FRTN10 Exercise 1. Control in Matlab

This exercise is intended to give a basic introduction to Matlab. The main focus will be on the use of Control System Toolbox for control system analysis and design. This toolbox will be used extensively during the upcoming exercises and laboratories in the course. A short Matlab reference guide and a guide to the most commonly used commands from Control System Toolbox can be found at the end of this exercise.

Getting started

Matlab is started by issuing the command

```
> matlab
```

which will bring up a Java-based interface with three different frames showing the current directory, your defined variables, and the Matlab command window. This gives a good overview, but may be slow.

An alternative way to start Matlab is by the command

```
> matlab -nodesktop
```

which will only open up the command window. You can then use the commands `ls` and `whos` to examine the current directory and your defined Matlab variables.

All Matlab commands have a help text, which is displayed by typing

```
>> help <command>
```

Try for example

```
>> help help
```

Use the help command frequently in the following exercises.

Matrices and system representations

Matrices in Matlab are created with the following syntax

```
>> A = [1 2; 3+i 4]

A =

1.0000             2.0000
3.0000 + 1.0000i   4.0000
```

with semi-colons being used to separate the lines of the matrix. The conjugate transpose of a matrix is written as

```
>> A'

ans =

1.0000             3.0000 - 1.0000i
2.0000             4.0000
```

1.1 Consider the following state-space model describing the dynamics of an inverted pendulum

$$\begin{aligned} \frac{dx}{dt} &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} x(t) + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u(t) \\ y(t) &= \begin{pmatrix} 0 & 1 \end{pmatrix} x(t) \end{aligned} \tag{1.1}$$

Enter the system matrices A, B, and C in Matlab and use the command `eig` to determine the poles of the system.

In Control System Toolbox, the basic data structure is the linear time-invariant (LTI) model. There is a number of ways to create, manipulate and analyze models. Some operations are best done on the LTI system, and others directly on the matrices of the model.

1.2 Define an LTI model of the pendulum system with the command `ss`. Use the command `tf` to determine the transfer function of the system.

1.3 Zeros, poles and stationary gain of an LTI model are computed with the commands `zero`, `pole` and `dcgain`, respectively. Use these commands on the inverted pendulum model. Compare with 1.1.

The command `tf` is used to create an LTI model from a transfer function. This is done by specifying the coefficients of the numerator and denominator polynomials, e.g. to specify the transfer function $P(s) = 1/(2s+1)$ you can use

```
>> P = tf(1, [2 1]);
```

Often it is most convenient to first define the variable `s` as

```
>> s = tf('s');
```

and then define the transfer functions in terms of `s`, i.e.,

```
>> P = 1 / (2*s + 1);
```

To include a time delay of 0.5 s in $P(s)$ you can use

```
>> P = 1 / (2*s + 1) * exp(-0.5*s);
```

or alternatively, after having defined `P = 1 / (2*s + 1);` you can set the property `InputDelay`,

```
>> P.InputDelay = 0.5;
```

To display all properties of an LTI model and their respective values, type

```
>> get(P)
```

1.4 Define an LTI model of the continuous-time transfer function

$$P(s) = \frac{1}{s^2 + 0.6s + 1} \cdot e^{-1.5s} \tag{1.2}$$

and use the commands `step`, `nyquist`, and `bode` to plot time and frequency responses of the system. Also use the command `pzmap` to plot the pole-zero map. Is the system stable? Will the closed-loop system be stable if unit gain negative feedback is applied?

State feedback example

Now return to the model of the inverted pendulum (1.1). We want to design a state-feedback controller

$$u(t) = -Lx(t)$$

such that the closed loop system gets the characteristic equation

$$s^2 + 1.4s + 1 = 0$$

1.5 Is the system controllable? Use the command `ctrb` to compute the controllability matrix. Then use the command `place` to determine the state-feedback vector L.

Connecting systems

LTI systems can be interconnected in a number of ways. For example, you may add and multiply systems (or constants) to achieve parallel and series connections, respectively. Assume that the system (1.2) *without* time delay is controlled by a PD controller, $C(s) = K(1 + sT_d)$, with $K = 0.5$ and $T_d = 4$, according to the standard block diagram to the left in Figure 1.1:

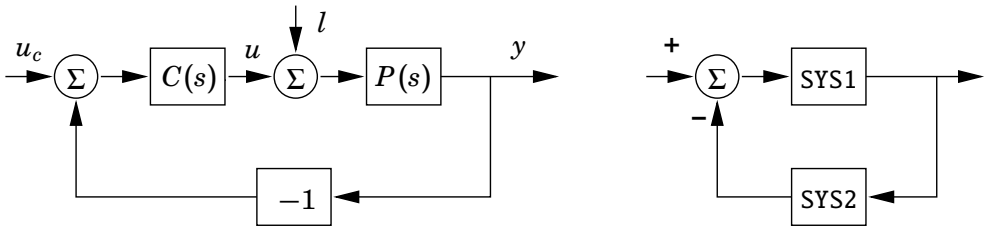

Figure 1.1 A closed-loop control system.

