Software for exact linear system identification

Ivan Markovsky, Jan C. Willems, and Bart De Moor

K.U.Leuven, ESAT-SCD, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium,

{ivan.markovsky,sabine.vanhuffel}@esat.kuleuven.ac.be

***Abstract*— A MATLAB toolbox for exact linear time-invariant system identification is presented. The emphasis is on the variety of possible ways to implement the mappings from data to parameters of the data generating system. The considered system representations are input/state/output, difference equation, and left matrix fraction. The reader is referred to the literature for the theory behind the implemented algorithm. The paper shows the implementation details on the level of documented MATLAB code.**

Keywords: Exact system identification, numerical implementation, MATLAB.

INTRODUCTION

This document contains a MATLAB implementation of the algorithms for exact linear time-invariant (LTI) system identification presented in [3], [1]. The exact identification problem refers to the situation when the data is generated by a system in a considered deterministic model class. Correspondingly the aim is to recover the data generating system exactly from a given finite data record. In the literature the exact identification problem is also known as deterministic identification problem, see [5, Chapter 2].

Although the algorithms are designed to work with exact data, they can be applied as heuristic methods for approximate identification. The software can handle the approximate identification case because exact operations like rank computation and solution of a compatible system of equations are automatically replaced by corresponding approximate operations like numerical rank computation (up to a given tolerance) and solution of an overdetermined linear system of equations in a least squares sense. With such adaptation on the level of the numerical implementation, the exact identification algorithms become heuristic approximate identification algorithms. However, the simulation study reported in [2] indicates that alternative algorithms designed for approximate and stochastic identification outperform the exact identification algorithms when there is no exact model in the considered model class.

Our goal in this paper is to emphasize a variety of possible paths to realize the mapping

$$w = \mathrm{col}(u,y) \mapsto (A,B,C,D)$$

from exact data w to the parameters (A,B,C,D) of an input/state/output representation

$$x(t+1) = Ax(t) + Bu(t), \qquad y(t) = Cx(t) + Du(t). \tag{I/S/O}$$

of the data generating system. Moreover an LTI system can be represented in a number of equivalent forms. Apart from the input/state/output representation, we use the kernel representation

$$R_0 w(t) + R_1 w(t+1) + \cdots + R_{\mathtt{l}} w(t+\mathtt{l}) = 0. \tag{KER}$$

A kernel representation is called minimal if the number of equations $g := \mathrm{row\,dim}(R_i)$ is as small as possible. It turns out that in a minimal kernel representation (KER), g is equal to the number of outputs of the system. Taking into account the input/output partitioning $w = \mathrm{col}(u,y)$ and defining accordingly the partitioning of the coefficients $R_i =: \begin{bmatrix} Q_i & -P_i \end{bmatrix}$, the kernel representation (KER) becomes the left matrix fraction representation

$$P_0 y(t) + P_1 y(t+1) + \cdots + P_{\mathtt{l}} y(t+\mathtt{l}) = Q_0 u(t) + Q_1 u(t+1) + \cdots + Q_{\mathtt{l}} u(t+\mathtt{l}). \tag{LMF}$$

The exact identification algorithms are decomposed into elementary building blocks that have independent significance. Table I lists the building blocks together with short descriptions. Full details about the usage and implementation of the functions can be found in the documentation of the corresponding m-files, listed in the Appendix.

Using the building blocks a variety of algorithms for exact identification, aiming at an input/state/output representation, are shown in Table II. With exact data the algorithms are equivalent. However, they require different number of floating point operations and have different performance in the approximate identification case. The demo file `demo.m` is included in the package to test the algorithms' performance.

The software is available from

`ftp://ftp.esat.kuleuven.ac.be/pub/SISTA/markovsky/abstracts/05-122.html`

TABLE I

ELEMENTARY BUILDING BLOCKS FOR THE EXACT IDENTIFICATION ALGORITHMS. (∗ — REQUIRES THE POLYNOMIAL TOOLBOX VERSION 1.5.)

Function	Description
`w2r`*	from time series to a kernel representation
`r2pq`*	from a kernel representation to a left matrix fraction representation
`pq2ss`	from a left matrix fraction representation to an input/state/output representation
`uy2h`	recursive computation of the impulse response from data
`uy2hblk`	block computation of the impulse response from data
`h2ss`	from impulse response to an input/state/output representation (Kung's algorithm)
`uy2y0`	recursive computation of sequential free responses
`uy2hy0`	recursive computation of the impulse response and sequential free responses
`y02ox`	from a set of free responses to an observability matrix and a state sequence
`h2ox`	from an impulse responses to an observability matrix and a state sequence
`uyo2ss`	from data and an observability matrix to an input/state/output representation
`uyx2ss`	from data and a state sequence to an input/state/output representation
`hy02xbal`	from the impulse response and sequential free responses to a balanced state sequence

TABLE II

ALGORITHMS AIMING AT AN INPUT/STATE/OUTPUT (I/S/O) REPRESENTATION OF THE IDENTIFIED SYSTEM.

