

MATLAB Control Practice Tutorial

Using the Control System Toolbox for Modern Control

Control Practice Instructor

Contents

1	Introduction	3
1.1	Software requirements and setup	3
2	Basic Models and Time Responses	3
2.1	First-order system: RC circuit / thermal model	3
2.1.1	Creating a transfer function	4
2.1.2	Step and impulse responses	4
2.2	Second-order system: mass–spring–damper	4
2.2.1	Creating and plotting the response	4
2.3	General transfer functions	5
2.3.1	Polynomial form	5
2.3.2	Zero–pole–gain form	5
3	Modeling Physical Systems in MATLAB	6
3.1	Mechanical example: mass–spring–damper	6
3.1.1	From ODE to transfer function	6
3.1.2	From ODE to state-space	6
3.1.3	Simulating state-space response	6
3.2	Electrical example: series RLC circuit	7
3.3	From physical parameters to models: checklist	7
4	Linearization and Small-Signal Models	7
4.1	Example: simple pendulum	7
4.2	Building the linearized model	8
5	State-Space Analysis and Solution	8
5.1	General LTI solution	8
5.2	Zero-input and zero-state responses	9
5.3	Controllability and observability	9
5.4	Canonical realizations and conversions	9
6	Block Diagrams and Interconnections	10
6.1	Unity feedback	10
6.2	Mixed series/parallel/feedback interconnection	11
7	PID and Basic Feedback Design	11
7.1	PID controller with pid and pidtune	11

8	State-Feedback and LQR Design	12
8.1	Pole placement with <code>place</code>	12
8.2	LQR design	12
9	Design Example: Cruise Control for a Car	12
9.1	Model	13
9.2	Step 1: Build and inspect the plant	13
9.3	Step 2: Open-loop step response	13
9.4	Step 3: PI cruise controller design	13
9.5	Step 4: Disturbance rejection (hill)	14
10	Mini Project: DC Motor Speed Control	14
10.1	Physical model	15
10.2	Project tasks	15
10.3	Skeleton MATLAB code	15
11	Practice Problems	16

1 Introduction

This tutorial is designed to be used together with the lecture slides of Chapters 1–3 on modern control, mathematical modeling, and state variable models. The goal is to teach you how to use **MATLAB** and the **Control System Toolbox** to solve the same kinds of problems you see in the theory class:

- Build models (ODE, transfer function, state-space) for mechanical, electrical, and thermal systems.
- Simulate time responses, including step, impulse, and arbitrary inputs.
- Connect blocks to form feedback systems and analyze closed-loop behavior.
- Work with state-space models, controllability, and observability.
- Move between physical state models and canonical forms (phase-variable, input–feedforward).
- Do a complete design example and a mini-project on DC motor control.

The document is organized to follow the logic of the lecture slides: from basic feedback ideas to modeling, linearization, and state-space.

1.1 Software requirements and setup

You should have:

- MATLAB (R2020a or later recommended).
- **Control System Toolbox**.
- (Optional but useful) Symbolic Math Toolbox and Simulink.

Open MATLAB and verify that the Control System Toolbox is installed:

```
1 ver control
```

You should see a line containing “Control System Toolbox”. If not, ask your instructor or system administrator.

Throughout this tutorial, you are encouraged to type the code yourself and modify parameters to see how the responses change.

2 Basic Models and Time Responses

2.1 First-order system: RC circuit / thermal model

In Chapter 1, you saw simple first-order systems (RC circuit, room temperature, tanks). A standard first-order transfer function is

$$G(s) = \frac{K}{\tau s + 1},$$

where K is the DC gain and τ is the time constant.

2.1.1 Creating a transfer function

Example: an RC circuit with $R = 1\text{ k}\Omega$ and $C = 100\text{ }\mu\text{F}$ has $\tau = RC = 0.1\text{ s}$ and unit gain $K = 1$.

```
1 % Parameters
2 R = 1e3; % Ohms
3 C = 100e-6; % Farads
4 tau = R*C;
5 K = 1;
6
7 % Create transfer function G(s) = K/(tau*s+1)
8 s = tf('s'); % symbolic Laplace variable
9 G = K/(tau*s + 1);
10
11 % Display and check DC gain
12 G
13 dc_gain = dcgain(G)
```

The command `tf('s')` creates a “transfer function variable” s . You can then write expressions using standard arithmetic.

