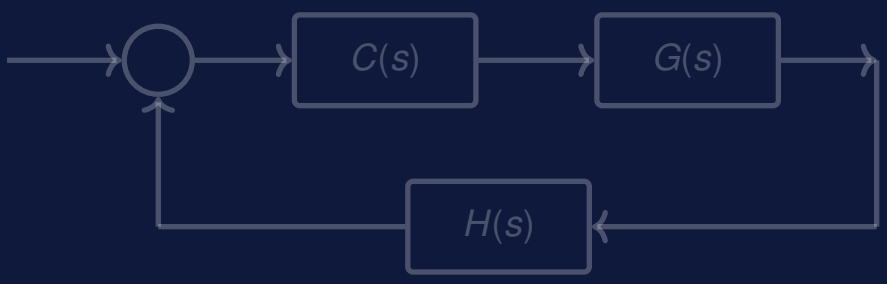
Feedback Control

Modelling, Analysis, and Design with MATLAB/Simulink

Dr İbrahim Küçükdemiral & Dr Geraint Bevan



Feedback Control: Modelling, Analysis, and Design with MATLAB/Simulink

DOI: `10.5281/zenodo.XXXXXXX`
Version: 1.0
Published: 2026
Publisher: Zenodo / CERN, Geneva

Citation:
Küçükdemiral, İ. and Bevan, G. (2026). *Feedback Control: Modelling, Analysis, and Design with MATLAB/Simulink*. Zenodo. `https://doi.org/10.5281/zenodo.XXXXXXX`

Typeset in LaTeX. Cover design by the authors.

Foreword

This book grew out of lecture notes that we have developed and refined over several years of teaching Control Engineering 3 and Control Engineering 4 at Glasgow Caledonian University. These modules serve Year 3 and Year 4 students in Mechanical and Electrical/Electronic Engineering, and the material reflects the scope, pace, and emphasis that we have found most effective in the classroom. Parts of the material also draw on the first author's earlier experience teaching Control Systems, Design of Control Systems, and Multivariable Control Systems at Yıldız Technical University, Istanbul.

The twelve chapters are arranged so that each corresponds roughly to one week of lectures. In a twelve-week term the book can be covered in full; in an eleven-week term the frequency-domain chapter (Chapter 9) can be omitted without loss of continuity, since the state-space chapters that follow do not depend on it — the frequency-domain material can then be deferred to a more advanced module. Each chapter is accompanied by worked examples, MATLAB/Simulink exercises, and end-of-chapter problems. A parallel set of laboratory sessions allows students to apply the theory to progressively more realistic control problems, culminating in a single integrated coursework that ties the entire module together.

We assume that students arrive with a solid grounding in university-level calculus — differentiation, integration, and ordinary differential equations — together with sufficient background in physics to be comfortable with Newton's laws, Kirchhoff's circuit laws, and basic energy concepts. No prior exposure to control theory is required; the book begins from first principles and builds systematically from feedback fundamentals through Laplace-domain analysis to state-space design.

The path we follow is deliberately practical. Time-domain methods and MATLAB verification are emphasised throughout, because we believe that students learn control engineering most effectively when they can see, simulate, and tune the systems they are studying. Frequency-domain techniques are introduced where they add genuine design insight, but the core thread runs from transfer-function modelling (Chapters 1–5), through time-response analysis and stability (Chapters 6–7), to controller design by PID (Chapter 8) and frequency-domain compensation (Chapter 9), and finally to the modern state-space toolkit of controllability, observability, state feedback, and observer-based control (Chapters 10–12).

We are grateful to our students, whose questions and feedback have shaped every chapter, and to our colleagues at Glasgow Caledonian University for their support and encouragement.

İbrahim Küçükdemiral and Geraint Bevan
Glasgow, 2026

Contents

Chapter 1

Introduction to Feedback Control

Learning Objectives

After completing this chapter, you should be able to:

- Distinguish between **open-loop** and **closed-loop** control systems and classify real-world examples into each category.
- Identify the components of a feedback loop: reference, error, controller, actuator, plant, sensor, and disturbance.
- Draw and interpret block diagrams for feedback control systems.
- Derive the closed-loop transfer function for a unity-feedback system and compute the effect of disturbances.
- Explain why feedback attenuates disturbances and improves tracking accuracy, using the cruise control and volume control examples.
- Describe the effect of hysteresis in on–off control and explain its practical purpose.

Why Study Control Engineering?

Control systems are everywhere. From the moment we wake up to the moment we go to sleep, we interact with systems that quietly measure, decide, and adjust in order to achieve a desired outcome. When your alarm clock rings at the correct time, when the central heating keeps a room comfortable, when a lift stops precisely level with the floor, or when a washing machine automatically selects the correct cycle — control is at work behind the scenes.

At its simplest, control engineering is about making things behave in the way we want. It allows us to regulate temperature, speed, position, pressure, voltage, and countless other physical quantities. Without control systems, many everyday tasks would require constant human supervision. With control systems, processes can run automatically, safely, and efficiently.

Control also enables us to go beyond normal human capability. Modern aircraft remain stable in turbulent conditions thanks to automatic flight control systems. Industrial robots position components with millimetre or even micrometre accuracy. Spacecraft navigate millions of kilometres through space with remarkable precision. In medicine, devices such as insulin pumps continuously monitor and regulate blood glucose levels. These achievements are possible because of carefully designed control systems.

Interestingly, control is not purely an engineering concept — it is also fundamental to nature. The human body continuously regulates its internal temperature, blood pressure, and oxygen levels through feedback mechanisms. When you feel cold, your body shivers to generate heat; when you are too warm, you sweat to cool down. This natural process of measurement, comparison, and correction is the same basic principle used in engineered control systems.

In engineering practice, well-designed control systems bring significant benefits. They improve safety by reducing human error, increase efficiency by minimising waste and energy consumption, enhance reliability by compensating for disturbances, and reduce cost through automation. In many cases, a system would simply not function without control.

In this module, you will learn how to model dynamic systems, analyse their behaviour, and design controllers that achieve stability, accuracy, and robustness. Although the mathematical tools may at first seem challenging, they provide a powerful framework for understanding and shaping the behaviour of real-world systems.

By the end of this course, you will be able to analyse a system, predict its behaviour, and design a controller that makes it do what you want.

1.1 Feedback All Around Us

Feedback appears wherever a system measures its current state, compares it with a desired condition, and adjusts accordingly. The examples below illustrate how pervasive this principle is.

Everyday Life

In everyday life, feedback appears in devices that seem ordinary precisely because they regulate themselves so effectively. Voice assistants can adjust their volume or microphone sensitivity when the surrounding noise changes. Elevators reopen their doors when sensors detect that a passenger is still entering, while navigation apps recalculate a route as soon as the driver misses a turn. Traffic lights equipped with sensors adapt their timing to traffic conditions, smart fridges maintain their internal temperature even when the door is opened repeatedly, and fitness watches monitor heart rate and prompt the user to slow down

or speed up. Coffee machines regulate water temperature and pressure while stopping the pour at the right moment, and central heating systems compare the measured room temperature with the thermostat setting before deciding whether to switch the boiler on or off.

Engineering and Technology

In engineering and technology, feedback is even more central. Aircraft autopilots continuously adjust control surfaces to keep the aircraft level and on course despite turbulence. Wind turbines vary blade pitch so that they capture useful energy without overloading the mechanism, and medical ventilators measure oxygen and pressure levels before adjusting airflow to match the patient's needs. Robotic arms in manufacturing rely on sensors to keep welding, assembly, and positioning accurate; electric vehicle battery management systems regulate temperature and charge for safety and efficiency; drones use gyroscopes and accelerometers to stabilise flight; and industrial control systems keep pressure, temperature, and chemical concentration within safe operating limits.

Biological and Natural Systems

Biological and natural systems also depend heavily on feedback. Blood sugar regulation relies on insulin and glucagon acting in opposition to keep glucose within a safe range. Body temperature is stabilised through sweating or shivering, balance during walking is maintained by feedback from the inner ear, muscles, and eyes, and pupil size changes automatically in response to light. At a larger scale, ecosystems exhibit feedback when predator and prey populations influence one another and create recurring population cycles.

Economics and Social Systems

Feedback is not confined to physical systems. In economics and social systems, central banks change interest rates in response to inflation and economic growth, supply and demand interact through price changes, stock markets can amplify movements when investors react to recent trends, and public health policies are tightened or relaxed in response to infection rates, which then alter future transmission.

Traffic and Infrastructure

Traffic and infrastructure provide further examples. Adaptive cruise control changes a vehicle's speed to maintain a safe distance from the car ahead. On motorways, congestion can emerge when small disturbances are amplified through positive feedback. In water reservoir management, release rates are adjusted according to rainfall and storage level so that flooding and shortages can both be avoided.

Climate and Environmental Systems

Climate and environmental systems also involve powerful feedback mechanisms. Ice melt reduces surface reflectivity and increases heat absorption, which is a classic example of positive feedback. By contrast, the carbon cycle contains balancing effects, since plant growth can absorb carbon dioxide and partially offset emissions.

Fun and Unexpected Examples

Even playful or unexpected systems are shaped by feedback. Video game controllers use haptic feedback to influence the player's actions, concert sound systems must suppress unwanted audio feedback to avoid screeching, and social media platforms alter what they show users in response to previous interaction, often reinforcing viewing patterns.

1.2 Classification of Control Systems

Control systems fall into two categories: Open-loop control and Closed-loop control. In an open-loop system, the control action is applied without measuring the output, so performance depends entirely on prior calibration. In a closed-loop system, the output is measured and compared with the reference. The difference (the error) adjusts the control action, making the system more accurate and resistant to disturbances.

1.2.1 Open-Loop Control Systems

An open-loop control system has the general form illustrated in Figure 1.1. Here, the desired input y_d is processed by a Controller to produce a control signal u, which is applied to the Plant via an Actuator. The plant produces the output y.

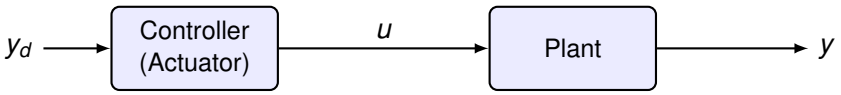


Figure 1.1: Block diagram of an open-loop control system.

