

# Implementation of the Ho-Kalman Algorithm via the Leverrier Resolvent

by

Robert W. Bass  
donquixote@innoventek.com  
Innoventek, Inc.

1. Statement of the Empirical System Identification Problem (**ESIP**)
2. The Leverrier/Souriau-Frame/Fadeeva **Resolvent** Algorithm
3. The General **MIMO** Case Using a Combined Leverrier/Ho-Kalman Algorithm

## Appendix

Internally-documented MATLAB Source-Code Listings

1. matCompanion
2. matSpec
3. generateChirpInput
4. generateNoisyExamples
5. getHankelMatrix
6. HoKalmanHankel
7. get\_ALPHmat
8. getIDb
9. LTIsystemID
10. LTIsystemIDa
11. LTIsystemIDm

## 1. Statement of the Empirical System Identification Problem (ESIP)

Consider an autonomous but otherwise unidentified *Linear Time Invariant (LTI)* system  $\mathcal{S}$  operating near quiescent equilibrium in discrete (digital) time. The sampling interval duration  $\tau > 0$  is normalized to be unity ( $\tau = 1$ ) and the sampling *times*  $\{t_k = k \cdot \tau \mid k = (1, 2, \dots, N)\}$  are therefore positive integers  $\{k\}$ .

In the *Single Input Single Output (SISO)* case, the recorded scalar **input** signals  $\{u(k) \mid k = (1, 2, \dots, N)\}$  and recorded scalar **output** signals  $\{y(k) \mid k = (1, 2, \dots, N)\}$  are regarded as real column  $N$ -vectors, or elements of real Euclidean  $N$ -space:  $u \in \mathbb{E}^N$ ,  $y \in \mathbb{E}^N$ .

In the *Multiple Input Multiple Output (MIMO)* case, there are (lower-case  $L$ )  $l \geq 1$  outputs  $y^{[i]} \in \mathbb{E}^N$ ,  $\{i = (1, 2, \dots, l)\}$  regarded as columns of an  $N \times l$  matrix  $y = [y^{[1]}, \dots, y^{[l]}] : \mathbb{E}^l \rightarrow \mathbb{E}^N$ , whose columns are the individually-measured **outputs**, and likewise  $m \geq 1$  **inputs**  $u^{[i]} \in \mathbb{E}^N$ ,  $\{i = (1, 2, \dots, m)\}$  regarded as columns of an  $N \times m$  matrix  $u = [u^{[1]}, \dots, u^{[m]}] : \mathbb{E}^m \rightarrow \mathbb{E}^N$ , whose columns define a suite of input commands.

But at time  $t = k$  the input  $u^k = u(k) = [u^{[1]}(k), \dots, u^{[m]}(k)]^T \in \mathbb{E}^m$  is regarded as a column  $m$ -vector, where  $^T$  denotes vector-matrix row-column **transposition** [given by ‘ in MATLAB], and likewise the output  $y^k = y(k) = [y^{[1]}(k), \dots, y^{[l]}(k)]^T \in \mathbb{E}^l$  is regarded as a column  $l$ -vector.

The celebrated **Ho-Kalman Lemma** asserts that under broadly plausible assumptions ([1]-[5]) widely accepted in systems engineering, and fruitful historically, there must exist an integer **state-dimension**  $n > 0$  and real matrices  $(A, B, C)$  of sizes respectively  $n \times n$ ,  $m \times n$ , and  $l \times n$ , such that for some *unknown* sequence of real column  $n$ -vectors  $\{x^k \mid k = (1, 2, \dots, N)\}$ ,  $x^k \in \mathbb{E}^n$ , called the system’s **state-vectors**, the actual LTI dynamics of the system  $\mathcal{S}$  are given by

$$\begin{aligned} x^{k+1} &= A \cdot x^k + B \cdot u^k, \\ y^k &= C \cdot x^k, \end{aligned} \quad (\mathcal{S})$$

for  $k = (1, 2, \dots, N)$ . Here  $A$  is called the **state-transition** matrix [computable by exponentiation of the continuous-time **dynamical coefficient** matrix],  $B$  is called the **input-coupling** matrix (or **actuator kinematics** matrix), and  $C$  is called the **output-coupling** matrix (or **sensor kinematics** matrix).

In the stochastic case we augment the preceding by

$$\begin{aligned} x^{k+1} &= A \cdot x^k + B \cdot u^k + v^k, \\ y^k &= C \cdot x^k + w^k, \end{aligned} \quad (\mathcal{S}_{\text{stch}})$$

where the sequences  $\{v^k\}$  &  $\{w^k\}$  are *zero-mean* Gaussian stochastic processes defined by their **covariance** and mutual **cross-variance** matrices  $(Q, R, S)$ , where  $Q = Q^T \geq 0$ , and  $R = R^T \geq 0$ , are symmetric non-negative definite matrices of sizes  $n \times n$ , and  $l \times l$ , and  $S$  is an  $l \times n$  matrix, such that, taking  $\mathcal{E}$  to denote the **expectation** operator

$$\mathcal{E}\{v^k\} = 0, \quad \mathcal{E}\{w^k\} = 0, \quad \mathcal{E}\{v^j(v^k)^T\} = Q \cdot \delta_{jk}, \quad \mathcal{E}\{w^j(w^k)^T\} = R \cdot \delta_{jk}, \quad \mathcal{E}\{w^j(v^k)^T\} = S \cdot \delta_{jk},$$

where  $\delta_{jk}$  denotes the *Kronecker delta* (i.e.  $\delta_{jk} = 1$  if  $j = k$  &  $\delta_{jk} = 0$  if  $j \neq k$ ). In this case  $(Q, R, S)$  may not be known but it is still desired to obtain an adequate **Identification (ID)** of  $(A, B, C)$  in the sense of acceptably small **prediction error**,  $\varepsilon_{\text{prd}}$  defined as follows.

Let **estimates**  $(A_{\text{est}}, B_{\text{est}}, C_{\text{est}})$  result from an ID procedure applied to the **data**  $\{u, y\}$ , and let the actual input sequence  $\{u\}$  be used to drive a computer-based numerical **simulation**

$$\begin{aligned} x^{k+1} &= A_{\text{est}} \cdot x^k + B_{\text{est}} \cdot u^k, \\ y_{\text{prd}}^k &= C_{\text{est}} \cdot x^k, \end{aligned} \quad (\mathcal{S}_{\text{sim}})$$

for  $k = (1, 2, \dots, N)$ . Then define *weighted* prediction error  $\varepsilon_{\text{prd}}$  as

$$\varepsilon_{\text{prd}} = \varepsilon_{\text{prd}}(\mathbf{n}) = \|y - y_{\text{prd}}\| \cdot \text{AIC}_{\text{fctr}},$$

where  $\|\cdot\|$  denotes Euclidean vector-matrix **norm**:  $\|x\| = (x^T \cdot x)^{1/2}$ ,  $\|M\| = \min\{\|M \cdot x\|/\|x\| \mid \forall x \neq 0\}$ , and where the positive factor

$$\text{AIC}_{\text{fctr}} = (N + n)/(N - n) > 1$$

is a crudely approximate realization of the *Akaike Information Criterion (AIC)* designed to penalize increasing  $n$  in order to avoid “modeling the noise instead of just the process.”