From data to a kernel representation to an I/S/O representation	`w2r`→`r2pq`→`pq2ss`
From data to impulse response to an I/S/O representation (by Kung's algorithm)	`uy2h`→`h2ss`
From data to impulse response to an I/S/O representation (another version)	`uy2h`→`h2o`→`uyo2ss`
From data to free responses to an I/S/O representation	`uy2y0`→`y02o`→`uyo2ss`
From data to a state sequence to an I/S/O representation	`uy2y0`→`y02x`→`uyx2ss`
From data to a balanced state sequence to a balanced I/S/O representation	`uy2hy0`→`hy02xbal`→`x2ss`
Balanced subspace algorithm of [5]	`uy2ssvd`
Balanced subspace algorithm of [4]	`uy2ssmr`

ACKNOWLEDGMENTS

Research supported by: **Research Council KUL:** GOA-Mefisto 666, GOA-Ambiorics, IDO/99/003 and IDO/02/009 (Predictive computer models for medical classification problems using patient data and expert knowledge), several PhD/postdoc & fellow grants; **Flemish Government: FWO:** PhD/postdoc grants, projects, G.0078.01 (structured matrices), G.0269.02 (magnetic resonance spectroscopic imaging), G.0270.02 (nonlinear Lp approximation), G.0240.99 (multilinear algebra), G.0407.02 (support vector machines), G.0197.02 (power islands), G.0141.03 (Identification and cryptography), G.0491.03 (control for intensive care glycemia), G.0120.03 (QIT), G.0452.04 (QC), G.0499.04 (robust SVM), research communities (ICCoS, ANMMM, MLDM); AWI: Bil. Int. Collaboration Hungary/ Poland; **IWT:** PhD Grants; GBOU (McKnow) **Belgian Federal Government:** DWTC (IUAP IV-02 (1996-2001) and Belgian Federal Science Policy Office IUAP V-22 (2002-2006) (Dynamical Systems and Control: Computation, Identification & Modelling)); PODO-II (CP/01/40: TMS and Sustainibility); **EU:** PDT-COIL, BIOPATTERN, eTUMOUR, FP5-Quprodis; ERNSI; Eureka 2063-IMPACT; Eureka 2419-FliTE; Contract Research/agreements: ISMC/IPCOS, Data4s, TML, Elia, LMS, IPCOS, Mastercard.

REFERENCES

[1] I. Markovsky. *Exact and approximate modeling in the behavioral setting*. PhD thesis, Dept. EE, K.U.Leuven, February 2005.

[2] I. Markovsky, J. C. Willems, and B. De Moor. Comparison of identification algorithms on the database for system identification DAISY. Technical Report 05–227, Dept. EE, K.U.Leuven, 2005.

[3] I. Markovsky, J. C. Willems, P. Rapisarda, and B. De Moor. Algorithms for deterministic balanced subspace identification. *Automatica*, 41(5):755–766, 2005.

[4] M. Moonen and J. Ramos. A subspace algorithm for balanced state space system identification. *IEEE Trans. on Aut. Control*, 38:1727–1729, 1993.

[5] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Kluwer, 1996.

APPENDIX

blkhank.m

```
% BLKHANK - Construct a block-Hankel matrix.
%
% H = blkhank(w,i,j)
%
% W - structure parameters W(:,:,i), i = 1,...,T
%  if W(:,:,i) are Qx1 vectors, W can be a TxNW matrix
% I - number of block rows
% J - optional number of block columns
%     default: J = T - I + 1 (the maximum)
% H - block-Hankel matrix parameterized by W

function H = blkhank(w,i,j)

if length(size(w)) == 3 % PxM matrix blocks
  [p,m,T]  = size(w);
  if nargin < 3 | isempty(j)
    j = T - i  + 1;
  end
  if j <= 0
    error('Not enough data.')
  end
  H = zeros(i*p,j*m);
  for ii = 1:i
    for jj = 1:j
      H((ii-1)*p+1:ii*p,(jj-1)*m+1:jj*m) = ...
          w(:,:,ii+jj-1);
    end
  end
else % DWx1 vector block
  [T,dw]  = size(w);
  if T < dw % for consistency with the subid functions
    w = w';
    [T,dw]  = size(w);
  end
  if nargin < 3 | isempty(j)
    j = T - i  + 1;
  end
  if j <= 0
    error('Not enough data.')
  end
  H=zeros(i*dw,j);
  w = w';
  for ii = 1:i
    H((ii-1)*dw+1:ii*dw,:) = w(:,ii:ii+j-1);
  end
end
```

h2ox.m

```
% H2OX - From an impulse response to an extended
%        observability matrix and a state sequence.
%
% [O,X] = h2ox(h,lmax,n,tol)
%
% H - first T>2*LMAX samples of the impulse response
%   for MIMO (M-inputs, P-outputs) system, H is PxMxT
%   for SISO system, H can be Tx1 vector
% LMAX - upper bound on the system lag
% N    - optional order of the system
% TOL  - tolerance for order selection, default 1e-7
% If TOL<0, the SVs are plotted and an input is asked.
% O - extended observability matrix
% X - state sequence

function [O,X] = h2ox(h,lmax,n,tol)

% Remove H(0) from H
if length(size(h)) == 2
  h = h(2:end,:);
else
  h = h(:,:,2:end);
end

% SVD of the Hankel matrix of Markov parameters
[U,S,V] = svd(blkhank(h,lmax+1),0);
s = diag(S);

% If not given, determine the order
if (nargin < 4 | isempty(tol))
  tol = [];
end
if (nargin < 3 | isempty(n))
  n = order(s,tol);
end

% Rank revealing factorization
sqrt_s = sqrt(1./s(1:n))';
O = sqrt_s(ones(size(U,1),1),:) .* U(:,1:n);
if nargout > 1
  X = (sqrt_s(ones(size(V,1),1),:) .* V(:,1:n))';
end
```