Key characteristics of open-loop control: Open-loop control depends strongly on calibration: the controller must be tuned beforehand if the system is to perform correctly. Because the output is not measured, the controller is effectively blind; it has no way of knowing whether the desired result has actually been achieved, and it cannot correct its own mistakes. For the same reason, open-loop systems are sensitive to disturbances and model changes, since any unexpected variation in the plant appears directly at the output.

Open-Loop Control Example (Toaster): A toaster toasts bread by setting a timer, as illustrated in Figure 1.2. The objective is to make the bread golden browned and crisp. However, the toaster does not measure the colour of the bread during the toasting process. For a fixed timer setting, the result can vary: in winter, the toast may remain pale, while in summer it may burn and turn black. This illustrates that calibration is crucial in open-loop systems. A more sophisticated toaster could use sensors to measure the bread's colour and actuators to adjust the timer accordingly, but this would increase both the cost and the complexity of the device.

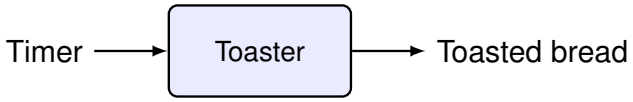


Figure 1.2: A toaster as an open-loop control system.

Figure 1.3: A laundry machine as an open-loop control system.

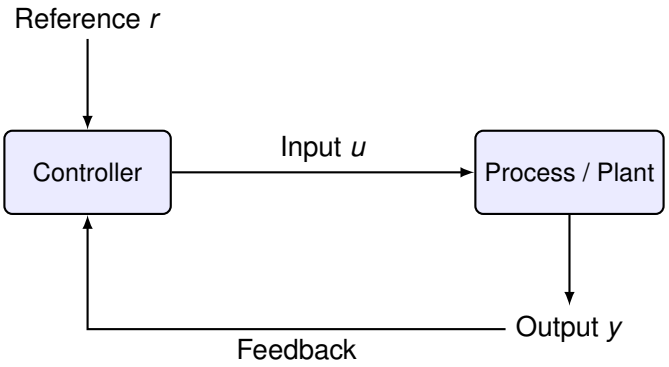


Figure 1.4: The basic structure of a feedback control system.

Open-Loop Control Example (Laundry Machine): A laundry machine washes clothes by running a program selected by the user, as shown in Figure 1.3. The program determines how long the machine agitates, rinses, and spins the clothes. The machine does not measure how clean the clothes actually become during the cycle. As a result, for the same program setting, some clothes may still remain dirty while others may be over-washed, depending on the load size, type of fabric, or how soiled the clothes were initially. This lack of feedback means that the performance of the machine depends entirely on the correct choice of program.

Control without measuring devices such as sensors is therefore called *open-loop control*.

1.2.2 Closed-loop Control : A Common Pattern in All These Examples

Despite their different appearances, all the examples above share a *common structure*.

For instance, in a **thermostat**, the actual room temperature is measured, compared with the temperature you set, and the heating is switched on or off accordingly. In **cruise control**, the car's actual speed is measured, compared with the chosen set speed, and the throttle is adjusted to reduce any mismatch. In the **human body**, internal temperature is monitored against a safe range, and the body responds by sweating or shivering so that the temperature remains close to 36 °C to 37 °C.

What all of these systems have in common is a Reference signal (or set point), a **measurement** of what is actually happening, a **comparison** between desired and actual behaviour, and an **action** that reduces the difference. This is the essential structure of a **feedback control system**.

This simple diagram in Figure 1.4 captures the essence of all the systems we have seen:

Remark 1.1

Feedback compares what we want with what we get, and acts to reduce the gap.

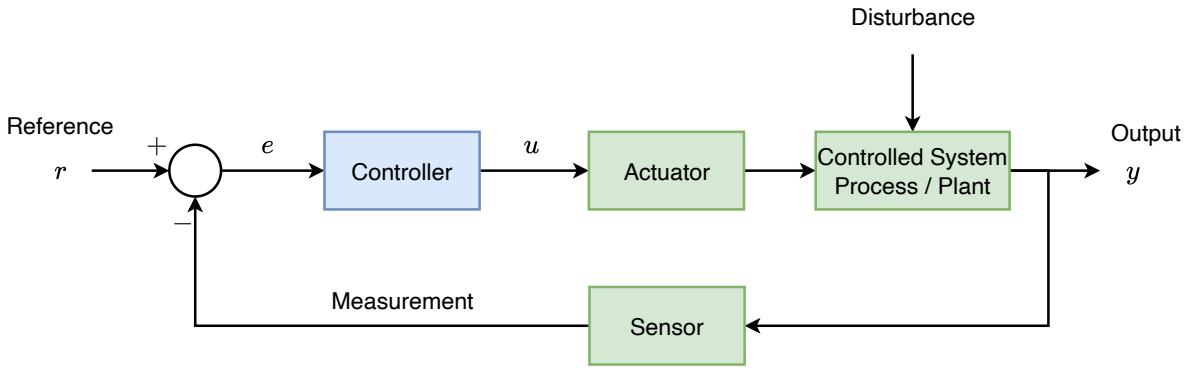


Figure 1.5: Detailed representation of a feedback control system.

1.3 A More Detailed View of Feedback Control

In practice, real feedback systems include physical components beyond the controller and plant. Figure 1.5 provides a more detailed representation.

The error e is formed by comparing the desired reference r with the measured output from the sensor. The controller processes this error and generates a control signal u, which is sent to the actuator. The actuator converts this signal into a physical action that drives the controlled system (the plant). At the same time, the system is affected by disturbances, which may push the output y away from its desired behaviour. The sensor measures the actual output and sends the measurement back for comparison, closing the loop.

The Summing junction is the comparison point, represented by the circular element in Figure 1.5, where the Error signal $e(t) = r(t) - y(t)$ is formed. A thermostat comparator that subtracts the measured room temperature from the user's setting is a simple example. The **controller** is the decision-making element of the loop: it processes the error according to a control law such as PID, LQR, or MPC and determines the control effort u. A drone flight computer that stabilises orientation from IMU data plays this role. The **actuator** then converts that usually low-power command into physical action, such as force, torque, flow, or voltage; examples include an electric motor in a robot joint or a hydraulic valve in an aircraft mechanism. The **plant** is the physical process being regulated, such as the vehicle in a cruise control problem or a heat exchanger in a chemical plant. Disturbance refers to unwanted external influences, such as wind gusts or load changes, that push the plant away from its desired behaviour. Finally, the Sensor closes the loop by measuring the actual output and converting it into a usable signal; a tachometer, a thermometer, or an ultrasonic range sensor all serve this purpose. The quality of feedback control therefore depends not only on the controller itself, but also on the actuators, sensors, and disturbances present in the real system.

Example 1.1: Room Heating System

A familiar case is the heating of a room in a house.

In this example, the reference r is the temperature chosen by the user on the thermostat, for example 21 °C. The error e is the difference between this desired value and the measured room temperature. The controller is the thermostat logic that decides whether to switch the heating system on or off, or, in more advanced designs, how much power to request. The actuator is the boiler or heating unit, the plant is the room and its contents, the disturbances include open windows, outside weather, and heat from people or appliances, and the sensor is the thermometer inside the thermostat that measures the actual room temperature. Figure 1.6 illustrates the complete system.

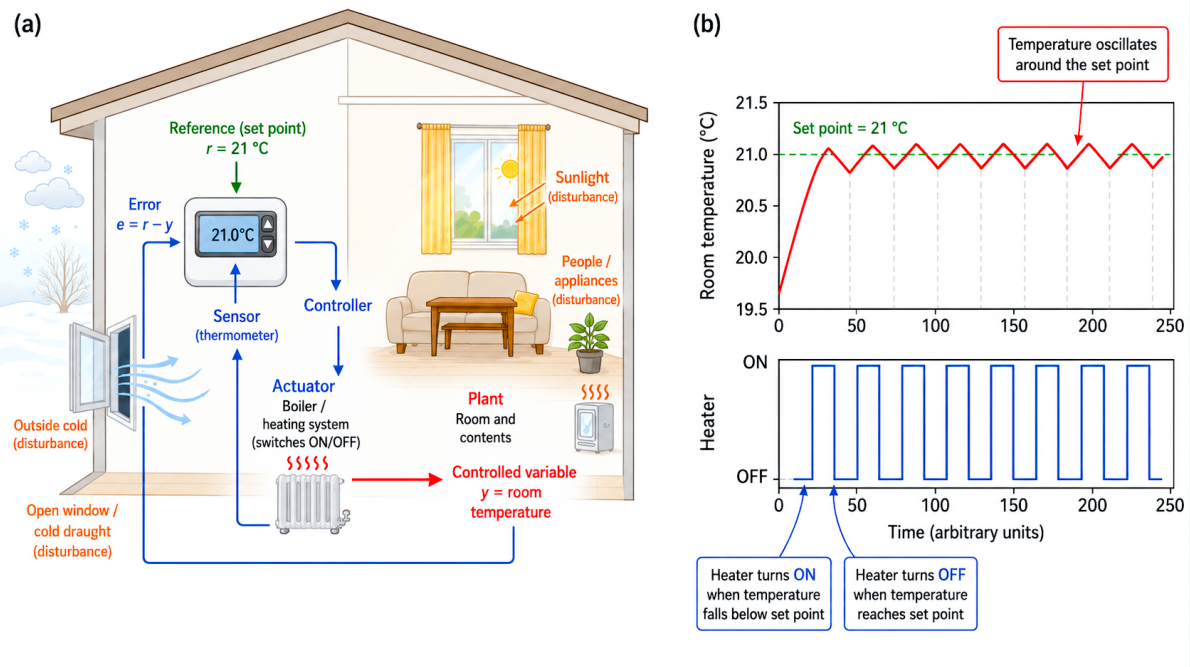


Figure 1.6: Room heating feedback control system. (a) Physical layout: the thermostat (controller) reads the room temperature from its built-in sensor, compares it with the user-set reference of 21 °C, and switches the boiler (actuator) on or off accordingly. The room and its contents form the plant, while open windows, outside weather, sunlight, and occupant activity act as disturbances. (b) Resulting on–off behaviour: the room temperature oscillates around the set point as the heater cycles between on and off states.