Returning to the ideal deterministic case, wherein it is supposed that all records are perfect and that one may perform “perfect” numerical analyses with infinite-precision arithmetic accuracy, the easiest way to appreciate the profound Ho-Kalman Lemma is to use Z-transform Analysis in order to eliminate the unknown states  $\{x^k\}$  as follows. Let  $z$  denote simultaneously a scalar **complex variable**  $z \in \mathbb{C}^1$  and also an **operator** such that  $\forall k$  (where  $k$  is any integer),  $z \cdot x^k = x^{k+1}$ , whence the preceding system becomes (courtesy of the assumption of initial quiescence, *i.e.* that  $x^k = 0$  for all  $k < 1$ , plus a few lines of elementary algebra)

$$y^k = \Phi(z) \cdot u^k,$$

(§z)

$$\Phi(z) = C \cdot (z \cdot I_n - A)^{-1} \cdot B \quad : \quad \mathbb{E}^m \rightarrow \mathbb{E}^l.$$

Using the well-known C. Neuman **resolvent** series for  $(z \cdot I_n - A)^{-1}$ , convergent  $\forall |z| > \|A\|$ , we have

$$\Phi(z) = \sum_{k=1}^{\infty} M_k \cdot z^{-k},$$

Where,  $\forall k \geq 1$ , the matrices  $\{M_k\}$ , defined by

$$M_k = C \cdot A^{k-1} \cdot B \quad : \quad \mathbb{E}^m \rightarrow \mathbb{E}^l,$$

are the all-important **Markov Parameters**. Standard engineering treatises on Linear Systems ([1],[2],[3]) show how to construct an infinite matrix  $H$ , called the Hankel matrix, wherein  $\forall i, j \geq 1$

$$H = (H_{i,j}) = (M_{i+j-1}),$$

such that if one computes the rank of the successively larger principal minors (or square block sub-matrices in the upper left-hand corner of  $H$ ) then this rank stops increasing and remains constant at the correct dimension  $n$ . The Ho-Kalman Lemma ([1], [2]) then shows how to “pull the Markov Parameters apart” in order to derive the system ID in the form  $(A, B, C)$  from the set  $(M_1, M_2, \dots, M_{2n})$ .

In the present approach to ESIP we note that for finite-precision arithmetic, as used on digital computers, *rank* is merely an approximation rather than an exact quantity.

Therefore it could be expedient, and is advocated by some, to simply try  $n = (1, 2, 3, \dots, n_{\text{mx}})$ , where for reasons that will be explained below  $n_{\text{mx}} < (N/[2 + l \cdot m])$ , and to select that value of  $n$  which minimizes the weighted output prediction error  $\varepsilon_{\text{prd}}(n)$ , although by application of the *Singular Value Decomposition (svd)* to  $H$  as in Section 4 below it could be that the original “rank test” might turn out to be even better if it were possible to determine a much larger  $H$  without any prior commitment to  $n$ . However, in the present strictly algebraic approach, it turns out to be mandatory to proceed via use of a “trial”  $n$ , after which testing the rank of  $H$  would be meaningless.

## 2. The Leverrier/Souriau-Frame/Fadeeva Resolvent Algorithm.

Rather than to use the preceding infinite-series approach to the system’s Z-domain **Transfer Function**  $\Phi(z)$ , it is more convenient to introduce some quantities which are **invariant** under non-singular coordinate changes of the type

$$x = T \cdot x_{\text{alt}}, \quad \det(T) \neq 0.$$

Note that now  $\mathcal{S}$  becomes

$$(\mathcal{S}_{\text{alt}}) \quad \begin{aligned} x_{\text{alt}}^{k+1} &= A_{\text{alt}} \cdot x_{\text{alt}}^k + B_{\text{alt}} \cdot u^k, \\ y^k &= C_{\text{alt}} \cdot x_{\text{alt}}^k, \end{aligned}$$

or  $\mathcal{S} = (A, B, C) \rightarrow \mathcal{S}_{\text{alt}} = (A_{\text{alt}}, B_{\text{alt}}, C_{\text{alt}})$ , wherein

$$A_{\text{alt}} = T^{-1} \cdot A \cdot T, \quad B_{\text{alt}} = T^{-1} \cdot B, \quad C_{\text{alt}} = C \cdot T.$$

But now, because, as is well known,  $(A_{\text{alt}})^j = T^{-1} \cdot A^j \cdot T$ , ( $j = 2, 3, \dots$ ),

$$M_k = C_{\text{alt}} \cdot A_{\text{alt}}^{k-1} \cdot B_{\text{alt}} = (C \cdot T) \cdot (T^{-1} \cdot A^{k-1} \cdot T) \cdot (T^{-1} \cdot B) = C \cdot A^{k-1} \cdot B,$$

or the Markov parameters are invariant under linear changes of coordinates, as are the poles of the resolvent of  $A$ , namely  $(z \cdot I_n - A)^{-1}$ , defined as the characteristic roots (or eigenvalues) of the characteristic polynomial

$$\begin{aligned} \Delta(z) &= \det(z \cdot I_n - A) = z^n + \alpha_{n-1} \cdot z^{n-1} + \dots + \alpha_k \cdot z^k + \dots + \alpha_1 \cdot z + \alpha_0 = \\ &= \sum_{j=0}^n \alpha_j \cdot z^j, \quad (\alpha_n = 1). \end{aligned}$$

It seems often to be inadequately appreciated that the system's  $l \cdot m \cdot (n - 1)$  zeroes (defined as the complex frequencies of maximal attenuation of the  $i^{\text{th}}$  input to the  $j^{\text{th}}$  output when the system is subjected to sinusoidal inputs) are equally important invariants, the identification of which, along with the poles, completely determines the system's dynamics, because of the following considerations.

Let  $C = [c^1, c^2, \dots, c^l]^T = [(c^1)^T; (c^2)^T; \dots; (c^l)^T]$  so that the  $c^i \in \mathbb{E}^n$  are the transposed rows of  $C$ , *i.e.* if the fundamental unit vectors  $\{e^i\}$ ,  $e^i \in \mathbb{E}^n$ , are defined by the columns of the identity  $I_n = [e^1, e^2, \dots, e^n]$ , then  $(c^i)^T = C \cdot e^i$ , ( $i = 1, 2, \dots, l$ ).

Similarly, let  $B = [b^1, b^2, \dots, b^m]$ ,  $b^i \in \mathbb{E}^n$ , ( $i = 1, 2, \dots, m$ ), define the columns  $b^i$  of  $B$ .

Under a non-singular linear change of coordinates  $x = T \cdot x_{\text{alt}}$ ,  $\det(T) \neq 0$ ,  $C \rightarrow C_{\text{alt}} = C \cdot T = [(c^1)^T; (c^2)^T; \dots; (c^l)^T] \cdot T = [(c^1)^T \cdot T; (c^2)^T \cdot T; \dots; (c^l)^T \cdot T]$ , or  $c^i \rightarrow T^T \cdot c^i$ , ( $i = 1, 2, \dots, l$ ), and likewise  $B \rightarrow B_{\text{alt}} = T^{-1} \cdot B = T^{-1} \cdot [b^1, b^2, \dots, b^m] = [T^{-1} \cdot b^1, T^{-1} \cdot b^2, \dots, T^{-1} \cdot b^m]$  so that  $b^i \rightarrow T^{-1} \cdot b^i$ , ( $i = 1, 2, \dots, m$ ).

Then the zeroes of the  $i^{\text{th}}$  input to the  $j^{\text{th}}$  output are the  $(n - 1)$  roots of the  $l \cdot m$  numerator polynomials  $\Delta_{ij}(z)$  of degree  $(n - 1)$  defined as follows.

First, recall the (oft rediscovered) Leverrier/Souriau/Frame/Fadeeva Algorithm ([5], [3, pp. 656-658], [6], [7],[8]) for an expression of the resolvent of  $A$  as a matrix polynomial of degree  $(n - 1)$  divided by its characteristic polynomial:

$$(z \cdot I_n - A)^{-1} = \{1 / \Delta(z)\} \cdot \sum_{k=1}^n z^{k-1} \cdot S_k, \quad (S_n = I_n),$$

where, using  $tr(M)$  to denote  $trace(M) \equiv M_{11} + M_{22} + \dots + M_{nn}$ , the Leverrier numerator matrices  $\{S_k\}$

$$S_k = \sum_{j=0}^{n-k} \alpha_{j+k} \cdot A^j, \quad (k = 1, 2, \dots, n),$$

and the coefficients  $\{\alpha_k\}$  of  $\Delta(z)$  can be computed, simultaneously and recursively from the following simple

and elegant, but notoriously “numerically fragile,” algorithm:

$$\alpha_n = 1, \quad S_n = I_n,$$

$$\alpha_{n-j} = -(1/j) \cdot \text{tr}(A \cdot S_{n-j+1}), \quad S_{n-j} = A \cdot S_{n-j+1} + \alpha_{n-j} \cdot I_n,$$

