Identification of Dynamical Systems: Homework 1

Ivan Markovsky

Ivan

Assignment

The tasks of the assignment are:

T1 describe a method for checking whether a given signal $w_{\mathrm{d}} \in (\mathbb{R}^q)^T$ is a trajectory of a system $\mathscr{B} \in \mathscr{L}^q$,

T2 implement the method in Matlab in a function `is_trajectory`,

T3 test the function `is_trajectory` on simulation examples.

However, part of the assignment is to critically reflect on it, make it more specific, and (if needed) modify / expand it.

Reformulated assignment

Without a bound on the system's complexity, there is always a system $\mathscr{B} \in \mathscr{L}^q$, for which any given signal $w_{\mathrm{d}} \in (\mathbb{R}^q)^T$ is an exact trajectory, *i.e.*, $w_{\mathrm{d}} \in \mathscr{B}|_T$. Indeed, when all variables are inputs the model $\mathscr{B} = (\mathbb{R}^q)^{\mathbb{N}} \in \mathscr{L}^q$ contains every signal $w_{\mathrm{d}} \in (\mathbb{R}^q)^{\mathbb{N}}$. Alternatively, taking the order of $\mathscr{B}$ sufficiently large (*e.g.*, $n = qT$), again $\mathscr{B}$ explains every finite signal $w_{\mathrm{d}} \in (\mathbb{R}^q)^T$. Task 1 therefore is meaningless (the answer is always "yes") without extra constraints.

How can we modify Task 1 in order to make it meaningful? One possibility is to bound the system's complexity:

> **Reformulation R1 of T1** checking whether a given signal $w_{\mathrm{d}} \in (\mathbb{R}^q)^T$ is a trajectory of a bounded complexity system $\mathscr{B} \in \mathscr{L}^q_{(m,n)}$, where the complexity bound (m,n) is given.

Another possible modification is to search for the smallest complexity exact model:

> **Reformulation R2 of T1** find the minimal complexity (m,n) of an exact model $\mathscr{B} \in \mathscr{L}^q_{(m,n)}$ for w_{d}.

Finally, we may consider the problem where the model is given:

> **Reformulation R3 of T1** checking whether a given signal $w_{\mathrm{d}} \in (\mathbb{R}^q)^T$ is a trajectory of a given system $\mathscr{B}$.

The realization that task 1 is meaningless is an important insight. It is an illustration of the accuracy-complexity trade-off in the case of exact data.

A method for solving the reformulated T1

In order to come up with computational methods, we need a representation of the system $\mathscr{B}$. In what follows, we use a kernel representation $\mathscr{B} = \ker R(\sigma)$. The reason for this choice is that it makes the problem linear in the parameter

$$R := \begin{bmatrix} R_0 & R_1 & \dots & R_\ell \end{bmatrix}.$$

For simplicity, we will consider the SISO case. Then, $m = 1$ and $R \in \mathbb{R}^{1\times 2(n+1)}$.

It can be shown that using the kernel representation, $w_{\mathrm{d}} \in \mathscr{B}|_T$ is equivalent to the system of equations

$$R\mathscr{H}_{n+1}(w_{\mathrm{d}}) = 0.$$

Therefore, with a given model, the test is simply to evaluate $R\mathscr{H}_{n+1}(w_{\mathrm{d}})$. With given complexity n, we need to check if $\operatorname{rank}\mathscr{H}_{n+1}(w_{\mathrm{d}}) = n$. Finally, the minimal complexity of an exact model can be found by computing the rank of the Hankel matrix constructed from w_{d} that is as close to square as possible. Alternatively, the minimal complexity of an exact model can be found recursively by checking the rank of $\mathscr{H}_L(w_{\mathrm{d}})$ for $L = 1, 2, \dots$ with a stopping criterion $\operatorname{rank}\mathscr{H}_L(w_{\mathrm{d}}) < 2L$. The minimal complexity of an exact model is then $n = L - 1$.

Matlab implementation

Since rank computation is a build function in Matlab, all we need for the implementation of `is_trajectory` is a function for the Hankel matrix constructor $\mathscr{H}_L(w_\mathrm{d})$. Such a function (`blkhank`) is described in [1, pages 26–27].

```
function [a, min_n] = is_trajectory(w, opt)
if ~exist('opt')                          % case R1: opt not specified
  a = true;
  L = floor((size(w, 1) + 1) / 3);        % as square as possible H_L(w)
  min_n = rank(blkhank(w, L)) - L;        % rank(H_L(w)) = L + n
elseif isa(opt, 'double')                 % case R2: opt is the order
  a = (rank(blkhank(w, opt + 1)) < 2 * (opt + 1));
elseif isa(opt, 'lti')                    % case R3: opt is a given system
  a = (norm(tf2ker(tf(opt)) * blkhank(w, order(opt) + 1)) < 1e-10);
else
  error('opt is wrong type')
end

function R = tf2ker(H)
[Q P] = tfdata(tf(H), 'v'); R = vec(fliplr([Q; -P]))';
```

Test example

```
%% hw1.m
n = 4; T = 100; sys = drss(n);
u = rand(T, 1); y = lsim(sys, u, [], rand(n, 1)); w = [u y];

[a, min_n] = is_trajectory(w) % test 1: find minimal complexity
is_trajectory(w, n - 1)       % test 2: under specify order
is_trajectory(w, n)           % test 3: given the exact order
is_trajectory(w, n + 1)       % test 4: over specify order
is_trajectory(w, sys)         % test 5: give the correct system
is_trajectory(w, drss(n))     % test 6: give wrong system
```

```
>> hw1
a = logical 1
min_n = 4
ans = logical 0
ans = logical 1
ans = logical 1
ans = logical 1
ans = logical 0
```

TODO

- generalize to MIMO systems
- approximation in case of noisy data

References

[1] I. Markovsky. *Low-Rank Approximation: Algorithms, Implementation, Applications*. 2nd edition. Springer, 2019.

Feedback

Stijn

- Prepare a single pdf file that contains:
  - explanation what you did,
  - the code you wrote and the results you obtained,
  - discussion of the results and questions.

Amedeo

- Using LaTeX is a good alternative to `publish`. It gives more control but is less straightforward to use (there are more choices to make and extra tools to learn).
- I like the contents section in the beginning.
- There are ways of including the m-files in the LaTeX document of the report. This has advantages:
  1. it avoids having two versions of the code that you need to synchronize manually,
  2. allows fortification of the code,
  3. avoids the need to change manually the source (`\{`, `\&`, *etc*), which is time consuming and error prone.
- By using LaTeX, why don't you write the text in the document and not as comments in the m-files? This is the main advantage of using LaTeX in the first place.
- Why you define two systems for the MIMO and SISO cases? You can run the test with different simulation parameters and SISO is just a special case of MIMO.
- When including figures put captions that explain what the figures shows and why you show it.
- I don't see the point of plotting a random trajectory (except maybe to check if there is something wrong with the data).
- `misfit = norm(R * createHankel(w, order(sys) + 1), 'fro')` is "residual" (or "equation error"). It is different from the "misfit" defined in the lectures.
- Add discussion of the results.
- Explain what `createHankel` does in case `l` is not specified. I don't understand the implementation of `createHankel`.
- "Can I use `findstates`, or should I implement it myself?" — Re-implement it yourself. This is an essential part of your solution.
- "How to compare 2 SS models?" -> this is a nice problem on its own! We will address it as separately. Meanwhile think of possible solutions.

Freja

- `y` = data_{out}= — use instead `norm(y - data_out) < tol` where `tol` is user defined tolerance for checking numerically when a number is zero.
- The sequence $w \mapsto h \mapsto \operatorname{rank} \mathscr{H}_L(h)$ is a possible solution of the problem. Compare the implementation of $w \mapsto h$ using Z-transform (symbolic toolbox) with the subspace method shown in the lectures.

Leander

- Next step: test the methods on noisy data.

Arno

- "Intended Solution" — "intended" suggests bias that I tried to avoid (but probably failed). Call it just what it is: "solution based on a kernel representation".
- Add the tests of `check_trajectory_2` and `get_complexity` right after the functions.
- Listing 7 shows a wrong function.
- "better in runtime" — the example is too small to make such a conclusion
- "MATLAB profiler" - nice to see that you use a profiler!
- Next step: test the methods on noisy data.

Reports

Stijn

```
clear
close all
clc

%%
n_real = randi(10);          %Randow system order
Ts = 0.1; %s

t = 0:Ts:10;              %Time vector
u = rand([1,length(t)]);     %Random input over time
x0 = rand([1,n_real]);       %Random starting conditions
sys_c = rss(n_real);
sys = c2d(sys_c,Ts);

y = (lsim(sys,u,t,x0))';
w = [u;y];

%%
[H,l] = construct_Hankel(w);

%%
[is_traj,n_min] = is_trajectory(w)
is_traj = is_trajectory(w,n_real-1)
is_traj = is_trajectory(w,n_real)
%is_traj = is_trajectory(w,sys)          %Still somewhere a mistake

%% Estimate a system from the impulse response
h = impulse(sys,t(end))';                %Take impulse response from system
sys_est = sys_estimation(h,n_real,Ts);     %Estimation not correct, cannot find mist

%% Determine m,n
% Theory class: Dim(nullspace(Hankel)) = m*L+n
```

```
% Choose two different L: L1 and L2
%
% dim1 = m*L1+n          (1)
% dim2 = m*L2+n          (2)
%
% (1)-(2) = m*(L1-L2) => m = ((1)-(2))/(L1-L2)
% Get n form (1) or (2)

L1 = 10;
L2 = 20;
dim1 = size(null(construct_Hankel(w,L1)),2);     %According to https://www.mathworks.
                                                 %The nullity of a matrix A
                                                 %is given by size(Z,2) with
                                                 %Z = null(A)
dim2 = size(null(construct_Hankel(w,L2)),2);

m = (dim1-dim2)/(L1-L2);
n = dim1-m*L1;

function [sys] = sys_estimation(imp,n,Ts)
D = imp(1);                                      %D=h(0)

H = construct_Hankel(imp(2:end));                %Construct square Hankel with h without D
[U,S,V] = svd(H);                                %SVD decomposition to get Observability and
U = U(:,1:n);
S = S(1:n,1:n);
V = V(:,1:n);
Obser = U*sqrt(S);
Contr = sqrt(S)*V.';

C = Obser(1,:);                                  %Cbser = [C; CA; CA²...]
B = Contr(:,1);                                  %Contr = [B, AB, AB²...]
A = Obser(n+1:end,:)\Obser(1:end-n,:);           %[C;CA;CA²...CA^(L-1)]*A = [CA;CA²...CA^L] =

sys = ss(A,B,C,D,Ts);
end
```