In this example, the controller's task is to ensure that the room temperature remains close to the user's chosen set point, despite disturbances such as cold draughts or the heat generated by sunlight. This illustrates how the theoretical components of the feedback diagram correspond directly to real physical elements in a system we encounter daily.

Figure 1.7 shows the variation of the room temperature and the operation of the controller (ON/OFF). Notice how the heater switches on whenever the temperature drops slightly below the set point and switches off as soon as the set point is reached. As a result, the temperature oscillates around the desired value. This type of control is called On–off control, or sometimes *bang–bang control*.

While simple and inexpensive to implement, pure on–off control has an important drawback: the heater may switch very frequently near the set point, which can cause wear and tear on the equipment and waste energy. Moreover, the temperature never settles exactly at the desired value; there is always a small error due to the switching.

A practical improvement is to introduce a **control band** (sometimes called Hysteresis). Instead of switching exactly at the set point, the heater is turned on only when the temperature falls below a lower threshold, and turned off only when it rises above an upper threshold. Figure 1.8 illustrates this approach. The two green dashed lines indicate the band around the set point. Within this band, the heater does not switch, so the controller avoids rapid toggling. The result is smoother operation, reduced wear on the actuator, and greater overall efficiency. Although the temperature still oscillates, the switching is far less frequent and the system performance is significantly improved.

This improvement is useful, but it is only a starting point. In the upcoming chapters, you will learn how more advanced control methods (such as proportional, integral, and derivative control) can achieve far smoother, more accurate, and energy-efficient regulation.

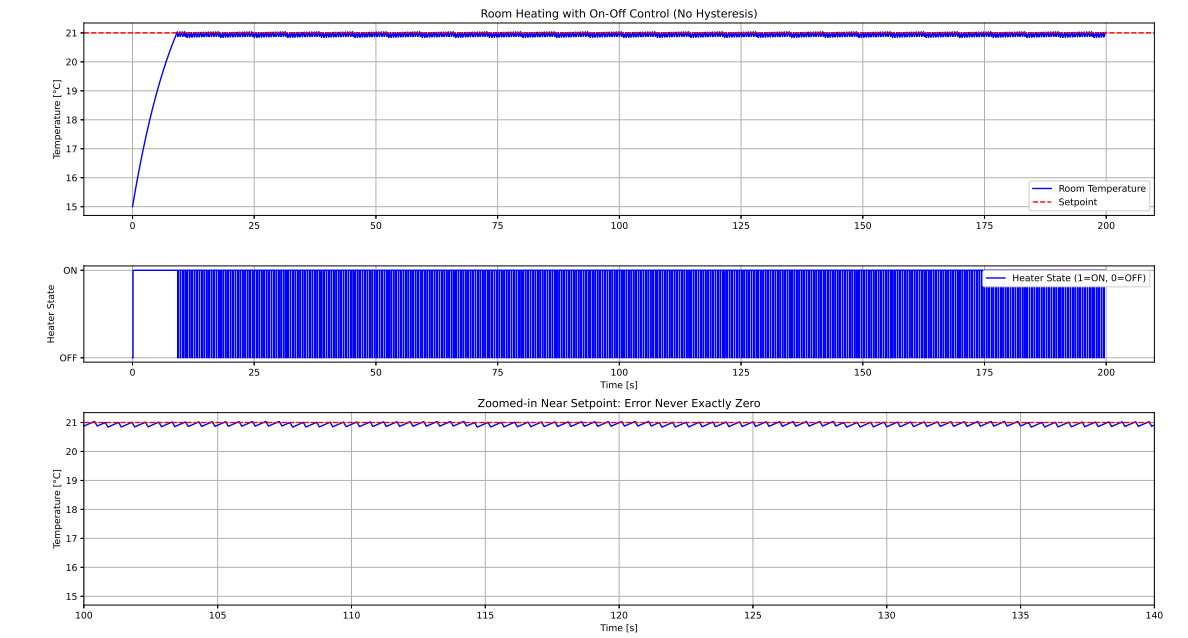


Figure 1.7: Heater example using simple on–off control. The **top plot** shows the variation of room temperature (blue line) compared with the desired set point (red dashed line). The temperature rises quickly and then oscillates about the set point. The **middle plot** shows the heater state (blue step line), where 1 indicates the heater is ON and 0 indicates the heater is OFF. Frequent switching occurs once the set point is reached. The **bottom plot** is a zoomed–in view near the set point, highlighting that the error never reaches exactly zero and small oscillations remain. This demonstrates the basic behaviour of ON–OFF control and motivates the need for improvements such as adding a hysteresis band.

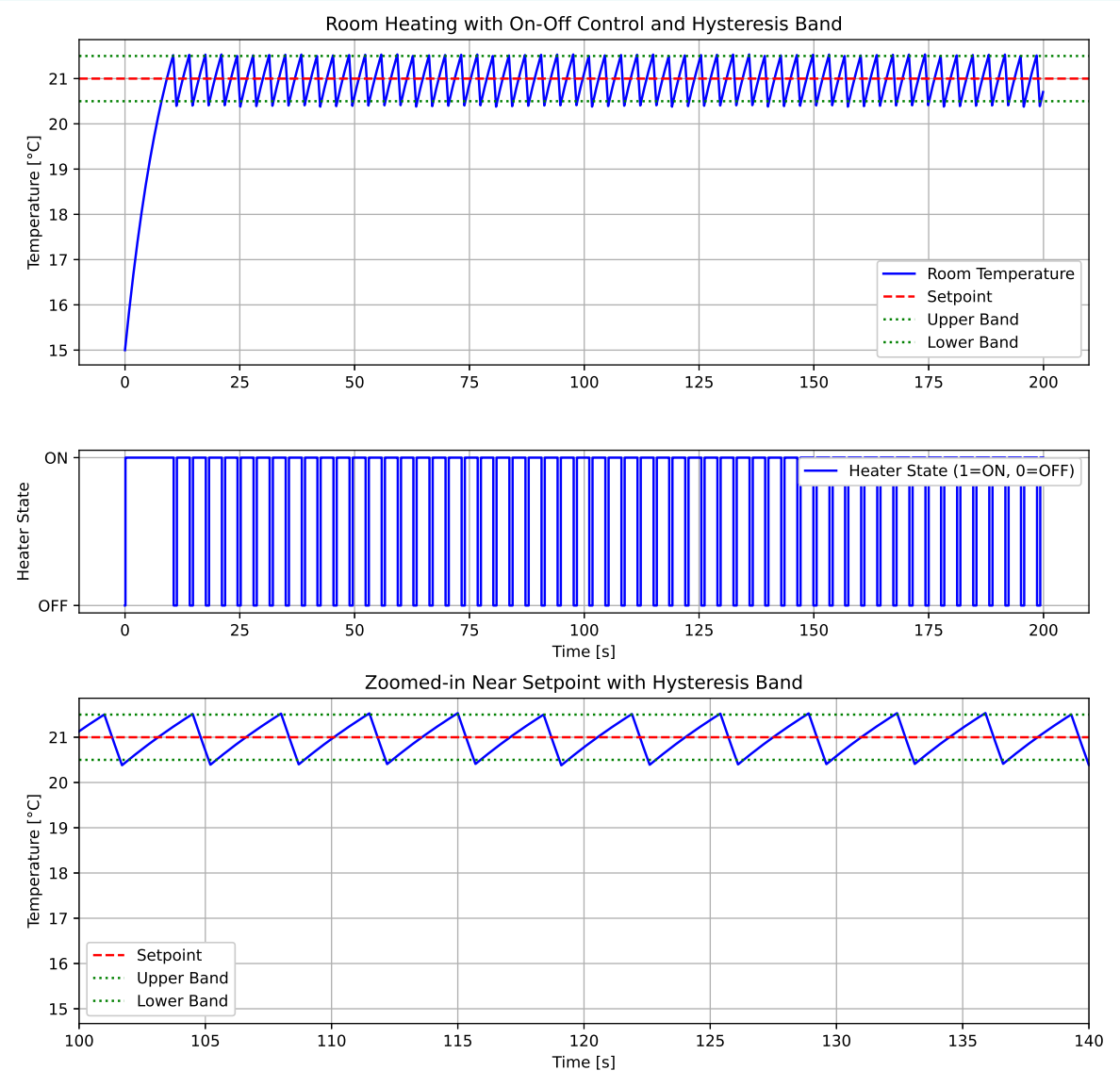


Figure 1.8: Heater example with ON / OFF control and a hysteresis band. The **top plot** shows the room temperature (blue line) compared with the desired set point (red dashed line) and the hysteresis band limits (green dotted lines). The temperature now oscillates within the band instead of switching at the set point, reducing the frequency of control actions. The **middle plot** shows the heater state (blue step line), with fewer ON/OFF transitions compared to pure on–off control, improving efficiency and reducing wear. The **bottom plot** provides a zoomed–in view near the set point, illustrating how the temperature fluctuates smoothly between the upper and lower band limits. This demonstrates how adding a control band achieves a more realistic and practical control behaviour. In practice, almost all household thermostats use a small hysteresis band. This is why you may notice a slight range of variation in room temperature rather than a perfectly constant value.

Example 1.2: Automatic Volume Control in a Car Radio

The Problem: Imagine you are driving on a motorway while listening to the radio. You set the volume to a comfortable level. However, as the speed of the car increases, the road and wind noise also increase. As a result, the music becomes difficult to hear. Without an automatic adjustment system, you would need to manually increase the volume. This is an example of an **open–loop system**.

Open–Loop System: In the open–loop case shown in Figure 1.9, the system has no way of knowing whether the output (sound in the cabin) is adequate. It simply multiplies the driver's volume setting by a fixed amplifier gain. Any additional noise directly affects the output.

$$y = Kr + d$$

where r denotes the driver's desired volume setting, K is the amplifier gain, d represents the disturbance caused by road noise, and y is the resulting sound level in the cabin.

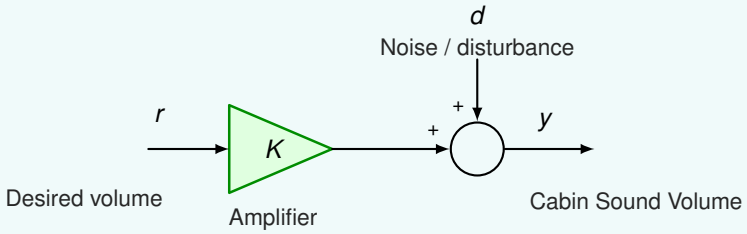


Figure 1.9: Manual control of the sound volume in the cabin.

