DRFC/CAD EUR-CEA-FC-1409

Identification of systems with distributed parameters

J.-M. MORET

october 1990

CEA EURATOM

ASSOCIATION EURATOM-C.E.A.
DEPARTEMENT DE RECHERCHES SUR LA FUSION CONTROLEE
C.E.N./CADARACHE
13108 SAINT PAUL LEZ DURANCE CEDEX

Identification of systems with distributed parameters

J.-M. Moret
Association Euratom CEA sur la Fusion Contrôlée
Centre d'Etudes Nucléaires de Cadarache B.P. n° 1
13108 Saint-Paul-lez-Durance Cédex, France.

The problem of finding a model for the dynamical response of a system with distributed parameters based on measured data is addressed. First a mathematical formalism is developed in order to obtain the specific properties of such a system. Then a linear iterative identification algorithm is proposed that includes these properties, and that produces better results than usual non linear minimisation techniques. This algorithm is further improved by an original data decimation that allow to artificially increase the sampling period without losing between sample information. These algorithms are tested with real laboratory data.

1. INTRODUCTION

System identification deals with the problem of finding mathematical models for dynamical system based on measured data. The area has matured mainly as a step in the conception of high performance automatic control. In this context it aims at determining in an adequate form the transfer function of a real system without any a priori knowledge of the underlying physical model, either that such a modelling would be too inaccurate or too difficult to establish.

This technique has found another field of application in experimental physics : it can be used as a tool for investigation of dynamical processes. In this approach, the dynamical response of the system is first identified from measurements of the temporal relaxation following an external perturbation. The properties of this experimentally established dynamical response are then used to build a physical model for the system. They allow for instance to discard or retain some given models or physical processes and to quantify their characteristic coefficients.

A privileged example is the study of particle and heat transport in Tokamak plasmas for which the method presented in his paper has been developed. Up to now this problem has defied any acceptable explanation. System identification will in this case provide a blind and therefore less biased approach to the problem [1].

In the framework of automatic control, the present state of the art in system identification consists in a collection of well established and well understood methods [2]. All of them apply only to lumped parameter systems, from the simple single-input single-output system to the more general multi-input multi-output system. Lumped parameter systems are described by a finite set of discrete inputs and outputs. This covers most of the applications in electrical and mechanical engineering (robotics, aircraft and sheep control, tracking, etc.). Other systems however involve distributed parameters, that is parameters that are no more discrete values, but

functions of one or more continuous variables. This is the case for instance for heat transfer, particle diffusion, chemical reactions and others. This paper is devoted to these systems.

In the next section, we will first give the transfer function of a system with distributed parameters an ad hoc formalism. Section 3 will present an iterative algorithm for system identification. This algorithm is also applicable to lumped parameter systems but is especially well suited to overcome the numerical difficulties inherent to systems with distributed parameters. In section 4 we will propose a modification of this algorithm in order to improve the bad conditioning of the problem. Appendix I recalls the basics of the z-transfer functions. Interested readers will find in Appendix II an implantation of the algorithms proposed in this report.

The exposed methods will be tested and presented with real laboratory signals extracted from the Tore Supra Tokamak data base [3]. These signals represent the relative perturbation of the plasma electron temperature arising after the injection of a solid hydrogen pellet. The measurements are performed in different locations in the plasma discharge. The ablation duration (~ 500 μs) is shorter than the sampling period (1024 μs), allowing to consider the stimulus signal as a delta function. Apart from instrumental uncertainties, noise also contains a contribution from quasi periodic plasma oscillations. This phenomena is outside the scope of the present study and will be considered as noise.

2. FORMALISM

In this section, we will develop a formalism for the description of the dynamical response of systems with distributed parameters. This formalism will give the transfer function of such systems a form tractable by usual identification methods. In addition, specific properties of these transfer functions will be established, which have to be included in the identification procedure.

Let us first treat the simpler case of a single-input single-output lumped parameter system such as one of those listed in the introduction. If linear, such a system is described by an ordinary differential equation

$$\sum_{n=0}^{N} \alpha_n \frac{d^n y(t)}{dt^n} = \sum_{m=0}^{M} \beta_m \frac{d^m x(t)}{dt^m} . \qquad (1)$$

If in addition the coefficients α_n and β_m in the equation are time independent, the system is stationary and its transfer function can be expressed as a rational function in the Laplace variable :

$$H(s) = \frac{y(s)}{x(s)} = \frac{\sum_{m=0}^{M} \beta_m s^m}{\sum_{n=0}^{N} \alpha_n s^n} . \qquad (2)$$

It is well known that time discretisation of this transfer function yields to an analogous expression in the z-variable which can be favourably used for identification purposes (see Appendix I).

For systems with distributed parameters, the signals x and y are no more discrete values depending only on the time, but are functions of one or more continuous variables. In the most common situations, this variable is the spatial position, but it could also be the particle momentum of a distribution function, the energy in neutron diffusion, the wave length in wave propagation problems, etc.. The system is then described by a partial differential equation. A few

examples are listed below :

Particle diffusion :

$$\frac{\partial n(\mathbf{r},t)}{\partial t} = S(\mathbf{r},t) + \nabla \cdot \left(D \nabla n(\mathbf{r},t)\right) \qquad (3)$$

n : particle density
S : particle source per unit time and volume
D : diffusion coefficient

Conductive heat transfer :

$$\frac{\partial(C\ T(\mathbf{r},t))}{\partial t} = Q(\mathbf{r},t) + \nabla \cdot \left(\kappa \nabla T(\mathbf{r},t)\right) \qquad (4)$$

T : temperature
Q : energy source per unit time and volume
C : specific heat
κ : heat conduction coefficient

Convective heat transfer :

$$\frac{\partial(C\ T(\mathbf{r},t))}{\partial t} = Q(\mathbf{r},t) + \nabla \cdot (\mathbf{v}\ C\ T(\mathbf{r},t)) \qquad (5)$$

$\mathbf{v}$: convection velocity

Magnetic field penetration :

$$\frac{\partial \mathbf{B}}{\partial t} = -\Phi - \nabla \times \left(\frac{1}{\mu_0 \sigma}(\nabla \times \mathbf{B})\right) \qquad (6)$$

$\mathbf{B}$: magnetic field
Φ : external flux source
σ : electric conductibility

All these examples can be written in the general form :

$$\frac{\partial y(\mathbf{r},t)}{\partial t} = x(\mathbf{r},t) + \mathcal{L}(\mathbf{r})\, y(\mathbf{r},t)\,, \tag{7}$$

where $x(\mathbf{r},t)$ is a source term and $\mathcal{L}(\mathbf{r})$ the transport operator. The source term plays here the same rôle as the input signal for a lumped parameter system. It is necessary to restrict our analysis to separable functions

$$x(\mathbf{r},t) = x(\mathbf{r})\, g(t)\,, \tag{8}$$

where $g(t)$ is a dimensionless function of the time and $x(\mathbf{r})$ gives the distribution of the source. $\mathcal{L}(\mathbf{r})$ is a linear differential operator with no explicit time dependance. The latter property is verified when the coefficients of the system are time independent, and allows together with the former to rewrite equation (7) in the Laplace variable as

$$s\,\frac{y(\mathbf{r},s)}{g(s)} = x(\mathbf{r}) + \mathcal{L}(\mathbf{r})\,\frac{y(\mathbf{r},s)}{g(s)}\,. \tag{9}$$

To get a tractable form, we will now develop the transfer function of the system in a Laurent series,

$$H(s) = \frac{y(\mathbf{r},s)}{g(s)} = \sum_{n=1}^{\infty} \frac{y_n(\mathbf{r})}{s - p_n}\,, \tag{10}$$

in which p_n and $y_n(\mathbf{r})$ respectively denote the poles and the residues of the transfer function. This can be understood as the expression of the impulse response as a sum of exponential decays :

$$h(\mathbf{r}, t\geq 0) = \sum_{n=1}^{\infty} y_n(\mathbf{r})\, e^{p_n t}\,. \tag{11}$$