Identification of Dynamical Systems
Homework

Amedeo Varano

October 10, 2021

1 Homework of 03/10/2021, corrected

Contents

Homework 03/10/21 corrected

```
%Amedeo

%Task: Create a function "is_trajectory" that can determine if a given data
%sequence is a trajectory of a given LTI system.
%Extensions:
% * Give "how close" the data sequence is to the given LTI system.
% * Without specifying any system, determine if the data could be the output of
% an LTI system.
% * Without specifying any system, determine if the data could be the output of
% an LTI system and say of which order the system (minimally) is.
```

Clean up workspace

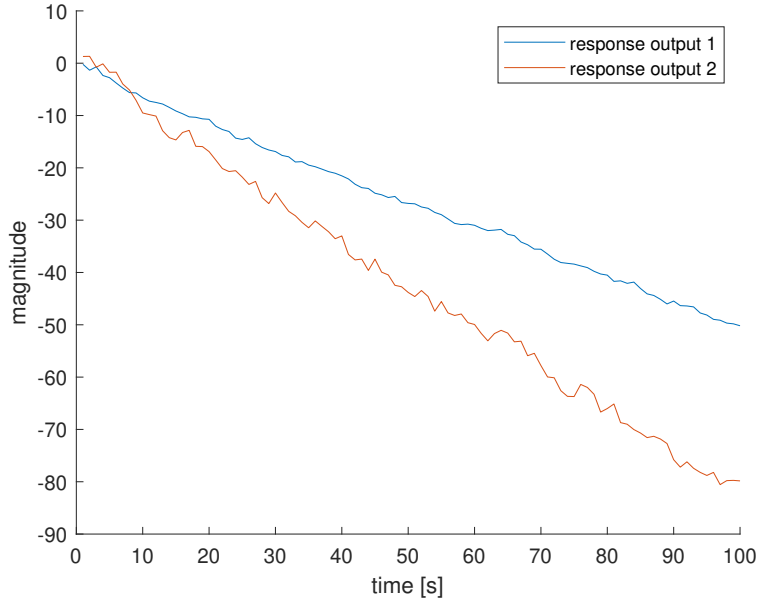
```
close all
clear
clc
```

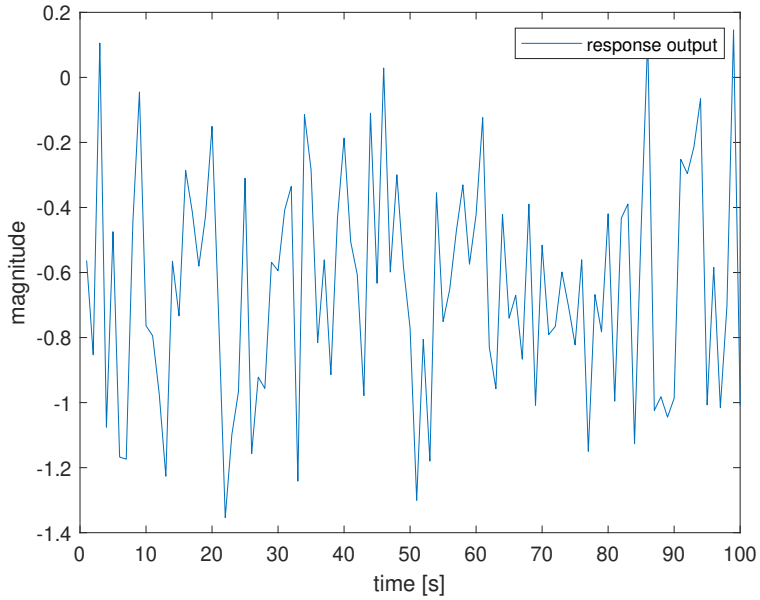
Initialize (discrete) random systems

```
n = 5; %order
m = 3; %# inputs
p = 2; %# outputs

sys = drss(n, p, m); %random CT system
sysSISO = drss(n, 1, 1); %random CT system
```

Create data sequence

```
T = 100;
u = rand([T, m]); %random input
x0 = rand([n, 1]); %random initial state

y = lsim(sys, u, [], x0); %system's response
w = [u, y].';

figure
hold on
for i = 1:p
plot(1:T, y(:, i), 'DisplayName', sprintf('response output %i', i))
end
xlabel('time [s]'), ylabel('magnitude'), legend

uSISO = rand([T, 1]); %random input
x0SISO = rand([n, 1]); %random initial state

ySISO = lsim(sysSISO, uSISO, [], x0SISO); %system's response
wSISO = [uSISO, ySISO].';

figure
plot(1:T, ySISO, 'DisplayName', 'response output')
xlabel('time [s]'), ylabel('magnitude'), legend
```





Given an LTI syst, determine if the given w is a trajectory

```
%The code below is not representative, because sys2kernel is not implemented for
%the MIMO case.
% R = sys2kernel(sys);
% misfit = norm(R * createHankel(w, order(sys) + 1), 'fro');
```

```
% fprintf('For the exact model and the exact w, the misfit is %g\n', misfit)
%
% R = sys2kernel(drss(n, p, m));
% misfit = norm(R * createHankel(w, order(sys) + 1), 'fro');
% fprintf('For the wrong model and the exact w, the misfit is %g\n', misfit)

R = sys2kernel(sysSISO);
misfit = norm(R * createHankel(wSISO, order(sysSISO) + 1), 'fro');
fprintf('For the exact model and the exact w, the misfit is %g\n', misfit)

R = sys2kernel(drss(n, 1, 1));
misfit = norm(R * createHankel(wSISO, order(sysSISO) + 1), 'fro');
fprintf('For the wrong model and the exact w, the misfit is %g\n', misfit)
%Something clearly goes wrong here...
%The exact misfit is way too large (sometimes larger than when the wrong system
%is given...
```

```
For the exact model and the exact w, the misfit is 11.317
For the wrong model and the exact w, the misfit is 106.643
```

Functions

createHankel.m

```
function H = createHankel(w, l, T)
%Given data sequence w, create the full Hankel matrix. w(k) = [u(k); y(k)],
%with the data stored as columns.
%Additional integers l and T can be specified to create a Hankel matrix with
%l rows and T columns. Note: T + l - 1 <= length(w)

switch nargin
case 1 %only w is specified (T = l)
l = floor((length(w) + 1)/2);
T = l;
case 2 %only l is specified
T = length(w) - l + 1;
case 3 %both l and T are specified
if T + l - 1 \ensuremath{>} length(w)
warning("T + l - 1 <= length(w) not satisfied, " + ...
"creating square Hankel matrix instead.");
l = floor((length(w) + 1)/2);
T = l;
end
end
```

```
step = size(w, 1);
H = NaN([step * l, T]); %initialise output to nonvalid entries
wprime = w(:); %create one big column vector of w
for i = 1:T
H(:, i) = wprime((1:step*l)+(i-1)*step); %fill Hankel matrix per column
end
end
```

sys2kernel.m

```
function R = sys2kernel(sys)
%Convert a given system to kernel representation, i.e. R * w = 0.

[num, denom] = tfdata(sys); %extract num and denom

%Coefficients in num, denom are ordered in decreasing power
%Reorder them in increasing power
for i = 1:size(num, 1)
for j = 1:size(num, 2)
num\{i, j\} = fliplr(num\{i, j\});
denom\{i, j\} = fliplr(denom\{i, j\});
end
end

%In general Y(f) = G(f) U(f)
%In the MIMO case, this becomes:
%[Y1 ... Yp].' = [G11 ... G1m; ...; Gp1 ... Gpm] [U1 ... Um].'
%
%The kernel representation: R w = 0
%Which can be written as
%[R^{U1} ... R^{Um} R^{Y1} ... R^{Yp}] [U1; ...; Um; Y1; ...; Yp] = 0
%
%In other words:
%R^{U1} U1 + ... + R^{Um} Um + R^{Y1} Y1 + ... R^{Yp} Yp = 0
%We also have that
%Yi = Gi1 U1 + ... Gim Um
%Substituting this results in the equation above results in a sum
%\sum_{i=1}^{m} (R^{Ui} + R^{Y1} G1i + ... + R^{Yp} Gpi) Ui = 0
%Since the equality must hold in general,
%(R^{Ui} + R^{Y1} G1i + ... + R^{Yp} Gpi) = 0 \forall i = 1, ..., m
%This can be written as
%[R^{Ui} R^{Y1} ... R^{Yp}] [1 G1i ... Gpi].' = 0
%i.e. a null-space calculation.

%I don't immediately see how to implement this general case in MATLAB, so I will
%stick to the SISO case for the moment...
```

```
if numel(num) == 1
%In the SISO case the equation simplifies to
%R^{U} U = - R^{Y} Y
%and
%Y = num/denom U
%--> R^{U} = num, R^{Y} = - denom,
R = [num\{1, 1\}, - denom\{1, 1\}];
else
R = zeros([1, order(sys) + 1]);
end
end
```

2 Homework of 10/10/2021

Contents

Homework 10/10/21

```
%Amedeo

%Tasks:
% * Create a function that extracts the order and number of inputs from w
% * Implement a function that creates a SS model starting from impulse response
% data
```

Clean up workspace

```
close all
clear
clc
```

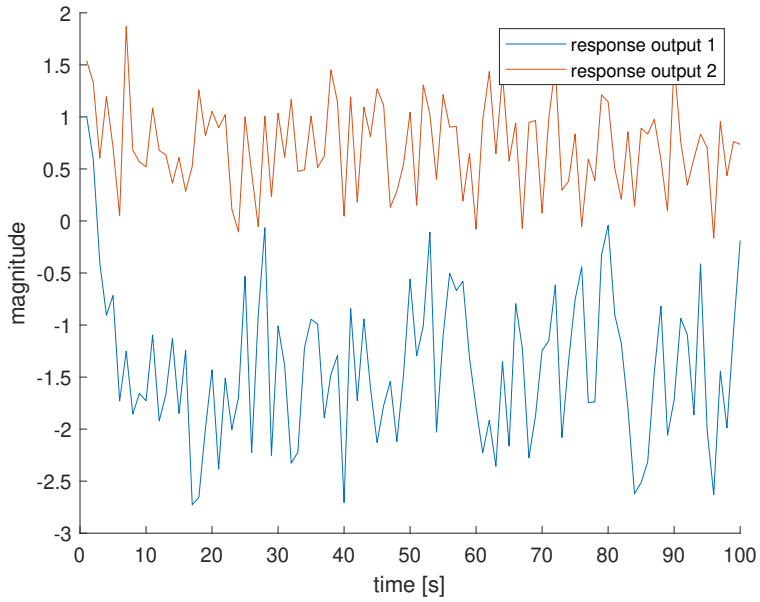
Initialize (discrete) random systems

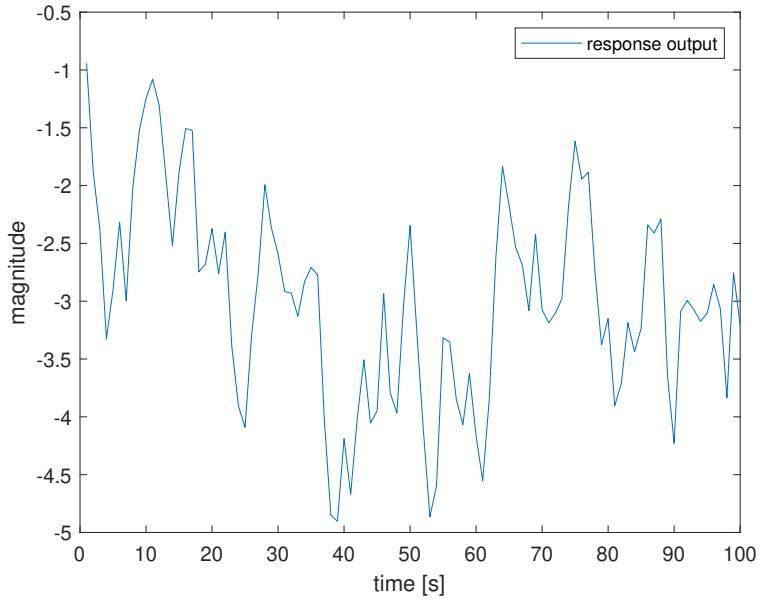
```
n = 5; %order
m = 3; %# inputs
p = 2; %# outputs
```

```
sys = drss(n, p, m); %random CT system
sysSISO = drss(n, 1, 1); %random CT system
```