for  $(j = 1, 2, \dots, n)$ . As both a theoretical and computational check on the calculations, including the original definition of the  $\{S_k\}$ , note that the final and  $n^{\text{th}}$  step is equivalent to the Cayley-Hamilton Theorem (that every matrix satisfies its own characteristic polynomial), namely

$$S_0 = \Delta(A) = 0,$$

Whence, if  $\alpha_0 \equiv (-1)^n \cdot \det(A) \neq 0$ , then  $A^{-1}$  exists and, by the preceding equation when  $j = (n - 1)$ ,

$$A^{-1} = -(1/\alpha_0) \cdot S_1.$$

Next, inserting the Leverrier Resolvent into the  $Z$ -domain transfer matrix  $\mathcal{S}_Z$ , we have [since  $(M \cdot N)^{-1} \equiv N^{-1} \cdot M^{-1}$  for arbitrary non-singular matrices  $M$  &  $N$ , and since  $I_n \equiv T^{-1} \cdot T$ ]

$$\begin{aligned} (\mathcal{S}_{\text{Zalt}}) \quad \Phi_{\text{alt}}(z) &= C_{\text{alt}} \cdot (z \cdot I_n - A_{\text{alt}})^{-1} \cdot B_{\text{alt}} = \\ &= C \cdot T \cdot (z \cdot I_n - T^{-1} \cdot A \cdot T)^{-1} \cdot T^{-1} \cdot B = \\ &= C \cdot [T \cdot (z \cdot I_n - T^{-1} \cdot A \cdot T)^{-1} \cdot T^{-1}] \cdot B = \\ &= C \cdot [T \cdot (z \cdot T^{-1} \cdot T - T^{-1} \cdot A \cdot T)^{-1} \cdot T^{-1}] \cdot B = \\ &= C \cdot [T \cdot T^{-1} \cdot (z \cdot I_n - A \cdot T)^{-1} \cdot T \cdot T^{-1}] \cdot B = \\ &= C \cdot (z \cdot I_n - A \cdot T)^{-1} \cdot B \equiv \Phi(z), \end{aligned}$$

so that  $\Phi(z)$  is indeed invariant, which of course implies the invariance of

$$\Delta_{ij}(z) := (e^i)^T \cdot \left\{ \sum_{k=1}^n z^{k-1} \cdot C \cdot S_k \cdot B \right\} \cdot e^j,$$

i.e.

$$\Phi = (\Phi_{ij}), \quad \Phi_{ij} = \Delta_{ij}(z) / \Delta(z),$$

where, for  $(i = 1, 2, \dots, m)$  &  $(j = 1, 2, \dots, l)$ ,

$$\Delta_{ij}(z) = \sum_{k=1}^n \beta_k^{ij} \cdot z^{k-1},$$

we have the  $l \cdot m \cdot n$  coefficients

$$\begin{aligned} \beta_k^{ij} &= (e^i)^T \cdot C \cdot S_k \cdot B \cdot e^j = \\ &= (e^i)^T \cdot \left\{ \sum_{p=0}^{n-k} \alpha_{p+k} \cdot C \cdot A^p \cdot B \right\} \cdot e^j = \end{aligned}$$

$$\begin{aligned}
&= (e^i)^T \cdot \left\{ \sum_{p=0}^{n-k} \alpha_{p+k} \cdot M_{p+1} \right\} \cdot e^j = \\
&= \sum_{p=0}^{n-k} \alpha_{p+k} \cdot \left\{ (e^i)^T \cdot M_{p+1} \cdot e^j \right\}.
\end{aligned}$$

Accordingly, the invariance of the Markov parameters  $\{M_p\}$  establishes the invariance of the coefficients  $\{\beta_k^{ij} \mid (k = 1, 2, \dots, n)\}$  of the  $l \cdot m$  numerator polynomials of degree  $(n - 1)$ , and so of the system's zeroes.

Therefore the most algebraically parsimonious possible solution to the ESIP, which cannot be improved upon, would be to use the  $(l + m) \cdot N$  data values to ascertain the  $n$  coefficients  $\{\alpha_j\}$  of  $\Delta(z)$  and the  $l \cdot m \cdot n$  coefficients  $\{\beta_k^{ij}\}$  of  $\Delta_{ij}(z)$ .

Consequently we shall seek an EXACT algebraic solution to a problem of  $l \cdot (N - n)$  linear equations in a total of  $n + l \cdot m \cdot n \equiv (1 + l \cdot m) \cdot n$  unknowns.

It may later seem more convenient in the case  $l > 1$  to find  $n \cdot (l + m) \cdot l$  unknowns and then to add constraints that certain of these unknowns must be identical and that others must vanish identically, but it has turned out in practice that this concern is unwarranted and that the numerical accuracy of the solution is not affected by simply ignoring the unknowns that should vanish and simply averaging each of the  $n$  sets of  $l$  unknowns that should be identical.

This concern about possible redundancy disappears in the MISO and SISO cases, wherein  $l = 1$ , and there are  $(N - n)$  linear equations in precisely the  $n + m \cdot n \equiv (1 + m) \cdot n$  unknowns specifying the coefficients of the denominator polynomials  $\Delta(z)$  and of the  $m$  numerator polynomials  $\Delta_{i1}(z)$ ,  $(i = 1, 2, \dots, m)$ . Consequently it is tempting to replace the general MIMO problem by a sequence of MISO problems, and to somehow combine the results to obtain a  $C$ -matrix with the required  $l > 1$  rows. However, it turns out that this leads to severe deterioration in the resultant numerical accuracy, and it is better to consider the MIMO case as a batch-process problem rather than to attempt to solve it piecemeal by combining a sequence of MISO problems.

### 3. The General MIMO Case Using a Combined Leverrier/Ho-Kalman Algorithm

By inspection of the preceding, it is easy to go back-and-forth between the Leverrier Resolvent Numerator matrices  $\{S_k \mid (k = 1, 2, \dots, n)\}$  and the powers  $\{A^{k-l} \mid (k = 1, 2, \dots, n)\}$  of  $A$  by simply inverting an  $n \times n$  lower-triangular Toeplitz matrix  $ALF = (ALF_{ij})$ , where  $ALF_{ij}$  vanishes if  $j > i$ , and is unity when  $j = i$ , and is equal to  $\alpha_{n-i+j}$  for  $i > j$ . As explained on page 96 of Friedland [6],  $ALF$  has another lower-triangular inverse matrix  $GAM$ , whose elements can be computed recursively. Then it is possible to go back and forth between the Leverrier matrices and the powers of  $A$  when they are stacked into block-matrix columns as follows.

Let  $S_{\text{vec}}$  denote a column whose  $j^{\text{th}}$  row is  $S_{n-j+1}$ , for  $(j = 1, 2, \dots, n)$ , and, similarly, let  $A_{\text{vec}}$  denote a column whose  $k^{\text{th}}$  row is  $A^{k-l}$ , for  $(k = 1, 2, \dots, n)$ . Then

$$S_{\text{vec}} = \{ALF \otimes I_n\} \cdot A_{\text{vec}}, \quad A_{\text{vec}} = \{GAM \otimes I_n\} \cdot S_{\text{vec}}$$

where  $\otimes$  denotes Kronecker product (as explained *e.g.* in Bellman [9], *p.* 235).

Now returning to the  $Z$ -domain formulation, we have  $y^q = \Phi(z) \cdot u^q$ , or, using  $z$  as an operator,

$$\Delta(z) \cdot y^q = \sum_{j=0}^n \alpha_j \cdot y^{j+q} = \sum_{k=1}^n D_k \cdot u^{q+k-l},$$

where  $D_k := C \cdot S_k \cdot B$ , for  $(k = 1, 2, \dots, n)$ . Next, let  $M_{\text{vec}}$  denote the Markov parameters stacked in descending order, so that the  $j^{\text{th}}$  row is  $M_j$ , and let  $D_{\text{vec}}$  denote the  $D_k$  stacked in ascending order, so that the  $j^{\text{th}}$  row is given by  $D_{n-j+1}$ . Then just as before, we can find the  $\{M_j\}$  if we know the  $\{D_k\}$  because

$$M_{\text{vec}} = \{GAM \otimes I_n\} \cdot D_{\text{vec}}.$$

As mentioned, the Ho-Kalman Lemma shows how to “unpack”  $M_{\text{vec}}$  to get  $(A,B,C)$  provided we already have the vector  $\mathbf{a}$  whose  $j^{\text{th}}$  row is  $\alpha_{n,j}$ , for  $(j = 1, 2, \dots, n)$ .