One can easily verify that

$$s\,H(s) = \sum_{n=1}^{\infty} \left(\frac{p_n\, y_n(\mathbf{r})}{s - p_n} + y_n(\mathbf{r})\right). \tag{12}$$

Taking together equations (10) and (12) and the properties of $\mathcal{L}$, we get the Laurent series form of equation (9) :

$$\sum_{n=1}^{\infty} \left(\frac{p_n \, y_n(\mathbf{r})}{s - p_n} + y_n(\mathbf{r}) \right) = x(\mathbf{r}) + \sum_{n=1}^{\infty} \frac{\mathcal{L}(\mathbf{r}) \, y_n(\mathbf{r})}{s - p_n} . \qquad (13)$$

To be valid on the whole s-plane, this equation must be verified for each partial fraction of the sum. We conclude that the poles and the residues of the transfer function are respectively eigenvalues and eigenfunctions of the operator $\mathcal{L}$:

$$p_n \, y_n(\mathbf{r}) = \mathcal{L}(\mathbf{r}) \, y_n(\mathbf{r}) . \qquad (14)$$

The normalisation of the eigenfunctions is deduced from the source term so that it satisfies :

$$\sum_{n=1}^{\infty} y_n(\mathbf{r}) = x(\mathbf{r}) . \qquad (15)$$

One particular feature of the transfer function of a system with distributed parameters is that its poles depend neither on the continuous variables of the system nor on the source term distribution. This property must be adequately introduce in the identification procedure, as it will be seen in the next section.

As an illustrative example, we consider the radial conductive heat transfer in a cylinder of radius a. The basic description lies in equation (4) written in cylindrical geometry :

$$\frac{\partial y(r,t)}{\partial t} = x(r) \, g(t) + \frac{\kappa}{C} \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial y(r,t)}{\partial r} \right) \qquad (16)$$

and subjected to the limit condition y(a) = 0, where y represent the temperature and x the heat source. In this case the eigenvalue equation is :

$$p_n \, y_n(r) = \frac{\kappa}{C} \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial y_n(r)}{\partial r} \right) \qquad (17)$$

Its solutions are first kind Bessel functions :

$$y_n(r) = J_0\left(\alpha_n \frac{r}{a}\right), \tag{18}$$

$$p_n = -\frac{\alpha_n^2 \kappa}{a^2 C}, \tag{19}$$

where α_n are the zeros of J_0. We note here that the value of the poles increases rapidly :

$$\frac{p_2}{p_1} = 5.27, \frac{p_3}{p_1} = 12.9, \frac{p_4}{p_1} = 24.0 . \tag{20}$$

That shows that the system has sparse time constants and makes difficult the adequate choice of the sampling rate, as discussed in section 4.

For practical reasons linked both to the time discretisation and to the noise level of the data, we will limit the identification to the N first poles. According to equation (10) the transfer function of a system with distributed parameters can then be approximated by a rational function in the Laplace variable, the denominator of which being independent on the continuous parameters.

3. IDENTIFICATION ALGORITHM

In this section we will present an iterative linear method for system identification that turned out to be more robust than usual non linear minimisation methods. This method will be developed in the more simple framework of a single-input single-output lumped parameter system. It will be then extended to handle either multi-input systems or systems with distributed parameters.

We try to model a system whose input x and output y have been recorded over K samples, with a transfer function expressed in term of the z-variable as

$$H(z)= \frac{\sum_{m=0}^{M} b_m z^{-m}}{\sum_{n=0}^{N} a_n z^{-n}} = \frac{B(z)}{A(z)}. \qquad (21)$$

The knowledge of the noise characteristics is in our case of less interest. We then decide to define the optimal model on the basis of the error e between the effective output y and the modelled output $\hat{y}$:

$$e(k) = y(k) - \hat{y}(k) = y(k) - \frac{B(z)}{A(z)} x(k). \qquad (22)$$

The optimisation criterion is the usual minimisation of the output error variance, i.e. the loss function

$$J^2 = \frac{1}{K} \sum_{k=1}^{K} e^2(k). \qquad (23)$$

It is well known that this loss function is not a quadratic function in the system parameters. Its direct minimisation is therefore not straightforward. Methods using non linear minimisation techniques such as the Gauss-Newton algorithm exist [2]. For our particular data however they lead to a poor result, as illustrated in figure 1, and are not convenient for our purpose. This can be due either to the fact that one can not prevent these search methods

from reaching a local minimum or to the fact that the loss function curvature around its minimum is too small to be handled correctly within the numerical precision.

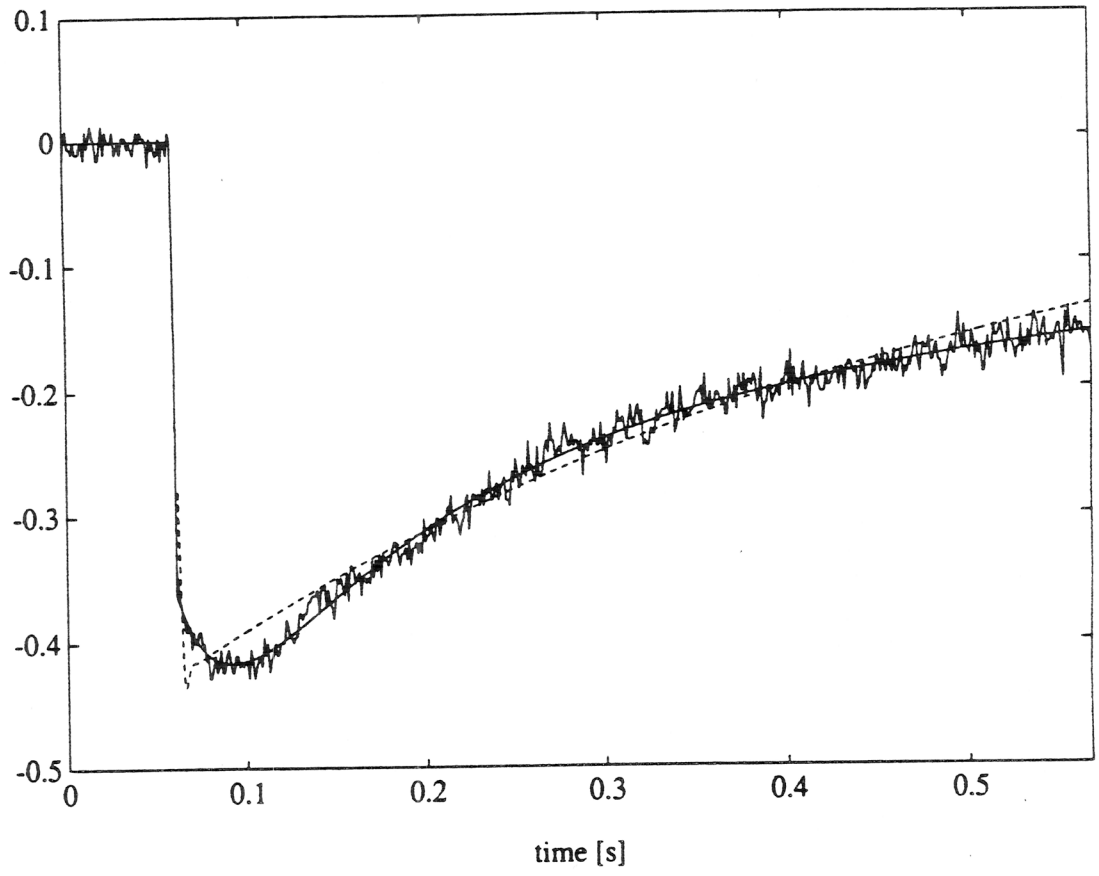

Fig. 1 : Identification of a third order z-transfer function with non linear minimisation of the loss function (dashed line) and with the proposed iterative algorithm (solid line).

