MATLAB code reproducing the simulation results in "Data-driven dynamic interpolation and approximation"

Algorithm

The main function is `ddint`. It implements the basic algorithm for data-driven interpolation as well as its generalizations. The input arguments are:

- `wd` — trajectory of the system (the longer dimension indicates samples, the shorter dimension indicates variables) or a linear time-invariant model,
- `w` — partially specified trajectory of the system (the longer dimension indicates samples, the shorter variables), with `NaN`'s indicating the missing values,
- `S` — (optional) element-wise weight matrix defining an approximation criterion `norm(S .* (w - wh))`,
- `m` — (optional) number of inputs of the data-generating system system,
- `n` — (optional) order of the data-generating system system, and
- `l` — (optional) 1-norm regularization parameter.

The output arguments are:

- `wh` — particular completion of `w`,
- `N` — basis for all possible completions `wh` of `w`, and
- `g` — the particular solution g of the system $\mathscr{H}_L(w_{\mathrm{d}})|g = w$.

The optional input arguments `m` and `n` are used to check assumption A1 and to do preprocessing of `wd` by low-rank approximation. If `m` and `n` are not specified, the raw data `wd` is used without preprocessing. If `wd` is an linear time-invariant model, then the interpolation is model-based using the specified model instead of data.

If an exact completion of `w` does not exist, an optimal approximation is computed with the criterion

`norm(S(Ig) .* (w(Ig) - wh(Ig))) + l * norm(g, 1)`, where `Ig = ~isnan(w)`.

The default value for `S` is `ones(size(w))` (uniform weights). Large values of `S` enforce interpolation points. (Specification of exact interpolation points by `inf` values in `S` is currently not implemented.) The default value for `l` is `0` (no 1-norm regularization).

```
function [wh, N, g] = ddint(wd, w, S, m, n, l)
[Ts, q] = size(w);
if ~exist('S') || isempty(S), S = ones(size(w)); end
if Ts < q, w = w'; S = S'; [Ts, q] = size(w); end
w_vec = vec(w'); S_vec = vec(S');
Ig = find(~isnan(w_vec));
if isa(wd, 'lti')
  m_ = size(wd, 2); n_ = order(wd); Tmin = (m_+1) * Ts + n_;
  ud = rand(Tmin, m_);
```

```
  yd = lsim(wd, ud, [], rand(n_, 1));
  wd = [ud yd]; clear ud yd
end
Hwd = blkhank(wd, Ts);
if exist('m') && exist('n') && ~isempty(m) && ~isempty(n)
  r = Ts * m + n; tol = 1e-12;
  if rank(Hwd, tol) < r, warning('wd not informative.'); end
  [~, Hwd] = lra(Hwd, r);
end
A = S_vec(Ig, ones(1, size(Hwd, 2))) .* Hwd(Ig, :);
b = S_vec(Ig) .* w_vec(Ig);
if ~exist('l') || isempty(l) || l == 0 || isa(wd, 'lti')
  g = pinv(A) * b;
else
  g = lasso_cvx(A, b, l);
end
wh = reshape(Hwd * g, q, Ts)';
if nargout > 1, N = Hwd * null(A); end

function g = lasso_cvx(A, b, l)
cvx_begin quiet
   variable g(size(A, 2), 1)
   minimize(norm(b - A * g) + l * norm(g, 1))
cvx_end
```

Simulation setup

Data generating systems

Marginally stable autonomous system 4th order

```
%% two-sins-system
z = [exp(i * 0.24) exp(i * 0.06)];
sys0 = ss(zpk([], [z conj(z)], -1));
sys0 = ss(sys0.a, [], sys0.c, [], -1);
```

Random marginally stable autonomous system

```
%% random-system
n = ell * p; sys0 = drss_ms(n, p, m);

%% random marginally stable system generation
function sys = drss_ms(n, p, m)
if ~exist('p'), p = 1; end
if ~exist('m'), m = 0; end
w = 2 * pi * rand(floor(n / 2), 1);
if mod(n, 2) ~= 0, wr = 0; else wr = []; end
sys = ss(zpk(1, exp(i * [w; -w; wr]), 1, 1));
sys = ss(sys.a, rand(n, m), rand(p, n), rand(p, m), -1);
```