Consequently the algebraic version of ESIP has been reduced to this: use the data  $y$  &  $u$  to find  $\mathbf{a}$  &  $D_{\text{vec}}$ . This motivates us to rewrite the preceding results as

$$y^{q+n} = - \sum_{j=0}^{n-1} \alpha_j y^{j+q} + \sum_{k=1}^n D_k u^{q+k-1},$$

for  $(q = 1, 2, \dots, N - n)$ . Upon transposing, this leads to the final problem formulation

$$Y_N = -Y \cdot \text{Alf} + U \cdot \text{Bet} = \text{Coeff} \cdot \text{Alf} \cdot \text{Bet},$$

$$\text{Coeff} := [-Y, U], \quad \text{Alf} \cdot \text{Bet} := [\text{Alf}; \text{Bet}],$$

where  $Y_N := [y^{1+n}, \dots, y^{k+n}, \dots, y^N]^T$  is an  $(N - n) \times l$  data matrix, and where  $Y$  is an  $(N - n) \times n \cdot l$  [block] data-matrix of row  $l$ -vectors,  $Y_{ij} = (y^{n+i-j})^T$ , and  $U$ , similarly, is an  $(N - n) \times n \cdot m$  [block] data-matrix of row  $m$ -vectors,  $U_{ij} = (u^{n+i-j})^T$ ,  $(i = 1, 2, \dots, N - n)$ ,  $(j = 1, 2, \dots, n)$ , and where

$$\text{Alf} := I_l \otimes \mathbf{a}, \quad \text{Bet} := [D_1, D_2, \dots, D_k, \dots, D_n]^T,$$

so that finding the unknown  $\text{Alf} \cdot \text{Bet}$  is equivalent to finding  $\mathbf{a}$  &  $D_{\text{vec}}$  which will enable completion of the system ID from the classic Ho-Kalman Lemma. If we define  $Z_N := \text{Coeff}^T \cdot Y_N$  then a necessary condition for the problem to have a solution is that

$$Z_N = \text{Prd} \cdot \text{Alf} \cdot \text{Bet}, \quad \text{Prd} := \text{Coeff}^T \cdot \text{Coeff},$$

should have a solution, and a necessary and sufficient condition that this algebraic system should have a *unique* solution is that the smallest singular value of  $\text{Prd}$  should be positive. In the rare cases where this condition might fail, it is because “the input is not sufficiently richly exciting,” in the sense that the input does not excite all observable modes of the system.

Here we are considering only *generic* systems, i.e. those which are stable and which satisfy Kalman’s criteria of complete *controllability* and complete *observability*. If the system has  $n$  poles, or  $n$  complex frequencies at which resonant amplification of sinusoidal signals occurs, it is considered advisable to use an input with a least  $n$  distinct *incommensurable* [non-harmonic] frequencies in its Fourier spectrum. Also the use of a *chirp* input (wherein the frequency of a sinusoidal input is itself increasing linearly with time, therefore sweeping over a continuum of rational & irrational frequencies) has been found particularly effective.

The appended suite of MATLAB Source Code programs has been developed to implement the preceding algorithm for effecting system ID via use of the Leverrier Resolvent rather than use of the C. Neuman resolvent, which necessarily requires ID of so many of the Markov parameters that the right-hand columns and the bottom rows of the very large Hankel matrix eventually become negligible in size, owing to the exponential decay of  $\|M_k\|$  as  $k$  increases without limit. That approach may lead to greater ultimate numerical accuracy, as in the CVA approach of Wallace Larimore, implemented in the AdaptX procedure made available by Adaptics Inc., but it also requires many other profound considerations which lie far beyond the presently stipulated scope.

## References

- [1] B. L. Ho, R.E. Kalman, “Effective construction of linear state variable models from input-output functions,” *Regelungstechnik*, Vol. 14, pp. 545-548, 1966.
- [2] Rudolf E. Kalman, Peter L. Farb, Michael A. Arbib, *Topics in Mathematical Systems Theory*, McGraw Hill, 1969, esp. pp. 288-308.
- [3] Thomas Kailath, *Linear Systems*, Prentice-Hall, 1980, esp. p. 442.
- [4] Wilson J. Rugh, *Linear System Theory*, Prentice-Hall, 2<sup>nd</sup> Ed., 1996, esp. pp. 194-201.
- [5] Lotfi A. Zadeh & Charles A. Desoer, *Linear System Theory: the State Space Approach*, McGraw-Hill, 1963; reprinted by Krieger, 1979, esp. pp. 303-306.
- [6] Bernard Friedland, *Control System Design: An Introduction to State-Space Methods*, McGraw-Hill, 1986, reprinted by Dover, 2005, esp. pp. 71-73.
- [7] Charles G. Cullen, *Matrices and Linear Transformations*, 2<sup>nd</sup> Ed., Addison-Wesley, 1966, reprinted by Dover, 1990, esp. pp. 278-279.
- [8] V. N. Faddeeva, *Computational Methods of Linear Algebra*, Dover, 1959.
- [9] Richard Bellman, *Introduction to Matrix Analysis*, 2<sup>nd</sup> Ed., McGraw-Hill, 1970, reprinted by SIAM, 1997.

```

function [E,compA,alfa] = matCompanion(A);
%*****
%
% Usage: [E,compA,alfa] = matCompanion(A);
%
% Output: E = nonsingular matrix such that A.E = E.compA where
% compA = Companion form of A, i.e.
% compA = Enm1 - en.alfa where
% In = eye(n) = (e1, e2, ..., en) & ek are unit vectors with
% ek(j) = Kronecker delta(j,k). Also use
% km1 =: k - 1, for k = 1, 2, ... , n and define
% Enm1 =: (0*en, e1, e2, ... , enm1). Also define
% alfa' =: [alpha[0], alpha[1], ... , alpha[n-1]] as a
% row vector formed from the coefficients of the
% characteristic polynomial d(s) of A, namely
% if d(s) = s^n + alpha[n-1].s^[n-1] + ... + alpha[0],
% then alph(k) = alpha[n - k + 1], (k = 1, 2, 3, ... , n+1),
% where, of course, alpha[n] = 1 always. Alternatively,
% alpha[j] = alph(n - j + 1), (j = 0, 1, 2, ... , n).
% Finally, alf = fliplr(alfa) is used below, where
% from the preceding, alf & alfa are n-vectors and
% alph and alpha are (n+1)-vectors.
%
% Copyright by Robert W. Bass, February 15, 2006
%
%*****
format compact
[n,n1] = size(A);
if abs(n - n1) > 0
    disp('input matrix must be square!')
    return
end
nm1 = n - 1;
alph = zeros(1,(n + 1));
alpha = alph;
a = alph;
a(1) = 1;
SIG = zeros(n,n,n); % a tensor, or row of matrices
In = eye(n);
en = In(:,n);
enm1 = In(:,nm1);
e1 = In(:,1);
SIGn = In;
SIG(:, :, n) = SIGn;
% now use Leverrier/Souriau-Frame/Fadeeva algorithm:
for j = 1:(n-1)
    SIGnmjpl = SIG(:, :, (n - j + 1));
    ASnmjpl = A*SIGnmjpl;
    ajpl = - trace(ASnmjpl)/j;
    a(j + 1) = ajpl;
    Snmj = ajpl*In + ASnmjpl;
    SIG(:, :, (n - j)) = Snmj;
end
SIGnmjpl = SIG(:, :, 1);
ASnmjpl = A*SIGnmjpl;
ajpl = - trace(ASnmjpl)/n;
a(n + 1) = ajpl;
So = ajpl*In + ASnmjpl;
% check: norm(So) = zero ?
alph = a;