To face up this difficulty, we propose the following algorithm. The evaluation of the output error, given explicitly by

$$e(k) = y(k) - \left(\sum_{m=0}^{M} b_m x(k-m) - \sum_{n=1}^{N} a_n \hat{y}(k-n) \right), \qquad (24)$$

can be slightly modified by substituting to the values of $\hat{y}$ the values of y :

$$e'(k) = \sum_{n=0}^{N} a_n y(k-n) - \sum_{m=0}^{M} b_m x(k-m) = A(z)\, y(k) - B(z)\, x(k) . \qquad (25)$$

In the last expression, the polynomials were normalised so that $a_0 = 1$. The model obtained by this way, known as an ARX model, has an advantage : its output error e' is a linear combination of the system parameters, the coefficients of the polynomials A(z) and B(z). The latter can therefore be estimated through an explicit matrix operation, by solving the linear over constraint system of equations :

$$y(k) = -\sum_{n=1}^{N} a_n y(k-n) + \sum_{m=0}^{M} b_m x(k-m) ; \quad k = N+1,\ldots,K . \qquad (26)$$

The drawback is the introduction of a bias in the parameter estimate [4]. This can be understood by noting that e and e' are linked by

$$e'(k) = A(z)\, e(k) . \qquad (27)$$

This shows that successive samples of e' depend on the present and past values of e and are therefore no more independent realisations of a random variable. This bias can be attenuated if we own an approximation for A(z), $A_0(z)$. In this case we can build a new error signal

$$A_0^{-1}(z)\, e'(k) = A(z)\, A_0^{-1}(z)\, e(k) = A(z)\, A_0^{-1}(z)\, y(k) - B(z)\, A_0^{-1}(z)\, x(k) \qquad (28)$$

which will be closer to e than e'. Therefore applying the ARX model to the new input-output signal pair $A_0^{-1}(z)\, x(k)$ and $A_0^{-1}(z)\, y(k)$ will yield to a less biased parameter estimate.

We are now ready to set up the following iterative procedure : starting with $A_0(z) = 1$, the most trivial choice, we perform a first estimation of A(z) and B(z) based on the signals x and y. Then $A_0(z)$ is replaced by what is deduced from this estimate for A(z) and the identification is applied to the signals $A_0^{-1}(z)\, x(k)$ and $A_0^{-1}(z)\, y(k)$. At each iteration, the newly estimated A(z) replaces $A_0(z)$. During this process the transmittances $A(z)\, A_0^{-1}(z)$ and $B(z)\, A_0^{-1}(z)$ will tend toward 1 and H(z)

respectively. This will transform equation (28) in equation (22), yielding to the sought result.

Our laboratory data set has been used to test the method. Figure 1 shows how it led to a more satisfactory identification than the non linear minimisation of the identically defined loss function. Table I presents for the same case some numerical results. One can note that non linear minimisation produces a pair of complex poles that has no physical ground. Figure 2 displays the system parameter estimate evolution along the iterative process.

		Non linear minimisation	Iterative algorithm
Loss function J		1.57 %	0.99 %
Denominator	a_1	-1.7808 ± 0.1410	-2.9505 ± 0.0037
	a_2	1.0915 ± 0.2401	2.9014 ± 0.0073
	a_3	-0.3094 ± 0.1055	-0.9509 ± 0.0037
Poles		0.9977	0.9990
		0.3916 + 0.3960 i	0.9916
		0.3916 - 0.3960 i	0.9599

Table I : Numerical result of the identification of a third order z-transfer function with non linear minimisation of the loss function and with the proposed iterative algorithm. The error bars are one standard deviation.

It is now easy to extend this method developed for a single-input single-output system to a system with distributed parameters. To do this we must rewrite the set of recurrent relations (26) for each of the L discrete measurements of the distributed output signal, keeping in mind that the transfer functions for all these signals have common poles, i.e. common denominators, but distinct numerators :

$$y_\ell(k) = -\sum_{n=1}^{N} a_n\, y_\ell(k-n) + \sum_{m=0}^{N-1} b_{\ell m}\, x(k-m)\ ;\ \ell = 1,\ldots,L\ ;\ k = N+1,\ldots,K\ . \qquad (29)$$

We have also taken into account the fact that the formalism developed in the previous section usually imposes M = N - 1. The definition of the loss function must also be adapted :

$$J^2 = \frac{1}{K\,L} \sum_{k=1}^{K} \sum_{\ell=1}^{L} \left(y_\ell(k) - \frac{B_\ell(z)}{A(z)}\, x(k) \right)^2 . \qquad (30)$$

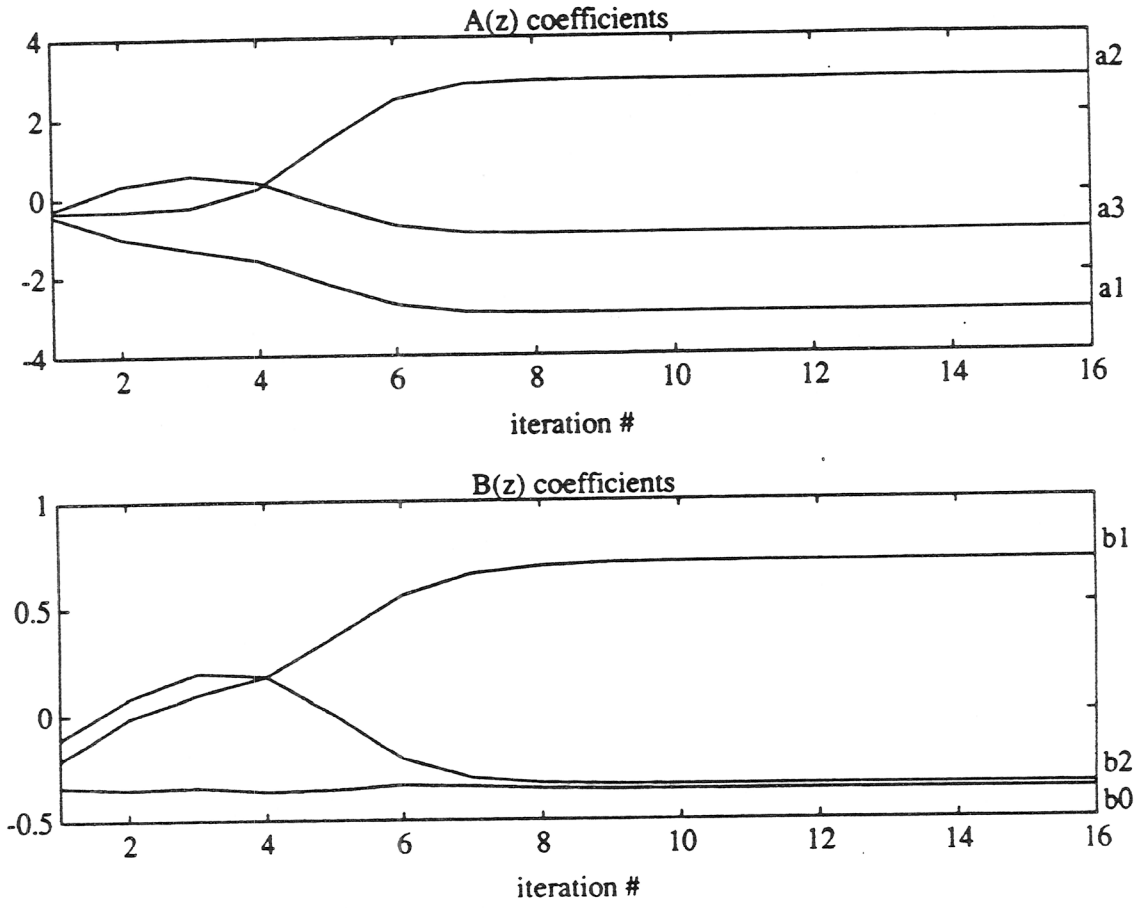

Fig. 2 : Evolution of the transfer function polynomial coefficients during the iterative process.