Autonomous system 6th order (model for the airpass data)

```
%% identify a model from the raw data
clear all, load airpass, T = length(y);
```

```
addpath ../detss/
n = 6; ell = n; sys0 = h2ss(y, n);
yh = impulse(sys0, T-1); norm(y - yh) / norm(y)
opt.sys0 = ss(sys0.a, [], sys0.c, [], -1);
[sys0, info, yh] = ident(y, 0, n, opt); norm(y - yh) / norm(y)
plot(y, 'r'), hold on, plot(yh, 'b--')
[M, yh, x0] = misfit(yh, sys0);
save('airpass-sys0.mat', 'sys0', 'x0', 'T')

%% airpass-system-data
%% simulate exact data from the model
load airpass-sys0
wd = initial(sys0, x0, T-1);
plot(wd, 'k-'), axis([1 144 100 610]), box off,
print_fig([name '-wd'], 15, 1)
L = 20; xs0 = [-50 180 -300 330 -190 50]';
w0 = initial(sys0, xs0, L-1);
```

Benchmark SISO system from [ddctr-benchmark]

The data generating system used in this example is the benchmark of [**ddctr-benchmark**]. It is a 4th order single-input single-output system $\mathscr{B}$ defined by the transfer function

$$H(z) = \frac{0.2826z + 0.5067z^2}{1 - 1.4183z + 1.5894z^2 - 1.3161z^3 + 0.8864z^4}.$$

```
%% benchmark-system
Q = [0 0 0 0.28261 0.50666];
P = [1 -1.41833 1.58939 -1.31608 0.88642];
sys0 = ss(tf(Q, P, -1)); n = order(sys0); ell = n;
```

Data: random trajectories `wd` and `w` of `sys0`

```
%% random-trajectories
ud = rand(T, m); wd = [ud lsim(sys0, ud, [], rand(n, 1))];
u0 = rand(L, m); w0 = [u0 lsim(sys0, u0, [], rand(n, 1))];
```

Test `ddint` and plot the results

```
%% test-interpolation
w = w0; wt = randn(L, m+p);
w = w0 + s * norm(w0) * wt / norm(wt); w(Im) = NaN;
[wh, N] = ddint(wd, w); norm(w0 - wh) / norm(w0)
[I, J] = ind2sub([L 2], Im);
for i = 1:(m+p)
  figure(i), hold on
  Imi = I(find(J == i)); Igi = setdiff(1:L, Imi);
  plot(w0(:, i),  'k:'), plot(w(:, i), 'k:')
  plot(Imi, w0(Imi, i), 'rx', 'markersize', 10)
  plot(Igi, w0(Igi, i), 'bx', 'markersize', 10)
  plot(Imi, wh(Imi, i), 'ro', 'markersize', 10)
  plot(Igi, wh(Igi, i), 'bo', 'markersize', 10)
  ax = axis; axis([1 L ax(3:4)])
  %plot(Imi, ax(3) * ones(size(Imi)), 'rx', 'markersize', 10)
```

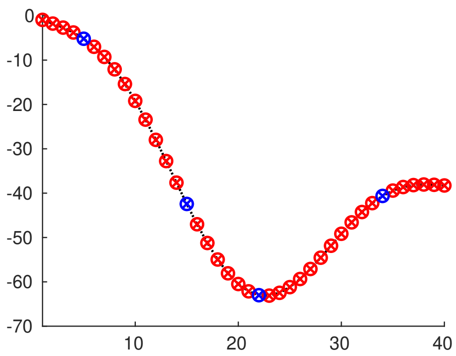
```
  %plot(Igi, ax(3) * ones(size(Igi)), 'b+', 'markersize', 10)
  % axis off,
  print_fig([name int2str(i)], 15), hold off
end
```

Simulations

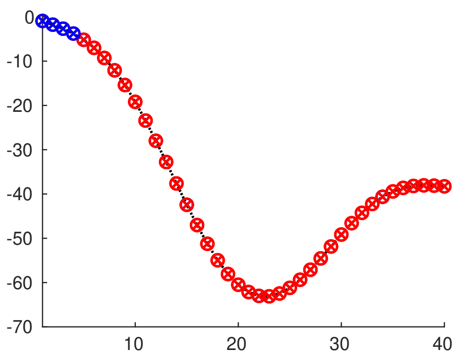
Illustrative examples

Autonomous system

```
clear all, close all, name = 'illustrative-example';
m = 0; p = 1; n = 4; s = 0; T = 100; L = 40;
<<two-sins-system>>
<<random-trajectories>>
ng = L * m + n; nt = L * (m + p); Im = randperm(nt, nt - ng);
<<test-interpolation>>
```