Create data sequence

```
T = 100;
u = rand([T, m]); %random input
x0 = rand([n, 1]); %random initial state

y = lsim(sys, u, [], x0); %system's response
w = [u, y].';

figure
hold on
for i = 1:p
plot(1:T, y(:, i), 'DisplayName', sprintf('response output %i', i))
end
xlabel('time [s]'), ylabel('magnitude'), legend

uSISO = rand([T, 1]); %random input
x0SISO = rand([n, 1]); %random initial state

ySISO = lsim(sysSISO, uSISO, [], x0SISO); %system's response
wSISO = [uSISO, ySISO].';

figure
plot(1:T, ySISO, 'DisplayName', 'response output')
xlabel('time [s]'), ylabel('magnitude'), legend
```





Extract rank

```
[mhat, nhat] = estimateMandN(w);
fprintf('Estimated number of inputs: %i (true value: %i)\n', mhat, m)
fprintf('Estimated model order: %i (true value: %i)\n', nhat, n)
%Something not quite right in the MIMO case...
```

```
[mhat, nhat] = estimateMandN(wSISO);
fprintf('Estimated number of inputs: %i (true value: %i)\n', mhat, 1)
fprintf('Estimated model order: %i (true value: %i)\n', nhat, n)
```

```
Estimated number of inputs: 5.000000e-01 (true value: 3)
Estimated model order: 0 (true value: 5)
Estimated number of inputs: 1 (true value: 1)
Estimated model order: 5 (true value: 5)
```

Create SS model from impulse response data

```
u = zeros([T, m]); %impulse input
u(1, :) = 1;
x0 = rand([n, 1]); %random initial state

h = lsim(sys, u, [], x0).'; %system's response

syshat = impresp2ss(h);
%MATLAB can estimate the initial state with
%x0 = findstates(syshat, h, Inf);
%Can I use this, or should I implement it myself?
%How to incorporate this estimate in the system?
%How to compare 2 SS models? Compare B and D (independent of the states)?

uSISO = [1, zeros(1, T - 1)];
x0SISO = rand([n, 1]); %random initial state

hSISO = lsim(sysSISO, uSISO, [], x0SISO); %system's response

sysSISOhat = impresp2ss(hSISO);
%Idem MIMO case
```

```
Warning: T + l - 1 <= length(w) not satisfied, creating square Hankel matrix
instead.
Warning: T + l - 1 <= length(w) not satisfied, creating square Hankel matrix
instead.
```

Functions

estimateMandN.m

```
function [mhat, nhat] = estimateMandN(w, maxiter)
if nargin == 1
```

```
maxiter = 20;
end
l = 0;
H = createHankel(w, l + 1);

while rank(H) == min(size(H)) \&\& l \ensuremath{<} maxiter %while not rank deficient && l r
l = l + 1;
H = createHankel(w, l + 1);
end

if size(w, 1) == 2 %In SISO case:
mhat = 1; %estimated number of inputs
nhat = l; %estimated model order
else %In MIMO case (dim(B|L) = m*L + n):
T = size(w, 2);
L1 = floor(0.8 * T);
L2 = floor(0.9 * T);

r1 = rank(createHankel(w(:, 1:L1)));
r2 = rank(createHankel(w(:, 1:L2)));

mhat = (r1 - r2)/(L1 - L2);
nhat = r2 - mhat * L2;
%Note: this does not seem to work properly...
end
end
```

impresp2ss.m

```
function sys = impresp2ss(h)
D = h(1);

l = length(h) - 1;
H = createHankel(h(:, 2:end), l, l);

%https://en.wikipedia.org/wiki/Rank_factorization#Singular_value_decomposition
[U, S, V] = svd(H);
r = rank(S);
obs = U(:, 1:r); %observability matrix
contr = S(1:r, 1:r) * V(:, 1:r)'; %controllability matrix

step = round(size(obs, 1)/l);
C = obs(1:step, :);
B = contr(:, 1:round(size(contr, 2)/l));

%Implement shift eq
```

```
A = obs(1:end-step, :)\ensuremath{\backslash}obs(1+step:end, :);

sys = ss(A, B, C, D, -1);
%How to estimate initial state?
%x0 = findstates(sys, h, Inf); %?
end
```

Homework – Determination complexity

Problem statement

Determine the complexity (the number of inputs m and the order n) of an LTI system from data without prior knowledge

Theoretical background

Given an LTI system B of order n with m inputs and p outputs, consider the subspace

$$B|_L = \{w(1), \dots, w(L) \mid w \in B\}$$

where the data

$$w = \begin{bmatrix} u \\ y \end{bmatrix} \in \mathbb{R}^q \text{ with } q = m + p$$

The dimension of this subspace is given by

$$\dim(B|_L) = mL + n \quad (1)$$

On the other hand, under the assumption that the fundamental lemma is satisfied, so if the system B is controllable and the input u is persistent of excitation of L + n, the following equality holds

$$B|_L = \text{image } (H_L(w))$$

where the Hankel matrix of the data $H_L(w)$ is constructed as

$$H_L(w) = \begin{pmatrix} w(1) & w(2) & \cdots & w(T-L+1) \\ w(2) & w(3) & \cdots & w(T-L+2) \\ \vdots & & \ddots & \vdots \\ w(L) & w(L+1) & \cdots & w(T) \end{pmatrix} \in \mathbb{R}^{qL \text{ x } (T-L+1)}$$

Therefore, the dimension of the subspace can also be found as

$$\dim(B|_L) = \dim\Big(\text{image}\left(H_L(w)\right)\Big) = rank\left(H_L(w)\right) \quad (2)$$

Combining (1) and (2) for two different lengths L_1 and L_2, results in a system of equations that can be solved to m and n

$$\begin{cases} mL_1 + n = rank\left(H_{L_1}(w)\right) \\ mL_2 + n = rank\left(H_{L_2}(w)\right) \end{cases} \Leftrightarrow \begin{pmatrix} L_1 & 1 \\ L_2 & 1 \end{pmatrix} \begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} rank\left(H_{L_1}(w)\right) \\ rank\left(H_{L_2}(w)\right) \end{pmatrix}$$

Note that, since the rank is bounded by the number of columns of $H_L(w)$

$$mL + n \ \leq \ T - L + 1 \ \Leftrightarrow \ L \leq \text{floor}\left(\frac{T+1-n}{1+m}\right)$$

However, this expression contains the unknowns m and n, which can be respectively estimated as q-1 (in the extreme case of only 1 output) and q, such that the upper bound on L becomes

$$L \leq \text{floor}\left(\frac{T+1-q}{q}\right)$$

Algorithm

1. Construct the Hankel matrix of the data for two different lengths L_1 and L_2
2. Compute the dimension of the subspace
3. Determine m and n by solving the system of equations

```
function [m,n] = complexity(w) % determine # inputs m and order n of the
system from data

    w = w(:,2:end);
    q = size(w,1);  % # inputs + # outputs
    T = size(w,2);  % # time instants

    % construct the Hankel matrix of the data
    L1 = floor((T+1-q)/q);
    L2 = L1-1;

    H1 = Hankel(w,L1,0);
    H2 = Hankel(w,L2,0);

    % compute the dimension of the subspace
    dim1 = rank(H1);
    dim2 = rank(H2);

    % determine m and n
    A = [L1 1; L2 1];
    b = [dim1 ; dim2];
    x = A\b;

    m = x(1);
    n = x(2);

end
```

The function `Hankel` constructs the square or rectangular Hankel matrix of the data

```
function H = Hankel(w,L,square) % construct the Hankel matrix of the data

    q = size(w,1);  % # inputs + # outputs
    T = size(w,2);  % # time instants

    if square == 1% square Hankel matrix
        col = L;
    else % rectangular Hankel matrix
        col = T-L+1;
    end

    H = zeros(q*L,col);
    for  t1 = 1:L
        for t2 = 1:col
            H((t1-1)*q+1:t1*q,t2) = w(:,t1+t2-1);
        end
    end

end
```

Simulation

```
%% Simulations
m = 2; % # inputs
p = 2; % # outputs
n = 3; % # states

T = 100; % # time instants
t = 0:T; % time vector

sys = rss(n,p,m);
sys = c2d(sys,1,'zoh');
A = sys.A; B = sys.B;
C = sys.C; D = sys.D;

u = randi(100,[m T+1]);      % input
y = lsim(sys,u,t)';          % output
w = [u; y];                  % data

[m2,n2] = complexity(w);
if abs(m2 - m) < 1e-8 && abs(n2 - n) < 1e-8
    fprintf('m and n are correctly determined from data \n')
end
```

The code works for different values of m and n

Homework - Realisation problem

Problem statement

Given a finite impulse response (Markov parameters)

$$h = \big(h(0), h(1), \dots h(T)\big)$$

Find a state space model with n states, m inputs and p outputs

$$\begin{cases} x(t+1) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \qquad (1)$$

Theoretical background

Consider the reverse problem: the impulse response matrix (p x m matrix) at time t can be found from the state space model as follows

$$h(t) = \begin{cases} D & t = 0 \\ CA^{t-1}B & t > 0 \end{cases} \qquad (2)$$

Therefore, the D matrix is immediately known

$$h(0) = \boldsymbol{D}$$

To find the A,B and C matrices, one has first to construct the τ-th Hankel matrix of the impulse response (pτ x mτ matrix)

$$H_\tau(h) = \begin{pmatrix} h(1) & h(2) & \cdots & h(\tau) \\ h(2) & h(3) & \cdots & h(\tau+1) \\ \vdots & & \ddots & \vdots \\ h(\tau) & h(\tau+1) & \cdots & h(2\tau-1) \end{pmatrix}$$

where $\tau_{\max} = \text{floor}\left(\frac{T+1}{2}\right)$ since the impulse response is only available until time instance T

As can be seen from (2), this Hankel matrix can be rewritten as

$$H_\tau(h) = \begin{pmatrix} \boldsymbol{C} \\ CA \\ \vdots \\ CA^{\tau-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{B} & AB & \cdots & A^{\tau-1}B \end{pmatrix} = O_\tau R_\tau$$

This is nothing more than a rank revealing factorisation: the pτ x mτ Hankel matrix $H_\tau(h)$ is factorised in the pτ x n observability matrix O_τ and the n x mτ reachability matrix R_τ, such that the rank of $H_\tau(h)$ is equal to the order n of the system.