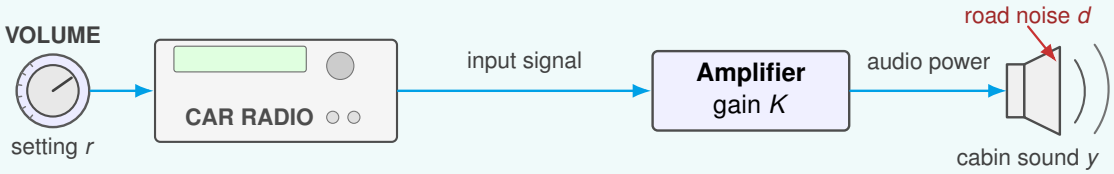


Figure 1.10: Car radio amplifier system.

Closed–Loop System: To improve performance, a small microphone is placed in the cabin to measure the actual sound level y. The measured sound is compared with the desired setting r, and the difference, called the *error* e, is used by the controller to adjust the amplifier gain. This arrangement creates a **feedback system**, also called a **closed–loop system**.

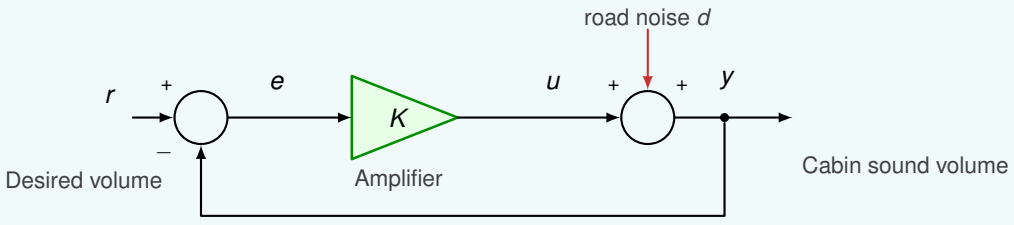


Figure 1.11: Closed–loop volume control system. The microphone measures the cabin sound level y, which is compared with the desired setting r to produce the error e. The controller then adjusts the amplifier input u accordingly.

$$e = r - y, \qquad u = Ke, \qquad y = u + d$$

From these relations we obtain

$$y = \frac{K}{1+K} r + \frac{1}{1+K} d$$

This expression shows two important effects. First, the contribution of the reference r is multiplied by $\frac{K}{1+K}$, which approaches 1 as K becomes large, so increasing the gain improves the accuracy with which the system reproduces the desired volume. Second, the disturbance d is multiplied by $\frac{1}{1+K}$, which becomes small when K is large, so the effect of road noise is attenuated by feedback.

Benefits of Feedback: The system automatically compensates for increased road noise, so the driver no longer needs to adjust the volume manually. The sensitivity to disturbances is reduced by the factor $\frac{1}{1+K}$.

Example 1.3: Cruise Control: Manual vs Feedback Control

We study cruise control — maintaining a desired constant speed in a car — first with **manual (open-loop) control**, then with **feedback (closed-loop) control**.

System Description and Assumptions: We consider the following simplified system model for a car operating around a nominal speed of approximately 55 mph:

A 1% change in throttle position is assumed to produce approximately +10 mph change in speed, while a 1% change in road grade produces approximately -5 mph change in speed. We also approximate the engine dynamics as instantaneous, so that the system can be treated as a static gain in this introductory example.

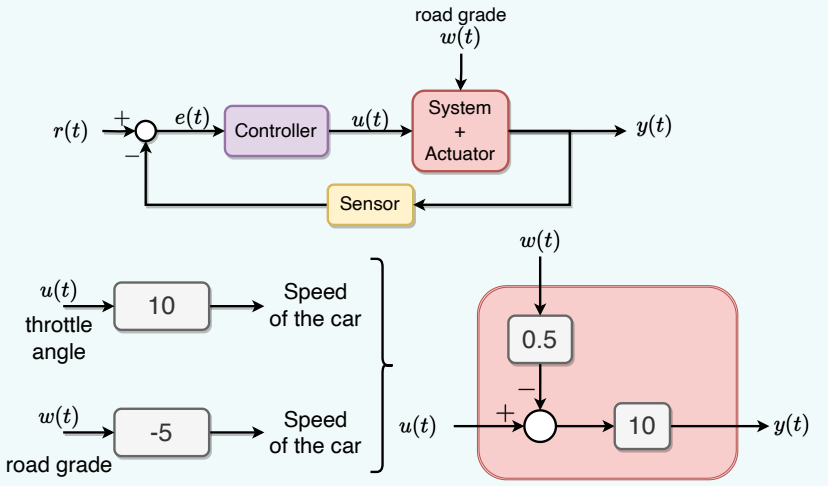


Figure 1.12: Block diagram of the cruise control system with feedback. The reference speed $r(t)$ is compared with the actual speed $y(t)$ to produce the error $e(t)$, which the controller uses to adjust the throttle input $u(t)$. The system and actuator convert the throttle command into vehicle speed, while the road grade $w(t)$ acts as a disturbance that reduces speed. The sensor measures the actual speed and feeds it back to the summing junction. The equivalent system model is shown on the right, leading to the relationship: $y(t) = 10u(t) - 5w(t) = 10[u(t) - 0.5w(t)]$.

The problem is formalised in Figure 1.12 and also visualised in Figure 1.13. Hence, the speed of the car can be modelled as

$$y(t) = 10u(t) - 5w(t),$$

where $y(t)$ denotes the vehicle speed in mph, $u(t)$ is the throttle angle in percent, and $w(t)$ is the road grade in percent.

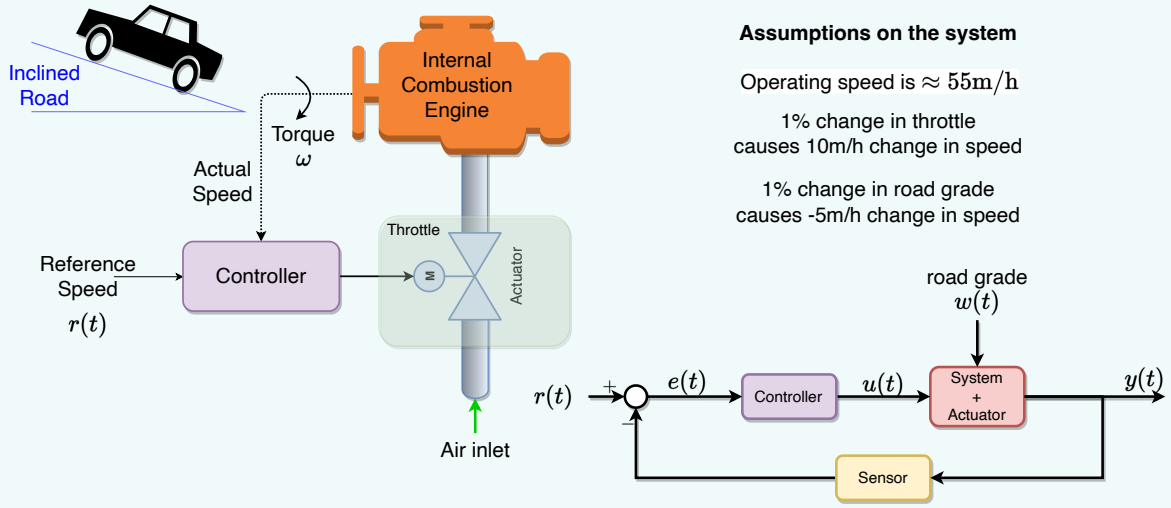


Figure 1.13: Cruise control system for a vehicle. The diagram shows both the physical interpretation (left) and the block diagram representation (right). The reference speed $r(t)$ is compared with the actual speed $y(t)$ to form the error $e(t)$, which the controller uses to adjust the throttle input $u(t)$. The system and actuator convert the throttle command into vehicle speed, while the road grade $w(t)$ acts as a disturbance. A sensor measures the actual speed, closing the feedback loop. The assumptions on throttle sensitivity and slope effects are given on the right.

Case 1: Manual (Open-Loop) Control: In open-loop control, the driver sets the throttle to a fixed value corresponding to the desired speed $r(t)$. There is no feedback from the speed sensor. So the system is completely blind. For example, the driver may use the rule

$$u(t) = \frac{1}{10}\, r(t).$$

to get $y(t) = r(t)$ when there is no disturbance (road flat, no wind, no friction etc.). Then, substituting into the system model gives:

$$y_{OL}(t) = 10 \cdot \frac{1}{10} r(t) - 5w(t) = r(t) - 5w(t).$$

Interpretation: This formula immediately shows the weakness of open-loop control. On a flat road, where $w = 0$, the speed matches the reference exactly. However, if the road grade increases so that $w > 0$, the vehicle speed falls; in this model, each 1% uphill slope reduces the speed by 5 mph. The driver must then notice the change and manually adjust the throttle to recover the desired speed.

Performance Measure: We define the *percentage error* as

$$PE = 100 \cdot \frac{|r(t) - y_{OL}(t)|}{r(t)}.$$

Let $r(t)$ = 55 mph. Table 1.1 shows the open-loop performance for different slopes $w(t)$.

$r(t)$	$w(t)$	$y_{OL}(t)$	Percentage Error (%)
55	0	55	0
55	1	50	9.1
55	2	45	18.2
55	0.1	54.5	0.9

Table 1.1: Open-loop cruise control performance.

We see that performance degrades directly with road slope.