```

```

np1 = n + 1;
alf = alph(2:np1);
alfa = fliplr(alf);
alfa = alfa';
Enm1 = zeros(n,n);
E = Enm1;
for k = 1:nm1
    Enm1(k,k+1) = 1;
end
compA = Enm1 - en*alfa';
un = en
E(:,n) = un;
for k = 1:nm1
    uk = E(:,n-k+1);
    ukm1 = A*uk + alf(k)*un;
    E(:,n-k) = ukm1;
end
% chk = norm(A*E-E*compA) % was 1/10^16 so algorithm's OK!
% end of matCompanion.txt

```

```

function [alph,SIG] = matSpec(A);
%*****
%
% Usage: [alph,SIG] = matSpec(A);
%
% Output: alph = row-vector defining [in reverse order] a polynomial
%         whose roots are the characteristic roots of A;
%
%         Let  $d(s) = \det(s.In - A)$  denote the characertistic polynomial.
%         If  $d(s) = s^n + \alpha[n-1].s^{[n-1]} + \dots + \alpha[0]$ ,
%         then  $\alpha(k) = \alpha[n - k + 1]$ , ( $k = 1, 2, 3, \dots, n+1$ ),
%         where, of course,  $\alpha[n] = 1$  always. Alternatively,
%          $\alpha[j] = \alpha[n - j + 1]$ , ( $j = 0, 1, 2, \dots, n$ ).
%
% Output: SIG is a tensor such that  $S_k = \text{SIG}(:, :, k)$ , ( $k = 1, 2, \dots, n$ ),
%         are matrices such that  $S_n = \text{eye}(n) = \text{diag}(1, 1, 1, \dots, 1)$  &
%
%  $d(s) \cdot \text{inv}(s.In - A) = S_1 + S_2.s + \dots + S_k.s^{[k-1]} + \dots + S_n.s^{[n-1]}$ .
%
% Here the {Sk} matrices & coefficients {alphak} are computed recursively
% via the Leverrier/Fadeeva/Soureau/Frame Algorithm which,
% short of the final step of factorization of the polynomial,
% constitutes a complete "spectral analysis" of the matrix A.
%
%
% Copyright August 22, 2006 by Robert W. Bass
%*****
format compact
[n,n1] = size(A);
if abs(n - n1) > 0
    disp('input matarix must be square!')
    return
end
alph = zeros(1,(n + 1));
a = alph;
a(1) = 1;
SIG = zeros(n,n,n);
In = eye(n);
SIGn = In; % S_n = In, nmj = n-j, nmjpl = n-j+1
SIG(:, :, n) = In;
for j = 1:(n-1)
    SIGnmjpl = SIG(:, :, (n - j + 1));
    ASnmjpl = A*SIGnmjpl;
    ajpl = - trace(ASnmjpl)/j; % alpha_nmj = -(1/j).tr(A.S_nmjpl)
    a(j + 1) = ajpl;
    Snmj = ajpl*In + ASnmjpl; % S_nmj = alpha_nmj.In + A.S_nmjpl
    SIG(:, :, (n - j)) = Snmj;
end
SIGnmjpl = SIG(:, :, 1);
ASnmjpl = A*SIGnmjpl;
ajpl = - trace(ASnmjpl)/n;
a(n + 1) = ajpl;
alph = a; % alpha = fliplr(alph)
% end of matSpec.m

```

```

function [uV] = generateChirpInput(omega1,phase1,omega2,phase2,N);
%*****
%
% usage: [uV] = generateChirpInput(omega1,phase1,omega2,phase2,N);%
%
% inputs: omegai (i = 1,2) = frequency
%          phasei (i = 1,2) = phase angle
%          N = number of samples
%
% an example that worked well was:
%       N = 100
% omega1 = pi/44,      omega2 = pi/(33^2),
% phase1 = 0.25*pi,   phase2 = 0.137*pi
%
% Copyright June 24, 2006 by Robert W. Bass
%*****
uV = zeros(N,1);
for k = 1:N
    uV(k) = sin(omega1*k + phase1) + sin(omega2*k*k + phase2);
end
% end of generateInput.m

```

```

function [A,B,C,Y,U] = generateNoisyExamplesMIMO(N,n,m,el,sigu,sigy);
%*****
%
% usage:
%   [A,B,C,Y,U] = generateNoisyExamples(N,n,m,el,sigu,sigy);
%
% inputs:
%
%   N = number of input & output samples at time-epochs k = 1, 2, ..., N
%   n = dimension of system state-vector x(k), k = 1, 2, ..., N
%   m = number of input signal channels
%   el = number of output signal channels
%   sigu = std of noise added to input u before generation of output y
%   sigy = std of noise added to output y
%
% outputs
%
%   A = system state-transition matrix (in companion form [w.l.o.g.])
%   B = system input-coupling matrix
%   C = system output-coupling matrix
%   U = N-by-m matrix of system input samples
%   Y = N-by-el matrix of system output samples generated by:
%
%   xnew = zeros(n,1)
%   for k = 1:N
%       xold = xnew;
%       uk = U(k,:)' ;
%       xnew = A.xold + B.uk;
%       yk = C.xold;
%       Y(k,:) = yk';
%
% Copyright June 24, 2006 by Robert W. Bass
%*****
format compact;
if m > 4
    disp('m cannot exceed 4')
    return
end
if el > n
    disp('el cannot exceed n')
    return
end
if m > n
    disp('m cannot exceed n')
    return
end
ndx = (1:N)';
np1 = n + 1;
ply = zeros(np1,1);
ply(1) = 1;
ply(np1) = 0.5^n;
rootz = roots(ply);
a = poly(rootz);
a(1) = [];
alf = -fliplr(a);
A = zeros(n,n);
nml = n-1;
for k = 1:nml
    A(k,k+1) = 1;
end

```

```

A(n,:) = alf;
B = zeros(n,m);
for k = 1:m
    B(n-k+1,k) = 1;
end
C = zeros(el,n);
for j = 1:el
    for k = 1:j
        C(j,k) = 1;
    end
end
end
x = zeros(n,1);
xnew = x;
u = zeros(N,1);
step = u;
No3 = round(N/3);
No3p1 = No3 + 1;
No3pNo3p1 = 2*No3 + 1;
for k = No3p1:(2*No3)
    step(k) = -0.5;
end
for k = No3pNo3p1:N
    step(k) = 0.5;
end
y = u;
omega1 = pi/44;
omega2 = pi/(33^2);
omg = [omega1 omega2]';
phase1 = 0.25*pi;
phase2 = 0.137*pi;
phs = [phase1 phase2]';
[u1] = generateChirpInput(omg(1),phs(1),omg(2),phs(2),N);
u1 = u1/max(abs(u1));
omg1 = omg.^0.5;
phs1 = phs.^0.5;
[u2] = generateChirpInput(omg1(1),phs1(1),omg1(2),phs1(2),N);
u2 = u2 + sin((2*pi/N)*ndx) + step;
u2 = u2/max(abs(u2));
omg2 = omg.^(1/3);
phs2 = phs.^(1/3);
[u3] = generateChirpInput(omg2(1),phs2(1),omg2(2),phs2(2),N);
u3 = u3 + sin((pi/N)*ndx);
u3 = step.*u3;
u3 = u3/max(abs(u3));
omg3 = omg.^(1/4);
phs3 = phs.^(1/4);
[u4] = generateChirpInput(omg3(1),phs3(1),omg3(2),phs3(2),N);
u4 = u4 + sin((pi/(2*N))*ndx);
u4 = u4/max(abs(u4));
Uo = [u1 u2 u3 u4];
U = Uo(:,1);
for j = 2:m
    U = [U Uo(:,j)];
end
Unoise = sigu*randn(size(U));
U = U + Unoise;
Y = zeros(N,el);
Ynoise = sigy*randn(size(Y));
xnew = zeros(n,1);
for k = 1:N
    xold = xnew;
    uk = U(k,:)' ;