To illustrate this procedure, the identification has been carried out on five signals simultaneously. Figure 3 shows these signals and their respective estimates. In figure 4 (a), we have plotted the z-poles identified once independently for each signal, and once

as common poles for all measurements of the distributed output. To prove that both situations can not be statistically distinguished, we have drawn a similar plot for the loss function, figure 4 (b). The value of the loss function for each output in the common pole case is only slightly worse than the value obtained by independent identifications. The benefit of imposing common poles is a decrease in their variance, as it can be seen on figure 4 (a). This is due to the fact that by this mean more information can be used to identify a reduced set of free parameters.

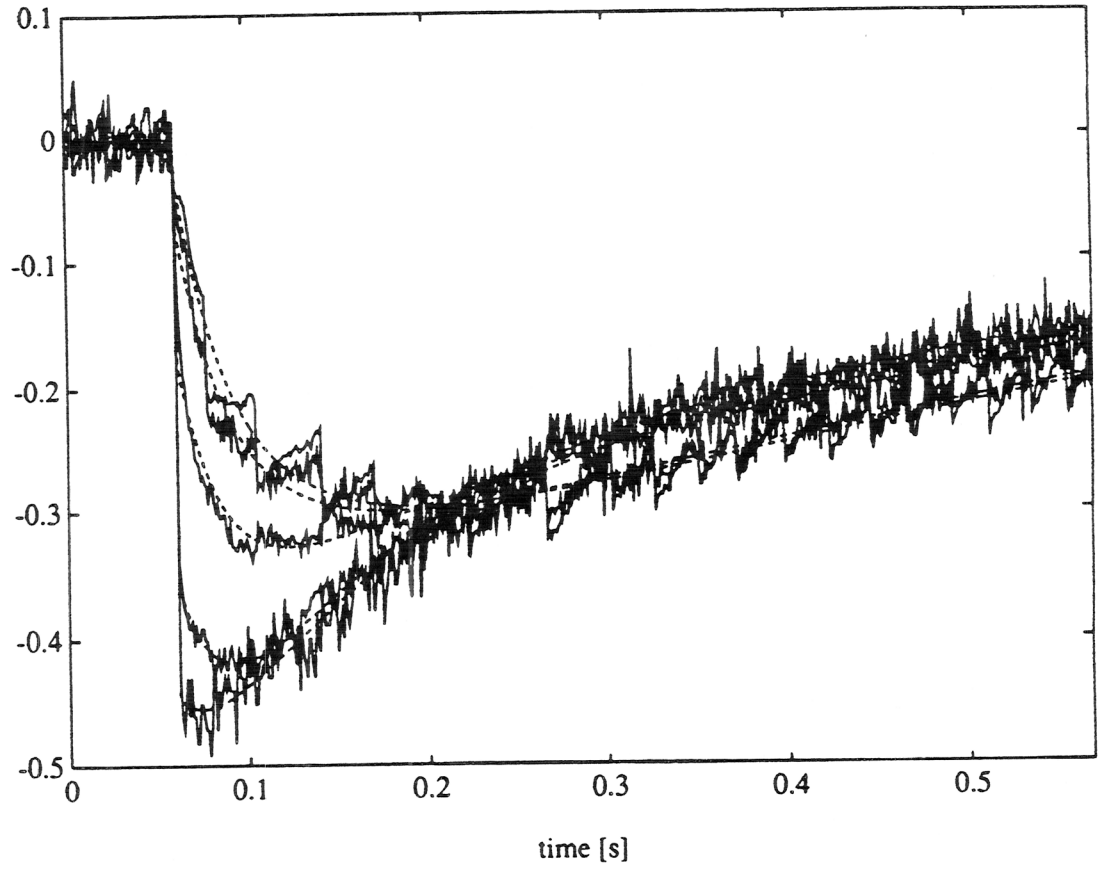

Fig. 3 : Identification of a third order transfer function for a system with a distributed output. Output is measured at five different locations. Poles are common to all these measurements.

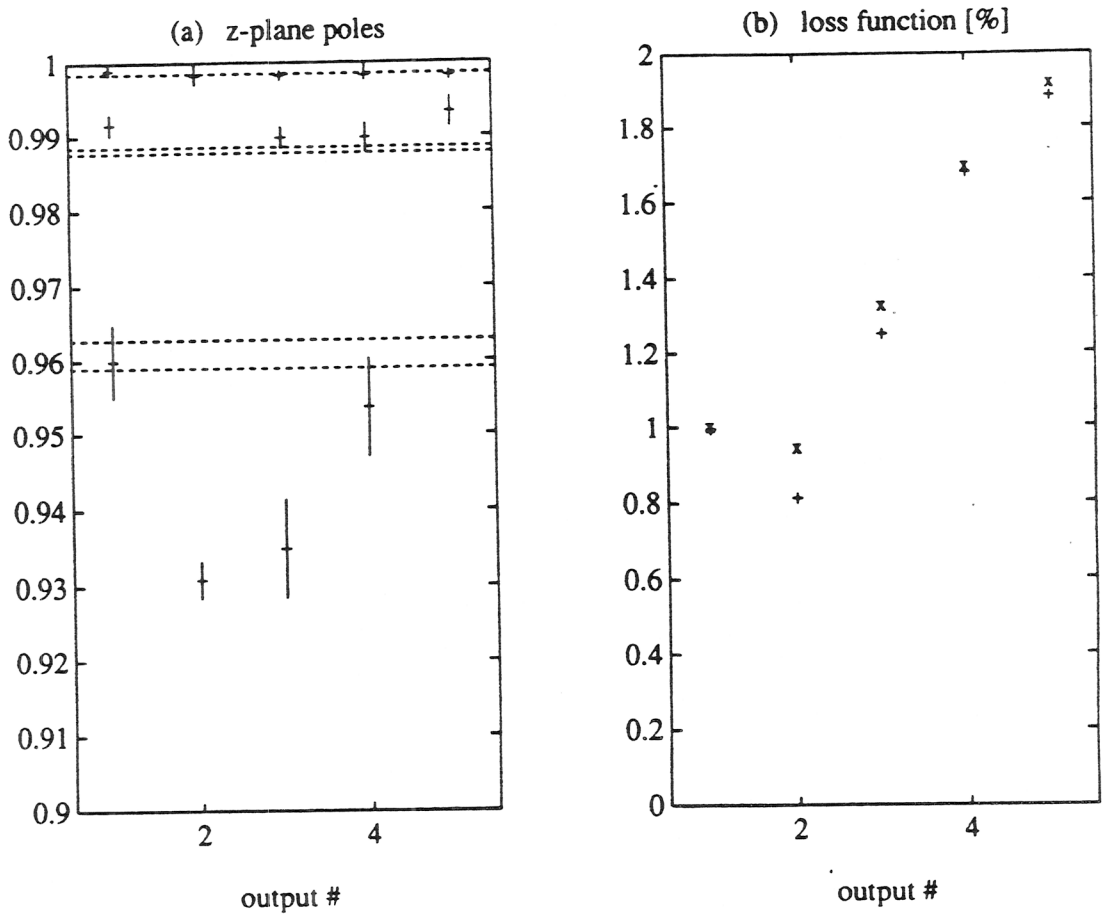

Fig. 4 : (a) Value of the identified z-plane poles for the case of the previous figure. Crosses are obtained with an independent identification for each output while dashed lines represent the one standard deviation confidence interval of the three common poles of a simultaneous identification.
(b) Value of the loss function J for each output obtained either with independent identification (+) or with a common pole identification (x).

4. DATA DECIMATION

We have seen both in the example treated in section 2 and in the real data identification of the previous section that a system with distributed parameters is usually characterised by sparse time constants. To give the smallest time constants the best chance to be correctly identified, it is obviously necessary to keep the sampling rate as fast as possible. The consequence of this choice is numerical difficulties in two operations. First the inversion of the correlation matrix necessary to minimise the loss function is ill conditioned. The reason is that two contiguous columns in the independent variable matrix used to build up this correlation matrix are formed with two successive samples in the data. If the sampling period is small, these columns will not differ very much, thus producing a nearly singular matrix. The second difficulty arises in the calculus of the roots of the denominator, i.e. the poles of the system. All the time constants long compared to the sampling period correspond to z-plane poles close to one. The reader can convince himself that in this case the jacobian of the mapping between the A(z) polynomial coefficients and the z-plane poles

$$J_{ij} = \frac{\partial z_j}{\partial a_i} \tag{31}$$

has a large determinant. This means that a small error on the estimate of the denominator will be amplified when computing its roots.