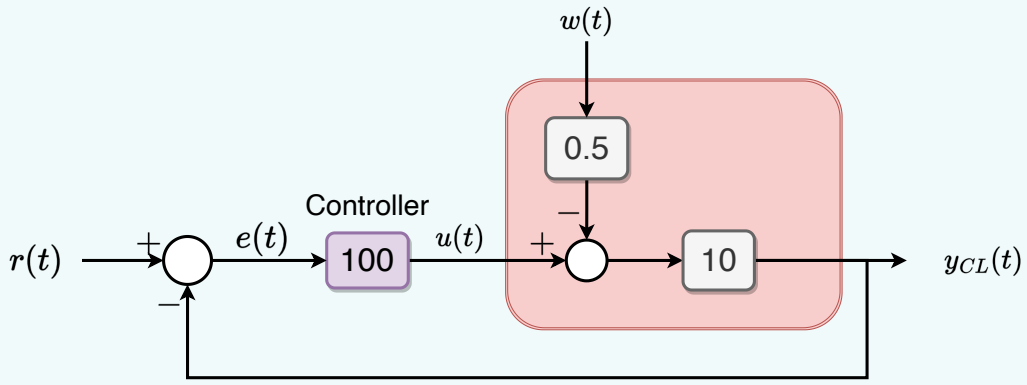
Case 2: Closed-Loop Cruise Control System: To overcome the open-loop sensitivity to disturbances, we now add feedback.

Block Diagram of the Closed-Loop System: Figure 1.14 shows the closed-loop cruise control system. The reference speed $r(t)$ is compared with the actual speed $y_{CL}(t)$ at a summing junction, producing the error signal

$$e(t) = r(t) - y_{CL}(t).$$

This error is amplified by the controller C which is set to 100, which produces the throttle command $u(t)$. The plant P (the system and actuator) then converts the throttle into the vehicle speed $y_{CL}(t)$.

In order to work out this system, one needs to know how to find the gain from the input R to the output Y in a basic feedback interconnection. This is described in the Figure 1.15. Here, a sensor H measures the actual speed and feeds it back to the summing junction which is absent in our system as we assume that the speed measurement is perfect and in the same unit with the reference speed. The road grade $w(t)$ enters the plant as a disturbance.



When $w = 0$ $\quad y_{CL_1} = \dfrac{10 \times 100}{1 + 10 \times 100} r(t)$

When $r = 0$ $\quad y_{CL_2} = \dfrac{-10 \times 0.5}{1 + 10 \times 100} w(t)$

$$y_{CL}(t) = y_{CL_1} + y_{CL_2} = \frac{1000}{1001} r(t) - \frac{5}{1001} w(t)$$

Figure 1.14: Closed-loop cruise control system. The error $e(t)$ is formed by comparing the reference $r(t)$ with the measured speed $y(t)$, and the controller adjusts the throttle $u(t)$. Because the system is linear, we can use the principle of superposition to analyse the effect of each input separately. This shows that the reference is almost perfectly tracked, while the disturbance effect is strongly attenuated by feedback.

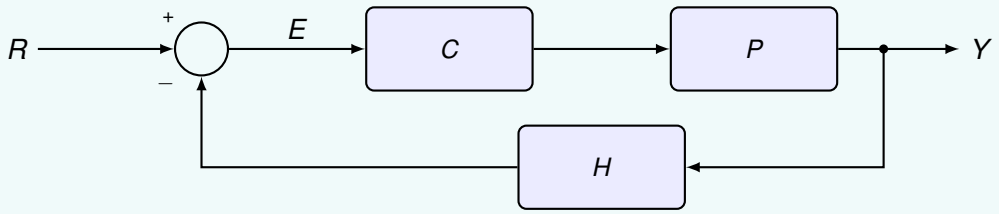


Figure 1.15: Basic feedback interconnection with controller C, plant P, and sensor H.

From Figure 1.15, the output and error signals satisfy

$$Y = P \cdot C \cdot E, \qquad E = R - H \cdot Y.$$

Substituting the first equation into the second:

$$E = R - H \cdot P \cdot C \cdot E \quad \Longrightarrow \quad (1 + HPC)\, E = R \quad \Longrightarrow \quad E = \frac{1}{1 + HPC}\, R.$$

Therefore the Closed-loop transfer function from R to Y is

$$T(s) = \frac{Y(s)}{R(s)} = \frac{P(s)\, C(s)}{1 + H(s)\, P(s)\, C(s)}.$$

Analysis: For our example:

$$P = 10, \quad H = 1, \quad C = K = 100.$$

Thus,

$$y_{CL}(t) = \frac{10 \cdot 100}{1 + 10 \cdot 100} r(t) \; - \; \frac{5}{1 + 10 \cdot 100} w(t).$$

Numerically:

$$y_{CL}(t) = \frac{1000}{1001} r(t) - \frac{5}{1001} w(t).$$

Table 1.2 shows the steady-state vehicle speeds, percentage errors, and system gain for various values of the disturbance $w(t)$.

$r(t)$	$w(t)$	$y_{CL}(t)$	Percentage Error (%)
55	0	54.945	0.10
55	1	54.940	0.11
55	2	54.935	0.12
55	5	54.920	0.15
55	10	54.895	0.19

Table 1.2: Closed-loop cruise control performance with $C = 100$. Even under significant disturbances $w(t)$, the speed remains close to the desired 55 mph with less than 0.2% error.

Comparison with Open-Loop Case: For reference, recall that in the open-loop design:

$$y_{OL}(t) = r(t) - 5w(t).$$

This meant that a 2% slope caused a 10 mph loss in speed, leading to an 18.2% error. In contrast, in the closed-loop design with $C = 100$, the error is reduced below 0.2%, even under large disturbances.

Interpretation: The reference-tracking gain is $\frac{1000}{1001} \approx 0.999$, so the speed nearly equals the desired value. The disturbance gain is $\frac{5}{1001} \approx 0.005$, so road grade becomes almost negligible. Compared with the open-loop case, the percentage error drops from roughly 9%–18% to below 0.2%. This is why we use the feedback interconnection exclusively in this module.

1.4 Examples of Feedback in Other Fields

Figure 1.16 illustrates feedback in two non-engineering domains.

1.4.1 Feedback in Macroeconomics:

In an economy, the government plays the role of a controller. It receives information about the current inflation rate and economic growth (feedback) and compares them with the desired targets. Based on the difference, it adjusts policy tools such as *interest rates* and *taxation*. These control actions influence the behaviour of the national economy, which in turn produces new values for inflation and growth rates, completing the feedback loop. Without such feedback, the economy could easily deviate from its desired path.

1.4.2 Feedback in Ecological Problems:

Think of an ecosystem with rabbits (prey) and foxes (predators). When rabbits are plentiful, foxes have more food and their numbers rise. As foxes increase, they consume more rabbits and the rabbit population falls. With fewer rabbits available, fox numbers then fall, allowing rabbits to recover. This interaction naturally creates repeating rises and falls.

Uncontrolled case: If nobody intervenes, the populations are left to evolve on their own. In Figure 1.17(a) (solid lines), the rabbit population occasionally drops to very low levels, which could be risky in reality.

Controlled case (feedback intervention): Now imagine a manager monitors the rabbits. When rabbit numbers become too low, they provide support (e.g., extra food or habitat help). This is **feedback**: measure what is happening and act only when necessary. In Figure 1.17(a) (dashed lines), the rabbits are prevented from falling as dangerously low, and the overall swings are reduced. Figure 1.17(c) shows the support effort over time: it is applied only when needed and is naturally limited by available resources.

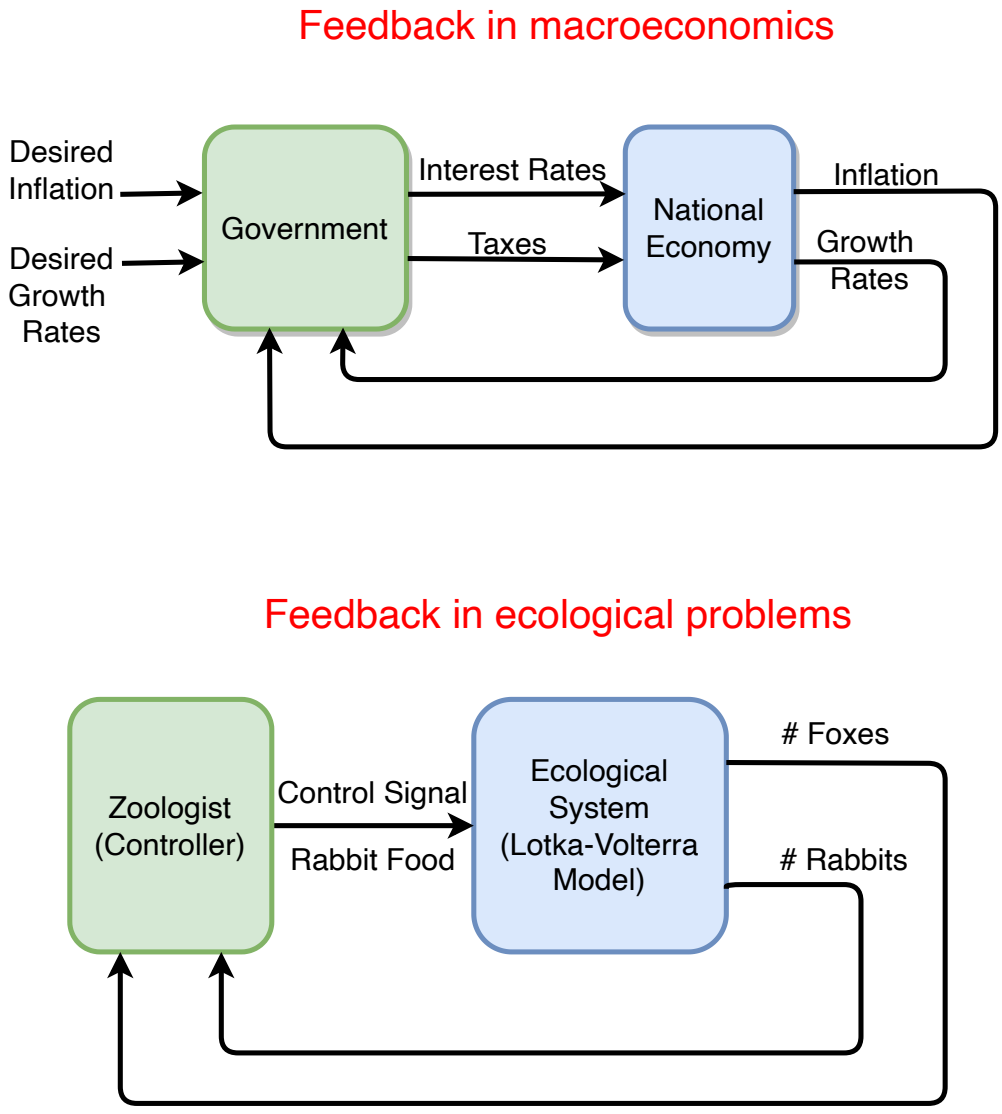
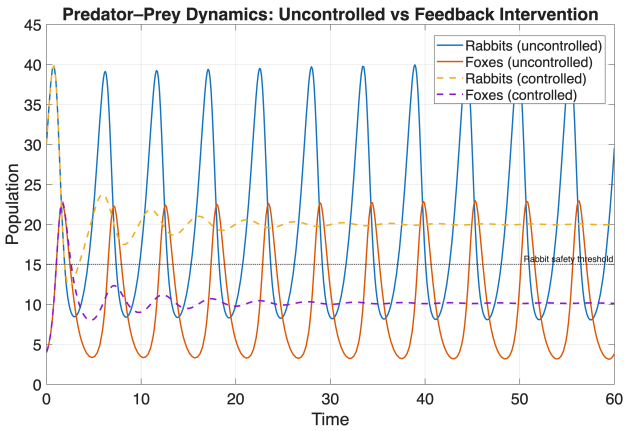
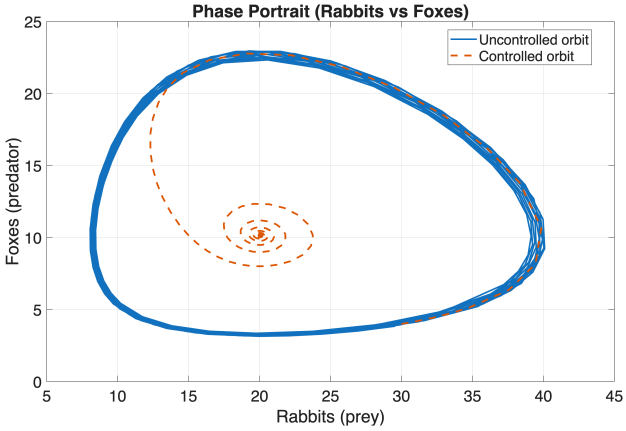