h2ss.m

```
% H2SS - From impulse response to an I/S/O
%        representation. (Kung's algorithm)
%
% sys = h2ss(h,n,tol)
%
% H - first T>2LMAX samples of the impulse response
%   for MIMO (M-inputs, P-outputs) system, H is PxMxT
%   for SISO system, H can be Tx1 vector
% N - optional system order
% TOL - tolerance for order selection, default 1e-7
% If TOL<0, the SVs are plotted and an input is asked.
% SYS - I/S/O representation in ss format
%       SYS is in a finite time floor(T/2) basis

function [sys1,sys2,sys3] = h2ss(h,n,tol)

% Check for the SI case
if length(size(h)) == 2
  T = size(h,1); m = 1; p = 1;
  d = h(1,:);
  h = h(2:end,:);
else
  [p,m,T] = size(h);
  d = h(:,:,1);
  h = h(:,:,2:end);
end

% SVD of the Hankel matrix of the Markov parameters
delta   = floor(T/2); % # of block rows
% Make the Hankel matrix as square as possible.
[U,S,V] = svd(blkhank(h,delta),0);
s = diag(S);

% Determine the order
if (nargin < 3 | isempty(tol))
  tol = [];
end
if (nargin < 2 | isempty(n))
  n = order(s,tol);
end

% Rank revealing factorization
sqrt_s = sqrt(s(1:n))';
O =  sqrt_s(ones(size(U,1),1),:) .* U(:,1:n);
C = (sqrt_s(ones(size(V,1),1),:) .* V(:,1:n))';
clear s sqrt_s U V

% Compute A,B,C,D
a1 = O(1:end-p,:)\O(p+1:end,:); % shift equation
a2 = C(:,m+1:end)/C(:,1:end-m); % shift equation
Cbar = C(:,m+1:end); Obar = O(p+1:end,:);
a3 = [kron(C(:,1:end-m)',eye(n));
      kron(eye(n),O(1:end-p,:))]\[Cbar(:);Obar(:)];
a3 = reshape(a3,n,n);

format long
e_obsv = norm(O(1:end-p,:)*a1 - O(p+1:end,:),'fro')...
         / norm(O(p+1:end,:),'fro');
e_ctrb = norm(a2*C(:,1:end-m) - C(:,m+1:end),'fro')...
         / norm(C(:,m+1:end),'fro');
format

b = C(:,1:m);
c = O(1:p,:);

sys1 = ss(a1,b,c,d,-1);
sys2 = ss(a2,b,c,d,-1);
sys3 = ss(a3,b,c,d,-1);
```

hy02xbal.m

```
% HY02XBAL - From impulse response and sequential
```

```
%    free responses to a balanced state sequence.
%
% XBAL = hy02xbal(h,y0,delta,n,tol)
%
% H - first T>2*DELTA samples of the impulse response
%   for MIMO (M-inputs, P-outputs) system, H is PxMxT
%   for SISO system, H can be Tx1 vector
% Y0 - matrix of DELTA samples sequential free resp.
%      (Y0(:,i) is such a response)
% DELTA - optional finite time balancing parameter
%  DELTA>LMAX+1, LMAX - upper bound on the system lag
%  default: DELTA = floor(T/2) (the maximum)
% N    - optional order of the system
% TOL - tolerance for order selection, default 1e-7
% If TOL<0, the SVs are plotted and an input is asked.
% XBAL - finite time DELTA balanced state sequence

function xbal = hy02xbal(h,y0,delta,n,tol)

% Remove H(0) from H
if length(size(h)) == 2
  T = size(h,1);
  h = h(2:end);
else
  T = size(h,3);
  h = h(:,:,2:end);
end

if nargin < 3 | isempty(delta)
  delta = floor(T/2);
end

% SVD of the Hankel matrix of the Markov parameters
[U,S,V] = svd(blkhank(h,delta),0);
s = diag(S);

% If not given, determine the order
if (nargin < 5 | isempty(tol))
  tol = [];
end
if (nargin < 4 | isempty(n))
  n = order(s,tol);
end

% Compute balanced state sequence
sqrt_s = sqrt(1./s(1:n));
xbal = (sqrt_s(:,ones(1,size(U,1))) .* U(:,1:n)') ...
       * y0(1:size(U,1),:);
```

pq2ss.m

```
% PQ2SS - from LMF to I/S/O representation.
%
% sys = pq2ss(p,q)
%
% P - PxMxL array of coefficients
%     for the polynomial P(z)
% Q - PxPxL array of coefficients
%     for the polynomial Q(z)
%     Note: for SISO system, P and Q can be vectors
% SYS - ss object of a minimal I/S/O representation
%
% Note: requires the Polynomial Toolbox Version 1.5

function sys = pq2ss(P,Q)

if length(size(P)) == 2 % SISO
  Q = Q(end:-1:1);
  P = P(end:-1:1);
  sys = ss(tf(Q,P,-1));
else % MIMO
  l1 = size(P,3); l = l1 - 1;
  m  = size(Q,1);
  p  = size(P,1);
  Q  = ppck(reshape(Q,p,l1*m),l);
  P  = ppck(reshape(P,p,l1*p),l);
  % call the Polynomial Toolbox
  [a,b,c,d] = lmf2ss(Q,P);
  sys = ss(a,b,c,d,-1);
end
```