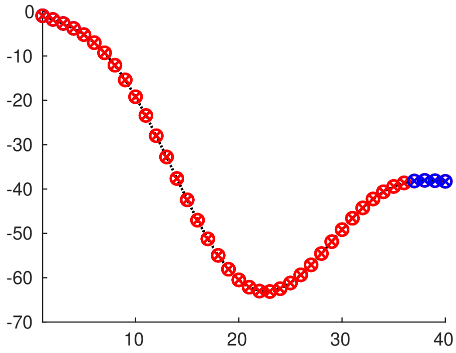

```
name = 'sim'; Im = n+1:L;
<<test-interpolation>>
```




```
name = 'sim-fc'; Im = 1:L-n;
<<test-interpolation>>
```

SISO system

The data generating system is defined in Section . The trajectory $w_{\mathrm{d}} \in \mathscr{B}|_{100}$ is a randomly selected trajectory of $\mathscr{B}$. The to-be-interpolated trajectory w is the step response with appended zero initial conditions $w \in \mathscr{B}|_{40}$ from the input



$u = w_1$ to the output $y = w_2$. The missing samples $w|_{\mathfrak{I}}$ are $w_2(5), \ldots, w_2(40)$, *i.e.*, the step response output samples. Figure **??** shows the simulated step response from the model and the interpolant computed by Algorithm **??**. As in the case of the autonomous system in Section **??**, the match is exact up to the machine precision.

```
clear all, close all, name = 'siso-sim';
m = 1; p = 1; s = 0; T = 100; L = 40;
<<benchmark-system>>
ud = rand(T, m); wd = [ud lsim(sys0, ud)];
u0 = [zeros(n, 1); ones(L-n, 1)]; w0 = [u0 lsim(sys0, u0)];
w_ = ones(L, m + p); w_(n+1:end, m+1:end) = NaN; Im = find(isnan(w_(:)));
w = w0; wt = randn(L, m+p); w = w0 + s * norm(w0) * wt / norm(wt); w(Im) = NaN;
<<test-interpolation>>
```

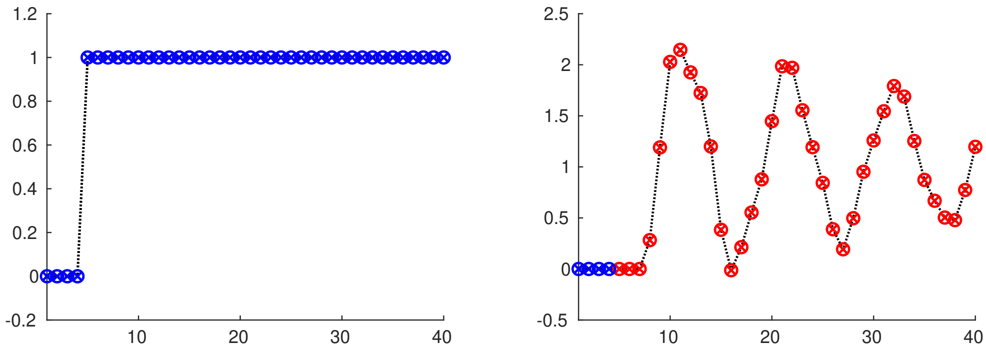


Figure 1: The step response of a single-input single-output system obtained by model-based simulation ($\times$) coincides with the data-driven interpolated trajectory by Algorithm **??** ($\circ$), where the given data is the zero initial conditions and the step input. Blue—given data samples $w|$ and their estimates $\widehat{w}|$, Red—missing samples $w|_{\mathfrak{I}}$ and their estimates $\widehat{w}|_{\mathfrak{I}}$.

