~=1
   error('Dataseries x has to be a columvector')
elseif size(C,2)~=1
   error('Coefficientvector C has to be a columvector')
elseif size(R,1)~=1 | size(R,2)~=1 | R(1,1)<=0
   error('Invalid value for derivative order R')
elseif size(T,1)~=1 | size(T,2)~=1 | T(1,1)<=0
```

```
   error('Invalid value for sampletime T')
end

L   =   size(x,1);
N   =   size(C,1)-1;

% Polynomial expansion of dataseries

pd_b            =  polyfit([1:R+1]',x(1:(R+1)),R);
pd_e            =  polyfit([1:R+1]',x(end-R:end),R);

xe_b            =   polyval(pd_b,[(-N+1):0]');
xe_e            =   polyval(pd_e,[R+2:R+2+N-1]');
xn              =   [xe_b; x; xe_e];
y   =   zeros(size(x));

if R/2==round(R/2)
   % r is even

   for l    =   1:1:L
   y(l,1)=sum(C.*(flipud(xn(l:l+N))+xn(l+N:l+2*N)));
    end

else
   % r is odd

   for l    =   1:1:L
   y(l,1)=sum(C.*(-flipud(xn(l:l+N))+xn(l+N:l+2*N)));
    end

end

y   =   y./(2*T^R);
```

C.5 tf2ssic.m

```
function [A,B,C,D,x0]=tf2ssic(num,den,u0,y0,res,method)
% TF2SSBC Converts transfer function H(s) to state-space representation.
% Initial state values can be calculated by prescribing initial values of input
% and output and their derivatives.
%
% [A,B,C,D,X0] = TF2SSIC(NUM,DEN,U0,Y0,RES,METHOD)
%
%                     l       l-1
%                  b s  + b  s  + ... + b
%        NUM(S)     l      l-1           0
% H(s)---------- = ------------------------
%        DEN(S)       k       k-1
%                  a s  + a  s  + ... + a
%                   k      k-1           0
%
% METHOD specifies the realisation of the state-space model:
%
% 'Ctr' : Controllable canonical form (default)
% 'Obs' : Obsevable canonical form
%
% u0 is a columvector containing the initial conditions
% for the input signal and it's derivatives up to order k-1:
%                    k-1
```

```
% [u(t=0) u'(t=0)...u (t=0)]'
%
% y0 is a columvector containing the initial conditions
% for the input signal and it's derivatives up to order l-1:
%                     l-1
% [y(t=0) y'(t=0)...y (t=0)]'
%
% The optional parameter RES sets the root resolution: all coefficients
% smaller than res will be neglected (set to 0). The default value of
% RES is 1e-14.

%   Author: Maurice Schneiders
%   Date:   July 2000

if nargin==6 & size(method,2) ~= 3
   error('The prescribed method is not specified')
end

if nargin<6
   if nargin<5
      res   =   1e-14;
   end
   method = 'Ctr';
end

if size(res,1)~=1 | size(res,2)~=1 | res(1,1)<0
   error('Invalid value of RES')
end

% check resolution and strip leading zeros from numerator
for k=1:length(num)
   if abs(num(k))<res
      num(k)=0;
   end
end
inz = find(num ~= 0);
num = num(inz(1):end);

% check resolution and strip leading zeros from denominator
for l=1:length(den)
   if abs(den(l))<res
      den(l)=0;
   end
end
inz = find(den ~= 0);
den = den(inz(1):end);

% check input dimensions
if size(u0,2) > 1 | size(y0,2) > 1
   error('u0 and y0 must both be columnvectors.')
elseif size(num,2)~=size(u0,1)+1 | size(den,2)~=size(y0,1)+1
   error('Number of initial conditions does not match with order of numerator or denumerator')
elseif den(1,1)==0
   error('First element of den must not equal zero.')
elseif size(num,1)~=1 | size(den,1)~=1
   error('Num and then must both be rowvectors.')
elseif size(num,2) > size(den,2)
   error('System is non-causal cannot be converted to a state space model.')
end

n   =   length(den);

num = [zeros(1,n-length(num)) num];
```

```
u0  = [u0; zeros(length(y0)-length(u0),1)];

a   =   num(2:n)/den(1);
b   =   den(2:n)/den(1);

if method=='Ctr'
   D    =   num(1)/den(1);
   C    =   [a - D*b];
   A    =   [-b ; eye(n-2,n-1)];
   B    =   [eye(n-1,1)];
elseif method=='Obs'
   D    =   num(1)/den(1);
   B    =   [a-D*b]';
   A    =   [-b' eye(n-1,n-2)];
   C    =   [eye(1,n-1)];
else
   error('Obsolete method-parameter')
end

% express initial values of y and u (and derivatives) in x:
%
% Obs*x0    =   y0 - L*u0

Obs =   zeros(n-1,n-1);
L       =   Obs;

for r=0:n-2

   Obs(r+1,:)    =   C*A^r;

   for s=1:n-1
      if r-s >= 0
         L(r+1,s)    =   C*A^(r-s)*B;
      else
         L(r+1,s)    =   0;
      end
   end
end

L       =   L + D*eye(n-1,n-1);

x0  =   Obs\(y0-L*u0);
```