2.1.2 Step and impulse responses

To visualize how the circuit responds to a step in input voltage:

```
1 % Step response from 0 to 1 V
2 figure;
3 step(G);
4 grid on;
5 title('First-order RC: Step Response');
6
7 % Impulse response (current or derivative-related signals)
8 figure;
9 impulse(G);
10 grid on;
11 title('First-order RC: Impulse Response');
```

Use `stepinfo` to measure performance metrics such as rise time, settling time, and overshoot:

```
1 info = stepinfo(G)
2
3 % Inspect fields: info.RiseTime, info.SettlingTime, info.Overshoot, etc.
```

2.2 Second-order system: mass–spring–damper

A standard mass–spring–damper system has equation

$$m\ddot{x} + c\dot{x} + kx = f(t),$$

with displacement output x and force input f . The transfer function from f to x is

$$G(s) = \frac{X(s)}{F(s)} = \frac{1}{ms^2 + cs + k}.$$

2.2.1 Creating and plotting the response

```

1 % Parameters
2 m = 1; % kg
3 c = 2; % N*s/m
4 k = 5; % N/m
5
6 % Transfer function from force to displacement
7 s = tf('s');
8 Gmsd = 1/(m*s^2 + c*s + k);
9
10 % Inspect poles and damping
11 pole(Gmsd)
12 [zeta, wn] = damp(Gmsd)
13
14 % Step response to unit step force
15 figure;
16 step(Gmsd);
17 grid on;
18 title('Mass-Spring-Damper: Step Response to Unit Force');
19
20 % Impulse response (corresponds to initial velocity)
21 figure;
22 impulse(Gmsd);
23 grid on;
24 title('Mass-Spring-Damper: Impulse Response');

```

The command `damp` returns the natural frequency and damping ratio corresponding to the theoretical formulas in the lecture.

2.3 General transfer functions

You can create transfer functions in several ways:

- Using numerator and denominator polynomials.
- Using zero/pole/gain data.
- Using symbolic expressions with `tf('s')`.

2.3.1 Polynomial form

```

1 % G(s) = (2s + 5) / (s^3 + 4s^2 + 3s + 1)
2 num = [2 5];
3 den = [1 4 3 1];
4 Gpoly = tf(num, den);
5
6 % Analyze poles and zeros
7 pzmap(Gpoly);
8 grid on;
9 title('Poles and Zeros of G(s)');

```

2.3.2 Zero-pole-gain form

```

1 % Same system in zero/pole/gain form
2 [z, p, k] = tf2zp(num, den);
3 Gzpk = zp(z, p, k);
4

```

3 Modeling Physical Systems in MATLAB

Chapter 2 focuses on building mathematical models from physics. Here we show how to go from ODEs to transfer functions and state-space models in MATLAB.

3.1 Mechanical example: mass–spring–damper

3.1.1 From ODE to transfer function

Starting from

$$m\ddot{x} + c\dot{x} + kx = f(t),$$

we have already built the transfer function `Gmsd`.

3.1.2 From ODE to state-space

Choose states $x_1 = x$, $x_2 = \dot{x}$. Then

$$\dot{x}_1 = x_2, \quad \dot{x}_2 = -\frac{k}{m}x_1 - \frac{c}{m}x_2 + \frac{1}{m}f.$$

In MATLAB:

```

1 A = [0 1;
2     -k/m -c/m];
3 B = [0;
4     1/m];
5 C = [1 0]; % output = position
6 D = 0;
7
8 Smsd = ss(A, B, C, D);
9
10 % Compare with transfer function
11 G_from_ss = tf(Smsd);
12 G_from_ss

```

3.1.3 Simulating state-space response

Zero-input (non-zero initial condition, input $f(t) = 0$):

```

1 x0 = [0.1; 0]; % initial displacement 0.1 m, zero velocity
2 t = 0:0.01:10; % time vector
3
4 figure;
5 initial(Smsd, x0, t);
6 grid on;
7 title('Zero-Input Response of Mass-Spring-Damper');

```

Forced response with step input force:

```

1 t = 0:0.01:10;
2 u = ones(size(t)); % unit step force
3
4 figure;
5 lsim(Smsd, u, t);
6 grid on;
7 title('Forced Response of Mass-Spring-Damper (u=1)');

```

3.2 Electrical example: series RLC circuit

From the lectures, the series RLC with input voltage $v(t)$ and state variables q (charge) and i (current) can be written as

$$\dot{x}_1 = \dot{q} = i, \quad \dot{x}_2 = \dot{i} = -\frac{1}{LC}q - \frac{R}{L}i + \frac{1}{L}v.$$

In MATLAB:

```
1 % Parameters
2 R = 3;
3 L = 1;
4 C = 0.5;
5
6 A = [ 0 1;
7       -1/(L*C) -R/L ];
8 B = [ 0;
9       1/L ];
10 Cmat = [0 1]; % output = inductor current, for instance
11 D = 0;
12
13 SRLC = ss(A,B,Cmat,D);
14
15 % Step response to a step in input voltage
16 figure;
17 step(SRLC);
18 grid on;
19 title('Series RLC: Current Response to Step Voltage');
```

3.3 From physical parameters to models: checklist

For any new physical system:

1. Draw a free-body diagram or circuit diagram.
2. Write down the governing equations using Newton/Kirchhoff laws.
3. Choose states (prefer energy variables: positions, velocities, currents, voltages).
4. Convert high-order ODEs into first-order state equations.
5. Implement the A, B, C, D matrices in MATLAB using `ss`.
6. Optionally convert to transfer function using `tf`.