MIMO

```
clear all, close all
np = 10; s = 0; mc = 20;
M  = round(linspace(1 , 100, np));
P  = round(linspace(10, 500, np));
N  = round(linspace(10, 500, np));
LL = round(linspace(50, 500, np));
TT = round(linspace(10000, 50000, np));
X = TT; name = 'T';
for j = 1:np
    for i = 1:mc
```

```
    m = M(1); p = P(1); n = N(1); L = LL(1); T = TT(j); ell = ceil(n / p);

    wm = [ones(ell, m + p); [ones(L - ell, m) NaN * ones(L - ell, p)]];
    Im = find(isnan(wm));

    sys0 = drss(n, p, m);
    ud = rand(T, m); wd = [ud lsim(sys0, ud, [], rand(n, 1))];
    u0 = rand(L, m); w0 = [u0 lsim(sys0, u0, [], rand(n, 1))];

    w = w0; wt = randn(L, m+p);
    w = w0 + s * norm(w0) * wt / norm(wt); w(Im) = NaN;

    H = blkhank(wd, L); w_vec = vec(w'); Ig = find(~isnan(w_vec));
    tic, wh = H * (H(Ig,:) \ w_vec(Ig));        t_dd(i, j) = toc;
    tic, ys0 = lsim(sys0, u0, [], rand(n, 1)); t_mb(i, j) = toc;
    e(i, j) = norm(vec(w0') - wh) / norm(w0(:));
  end
end

figure(1), plot(X, mean(t_dd), '-k'), % hold on, plot(X, t_mb, '-.r')
box off, ax = axis; axis([X(1) X(end), ax(3:4)])
xlabel(['$' name '$'],'Interpreter','latex')
ylabel('time, sec','Interpreter','latex')
print_fig(['mimo-t-' name], 20)

figure(2), plot(X, mean(e), '-k'), print_fig('mimo-e-p', 15)
box off, ax = axis; axis([X(1) X(end), ax(3:4)])
xlabel(['$' name '$'],'Interpreter','latex')
ylabel('$e$, sec','Interpreter','latex')
print_fig(['mimo-e-' name], 20)
```

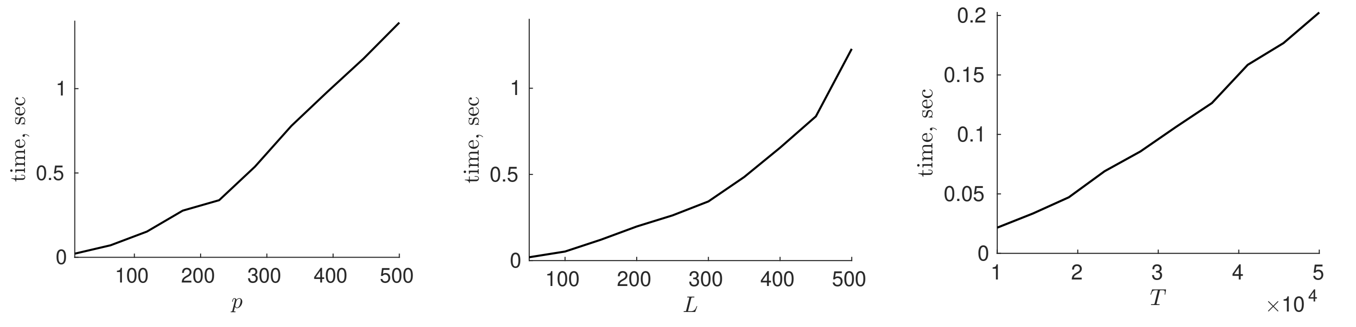


Figure 2: The computation time is quadratic in the number of outputs p (left plot) as well as in the length L of the trajectory w (middle plot), and linear in the length T of the trajectory w_d (right plot).

Nonunique solution

Set of free responses

```
clear all, close all, name = 'Y0';
m = 1; p = 1; T = 100; L = 40;
<<benchmark-system>>
ud = rand(T, m); wd = [ud lsim(sys0, ud,  [], rand(n, 1))];
```

