Computer Exercises for Advanced Control Systems

Alireza Karimi
Vaibhav Gupta, Zhaoming Qin

Spring 2024

1 CE1: Norms of Systems and Model Uncertainty

Exercises are done using MATLAB. Required toolboxes: control system toolbox, system identification toolbox, robust control toolbox.

1.1 Norms of SISO systems

Consider the following second-order model:

$$G(s) = \frac{s-1}{s^2+2s+10}$$

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:

$$\begin{aligned}\dot{x} &= Ax + Bu\\ y &= Cx + Du\end{aligned}$$

with

$$A = \begin{bmatrix} 20 & -27 & 7 \\ 53 & -63 & 13 \\ -5 & 12 & -8 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & -1 \\ -2 & -1 \\ -3 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & -2 \\ 1 & -1 & -1 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

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 $a = 11 \pm 1$, $b = 4 \pm 1$ and $c = 9 \pm 2$.

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 velocity 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).

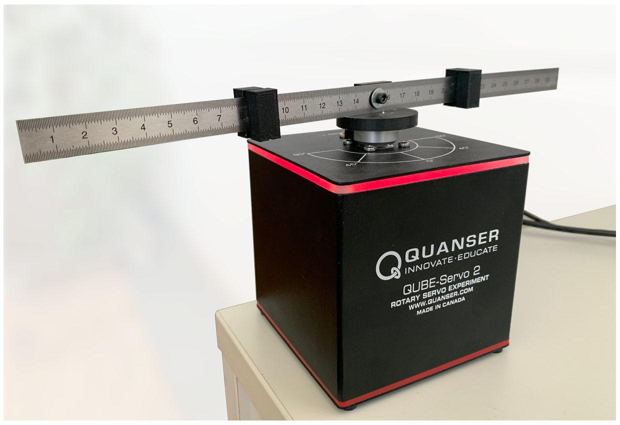

Figure 1: Quanser Servo-Qube with weights attached on top

For the identification purpose, the system is excited with a PRBS signal and the acquired data are saved in `*.mat` files. The sampling period is 0.002 s. The experimented is conducted with weights attached to the flexible element at different positions.

2.1 Multiplicative uncertainty

The objective of this part is to study the model uncertainty in the system originating from the measurement noise as well as 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 one of the `*.mat` files which contains the input/output data of the system as an `iddata` object in Matlab. Identify the parametric model for the system using the *Output Error* structure.

   ```
   G = oe(data, [8, 8, 1])
   ```

   Also, identify the non-parametric model for the system using the `spa` method.

   ```
   freqs = (pi/4096:pi/4096:pi) / Ts;
   Gf1 = spa(data, 8191, freqs)
   ```

   The following codes should be used to obtain the same model for all groups.

