Code for the paper
"How to represent and identify affine time-invariant systems?"

Ivan Markovsky

The code is based on the Behavioral Toolbox [1], available from

`https://imarkovs.github.io/bt/`

The code listed below is available from

`https://imarkovs.github.io/software/ati-sysid.tar`

The function `ati_ident` for approximate identification uses the SLRA package [3], available from

`https://slra.github.io/software-slra.html`

1 Implementation

1.1 Checking if $w \in \mathcal{B}|_T$

```
function ans = ati_w_in_B(w, Rlin, Rc, tol);
if ~exist('tol'), tol = 1e-10; end
[T, q] = size(w); ell = size(Rlin, 2) / q - 1;
ans = norm(multmat(Rlin, q, T) * vec(w') - kron(ones(T - ell, 1), -Rc)) < tol;
```

1.2 Random trajectory simulation

```
function w = ati_R2w(R, Rc, q, T);
ell = size(R, 2) / q - 1;
M = multmat(R, q, T); N = null(M); z = rand(size(N, 2), 1);
w =  pinv(M) * kron(ones(T - ell, 1), -Rc) + N * z;
w = reshape(w, q, T)';
```

1.3 Min-norm and constant offset

```
function wc = min_norm_wc(Rlin, Rc, q, T);
ell = size(Rlin, 2) / q - 1;
wc = pinv(multmat(Rlin, q, T)) * kron(ones(T - ell, 1), -Rc);
wc = reshape(wc, q, T)';
```

```
function [wc, bwc] = const_wc(Rlin, Rc, q, T);
ell = size(Rlin, 2) / q - 1;
bwc = pinv(multmat(Rlin, q, T) * kron(ones(T, 1), eye(q))) * kron(ones(T - ell, 1), -Rc);
wc = kron(ones(T, 1), bwc');
```

1.4 Exact identification

```
function [Rlin, Rc] = ati_w2R(wd, c, opt)
[Td, q] = size(wd); L = c(2) + 1; r = c(1) * L + c(3) + 1;
if ~exist('opt')
  [~, R] = lra([ones(1, Td-c(2)); hank(wd, L)], r);
else
  [~, R] = glra2([ones(1, Td-c(2)); hank(wd, L)]', 1, r); R = R';
end
Rc = R(:, 1); Rlin = R(:, 2:end);
```

1.5 Approximate identification

```
function [Rlin, Rc, wh, info] = ati_ident(wd, m, ell, opt)
[Td, q] = size(wd); L = ell + 1;
if ~exist('opt'); opt = []; end

ss.m = [1; L * ones(q, 1)];
ss.w = [inf * ones(Td-ell, 1); ones(q * Td, 1)];
[ph, info] = slra([ones(Td-ell, 1); vec(wd)], ss, q * ell + m + 1, opt);
wh = reshape(ph(Td-ell+1:end), Td, q);

R_ = info.Rh(:, 2:end); Rlin = []; for i = 1:L, Rlin = [Rlin R_(:, i:L:end)]; end
Rc = info.Rh(:, 1);
```

1.6 ATI interpolation and approximation

Model-based

```
function wh = ati_mbint(Blin, wc, w)
[T, q] = size(w);
wh = ddint(Blin, w - wc) + wc;
```

Data-driven version 1: adding the equation $\mathbf{1}^\top g = 1$

```
function wh = ati_ddint(wd, w)
[T, q] = size(w);
H = hank(wd, T); vec_w = vec(w'); I = find(~isnan(vec_w));
A = [H(I, :); ones(1, size(H, 2))]; b = [vec_w(I); 1]; g = pinv(A) * b;
% lasso_cvx(A, b, 1); % lseq(H(I, :), vec_w(I), ones(1, size(H, 2)), 1);
wh = reshape(H * g, q, T)';
```