4 Linearization and Small-Signal Models

Chapter 2 introduces linearization and small-signal approximations. Here we show how to do this in MATLAB for a simple nonlinear system.

4.1 Example: simple pendulum

Nonlinear state equations:

$$\dot{x}_1 = x_2, \quad \dot{x}_2 = -\frac{g}{L} \sin x_1 + \frac{1}{mL^2}u.$$

We will compute the Jacobian at the equilibrium $(x_1, x_2, u) = (0, 0, 0)$ using Symbolic Math Toolbox.

```

1 syms x1 x2 u g L m real
2
3 % Nonlinear dynamics
4 f1 = x2;
5 f2 = -(g/L)*sin(x1) + 1/(m*L^2)*u;
6
7 f = [f1; f2];
8
9 % State vector and input
10 x = [x1; x2];
11
12 % Jacobians
13 A = jacobian(f, x);
14 B = jacobian(f, u);
15
16 % Evaluate at equilibrium: x1=0, x2=0, u=0
17 g_val = 9.81;
18 L_val = 1;
19 m_val = 1;
20
21 A_lin = double(subs(A, {x1,x2,u,g,L,m}, {0,0,0,g_val,L_val,m_val}));
22 B_lin = double(subs(B, {x1,x2,u,g,L,m}, {0,0,0,g_val,L_val,m_val}));
23
24 A_lin
25 B_lin

```

You should obtain

$$A_{\text{lin}} = \begin{bmatrix} 0 & 1 \\ -g/L & 0 \end{bmatrix}, \quad B_{\text{lin}} = \begin{bmatrix} 0 \\ 1/(mL^2) \end{bmatrix},$$

matching the small-angle linearization in the lecture.

4.2 Building the linearized model

```

1 C = [1 0]; % output = angle
2 D = 0;
3
4 Spend = ss(A_lin, B_lin, C, D);
5
6 figure;
7 step(Spend);
8 grid on;
9 title('Linearized Pendulum: Step Response (Torque Input)');

```

Compare this linear model with a nonlinear simulation (if you have Simulink) to see the range of validity of the small-angle approximation.

5 State-Space Analysis and Solution

Chapter 3 develops the general solution of LTI state equations and signal-flow graphs. Here we focus on what you actually do in MATLAB.

5.1 General LTI solution

For $\dot{x} = Ax + Bu$, $y = Cx + Du$, the solution is

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau) d\tau.$$

MATLAB will not typically ask you to compute this integral by hand; instead you use simulation commands like `initial`, `step`, and `lsim`. However, you can inspect e^{At} directly with `expm`:

```
1 % Example: mass-spring-damper A matrix
2 A = [0 1;
3      -k/m -c/m];
4
5 % State transition matrix at t = 0.5 s
6 t = 0.5;
7 Phi = expm(A*t)
```

5.2 Zero-input and zero-state responses

To separate zero-input and zero-state responses:

- **Zero-input:** use `initial` with nonzero initial conditions and input $u(t) = 0$.
- **Zero-state:** use `lsim` or `step` with zero initial conditions.

```
1 % Zero-input response
2 x0 = [0.1; 0];
3 t = 0:0.01:10;
4 figure;
5 initial(Smsd, x0, t);
6 title('Zero-Input Response');
7
8 % Zero-state response to step
9 u = ones(size(t));
10 figure;
11 lsim(Smsd, u, t, [0;0]);
12 title('Zero-State Response to Step Input');
```

5.3 Controllability and observability

From Chapter 1 and Chapter 3, controllability and observability are tested by rank conditions.

In MATLAB:

```
1 A = Smsd.A;
2 B = Smsd.B;
3 C = Smsd.C;
4
5 Co = ctrb(A,B);
6 Ob = obsv(A,C);
7
8 rank_Co = rank(Co)
9 rank_Ob = rank(Ob)
```

If both ranks equal n (the state dimension), the system is controllable and observable.