1.6 Define an LTI model of the controller, $C(s)$. Compute the amplitude margin and phase margin for the loop transfer function. Use the Matlab command `margin`.

To obtain the closed loop transfer function for a feedback interconnection of systems it is best to use to command `feedback`. To obtain the transfer function in the block diagram to the right in Figure 1.1 you write `feedback(SYS1,SYS2)`. Note the sign conventions. Using `feedback` is more numerically stable than just writing `SYS1/(1 + SYS1*SYS2)`.

To find the transfer function from the set point u_c to the output y for the system in the left of Figure 1.1, you identify that `SYS1` corresponds to $P(s)C(s)$ and `SYS2` corresponds to 1.

1.7 Compute the transfer function for the closed-loop system, both using the `feedback` command and by direct computation (use `minreal` to simplify) according to the formula

$$G_{cl}(s) = \frac{P(s)C(s)}{1 + P(s)C(s)}$$

1.8 Plot the step response of the closed-loop system. What is the stationary gain from u_c to y?

1.9 The systems you have seen so far have been SISO (single-input, single-output) systems. In this course we will also work with MIMO (multiple-input, multiple-output) systems. This problem gives you an example of a MIMO system.

A rough model for the pitch dynamics of a JAS 39 Gripen is given by

$$\begin{aligned} \dot{x} &= \begin{pmatrix} -1 & 1 & 0 & -1/2 & 0 \\ 4 & -1 & 0 & -25 & 8 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -20 & 0 \\ 0 & 0 & 0 & 0 & -20 \end{pmatrix} x + \begin{pmatrix} 0 & 0 \\ 3/2 & 1/2 \\ 0 & 0 \\ 20 & 0 \\ 0 & 20 \end{pmatrix} u \\ y &= \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} x \end{aligned} \tag{1.3}$$

Enter this system into Matlab using the `ss` command and determine if the system is stable/controllable/observable? What is your conclusion from this analysis?

What is the transfer function from the second input (the elevator rudder command) to the first output (the pitch rate)?

1.10 In Matlab, a transfer function can be represented either in `tf` form, corresponding to $G_1(s)$ and $G_3(s)$ below, or in `zpk` form, corresponding to $G_2(s)$ and $G_4(s)$.

Calculate the poles of the following systems

$$G_1(s) = \frac{1}{s^3 + 3s^2 + 3s + 1}$$

$$G_2(s) = \frac{1}{(s+1)^3}$$

During calculations, numerical round-off errors can arise, which can lead to changed dynamics of the system. Calculate the poles of the following systems where one coefficient has been modified.

$$G_3(s) = \frac{1}{s^3 + 2.99s^2 + 3s + 1}$$

$$G_4(s) = \frac{1}{(s+0.99)^3}$$

In view of your results, discuss which format is numerically better.

1.11 (*) Consider the following state-space model of a system

$$\begin{aligned} \frac{dx}{dt} &= \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} x(t) + \begin{pmatrix} 1 \\ 2 \end{pmatrix} u(t) \\ y(t) &= \begin{pmatrix} 3 & 4 \end{pmatrix} x(t) \end{aligned} \tag{1.4}$$

Is the system observable? Is the system controllable? Motivate your answer by calculations, but also with an insight how the system behaves.

1.12 **(*)** Given a mass-spring system in state-space form without a damper with $m = 0.5$ kg and $k = 10$ N/m, compute the transfer function of the system using the commands `ss` and `zpk`. Design a PID controller such that the closed-loop system gets the characteristic equation

$$s^3 + s^2(2\zeta\omega + \omega) + s(\omega^2 + 2\zeta\omega^2) + \omega^3 = 0$$

$$\begin{aligned} \frac{dx}{dt} &= \begin{pmatrix} 0 & 1 \\ -k/m & 0 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ 1/m \end{pmatrix} u(t) \\ y(t) &= \begin{pmatrix} 1 & 0 \end{pmatrix} x(t) \end{aligned} \tag{1.5}$$

As design parameters use $\omega = 1$ and $\zeta = 0.4$ as start values. Then try different values of ω and ζ, such that during a step response, the settling time is less than 1.0 seconds and the maximum overshot is less than 15%. The settling time is defined as the minimum time T such that $1-p < y(t) < 1+p$ for all $t > T$ where y is the step response and p can be 5%.

