Computer Exercises for Advanced Control Systems

Alireza Karimi
Vaibhav Gupta, Zhaoming Qin

Spring 2025

1 CE1: Norms of Systems and Model Uncertainty

Exercises are done using MATLAB—the required toolboxes are the control system toolbox, system identification toolbox, and robust control toolbox.

1.1 Norms of SISO systems

Consider the following second-order model:

$$G(s) = \frac{s - 10}{s^2 + 5s + 26.5}$$

1.1.1 2-Norm

Compute the two-norm of G using:

1. The residue theorem (pen and paper).
2. The frequency response of G (by approximation of the integral).
3. The impulse response of G (use `impulse`).
4. The state-space method (use `are` to solve the Algebraic Riccati Equation and find L, and `[A,B,C,D] = ssdata(G)` to obtain the corresponding state-space matrices).
5. Validate your results with the Matlab command `norm`.

1.1.2 ∞-Norm

Compute the infinity norm of G using:

1. The frequency response of G.
2. The bounded real lemma (iterative bisection algorithm).
3. Validate your results with the Matlab command `norm`.

1.2 Norms of MIMO systems

Consider the following state-space model of a MIMO system:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

with

$$A = \begin{bmatrix} -8.0 & 2.5 & -2.5 \\ 7.8 & -3.2 & 13.8 \\ 0.1 & -0.5 & -5.7 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & -2 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

1.2.1 2-Norm

Compute the two-norm using:

1. The frequency response method (by approximation of the integral).
2. The state-space method (use `are` to solve the Algebraic Riccati Equation and find L).
3. Validate your results with the Matlab command `norm`.

1.2.2 ∞-Norm

Compute the infinity norm using:

1. The frequency response method.
2. The bounded real lemma (iterative bisection algorithm).
3. Validate your results with the Matlab command `norm`.

1.3 Uncertainty modeling

The objective of this part is to convert parametric uncertainty to multiplicative frequency-domain uncertainty. Consider the following model:

$$G(s) = \frac{a}{s^2 + bs + c}$$

where $a = 2 \pm 1$, $b = 3 \pm 2$ and $c = 10 \pm 3$.

Compute the weighting filter $W_2(s)$ using Matlab:

- Define the uncertain parameters a, b, c using `ureal` command.
- Define the uncertain LTI system using the uncertain parameters (you can observe the step response and Bode or Nyquist diagram of the uncertain system using `step, bode, nyquist` commands).
- Use `usample` to generate two multimodel uncertainty sets: one based on 20 samples and the other one based on 200 samples.
- Use `ucover` command to convert the multimodel uncertainty to multiplicative one. Compare the two weighting filters for 20 and 200 samples.

2 CE2: Robust control of an electro-mechanical system

The plant focuses on the position control of a DC motor with a flexible element attached on top, resulting in large resonant modes at high frequencies. Different weights can be attached to the flexible element at different positions, resulting in different loadings and creating a multimodel uncertainty (Fig. 1).

Figure 1: Quanser Servo-Qube with weights attached on top

For identification purposes, the system is excited with a PRBS signal and the acquired data are saved in `data_position.mat` file. The data is stored as a cell array and to extract the data of i^{th} experiment use:

```
experiment_name = data{i}.name;
y = data{i}.y;
u = data{i}.u;
```

The sampling period is 0.002 s. The experiment is conducted with weights attached to the flexible element at different positions.

2.1 Multiplicative uncertainty

This part aims to study the model uncertainty in the system originating from the measurement noise and uncertainty originating from different loads. Finally, multi-model uncertainty is transformed into multiplicative uncertainty and the uncertainty filter is computed.

In order to study the model uncertainty from the measurement noise:

1. Load the `.mat` file which contains the input/output data of the system in Matlab. First covert the data into an `iddata` object for System Identification.

   ```
   Z = detrend(iddata(y, u, Ts, "Period", 8191));
   ```

   Identify the non-parametric model for the system using the `spa` method.

   ```
   freqs = (pi/4096:pi/4096:pi) / Ts;
   Gf = spa(data, 8191, freqs)
   ```

   Also, identify the parametric model for the system using the *Output Error* structure. First, compute the output derivative,

```
% Compute output derivative
y_derivative = lsim(1 - tf('z', Ts)^-1, y);
Z = detrend(iddata(y_derivative, u, Ts, "Period", 8191));
```

Then, use `oe` to compute the model for the system derivative and add back the integrator.

```
z = tf('z', Ts);
G_derivative = oe(Z, [10, 10, 1]);
G = G_derivative / (1 - z^-1);
```

The same code should be used to obtain the models for all experiments.