r2pq.m

```
% R2PQ - From kernel to LMF representation.
%
%  [P,Q] = pq2ss(R)
%
% R - 3d array of parameter of a kernel represent.
%     ( R0 = R(:,:,1), ... ,Rl = R(:,:,l+1) )
% P,Q - 3d array of parameters of an LMF represent.
%     ( the same format as R )
%
% Note 1: the input/output partition is fixed-first
% M variables inputs, the rest outputs.
%
% Note 2: requires the Polynomial Toolbox Version 1.5

function [P,Q] = pq2ss(R)

% Constants
[g,dw,l1]  = size(R);
l  = l1 - 1;

% Find a minimal kernel representation (R rowreduced)
addpath ../polyx

% Unfold R in a 2d matrix [R0 R1 ... Rl]
R = reshape(R,g,(l1)*dw);

% Call polyx for the row reduction
R = ppck(R,l);
R = prowred(R,'z');

% Cancel zero rows
p = prank(R);
R = psel(R,1:p,':');
R = punpck(R);

% Transform R back to a 3d array
R = reshape(R,p,dw,l1);

% Partition R
m = dw - p;
Q = R(:,1:m,:);
P = -R(:,m+1:end,:);
```

uy2hblk.m

```
% UY2HBLK - From data to impulse response.
%           (Block algorithm)
%
% [h, delta] = uy2hblk(u, y, lmax, delta)
%
% U,Y - input (TxM matrix) and output (TxP matrix)
% LMAX  - upper bound for the system lag
% DELTA - desired length of the impulse response
%  default: the maximal possible number is used
% H - first DELTA samples of the impulse response
%  for MIMO (M-inputs, P-outputs) system, h is PxMxT
%  for SISO system, h can be Tx1 vector

function [h, delta] = uy2hblk(u,y,lmax,delta)

% Constants
[T,m] = size(u);
p = size(y,2);

if ( nargin < 4 | isempty(delta) )
  delta = ceil((T+1)/(m+1) - lmax * (1 + p));
  if delta < 1
    error(sprintf('Not enough data for DELTA=%d.', ...
                  delta));
  end
end

L = lmax+delta; % # of block rows of the Hankel matrix
nrows = L*m+lmax*p; % # of rows in the system (7.4)

% QR of the Hankel matrix of the data (LHS of (7.5))
r = triu(qr([blkhank(u,L); blkhank(y,L)]'))';

% Select the R11 and R21 blocks of R
r11 = r(1:nrows,1:nrows);
r21 = r(nrows+1:end,1:nrows);

% Right-hand side of the system (7.4)
B = zeros(nrows,m);
B(lmax*m+1:(lmax+1)*m,:) = eye(m);
```

```
% Solve the system (7.4) for G and compute H = Yf * G
h = r21 * pinv(r11) * B;

if ( m > 1 | p > 1 ) % MIMO
  % Construct a 3d output array
  h_ = h;
  h = zeros(p,m,delta);
  for i = 1:delta
    h(:,:,i) = h_((i-1)*p+1:i*p,:);
  end
else % SISO
  h = h(1:delta,:);
end
```

uy2h.m

```
% UY2H - From data to impulse response.
%        (Recursive algorithm)
%
% [h, delta] = uy2h(u, y, lmax, l, delta)
%
% U,Y - input (TxM matrix) and output (TxP matrix)
% LMAX  - upper bound for the system lag
% L     - number of samples in a computed block
% DELTA - desired length of the impulse response
%   If skipped the response is computed till
%   ||H(delta)||_F > EPS.
% H - first DELTA samples of the impulse response
%   for MIMO (M-inputs, P-outputs) system, h is PxMxT
%   for SISO system, h can be Tx1 vector

function [h,delta] = uy2h(u,y,lmax,l,delta)

% Constants
EPS  = 1e-5;
IMAX = 100;
[T,m] = size(u);
p = size(y,2);

% Check inputs
if (nargin < 3)
  error('Not enough input arguments');
end
if (nargin < 4 | isempty(l))
  l = lmax;
end
L = lmax + l; % # of rows in the Hankel matrix
if (nargin < 5 | isempty(delta))
  find_delta = 1;
  delta = IMAX;
else
  find_delta = 0;
end

% Needed in the iterative computation
r = triu(qr([blkhank(u,L);blkhank(y,L)]'))';
in  = 1:L*m+lmax*p;
r11 = r(in,in);
r21 = r(in(end)+1:end,in);
pinvUY1 = pinv(r11);
Y2      = r21;

% Initialization
fu = zeros(L*m,m);
fu(lmax*m+1:(lmax+1)*m,:) = eye(m);
fy = zeros(L*p,m);

% Iteration
imax = ceil(delta/l);
h = zeros(p*l*imax,m);
for i = 1:imax
  % Solve the system
  g = pinvUY1 * [fu;fy(1:lmax*p,:)];
  fy(lmax*p+1:end,:) = Y2 * g;
  % Store the result
  h((i-1)*p*l+1:i*p*l,:) = fy(lmax*p+1:end,:);
  % Check exit condition if delta is not given
  if find_delta
    if ( i*l*min(m,p) > 2*lmax & ...
         norm(fy(lmax*p+1:end,:),'fro') < EPS)
      delta = i*l;
      break;
    end
  end
  % Shift for the next iteration
  fu(1:m*l,:) = [];
  fu = [fu; zeros(m*l,m)];
  fy(1:p*l,:) = [];
  fy = [fy; zeros(p*l,m)];
end

if ( m > 1 | p > 1 ) % MIMO
  % Construct a 3d output array
  h_ = h;
  h = zeros(p,m,delta);
  for i = 1:delta
    h(:,:,i) = h_((i-1)*p+1:i*p,:);
  end
else % SISO
  h = h(1:delta,:);
end
```