We propose a parade to these problems by artificially reducing the sampling rate but keeping between sample information. In this way we avoid data decimation which would statistically degrade the identification quality. To access the signals between two sampling times, we make use of the modified z-transform, defined by

$$y(z,m) = \sum_{k=-\infty}^{\infty} y(k+m)\, z^{-k} \; ; \; 0 \le m \le 1 \; . \tag{32}$$

Let us first expand the z-transfer function in partial fractions :

$$H(z) = \frac{B(z)}{A(z)} = \sum_{n=1}^{N} \frac{c_n}{1 - z_n z^{-1}}. \qquad (33)$$

The impulse response invariance allows to give an expression for the modified z-transfer function :

$$H(z,m) = \frac{B(z)}{A(z)} = \sum_{n=1}^{N} \frac{c_n z_n^m}{1 - z_n z^{-1}}. \qquad (34)$$

This can be understood by noting that the impulse response at time (k+m)T is

$$h(k,m) = \sum_{n=1}^{\infty} c_n e^{p_n kT} e^{p_n mT}. \qquad (35)$$

To be used in a recurrent relation for signal estimation in the temporal space, expression (34) must be written as a rational function :

$$H(z,m) = \frac{\sum_{n=1}^{N} c_n z_n^m Q_n(z)}{A(z)} = \frac{B_m(z)}{A(z)}, \qquad (36)$$

where

$$Q_n(z) = \prod_{\substack{i=1 \\ i \neq n}}^{N} (1 - z_i z^{-1}) = \sum_{i=0}^{N-1} q_{ni} z^{-i}. \qquad (37)$$

In order to include this new expression in an identification process, we must find a relation between the coefficients of B(z,m) and the parameters of the system, i.e. the coefficients of A(z) and B(z). To this purpose, we write the B(z,m) coefficients in a vector notation, B(m) being the row vector formed with the B(z,m) coefficients :

$$B(m) = (c_1\ c_2 \cdots c_N) \begin{pmatrix} z_1^m & 0 & \cdots & 0 \\ 0 & z_2^m & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & z_N^m \end{pmatrix} \begin{pmatrix} q_{10} & q_{11} & \cdots & q_{1N-1} \\ q_{20} & q_{21} & \cdots & q_{2N-1} \\ \vdots & \vdots & \ddots & \vdots \\ q_{N0} & q_{N1} & \cdots & q_{NN-1} \end{pmatrix}. \qquad (38)$$

$\mathcal{Q}$ will symbolise the last matrix whose rows are the coefficients of the Q_n polynomials while Z(m) will represent the diagonal matrix formed with the m-th power of the z-poles. Using the particular case for m = 0,

$$B(m = 0) = (c_1\ c_2 \cdots c_N)\,\mathcal{Q}\,, \qquad (39)$$

we can formulate the desired relation :

$$B(m) = B\,\mathcal{Q}^{-1} Z(m)\,\mathcal{Q} = B\,P(m)\,. \qquad (40)$$

The recurrent relation for the output signal estimator between sampling times can now be explicitly given :

$$y(k,m) = \sum_{n=0}^{N-1} b_n \sum_{i=0}^{N-1} P_{ni}(m)\,x(k-i) - \sum_{n=1}^{N} a_n\,y(k-n,m)\,. \qquad (41)$$

We note that, given the P(m) matrix, this expression is a linear combination of the system parameters. When compared with the non decimated case, we see a change in the independent variable set : raw input signal is now filtered through a finite impulse response filter whose coefficients are given by the rows of the P(m) matrix.

It turns out that the iterative identification algorithm presented in the previous section is particularly well suited to handle this formulation. In fact the P(m) matrix, which depends on the coefficients of the A(z) polynomial, can be approximated at each iteration using the previous estimate of these coefficients. In this way we can identify the system parameters with an artificially enhanced sampling period. The advantage is that intermediate samples can be kept in the regression process, an information that would have been thrown away by a simple decimation. This longer sampling period will improve the problem conditioning, thus lessen the constraints

on the quality of the matrix inversion. In the context of adaptative control, this will also make the real time implantation of the algorithm easier.

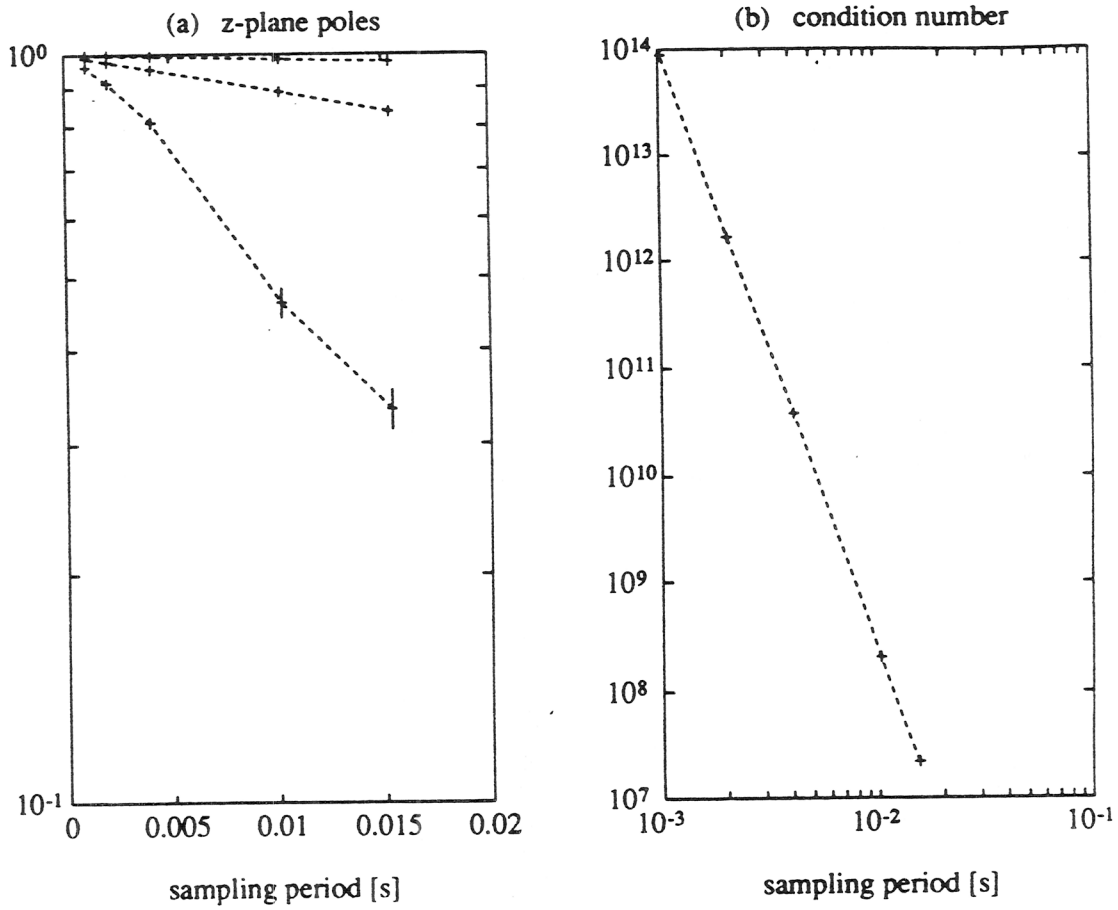

Fig. 5 : (a) Position of the z-plane poles of a third order transfer function identified with an increasing apparent sampling period.
(b) Condition number improvement when the apparent sampling period is increased.