In practice, this rank revealing factorisation is obtained via singular value decomposition: the pτ x mτ matrix $H_\tau(h)$ of rank n is factorised in pτ x pτ matrix U, mτ x mτ matrix V and pτ x mτ pseudo-diagonal matrix Σ, of which only the first n diagonal elements are nonzero.

$$H_\tau(h) = U\Sigma V^T = \begin{pmatrix} U_1^{p\tau \text{ x } n} & U_2^{p\tau \text{ x } (p\tau-n)} \end{pmatrix} \begin{pmatrix} \Sigma^{n \text{ x } n} & 0^{n \text{ x } (m\tau-n)} \\ 0^{(p\tau-n) \text{ x } n} & 0^{(p\tau-n) \text{ x } (m\tau-n)} \end{pmatrix} \begin{pmatrix} V_1^{m\tau \text{ x } n^T} \\ V_2^{m\tau \text{ x } (m\tau-n)^T} \end{pmatrix}$$

$$= U_1^{p\tau \text{ x } n} \Sigma^{n \text{ x } n} V_1^{m\tau \text{ x } n^T} = O_\tau R_\tau$$

Finally, the matrix A can be found from the following shift equations

$$O_\tau = \begin{pmatrix} C \\ CA \\ \vdots \\ CA^{\tau-1} \end{pmatrix} \Rightarrow \begin{pmatrix} C \\ CA \\ \vdots \\ CA^{\tau-2} \end{pmatrix} \boldsymbol{A} = \begin{pmatrix} CA \\ CA^2 \\ \vdots \\ CA^{\tau-1} \end{pmatrix}$$

Non-uniqueness

Note that the realisation problem has no unique solution, because the states and therefore also the state space model (1) are not unique

$$z(t) = Tx(t) \Leftrightarrow x(t) = T^{-1}z(t)$$

with T an invertible n x n matrix

$$\begin{cases} z(t+1) = TAT^{-1}z(t) + TBu(t) \\ y(t) = CT^{-1}z(t) + Du(t) \end{cases}$$

This non-uniqueness can be found in the rank factorisation step

$$H_\tau(h) = O_\tau R_\tau = (O_\tau T^{-1})(TR_\tau)$$

Algorithm

The conversion from state space model to finite impulse response can be done by implementing (2) or using the Matlab function `impulse`

```
function [h] = ss2impulse(A,B,C,D,T)

    m = size(D,2); % # inputs
    p = size(D,1); % # outputs

    h = zeros(p,m,T+1);
    h(:,:,1) = D;
    for t = 1:T
        h(:,:,t+1) = C*A^(t-1)*B;
    end

end
```

The conversion from finite impulse response to state space model consists of the following steps

0. $h(0) \Rightarrow$ **D**
1. Construct $H_\tau(h)$
2. Determine O_τ and $R_\tau \Rightarrow$ **B & C**
3. Solve the shift equations $\Rightarrow$ **A**

```
function [A,B,C,D] = impulse2ss(h)

    D = h(:,:,1);        % D = h(0)
    h = h(:,:,2:end);    % h = (h(1),...,h(T))

    p = size(h,1); % # outputs
    m = size(h,2); % # inputs
    T = size(h,3); % # time instants

    % Construct the Hankel matrix of the impulse response
    tau = floor((T+1)/2);
    H = zeros(p*tau,m*tau);
    for  t1 = 1:tau
        for t2 = 1:tau
            H((t1-1)*p+1:t1*p,(t2-1)*m+1:t2*m) = h(:,:,t1+t2-1);
        end
    end
```

```
    % Determine the reachability and observability matrix
    [U,S,V] = svd(H);
    n = rank(S);
    O = U(:,1:n);
    R = S(1:n,1:n)*V(:,1:n)';

    B = R(:,1:m);
    C = O(1:p,:);

    % Solve the shift equations
    O1 = O(1:(tau-1)*p,:);
    O2 = O(p+1:tau*p,:);

    A = O1\O2;

end
```

Simulation

```
%% Simulations
m = 2;  % # inputs
p = 2;  % # outputs
n = 3;  % # states
T = 10; % # time instants

sys = rss(n,p,m);
sys = c2d(sys,1,'zoh');
A = sys.A; B = sys.B;
C = sys.C; D = sys.D;
```

```
% ss 2 impulse
hm = impulse(sys,T);         % compare with Matlab function impulse
hm = permute(hm,[2 3 1]);    % reorder impulse response matrix
h1 = ss2impulse(A,B,C,D,T);

if norm(hm(:)-h1(:),'fro') <= 1e-4 % Frobenius norm < tolerance
    fprintf("The conversion from state space to impulse response works \n")
end
```

```
% impulse 2 ss
[A,B,C,D] = impulse2ss(h1); % different A,B,C,D matrices because realisation
problem has no unique solution
h2 = ss2impulse(A,B,C,D,T); % but same impulse response

if norm(h2(:)-h1(:),'fro') <= 1e-4 % Frobenius norm < tolerance
    fprintf("The conversion from impulse response to state space works \n")
end
```

The code works!

Identification of Dynamical Systems – Homework

Question: Is the time series $w_d = \begin{bmatrix} u_d \\ y_d \end{bmatrix} = \left(\begin{bmatrix} u_d(1) \\ y_d(1) \end{bmatrix}, \dots, \begin{bmatrix} u_d(T) \\ y_d(T) \end{bmatrix} \right)$ a trajectory of the LTI system B

Mathematical interpretation

The LTI system B can be seen as a set of trajectories from the input u to the output y. These trajectories are described by equations (differential equations, transfer functions, state space model).

$$B = \left\{ w = \begin{bmatrix} u \\ y \end{bmatrix} \,\middle|\, equations \right\}$$

Therefore, w_d is a trajectory of the LTI system B if $w_d \in B$.

When the parameters of the equations, and thus the model, are not known, it is always possible to find a non-parametric model B (# parameters = # data points) that fits exactly the data w_d. In that case, the more interesting question is to find the parametric model B (# parameters < # data points) with the smallest possible model order (minimal realisation).

On the other hand, when the parameters of the equations, and thus the model, are given, all one has to do to check whether $w_d \in B$ is plug in the data w_d into the equations and verify that they hold.

Algorithm

Assume a state space model

$$\begin{cases} x(t+1) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases}$$

with n states, m inputs and p outputs

If the model is given, the output data has to be compared with the simulated output obtained by applying the input data to the system.

```
function [] = istrajectory(data_in,data_out,model)

    if nargin == 3 % model is given

        A = model.A; B = model.B;
        C = model.C; D = model.D;

        m = size(D,2); % # inputs
        p = size(D,1); % # outputs
        n = size(A,1); % # states
        T = size(data_in,2); % # time instants

        x = zeros(n,T+1); % assume zero initial conditions
        y = zeros(p,T);
        for t = 1:T
            y(:,t) = C*x(:,t) + D*data_in(:,t);
            x(:,t+1) = A*x(:,t) + B*data_in(:,t);
        end

        if y == data_out
            fprintf("the data is a trajectory of the given system \n")
        else
            fprintf("the data is not a trajectory of the given system \n")
        end
    end
end
```

If the model is not given, a method for constructing a minimal realisation was derived in the course 'System Control Design'. The method is based on the Hankel matrix of the impulse response, which can be calculated from the given data as follows.

The transfer function matrix $H(z)$ can be computed by taking the Z-transform of the discrete time data series. The superscripts j and i denote the indexes in the input and output data vectors respectively.

$$H_{ij}(z) = \frac{Y_d^i(z)}{U_d^j(z)} = \frac{Z\{y_d^i(t)\}}{Z\{u_d^j(t)\}} = \frac{\sum_{t=1}^{T} y_d^i(t)z^{-t}}{\sum_{t=1}^{T} u_d^j(t)z^{-t}} = \frac{\sum_{t=1}^{T} y_d^i(t)z^{T-t}}{\sum_{t=1}^{T} u_d^j(t)z^{T-t}}$$

The impulse response matrix $h(t)$ is then found as the inverse Z-transform of the transfer function matrix $H(z)$.

$$h_{ij}(t) = Z^{-1}\{H_{ij}(z)\}$$

```
function h_series = impulseresponse(data_in,data_out)

    m = size(data_in,1);     % # inputs
    p = size(data_out,1);    % # outputs
    T = size(data_in,2);     % # time instants

    syms z

    Y = zeros(p,1)*z;
    for i = 1:p
        for t = 1:T
            Y(i) = Y(i) + data_out(i,t)*z^(T-t);
        end
    end
    U = zeros(m,1)*z;
    for j = 1:m
        for t = 1:T
            U(j) = U(j) + data_in(j,t)*z^(T-t);
        end
    end
    H = zeros(p,m)*z;
    for i = 1:p
        for j = 1:m
            H(i,j) = Y(i)/U(j);
        end
    end

    h = iztrans(H); % inverse Z transform

    h_series = zeros(p,m,T);
    for t = 1:T
        h_series(:,:,t) = subs(h,t);
    end

end
```

For simplicity, assume a SISO system. The t-th Hankel matrix of the impulse response is then given by

$$H_t = \begin{pmatrix} h(1) & h(2) & \cdots & h(t) \\ \vdots & & \ddots & \vdots \\ h(t) & h(t+1) & \cdots & h(2t-1) \end{pmatrix}$$

Note that because data is only available until time instance T, the maximum $t_{\max} = floor(\frac{T+1}{2})$.
The minimum order $t_{\min}$ is such that $\det(H_t) = 0 \ \forall\, t: t_{\min} < t \leq t_{\max}$. In other words, $H_{t_{\min}}$ is the largest full rank Hankel matrix.

```
function [] = istrajectory(data_in,data_out,model)

    if nargin == 3 % model is given

        …

    else % model is not given

        h = impulseresponse(data_in,data_out); % assume SISO

        T = size(data_in,2); % # time instants
        tmax = floor((T+1)/2);
        for t = tmax:-1:1
            H = hankel(h(1:2*t-1));
            H = H(1:t,1:t);
            if round(det(H),10) ~= 0
                break
            end
        end

        if t == tmax
            fprintf("no minimal realisation with this method \n")
        else
            fprintf("minimal realisation of order %.i \n",t)
        end

    end

end
```