2. For the identified model using the spectral analysis, observe the frequency-domain uncertainty originating from measurement noise in the Nyquist diagram with the following code:

```
opts = nyquistoptions;
opts.ConfidenceRegionDisplaySpacing = 3;
opts.ShowFullContour = 'off';
% Non-parametric models
figure();
nyquistplot(Gf, freqs, opts, 'sd', 2.45);
```

to obtain the uncertainties with a 95% confidence level.

3. Observe the frequency-domain uncertainty in the Nyquist diagram using the parametric model. Comment on the form and size of uncertainty obtained from the parametric models.

```
opts = nyquistoptions;
opts.ConfidenceRegionDisplaySpacing = 3;
opts.ShowFullContour = 'off';
% Parametric models for system derivative
figure();
nyquistplot(G_derivative, freqs, opts, 'sd', 2.45);
```

4. With the same order and structure identify the parametric and the non-parametric model for the system for all the experimental data.

5. Choose a nominal model and compute the weighting filter W_2 that converts the multimodel uncertainty to multiplicative uncertainty.

   Hint: You can use `Gmm = stack(1, G1, G2, ..)` to define a multimodel set and `[Gu, info] = ucover(Gmm, Gnom, N)` to compute an uncertain model (multiplicative by default) and `W2 = info.W1opt` to compute the uncertainty optimal filter (it can also be approximated with a rational fixed-order filter of order N, see `W2 = info.W1`).

6. What is the best choice for the nominal model and why?

2.2 Model-based $\mathcal{H}_\infty$ control design

In this part, we design a discrete-time $\mathcal{H}_\infty$ controller for the system using a mixed sensitivity method.

1. Design the weighting filter $W_1(z)$ for

   - Zero steady state tracking error for step reference.
   - A modulus margin of at least 0.5.
   - The shortest settling time for the nominal model.

   Hint: You can design your weighting filter in continuous time and then convert it to discrete time. Check the value of the magnitude of W^{-1} in high frequencies (it should be less than 6dB). You can alternatively use the `makeweight` command.

2. Use W_2 for multiplicative uncertainty and design the $\mathcal{H}_\infty$ controller for robust performance using the mixed sensitivity approach (use `mixsyn`).

3. **Validation Plots**: Plot the step response of the closed-loop system (output and control signal), the magnitude of the input sensitivity function $\mathcal{U}(z)$, the sensitivity function $\mathcal{S}(z)$ and the complementary sensitivity function $\mathcal{T}(z)$. Do not forget to plot the respective filter inverses.

   Hint: Use the `feedback` command as `S = feedback(1, G*K)`, `T = feedback(G*K,1)` and `U = feedback(K,G)` to compute the sensitivity functions.

4. For a unit step reference signal, the control signal $u(t)$ should be within the range $\pm 5\,\mathrm{V}$ to avoid any saturation in real-time implementation. If these conditions are not met, use W_3 to reduce the magnitude of $U(e^{j\omega})$ and consequently the magnitude of $u(t)$. Validate with the plots.

5. The order of the final controller may be too large (especially if the order of W_2 is large). Check if there is zero/pole cancellation in the controller using `pzmap`. The order of the controller can be reduced using the `reduce` or `minreal` command. Check the stability and performance of the closed-loop system with the reduced order controller. Validate with the plots.

6. Comment on the closed-loop norms

$$\left\| \begin{bmatrix} W_1\mathcal{S} \\ W_2\mathcal{T} \end{bmatrix} \right\|_\infty, \quad \|W_1\mathcal{S}\|_\infty, \quad \|W_2\mathcal{T}\|_\infty$$

   when using the nominal model `Gnom`. What can be concluded for the same norms for the multimodel set `Gmm`?

2.3 Model-Based $\mathcal{H}_2$ Controller Design

In this part, $\mathcal{H}_2$ state feedback controller is to be designed for the system. The objective is to design a state feedback controller such that the sum of the (squared) two-norm of the closed-loop transfer functions from the input disturbance to the output and to the system's input is minimized.