The modification of the algorithm has been tested on our laboratory data. Figure 5 (a) shows how the z-plane poles move away form the critical z = 1 point when the apparent sampling period is increased. In this representation, constant s-plane poles sit on straight lines crossing z = 1 when the sampling period tends toward zero. A small discrepancy appears for the third pole at the highest sampling period. In this situation, the corresponding time constant

only slightly exceeds the sampling period, and that explains the observed difference. Figure 5 (b) demonstrates the drastic improvement in the conditioning, inferred here from the ratio of the largest to the smallest singular value of the covariance matrix.

5. CONCLUSION

Identification of the dynamical response of a system with distributed parameters requires an ad hoc treatment. First the poles of such a system do not depend on the continuous parameters, a feature that has been included in the proposed identification procedure. In addition those systems are characterised by sparse time constants. The determination of the system dynamic is therefore an ill posed problem. This makes difficult the use of non linear minimisation algorithms. These techniques were favourably replaced by an iterative algorithm whose steps consist in a simple linear minimisation. The bad conditioning of the problem was further improved by a data decimation technique allowing to increase the apparent sampling period without losing intermediate samples.

In this way a system identification method has been elaborated and tested, that applies particularly well to distributed parameter systems, either for control purposes, or for fundamental study of their dynamic. Work involving this method as a tool for investigating transport in the Tore Supra Tokamak is in progress.

APPENDIX I

In this appendix we will briefly expose, for those who are not familiar with, the main features of a z-transfer function.

The z-transform is a particular formulation of the Laplace transform for sampled signals. Such a signal, recorded with a sampling period T, can be represented by a sum of delta functions :

$$u(t) = \sum_{k=-\infty}^{\infty} u(k)\,\delta(t - kT)\,. \tag{42}$$

The Laplace transform of this signal then reduces to

$$u(s) = \int_{-\infty}^{\infty} u(t)\,e^{-st}\,dt = \sum_{k=-\infty}^{\infty} u(k)\,e^{-skT}\,. \tag{43}$$

It is of course tempting to rewrite this particular expression as

$$u(z) = \sum_{k=-\infty}^{\infty} u(k)\,z^{-k}\,, \tag{44}$$

defining in this way the z-transform u(z) of the sampled signal u(k).

The most useful property of the z-transform is its ability to delay a signal. It is easy to show that the z-transform of the one sampling period delayed signal u(k-1) is given by $z^{-1}\,u(z)$, a statement that we will abusively summarise as

$$z^{-1}\,u(k) = u(k - 1)\,. \tag{45}$$

This property allows to apply a z-transfer function expressed as a rational function in the z^{-1} variable

$$H(z) = \frac{\sum_{m=0}^{M} b_m z^{-m}}{\sum_{n=0}^{N} a_n z^{-n}} \qquad (46)$$

to any input time sequence x(k) easily. The output of such a system is estimated with the recurrent relation

$$y(k) = - \sum_{n=1}^{N} a_n y(k-n) + \sum_{m=0}^{M} b_m x(k-m) , \qquad (47)$$

where the numerator and denominator have been normalised so that $a_0 = 1$.

It is shown in section 2 that continuous time systems are also characterised by transfer functions expressed in the Laplace variable as rational functions. It is thus necessary to discretise these systems in a way such that the mapping between the s-plane and the z-plane do not alter this form. Among the numerous discretisation approximations, we will retain two methods. The first one is the bilinear transformation, which is an approximation valid for small values of s :

$$s = \frac{\ln z}{T} \cong \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} . \qquad (48)$$

The advantage of this transformation is that it can be made exact for all imaginary values of $s = j\omega$ with the substitution

$$z^{-1} = \frac{2 - j\omega}{2 + j\omega} = e^{-2j \arctan \frac{T\omega}{2}} . \qquad (49)$$

It is therefore well adapted when one wants to keep a link with the Fourier formalism. The second methods, known as the impulse response invariance method, states that both the continuous and discrete time impulse response of the system match. This imposes a mapping between the residues and the poles of the s-plane transfer function, respectively r_n and p_n, and those of H(z), respectively c_n

and z_n :

$$z_n = e^{p_n T} , \tag{50}$$

$$r_n = c_n^* . \tag{51}$$

Here the asterix represents the complex conjugate. This approximation is widely used since it is exact for all two valued stimuli, including delta function, step, square wave and binary random sequence.

APPENDIX II

The interested reader will find in this appendix three implantations of the algorithms described in this report. Two of them use MATLAB™ and are as far as possible compatible with its "Identification Toolbox". The third is written in C and includes calls to the LINPACK library. It offers less flexibility but will find its place on a super-computer for large scale problems.

`zti.m` Identification of a multi input system with distributed parameters. It requires a modification of `idsim.m` to handle more than one output.

`ztim.m` Identification of a multi input single output system with data decimation.

`zti.c` Identification of a multi input system with distributed parameters.

```
function [th,thm] = zti(z,nn,ny,maxiter,tol,Tsamp)

%ZTI Computes iteratively the parameter estimate of an output-error
%    model for a MIMO system.
%
%    TH = ZTI(Z,NN)
%
%    TH: Returned as the estimated parameters of the output-error
%    model along with estimated covariances and structure information.
%    For the exact format of TH, see HELP THETA. B polynomial order is
%    [B(y1,u1) B(y1,u2) ... B(y2,u1) B(y2,u2) ...]. In addition, entry
%    TH(2,8) contains the condition number of the covariance matrix
%    and entry TH(2,9) the number of outputs NY.
%
%    Z: The output-input data Z=[y1 y2 ... u1 u2 ...], with yn and un
%    being column vectors.
%
%    NN: Initial value and structure information, given either as
%    NN = [na nb nk], the orders and delays of the above model, or as
%    NN = THI, with THI being a theta matrix of the standrad format.
%    In the latter case the iteration is initialized at THI. In the
%    former case if NN = -[na nb nk], inputs will be prefiltered for
%    -1 zeros (this feature is not available from THI).
%
%    When called as [TH,THM] = ZTI(Z,NN), ZTI will return in row k of
%    THM the k-th iteration parameter estimate in 'alphabetical
%    order'.
%
%    Some parameters associated with the algorithm are accessed by
%    TH = ZTI(Z,NN,NY,MAXITER,TOL,T)
%    NY: The number of outputs, one by default. See HELP AUXVAR for an
%    explanation of the others, and their default values.

% default arguments
if nargin < 3, ny = 1; end
if nargin < 4, maxiter = -1; end
if nargin < 5, tol = -1; end
if nargin < 6, Tsamp = -1; end
if maxiter < 0, maxiter = 10; end
if tol < 0, tol = .01; end
if Tsamp < 0, Tsamp = 1; end

% z consistency checks
[Ncap,nz] = size(z); nu = nz - ny; nuy = nu*ny;
if nz > Ncap, error('Data organised in column vectors'), end
if nu <= 0, error('ZTI makes sense only if an input is present'), end

% model orders or initial theta
[nk,nc] = size(nn);
if nk == 1
 if nc ~= 2*nuy + 1, error('Orders should be  nn = [na nb nk]'), end
 prefilt = any(nn < 0); nn = abs(nn); A = [1 zeros(1,nn(1))];
else
 nuy = nn(1,3); ny =  nn(2,9); nu = nuy / ny;
 if nz ~= nu + ny, error('THI not matching number of IOs'), end
 A = real(poly(roots([1 nn(3,1:nn(1,4))]).^(Tsamp/nn(1,2))));
 prefilt = 0; nn = nn(1,[4:nuy+4,2*nuy+7:3*nuy+6]);
end
```

```
% sort out orders
na = nn(1); nb = reshape(nn(2:nuy+1),nu,ny);
nk = reshape(nn(nuy+2:2*nuy+1),nu,ny);
ni = max([na;nb(:)+nk(:)-1]); n = na + sum(nb(:)); Neff = Ncap - ni;

% numerator for prefiltering : (1+q)^n
if prefilt & (any(nb(:)>na+1) | any(diff(nb(:))))
 error('-1 zeros prefiltering requires constant nb <= na+1')
end
P = ones(nu+ny,1);
if prefilt
 P(ny+1:ny+nu,1:na-nb(1)+2) = ...
   ones(nu,1)*diag(flipud(pascal(na-nb(1)+2)))';
end

% constant indices and allocation
jj = ni+1:Ncap; ncum = [0;cumsum(nb(:))] + na;
phy = zeros(Neff*ny,1); phi = zeros(Neff*ny,n);
zf = zeros(Ncap,nz); t = zeros(n,1) + Inf; thm = [];

% loop initiation and loop
iter = 0; G = Inf; condition = 0; ktrace = 1:max(na,(n<21)*n);

while iter < maxiter & G > tol
 iter = iter + 1;

 % prefiltering
 for k = 1:nz, zf(:,k) = filter(P(k,:),A,z(:,k)); end

 % build up regression matrix
 for ky = 1:ny
  kk = (ky-1)*Neff+1:ky*Neff;
  phy(kk) = zf(jj,ky);
  for k = 1:na
   phi(kk,k) = -zf(jj-k,ky);
  end
  for ku = 1:nu
   l = ky + (ku-1)*ny;
   lphi = ncum(l);
   for k = 1:nb(l)
    lphi = lphi + 1;
    phi(kk,lphi) = zf(jj-(k+nk(l)-1),ny+ku);
   end
  end
 end
 tnew = phi \ phy;
 if any(tnew == 0)
  condition = Inf;
  disp('Identification leads to a rank deficient regression matrix')
  disp('Iterations therefore terminated')
  break
 end

 thm = [thm; tnew']; g = tnew - t; G = norm(g);

 % trace
 clc
 disp([' ITERATION # ' int2str(iter) ' '])
 disp( 'Current th  prev. th   difference')
 theta = [tnew(ktrace) t(ktrace) g(ktrace)]
 disp(['Norm of difference: ' num2str(G)])
```

```
 % update model
 t = tnew; A = [1 t(1:na).'];

end

% coefficient covariance matrix
if finite(condition)
 R = phi.' * phi; e = phy - phi*t; R = e.'*e / Neff / ny * inv(R);
 condition = cond(R);
else
 R = zeros(n) + Inf;
end

% prefiltering convolution matrix
C = eye(n);
if prefilt
 c = flipud(hankel([zeros(1,nb(1)-1)  P(nz,:)],...
              [1 zeros(1,nb(1)-1)]));
 for ku = 1:nuy
  kphi = size(C); k = kphi + size(c);
  C(kphi(1)+1:k(1),kphi(2)+1:k(2)) = c;
 end
end

% expanded numerators for prefiltering
tnew = C * t; R = C * R * C.';
if prefilt, nb = ones(1,nuy) + na; end

% build up theta matrix
th = [Inf Tsamp nuy na nb(:)' 0 0 zeros(1,nuy) nk(:)'];
th(2,1) = Inf; th(2,2:7) = clock; th(2,2) = th(2,2) / 100;
th(2,7:9) = [13 condition ny]; % 13 is the ZTI id
k = length(tnew); th(3,1:k) = tnew.'; th(4:k+3,1:k) = R;

% introduce correct loss functions
if finite(condition)
 zf = idsim(z(:,ny+1:nz),th); e = z(:,1:ny); e = zf(:) - e(:);
 Ncap = Ncap * ny;
 th(1:2,1) = e.'*e * [1;(Ncap+n)/(Ncap-n)] / (Ncap-n);
end
```