1.13 **(*)** Given a transfer function for the following system

$$P(s) = \frac{3 - s}{(s + 1)(s + 2)}$$

Compute a state-space realization using the command `ssdata`. Is the system controllable? Design a state feedback controller, such that the closed-loop system gets the characteristic equation

$$s^2 + 5.6s + 16 = 0$$

Simulate a step response of the closed-loop system. What is the static gain? What is the system called when the step response starts "in the wrong direction"? How can we directly see this property in the transfer function of the process?

Quick Matlab Reference

Some Basic Commands

Note: the command syntax is case-sensitive!

help <command>	display the Matlab help for <command>.
who	lists all of the variables in your matlab workspace.
whos	list the variables and describes their matrix size.
clear	deletes all matrices from active workspace.
clear x	deletes the matrix x from active workspace.
save	saves all the matrices defined in the current session into the file matlab.mat.
load	loads contents of matlab.mat into current workspace.
save filename	saves the contents of workspace into filename.mat
save filename x y z	saves the matrices x, y and z into the file titled filename.mat.
load filename	loads the contents of filename into current workspace; the file can be a binary (.mat) file or an ASCII file.
!	the ! preceding any unix command causes the unix command to be executed from matlab.

Matrix commands

[1 2; 3 4]	create the matrix $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.
zeros(n)	creates an n × n matrix whose elements are zero.
zeros(m,n)	creates an m-row, n-column matrix of zeros.
ones(n)	creates an n × n square matrix whose elements are 1's
ones(m,n)	creates an m × n matrix whose elements are 1's.
ones(A)	creates an m × n matrix of 1's, where m and n are based on the size of an existing matrix, A.
zeros(A)	creates an m × n matrix of 0's, where m and n are based on the size of the existing matrix, A.
eye(n)	creates the n × n identity matrix with 1's on the diagonal.
A'	(complex conjugate) transpose of A
diag(V)	creates a matrix with the elements of V on the diagonal.
blkdiag(A,B,C)	creates block matrix with the matrices A, B, and C on the diagonal

Plotting commands

plot(x,y)	creates an Cartesian plot of the vectors x & y.
stairs(x,y)	creates a stairstep of the vectors x & y.
semilogx(x,y)	plots log(x) vs y.
semilogy(x,y)	plots x vs log(y)
loglog(x,y)	plots log(x) vs log(y).
grid	creates a grid on the graphics plot.
title('text')	places a title at top of graphics plot.
xlabel('text')	writes 'text' beneath the x-axis of a plot.
ylabel('text')	writes 'text' beside the y-axis of a plot.
gtext('text')	writes text according to placement of mouse
hold on	maintains the current plot in the graphics window while executing subsequent plotting commands.
hold off	turns off the 'hold on' option.
print filename -dps	writes the contents of current graphics to 'filename' in postscript format.

Misc. commands

length(x)	returns the number elements in a vector.
size(x)	returns the size m(rows) and n(columns) of matrix x.
rand	returns a random number between 0 and 1.
randn	returns a random number selected from a normal distribution with a mean of 0 and variance of 1.
rand(A)	returns a matrix of size A of random numbers.
fliplr(x)	reverses the order of a vector. If x is a matrix, this reverse the order of the columns in the matrix.
flipud(x)	reverses the order of a matrix in the sense of exchanging or reversing the order of the matrix rows. This will not reverse a row vector!
reshape(A,m,n)	reshapes the matrix A into an m $\times$ n matrix from element (1,1) working column-wise.
squeeze(A)	remove empty dimensions from A
A.x	access element x in the struct A