uy2hy0.m

```
% UY2HY0 - From data to impulse and sequential
%          free responses.
%
% [h, y0, delta] = uy2hy0(u, y, lmax, l, delta, j)
%
% U,Y - input (TxM matrix) and output (TxP matrix)
% LMAX  - upper bound for the system lag
% L     - optional number of samples in a block
% DELTA - optional desired length of the responses
%   If skipped the response is computed till
%   ||H(delta)||_F > EPS
% J - optional number of free responses
% H  - first DELTA samples of the impulse response
%   for MIMO (M-inputs, P-outputs) system, h is PxMxT
%   for SISO system, h can be Tx1 vector
% Y0 - matrix of J, DELTA samples long sequential
%   free resp. the i-th response is Y(:,i)

function [h,y0,delta] = uy2h(u,y,lmax,l,delta,j)

% Constants
EPS  = 1e-5; % tolerance for the convergence of the
             % response
IMAX = 100;  % maximum response length
[T,m] = size(u);
p = size(y,2);

% Check inputs
if (nargin < 3)
  error('Not enough input arguments');
end
if (nargin < 4 | isempty(l))
  l = lmax;
end
L = lmax + l; % # of rows in the Hankel matrix
if (nargin < 5 | isempty(delta))
  find_delta = 1;
  delta = IMAX;
else
  find_delta = 0;
end
if (nargin < 6 | isempty(j))
  j = T - L + 1;
end

% SVD of the Hankel matrix of the data
r   = triu(qr([blkhank(u,L);blkhank(y,L)]'))';
in  = 1:L*m+lmax*p;
r11 = r(in,in);
r21 = r(in(end)+1:end,in);
pinvUY1 = pinv(r11);
Y2      = r21;

% Initialization for H
fu_h = zeros(L*m,m);
fu_h(lmax*m+1:(lmax+1)*m,:) = eye(m);
fy_h = zeros(L*p,m);

% Initialization for Y0
fu_y0 = zeros(L*m,j);
fu_y0(1:lmax*m,:) = blkhank(u,lmax,j);
fy_y0 = zeros(L*p,j);
fy_y0(1:lmax*p,:) = blkhank(y,lmax,j);

% Iteration
imax = ceil(delta/l);
```

```
h    = zeros(p*l*imax,m);
y0   = zeros(delta*p,j);
for i = 1:imax
  % Solve the system for H
  g = pinvUY1 * [fu_h; fy_h(1:p*lmax,:)];
  fy_h(p*lmax+1:end,:)  = Y2 * g;
  % Solve the system for Y0
  g = pinvUY1 * [fu_y0; fy_y0(1:p*lmax,:)];
  fy_y0(p*lmax+1:end,:) = Y2 * g;
  % Store the results
  h((i-1)*p*l+1:i*p*l,:)  = fy_h(p*lmax+1:end,:);
  y0((i-1)*l*p+1:i*l*p,:) = fy_y0(p*lmax+1:end,:);
  % Check exit condition (if delta is not given)
  if (find_delta & i*l*min(m,p) > ...
    2*lmax & norm(fy_h(p*lmax+1:end,:),'fro') < EPS)
    delta = i*l;
    break;
  end
  % Shift FU_H, FY_H for the next iteration
  fu_h(1:m*l,:) = [];
  fu_h = [fu_h; zeros(m*l,m)];
  fy_h(1:p*l,:) = [];
  fy_h = [fy_h; zeros(p*l,m)];
  % Shift FU_Y0, FY_Y0 for the next iteration
  fu_y0(1:m*l,:) = [];
  fu_y0 = [fu_y0; zeros(m*l,j)];
  fy_y0(1:p*l,:) = [];
  fy_y0 = [fy_y0; zeros(p*l,j)];
end

if ( m > 1 | p > 1 ) % MIMO
  % Construct a 3d output array
  h_ = h;
  h = zeros(p,m,delta);
  for i = 1:delta
    h(:,:,i) = h_((i-1)*p+1:i*p,:);
  end
else % SISO
  h = h(1:delta,:);
end

y0 = y0(1:delta*p,:);
```