Simulation

```
% model is given
m = 2; % # inputs
p = 2; % # outputs
n = 3; % # states

T = 10; % # time instants
t = 0:T-1; % time vector

A = [3 0 5; 1 2 4; 0 6 3];  % nxn matrix
B = [2 0; 0 1; 4 2];        % nxm matrix
C = [1 3 6; 5 0 4];         % pxn matrix
D = [1 0; 0 1];             % pxm matrix
sys = ss(A,B,C,D,-1);       % undefined sampling time

% example 1: data and model are given - data is a trajectory (data is
generated as such)
ud = randi(100,[m T]);      % input
yd = lsim(sys,ud,t)';       % output (assume zero initial conditions)

istrajectory(ud,yd,sys)

% example 2: data and model are given - data is not a trajectory (data is
randomly generated)
ud = randi(100,[m T]);      % input
yd = randi(100,[p T]);      % output

istrajectory(ud,yd,sys)
```

```
% model is not given
m = 1; % # inputs
p = 1; % # outputs
n = 3; % # states

T = 10; % # time instants
t = 0:T-1; % time vector

A = [2 1; 0 1];          % nxn matrix
B = [1; 2];              % nxm matrix
C = [1 0];               % pxn matrix
D = 0;                   % pxm matrix
sys = ss(A,B,C,D,-1);    % undefined sampling time

% example 3: only data is given - find minimum realisation (data is
generated from known model of order 2)
ud = randi(100,[m T]);       % input
yd = lsim(sys,ud,t)';        % output (assume zero initial conditions)

istrajectory(ud,yd)

% example 4: only data is given - find minimum realisation (data is randomly
generated)
ud = randi(100,[m T]);       % input
yd = randi(100,[p T]);       % output

istrajectory(ud,yd)
```

The results are as expected:

```
the data is a trajectory of the given system
the data is not a trajectory of the given system
a minimal realisation of order 2 can be constructed
no minimal realisation can be constructed with this method
```

Compare with Kernel representation

The criterium for an exact trajectory becomes

$$w_d \in B \Leftrightarrow [R_0 \dots R_L] H_{L+1}(w_d) = 0$$

Where the Hankel matrix of the data is given by

$$H_{L+1}(w) = \begin{pmatrix} w(1) & w(2) & \cdots & w(T-L) \\ w(2) & w(3) & \cdots & w(T-L+1) \\ \vdots & & \ddots & \vdots \\ w(L+1) & w(L+2) & \cdots & w(T) \end{pmatrix}$$

```
function H = Hankel(w,L) % construct the Hankel matrix of the data

    q = size(w,1);  % # inputs + # outputs
    T = size(w,2);  % # time instants

    H = zeros(q*L,T-L+1);
    for  t1 = 1:L
        for t2 = 1:T-L+1
            H((t1-1)*q+1:t1*q,t2) = w(:,t1+t2-1);
        end
    end

end
```

Using the Matlab functionality `tic toc`, the computation time of the symbolic approach tends to be higher than the one of the kernel representation approach.

Leander Hemelhof: Assignment 1 (+ Assignment 2)

Introduction

The goal of the first assignment is twofold:

1. Given a reference LTI model and a trajectory of valid size for the model, test if this trajectory could come from the reference model.
2. Given only a trajectory, number of inputs and maximal order, verify if this trajectory could come from a bounded complexity LTI model, and if this is the case, give a minimal realization of this model.

The first part is mostly a repeat of the first report, but with a few tweaks based on previous feedback. In the second part some additions are made: based on the new information available, two new functions were made: a function to estimate the model order *n* and the number of inputs *m* based only on the trajectory itself and a function that checks whether the given trajectory comes from a bounded complexity LTI system using a kernel based approach combined with the previous function. A simple comparison of the mean runtime between this new function and the previous subspace approach will be made at the end.

Initialization

This part of the code initializes the amount of trajectory points, the order and dimension of the reference model, and generates a random discrete model to use as reference. Two trajectories are then generated using a random input signal, one with the outputs given by the model, the other with an uncorrelated random output:

```
clear
close all
clc

N = 1000;
n = 5;
nu = 1;%2;
ny = 1;%3;
x1ref = 15*randn(n, 1);
ref = drss(n, ny, nu);
ref.Ts = 1/50;
u1 = randn(nu, N);
u2 = randn(nu, N);
y1 = lsim(ref, u1, [], x1ref).';
traj1 = [u1;y1];
y2 = randn(ny, N);
traj2 = [u2;y2];
```

Verification

This part of the code runs the functions for the two parts:

```
is1 = verifyModelOrigin(ref, traj1, nu);
```

```
is2 = verifyModelOrigin(ref, traj2, nu);
[is21, model1, goodModelFound1] = isSystemTrajectory(traj1, nu, 20, 1/50);
[is22, model2, goodModelFound2] = isSystemTrajectory(traj2, nu, 20, 1/50);
```

Results

Knowing that *traj1* is a part of the model and *traj2* isn't, it looks like the functions give the expected result. When a trajectory generated with the model is entered, both functions are able to verify this, while not finding a valid model with the random trajectory. The functions will be discussed in more detail in the next two sections. As a sidenote, during the writing of this report, the code was run dozens of times, each time with a different random realization. They all gave the correct result, lending credibility to this implementation.

```
fprintf("Given model:\n")
fprintf("\tTrajectory from reference model given: %u (1=trajectory from
this model;0=not a trajectory from this model)\n", is1)
fprintf("\tRandom noise trajectory given: %u (1=trajectory from this
model;0=not a trajectory from this model)\n", is2)
fprintf("\nNo given model:\n")
fprintf("\tTrajectory from reference model given:\n")
fprintf("\t\tIs model output? %u (1=yes;0=no)\n", is21)
fprintf("\t\tWas a good LTI model found for it? %u (1=yes;0=no)\n",
goodModelFound1)
fprintf("\tRandom noise trajectory given:\n")
fprintf("\t\tIs model output? %u (1=yes;0=no)\n", is22)
fprintf("\t\tWas a good LTI model found for it? %u (1=yes;0=no)\n",
goodModelFound2)
```

```
Given model:

      Trajectory from reference model given: 1 (1=trajectory from this
model;0=not a trajectory from this model)

      Random noise trajectory given: 0 (1=trajectory from this model;0=not
a trajectory from this model)


No given model:

      Trajectory from reference model given:

            Is model output? 1 (1=yes;0=no)

            Was a good LTI model found for it? 1 (1=yes;0=no)

      Random noise trajectory given:

            Is model output? 0 (1=yes;0=no)

            Was a good LTI model found for it? 0 (1=yes;0=no)
```

Question 1

This function takes in the reference model, a trajectory to check, and the relevant sizes. It starts off by estimating the initial state by solving a system of equations. Using that initial state, the input part of the given trajectory is used to simulate the output of the reference model. This generated output

is then compared to the output part of the given trajectory. The function returns true if the Frobenius norm of the difference is zero within tolerance.

```
function [isSame] = verifyModelOrigin(ref, traj, nu)
    ny = size(traj, 1)-nu;
    u = traj(1:nu, :);
    y = traj(nu+1:end, :);
    A = ref.A; B = ref.B; C = ref.C; D = ref.D;
    n = size(A, 1);


    % Estimate the initial state with a regression based on the
    % first max(20, n) trajectory points. Normally, as long as all states
    % are observable only at most n points are needed, but more are used
    % here for robustness.
    Ys = [];
    Us = [];
    O = [];
    F = [];
    nn = max(20, n);
    for t=1:max(20, n)
        Ys = [Ys;y(:, t)];
        Us = [Us; u(:, t)];
        O = [O;C*A^(t-1)];
        tmp = zeros(ny, nu*(nn-t));
        tmp = [D tmp];
        for i=t-1:-1:1
            CA = C*A^(t-i-1);
            tmp = [CA*B tmp];
        end
        F = [F;tmp];
    end
    x1 = O\(Ys-F*Us);
    yref = lsim(ref, u, [], x1).'; % generate the model trajectory for the
given input
    isSame = norm(y-yref, 'fro') < 1e-9; % see if the outputs are
equivalent with slight tolerance
end
```

Question 2

With the knowledge of my master thesis, my most familiar way to solve this part is using a subspace method, since this can be done linearly and answers the question of existence of a limited complexity LTI model early in the algorithm. This function is my implementation of a MOESP method for deterministic systems, found in Katayama, T. (2005). Subspace methods for system identification. Springer. pp. 157-160.

To leave the code clean and readable, some notes about the parts that speak less for itself will be put here for reference. The algorithm starts off by calculating the block Hankel matrix of the trajectory, in this case with the inputs and outputs separated. On this matrix an LQ decomposition is done to more easily split the influences of the input and output:

$$\begin{pmatrix} U_{1|i-1} \\ Y_{1|i-1} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}$$

$$\begin{cases} U_{1|i-1} = L_{11} Q_1^T \\ Y_{1|i-1} = L_{21} Q_1^T + L_{22} Q_2^T \end{cases}$$

Using $Y_{1|i-1} = \Gamma_i X_0 + \psi_i U_{1|i-1}$ it can be found using the orthogonality of Q_1 and Q_2 that $\Gamma_i X_0 Q_2 = L_{22}$. It follows that the rank of L_{22} is n since $X_0 Q_2$ has a full row rank of n and Γ_i has rank n. (paraphrased from the book)

Using the SVD of $L_{22} = U_1 S_1 V_1^T$ we can find n as the number of singular values significantly different from zero. If this value is bigger than the allowed complexity, the algorithm can terminate and the first subquestion (the existence of a valid LTI model) is solved.

This SVD is then used to get the extended observability matrix from the earlier equations. It is chosen to be $U_1 S_1^{0.5}$ for symmetry reasons. This choice is often made in literature but is not unique. Any invertible nxn matrix can be right multiplied to give the extended observability matrix of an equivalent system.

C and A are easily found from this extended observability matrix: C is the first block and can be extracted directly. A is the solution of the system of equations formed by noticing that each block is the block above it times A: Gammai(1:((i-1)*ny), :)*A=Gammai(ny+1:end, :).

B and D can be found by constructing relation (eq 6.44, p.159) in the book: L*[D;B]=M With L constructed from the columns of U2T and the extended observability matrix, and M by taking U2T*(L21/L11) and stacking the block columns on top of each other. As also stated in the book, as long as i is taken bigger than n, this system gives a unique solution for B and D.