5.4 Canonical realizations and conversions

Given a transfer function, you can obtain state-space realizations automatically.

```
1 % Example transfer function
2 num = [2 5];
3 den = [1 4 3 1];
4 G = tf(num, den);
```

```

5
6 % Default state-space realization
7 S_default = ss(G)
8
9 % Phase-variable (controllable companion) form
10 [S_pv, T_pv] = canon(S_default, 'companion');
11
12 % Observable / input-feedforward companion form
13 [S_if, T_if] = canon(S_default, 'modal'); % or 'observable' depending on version

```

The matrix T returned by `canon` is the similarity transformation between physical and canonical coordinates:

$$A_{\text{can}} = T^{-1}AT, \quad B_{\text{can}} = T^{-1}B, \quad C_{\text{can}} = CT.$$

You can convert back and forth between state-space and transfer function:

```

1 [num2, den2] = ss2tf(S_default.A, S_default.B, S_default.C, S_default.D);
2 G2 = tf(num2, den2);

```

6 Block Diagrams and Interconnections

Chapter 2 discusses block diagrams and mixed interconnections. The Control System Toolbox provides standard functions:

- `series(G1,G2)`: forms G_2G_1 (note the order).
- `parallel(G1,G2)`: forms $G_1 + G_2$.
- `feedback(G,H)`: forms $\frac{G}{1+GH}$ (negative feedback by default).

6.1 Unity feedback

For a plant G and controller G_c in unity feedback:

$$T(s) = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)}.$$

In MATLAB:

```

1 % Plant and controller
2 G = tf(1, [1 2 5]); % example plant
3 Gc = tf(10, [1 1]); % simple PI or lead-like controller
4
5 % Open-loop and closed-loop
6 Gopen = series(Gc, G);
7 Tc1 = feedback(Gopen, 1); % unity feedback H(s)=1
8
9 figure;
10 step(Tc1);
11 grid on;
12 title('Closed-loop Step Response');

```

6.2 Mixed series/parallel/feedback interconnection

Reproduce the “mixed interconnection” example:

```
1 G1 = tf(1, [1 1]);
2 G2 = tf(2, [1 2]);
3 H = tf(1, [1 4]);
4 Gb = tf(3, [1 3]);
5
6 % Series G2*G1
7 Gf = series(G2, G1);
8
9 % Close the feedback loop with H
10 Gcl = feedback(Gf, H); % negative feedback
11
12 % Add parallel bypass
13 Geq = parallel(Gcl, Gb);
14
15 Geq
```

7 PID and Basic Feedback Design

Chapter 1 introduces classical feedback and PID control. Here we use MATLAB to design simple PID controllers and examine time responses.

7.1 PID controller with pid and pidtune

Suppose the plant is a first-order system:

$$G(s) = \frac{1}{0.5s + 1}.$$

```
1 G = tf(1, [0.5 1]);
2
3 % Automatic tuning of PID controller
4 Gc = pidtune(G, 'PID');
5
6 Gc
7
8 % Closed-loop transfer function under unity feedback
9 Tc1 = feedback(Gc*G, 1);
10
11 figure;
12 step(Tc1);
13 grid on;
14 title('Closed-loop Step Response with PID');
15 stepinfo(Tc1)
```

Try changing the target bandwidth:

```
1 [Gc_fast, info_fast] = pidtune(G, 'PID', 4); % target bandwidth ~4 rad/s
2 Tc1_fast = feedback(Gc_fast*G, 1);
3
4 figure;
5 step(Tc1, Tc1_fast);
6 grid on;
7 legend('Nominal PID', 'Faster PID');
8 title('Comparison of PID Designs');
```

8 State-Feedback and LQR Design

Chapter 3 discusses state-space, controllability, and canonical forms. Here we implement state-feedback and LQR designs.

8.1 Pole placement with place

For the mass-spring-damper in state-space form, choose desired poles and design K such that $u = -Kx$.

```
1 A = Smsd.A;
2 B = Smsd.B;
3 C = Smsd.C;
4
5 % Desired closed-loop poles (more negative => faster)
6 p_des = [-2 -3];
7
8 K = place(A, B, p_des);
9
10 Acl = A - B*K; % closed-loop A-matrix
11 Scl = ss(Acl, B, C, 0);
12
13 figure;
14 step(Scl);
15 grid on;
16 title('Closed-loop with State-Feedback (Pole Placement)');
```

8.2 LQR design

Define cost

$$J = \int_0^\infty (x^\top Q x + u^\top R u) dt.$$

```
1 Q = C'*C; % penalize output (position) strongly
2 R = 0.1; % penalize input mildly
3
4 [K_lqr, S_lqr, e_lqr] = lqr(A, B, Q, R);
5
6 Acl_lqr = A - B*K_lqr;
7 Scl_lqr = ss(Acl_lqr, B, C, 0);
8
9 figure;
10 step(Scl_lqr);
11 grid on;
12 title('Closed-loop with LQR State-Feedback');
```

Compare pole locations:

```
1 eig(A) % open-loop
2 eig(Acl_lqr) % LQR closed-loop
```

9 Design Example: Cruise Control for a Car

This design example pulls together the main learning points (modeling, TF/SS, feedback, PID, and state-space). The mini project at the end will focus on a DC motor; here we use a different system: vehicle cruise control.