Some useful functions from Control System Toolbox

Do help <*function*> to find possible input and output arguments.

```
Creation and conversion of LTI models.
ss            - Create or convert to a state-space model.
tf            - Create or convert to a transfer function model.
zpk           - Create or convert to a zero/pole/gain model.
ssdata etc.   - Extract data from an LTI model.
set           - Set/modify properties of LTI models.
get           - Access values of LTI model properties.
minreal       - Minimal realization and pole/zero cancellation.
ss2ss         - State coordinate transformation.
canon         - State-space canonical forms.
ctrlpref      - Open GUI for setting Control System Toolbox Preferences.

Model dynamics.
pole          - System poles.
zero          - Zeros and gain of SISO system.
tzero         - Invariant zeros of MIMO system.
pzmap         - Pole-zero map.
covar         - Covariance of response to white noise.

Time response.
step          - Step response.
impulse       - Impulse response.
initial       - Response of state-space system with given initial state.
lsim          - Response to arbitrary inputs.
ltiview       - Response analysis GUI.

Frequency response.
bode          - Bode plot of the frequency response.
margin        - Bode plot with phase and gain margins.
sigma         - Singular value plot.
nyquist       - Nyquist plot.
nichols       - Nichols plot.
dcgain        - Steady state (DC) gain.

System interconnections.
+ and -       - Add and subtract systems (parallel connection).
*             - Multiplication of systems (series connection).
/ and \       - Division of systems (right and left, respectively).
inv           - Inverse of a system.
[ ]           - Horizontal/vertical concatenation of systems.
feedback      - Feedback connection of two systems.

Classical design tools.
rlocus        - Root locus.
place, acker  - Pole placement (state feedback or estimator).
estim         - Form estimator given estimator gain.
reg           - Form regulator given state-feedback and estimator gains.

LQG design tools.
lqr           - Linear-quadratic (LQ) state-feedback regulator.
lqry          - LQ regulator with output weighting.
lqe           - LQ estimator.
kalman        - Kalman estimator.
lqgreg        - Form LQG regulator given LQ gain and Kalman estimator.

Matrix equation solvers.
lyap          - Solve continuous Lyapunov equation.
care          - Solve continuous algebraic Riccati equation.
```

Solutions to Exercise 1. Control in Matlab

1.1

```
>> A = [0 1; 1 0];
>> B = [1 0]';
>> C = [0 1];
>> D = 0;
>> eig(A)

ans =

-1
1
```

1.2

```
>> sys = ss(A,B,C,D);
>> tf(sys)

Transfer function:
1
-------
s^2 - 1
```

1.3

```
>> zero(sys)

ans =

Empty matrix: 0-by-1

>> pole(sys)

ans =

-1
1

>> dcgain(sys)

ans =

-1
```

1.4

```
>> s = tf('s');
>> P = 1/(s^2+0.6*s+1)

Transfer function:
1
---------------
s^2 + 0.6 s + 1

>> P.InputDelay = 1.5

Transfer function:
1
exp(-1.5*s) * ---------------
s^2 + 0.6 s + 1
```

```
>> bode(P)
>> grid
>> nyquist(P)
>> pzmap(P)
>> step(P)
```

As seen in the pole-zero map, the *open-loop* system is stable, as also indicated by the step response. The Bode and Nyquist plots show that the *closed-loop* system will be unstable.

1.5

```
>> Wc = ctrb(A,B);
>> rank(Wc)

ans =

2
```

Since the controllability matrix has full rank, the system is controllable.

```
>> p=[1 1.4 1];
>> L=place(A,B,roots(p))

L =

1.4000    2.0000
```

1.6

```
P = 1/(s^2+0.6*s+1);
>> C = 0.5*(1+4*s);
>> margin(C*P)
```

The amplitude margin is infinite, whereas the phase margin is 101°.

1.7

```
>> CLSYS = feedback(C*P,1)

Transfer function:
2 s + 0.5
-----------------
s^2 + 2.6 s + 1.5

>> CLSYS = minreal(C*P/(1+C*P))

Transfer function:
2 s + 0.5
-----------------
s^2 + 2.6 s + 1.5
```

1.8

```
>> step(CLSYS)
>> dcgain(CLSYS)

ans =
0.3333
```

1.9

```
>> A=[-1 1 0 -1/2 0; 4 -1 0 -25 8; 0 1 0 0 0; 0 0 0 -20 0; 0 0 0 0 -20];
>> B=[0 0; 3/2 1/2; 0 0; 20 0; 0 20];
>> C=[0 1 0 0 0; 0 0 1 0 0];
>> pitch_dynamics=ss(A,B,C,[0 0; 0 0]);
>> pole(pitch_dynamics)
ans =
```

```
0
1.0000
-3.0000
-20.0000
-20.0000
>> rank(ctrb(pitch_dynamics))
ans =
5
>> rank(obsv(pitch_dynamics))
ans =
4
```

We see that the system is unstable. This means that without some type of control, the plane will crash. Fortunately, the system is controllable, which means that it is possible to stabilise the aircraft with the given actuators. However, since we do not have observability, we need to have some other combination of sensors if we to use feedback from observed states.