uy2ssmr.m

```
% UY2SSMR - Subspace algorithm of Moonen-Ramos.
%
% sys = uy2ssmr(u,y,i,n,tol)
%
% U,Y - input (TxM matrix) and output (TxP matrix)
% I   - finite time balancing parameter
%    I > NMAX, an upper bound of the system order
% N   - optional system order
% TOL - tolerance for order selection, default 1e-7
% If TOL<0, the SVs are plotted and an input is asked.
% SYS - identified system in a ss format
%       in a finite time I balanced basis

function sys = uy2ssmr(u,y,i,n,tol)

% Constants
[T,m] = size(u);
p = size(y,2);
j = T - 2*i + 1;

% Hankel matrix of the data
Y  = blkhank(y,2*i,j);
Yp = Y(1:p*i,:);
Yf = Y(p*i+1:2*p*i,:);
U  = blkhank(u,2*i,j);
Up = U(1:m*i,:);
Uf = U(m*i+1:2*m*i,:);

% Step 1: compute the annihilators
H = [Up; Yp; Uf; Yf];
R = triu(qr(H'))';
[U_,S,V] = svd(R(:,1:size(H,1)));
t  = U_(:,(m+2*p)*i+1:end)';
t1 = t(:,1:m*i);
t2 = t(:,m*i+1:(m+p)*i);
t3 = t(:,(m+p)*i+1:(2*m+p)*i);
t4 = t(:,(2*m+p)*i+1:2*(m+p)*i);
pinvT4 = pinv(t4);

% Step 2: SVD of the Hankel matrix of Markov param.
[U_,S,V] = svd(pinvT4 * (t2 * pinvT4 * t3 - t1),0);
s = diag(S);

% Determine the order
if (nargin < 5 | isempty(tol))
  tol = [];
end
if (nargin < 4 | isempty(n))
  n = order(s,tol);
end

% Step 3: compute balanced state sequence
sqrt_s = sqrt(1./s(1:n));
T = sqrt_s(:,ones(1,size(U_,1))) .* U_(:,1:n)';
X = T *  pinvT4 * [t1 t2] * [U(1:m*i,:); Y(1:p*i,:)];

% Step 4: compute A,B,C,D
M = [X(:,2:end-1); y(i+1:i+j-2,:)'] / ...
    [X(:,1:end-2); u(i+1:i+j-2,:)'];

a = M(1:n,1:n);
b = M(1:n,n+1:end);
c = M(n+1:end,1:n);
d = M(n+1:end,n+1:end);

sys = ss(a,b,c,d,-1);
```

uy2ssvd.m

```
% UY2SSVD - Algorithm of Van Overschee-De Moor.
%
% sys = uy2ssvd(u,y,i,n,tol)
%
% U,Y - input (TxM matrix) and output (TxP matrix)
% I   - finite time balancing parameter
%    I > NMAX, an upper bound of the system order
% N   - optional system order
% TOL - tolerance for order selection, default 1e-7
% If TOL<0, the SVs are plotted and an input is asked.
% SYS - identified system in a ss format
%       in approximately finite time I balanced basis

function sys = uy2ssvd(u,y,i,n,tol)

% Constants
[T,m] = size(u);
p = size(y,2);
j = T - 2*i + 1;

% Hankel matrix of the data
Y  = blkhank(y,2*i,j);
Yp = Y(1:p*i,:);
Yf = Y(p*i+1:2*p*i,:);
U  = blkhank(u,2*i,j);
Up = U(1:m*i,:);
Uf = U(m*i+1:2*m*i,:);

% This is the choice of W2 from the book
% Lp       = chol(Up*Up')';
% V = Up'/(Up*Up')*inv(Lp)*Up; % = W2
% We use an alternative one

% Step 1: Compute sequential free responses
V       = Up'/(Up*Up');
barYf   = obl_proj(Yf,Uf,[Up;Yp]);

% Step 2: SVD of the Hankel matrix of Markov param.
[U_,S,V] = svd(barYf*V,0);
s        = diag(S);

% Determine the order
if (nargin < 5 | isempty(tol))
  tol = [];
end
if (nargin < 4 | isempty(n))
  n = order(s,tol);
end

% Step 3: compute balanced state sequence
sqrt_s  = sqrt(1./s(1:n));
T       = sqrt_s(:,ones(1,size(U_,1))) .* U_(:,1:n)';
X       = T * barYf;

% Step 4: compute A,B,C,D
M       = [X(:,2:end-1); y(i+1:i+j-2,:)'] / ...
```

```
         [X(:,1:end-2); u(i+1:i+j-2,:)'];

a = M(1:n,1:n);
b = M(1:n,n+1:end);
c = M(n+1:end,1:n);
d = M(n+1:end,n+1:end);

sys = ss(a,b,c,d,-1);

% Computation of the oblique projection O = A /_B C
function O = obl_proj(A,B,C)

r = size(C,1);

temp = pinv([C; B] * [C; B]');
O = A * [C' B'] * temp(:,1:r) * C;
```

uy2y0.m

```
% UY2Y0 - From data to sequential free responses.
%
% [y0, delta] = uy2y0(u, y, lmax, l, delta, j)
%
% U,Y - input (TxM matrix) and output (TxP matrix)
% LMAX  - upper bound for the system lag
% L     - optional number of samples in a block
% DELTA - desired length of the impulse response
%   If skipped the response the stop condition is
%   |y0(delta)| > EPS.
% Y0 - matrix of J, DELTA samples long sequential
%   free resp. the i-th response is Y(:,i)

function [y0,delta] = uy2y0(u,y,lmax,l,delta,j)

% Constants
EPS  = 1e-5; % tolerance for the convergence of
             % the response
IMAX = 100;  % maximum response length
[T,m] = size(u);
p = size(y,2);

% Check inputs
if (nargin < 3)
  error('Not enough input arguments');
end
if (nargin < 4 | isempty(l))
  l = lmax;
end
L = lmax + l; % # of rows in the Hankel matrix
if (nargin < 5 | isempty(delta))
  find_delta = 1;
  delta = IMAX;
else
  find_delta = 0;
end
if (nargin < 6 | isempty(j))
  j = T - L + 1;
end

% SVD of the Hankel matrix of the data
r   = triu(qr([blkhank(u,L);blkhank(y,L)]'))';
in  = 1:L*m+lmax*p;
r11 = r(in,in);
r21 = r(in(end)+1:end,in);
pinvUY1 = pinv(r11);
Y2      = r21;

% Initialization
fu = zeros(L*m,j);
fu(1:lmax*m,:) = blkhank(u,lmax,j);
fy = zeros(L*p,j);
fy(1:lmax*p,:) = blkhank(y,lmax,j);

% Iteration
maxi = ceil(delta/l);
y0   = zeros(delta*p,j);
for i = 1:maxi
  % Solve the system
  g = pinvUY1 * [fu;fy(1:lmax*p,:)];
  fy(lmax*p+1:end,:) = Y2 * g;
  % Store the result
  y0((i-1)*l*p+1:i*l*p,:) = fy(lmax*p+1:end,:);
  % Check exit condition (if delta is not given)
  if (find_delta & norm(fy(lmax*p+1:end,1)) < EPS)
    delta = i;
    break;
  end
  % Shift for the next iteration
  fu(1:m*l,:) = [];
  fu = [fu; zeros(m*l,j)];
  fy(1:p*l,:) = [];
  fy = [fy; zeros(p*l,j)];
end

y0 = y0(1:delta*p,:);
```