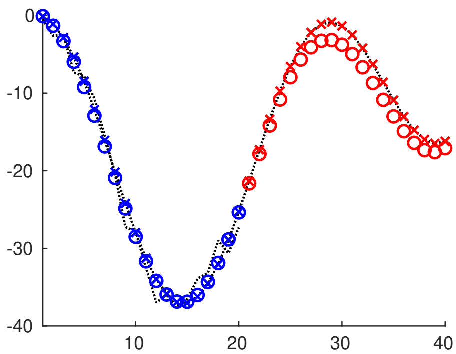
```
u0 = [zeros(n, 1); ones(L-n, 1)]; w0 = [u0 lsim(sys0, u0)];
w = zeros(L, m + p); w(:, m+1:end) = NaN;
[wh, W0] = ddint(wd, w); Y0 = W0(2:2:end, :);

O = sys0.c; for i = 2:L, O = [O; O(end, :) * sys0.a]; end
rank([O Y0]) == rank(O) % test if image O = image Y0
```

Set of zero-initial conditions responses

```
clear all, close all, name = 'B0';
m = 1; p = 1; s = 0; T = 100; L = 40;
<<benchmark-system>>
ud = rand(T, m); wd = [ud lsim(sys0, ud)];
w = zeros(L, m + p); w(n+1:end, :) = NaN;
[wh, B0] = ddint(wd, w); B0 = B0(2 * n + 1:end, :);

ut = rand(L-n, m); wt = [ut lsim(sys0, ut)]; % random zero ini. cond. response
rank([vec(wt') B0]) == rank(B0) % test if wt \in image B0
```

Smoothing with an autonomous system

```
clear all, close all, name = 'illustrative-example-smooth';
m = 0; p = 1; n = 4; s = 0.1; T = 100; L = 40;
<<two-sins-system>>
<<random-trajectories>>
Im = 21:40;
<<test-interpolation>>
```



Noisy data

Validation criterion

Relative percentage approximation error:

```
e = @(wh, I) 100 * norm(w0(I) - wh(I)) / norm(w0(I));
```

Approximation error as a function of λ

```
%% plot-lambda
np = 20; L = linspace(0, 2 * l, np);
```

```
for j = 1:np, wh = ddint(wd, w, [], [], [], L(j)); el1(j) = e(wh,Im); end
[min_e, min_i] = min(el1); l = L(min_i)
figure, hold on
plot(L, el1(1:length(L))), plot(l, min_e, 'o')
ax = axis; axis([L(1), L(end), ax(3:4)]), box off
xlabel('$\lambda$','Interpreter','latex')
ylabel('$e_{\rm missing}$','Interpreter','latex')
print_fig([datasets{i} '-l'], 20, 1)
```

Check the sparsity level

```
%% plot-g
[wh, ~, g] = ddint(wd, w, [], [], [], l);
figure, hold on
stem(sort(abs(g), 'descend'), 'linewidth', 2), hold on, ax = axis; box off
r = size(w, 1) * m + n; plot(r * ones(1,2), ax(3:4), ':', 'linewidth', 2)
axis([1 length(g) ax(3:4)])
xlabel('$i$','Interpreter','latex')
ylabel('sorted $|g_i|$','Interpreter','latex')
print_fig([datasets{i} '-g'], 20, 1)
```

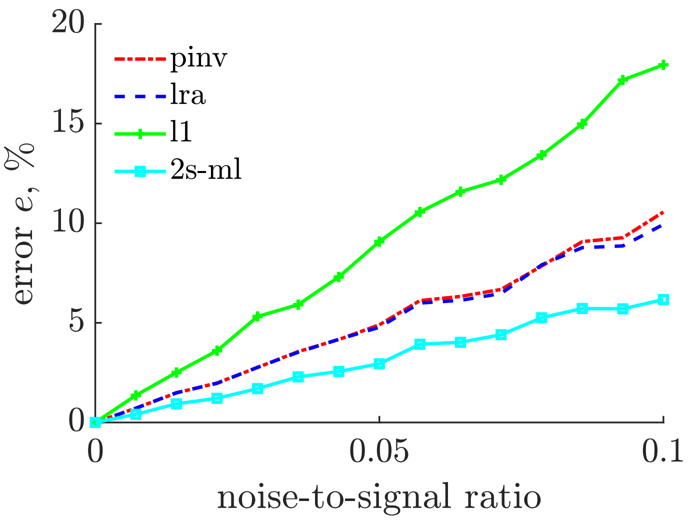
Methods

```
methods.name{1} = 'pinv';  methods.ls{1} = '-.r'; methods.comp{1} = 'wh = ddint(wd, w
methods.name{2} = 'lra';   methods.ls{2} = '--b'; methods.comp{2} = 'wh = ddint(wd, w
methods.name{3} = 'l1';    methods.ls{3} = '-+g'; methods.comp{3} = 'wh = ddint(wd, w
methods.name{4} = '2s-ml'; methods.ls{4} = '-sc'; methods.comp{4} = 'wh = ddint(ident
methods.name{5} = 'ml';    methods.ls{5} = '-k' ; methods.comp{5} = '[sysh, info, wh
methods.name{6} = 'mb';    methods.ls{6} = '-k' ; methods.comp{6} = 'wh = ddint(ss(p
```

