MATLAB code reproducing the results in "Data-driven simulation of nonlinear systems via linear time-invariant embedding"

Ivan Markovsky

Polynomial time-invariant systems

Implementation of the map $\mathbf{x} : w \to x$

```
function x = form_x(w, ell)
xext = blkhank(w, ell + 1); x = xext(1:end-1, :);
```

Convert the degrees specification `N` to the vector of monomials ϕ

```
function phi = n2phi(N)
[nth, nx] = size(N); str = '';
for i = 1:nth
    str_i = ['x(1, :) .^ ' int2str(N(i, 1))];
    for j = 2:nx
        str_i = [str_i ' .* x(' int2str(j) ', :) .^ ' int2str(N(i, j))];
    end
    if i == 1, str = str_i; else, str = [str '; ' str_i]; end
end
eval(sprintf('phi = @(x) [%s];', str));
```

The lag ℓ of the difference equation can be inferred also from the degrees specification: $\ell = (n_x + 1)/2 + 1$.

```
ell = (size(N, 2) + 1) / 2 - 1; %% <N-to-ell>
```

The model is a structure with fields: the vector of monomials ϕ, the parameter vector θ, and the the degrees specification matrix `N`.

```
%% <define-sys>
phi = n2phi(N);
<<N-to-ell>>
sys.phi = phi; sys.th = th; sys.ell = ell; sys.N = N;
```

The degrees matrix for all monomials with a bound $\mathtt{n}_{\max}$ on the degree is constructed by the function `monomials`.

```
function N = monomials(nx, n_max);
if n_max == 0, N = zeros(1, nx); return, end
if n_max == 1, N = eye(nx); return, end
if nx == 1, N = (0:n_max)'; return, end
N = [];
for i = 0:n_max
    t = monomials(nx - 1, n_max);
    ni = [i * ones(size(t, 1), 1), t];
    N = [N; ni];
end
N(sum(n, 2) > n_max, :) = [];
```

Examples

Hammerstein

```
%% <example-h>
%     u(t)  y(t)  u(t+1)
ell = 1; th  = [0.95 1 2]';
N = [0     1     0 ;
     0     0     1 ;
     0     0     2 ];
<<define-sys>>
```

Finite-lag Volterra

```
%% <example-v>
%     u(t)  y(t)  u(t+1)  y(t+1)  u(t+2)
ell = 2;
Nnl = [0     0     0       0       2      ;
       0     0     1       0       1      ;
       1     0     1       0       1      ];
sysl = drss(ell); [q p] = tfdata(tf(sysl), 'v');
thl  = vec(fliplr([q; -p]));
th   = [thl(1:end-1); 0.1 * ones(size(Nnl, 1), 1)];
N = [eye(2 * ell + 1); Nnl];
<<define-sys>>
```

Generalized bilinear

```
%% <example-b>
%     u(t)  y(t)  u(t+1)  y(t+1)  u(t+2)
ell = 2;
Nnl = [0     1     0       0       2      ;
       0     0     1       1       1      ;
       1     0     1       0       1      ];
sysl = drss(ell); [q p] = tfdata(tf(sysl), 'v');
thl  = vec(fliplr([q; -p]));
th   = [thl(1:end-1); 0.1 * ones(size(Nnl, 1), 1)];
N = [eye(2 * ell + 1); Nnl];
<<define-sys>>
```

Simulation of a polynomial time-invariant model

The special structure of the difference equation $y = \theta^\top \phi(x)$, allows us to compute the response of a system to a given input and initial condition w_{ini} by recursive evaluation forward in time.

```
function y = sim_pti(sys, u, wini)
T = size(u, 1); ell = size(wini, 1);
w = [wini; zeros(T, 2)]; w(ell + 1:end, 1) = u;
for t = 1:T
  w(t + ell, 2) = sys.th' * sys.phi([vec(w(t:t + ell -1, :)'); w(t + ell, 1)]);
end
y = w(ell + 1:end, 2);
```

```
function ans = is_trajectory(w, sys)
ans = norm(sys.th' * sys.phi(blkhank(w, sys.ell + 1)) - w(sys.ell+1:end, 2)');
```

Data-driven simulation algorithm

Model based

```
function [ys, sysh] = ddsim_m(wd, us, N, wini)
<<N-to-ell>> L = length(us); phi = n2phi(N);
if ~exist('wini'), wini = zeros(ell, 2); end
thh_ext = lra([phi(blkhank(wd, ell + 1)); wd(ell+1:end, 2)']])';
sysh.th = - thh_ext(1:end-1) / thh_ext(end);
sysh.phi = phi; sysh.ell = ell; sysh.N = N;
ys = sim_pti(sysh, us, wini);

function [R, P, dh] = lra(d, r)
if nargin == 1, r = size(d, 1) - 1; end
[u, s, v] = svd(d); R = u(:, (r + 1):end)'; P = u(:, 1:r);
if nargout > 2, dh = u(:, 1:r) * s(1:r, 1:r) * v(:, 1:r)'; end
```