uyo2ss.m

```
% UYO2SS - From data and an extended observability
%          matrixto an I/S/O representation.
%
% sys = uyo2ss(u,y,o)
%
% U,Y - input (TxM matrix) and output (TxP matrix)
% O   - extended observability matrix
% SYS - identified system in a ss format

function sys = uyo2ss(u,y,o)

n = size(o,2);
m = size(u,2);
p = size(y,2);
l = ceil(n/p);

% Solve the shift equation for A
a = o(1:end-p,:) \ o(p+1:end,:);
c = o(1:p,:);

% Determine B and D from the response
L = l * (m+1) + m; % number of block rows in the eqn
A = zeros(L*p,n+p*m+n*m);
A(1:p,:) = [c kron(u(1,:),eye(p)) zeros(p,n*m)];
% columns corresponding to vec(B)
in_B = n+m*p+1:n+m*p+n*m;
for i = 2:L
  in_i = (i-1)*p+1:i*p;
  tmp = zeros(p,n*m);
  for j = 1:i-1
    in_j = (j-1)*p+1:j*p;
    tmp  = tmp + kron(u(i-j,:),A(in_j,1:n));
  end
  A(in_i,:) = [A(in_i-p,1:n)*a, ...
               kron(u(i,:),eye(p)), tmp];
end
B = y(1:L,:)'; B = B(:);
M = A\B;
xini = M(1:n);
d = reshape(M(n+1:n+m*p),p,m);
b = reshape(M(n+m*p+1:end),n,m);

sys = ss(a,b,c,d,-1);
```

uyx2ss.m

```
% UYX2SS - From data and a state sequence to an
%          I/S/O representation.
%
% sys = uyx2ss(u,y,x)
%
% U,Y - input (TxM matrix) and output (TxP matrix)
% X   - state sequence
% SYS - identified system in a ss format

function sys = uyx2ss(u,y,x)

n  = size(x,1);
n1 = n + 1;
j  = size(x,2) - 1;

% Solve the system of equations for A,B,C,D
M = [x(:,2:end);   y(1:j,:)'] / ...
    [x(:,1:end-1); u(1:j,:)'];

a = M(1:n,1:n);
b = M(1:n,n+1:end);
c = M(n1:end,1:n);
d = M(n1:end,n1:end);

sys = ss(a,b,c,d,-1);
```

w2r.m

```
% W2R - From time series to kernel representation
%       of the MPUM.
%
% R = w2r(w,lmax,m,n,tol)
%
% W - time series
% LMAX  - upper bound for the system lag
% M - optional number of inputs
% N - optional system order
% TOL - tolerance for order selection, default 1e-7
% R - 3d array of parameter of a kernel represent.
%     R0 = R(:,:,1), ... ,Rlmax = R(:,:,lmax+1)

function R = w2r(w,lmax,m,n,tol)

dw  = size(w,2); % # of variables
lmax_1 = lmax+1;
T = size(w,1);
j = T - lmax;

if nargin < 5 | isempty(tol)
  tol = 1e-7; % default
end

% Left kernel of the Hankel matrix
r = triu(qr(blkhank(w,lmax_1,j)'))';
[U,S,V] = svd(r);
s = diag(S);
% Determine the order
if nargin < 4 | isempty(m) | isempty(n)
  rk = sum( s > tol);
  if exist('m') == 1 & exist('n') ~= 1
    n = rk - lmax_1*m;
  elseif exist('m') ~= 1 & exist('n') == 1
    m = (rk - n) / lmax_1;
    if ~isinteger(m)
      warning('The specified order is ignored.')
      m = floor(rk / lmax_1);
      n = rk - lmax_1 * m;
    end
  else
    m = floor(rk / lmax_1);
    n = rk - lmax_1 * m;
  end
else
  rk = lmax_1*m + n;
end
R = U(:,rk+1:end)';
g = size(R,1);

% Convert R to a 3d array
R_ = R;
R  = zeros(g,dw,lmax_1);
for i = 1:lmax_1
  R(:,:,i) = R_(:,(i-1)*dw+1:i*dw);
end
```