The found system ss(A, B, C, D, Ts) is then tested using the function of Question 1 to see if the found model matches the trajectory. If everything is implemented correctly this should always succeed, but it is done here as an extra sanity check.

```
function [isST, model, goodModelFound] = isSystemTrajectory(traj, nu,
maxOrder, Ts)
    u = traj(1:nu, :);
    y = traj(nu+1:end, :);
    ny = size(y, 1);

    % i will be the maximal number of singular values, so this value is
    % taken to be able to verify if the needed complexity is bounded
    i = 2*maxOrder;
    %i+j-1=N=>j=N-i+1
    j = size(u, 2)-i+1;
    %%
    % Create an LQ decomposition of the trajectory (based on the algorithm
    % given in the book) using respectively $U_{1|i-1}$ and $Y_{1|i-1}$
    U1im1 = blockHankel(u, i, j);
    Y1im1 = blockHankel(y, i, j);
    [Q, L] = qr([U1im1;Y1im1].', 0);
    Q = Q.'; L = L.';
    L11 = L(1:size(U1im1, 1), 1:size(U1im1, 1));
    L21 = L(size(U1im1, 1)+1:end, 1:size(U1im1, 1));
    L22 = L(size(U1im1, 1)+1:end, size(U1im1, 1)+1:end);

    %%
    % After some playing around with the equations it turns out that the
    % rank of $L_{22}$ is also the rank of the extended observability
    % matrix, and as such the amount of singular values of $L_{22}$
    % different from zero is the order the LTI model needs to explain the
    % given trajectory
    [U, S, ~] = svd(L22);
    n = rank(S);
    if n > maxOrder % Exit if the needed order is bigger than allowed
```

```
            isST = false;
            model = [];
            goodModelFound = false;
            return;
        end
        U1 = U(:, 1:n);
        U2 = U(:, n+1:end);
        U2T = U2.';
        S1 = S(1:n, 1:n);

        Gammai = U1*sqrt(S1);
        % A and C are easily found from the extended observability matrix:
        C = Gammai(1:ny, :);
        A = Gammai(1:((i-1)*ny), :)\Gammai(ny+1:end, :);

        M = U2T*(L21/L11);

        Ms = [];
        L1 = [];
        L2 = [];
        for Lidx = 1:i
            Lk = U2T(:, (Lidx-1)*ny+(1:ny));
            Mk = M(:, (Lidx-1)*nu+(1:nu));
            L1 = [L1;Lk];
            if Lidx ~= i
                L2 = [L2;U2T(:, (Lidx*ny+1):end)*Gammai(1:((i-Lidx)*ny), :)];
            else
                L2 = [L2;zeros(ny*i-n, n)];
            end
            Ms = [Ms;Mk];
        end
        DB = [L1 L2]\Ms;
        D = DB(1:ny, :);
        B = DB(ny+1:end, :);
        model = ss(A, B, C, D, Ts);
        isST = true;
        goodModelFound = verifyModelOrigin(model, traj, nu);
    end
```

Addition 1: estimateOrder function

This function takes a trajectory of sufficient length and returns the model order *n* and the number of inputs *m*. It makes use of the fact that the column rank of the block Hankel matrix of the trajectory with L block rows is mL+n. By making the matrix as square as possible I try to take care of some underlying assumptions that are made, like for example that the matrix needs at least mL+n columns, which is difficult to check without knowing *m* and *n*. By taking the matrix square, the minimum of the number of rows and the number of columns (the maximal rank of the matrix) is maximized. The rank of a block Hankel matrix with L-1 block rows and one with L block rows is found. This leads to a system of equations with two equations and two unknowns, which has a unique solution. How sensical this solution is has to be ascertained by the caller of the function.

```
function [n, m] = estimateOrder(w)
    [k, N] = size(w);
    % j=k*Lmax and Lmax+j=N => N-Lmax=k*Lmax => Lmax=N/(k+1)
    L = floor(N/(k+1));
    H = blockHankel(w, L-1, k*(L-1));
    c1 = rank(H);
```

```
    H = blockHankel(w, L, k*L);
    c2 = rank(H);
    % c = m*L+n => c1=m*(L-1)+n and c2=m*L+n
    m = c2-c1;
    n = c2-m*L;
end
```

Addition 2: isSystemTrajectory2 function

This function serves the same purpose as the subspace method implemented earlier, but doesn't ask for the number of inputs. Since the exact model order *n* is needed to get a useful R matrix the number of inputs can be found together with it, making it also useful in the cases where the number of inputs is not known a priori. The found values of *n* and *m* are checked to see if they lead to a valid system. The number of inputs needs to be less than the size of a trajectory vector-1 to leave space for at least one output and at least zero. The model order is bounded by the maximal order passed to the function and needs to be bigger than zero.

This implementation can check the existence of a model in any valid case, but only gives a model in the SISO case. It should be possible to extend the logic to MIMO, but I didn't check due to time constraints.

```
function [isST, model, goodModelFound] = isSystemTrajectory2(w, nmax, Ts)
    [k, N] = size(w);
    [n, m] = estimateOrder(w);
    if m<0 || m>k-1 || n>nmax || n<1
        isST = false;
        model = [];
        goodModelFound = false;
        return;
    end
    isST = true;
    if m~=1 || k~=2
        fprintf("Valid MIMO LTI system trajectory, but this function only
supports model calculation for SISO.\n")
        model = [];
        goodModelFound = false;
        return;
    end
    H = blockHankel(w, n+1, N-n-1);
    R = null(H')';
    Ry = fliplr(R(:, 1:2:end));
    Ru = fliplr(R(:, 2:2:end));
    model = ss(tf(-Ry, Ru, Ts));
    goodModelFound = verifyModelOrigin(model, w, 1);
end
```

Extended results

First the order estimation function is tested. This is done using three test trajectories: one of the reference system (n=1, m=1), one random noise trajectory (one input and one output) and one trajectory from a higher order model (n=10, m=3):

```
[n1, m1] = estimateOrder(traj1);
[n2, m2] = estimateOrder(traj2);

ref2 = drss(10, 5, 3);
```

```
ref2.Ts = 1/50;
u3 = randn(3, N);
y3 = lsim(ref2, u3, [], randn(10, 1)*15).';
[n3, m3] = estimateOrder([u3;y3]);
[is23, model3, goodModelFound3] = isSystemTrajectory2(traj1, 20, 1/50);
fprintf("\nOrder estimation:\n")
fprintf("\tTrajectory from reference model given: n=%u m=%u\n", n1, m1)
fprintf("\tRandom noise trajectory given: n=%u m=%u\n", n2, m2)
fprintf("\tTrajectory from higher order model (n=10, m=3) given: n=%u
m=%u\n", n3, m3)
```

```
Order estimation:

      Trajectory from reference model given: n=5 m=1

      Random noise trajectory given: n=0 m=2

      Trajectory from higher order model (n=10, m=3) given: n=10 m=3
```

These results look correct. Note that in the case of the random noise trajectory a nonsensical answer with a model order of zero and number of outputs the size of a trajectory vector was given.

After this, the kernel method function was tested the same way as the subspace method:

```
[is23, model3, goodModelFound3] = isSystemTrajectory2(traj1, 20, 1/50);
[is24, model4, goodModelFound4] = isSystemTrajectory2(traj2, 20, 1/50);
fprintf("\nNo given model (with kernel method):\n")
fprintf("\tTrajectory from reference model given:\n")
fprintf("\t\tIs model output? %u (1=yes;0=no)\n", is23)
fprintf("\t\tWas a good LTI model found for it? %u (1=yes;0=no)\n",
goodModelFound3)
fprintf("\tRandom noise trajectory given:\n")
fprintf("\t\tIs model output? %u (1=yes;0=no)\n", is24)
fprintf("\t\tWas a good LTI model found for it? %u (1=yes;0=no)\n",
goodModelFound4)
```

```
No given model (with kernel method):
      Trajectory from reference model given:

            Is model output? 1 (1=yes;0=no)

            Was a good LTI model found for it? 1 (1=yes;0=no)

      Random noise trajectory given:

            Is model output? 0 (1=yes;0=no)

            Was a good LTI model found for it? 0 (1=yes;0=no)
```

These also look correct. Finally a comparison in runtime was made between the subspace and kernel method. As it turns out, the calculation of the order and number of inputs is by far the slowest part, so that was profiled by itself to compare. The runtime is calculated by checking how long it takes to run 50 times and dividing it by 50:

```
its = 50;
tic
for i=1:its
    [is21, model1, goodModelFound1] = isSystemTrajectory(traj1, nu, 20,
1/50);
end
t1 = toc/its;
tic
```

```
for i=1:its
    [is24, model4, goodModelFound4] = isSystemTrajectory2(traj2, 20, 1/50);
end
t2 = toc/its;
tic
for i=1:its
    [nn, m] = estimateOrder(traj1);
end
t3 = toc/its;

fprintf("\nTime comparison: subspace method took on average %.3fs vs %.3f
for the kernel based method\n", t1, t2)
fprintf("Time comparison: Explicitly estimating exact order and number of
inputs takes on average %.3fs\n", t3)
```

```
Time comparison: subspace method took on average 0.013s vs 0.099 for the
kernel based method

Time comparison: Explicitly estimating exact order and number of inputs
takes on average 0.096s
```

Notice that the subspace method takes significantly less time on average compared to the kernel based method, but this is a slightly unfair comparison. The subspace method gets passed the number of inputs and as such doesn't need the slow function to estimate it. Taking that into account the kernel based method is faster in the case of unknown number of inputs, since the subspace method would take on average 0.109s vs 0.099s.

The order estimation function is so slow due to the need to construct and calculate the rank of two relatively large matrices. Some possible improvements are:

- Use relevant information: the number of inputs is bounded and so is the maximal allowed complexity. Taking these into account lowers the size of the matrices and as such improves the runtime considerably.
- Use modular arithmetic to separate *m* and *n*: c=mL+n => n=c mod L and m=(c-n)/L. This works as long as L is larger than the maximal allowed order, which should be the case.

These improvements were not explored due to time limitations.

Identification of Dynamical Systems: Homework 1&2

Arno Hemelhof

October, 2021

1 Homework 1

1.1 Problem Statement

Given a time-series

$$w_d = \begin{bmatrix} u_d \\ y_d \end{bmatrix} = \left(\begin{bmatrix} u_d(1) \\ y_d(1) \end{bmatrix}, ..., \begin{bmatrix} u_d(T) \\ y_d(T) \end{bmatrix} \right)$$

check if it's a trajectory of an LTI system (with bounded complexity). This assignment is made up of three parts:

1. Formalize what it means for "w_d to be a trajectory of an LTI system"
2. Write an algorithm to perform this check
3. Construct a simulation example to test the algorithm

1.2 Implementation

1.2.1 Formalizing the Requirement

Since LTI systems (of order n) can be represented by a state space model of order n with an initial state X_0, this representation will be used throughout this solution.