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';
figure();
nyquistplot(Gf1, G, freqs, opts, 'sd', 2.45);
```

to obtain the uncertainties with 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 two models.

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 or 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 mixed sensitivity method.

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

   - Zero steady state tracking error for a 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. Plot the step response of the closed-loop system (output and control signal), the magnitude of the input sensitivity function $\mathcal{U}(z)$ and the sensitivity function $\mathcal{S}(z)$.

   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 $\pm 1.5\,\mathrm{V}$ 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 $u(t)$.

   Note: Ensure that the robust performance conditions are still met!

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.

6. Comment on the closed-loop norm

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

when using the nominal model `Gnom`, or when used 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 input of the system 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). In order 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 represents 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. Solve the same $\mathcal{H}_\infty$ problem as before. For the frequency grid, use 400 logarithmically spaced frequencies between 0 and $\pi/\mathrm{T_s}$. Choose the initial stabilizing controller as a small gain integrator controller, i.e., for some constant c

$$K_{init} = \frac{c}{1 - z^{-1}}$$

2. Comment on the difference in performance. What explains the difference?
3. Design a reduced order controller with similar performance.

2.4.2 Multimodel uncertainty

1. Solve the same $\mathcal{H}_\infty$ problem using multimodel uncertainty instead of multiplicative uncertainty. For the frequency grid, use 400 logarithmically spaced frequencies between 0 and $\pi/\mathrm{T_s}$. Use the same initial stabilizing controller as before.
2. Comment on the difference in performance. What explains the difference?
3. Design a reduced order controller with similar performance.
4. Comment on the advantages/drawbacks of this approach compared to the model-based approach.

3 CE3: Digital Control

3.1 RST Controller

The objective of this exercise is to design a RST controller for the system. You need to write a pole placement function in MATLAB and use it for controller design. Finally, the controller would be made robust using Q-parameterisation.

1. Choose a nominal model for the system and extract the coefficients of the polynomials B and A using `[B, A] = tfdata(G, '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 and H_s as the fixed terms in $R(q^{-1})$ and $S(q^{-1})$ respectively. Note that all the variables are vector including the coefficients of the polynomials. The signature of the function should be:

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

3. Choose the desired dominant closed-loop poles to be 0.8, 0.9 and 0.95.

   Alternative: The desired dominant closed-loop poles can be set to 0.95, 0.95 and 0.99. This might give an easier Q-parametrisation to solve.

4. Using your Matlab function, compute $R(q^{-1})$ and $S(q^{-1})$ to place the closed-loop poles of the system in the desired place. Do not forget to include the integrator as a fixed term in the controller.

5. Compute the achieved closed-loop poles using `P = conv(A,S)+conv(B,R)` and verify your design. (Computed closed-loop poles should be nearly equal to the desired ones)

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

7. 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.

8. Verify if the designed controller can be implemented on the real system. The step measurement noise should be rejected within $\pm 10\,\mathrm{V}$ and a modulus margin should be atleast 0.5.

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

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

10. If the constraints on the magnitude of U and the modulus margin are not satisfied, write an optimization problem using the Q-parametrisation to meet the constraints. Compare the resulting RST controller with the initial ones.

    Hint: You can use `fmincon` of Matlab for optimization. You should write a function that represents the objective to be minimized and a constraint function for the constraints. The optimisation problem should be,

$$\begin{aligned} & \min \|\mathcal{U}\|_\infty \\ s.t., \quad & \|W_1\mathcal{S}\|_\infty \leq 1 \\ & \|W_2\mathcal{T}\|_\infty \leq 1 \end{aligned}$$

3.2 Real-time Implementation

In this section, the synthesised controllers would be implemented to 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.

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 $15\,^\circ\,\mathrm{s}^{-1}$. The experiment would be done for 3 positions of the weights: 4 cm, 8 cm and 14 cm away from the middle.

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

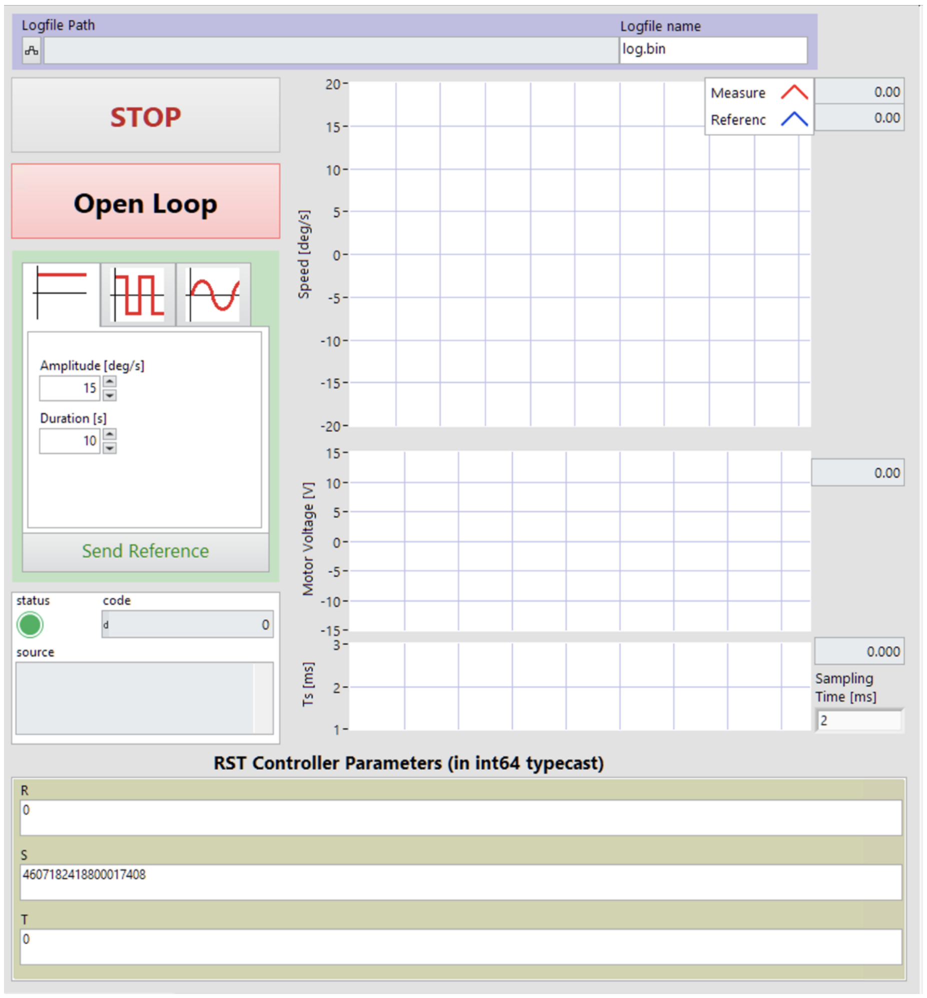

Figure 2: LabVIEW interface for Quanser Servo-Qube with weights attached on top

Copy parameters from MATLAB The controller parameters in MATLAB are in the type `double`, which should be typecasted 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,

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