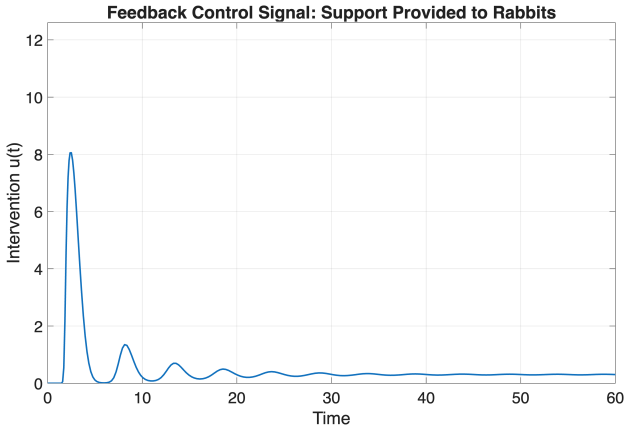
Figure 1.16: Examples of feedback in macroeconomics and in ecological systems.



(a) Populations versus time (solid: uncontrolled, dashed: controlled).



(b) Rabbits vs. foxes (feedback changes the orbit).



(c) Support action over time (applied only when needed).

Figure 1.17: Feedback control idea in a predator–prey ecosystem.

Example 1.4: Feedback Control in Blood Glucose Regulation

An important medical application of feedback is the **artificial pancreas** used by diabetic patients, shown in Figure 1.18.

A continuous glucose monitor (CGM) measures blood glucose every few minutes, a control algorithm computes how much insulin is needed, and an insulin pump delivers it. The objective is to keep blood glucose within its normal range. In control terminology, the blood glucose level $y(t)$ is the controlled variable, the insulin infusion rate $u(t)$ is the manipulated variable, and meals, exercise, and stress act as disturbances. The continuous glucose monitor (CGM) is the sensor, the insulin pump is the actuator, and the patient's glucose–insulin dynamics form the plant. The controller is the algorithm running on a microprocessor, which computes $u(t)$ from the error $e(t) = r(t) - y(t)$, where $r(t)$ is the desired glucose level and $y(t)$ is the measured glucose level. In operation, the CGM measures glucose continuously, the controller compares that measurement with the desired target, and the pump delivers insulin accordingly. In this way, the loop compensates for disturbances such as meals and helps maintain glucose within safe bounds.

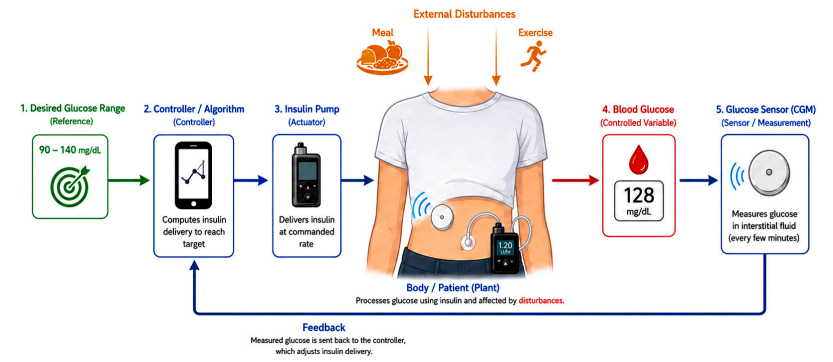


Figure 1.18: Artificial pancreas closed-loop control system. A continuous glucose monitor (CGM) worn on the body measures blood glucose every few minutes and transmits the reading to a controller (typically a smartphone or dedicated device). The controller compares the measured glucose level with the target range (70–180 mg/dL) and computes the required insulin dose. An insulin pump then delivers the computed dose subcutaneously, closing the feedback loop. Meals, exercise, and stress act as disturbances that the loop continuously compensates for.

1.5 Systematic Controller Design Process

Control system design follows four main stages, illustrated in Figure 1.19.

Modelling

The first step is to develop a mathematical model of the physical system (plant). The model captures the essential dynamics of the system, including the effects of actuators, sensors, and possible disturbances. Depending on the application, this can be achieved using physical laws, system identification techniques, or a combination of both.

Analysis

Once the model is established, it is analysed to understand the system's behaviour. This includes studying stability, transient response, and frequency response. Analysis helps to identify system limitations and determine the requirements for control.

Design

Based on the analysis, a controller is designed to meet the desired performance objectives. The controller can be designed using techniques such as PID tuning, root locus, frequency response, or modern state-space methods. The goal is to ensure accuracy, robustness, and efficiency.

Implementation

The designed controller is then implemented, often using digital hardware and software. Practical considerations such as sensor noise, actuator saturation, and computational delays are taken into account. At this stage, the system is tested and, if necessary, the model and controller are refined.

Remark 1.2

The controller design process is iterative. Feedback from the implementation stage may require revisiting the modelling and analysis steps to achieve satisfactory performance.

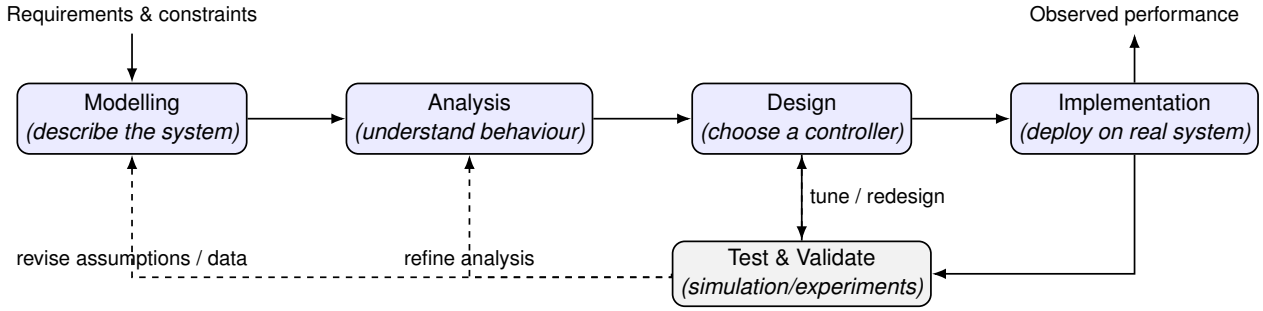


Figure 1.19: Controller design workflow. The process starts with modelling and proceeds through analysis, design, and implementation. Testing and validation may reveal performance gaps, requiring iteration (dashed arrows) back to earlier stages.

1.6 Historical Context of Feedback Control

1.6.1 Early Origins: Automatic Regulation

The roots of feedback control can be traced back to ancient regulation devices. Water clocks (clepsydras) in ancient Greece and China used float regulators to maintain a roughly constant water level, making them early examples of automatic adjustment. During the Islamic Golden Age, Al-Jazari designed mechanical devices such as fountains and clocks that used rudimentary feedback-like mechanisms [1]. A major industrial milestone came in the eighteenth century, when James Watt's **centrifugal governor** (1788) was used to regulate steam-engine speed automatically [13].