The required statement can then be formalized as:

A time-series w_d is a trajectory of an LTI system of bounded complexity (order $\leq n$) if there exists a state space model (A, B, C, D) of order $\leq n$ and an initial state X_0 such that

$$y_d = O_T \cdot X_0 + \Psi_T \cdot u_d \tag{1}$$

Where

$$O_k = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{k-1} \end{bmatrix}, \Psi_k = \begin{bmatrix} D & 0 & \dots & 0 \\ CB & D & \dots & 0 \\ \vdots & & & \\ CA^{k-2}B & CA^{k-3}B & \dots & D \end{bmatrix}$$

and u_d and y_d are the time-series of the input and output (contained in w_d) stacked as a single column-vector each, e.g.

$$u_d = \begin{bmatrix} u_d(1) \\ u_d(2) \\ \vdots \\ u_d(T) \end{bmatrix}$$

with $u_d(i)$ being a column-vector itself of all the inputs at time i.

To check this condition, I will work in two parts:

1. Construct a minimal realisation of a system that should satisfy the condition
2. Check if, given this realisation, w_d is a trajectory of the resulting model

Since the last item is the easiest, I will start with that one.

1.2.2 Check If System Is a Solution

If the state space representation of an LTI is given, check if an initial state exists such that Equation 1 is satisfied. In the most simple case, this state is found by solving following equation:

$$y_d(1) = C \cdot X_0 + D \cdot u_d(1)$$

This only gives a unique solution if $rank(C) = n$. To work for the general case, Equation 1 restricted to n inputs and outputs is used to solve for X_0. More terms than this are not needed since $\text{rank}(O_n) = n$ for an observable system. If the system is not observable, no state can be uniquely determined from the inputs and outputs. The solution is then checked by simulating the system with this initial state and comparing the outputs:

Listing 1: Solution Check Implementation

```
function X0 = EstimateInitialState(us, ys, H)

idx_max = min(size(H.A, 1), size(us, 1));

y_full = [];
O_full = [];
for idx = 1:idx_max
```

```
    U = us.';
    U = reshape(U(:, 1:idx), [], 1);
    PSI_i = zeros(size(H.D, 1), idx*size(H.D, 2));
    tmp = H.C;
    for i = idx-1:-1:1
        start = (i-1)*size(H.D, 2)+1;
        PSI_i(:, start:(start+size(H.D, 2)-1)) = tmp*H.B;
        tmp = tmp * H.A;
    end
    PSI_i(:, ((idx-1)*size(H.D, 2)+1):end) = H.D;
    yi = ys(idx, :).' - PSI_i*U;
    y_full = [y_full ; yi];
    O_full = [O_full ; tmp];
end
X0 = O_full\y_full;

end

function result = IsTrajectoryOfSystem(us, ys, H)

X0 = EstimateInitialState(us, ys, H);

y_new = lsim(H, us, [], X0);

mse = mean((ys-y_new).^2, 'all');
result = (mse/mean(abs(ys), 'all') < 1e-8);

end
```

1.2.3 Finding a Minimal System

For this part, I had to do some research online and got pointed to *Subspace Methods for System Identification* by Tohru Katayama [1]. This part is mostly an implementation of the MOESP Method outlined in that source.

First, some notation should be introduced:

$$U_{1|k} = \begin{bmatrix} u_d(1) & u_d(2) & \dots & u_d(N) \\ u_d(2) & u_d(3) & \dots & u_d(N+1) \\ \vdots & & & \\ u_d(k) & u_d(k+1) & \dots & u_d(k+N-1) \end{bmatrix}$$

This is a block Hankel matrix based on the inputs. A similar notation is used for the outputs: $Y_{1|k}$. k should be chosen to be strictly greater than the order of the system (dimension of the state vector), which in practice means that it

should be greater than the upper bound of the expected order. N should be chosen large enough. To utilise all data, I chose $N = T - k + 1$.

These two block Hankel matrices can be combined into a Data Matrix:

$$W_{1|k} = \begin{bmatrix} U_{1|k} \\ Y_{1|k} \end{bmatrix}$$

The condition of Equation 1 can be rewritten by splitting it up into chunks:

$$Y_{1|k} = O_k \cdot X + \Psi_k \cdot U_{1|k} \tag{2}$$

where $X = \begin{bmatrix} X(1) & X(2) & \dots & X(N) \end{bmatrix}$ are the state vectors in function of time.

The goal of rewriting it like this is to be able to isolate the term with O_k (containing information on A and C), and the term with Ψ_k (containing information on A, B, C and D). This will help with that using following observation:

If $W_{1|k}$ is decomposed into a product of a lower triangular matrix and an orthonormal matrix (using LQ decomposition), we have:

$$W_{1|k} = \begin{bmatrix} U_{1|k} \\ Y_{1|k} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \cdot \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix}$$

$$U_{1|k} = L_{11} \cdot Q_1^T \tag{3a}$$

$$Y_{1|k} = L_{21} \cdot Q_1^T + L_{22} \cdot Q_2^T \tag{3b}$$

Filling Equation 3a and Equation 3b into Equation 2, we obtain

$$L_{21}Q_1^T + L_{22}Q_2^T = O_k X + \Psi_k L_{11} Q_1^T \tag{4}$$

Since Q_1 and Q_2 are orthonormal, the terms can be isolated by post-multiplying with the correct matrix. For example:

$$L_{22} = O_k X Q_2$$

With the assumption of this method that the system is observable, the rank of O_k is the real order of the system n as is the rank of XQ_2. This means the rank of their product L_{22} is also n. Doing the singular value decomposition on L_{22} and retaining only the non-zero part gives

$$L_{22} = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} = U_1 \Sigma_1 V_1^T$$

O_k is chosen to be $U_1\Sigma_1^{1/2}$. This choice is not unique since post-multiplying O_k with any invertible $n \times n$ matrix will result in a model for the same system, but with another basis for the state.

Looking at the definition of O_k as used in Equation 1, the C matrix can be extracted as the top left $p \times n$ block with p being the number of outputs.

Since C is in general not left-invertibe, this can't be directly used to calculate A using a different block, but A can be obtained by solving following system of equations:

$$O_{k-1}A = O_k(p+1 : \text{end}, :)$$

Since $\text{rank}(O_k) = n$ for any $k \geq n$ and k was chosen strictly greater than n, O_{k-1} is left-invertible and, as such, this equation has a unique solution.

Pre-multiplying Equation 4 by U_2^T isolates the Ψ_k term since U_1 and U_2 are orthonormal:

$$\begin{aligned} U_2^T L_{21} Q_1^T + U_2^T L_{22} Q_2^T &= U_2^T O_k X + U_2^T \Psi_k L_{11} Q_1^T \\ \Rightarrow\ U_2^T L_{21} Q_1^T + U_2^T U_1 \Sigma_1 V_1^T Q_2^T &= U_2^T U_1 \Sigma_1^{1/2} X + U_2^T \Psi_k L_{11} Q_1^T \\ \Rightarrow\ U_2^T L_{21} Q_1^T &= U_2^T \Psi_k L_{11} Q_1^T \end{aligned}$$

Post-multiplying by $Q_1 L_{11}^{-1}$ then gives:

$$U_2^T L_{21} L_{11}^{-1} = U_2^T \Psi_k$$

Since U_2^T is not left-invertible (it has less rows than columns), this equation can't be solved directly. Writing $U_2^T = [\mathcal{L}_1 \dots \mathcal{L}_k]$ and $U_2^T L_{21} L_{11}^{-1} = [\mathcal{M}_1 \dots \mathcal{M}_k]$, it can be shown that

$$\begin{bmatrix} \mathcal{L}_1 & \bar{\mathcal{L}}_2 O_{k-1} \\ \mathcal{L}_2 & \bar{\mathcal{L}}_3 O_{k-2} \\ \vdots & \\ \mathcal{L}_{k-1} & \bar{\mathcal{L}}_k O_1 \\ \mathcal{L}_k & 0 \end{bmatrix} \begin{bmatrix} D \\ B \end{bmatrix} = \begin{bmatrix} \mathcal{M}_1 \\ \mathcal{M}_2 \\ \vdots \\ \mathcal{M}_{k-1} \\ \mathcal{M}_k \end{bmatrix}$$

with $\bar{\mathcal{L}}_i = [\mathcal{L}_i \dots \mathcal{L}_k]$ where it can be shown that the left matrix has full column rank and, as such, is left-invertible giving a unique solution for B and D given A and C.

Finaly, the initial state can be estimated as done in the previous section to get the minimal realization and intial state that generates the given trajectory. These are than checked similarly to the *IsTrajectoryOfSystem* function.

The code for this part is given below

Listing 2: Implementation for finding the minimal system

```
function [result, H, X0]  = CheckLTI(us, ys, max_order)

% MOESP Method based on Subspace Methods for System Identification by
% Tohru Katayama

% size(us) = num_samples x num_inputs
% size(ys) = num_samples x num_outputs
[T, m] = size(us);
p = size(ys, 2);

k = min(2 * max_order, T - 1); % Must be greater than the needed order
N = T - k + 1;

% Construct Block Hankel Matrix
UY = zeros(k * (m+p), N);
for i = 1:T
    start_u = m*(i-1) + 1;
    start_y = p*(i-1) + k*m + 1;
    for j = max(1, i-k+1):min(i, N)
        UY(start_u + (0:m-1) - m*(j-1), j) = us(i, :).';
        UY(start_y +(0:p-1) - p*(j-1), j) = ys(i, :).';
    end
end

% LQ Decomposition
[Q, L] = qr(UY.', 0);
L = L.';
L11 = L(1:k*m, 1:k*m);
L21 = L((k*m+1):end, 1:k*m);
L22 = L((k*m+1):end, (k*m+1):end);
Q1 = Q(:, 1:(k*m));
Q2 = Q(:, (k*m+1):end);

% Obtaining O_k
[U, S, V] = svd(L22);
n = rank(S);

if n > max_order
    result = false;
    H = [];
    X0 = [];
```

```
    return;
end

U1 = U(:, 1:n);
U2 = U(:, (n+1):end);
V1 = V(:, 1:n);
S = S(1:n, 1:n);

Ok = U1 * sqrt(S);

% Getting A and C from O_k
C = Ok(1:p, :);
A = Ok(1:p*(k-1), :)\Ok((p+1):k*p, :);

% Getting B and D
L_base = U2.';
M_base = L_base * L21/L11;

Ls = zeros(k*(k*p-n), p+n);
M = zeros(k*(k*p-n), m);

for i = 1:k
    start1 = (i-1)*(k*p-n)+1;
    start2 = (i-1)*p+1;
    start3 = (i-1)*m+1;
    Ls(start1:(start1+k*p-n-1), 1:p) = L_base(:, start2:(start2+p
        -1));
    Ls(start1:(start1+k*p-n-1), (p+1):end) = L_base(:, start2+p:end
        )*Ok(1:p*(k-i), :);

    M(start1:(start1+k*p-n-1), :) = M_base(:, start3:(start3+m-1));
end

BD = Ls\M;
D = BD(1:p, :);
B = BD((p+1):end, :);

H = ss(A, B, C, D, -1);
X0 = EstimateInitialState(us, ys, H);

y_new = lsim(H, us, [], X0);
mse = mean((ys-y_new).^2, 'all');
result = (mse/mean(abs(ys), 'all') < 1e-8);

end
```

1.3 Testing

To test the implemented parts, additional code was written to generate LTI systems, generate trajectories for them and apply the algorithms to those. A stable third order system with three inputs and two outputs was made by hand. In addition, a sixth order system with, again, three inputs and two outputs was generated using the *drss* function.