Data-driven version 2: averaging the columns of the Hankel matrix

```
function wh = ati_ddint2(wd, w)
[T, q] = size(w);
H = hank(wd, T); vec_w = vec(w'); I = find(~isnan(vec_w));
wch = mean(H, 2); H = H - wch(:, ones(1, size(H, 2))); vec_w = vec_w - wch;
A = H(I, :); b = vec_w(I); g = pinv(A) * b;
wh = reshape(H * g + wch, q, T)';
```

2 Simulation examples

2.1 Exact

Consider the 4th order single-input single-output ATI system $\mathcal{B}$ defined by a difference equation representation with parameters

$$R_{\text{lin}}(z) = \begin{bmatrix} 0.5 & -0.9 \end{bmatrix} z^0 + \begin{bmatrix} 0.3 & 1.3 \end{bmatrix} z^1 + \begin{bmatrix} 0 & -1.6 \end{bmatrix} z^2 + \begin{bmatrix} 0 & 1.4 \end{bmatrix} z^3 + \begin{bmatrix} 0 & -1 \end{bmatrix} z^4 \quad \text{and} \quad R_{\text{c}} = 1.$$

```
Rlin = [0.5 -0.9 0.3 1.3 0 -1.6 0 1.4 0 -1]; Rc = 1;
```

It has a constant offset $\overline{w}_{\text{c}} = \left[\begin{smallmatrix} -0.625 \\ 0.625 \end{smallmatrix}\right]$, which can be found by solving the system of linear equations

$$\mathcal{M}_T(R_{\text{lin}})(\mathbf{1}_{T-\ell} \otimes I_q)\overline{w}_{\text{c}} = \mathbf{1}_{T-\ell} \otimes R_{\text{c}},$$

where $\mathcal{M}_T$ is the multiplication matrix [2]

$$\mathcal{M}_T(R) := \begin{bmatrix} R_0 & R_1 & \cdots & R_\ell & & & \\ & R_0 & R_1 & \cdots & R_\ell & & \\ & & \ddots & \ddots & & \ddots & \\ & & & R_0 & R_1 & \cdots & R_\ell \end{bmatrix} \in \mathbb{R}^{g(T-\ell) \times qT}.$$

The minimum-norm trajectory with length T

$$w_c^* := \arg \min_{w_c \in \mathcal{B}|_T} \|w_c\|$$

is computed by solving the linear least-squares problem

$$\text{minimize} \quad \text{over } w \in (\mathbb{R}^q)^T \quad \|\mathcal{M}_T(R_{\text{lin}})w - \mathbf{1}_{T-\ell} \otimes R_c\|.$$

The minimum-norm trajectory w_c^* with length $T = 100$ and the constant offset trajectory are plotted in Figure 1.

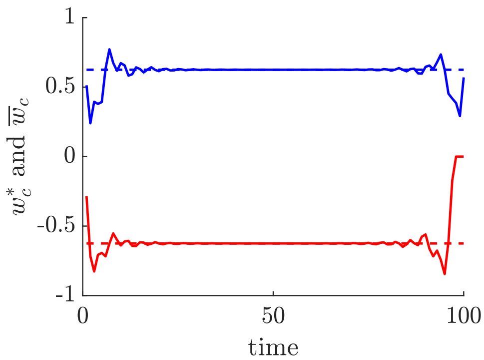

Figure 1: Minimum-norm (solid line) and constant (dashed line) offsets with length $T = 100$.