1. Convert the discrete-time model to a continuous-time model (use `d2c`).

2. Write the state space equations of the closed-loop system (see Chapter 2, slide 28). To minimize the two transfer functions, we define

$$y_1(t) = Cx(t),$$
$$y_2(t) = -Kx(t).$$

The sum of the squared norm of the two transfer functions corresponds then to

$$\left\| \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} \right\|_2^2$$

3. Write a convex optimization problem using LMIs that represent this problem.

4. Compute the controller K by solving the SDP problem using `YALMIP`.

   - Download `YALMIP` and a numerical solver. It is also recommended to use `MOSEK`, required for future CEs. Academic licenses are available for students (`https://www.mosek.com/products/academic-licenses/`).

5. Plot the step response of the closed-loop system.

6. Compare your result with the Linear Quadratic Regulator computed using `lqr` command of Matlab.

2.4 Data-driven controller

Download the necessary files for data-driven controller design from the Moodle page. `YALMIP` is required, along with `MOSEK` (recommended) or `SeDuMi`.

2.4.1 Multiplicative uncertainty

1. Design a reduced order controller of maximum order of 5 for the same $\mathcal{H}_\infty$ problem as before. Validate with the plots.

   (a) Use 200 logarithmically spaced frequencies between $0.4\,\mathrm{rad\,s^{-1}}$ and Nyquist frequency, 101 linearly spaced frequencies between $100\,\mathrm{rad\,s^{-1}}$ and $1000\,\mathrm{rad\,s^{-1}}$, and 101 linearly spaced frequencies between $300\,\mathrm{rad\,s^{-1}}$ and $400\,\mathrm{rad\,s^{-1}}$ for the frequency grid.

```
omegas = unique([
    logspace(log10(0.4), log10(pi/Ts), 200), ...
    linspace(300, 400, 101), ...
    linspace(100, 1000, 101), ...
    ]);
```

   (b) Choose the initial stabilizing controller as a proportional controller, i.e., $K_{init} = c$ for some constant c.

   Hint: You might need to relax some performance filters, i.e., increase the desired settling time.

2. Can you attain a similar reduced-order controller using the model-based methods? Discuss.

3. Design a controller with nominal performance and robust stability to achieve the shortest settling time. Validate with the plots.

   Hint: You can add robust stability constraints in the data-driven framework.

2.4.2 Multimodel uncertainty

1. Design a reduced order controller of maximum order of 5 for the same $\mathcal{H}_\infty$ problem to achieve the shortest settling time. Consider multimodel uncertainty instead of multiplicative uncertainty. Validate with the plots.

   (a) Use 200 logarithmically spaced frequencies between $0.4\,\mathrm{rad\,s^{-1}}$ and Nyquist frequency, 101 linearly spaced frequencies between $100\,\mathrm{rad\,s^{-1}}$ and $1000\,\mathrm{rad\,s^{-1}}$, and 101 linearly spaced frequencies between $300\,\mathrm{rad\,s^{-1}}$ and $400\,\mathrm{rad\,s^{-1}}$ for the frequency grid.

   ```
   omegas = unique([
       logspace(log10(0.4), log10(pi/Ts), 200), ...
       linspace(300, 400, 101), ...
       linspace(100, 1000, 101), ...
       ]);
   ```

   (b) Choose the initial stabilizing controller as a proportional controller, i.e., $K_{init} = c$ for some constant c.

2. Can the desired settling time be improved compared to the multiplicative case? If so, why?

3. Comment on the advantages/drawbacks of the data-driven approach compared to the model-based approach.

3 CE3: Digital Control

3.1 RST Controller

The objective of this exercise is to design an RST controller for the system. You will need to write a pole placement function in MATLAB and use it for controller design.

1. Choose a nominal model for the system and reduce the plant order to 6 using `reduce` command. Then, extract the coefficients of the polynomials B and A using `tfdata`.

   ```
   G_reduced = reduce(G, 6)
   [B, A] = tfdata(G_reduced, 'v')
   ```

2. Write a pole placement function in MATLAB for computing the coefficients of polynomial $R(q^{-1})$ and $S(q^{-1})$ that places the closed-loop poles at $P(q^{-1})$. Take $H_r(q^{-1})$ and $H_s(q^{-1})$ as the fixed terms in $R(q^{-1})$ and $S(q^{-1})$ respectively. Note that all the polynomials are represented as a vector of the polynomial coefficients. The signature of the function should be:

   ```
   [R, S] = poleplace(B, A, Hr, Hs, P, aux_poles)
   ```

   where `aux_poles` is the location of the auxiliary poles.