The code and its resulting output are given below

Listing 3: Testing the Implementations

```
clc;
clear;
close all hidden;

% Fix Seed
rng(957342445);

A = [-0.1, 0, 0; 4, -0.5, 0; 7, 8, -0.9];
B = [1, 2; 3, 4; 5, 6];
C = eye(3);
D = 0;

H = ss(A, B, C, D, -1);
rand_sys = drss(6, 3, 2);

u = randn(100, 2);
X0g1 = randn(3, 1);
X0g2 = randn(6, 1);

Y1 = lsim(H, u, [], X0g1);
Y2 = lsim(rand_sys, u, [], X0g2);

bool1 = IsTrajectoryOfSystem(u, Y1, H);
bool2 = IsTrajectoryOfSystem(u, Y1, rand_sys);
bool3 = IsTrajectoryOfSystem(u, Y2, H);
bool4 = IsTrajectoryOfSystem(u, Y2, rand_sys);

snip = ['not ' ''];
fprintf('[u; Y1] is %sa trajectory of H\n', snip(bool1+1))
fprintf('[u; Y1] is %sa trajectory of rand_sys\n', snip(bool2+1))
fprintf('[u; Y2] is %sa trajectory of H\n', snip(bool3+1))
fprintf('[u; Y2] is %sa trajectory of rand_sys\n', snip(bool4+1))

[res1, model1, X01] = CheckLTI(u, Y1, 5);
[res2, model2, X02] = CheckLTI(u, Y2, 5);
[res3, model3, X03] = CheckLTI(u, Y1, 10);
```

```
[res4, model4, X04] = CheckLTI(u, Y2, 10);

fprintf('A system of order <= 5 was %sfound for Y1\n', snip(res1+1)
    );
if res1
    fprintf('\tOrder was %d\n', size(model1.A, 1));
end

fprintf('A system of order <= 5 was %sfound for Y2\n', snip(res2+1)
    );
if res2
    fprintf('\tOrder was %d\n', size(model2.A, 1));
end

fprintf('A system of order <= 10 was %sfound for Y1\n', snip(res3
    +1));
if res3
    fprintf('\tOrder was %d\n', size(model3.A, 1));
end

fprintf('A system of order <= 10 was %sfound for Y2\n', snip(res4
    +1));
if res4
    fprintf('\tOrder was %d\n', size(model4.A, 1));
end
```

Listing 4: Output of the Code

```
[u; Y1] is a trajectory of H
[u; Y1] is not a trajectory of rand_sys
[u; Y2] is not a trajectory of H
[u; Y2] is a trajectory of rand_sys
A system of order <= 5 was found for Y1
        Order was 3
A system of order <= 5 was not found for Y2
A system of order <= 10 was found for Y1
        Order was 3
A system of order <= 10 was found for Y2
        Order was 6
```

This is the expected output.

2 Homework 2

2.1 Introduction

This homework again had a couple of different objectives:

1. The intended solution to the previous homework should be implemented and compared to the solution I used
2. The theory that was seen in class should be used to find the minimal complexity of a system (number of inputs and system order) when given only a trajectory w_d.
3. As a bonus, the basic realisation algorithm can be implemented

Since the work done by my solution of the first goes a lot further than the intended solution, no comparison is done. Instead, the realisation algorithm will be compared to the subspace solution with the realisation algorithm being augmented by a pre-processing step to get the impulse response from the trajectory.

2.2 Intended Solution

It can be shown that $\text{rank}(\mathcal{H}_L(w_d)) = m \cdot L + n$ with m the number of inputs, and n the order of the system. This means that if we have a maximal system order of n_{max}, we know w is the trajectory of an LTI system with bounded complexity by looking at the rank of $\mathcal{H}_{n_{max}+1}(w_d)$.

If the matrix has full row rank $(q \cdot (n_{max} + 1))$, the only option with an order less than or equal to n_{max} is the system without outputs, which is not a valid solution. Any lower rank gives rise to solutions with outputs and orders greater than zero.

The bounded representation of the system can then be obtained by the kernel representation which gives the coefficients of a system of equations of difference equations describing the system.

The implementation of this algorithm is given below:

Listing 5: Implementation of Intended Solution

```
function [result, R] = check_trajectory_2(w, max_n)

% size(w) = num_samples x (num_inputs+num_outputs)

k = max_n + 1;
H = block_hankel(w.', k);

mk_plus_n = rank(H);

if mk_plus_n == size(H, 1)
```

```
    result = false;
    R = [];
    return;
end

n = mod(mk_plus_n, k);

H = block_hankel(w.', n+1);
R = null(H.').';
result = true;

end
```

2.3 Getting the Complexity From a Trajectory

Obtaining the complexity is also done using the observation that $\text{rank}(\mathcal{H}_L(w_d)) = m \cdot L + n$. This time L is chosen greater than n (or assumed to be if enough samples are provided) and such that the block Hankel matrix is as square as possible to help with the accuracy of the rank calculation. We then obtain:

$$n = \text{rank}(\mathcal{H}_L(w_d)) \mod L$$
$$m = \frac{\text{rank}(\mathcal{H}_L(w_d)) - n}{L}$$

The implementation is given below:

Listing 6: Implementation of Complexity Calculation

```
function [m, n] = get_complexity(w)

% size(w) = num_samples x (num_inputs+num_outputs)
% (q*k x T-k+1)
% q*k = T-k+1 => k = (T+1)/(q+1)
% Floored to prevent the number of columns from limiting the rank

[T, q] = size(w);
k = floor((T+1)/(q+1));
H = block_hankel(w.', k);

mk_plus_n = rank(H);
n = mod(mk_plus_n, k);
m = (mk_plus_n-n)/k;

end
```

2.4 State Space System From Trajectory

2.4.1 Impulse Response From Trajectory

Since the impulse response will be used to construct a block Hankel matrix with $n+1$ block-rows, the number of elements of the impulse response are chosen to make this matrix as square as possible, but wider than tall if square is not possible. The addition by one at the end is because we need one extra value of the impulse response since $H(0)$ will be the first one, and will not be used for the block Hankel matrix.

$$\begin{aligned} p \cdot (n+1) &= m \cdot ((t-1) - (n+1) + 1) \\ \Rightarrow t &= \frac{p}{m} \cdot (n+1) \\ \Rightarrow t_{used} &= \left\lceil \frac{p}{m} \cdot (n+1) + 1 \right\rceil \end{aligned}$$

The maximal lag of the system is taken to be equal to the order of the system which is obtained as in the previous exercise. The condition that should then be satisfied is the following:

$$\text{image}\, \mathcal{H}_{l_{max}+t}(w_d) = \mathcal{B}|_{l_{max}+t}$$

It is assumed that the Fundamental Lemma always applies, thus satisfying this condition. We then get the impulse response by finding the coefficients needed to right-multiply with the block Hankel matrix to obtain a trajectory starting with 0 initial conditions for l_{max} samples, and an impulse input for the t samples after. The "future" outputs are left out of this equation by omitting the last $p \cdot t$ rows of the block Hankel matrix $Y = \mathcal{H}_{l_{max}+t}(y_d)$. This omitted part is then right-multiplied with the coefficients after those are obtained to get the impulse response H.

Note that due to the rank-deficiency of the block Hankel matrix, there is no unique matrix of coefficients, but a single solution is enough since the result of the Fundamental Lemma guarantees that output will be the continuation of our constructed "impulse" trajectory.

2.4.2 Realisation Algorithm

The first $p \times m$ block of the impulse response is the D matrix. The rest of the matrices are obtained using the Rank-Revealing Decomposition calculated using the Singular Value Decomposition:

$$\mathcal{H}_{n+1}(H) = U \Sigma V^t = P \cdot L$$

With $\Sigma_n = \Sigma(1:n, 1:n)$, $P = U \cdot \Sigma_n^{1/2}$ and $L = \Sigma_n^{1/2} \cdot V^t$

It can be shown that like this

$$P = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^n \end{bmatrix}$$
$$L = \begin{bmatrix} B & AB & \dots & A^nB \end{bmatrix}$$

C is then the first $p \times n$ block of P and B is the first $n \times m$ block of L. A is obtained using the shift property:

$$P(1 : \text{end} - n, :)A = P(n + 1 : \text{end}, :)$$

2.4.3 Implementation

Listing 7: Implementation of the Realisation ALgorithm

```
function [result, R] = check_trajectory_2(w, max_n)

% size(w) = num_samples x (num_inputs+num_outputs)

k = max_n + 1;
H = block_hankel(w.', k);

mk_plus_n = rank(H);

if mk_plus_n == size(H, 1)
    result = false;
    R = [];
    return;
end

n = mod(mk_plus_n, k);

H = block_hankel(w.', n+1);
R = null(H.').';
result = true;

end
```

2.5 Testing

Following code was used to test these algorithms

Listing 8: Implementation of Testing Code

```
clc;
clear;
close all hidden;

% Fix Seed
rng(957342445);

rand_sys = drss(6, 3, 2);

u = randn(100, 2);
X0g2 = randn(6, 1);

Y1 = lsim(rand_sys, u, [], X0g2);
Y2 = randn(size(Y1));

[m, n] = get_complexity([u Y1])
bool1 = check_trajectory_2([u Y1], 10)
bool2 = check_trajectory_2([u Y2], 10)

sys = RealizeMinimal(u, Y1);
bool3 = IsTrajectoryOfSystem(u, Y1, sys)

T1 = tic;
for i = 1:100
    CheckLTI(u, Y1, 10);
end
T2 = toc(T1);

fprintf('MOESP took %.3fms per run\n', T2*1e3/100);

T3 = tic;
for i = 1:100
    RealizeMinimal(u, Y1);
end
T4 = toc(T3);

fprintf('Basic Realization took %.3fms per run\n', T4*1e3/100);
```

Giving following output:

```
m =

     2


```

```
n =

     6


bool1 =

  logical

   1


bool2 =

  logical

   0


bool3 =

  logical

   1

MOESP took 8.313ms per run
Basic Realization took 13.532ms per run
```

The algorithms seem to work perfectly.

Here we can also see that the MOESP implementation of the previous homework still performs better in runtime (if the initial state estimation is removed to make the comparison fair). This is due to the fact that the new algorithm constructs more block Hankel matrices. For both algorithms those constructions take up most of the time. The MATLAB profiler also gave the insight that about 1/3 of the runtime of the new algorithm could be removed if a (realistic) upper bound for the lag were provided.

References

[1] Tohru Katayama. *Subspace Methods for System Identification*. Communications and Control Engineering. Springer-Verlag, London, 2005.