9.1 Model

A simple longitudinal model:

$$m\dot{v} = -bv + u + d,$$

where:

- v is vehicle speed (m/s),
- u is the engine force command (N),
- d is a disturbance (e.g., hill grade),
- m is mass, b is drag coefficient.

Transfer function from u to v (ignoring d for now):

$$G(s) = \frac{1}{ms + b}.$$

9.2 Step 1: Build and inspect the plant

```
1 m = 1000; % kg
2 b = 50; % N*s/m
3
4 s = tf('s');
5 Gv = 1/(m*s + b);
6
7 Gv
8 pole(Gv)
9 dcgain(Gv)
```

9.3 Step 2: Open-loop step response

Interpret this as applying a constant engine force and seeing how speed evolves.

```
1 figure;
2 step(Gv);
3 grid on;
4 title('Vehicle Speed: Response to Unit Force');
```

9.4 Step 3: PI cruise controller design

We want to regulate speed to a reference v_{ref} (e.g., 20 m/s). A PI controller is

$$G_c(s) = K_p + \frac{K_i}{s}.$$

```
1 % Simple PI tuning with pidtune
2 Gc_PI = pidtune(Gv, 'PI');
3
4 Gc_PI
5
6 % Closed-loop transfer function
7 Tc1 = feedback(Gc_PI*Gv, 1);
8
9 figure;
10 step(Tc1);
```

```

11 grid on;
12 title('Cruise Control: Closed-loop Step Response (Speed)');
13 stepinfo(Tc1)

```

9.5 Step 4: Disturbance rejection (hill)

Approximate a hill as an additional constant drag force d that appears suddenly at $t = 20$ s.

```

1 % Simulate disturbance as an additive input using lsim on state-space form
2 Sv = ss(Gv);
3
4 t = 0:0.1:60;
5 r = 20*ones(size(t)); % desired speed, 20 m/s
6
7 % Controller in continuous-time
8 Gc = Gc_PI;
9
10 % Use closed-loop with disturbance model:
11 % We'll build a simple loop: u = Gc*(r - v), plant Gv, disturbance added at plant
    input.
12
13 % For simulation, use a custom script (simplest via Simulink),
14 % but approximate here by using feedback() and a step in disturbance.
15 Gd = Gv; % disturbance path (additive at same point)
16
17 % Closed-loop from reference to output
18 Tyr = feedback(Gc*Gv, 1);
19
20 % Closed-loop from disturbance to output (standard formula)
21 Tyd = feedback(Gd, Gc); % disturbance enters at plant input
22
23 % Build composite response
24 r_input = 20*ones(size(t));
25 d_input = zeros(size(t));
26 d_input(t >= 20) = -2000; % constant opposing force after 20 s
27
28 [y_r, ~] = lsim(Tyr, r_input, t);
29 [y_d, ~] = lsim(Tyd, d_input, t);
30
31 y = y_r + y_d;
32
33 figure;
34 plot(t, y, 'LineWidth', 1.5);
35 hold on;
36 plot(t, r_input, '--', 'LineWidth', 1.2);
37 grid on;
38 xlabel('Time [s]');
39 ylabel('Speed [m/s]');
40 legend('Actual speed', 'Reference', 'Location', 'Best');
41 title('Cruise Control: Disturbance Rejection Example');

```

Observe how feedback restores the speed close to the reference even after the disturbance.

10 Mini Project: DC Motor Speed Control

This mini project is intended to be done after you complete the tutorial sections. You should be able to complete it in a few hours.

10.1 Physical model

From Chapter 3, a simple DC motor model is:

$$L\dot{i} = -Ri - K_e\omega + v, \quad J\dot{\omega} = K_t i - b\omega,$$

with states $x = [i, \omega]^\top$, input v , and output ω .