ztm.m

```
function [th,thm] = ztim(z,nn,ns,maxiter,tol,Tsamp)

%ZTIM Computes iteratively the parameter estimate of an output-error
%    model for a SISO or MISO system with data decimation.
%
%    TH = ZTIM(Z,NN,NS)
%
%    TH: Returned as the estimated parameters of the output-error
%    model along with estimated covariances and structure information.
%    For the exact format of TH, see HELP THETA. In addition, entry
%    TH(2,8) contains the condition number of the identification.
%
%    Z : The output-input data Z=[y u], with y and u being column
%    vectors. For MISO systems, u=[u1 u2 ... un]. For input such as
%    delta- or step- functions, it is recommended that the transition
%    sample index is a multiple of NS plus ones.
%
%    NN: Initial value and structure information, given either as
%    NN = [na nb nk], the orders and delays of the above model, or as
%    NN = THI, with THI being a theta matrix of the standrad format.
%    In the latter case the iteration is initialized at THI.
%
%    NS: The data decimation number.
%
%    When called as [TH,THM] = ZTIM(Z,NN,NS), ZTIM will return in row
%    k of THM the estimated parameter in 'alphabetical order' at the
%    k-th iteration.
%
%    Some parameters associated with the algorithm are accessed by
%    TH = ZTIM(Z,NN,NS,MAXITER,TOL,T)
%    See HELP AUXVAR for an explanation of these, and their default
%    values.

% default arguments
if nargin < 4, maxiter = -1; end
if nargin < 5, tol = -1; end
if nargin < 6, Tsamp = -1; end
if maxiter < 0, maxiter = 10; end
if tol < 0, tol = .01; end
if Tsamp < 0, Tsamp = 1; end

% z consistency checks
[Ncap,nu] = size(z); nu = nu -1;
if nu > Ncap, error('Data organized in column vectors'), end
if nu == 0, error('ZTIM makes sense only with an input'), end

% model orders or initial theta
[nk,nc] = size(nn);
if nk == 1
 if nc ~= 2*nu+1, error('Orders should be nn = [na nb nk]'), end
 A = [];
else
 if nu ~= nn(1,3), error('THI not matching number of inputs'), end
 A = real(poly(roots([1 nn(3,1:nn(1,4))]).^(Tsamp/nn(1,2))));
 nn = nn(1,[4:nu+4,2*nu+7:3*nu+6]); t = [];
end

% sort out orders
na = nn(1); nb = nn(2:nu+1); nk = nn(nu+2:2*nu+1);
```

```
ni = max([na nb+nk-1]); n = na + sum(nb);
if any(nb ~= na), error('ZTIM requires na = nb'), end
if any(nk ~= 0) , error('ZTIM requires nk = 0'), end

% some usefull variables
badcond = 1/(Ncap*eps);
js = 1:ns:Ncap+1-ns; ji = ni*ns+1:ns:Ncap+1-ns;
Neff = length(ji); Ncap = Neff*ns;
Js = 1:Neff; Ji = Js + ni;

% initial estimate
if  isempty(A)
 phi = zeros(Neff,n);
 for k=1:na
  phi(:,k) = -z(ji-k*ns,1);
  phi(:,na+k:na:na*nu+k) = z(ji-(k-1)*ns,2:nu+1);
 end
 condition = cond(phi);
 if condition > badcond, error('Rank deficient matrix'), end
 t = phi \ z(ji,1);
 A = [1 t(1:na).'];
 clc
 disp(['  INITIAL ESTIMATE'])
 disp(['Conditioning: ' num2str(condition)])
 disp([' th-vector'])
 theta = t
end

% iterations
iter = 0; G = Inf; thm = t.';
phi = zeros(Ncap,n); phy = phi(:,1);

while iter < maxiter & G > tol
 iter = iter + 1;

 % computes polynomials for between-sample estimation
 p = roots(A);
 for k = 1:na, Q(k,:) = poly(p([1:k-1 k+1:na])); end
 Qinv = inv(Q);

 % build up regression matrix
 for km = 0:ns-1
  jj = Js + Neff*km;
  P = real(Qinv * diag(p.^(km/ns)) * Q);
  v = filter(1,A,z(js+km,1));
  for k = 1:na
   phi(jj,k) = -v(Ji-k);
   for ku = 1:nu
    u = filter(P(k,:),A,z(js,ku+1));
    phi(jj,k+ku*na) = u(Ji);
   end
  end
  phy(jj) = v(Ji);
 end
 condition = cond(phi);
 if condition > badcond
  condition = Inf;
  disp('Identification leads to a rank deficient regression matrix')
  disp('Iterations therefore terminated')
  break
 end
 tnew = phi \ phy; thm = [thm; tnew'];
```

```
 % trace
 clc
 disp(['       ITERATION # ' int2str(iter)])
 disp(['Conditioning: ' num2str(condition)])
 disp( 'Current th  prev. th   difference')
 g = tnew - t; G = norm(g);
 theta = [tnew t g]
 disp(['Norm of difference: ' num2str(G)])

 % update model
 t = tnew; A = [1 t(1:na).'];
end

% covariance matrix
if finite(condition)
 e = phy - phi*t; R = e.'*e / (Ncap-n) * inv(phi.'*phi);
else
 R = zeros(n) + Inf;
end

% build up theta matrix
th = [Inf Tsamp*ns nu na nb 0 0 zeros(1,nu) nk];
th(2,1) = Inf; th(2,2:7) = clock; th(2,2) = th(2,2) / 100;
th(2,7) = 13; th(2,8) = condition; th(3,1:n) = t.'; th(4:n+3,1:n) = R;

% introduce correct loss functions
if finite(condition)
 e = idsim(z(js,2:nu+1),th) - z(js,1);
 th(1:2,1) = e.'*e * ns / (Ncap-n) * [1;(Ncap+n)/(Ncap-n)];
end
```

```
# include <stdio.h>
# include <malloc.h>
# include <string.h>
# include <math.h>

/* external LINPACK routines */
void SQRDC (); void SQRSL (); void SPPDI ();

void filter (y, x, n, A, na)
  double *y;                    /* output filtered signal */
  double *x;                    /* input signal */
  int     n;                    /* number of samples */
  double *A;                    /* A coeff. without the leading 1 */
  int     na;                   /* A degree */

{ register int ka, k;           /* counters */

  for (k = 0; k < na; k++) y[k] = 0; /* initial values */
  for (k = na; k < n; k++) {
    y[k] = x[k];                /* B is assumed to be 1 */
    for (ka = 0; ka < na; ka++) y[k] -= A[ka]*y[k - ka - 1];
  }
}

int zti (u, y, nu, ny, ncap, A, na, nb, maxiter, tol, thm, R)
/* returns the number of iterations or -1 when out of memory or -2
   when identification lead to a rank deficient regression matrix */
  double   *u;                  /* nu input signals of ncap samples */