Simulated data from a SISO system in the EIV setup

```
clear all, close all
m = 1; p = 1; s = 0; T = 100; L = 10; N = 100;
np = 15; SD = linspace(0, 0.1, np); l = 0.1; K = 1:4;
<<benchmark-system>>
<<methods>>

ud0 = rand(T, m); wd0 = [ud0 lsim(sys0, ud0)];
u0 = [zeros(n, 1); ones(L-n, 1)]; w0 = [u0 lsim(sys0, u0)];
w_ = ones(L, m + p); w_(n+1:end, m+1:end) = NaN;
Ig = find(~isnan(w_(:))); Im = find(isnan(w_(:)));
w = w0; wt = randn(L, m+p); w = w0 + s * norm(w0) * wt / norm(wt); w(Im) = NaN;
<<error>>

%wn = randn(T, m + p); wd = wd0 + SD(5) * norm(wd0) * wn / norm(wn);
%i = 1; datasets{i} = 'compare';
%<<plot-lambda>>
%<<plot-g>>

for j = 1:np, j
  for i = 1:N
    wn = randn(T, m + p); wd = wd0 + SD(j) * norm(wd0) * wn / norm(wn);
```

```
    for k = K, eval(methods.comp{k}); Eg{k}(i,j) = e(wh, Ig); Em{k}(i,j) = e(wh, Im);
  end
end

figure, hold on
for k = K, plot(SD, mean(Em{k}), methods.ls{k}), end
box off, ax = axis; axis([SD(1) SD(end), ax(3:4)])
xlabel('noise-to-signal ratio','Interpreter','latex')
ylabel('error $e$, \%','Interpreter','latex')
legend(methods.name(K), 'location', 'northwest')
legend boxoff, print_fig('compare')
save compare
```



Airpass

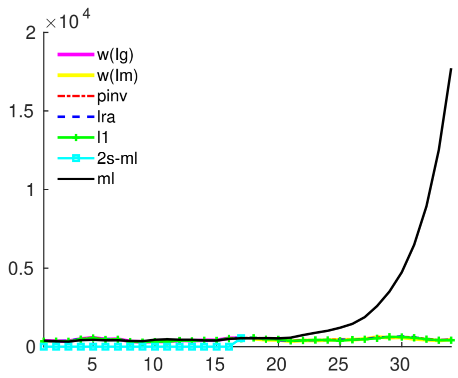
Testing script

```
%% test-airpass

%% data
T = length(y); Td = 110; Tg = 17; Tm = T - Td - Tg;
wd = y(1:Td); w0 = y(Td+1:end); l = 72; K = 1:4;
w  = [w0(1:Tg); NaN(Tm, 1)]; Ig = 1:Tg; Im = Tg+1:Tg+Tm;
<<error>>
ell = n;

i = 1; datasets{i} = name;
<<plot-lambda>>

<<methods>>
for k = K,
  eval(methods.comp{k}); Wh{k} = wh; Eg(k) = e(wh, Ig); Em(k) = e(wh, Im);
end

%% results
res = [{'' 'Ig' 'Im'}; [methods.name(K)', num2cell([Eg' Em'])]]
```

```
figure, hold on
plot(Ig, w0(Ig), 'bo:', 'markersize', 10)
plot(Im, w0(Im), 'r+:', 'markersize', 10)
for k = K, plot(Wh{k}, methods.ls{k}), end %'bo:', 'markersize', 10
legend(['w(Ig)' 'w(Im)' methods.name(K)], 'location', 'northwest')
legend boxoff, box off, ax = axis; axis([1 Tg+Tm 300 650])
print_fig(name), hold off
```