10.2 Project tasks

1. **Derive the state-space model** (A, B, C, D) for the motor using given parameters (choose realistic values, e.g. $R = 2\ \Omega$, $L = 0.5\ \text{H}$, $K_t = 0.1\ \text{Nm/A}$, $K_e = 0.1\ \text{V/(rad/s)}$, $J = 0.01\ \text{kg m}^2$, $b = 0.001\ \text{N m s}$).
2. **Implement the state-space model in MATLAB** using `ss`. Simulate the open-loop response to a step input in voltage.
3. **Convert to transfer function** from v to ω using `tf`. Inspect poles and zeros, and compute the DC gain.
4. **Design a PI speed controller** using `piddtune` so that the closed-loop speed response has:
 - rise time less than 0.5 s,
 - overshoot less than 10%,
 - steady-state error less than 1% for a step from 0 to 100 rad/s.

Verify the specifications using `stepinfo`.

5. **Investigate disturbance rejection:** Add a load torque step (e.g. $0 \rightarrow 0.05\ \text{Nm}$ at $t = 1\ \text{s}$) as a disturbance. Model its effect as an additional input to the mechanical equation and show how feedback reduces the speed deviation.
6. **State-feedback extension (optional):** Design an LQR controller based on the motor state-space model. Compare step responses and control signals between PI and LQR designs.
7. **Report:** Prepare a short report summarizing:
 - Your chosen parameters and model.
 - MATLAB code snippets for each step.
 - Plots of open-loop and closed-loop responses.
 - A brief discussion of performance trade-offs (speed vs. control effort vs. robustness).

10.3 Skeleton MATLAB code

```
1 % Parameters (choose and document in your report)
2 R = 2; % Ohm
3 L = 0.5; % H
4 Kt = 0.1; % Nm/A
5 Ke = 0.1; % V/(rad/s)
6 J = 0.01; % kg*m^2
7 b = 0.001; % N*m*s
8
9 % State-space matrices
10 A = [ -R/L -Ke/L;
```

```

11      Kt/J -b/J ];
12 B = [1/L; 0];
13 C = [0 1]; % output = speed
14 D = 0;
15
16 Smotor = ss(A,B,C,D);
17
18 % Open-loop step response (voltage input)
19 figure;
20 step(Smotor);
21 grid on;
22 title('DC Motor: Open-loop Speed Response to Step Voltage');
23
24 % Transfer function v -> omega
25 Gmotor = tf(Smotor)
26
27 % PI controller design (you refine this)
28 Gc_PI = pidtune(Gmotor, 'PI');
29 Gc_PI
30
31 Tc1 = feedback(Gc_PI*Gmotor, 1);
32 figure;
33 step(Tc1);
34 grid on;
35 title('DC Motor: Closed-loop Speed Response with PI');
36
37 info = stepinfo(Tc1)

```

You should augment this skeleton with your own code for disturbance modeling, LQR design (if attempted), and plotting.

11 Practice Problems

The following 10 problems are designed to reinforce the topics from the slides and the tutorial. They are intended to take several hours of work including thinking, coding, and writing.

For each problem, include your MATLAB code and key plots in your solution.

Problem 1: First-order thermal system

Model a room heating system as a first-order process:

$$C\dot{T} = -\frac{1}{R}(T - T_{\text{amb}}) + u,$$

where T is room temperature, T_{amb} is ambient temperature, u is heating power input, R is thermal resistance, and C is thermal capacitance.

1. Derive the transfer function from u to T (deviation from ambient).
2. Implement the transfer function in MATLAB and compute the step response to a unit heating input.
3. Use `stepinfo` to report time constant and 2% settling time.

Problem 2: Mass–spring–damper state-space vs. transfer function

Using the mass–spring–damper system:

1. Build the state-space model with states $x_1 = x$, $x_2 = \dot{x}$.
2. Convert the state-space model to a transfer function using `tf`.
3. For the same system, directly define the transfer function from f to x . Verify that the two transfer functions are identical (or differ only by numerical round-off).

Problem 3: Linearization of a nonlinear spring

Consider a nonlinear spring force $f(y) = k_1 y + k_3 y^3$ acting on a mass M :

$$M\ddot{y} = -f(y).$$

1. Find the equilibrium y_0 for which $f(y_0) = 0$.
2. Use Symbolic Math Toolbox to compute $f'(y_0)$ and derive the linearized small-signal model.
3. Implement both the nonlinear ODE (using `ode45`) and the linearized model (using `ss`) in MATLAB. Compare responses for small initial perturbations.

Problem 4: Controllability and observability

Consider the second-order system with transfer function

$$G(s) = \frac{2s + 5}{s^2 + 3s + 2}.$$

1. Use `tf` to define $G(s)$.
2. Use `ss` to obtain a default state-space realization.
3. Compute controllability and observability matrices using `ctrb` and `obsv`, and determine whether the system is controllable and observable.
4. Convert to a phase-variable companion form using `canon` and check controllability and observability again.