  double   *y;                  /* ny output signals of ncap samples */
  int       nu;                 /* number of input signals */
  int       ny;                 /* number of output signals */
  int       ncap;               /* number of samples */
  double   *A;                  /* init. A coeff. without the leading
                                   1 */
  int       na;                 /* denominator degree */
  int       nb;                 /* numerator degree plus 1 */
  int       maxiter;            /* maximum number of iterations */
  double    tol;                /* convergence tolerance */
  double  **thm;                /* model parameters , storage format is
                                   {a1 a2 ... b0(u1,y1) b1(u1,y1) ...
                                   b0(u2,y1) b1(u2,y1) ... b0(u1,y2)
                                   b1(u1,y2) ... }, repeated for each
                                   iteration. The final result is
                                   returned at the end of thm */
  double  **R;                  /* upper triangle of the covariance
                                   matrix, stored column wise */

{ int               ni,         /* number of initial conditions */
                    neff,       /* number of usefull samples */
                    n,          /* number of free parameters */
                    neq;        /* number of equations */

  double           *uf,         /* filtered inputs */
                   *yf,         /* filtered outputs */
                   *phy,        /* dependant variable vector */
                   *phi,        /* independant variable matrix */
                   *res,        /* residue after linear regression */
                   *aux;        /* auxilliary space for SQRDC */
  double            g, G;      /* norm of diff. */
```

```
int             job,info /* LINPACK job selector and result */

register int    k, i, j, /* counters */
                ku, ky,  /* input output counter */
                ka, kb,  /* A, B coefficient counter */
                iter=0;  /* iteration counter */

/* usefull quantities */
neff = ncap - (ni = na >= nb ? na : nb - 1);
n = na + nu*ny*nb;
neq = neff*ny;

/* memory allocation */
uf   = (double *) malloc (ncap*nu*sizeof (double));
yf   = (double *) malloc (ncap*ny*sizeof (double));
phy  = (double *) malloc (neq*sizeof (double));
phi  = (double *) malloc (neq*n*sizeof (double));
res  = (double *) malloc (neq*sizeof (double));
aux  = (double *) malloc (neq*sizeof (double));
*thm = (double *) malloc (maxiter*n*sizeof (double));
*R   = (double *) malloc (n*n*sizeof (double));
if (!uf || !yf || !phy || !phi || !res || !aux || !*thm || !*R)
  return - 1;
memset (*thm, 0, maxiter*n*sizeof (double));

/* iterate */
G = 2*tol;
while (iter < maxiter && G > tol) {

  /* prefiltering */
  for (ku = 0; ku < nu; ku++)
    filter (uf + ku*ncap, u + ku*ncap, ncap, A, na);
  for (ky = 0; ky < ny; ky++)
    filter (yf + ky*ncap, y + ky*ncap, ncap, A, na);

  /* build up regression matrix */
  memset (phi, 0, neq*n*sizeof (double));
  /* has been altered by SQRDC */
  for (ky = 0; ky < ny; ky++) {
    memcpy (phy + ky*neff,
       yf + ky*ncap + ni, neff*sizeof (double));
    for (ka = 0; ka < na; ka++)
      memcpy (phi + ka*neq + ky*neff,
         yf + ky*ncap + ni - ka - 1, neff*sizeof (double));
    for (ku = 0; ku < nu; ku++)
      for (kb = 0; kb < nb; kb++)
        memcpy (phi + (na + kb + (ky*nu + ku)*nb)*neq + ky*neff,
           uf + ku*ncap + ni - kb, neff*sizeof (double));
  }
  for (k = 0; k < neq*na; k++) phi[k] *= -1;

  /* solve over constraint system */
  job = 0; SQRDC (phi, &neq, &neq, &n, aux, NULL, NULL, &job);
  /* save upper triangle of phi for covariance matrix */
  for (j = 0, k = 0; j < n; j++)
    for (i = 0; i <= j; i++) *(*R + k++) = phi[j*neq + i];
  job = 110;
  SQRSL (phi, &neq, &neq, &n, aux, phy, NULL,
         res, *thm + n*iter, res, NULL, &job, &info);
  /* rank deficiance test */
  if (info != 0) {
    free (uf); free (yf); free (phy);
```

```
      free (uf); free (yf); free (phy);
      free (phi); free (res); free (aux);
      fprintf (stderr, "ZTI - rank deficient matrix.\n");
      return - 2;
    }

    /* computes difference with the previous iteration */
    if (iter > 0) {
      for (k = 0, G = 0; k < n; k++) {
        g = *(*thm + n*(iter - 1) + k) - *(*thm + n*iter + k);
        G += g*g;
      } G = sqrt (G / n);
    }

    /* update iteration */
    memcpy (A, *thm + n*iter, na*sizeof (double));
    iter++;
  }

  realloc (*thm, n*iter*sizeof (double));

  /* covariance matrix */
  job = 1; SPPDI (*R, &n, NULL, &job);
  for (k = 0, G = 0; k < neq; k++) G += res[k]*res[k]; G /= neq;
  for (k = 0; k < n*(n + 1) / 2; k++) *(*R + k) *= G;

  /* clean up memory */
  free (uf); free (yf); free (phy);
  free (phi); free (res); free (aux);
  return iter;
}
```

REFERENCES

[1] TH. DUDOK DE WIT, B. JOYE, J.B. LISTER, J.-M. MORET, in *Proceedings of the 17th Europ. Conf. on Cont. Fusion and Plasma Physics, Amsterdam, 1990*, published by European Physical Society, p. 187.

[2] L. LJUNG, *System identification*, (Prentice-Hall, London, 1987).

[3] EQUIPE TORE-SUPRA, in *IAEA Conf. on Plasma Phys. and Nuclear Fusion, Nice, 1988*, published by the International Atomic Energy Agency, vol. I, p. 9.

[4] G. FAVIER, «*Filtrage, modélisation et identification de systèmes linéaires stochastiques à temps discret*», (Ed. du CNRS, Paris, 1982).