To get the transfer function, we use

```
>> G=tf(pitch_dynamics)

Transfer function from input 1 to output...
1.5 s^2 - 468.5 s - 510
#1:  ------------------------
s^3 + 22 s^2 + 37 s - 60

1.5 s^2 - 468.5 s - 510
#2:  ----------------------------
s^4 + 22 s^3 + 37 s^2 - 60 s

Transfer function from input 2 to output...
0.5 s^2 + 170.5 s + 170
#1:  ------------------------
s^3 + 22 s^2 + 37 s - 60

0.5 s^2 + 170.5 s + 170
#2:  ----------------------------
s^4 + 22 s^3 + 37 s^2 - 60 s

>> G(1,2) % To output 1 from input 2 (note the order of indexing)

Transfer function:
0.5 s^2 + 170.5 s + 170
-----------------------
s^3 + 22 s^2 + 37 s - 60
```

1.10

```
>> G1 = 1/(s+1)^3

Transfer function:
1
---------------------
s^3 + 3 s^2 + 3 s + 1

>> pole(G1)
```

```
ans =

-1.0000
-1.0000 + 0.0000i
-1.0000 - 0.0000i

>> G2 = zpk(1/(s+1)^3)

Zero/pole/gain:
1
-------
(s+1)^3

>> pole(G2)

ans =

-1.0000
-1.0000 + 0.0000i
-1.0000 - 0.0000i

>> G3 = 1/(s^3+2.99*s^2+3*s+1);
>> pole(G3)

ans =

-1.0888 + 0.2131i
-1.0888 - 0.2131i
-0.8124

>> G4 = 1/(s+0.99)^3;
>> pole(G4)

ans =

-0.9900 + 0.0000i
-0.9900 - 0.0000i
-0.9900
```

We see that the same small modification in a parameter, causes larger changes in the dynamics when the system is represented as G_3. The transfer function form of G_4 (three poles in the same spot as for G_2), which can be kept with the `zpk` command, is in general better numerically compared to the form in which G_3 is represented (the same form as the command `tf` gives).

1.11

```
>> Wo = obsv(A,C)

ans =

3     4
-3    -4

>> rank(Wo)

ans =

1
```

```
>> rank(ctrb(A,B))

ans =

1
```

Since neither the observability matrix nor the controllability matrix has full rank, the system is neither observable nor controllable. It can be seen directly from the state equations, where we have two states that are completely discoupled from each other and have the same eigenvalue. This means that evolution of the states will look exactly the same for any control signal $u(t)$ (assuming that the initial state is at the origin). Therefore we will never be able to control these states arbitrarily. We will only be able to control them along some controllable subspace. The same goes for the observability.

1.12 The transfer function for the mass-spring system will be

```
>> zpk(ss(A,B,C,D))

Zero/pole/gain:
2
-----------
(s^2  + 20)
```

The transfer function of a PID controller is

$$R = \frac{K(sT_i + 1 + s^2 T_d T_i)}{sT_i}$$

and the closed-loop transfer function is

$$G_{cl}(s) = \frac{R(s)P(s)}{1 + R(s)P(s)} = \frac{2K(s^2 T_d + s + 1/T_i)}{s^3 + s^2 2KT_d + s(20 + 2K) + 2K/T_i}$$

The closed-loop characteristic equation is then

$$s^3 + s^2(2KT_d) + s(20 + 2K) + 2K/T_i = 0$$

Identify the coefficients and solve for K, T_i and T_d as functions of ω and ζ:

$$\begin{aligned} K &= 0.5(\omega^2 + 2\zeta\omega^2 - 20) \\ T_i &= \frac{2K}{\omega^3} \\ T_d &= \frac{2\zeta\omega + \omega}{2K} \end{aligned}$$

The closed-loop system is then

```
>> G_cl = feedback(R*P,1);
>> step(G_cl)
```

The specification is met for many different choices of ω and ζ. One choice can be $\omega = 6$ and $\zeta = 0.7$.

1.13

```
>> s = tf('s');
>> P = (3-s)/((s+1)*(s+2));
>> [A,B,C,D] = ssdata(P);
>> rank(ctrb(A,B))

ans =

2

>>  p = [1 5.6 16];
>>  L = place(A,B,roots(p));
```

The system is controllable, since the controllability matrix has full rank. With the control law $u(t) = -Lx + r$, the closed-loop system get the following appearance

```
>> A_cl = A-B*L;
>> B_cl = B;
>> C_cl = C;
>> D_cl = 0;
>> G_cl = ss(A_cl,B_cl,C_cl,D_cl);
>> step(G_cl)
>> dcgain(G_cl)

ans =

0.1875
```

The system is non-minimum phase, which we can see directly since the process has a zero in the right half plane.