Problem 5: Block diagram simplification using MATLAB

Use the mixed interconnection example from Chapter 2 (series, feedback, and parallel blocks).

1. Implement the three transfer functions G_1 , G_2 , H , and the bypass G_b .
2. Use `series`, `feedback`, and `parallel` to obtain the overall equivalent transfer function.
3. Confirm your result by hand using algebra and compare coefficients.

Problem 6: Cruise control parameter variation

Extend the cruise control design example:

1. Starting from the plant $G_v(s) = 1/(ms + b)$, design three PI controllers with different closed-loop bandwidths (e.g. slow, medium, and fast) using `pidtune`.
2. Use `step` to plot all three speed responses on the same figure.
3. Discuss trade-offs between rise time, overshoot, and control effort (estimate control effort via `lsim` of the control signal).

Problem 7: State-feedback design for a third-order system

Given

$$G(s) = \frac{2s + 5}{s^3 + 4s^2 + 3s + 1},$$

do the following:

1. Obtain a state-space realization using `ss`.
2. Design a state-feedback gain K using `place` to achieve desired closed-loop poles at -1 , -2 , and -3 .
3. Simulate step responses of the open-loop and closed-loop systems, and compare.

Problem 8: LQR vs. pole placement

Using the mass–spring–damper system:

1. Design a state-feedback gain K_{pp} using pole placement with desired poles at $-2 \pm j$.
2. Design an LQR gain K_{lqr} using suitable Q and R .
3. Compare the closed-loop step responses and control signals $u(t)$ for both designs.
4. Discuss which design you prefer and why.

Problem 9: Discretization and digital implementation

Take the first-order RC system with time constant $\tau = 0.1$ s.

1. Build the continuous-time model $G(s) = 1/(\tau s + 1)$.
2. Discretize the model with sampling times $T_s = 0.02$ s and $T_s = 0.2$ s using `c2d`.
3. Compare discrete-time step responses with the continuous-time response.
4. Comment on how the choice of sampling time affects accuracy.

Problem 10: Canonical forms and similarity transformations

For the DC motor from the mini project:

1. Obtain the physical state-space model $(A_{\text{phys}}, B_{\text{phys}}, C_{\text{phys}}, D)$.
2. Convert to phase-variable canonical form $(A_{\text{can}}, B_{\text{can}}, C_{\text{can}}, D)$ using `canon`.
3. Retrieve the transformation matrix T and verify numerically that $A_{\text{can}} = T^{-1}A_{\text{phys}}T$, $B_{\text{can}} = T^{-1}B_{\text{phys}}$, $C_{\text{can}} = C_{\text{phys}}T$.
4. Discuss why multiple realizations correspond to the same input-output behavior.

Closing remarks

If you work through the tutorial sections, the design example, the DC motor mini project, and the 10 practice problems, you will have a solid working knowledge of how to use MATLAB and the Control System Toolbox to implement the concepts from Chapters 1–3 of the control theory course.

Hand-in Requirements for Problems and Mini Project

Each student must submit a written report and the corresponding MATLAB files. The submission should be clear, reproducible, and well organized. This section specifies in detail what must be included.

1. General Format

- The main report must be submitted as a single **PDF** document.
- MATLAB code must be submitted as zip files, which separate .m files are included.
- Use a clear and consistent structure with numbered sections and subsections.
- Figures should be readable (axes labeled, legends included, units indicated).
- All plots should be generated by MATLAB and referenced in the text.
- The files should be sent to homeworkscut@126.com before Dec. 21 2025.

2. Report Structure

The report should follow this structure:

1. Cover Page

- Course name and code.
- Assignment title (e.g., “MATLAB Control Practice: Problems and Mini Project”).
- Student name(s) and ID(s).
- Instructor name.
- Date of submission.

2. Table of Contents

- List all major sections: “Introduction”, “Problem 1”, ..., “Problem 10”, “Mini Project”, “Conclusion”, “Appendix (Code)”.

3. Introduction (1–2 pages)

- Briefly describe the overall goal of the assignment (using MATLAB to implement and analyze control systems from the lecture).
- Summarize what is included in the report: 10 practice problems and one mini project.

4. Problem Solutions (Problems 1–10)

- Create one subsection per problem (e.g., “Problem 1: First-order Thermal System”).
- For each problem, include:
 - (a) *Problem statement (short)*: Restate or summarize the task in your own words.
 - (b) *Modeling steps*: Equations, assumptions, and derivations (e.g., ODE, transfer function, state-space).
 - (c) *MATLAB code snippets*:
 - Show the key code used to solve the problem (not necessarily every line, but enough to understand the approach).
 - Code should be commented to explain important commands and parameters.