Bilinear time-invariant

```
function ys = ddsim_b(wd, us, N, wini, pp)
<<N-to-ell>> L = length(us); T = size(wd, 1);
if ~exist('wini'), wini = zeros(ell, 2); end

%% seperate linear and nonlinear terms
N_l = N(sum(N, 2) == 1, :); phi_l = n2phi(N_l);
N_n = N(sum(N, 2) ~= 1, :); phi_n = n2phi(N_n);
mn = size(N_n, 1); q = mn + 2;

%% construct the extended data matrix
xd = form_x(wd, ell); wdext = [wd(1:T-ell, :) phi_n(xd)'];
Swd = blkhank(wdext, ell + L);
if exist('pp') && pp == 1
  [~, Swd] = lra(Swd, (mn+1) * (ell+L) + ell);
end
Ud = Swd(1:q:end, :); Yd = Swd(2:q:end, :);
Nd = Swd; Nd([1:q:end 2:q:end], :) = [];

%% construction of the Phi(u) matrix
I = eye(ell+L); u = [wini(:, 1); us];
phi0 = vec(phi_n(form_x([u zeros(ell+L, 1)], ell)));
for i = 1:ell+L
  Phi(:, i) = vec(phi_n(form_x([u I(:, i)], ell))) - phi0;
end

%% solve the system of equations
A = [Ud; Yd(1:ell, :);
     Nd(1:end-mn*ell, :) - Phi(:, ell+1:end) * Yd(ell+1:end, :)];
b = [u; wini(:, 2); phi0 + Phi(:, 1:ell) * wini(:, 2)];
ys = Yd(ell+1:end, :) * pinv(A) * b;
```

Simulation examples

Illustrative examples

```
%% setup
clear all, s = 0; T = 100; L = 10;

% Hammerstein
<<example-h>>
<<simulate-data>>
wsini = rand(ell, 2); us = rand(L, 1); ys = sim_pti(sys, us, wsini); name = 'h';
<<plot-results>>

% Volttera
<<example-v>>
<<simulate-data>>
wsini = ones(ell, 2); us = zeros(L, 1); ys = sim_pti(sys, us, wsini); name = 'v';
<<plot-results>>

% Bilinear
<<example-b>>
<<simulate-data>>
wsini = zeros(ell, 2); us = ones(L, 1); ys = sim_pti(sys, us, wsini); name = 'b';
<<plot-results>>

% <simulate-data>
ud0 = rand(T, 1); wdini0 = zeros(ell, 2);
yd0 = sim_pti(sys, ud0, wdini0); wd0 = [ud0, yd0];

% <plot-results>
ysh = ddsim_b(wd0, us, N, wsini); check = norm(ys - ysh)
figure(1)
plot([wsini(:, 1); us]), ax = axis; axis([1 ell + L ax(3:4)]), box off
xlabel('$t$', 'interpreter', 'latex'), ylabel('$u$', 'interpreter', 'latex')
print_fig([name '-u'])
figure(2)
plot([wsini(:, 2); ys], 'r-'), hold on, plot([wsini(:, 2); ysh], 'b--')
ax = axis; axis([1 ell + L ax(3:4)]), box off, hold off
xlabel('$t$', 'interpreter', 'latex'), ylabel('$y$', 'interpreter', 'latex')
print_fig([name '-y'])
```

Noisy data in the errors-in-variables setup

```
%% setup
clear all, close all
T = 1000; L = 10; NN = 100; np = 10; S = linspace(0, 0.005, np);
<<example-b>>
wsini = rand(ell, 2); us = rand(L, 1); ys = sim_pti(sys, us, wsini);
e = @(ysh) 100 * norm(ys - ysh) / norm(ys);
<<simulate-data>>
for j = 1:np, s = S(j);
  for i = 1:NN
    wn  = randn(T, 2); wd = wd0 + s * norm(wd0) * wn / norm(wn);
```

```
    ed(i, j) = e(ddsim_b(wd, us, N, wsini));
    em(i, j) = e(ddsim_m(wd, us, N, wsini));
  end
end
plot(S, mean(ed), 'b--'), hold on, plot(S, mean(em), 'r-')
ax = axis; axis([S(1) S(end) ax(3:4)]), box off
xlabel('noise to signal ratio', 'interpreter', 'latex')
ylabel('$e$, \%', 'interpreter', 'latex'), print_fig('error')
```

Real data from DAISY

```
clear all, close all
switch 1 % choose and example
  case 1, dryer,          ell = 5; Ti = 1:800;  Tv = 801:1000;
  case 2, ballbeam,       ell = 2; Ti = 1:800;  Tv = 801:1000;
  case 3, flutter,        ell = 5; Ti = 1:800;  Tv = 801:1024;
  case 4, robot_arm,      ell = 4; Ti = 1:800;  Tv = 801:1024;
  case 5, heating_system, ell = 2; Ti = 1:600;  Tv = 601:801;
  case 6, exchanger,      ell = 2; Ti = 1:3200; Tv = 3201:4000;
end
N = [eye(2 * ell + 1); zeros(1, 2 * ell + 1)];
wd = [u(Ti) y(Ti)];
w  = [u(Tv) y(Tv)];
wini = w(1:ell, :);
us = w(ell+1:end, 1);
ys = w(ell+1:end, 2);
e = @(ysh) 100 * norm(ys - ysh) / norm(ys);
[ysh_m, sysh] = ddsim_m(wd, us, N, wini);
ysh_d = ddsim_b(wd, us, N, wini);
plot(ys, 'k'), hold on, plot(ysh_m, 'r-.'), plot(ysh_d, 'b--')
[e(ysh_m) e(ysh_d)]
```