```

```

    xnew = A*xold + B*uk;
    yk = C*xold;
    Y(k,:) = yk';
end
Y = Y + Ynoise;
for j = 1:el
    Y(1,j) = 0;
end
sclfctr = max(abs(Y));
Yscl = Y/sclfctr;
T = (1:N)';
plot(T,U,'xr',T,Yscl,'+b',T,U,'-r',T,Yscl,':b');
xlabel('Time');ylabel('Amplitude');
title('Example of Input/Output Pairs');
legend('xxx input','+++ output')
% end of generateNoisyExamplesMIMO.m

```

```

function [H,zH] = getHankelMatrix(n,MARKOV);
%*****
%
% usage: [H,zH] = getHankelMatrix(n,MARKOV);
%
% inputs: n = presumed system state-dimension (McMillan Degree)
%         MARKOV = tensor of Markov Parameters  $M_k = C.A^{k-1}.B$ ,  $k = 1, 2, \dots, 2.n$ 
%
% outputs: H = HANKEL =
%           Hankel Matrix of block-matrix size (n+1)-by-(n+1) where the
%           block in the jth block-row and kth block-column is
%            $M_q$  ( $q = j+k-1$ )
%
% Copyright September 4, 2006 by Robert W. Bass
%*****
format compact
[el,m,npn] = size(MARKOV);
nl = (npn)/2;
if n > nl
    disp('input n cannot exceed the n implicit in MARKOV')
    return
end
np1 = n + 1;
np2 = n + 2;
J = el*np1;
J1 = el*n;
K = m*np1;
K1 = m*n;
HANKEL = zeros(J1,K1);
for j = 1:n
    for k = 1:n
        HANKEL(((j-1)*el+1):(j*el),((k-1)*m+1):(k*m)) = MARKOV(:,j+k-1);
    end
end
H = HANKEL;
zH = zeros(J1,K1);
for j = 1:n
    for k = 1:n
        zH(((j-1)*el+1):(j*el),((k-1)*m+1):(k*m)) = MARKOV(:,j+k);
    end
end
% end of getHankelMatrix.m

```

```

function [A,B,C] = HoKalmanHankel(n,H,zH);
%*****
%
% usage: [A,B,C] = HoKalmanHankel(n,H,zH);
%
% inputs:
%     n = McMillan Degree
%     H = Hankel matrix of size el.n-by-m.n
%     zH = shifted Hankel matrix (same size but shifted one time-step forward)
%
% outputs:
%     (A,B,C) = LTI system ID
%
% Copyright September 4, 2006 by Robert W. Bass
%*****
format compact
rnkH = rank(H);
rnkzH = rank(zH);
nstr = num2str(n);
rnkHstr = num2str(rnkH);
rnkzHstr = num2str(rnkzH);
rnktst = abs(rnkH - n) + abs(rnkzH - n);
if rnktst > 0
    disp(['problem with assuming n = ' nstr])
    disp(['rank of H is = ' rnkHstr])
    disp(['rank of zH is = ' rnkzHstr])
    pause
end
[p,q] = size(H);
el = p/n;
m = q/n;
[U,S,V] = svd(H);
sig = svd(H);
SL = zeros(p,p);
SR = zeros(q,q);
for k = 1:n
    SL(k,k) = 1/sqrt(sig(k));
    SR(k,k) = SL(k,k);
end
P = SL*U';
M = V*SR;
E1 = zeros(n,p);
E2 = zeros(q,n);
E3 = zeros(q,m);
E4 = zeros(el,p);
for k = 1:n
    E1(k,k) = 1;
    E2(k,k) = 1;
end
for k = 1:m
    E3(k,k) = 1;
end
for k = 1:el
    E4(k,k) = 1;
end
A = E1*P*zH*M*E2;

B = E1*P*H*E3;
C = E4*H*M*E2;
% end of HoKalmanHankel.m

```



```

function [ALPHA] = get_ALPHmat(alpha);
%*****
%
% Usage:  [ALPHA] = get_ALPHmat(alpha);
%
% Input:  alpha = row-vector defining polynomial
%         whose roots are the characteristic
%         roots of F;
%
%         if  $d(s) = s^n + \alpha[n-1].s^{n-1} + \dots + \alpha[0]$ ,
%         then  $\alpha(k) = \alpha[n - k + 1]$ , ( $k = 1, 2, 3, \dots, n+1$ ),
%         where, of course,  $\alpha[n] = 1$  always.  Alternatively,
%          $\alpha[j] = \alpha(n - j + 1)$ , ( $j = 0, 1, 2, \dots, n$ ).
%
% Output: ALPHA = upper-triangular matrix with ones on principal
%         counter-diagonal and coefficients alpha starting
%         at  $\alpha(1)$  in upper LHC and shifted to left
%         at each next row
%
% Copyright June 24, 2006 by Robert W. Bass
%*****
format compact
[n1,n] = size(alpha);
if abs(1 - n1) > 0
    disp('input vector must be a row!')
    return
end
n = n-1;
ALPHA = zeros(n,n);
for i = 1:n
    for j = 1:(n - i + 1)
        ALPHA(i,j) = alpha(n - i - j + 2);
    end
end
% end of get_ALPHmat.m

```

```

function [Aest,Best,Cest] = getIDb(Aest,Bet);
%*****
%
% usage: [Aest,Best,Cest] = getIDb(Aest,Bet); % use \ instead of inv
%
% inputs: Aest = state-transition matrix in companion-matrix form
%         Bet  = n.m-by-el matrix of Leverrier Parameters
%
% outputs: (Aest,Best,Cest) = LTI system ID in MIMO case
%
% Copyright September 4, 2006 by Robert W. Bass
%*****
format compact
[n,n1] = size(Aest);
if norm(n - n1) > 0
    disp('input Aest must be a square matrix')
    return
end
npl = n+1;
npr = 2*n;
[mn,el] = size(Bet);
m = mn/n;
[alph,SIG] = matSpec(Aest);
[ALPHA] = get_ALPHmat(alph);
%[bet,BETA]=get_BETA(alph);% line 37: backslash instead of BETA = inv(ALPHA)
MARKOVt = zeros(n*m,el);
BetDiv = zeros(n,m,el);
MarkovDiv = BetDiv;
for k = 1:m
    for j = 1:n
        for q = 1:el
            BetDiv(j,k,q) = Bet(k + m*(j-1),q);
        end
    end
end
for q = 1:el
    MarkovDiv(:, :, q) = flipud(ALPHA\BetDiv(:, :, q));
end
for k = 1:m
    for j = 1:n
        for q = 1:el
            MARKOVt(k+m*(j-1),q) = MarkovDiv(j,k,q);
        end
    end
end
end
MARKOVmat = MARKOVt'; % MARKOVmat = block matrix [M1,M2, ... ,Mn]
%                               where Mk = C.A^[k-1].B is an el-by-m matrix
alfV = alph(2:npl)';
alfV = flipud(alfV);
alfVmat = kron(alfV,eye(m));
MARKOV = zeros(el,m,npr);
for k = 1:n
    MARKOV(:, :, k) = MARKOVmat(1:el, (1+(k-1)*m):k*m); % 1st half of MARKOV
end
MARKOVmatnew = MARKOVmat;
for k = npl:npr
    MARKOVmatold = MARKOVmatnew;
    Mknew = -MARKOVmatold*alfVmat;
    MARKOVmatnew(:, 1:m) = [];
end