- (d) *Simulation results*:
 - Plots of key responses (step response, impulse response, time-domain outputs, pole-zero maps, etc. as required).
 - Each figure must have a figure number, title, axis labels, and legend if needed.
- (e) *Interpretation and discussion*:
 - Briefly explain what the results mean: stability, damping, steady-state error, effect of parameter changes, etc.
 - Connect the results to concepts from the lecture (e.g., type of system, time-domain specifications, controllability).

5. Mini Project: DC Motor Control

- Organize the mini project section into clear subsections. As a minimum, include:
 - (a) *System Description and Specifications*
 - Describe the physical DC motor model (parameters, states, inputs, outputs).
 - State the control objectives and design specifications (e.g., rise time, overshoot, steady-state error, disturbance rejection).
 - (b) *Modeling*
 - Derive the state-space model from the physical equations.
 - Convert to transfer function form in MATLAB and verify consistency.
 - Show the MATLAB commands used to build `ss` and `tf` models.
 - (c) *Initial Controller Design*
 - Explain how you selected initial controller parameters (PI or PID), based on the specifications.
 - If you used time-domain formulas or pole-placement intuition, write down the equations and predicted behavior.
 - Provide the initial MATLAB code for generating the controller and closed-loop system.
 - (d) *Simulation Results and Analysis*
 - Show step responses for speed (and possibly control signal).
 - Use `stepinfo` and `dcgain` to measure rise time, overshoot, settling time, and steady-state error.
 - Compare results with design specifications and predictions.
 - (e) *Parameter Tuning and Justification*
 - Describe how you adjusted controller gains when the initial design did not meet the specifications.
 - Show before/after plots that demonstrate the effect of changing each parameter (e.g., K_p , K_i , K_d).
 - Predict the effect of a parameter change (e.g., “increasing K_i will reduce steady-state error but may increase overshoot”) and then confirm with simulation.
 - (f) *Disturbance Rejection*
 - Describe how you modeled load torque disturbance.
 - Show simulation results of the closed-loop system under disturbance.
 - Comment on how well the controller rejects disturbances and maintains speed.
 - (g) *(Optional) State-Feedback / LQR Design*
 - If completed, include design steps for state-feedback or LQR.

- Compare the performance with the PI/PID controller.
- (h) *Discussion and Conclusions for the Mini Project*
 - Summarize key findings: trade-offs between speed, overshoot, control effort, and robustness.
 - Reflect on how MATLAB tools (e.g., `tf`, `ss`, `step`, `piddtune`, `place`, `lqr`) helped implement the theoretical concepts.

6. Overall Conclusion (0.5–1 page)

- Summarize what you have learned from solving the 10 problems and the mini project.
- Briefly connect the computational work to the theory in the lecture slides (Chapters 1–3).

7. Appendix: MATLAB Code

- Include complete MATLAB scripts/functions in an appendix or refer to attached files.
- Each script should have a header comment explaining its purpose (e.g., “Script for Problem 3: Nonlinear Spring Linearization”).
- Ensure the code is well commented and can be run without modification (set random seeds or parameter values inside the script).

3. MATLAB File Submission

In addition to the PDF report, submit:

- One folder named `Surname_ID_ControlPractice` (replace with your name and ID).
- Inside the folder, include:
 - `Problem1.m`, `Problem2.m`, ..., `Problem10.m` (or clearly named scripts/functions per problem).
 - A main script for the mini project (e.g., `MiniProject_DCMotor.m`) and any helper functions.
 - Optional: a `README.txt` briefly explaining how to run the scripts.
- Ensure all paths are relative (no hard-coded local paths) so that the instructor can run the files directly.

4. Grading Criteria (Indicative)

The assignment will be evaluated according to the following aspects:

- **Correctness of modeling and analysis:**
 - Equations and MATLAB models correctly reflect the physical systems.
 - Controller designs are consistent with specifications.
- **Use of MATLAB and Control System Toolbox:**
 - Appropriate use of functions such as `tf`, `ss`, `step`, `lsim`, `bode`, `ctrb`, `obsv`, `pid`, `piddtune`, `place`, `lqr`, etc.
 - Code is clean, commented, and reproducible.
- **Interpretation and discussion:**

- Results are clearly interpreted in terms of control concepts (stability, transient response, steady-state error, robustness).
- Parameter choices and adjustments are justified logically (not just trial-and-error).
- **Clarity and presentation:**
 - Report is well organized, with clear figures and explanations.
 - All requirements of this section are satisfied.