The minimum-norm trajectory is orthogonal to $\mathcal{B}_{\text{lin}}|_T$. Using the Behavioral Toolbox [1], this fact is verified as follows:

```
norm(R2BT(Rlin, q, T)' ...
     * vec(wc_min_norm')) % -> 0
```

It does not satisfy the generalized persistency of excitation condition for identifiability.

```
rank(hank(wc_min_norm, 5)) % -> 9
```

A randomly selected trajectory $w_d \in \mathcal{B}|_{100}$, however, satisfies the generalized persistency of excitation condition for identifiability.

```
rank(hank(wd, 5)) % -> 10
```

Using w_d, Algorithm 1 recovers the data-generating system $\mathcal{B}$. In contrast, the two-step approach—center the data and identify $\mathcal{B}_{\text{lin}}$ from the centered data—does not recover the true system. There is an exact model for the data w_d in the model class $\mathcal{L}^2_{(1,5,5)}$ that we obtain with the function `w2R` from the Behavioral Toolbox

```
Raug = w2R(wd, [m, n+1 n+1]);
```

Next, we verify that the augmented system $\mathcal{B}_{\text{aug}}$ has the eigenvalues of $\mathcal{B}_{\text{lin}}$ and an extra one at 1:

```
>> roots(flip(Rlin(2:2:end)))

ans =

  -0.0937 + 0.9613i
  -0.0937 - 0.9613i
   0.7937 + 0.5786i
   0.7937 - 0.5786i

>> roots(flip(Raug(2:2:end)))

ans =
```

```
  -0.0937 + 0.9613i
  -0.0937 - 0.9613i
   0.7937 + 0.5786i
   0.7937 - 0.5786i
   1.0000 + 0.0000i

Rlin = [0.5 -0.9 0.3 1.3 0 -1.6 0 1.4 0 -1]; ; Rc = 1;
m = 1; q = 2; n = 4; c = [m n n]; T = 100;

wc_min_norm = min_norm_wc(Rlin, Rc, q, T); ati_w_in_B(wc_min_norm, Rlin, Rc, q)
wc_const = const_wc(Rlin, Rc, q, T);       ati_w_in_B(wc_const, Rlin, Rc, q)

figure(1), hold on
plot(wc_min_norm(:, 1), 'r'), hold on
plot(wc_min_norm(:, 2), 'b')
plot(wc_const(:, 1), 'r--')
plot(wc_const(:, 2), 'b--')
% legend()
xlabel('time'), ylabel('$w_c^*$ and $\overline{w}_c$')
box off, print_fig('wc')

norm(R2BT(Rlin, q, T)' * vec(wc_min_norm')) % -> 0
rank(hank(wc_min_norm, n+1))

wd = ati_R2w(Rlin, Rc, q, T);
rank(hank(wd, n+1))

[Rlin_, Rc_] = ati_w2R(wd, c);
Raug = w2R(wd, [m, n+1 n+1]);

roots(flip(Raug(2:2:end)))
roots(flip(Rlin(2:2:end)))

Blin_ = n4sid(detrend(iddata(wd(:, 2), wd(:, 1))), n, ...
              'DisturbanceModel', 'none', 'Feedthrough', 1);
```