```

```
MARKOVmatnew = [MARKOVmatnew Mknew];  
end  
for k = np1:npn  
    MARKOV(:, :, k) = MARKOVmatnew(1:el, (1+(k-1-n)*m):(k-n)*m); % MARKOV, 2nd half  
end  
[H, zH] = getHankelMatrix(n, MARKOV);  
[Aest, Best, Cest] = HoKalmanHankel(n, H, zH);  
% end of getIDb.m
```

```

function [Aest,Best,Cest,Yprd,AICtest,sigest,res] = LTIsystemID(n,U,Y);
%*****
%
% usage: [Aest,Best,Cest,Yprd,AICtest,sigest,res] = LTIsystemID(n,U,Y);
%           where as usual
%           LTI = Linear Time Invariant
% systemID = System Identification, i.e. estimate (A,B,C) from (n,U,Y)
%           [note: here Multiple Input Multiple Output (MIMO) is allowed]
%
% inputs:
%   n = state-vector dimension [assumed given or hypothesized]
%   U = N-by-el input-data matrix such that
%       U(:,k) = column N-vector of inputs {uk(j) | j = 1,2, ..., N}, k = 1,2,...,m
%   Y = N-by-m output-data matrix such that
%       Y(:,k) = column N-vector of outputs {yk(j) | j = 1,2, ..., N}, k = 1,2,...,el
%           where y(1) = 0 {assuming system started from quiescence and
%           no direct [instantaneous] feed-through of input to output and
%           that input/output measurements are "perfect," i.e.
%           no process-disturbances nor measurement noises
%
% outputs:
%   Aest = n-by-n state-transition dynamics matrix, estimate of A
%   Best = n-by-m input-coupling [or actuator-signal] kinematics matrix,
%           estimate of B
%   Cest = el-by-n output-coupling [or sensor-signal] kinematics matrix,
%           estimate of C
%   Yprd = predicted output from given input, using the sysIDest
%   AICtest = relative prediction error, weighted by Akaiki Info Criterion
%   sigest = crude estimate of rms mean std of additive noises on u & y
%   res = normalized (or percent-relative) residual error in the
%           solution of the linear equations involving [U,Y) & IDest
%
%           such that the unknown LTI system can be modeled as
%
%            $x(k+1) = Aest.x(k) + Best.u(k), \quad (k = 1,2, \dots, N)$ 
%            $y(k) = Cest.x(k),$ 
%
%           where ' denotes vector-matrix transposition and
%           where x(k) is a column n-vector whose initial value x(1) = [0,...,0]', and
%           where u = [u1 u2 ... um]' and y = [y1 y2 ... yel]'.
%
%           Copyright 9/04/06 by Robert W. Bass
%*****
format compact
[N,m] = size(U);
if norm(Y(1,:)) > 0
    disp('each y(1,k), (k = 1, 2, ..., el) must be zero')
    return
end
[N1,el] = size(Y);
if abs(N - N1) > 0
    disp('both inputs must have same length')
    return
end
np1 = n+1;
Nm1 = N-1;
Nmn = N - n;
if ((el+m)*n - Nm1) > 0
    disp('N must exceed (el+m).n')

```

```

    return
end
sn = (el + m)*n;
Nmsn = N - sn;
Um = zeros(Nmn,m*n);
Ym = zeros(Nmn,el*n);
yNm = zeros(Nmn,el);
In = eye(n);
Isn = eye(sn);
en = zeros(n,1);
en(n) = 1;
Enml = zeros(n,n);
for k = 2:n
    for j = 1:k
        Enml(j,k) = In(j,k-1);
    end
end
% in theory, Enml^(nml) = 0*In
for j = 1:Nmn
    for k = 1:n
        for q = 1:m
            Um(j,m*(k-1)+q) = U(n-k+j,q);
        end
    end
end
for j = 1:Nmn
    for k = 1:n
        for q = 1:el
            Ym(j,el*(k-1)+q) = Y(n-k+j,q);
        end
    end
end
for j = 1:Nmn
    for q = 1:el
        yNm(j,q) = Y(n + j,q);
    end
end
Coeff = [-Ym Um];
% now yNm = -Ym.Alf + Um.Bet, where Ym is output data & Um is input data and
% Alf is a block n-vector of coefficients of the characteristic polynomial
% each multiplied by an el-by-el identity matrix Iel, while
% Bet is a block n-vector of transposed matrix coefficients of the numerator
% matrix-polynomial, each of which is an m-by-el matrix so that the
% Z-domain input/output transfer matrix is completely determined by the
% ratio of these two polynomials in the inverse of the delay operator z[-1]
zNm = Coeff'*yNm;
Prod = Coeff'*Coeff; % now zNm = Prod.AlfBet, where AlfBet = [Alf;Bet]
AlfBet = Prod\zNm;
Alf = AlfBet(1:el*n,:);
alf = zeros(n,1);
for q = 1:n
    Alfblockq = Alf(1+(q-1)*el:q*el,:);
    alfq = trace(Alfblockq);
    alf(q) = alfq/el;
end
Bet = AlfBet((el*n+1):((el+m)*n),:);
Aestc = Enml - en*flipud(alf)';
maxabspole=max(abs(eig(Aestc)));
if maxabspole > 1
    disp('identified system is unstable')
end
[Aest,Best,Cest] = getIDb(Aestc,Bet); % unpack Leverrier Parameters via HoKalmanHankel

```

```

xnew = zeros(n,1);
Yprd = 0*Y;
for k = 1:N
    xold = xnew;
    uk = U(k,:)';
    xnew = Aest*xold + Best*uk;
    Yprd(k,:) = (Cest*xold)';
end
AICfctr = (N + n)/Nm;
AICtest = (norm(Yprd - Y)/norm(Y))*AICfctr;
% disp('small AICtest implies output prediction success')
rm = yNm - Coeff*AlfBet; % residual error
tstq1 = norm(yNm);
tstq2 = norm(yNm - rm);
tstq3 = norm(AlfBet);
if tstq2 >= tstq1
    disp('sig cannot be estimated, even crudely')
end
sigsq = (tstq1^2 - tstq2^2)/tstq3^2;
sigest = sqrt(sigsq); % rms mean of sigu & sigy
res = norm(rm)/norm(yNm);
% end of LTIsystemID.m

```

```

function [Aest,Best,Cest,Yprd,AICtest,sigest,res] = LTIsystemIDa(nmax,U,Y);
%*****
%
% The suffix a stands for "automatic state-dimension n selection"
%
% usage: [Aest,Best,Cest,Yprd,AICtest,sigest,res] = LTIsystemIDa(nmax,U,Y);
%           where as usual
%
%       LTI = Linear Time Invariant
% systemID = System Identification, i.e. estimate (A,B,C) from (n,U,Y)
%           [note: Multiple Input Multiple Output (MIMO) is allowed]
%
% inputs:
%       nmax = maximum state-vector dimension to be considered
%       U = N-by-el input-data matrix such that
%       U(:,k) = column N-vector of inputs {uk(j) | j = 1,2, ..., N}, k = 1,2,...,m
%       Y = N-by-m output-data matrix such that
%       Y(:,k) = column N-vector of outputs {yk(j) | j = 1,2, ..., N}, k = 1,2,...,el
%           where y(1) = 0 {assuming system started from quiescence and
%           no direct [instantaneous] feed-through of input to output and
%           that input/output measurements are "perfect," i.e.
%           no process-disturbances nor measurement noises
%
% outputs:
%       Aest = n-by-n state-transition dynamics matrix, estimate of A
%       Best = n-by-m input-coupling [or actuator-signal] kinematics matrix,
%           estimate of B
%       Cest = el-by-n output-coupling [or sensor-signal] kinematics matrix,
%           estimate of C
%       Yprd = predicted output from given input, using the sysIDest
%       AICtest = relative prediction error, weighted by Akaiki Info Criterion
%       sigest = crude estimate of rms mean std of additive noises on u & y
%       res = normalized (or percent-relative) residual error in the
%           solution of the linear equations involving [U,Y) & IDest
%
%       such that the unknown LTI system can be modeled as
%
%        $x(k+1) = Aest.x(k) + Best.u(k), \quad (k = 1,2, \dots, N)$ 
%        $y(k) = Cest.x(k),$ 
%
%       where ' denotes vector-matrix transposition and
%       where x(k) is a column n-vector whose initial value x(1) = [0,...,0]', and
%       where u = [u1 u2 ... um]' and y = [y1 y2 ... yel]'.
%
%       Copyright 9/04/06 by Robert W. Bass
%*****
format compact
[N,m] = size(U);
[N1,e1] = size(Y);
if abs(N - N1) > 0
    disp('inputs U & Y must have the same number of rows!')
    return
end
nvec = [2 e1 m]';
nmin = max(nvec);
AICtstvec = zeros(nmax-nmin+1,1);
tic
for k = nmin:nmax
[Aest,Best,Cest,Yprd,AICtestk,sigest,res] = LTIsystemID(k,U,Y);
if AICtestk > 1
    AICtestk = 1;