y02ox.m

```
% Y02OX - From a matrix of sequential free responses
%         to an extended observability matrix and a
%         state sequence.
%
% [O,X] = y02ox(y0,n,tol)
%
% Y0 - matrix of (sequential) free responses
%      (Y0(:,i) is a free response)
% LMAX - upper bound on the system lag
% N - optional order of the system
% TOL - tolerance for order selection, default 1e-7
% If TOL<0, the SVs are plotted and an input is asked.
% O - extended observability matrix
% X - state sequence

function [O,X] = y02ox(y0,n,tol)

% SVD of the matrix of (sequential) free responses
[U,S,V] = svd(y0,0);
s = diag(S);

% if not given, determine the order
if (nargin < 3 | isempty(tol))
  tol = [];
end
if (nargin < 2 | isempty(n))
  n = order(s,tol);
end

% Rank revealing factorization
sqrt_s = sqrt(1./s(1:n))';
O = sqrt_s(ones(size(U,1),1),:) .* U(:,1:n);
if nargout > 1
  X = (sqrt_s(ones(size(V,1),1),:) .* V(:,1:n))';
end
```

order.m

```
% ORDER - order delection from the singular values.
%
% n = order(s,tol)
%
% S   - vector of singular values
% TOL - tolerance for order selection, default 1e-7
% If TOL<0, the SVs are plotted and an input is asked.
% N   - order

function n = order(s,tol)

TOL = 1e-7; % default value for the tolerance

if nargin < 2 | isempty(tol)
  tol = TOL;
end

if tol > 0
  n = sum(s > tol);
else
  bar(s);
  title('singular values'), xlabel('i'), ylabel('s_i'),
  n = input('Choose the order = ');
end
```

demo.m

```
% TEST the functions in the toolbox.

% ---- Simulation parameter
n = 5; m = 2; p = 3;

l = ceil(n/p);

lmax  = l;
nmax  = n;
delta = nmax+1;

s = 0;   % additive noise standard deviation
T = 500; % time horizon
L = 50;  % # of samples to be computed from
         % the impulse response

% ---- Simulate impulse response
sys0 = drss_(n,p,m);
h0   = impulse(sys0,L);
if (m == 1)
  if (p > 1)
    % transform the TxM H0 in PxMxT format
    h0s = h0; h0 = zeros(p,m,L);
    for t = 1:L
      h0(:,:,t) = h0s(t,:)';
    end
    clear h0s
  end
else
  h0  = shiftdim(h0,1);
end
ht   = randn(size(h0));
h    = h0 + s * ht;

% ---- Simulate input/output data
u0   = rand(T,m);
x0   = rand(n,1);
y0   = lsim(sys0,u0,[],x0);
ut   = randn(T,m);
yt   = randn(T,p);
u    = u0 + s * ut;
y    = y0 + s * yt;
w    = [u y];
```

```
% ---- Apply various (combinations of) functions

% UY2H
hh = uy2h(u,y,lmax,delta,L);
fprintf('%24s: error    = %f\n', 'uy2h', ...
        norm(h0(:)-hh(:)))

% UY2HBLK
hhblk = uy2hblk(u,y,lmax,L);
fprintf('%24s: error    = %f\n', 'uy2hblk', ...
        norm(h0(:)-hhblk(:)))

% UY2HY0
[hh,y0h] = uy2hy0(u,y,lmax,delta,L);
y0h_     = uy2y0(u,y,lmax,delta,L);
fprintf('%24s: error_h  = %f\n', 'uy2hy0', ...
        norm(h0(:)-hh(:)))
fprintf('%24s: error_y0 = %f\n', 'uy2hy0', ...
        norm(y0h(:)-y0h(:)))

% H2SS
sysh = h2ss(h);
fprintf('%24s: error    = %f\n', 'h2ss', ...
        norm(sys0-sysh,'inf'))

% UY2SSMR
sysh = uy2ssmr(u,y,lmax+1,n);
fprintf('%24s: error    = %f\n', 'uy2ssmr', ...
        norm(sys0-sysh,'inf'))

% UY2SSVD
sysh = uy2ssvd(u,y,lmax+1);
fprintf('%24s: error    = %f\n', 'uy2ssvd', ...
        norm(sys0-sysh,'inf'))

% UY2H -> H2SS
sysh  = uy2h2ss(u,y,lmax,delta);
fprintf('%24s: error    = %f\n', 'uy2h2ss', ...
        norm(sys0-sysh,'inf'))

% UY2Y0 -> Y02OX -> UYO2SS
sysh  = uy2o2ss(u,y,lmax,delta);
fprintf('%24s: error    = %f\n', 'uy2o2ss', ...
        norm(sys0-sysh,'inf'))

% UY2Y0 -> Y02OX -> UYX2SS
sysh  = uy2x2ss(u,y,lmax,delta);
fprintf('%24s: error    = %f\n', 'uy2x2ss', ...
        norm(sys0-sysh,'inf'))

% UY2H -> H2OX -> UYO2SS
sysh  = uy2h2o2ss(u,y,lmax,delta);
fprintf('%24s: error    = %f\n', 'uy2h2o2ss', ...
        norm(sys0-sysh,'inf'))

% UY2HY0 -> HY02OXBAL -> UYX2SS
sysh  = uy2ssbal(u,y,lmax,delta);
fprintf('%24s: error    = %f\n', 'uy2ssbal', ...
        norm(sys0-sysh,'inf'))

return

% W2R -> R2PQ -> PQ2SS
sysh  = w2r2ss([u,y],lmax);
fprintf('%24s: error    = %f\n', 'w2r2ss', ...
        norm(sys0-sysh,'inf'));
```