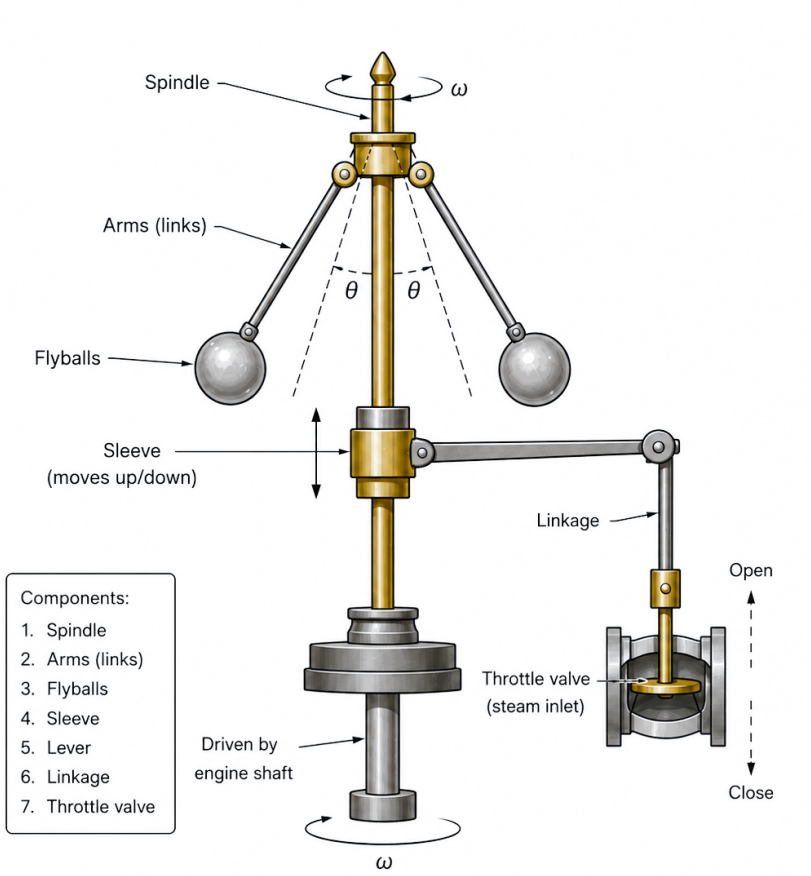


Figure 1.20: James Watt's centrifugal governor (1788). The engine shaft drives the spindle at angular velocity ω, rotating two flyballs on hinged arms. As ω increases, centrifugal force swings the balls outward (increasing the arm angle θ), which raises the sleeve. The sleeve motion is transmitted through a linkage to the throttle valve, partially closing it and reducing steam flow to the engine. Conversely, if the engine slows, the balls drop, the sleeve descends, and the valve opens to admit more steam. This continuous cycle of measurement (spindle speed), comparison (ball position versus equilibrium), and actuation (valve adjustment) constitutes a closed-loop feedback system — one of the earliest in industrial use.

Figure 1.21: Rudolf Emil Kálmán (May 19, 1930 – July 2, 2016) was a Hungarian–American electrical engineer, mathematician, and inventor. He is most noted for his co-invention and development of the Kalman filter, a mathematical algorithm widely used in signal processing, control systems, and guidance, navigation, and control.

Mathematical Foundations

The mathematical study of feedback began in earnest in the nineteenth century. James Clerk Maxwell's 1868 paper on Watt's governor introduced differential equations as a tool for analysing feedback stability [13]. Later, Edward Routh (1877) and Adolf Hurwitz (1895) developed algebraic tests for stability, which together became the well-known *Routh–Hurwitz stability criterion* [17].

The Frequency-Domain Era

In the early twentieth century, telephone and radio engineers, particularly at Bell Labs, were forced to confront instability in amplifiers and communication systems. Harry Nyquist responded by introducing the Nyquist stability criterion in 1932, using frequency-response ideas to determine when feedback systems remain stable [15]. Hendrik Bode then developed gain–phase plots and related design methods in the 1930s and 1940s, tools that are still taught today [2]. During World War II, the urgent demand for accurate gun aiming, radar, and autopilot systems accelerated progress in automatic control even further.

The State-Space Revolution

In the 1960s, Rudolf Kalman introduced **state-space methods**, which made it possible to study multivariable systems and formulate control problems in a much more general way. The **Kalman filter** then provided a powerful method for estimating system states in real time from noisy measurements [9]. From this point onward, control theory became closely linked with linear algebra, estimation, and systems theory.

Modern Control and Beyond

From the 1970s onward, the subject broadened again. **Robust control** emerged in the 1970s and 1980s to deal explicitly with model uncertainty and disturbance rejection, with tools such as H_∞ methods and μ-synthesis [21]. In the 1980s and 1990s, **Model Predictive Control (MPC)** became widely used in process industries such as oil refining and chemical plants [14]. Since the 2000s, control has become deeply integrated with digital computation, data-driven methods, and artificial intelligence. Applications now range from autonomous vehicles and renewable energy systems to biomedical devices such as artificial pancreases, unmanned air and underwater vehicles, and smart grids.

Control has evolved from mechanical regulators in ancient water clocks to advanced algorithms running on microprocessors. Throughout, the central idea has remained the same: measure, compare, and adjust.

MATLAB Companion: Your First Feedback System

Try it yourself

The following short MATLAB script builds the cruise control example from this chapter and compares open-loop with closed-loop behaviour. Type these commands in the MATLAB Command Window or save them as a script (Listing 1.1).

```
% Cruise control: open-loop vs closed-loop comparison
P = 10;             % Plant gain (mph per % throttle)
C = 100;            % Controller gain

% Open-loop: output = P * input (no feedback)
G_ol = tf(P, 1);    % Transfer function: 10/1

% Closed-loop: T = PC / (1 + PC)
G_cl = feedback(G_ol * C, 1);

% Compare step responses
figure;
step(G_ol, G_cl, 5);    % Simulate for 5 seconds
legend('Open-loop (no feedback)', 'Closed-loop (with feedback)');
title('Cruise Control: Open-Loop vs Closed-Loop');
grid on;

% Display the closed-loop transfer function
disp('Closed-loop transfer function:');
G_cl
```

Listing 1.1: Open-loop vs closed-loop cruise control comparison.

Key Takeaways

- Open-loop control acts without measuring the output; closed-loop (feedback) control measures the output and corrects errors automatically.
- Every feedback loop has the same structure: reference, error, controller, actuator, plant, sensor, and disturbance.
- The closed-loop transfer function for a Unity feedback system with loop gain L is $T = \frac{L}{1+L}$, and the disturbance sensitivity is $\frac{1}{1+L}$.
- High loop gain improves tracking accuracy and disturbance rejection, but practical limits (actuator saturation, instability) prevent making the gain arbitrarily large.
- On–off control with hysteresis is the simplest feedback strategy; more advanced methods (PID, state feedback) are introduced in later chapters.
- Feedback appears across engineering, biology, economics, and ecology — the same measure-compare-adjust principle applies everywhere.
- Controller design follows a systematic cycle: model, analyse, design, implement, and validate.

End-of-Chapter Exercises

1. **(Identifying Open-Loop vs Closed-Loop Systems)** For each of the following systems, state whether it operates in *open-loop* or *closed-loop* mode, and briefly justify your answer.
   (a) A washing machine that runs a fixed 40-minute cycle regardless of how clean the clothes are.
   (b) A self-driving car that adjusts its steering based on camera and LIDAR measurements of the road.
   (c) A sprinkler system on a timer that waters the garden every morning at 7 am.
   (d) An oven with a built-in thermostat that switches the heating element on and off to maintain a set temperature.
   (e) A traffic light that changes on a fixed timer with no sensors.
   (f) A modern air conditioning unit that adjusts its fan speed based on a room temperature sensor.
2. **(Feedback Components)** Consider a domestic hot water system where a boiler heats water to a user-selected temperature. A temperature sensor measures the water temperature and a controller switches the burner on or off.
   (a) Identify the *reference*, *sensor*, *controller*, *actuator*, *plant*, and *disturbance* in this system.
   (b) Draw a simple Block diagram of the feedback loop, labelling all signals (r, e, u, y) and components.
   (c) What would happen if the temperature sensor failed and always reported 0 °C? Explain in terms of the error signal.
3. **(Drawing Block Diagrams)** A student uses a desk lamp with a dimmer to read a book. The student observes the brightness on the page (with their eyes), compares it to the brightness they want, and adjusts the dimmer knob accordingly.
   (a) Is this an open-loop or closed-loop system? Justify your answer.

(b) Draw a block diagram of this system, labelling the reference (desired brightness), sensor (eyes), controller (brain), actuator (hand on dimmer), plant (lamp and room), and output (actual brightness on the page).

(c) Suggest one disturbance that could affect this system.

4. **(Cruise Control Calculations — Varying the Controller Gain)** Recall the cruise control example from this chapter. The plant gain is $P = 10$, the sensor gain is $H = 1$, and the system model is $y(t) = 10\,u(t) - 5\,w(t)$. The closed-loop output is:

$$y_{CL}(t) = \frac{10C}{1 + 10C}\,r(t) - \frac{5}{1 + 10C}\,w(t),$$

where C is the controller gain.

(a) For $C = 10$, compute the closed-loop speed y_{CL} when $r = 55$ mph and $w = 2\%$. What is the percentage error?

(b) Repeat the calculation for $C = 50$ and $C = 500$.

(c) Fill in a table comparing the three controller gains ($C = 10,\ 50,\ 500$) in terms of steady-state speed and percentage error for $w = 2\%$.

(d) What happens to the disturbance rejection as C increases? Is there any practical reason we might not want C to be extremely large? (*Hint:* think about what a very large gain does to the control signal $u(t)$.)

5. **(Benefits and Limitations of Feedback)**

(a) List three advantages of closed-loop (feedback) control over open-loop control.

(b) Give one example of a situation where open-loop control might be acceptable or even preferred over closed-loop control.

(c) A friend argues: “Feedback control always makes a system better.” Explain, using the concept of instability, why this statement is not always true.

6. **(Cruise Control — Open-Loop Analysis)** Using the open-loop cruise control model from the chapter:

$$y_{OL}(t) = r(t) - 5\,w(t),$$

(a) If the desired speed is $r = 70$ mph and the road grade is $w = 3\%$, what is the actual speed? What is the percentage error?

(b) What road grade w would cause the open-loop speed to drop to 40 mph when $r = 55$ mph?

(c) Explain in your own words why the open-loop system cannot correct for disturbances.

7. **(Feedback in Everyday Life)** Choose *one* of the following systems and describe, in your own words, how feedback operates within it. Identify the reference, sensor, controller, actuator, plant, and at least one disturbance.

(a) A person balancing on one leg.

(b) A toilet cistern refilling after a flush.

(c) The pupil of your eye adjusting to bright sunlight.

8. **(On–Off Control and Hysteresis)** Consider the room heating example from the chapter, where the set point is 21 °C.

(a) In pure on–off control (no hysteresis), explain why the heater switches very frequently near the set point.

(b) A hysteresis band of ± 1 °C is introduced. At what temperatures does the heater switch on and off?

(c) Explain one advantage and one disadvantage of increasing the hysteresis band to ± 3 °C.