3. Choose the desired dominant closed-loop poles such that the closed-loop system has an overshoot of 1 % and settling time of 0.75 s.

4. Add 4 additional auxiliary poles to the system to dampen the system's open-loop poles. The poles have the natural frequencies of 379 rad s^{-1} and 792 rad s^{-1} with the damping ratio of 0.1.
$$(\omega_1, \zeta_1) = (379, 0.1) \quad ; \quad (\omega_2, \zeta_2) = (792, 0.1)$$

5. Using your Matlab function, compute $R(q^{-1})$ and $S(q^{-1})$ to place the closed-loop poles of the system at the desired poles.

6. Compute the achieved closed-loop poles using `P = conv(A,S)+conv(B,R)` and verify your design.

   Hint: Computed closed-loop poles should be nearly equal to the desired ones.

7. Compute $T(q^{-1})$ to have the same dynamics for tracking and regulation.

8. Compute the tracking step response of the closed-loop system. The closed-loop transfer function can be computed using:
```
CL=tf(conv(T,B),P,Ts,'variable','z^-1')
```
   Use `step` and `stepinfo` and verify that the time-domain performance is achieved.

9. Verify if the designed controller can be implemented on the real system.
   - Modulus margin should be at least 0.5.
   - Infinity norm of $\mathcal{U}$, the transfer function between the measurement noise and the control signal, should be small (close to 30 dB).

   Remark: Note that $\mathcal{U}$, defined in the class notes, is the transfer function between the measurement noise and the control signal, which differs from the transfer function between the reference and the control signal!

10. If the designed RST controller does not meet the constraints for implementation, introduce the fixed term
$$H_R(q^{-1}) = 1 + q^{-1}$$
    in $R(q^{-1})$. This will open the loop at the Nyquist frequency and reduce the magnitude of $\mathcal{U}$ in high frequencies.

11. If the desired behaviour is still not attained, add some additional auxiliary poles.

3.2 RST Controller with Q-parameterisation

The RST controller designed in the previous section would be made robust using Q-parameterisation. If the constraints on the magnitude of U and the modulus margin are not satisfied, write an optimisation problem using the Q-parametrisation to meet the constraints. The optimisation problem would be,

$$\begin{aligned} & \min \|\mathcal{U}\|_\infty \\ s.t., \quad & \|W_1 \mathcal{S}\|_\infty \leq 1 \end{aligned}$$

1. Write the closed-loop transfer functions in Controllable Canonical form.

   Hint: Transfer function of the form

   $$G(q^{-1}) = \frac{b_0 + b_1 q^{-1} + \cdots + b_n q^{-n}}{1 + a_1 q^{-1} + \cdots + a_n q^{-n}}$$

   can be converted to the Controllable Canonical form as:

   $$\begin{aligned} x[k+1] &= Ax[k] + Bu[k] \\ y[k] &= Cx[k] + Du[k] \end{aligned}$$

   where

   $$A = \begin{bmatrix} -a_1 & -a_2 & \cdots & -a_{n-1} & -a_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}$$

   $$C = \begin{bmatrix} b_1 - b_0 a_1 & b_2 - b_0 a_2 & \cdots & b_n - b_0 a_n \end{bmatrix} \qquad D = \begin{bmatrix} b_0 \end{bmatrix}$$

2. Write the equivalent $\mathcal{H}_\infty$ objective as an SDP using the Bounded Real lemma for discrete-time systems.

   Bounded Real lemma for discrete-time systems: A discrete-time system G has $\mathcal{H}_\infty$ norm of γ, i.e. $\|G\|_\infty \leq \gamma$ if there exists $P = P^T \succeq 0$ such that:

   $$\begin{bmatrix} A^T P A - P & A^T P B & C^T \\ B^T P A & B^T P B - \gamma & D^T \\ C & D & -\gamma \end{bmatrix} \preceq 0$$

3. Convert the desired Q-parametrisation into a convex optimisation and find an optimal RST controller.

4. Compare the resulting RST controller with the initial ones using the validation plots.

5. Validate if the obtained controller is robust for the multiplicative uncertainty.

3.3 Real-time Implementation

In this section, the synthesised controllers will be implemented on the system.

1. Implement the designed data-driven controller on the hardware and check the controller performance for the weight at different positions.

2. Implement the designed RST controller on the hardware and check the controller performance for the weight at different positions.