```

```

end
if AICtestk < 1/10^5
    nmax = k
    break
end
AICtstvec(k-nmin+1) = AICtestk;
end
ndx = (nmin:nmax)';
[AICtestmin,kmin] = min(AICtstvec);
n = kmin + nmin - 1;
[Aest,Best,Cest,Yprd,AICtest,sigest,res] = LTIsystemID(n,U,Y);
toc
AbsEigAest = abs(eig(Aest))
AICtest_is = AICtest % normalized prediction error weighted by AIC increasing-n-penalty
sigest_is = sigest
res_is = res
% end of LTIsystemIDa.m

```

```

function[Aest,Best,Cest,Yprd,AICtest,sigest,res] = LTISystemIDm(nmax,U,Y);
%*****
%
% The suffix m stands for "manual state-dimension n selection"
%
% usage: [Aest,Best,Cest,Yprd,AICtest,sigest,res] = LTISystemIDm(nmax,U,Y);
%               where as usual
%           LTI = Linear Time Invariant
% systemID = System Identification, i.e. estimate (A,B,C) from (n,U,Y)
%           [note: Multiple Input Multiple Output (MIMO) is allowed]
%
% inputs:
%   nmax = maximum state-vector dimension to be considered
%   U = N-by-el input-data matrix such that
%       U(:,k) = column N-vector of inputs {uk(j) | j = 1,2, ..., N}, k = 1,2,...,m
%   Y = N-by-m output-data matrix such that
%       Y(:,k) = column N-vector of outputs {yk(j) | j = 1,2, ..., N}, k = 1,2,...,el
%           where y(1) = 0 {assuming system started from quiescence and
%           no direct [instantaneous] feed-through of input to output and
%           that input/output measurements are "perfect," i.e.
%           no process-disturbances nor measurement noises
%
% outputs:
%   Aest = n-by-n state-transition dynamics matrix, estimate of A
%   Best = n-by-m input-coupling [or actuator-signal] kinematics matrix,
%           estimate of B
%   Cest = el-by-n output-coupling [or sensor-signal] kinematics matrix,
%           estimate of C
%   Yprd = predicted output from given input, using the sysIDest
%   AICtest = relative prediction error, weighted by Akaiki Info Criterion
%   sigest = crude estimate of rms mean std of additive noises on u & y
%   res = normalized (or percent-relative) residual error in the
%           solution of the linear equations involving [U,Y) & IDest
%
%           such that the unknown LTI system can be modeled as
%
%            $x(k+1) = Aest.x(k) + Best.u(k), \quad (k = 1,2, \dots, N)$ 
%            $y(k) = Cest.x(k),$ 
%
%           where ' denotes vector-matrix transposition and
%           where x(k) is a column n-vector whose initial value x(1) = [0,...,0]', and
%           where u = [u1 u2 ... um]' and y = [y1 y2 ... yell]'.
%
%           Copyright 9/04/06 by Robert W. Bass
%*****
format compact
[N,m] = size(U);
[N1,el] = size(Y);
if abs(N - N1) > 0
    disp('inputs U & Y must have the same number of rows!')
    return
end
nvec = [2 el m]';
nmin = max(nvec);
AICtestvec = zeros(nmax-nmin+1,1);
tic
for k = nmin:nmax
    [Aestc,Aest,Best,Cest,Yprd,AICtestk,sigest,res] = LTISystemID(k,U,Y);
    if AICtestk > 1
        AICtestk = 1;
    end
end

```

```

end
if AICtestk < 1/10^5
    nmax = k
    break
end
AICtestvec(k-nmin+1) = AICtestk;
end
ndx = (nmin:nmax)';
toc
plot(ndx,AICtestvec);xlabel('assumed state-dimension n');ylabel('AICtest');
title('Weighted Prediction-Error')
pause
format long
disp('          n          AICtest')
[ndx AICtestvec]
[AICtestmin,kmin] = min(AICtestvec);
n_optimal = kmin + nmin - 1
pause
format short
n = input('INPUT selected n = ')
[Aest,Best,Cest,Yprd,AICtest,sigest,res] = LTISystemID(n,U,Y);
AbsEigAest = abs(eig(Aest))
AICtest_is = AICtest % normalized prediction error weighted by AIC increasing-n-penalty
sigest_is = sigest
res_is = res
% end of LTISystemIDm.m

```

UPDATE 3/7/10: replace LTIsysID by the following improved version:

```
function [n_est,Aest,Best,Cest,Dest] = LTIsysIDviaHKLevAndN4sidPEM(nmin,nmax,Y,U);
%*****
%
% use:
%   [n_est,Aest,Best,Cest,Dest] = LTIsysIDviaHKLevAndN4sidPEM(nmin,nmax,Y,U);
%
% inputs:
%   nmin = minimum state-vector dimension n > 1 to be considered
%   nmax = maximum state-vector dimension n > 1 to be considered
%   Y = N-by-el data-matrix of el-channel system-output measured at
%       N successive epochs of discrete time
%   U = N-by-m data matrix of m measured system inputs, at N epochs,
%       of an initially quiescent system (started from equilibrium),
%       assuming that small inputs yield small outputs and that the
%       system (at least, sufficiently near to equilibrium) may be
%       modeled as a Linear Time Invariant (LTI) system
% outputs:
%   n_est = optimal estimate of system state-vector dimension n
%(Aest,Best,Cest,Dest) = optimal estimates of matrices such that if [absent noise]
%   PHI(z) = C.inv(z.In - A).B is an el-by-m Z-transform then
%   Y(z) = PHI(z).U(z) + D.U(z) identifies input/ouput relationship
%
% Copyright July 4, 2006 by Robert W. Bass
% Acknowledgement: This was based largely on work supported by DARPA under
% Purchase Order HR0011-06-P-0016, January 13, 2006, Final Report published at:
% http://www.innoventek.com/Bass2006\_HoKalmanViaLeverrierResolvent.pdf [9/05/06]
%
% Improved in March, 2010 by inclusion of N4sid & PEM, in part as suggested by Lennart Ljung.
% The original 2006 version used the HoKalman Lemma implemented by Leverrier's Algorithm,
% which may be numerically fragile, though in tests this approach does best in finding the optimal
% state-dimension n. Subsequently the MATLAB Toolbox IDENT function PEM is used, which starts with
% finding a perliminary ID via the purely linear algebra of the Subspace approach, via N4sid,
% followed by its nonlinear iterative refinement via the most powerful functional-minimization
% methods used in PEM, that depend upon gradient estimation & steep-descent line search.
%
%*****
format compact
[N,m] = size(U);
[N1,el] = size(Y);
if abs(N - N1) > 0
    disp('inputs U & Y must have the same number of rows!')
    return
end
nvec = [2 el m]';
nmin0 = max(nvec);
nmin = max([nmin;nmin0]);
Nmnmin = N - nmin;
% nmax = 38;      <== worked OK with nmax this large!
if el > nmax
    disp('number of output columns el must not exceed nmax')
    return
end
if m > nmax
    disp('number of input columns m must not exceed nmax')
    return
end
if (el + m + 1)*nmax > Nmnmin
    disp(' (el + m + 1).nmax must not exceed (N - n) ')
    return
end
```

```

end
z = iddata(Y,U);
[Yscl,Scly,Ymean] = dataprep(Y);
[Uscl,Scly,Umean] = dataprep(U);
for k = 1:el
    Y(1,k) = 0;
end
for k = nmin:nmax
    nk = k;
[Aest,Best,Cest,Yprd,AICtestk,sigest,res] = LTIsysID_MIMO(nk,Uscl,Yscl);
AICvec(k-nmin+1,1) = AICtestk;
    if AICtestk < 1/10^5
        nmax = k
        break
    end
end
end
ndx = (nmin:nmax)';
lngth = length(ndx);          % lngth = nmax - nmin + 1
AICvect = AICvec(1:lngth,1);
[minAICvect,nopt] = min(AICvect);
noptimal = nopt + nmin - 1;
nOPT = noptimal;
m = pem(z,nOPT);
n_est = nOPT;
Aest = m.a;
Best = m.b;
Cest = m.c;
Dest = m.d;
% end of LTIsysIDviaHKLevAndN4sidPEM.m

```