Airpass (true + noise)

```
clear all, close all, load airpass-sys0, name = 'test-airpass-sim';
m = 0; p = 1; n = 6; s = 0.05;
wd0 = initial(sys0, x0, T-1); wn = randn(T, 1);
wd = wd0 + s * norm(wd0) * wn / norm(wn); y = wd;
<<test-airpass>>
```

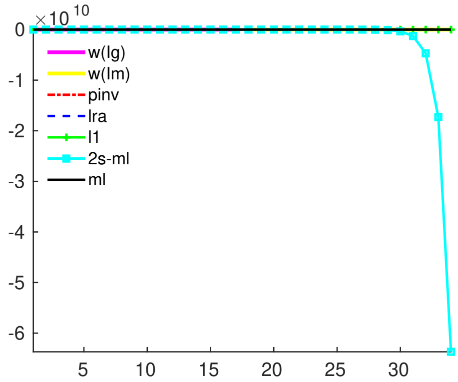



```
{0x0 char}    {'Ig'          }    {'Im'          }
{'pinv'   }   {[1.1296e-13]}     {[    8.8816]}
{'lra'    }   {[    6.2448]}     {[    9.8779]}
{'l1'     }   {[    3.9454]}     {[    9.0138]}
{'2s-ml'  }   {[   95.0686]}     {[       NaN]}
{'ml'     }   {[   15.4850]}     {[1.2589e+03]}
```

Airpass benchmark

```
clear all, close all, load airpass, name = 'test-airpass-real';
m = 0; p = 1; n = 6;
<<test-airpass>>
```

```
{0x0 char}    {'Ig'          }    {'Im'          }
{'pinv'   }   {[5.0719e-14]}     {[    3.9168]}
{'lra'    }   {[    4.0384]}     {[    5.2688]}
{'l1'     }   {[    3.3664]}     {[    3.3387]}
{'2s-ml'  }   {[    4.0572]}     {[3.5208e+09]}
```



DAISY interpolation

N	data set name	T	m	p	ℓ
1	Destillation column	90	5	3	1
2	pH process	2001	2	1	6
3	Hair dryer	1000	1	1	5
4	Heat flow density	1680	2	1	2
5	Heating system	801	1	1	2
6	Lake Erie	57	5	2	1
7	CD-player arm	2048	2	2	1
8	Robot arm	1024	1	1	4
9	Steam heat exchanger	4000	1	1	2
10	Tank reactor	7500	1	2	1
11	Steam generator	9600	4	4	1

```
%% data-sets
switch i
  case 1, destill          , ell = 1; l = 2.63;
  case 2, pHdata           , ell = 6; l = 6.5;
  case 3, dryer            , ell = 5; l = 0.3;
  case 4, thermic_res_wall, ell = 2; l = 5;
  case 5, heating_system   , ell = 2; l = 0.5;
  case 6, erie             , ell = 1; l = 25;
  case 7, CD_player_arm    , ell = 1;
  case 8, robot_arm        , ell = 4;
  case 9, exchanger        , ell = 2;
  case 10, cstr            , ell = 1;
  case 11, steamgen        , ell = 1;
end
```

```
clear all, close all, rng(0, 'twister'), addpath ~/slra/ident/data/
datasets = {'destill', 'pHdata', 'dryer', 'thermic-wall', 'heating-system'};
<<methods>>
F = [0.05 0.1 0.2]; K = [1 3]; I = 1:5;
for i = 1:length(I)
  datasets{i}
  <<data-sets>>
  m = size(u, 2); p = size(y, 2); n = ell * p;
  T = size(u, 1); Td = round(3 / 4 * T); Ti = 1:Td; Tv = Td+1:T;
  wd = [u(Ti, :) y(Ti, :)];
  w0 = [u(Tv, :) y(Tv, :)];
  nt = length(w0(:)); nm = round(F(2) * nt)
```

```
  rng(0), Im = randperm(nt, nm); w = w0; w(Im) = NaN;
  Ig = setdiff(1:nt, Im);
  <<error>>
  <<plot-lambda>>
  for k = 1:length(K)
    eval(methods.comp{K(k)});
    Eg(i, k) = e(wh, Ig);
    Em(i, k) = e(wh, Im);
  end
  <<plot-g>>
end

%% results
res = [[' '; datasets'] [methods.name(K); num2cell(Em)]]
```