3. Discuss the obtained results and differences between the two methods.

LabVIEW Interface

The Figure 2 shows the front-panel of the LabVIEW interface for the velocity control of the Quanser Servo-Qube with weights attached on top.

All teams should implement the controllers and check the response for the mixed reference of amplitude 45°. The experiment would be done for 3 positions of the weights: 6 cm, 10 cm and 12 cm away from the middle.

Note: If significant saturation is observed, the amplitude should be reduced to something reasonable.

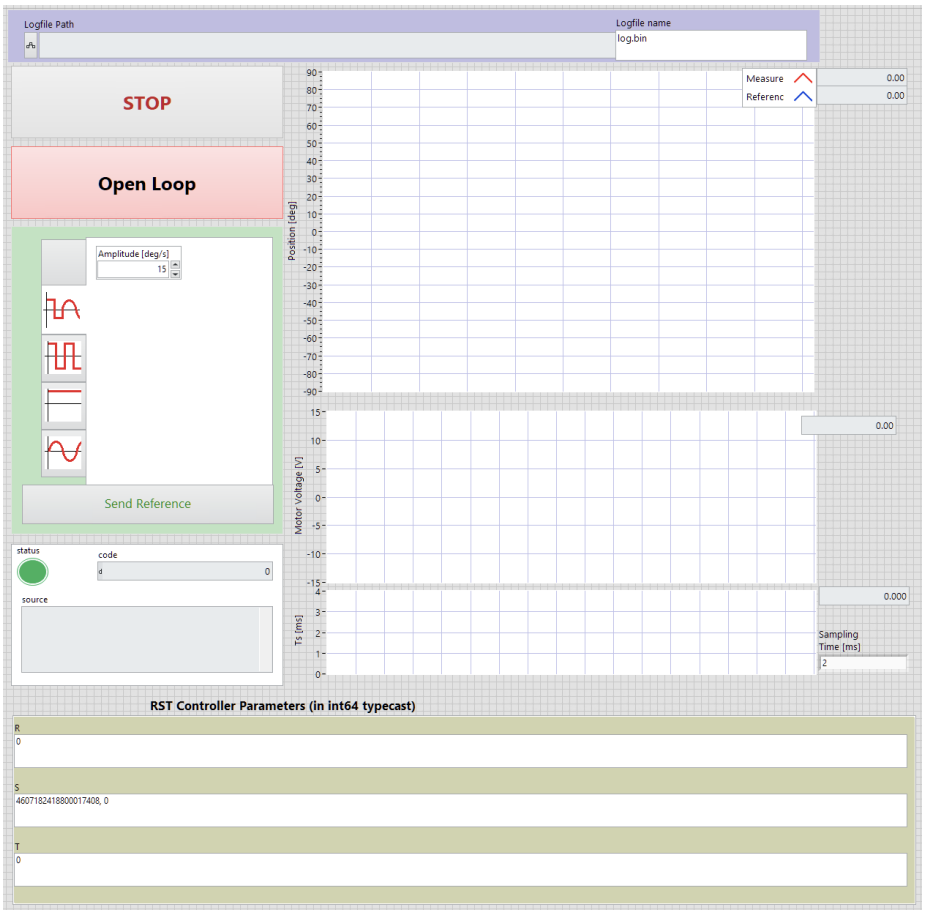

Figure 2: LabVIEW interface for Quanser Servo-Qube with weights attached on top

Copy parameters from MATLAB The controller parameters in MATLAB are of type `double`, which should be typecast to `int64` to avoid loss of precision. Use the following code to copy the parameters from MATLAB:

```
STR = join(compose('%d', typecast(VARIABLE,'int64')), ',');
clipboard('copy', STR{1})
```

Here, `VARIABLE` is the array to be copied.

LabVIEW output in MATLAB LabVIEW saves a `.bin` file for each run with the measurements, which can be read using the provided function

```
[y, r, u, d, anyRef, sample] = read_binary_file(path)
```

Here,

- $path \rightarrow$ Location of the `.bin` file
- $y \rightarrow$ Measurements (Only the first measurement is utilised)
- $r \rightarrow$ Reference signals (Only the first reference is utilised)
- $u \rightarrow$ Unsaturated motor command
- $d \rightarrow$ Added disturbance
- $anyRef \rightarrow$ Boolean indicating when a reference is applied
- $sample \rightarrow$ Sample number (always increases by 1)