2.2 Approximate

```
clear all, close all

%% data-generating system
M = 1; d = 0.1; k = 1;
B = c2d(ss(tf(1, [M d k])), 0.1);
B = ss(B.a, [], B.c, [], -1);

%% simulation parameters
[p, m] = size(B); n = order(B);
Td = 100; T = 10; nm = T - n;
N = 100; np = 5; S = linspace(0, 0.001, np);

%% data-generating system
% B  = drss(n, p, m);
q = m + p; R = B2R(B); Rc = 1 * ones(size(R, 1), 1);
ell = lag(B); c = [m, ell, n];

%% simulated data
wd0 = ati_R2w(R, Rc, q, Td); % plot(wd0)
w0  = ati_R2w(R, Rc, q, T);
```

```matlab
w = w0;
% Im = randperm(q * T); Im = Im(1:nm);
Im = T-nm+1:T;
w(Im) = NaN;

%% validation critera
eBh = @(Rh) 100 * norm([Rc R] - Rh) / norm([Rc R]);
ewh = @(wh) 100 * norm(w0(Im) - wh(Im)) / norm(w0(Im));

for j = 1:np
  s = S(j)
  for i = 1:N, i = i
    wn = randn(size(wd0));
    wd = wd0 + s * norm(wd0) * wn / norm(wn);

    %% ATI direct data-driven method
    wh1  = ati_ddint(wd, w); % wh1_ = ati_ddint2(wd, w);

    %% Method based on difference equation representation
    [Rh3, Rch3] = ati_w2R(wd, c); R3 = [Rch3 Rh3]; R3 = R3 / R3(1);
    Bh3 = R2ss(Rh3, q, 1:q, c);
    wch3 = min_norm_wc(Rh3, Rch3, q, T);
    wh3  = ati_mbint(Bh3, wch3, w);

    [Rh3_, Rch3_] = ati_w2R(wd, c, 1); R3_ = [Rch3_ Rh3_]; R3_ = R3_ / R3_(1);
    Bh3_ = R2ss(Rh3_, q, 1:q, c);
    wch3_ = min_norm_wc(Rh3_, Rch3_, q, T);
    wh3_ = ati_mbint(Bh3_, wch3_, w);

    %% centering + LTI identification
    wch4 = mean(wd);
    wd_ = wd - wch4(ones(Td, 1), :);
    Bh4 = ss(n4sid(iddata(wd_(:, m+1:end), wd_(:, 1:m)), n, 'Feedthrough', ones(1, m)));
    wh4  = ati_mbint(Bh4, wch4(ones(T, 1), :), w);
    R4 = [wch4 B2R(Bh4)]; R4 = R4 / R4(1);

    %% ML method based on SLRA
    opt.opt.Rini = [Rh3_, Rch3_];
    [Rh5, Rch5, wh, info] = ati_ident(wd, m, ell, opt); R5 = [Rch5 Rh5]; R5 = R5 / R5(1);
    Bh5 = R2ss(Rh5, q, 1:q, c);
    wch5 = min_norm_wc(Rh5, Rch5, q, T);
    wh5  = ati_mbint(Bh5, wch5, w);

    EB(i, j, :) = [NaN,      eBh(R3), eBh(R3_), eBh(R5), eBh(R4)];
    EW(i, j, :) = [ewh(wh1), ewh(wh3), ewh(wh3_), ewh(wh5), ewh(wh4)];
  end
end

mns = {'std' 'DDR', 'LRA', 'GLRA', 'SLRA', '2-step'};
mEB = squeeze(mean(EB));
mEW = squeeze(mean(EW));

[mns; num2cell([S' mEB])]
[mns; num2cell([S' mEW])]

% return

%% results
```

```
ls = {'b--', 'r-.', 'c-.', 'g-', 'k:'};

figure(1), hold on
for i = 2:5, plot(S, mEB(:, i), ls{i}), end
xlabel('noise level'), ylabel('$e_R$, \%')
legend(mns{3:end}, 'Location', 'northeast')
legend('boxoff')
ax = axis; axis([S(1) S(end) 0 ax(4)]),
print_fig('eb')

figure(2), hold on
for i = 1:5, plot(S, mEW(:, i), ls{i}), end
xlabel('noise level'), ylabel('$e_y$, \%')
legend(mns{2:end}, 'Location', 'northeast')
ax = axis; axis([S(1) S(end) 0 ax(4)]),
legend('boxoff')
print_fig('ew')

Bh = ss(n4sid(iddata(wd0(:, m+1:end), wd0(:, 1:m)), n+1, 'Feedthrough', ones(1, m)));
Bh = modalreal(Bh)
```

References

[1] I. Markovsky. “The Behavioral Toolbox”. In: *Proceedings of Machine Learning Research*. Vol. 242. 2024, pp. 130–141.

[2] I. Markovsky and F. Dörfler. “Identifiability in the behavioral setting”. In: *IEEE Trans. Automat. Contr.* 68 (3 2023), pp. 1667–1677.

[3] I. Markovsky and K. Usevich. “Software for weighted structured low-rank approximation”. In: *J. Comput. Appl. Math.* 256 (2014), pp. 278–292.