Chapter 2

Mathematical Background and the Laplace Transform

Learning Objectives

After completing this chapter, you should be able to:

- Compute the Laplace transform of common signals (step, ramp, impulse, exponentials, sinusoids) using the definition and standard properties.
- Apply the differentiation, integration, and shifting properties to transform differential equations into algebraic equations.
- Solve initial-value problems using the Laplace transform workflow: transform, solve algebraically, invert.
- Derive the transfer function $G(s) = Y(s)/U(s)$ from a linear ODE under zero initial conditions.
- Perform partial fraction expansion for expressions with distinct real, complex, and repeated poles, and compute the corresponding inverse Laplace transforms.
- Apply the Initial and Final Value Theorems to determine $f(0^+)$ and $\lim_{t\to\infty} f(t)$ directly from $F(s)$.

2.1 Introduction and Roadmap

This chapter introduces the Laplace transform gradually: we first learn what the transform does, then use it to solve differential equations, and only after that interpret complete system responses.

Specifically, we focus on:

- **The Laplace transform:** We begin with the transform pair

$$F(s) = \mathcal{L}\{f(t)\} = \int_0^\infty e^{-st} f(t)\, dt \quad \rightsquigarrow \quad f(t) = \mathcal{L}^{-1}\{F(s)\},$$

  and then formalise the definition and develop the properties that are used most often in control: linearity, differentiation in time, integration, shifting, and standard transform pairs.

- **Solving ODEs via the Laplace transform:** We convert initial-value problems in the time domain into algebraic equations in the complex variable s, solve for $F(s)$, and recover $f(t)$ via the Inverse Laplace transform.
- **Transfer functions:** By applying the Laplace transform to Linear time-invariant (LTI) system systems under zero initial conditions, we derive Transfer functions $G(s)$ and connect them to input–output modelling in control.
- **Impulse response and convolution:** Once the transform machinery is in place, we return to the system viewpoint and show how Impulse response and Convolution fit naturally into the Laplace-domain framework.

To design a control system, we need a dynamic model expressed as differential equations. These equations, together with initial conditions such as $x(0)$ or $\dot{x}(0)$, form an initial value problem (IVP). The Laplace transform converts these time-domain differential equations in $x(t)$ into algebraic equations in $X(s)$. After solving for $X(s)$, we use the inverse Laplace transform to obtain $x(t)$.

This workflow is illustrated in Figure 2.1. The figure also reminds us that a direct time-domain route from the ODE to $x(t)$ does exist; the Laplace-domain path is valuable because it often turns "solve a differential equation" into "solve an algebraic equation", and this shift is one of the main reasons Laplace methods are so widely used in systems and control.

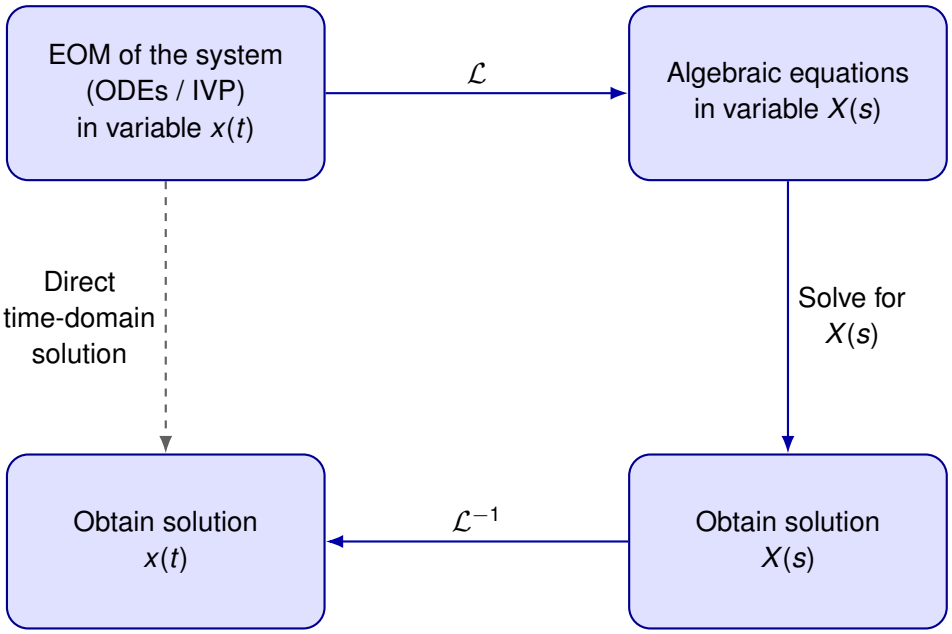


Figure 2.1: Two routes from the governing ODE to the time response $x(t)$: direct time-domain solution, or the Laplace-domain workflow that passes through algebraic equations in $X(s)$.

2.2 Motivation: Why the Laplace Domain Helps

Even for a simple first-order system, the direct time-domain solution can be awkward to manipulate by hand. The following example illustrates the difficulty.

Example 2.1: A motivating RC circuit

We model the circuit in Figure 2.2.

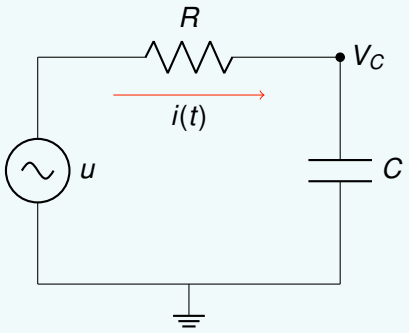


Figure 2.2: Motivating SISO R–C circuit.

For the capacitor, the voltage–current relation is

$$C\,\frac{dV_C(t)}{dt} = i(t),$$

and by KVL around the loop

$$u(t) = R\,i(t) + V_C(t) \quad\Longrightarrow\quad i(t) = \frac{u(t) - V_C(t)}{R} \quad\Longrightarrow\quad \frac{dV_C(t)}{dt} = -\frac{1}{RC}V_C(t) + \frac{1}{RC}u(t).$$

Solving this first-order ODE directly yields

$$V_C(t) = V_C(0)e^{-t/(RC)} + \frac{1}{RC}\int_0^t e^{-(t-\tau)/(RC)}\,u(\tau)\,d\tau. \tag{2.1}$$

The integral term in (2.1) shows that the present output depends on the past history of the input. For more complex inputs $u(t)$, or higher-order systems, repeated time-domain integration quickly becomes cumbersome. The Laplace transform streamlines the process because it:

- replaces $\frac{d}{dt}$ with multiplication by s, turning ODEs into polynomial equations in s;
- uses transform properties and tables to handle many inputs and initial conditions systematically;
- enables block-diagram algebra for combining and simplifying interconnected systems;
- recovers time-domain responses via the inverse Laplace transform when needed.

2.3 The Laplace Transform: Definition and Core Properties

For this chapter, view s as a bookkeeping variable that turns differentiation into algebra. The main skill is learning how to move between the time domain and the s-domain.

2.3.1 Definition, notation, and convergence

Definition 2.1: Laplace transform

Let $f(t)$ be piecewise continuous for $t \geq 0$ and of exponential order. Its (one-sided) Laplace transform is

$$F(s) = \mathcal{L}\{f(t)\} \triangleq \int_0^\infty f(t)\, e^{-st}\, dt,$$

where $s = \sigma + j\omega$ is a complex variable and σ is chosen to ensure convergence.

Conceptually, the transform repackages the signal $f(t)$ into a new form $F(s)$ where differentiation becomes algebraic multiplication.

We write the forward and inverse transforms as

$$F(s) = \mathcal{L}\{f(t)\}, \qquad f(t) = \mathcal{L}^{-1}\{F(s)\},$$

and adopt the convention of lower-case letters for time-domain signals and upper-case for their transforms, e.g. $f(t) \leftrightsquigarrow F(s)$, $y(t) \leftrightsquigarrow Y(s)$. This convention helps keep track of which domain we are working in.

2.3.2 Linearity, differentiation, and integration

Linearity allows us to transform each piece of a composite input separately and combine the results.

Theorem 2.1: Linearity

For any constants a, b and functions $f(t), g(t)$ whose Laplace transforms exist,

$$\mathcal{L}\{af(t) + bg(t)\} = a\,\mathcal{L}\{f(t)\} + b\,\mathcal{L}\{g(t)\}.$$

Proof. Directly from the definition:

$$\begin{aligned}\mathcal{L}\{af(t) + bg(t)\} &= \int_0^\infty \big[af(t) + bg(t)\big] e^{-st}\, dt \\ &= a\int_0^\infty f(t)\, e^{-st}\, dt + b\int_0^\infty g(t)\, e^{-st}\, dt = a\,F(s) + b\,G(s).\end{aligned}$$

□

The second, and arguably most important, property is that a derivative in time becomes an algebraic expression in s. This is the key step that turns an initial-value ODE into an equation that can be solved by ordinary algebra.

Result 2.1: Differentiation in time

Let $f(t)$ be differentiable for $t \geq 0$ with $F(s) = \mathcal{L}\{f(t)\}$. Then

$$\mathcal{L}\left\{\frac{d}{dt}f(t)\right\} = sF(s) - f(0).$$

Proof. Integration by parts with $u = e^{-st}$ and $dv = f'(t)\,dt$:

$$\begin{aligned}\int_0^\infty f'(t)\,e^{-st}\,dt &= \Big[f(t)\,e^{-st}\Big]_0^\infty + s\int_0^\infty f(t)\,e^{-st}\,dt \\ &= 0 - f(0) + s\,F(s) = s\,F(s) - f(0),\end{aligned}$$

where the boundary term at $t \to \infty$ vanishes because $f(t)$ is of exponential order and $\Re(s)$ is chosen large enough. □

Applying the differentiation rule twice gives the second-derivative formula:

$$\mathcal{L}\{\ddot{f}(t)\} = s^2F(s) - s\,f(0) - \dot{f}(0).$$

These formulas will be used repeatedly. Whenever you see $\dot{x}(t)$, replace it by $sX(s) - x(0)$; whenever you see $\ddot{y}(t)$, replace it by $s^2\,Y(s) - sy(0) - \dot{y}(0)$.

The companion property for integration follows immediately. If $g(t) = \int_0^t f(\tau)\,d\tau$, then $\dot{g}(t) = f(t)$ with $g(0) = 0$. Applying the differentiation rule to $g(t)$ gives $s\,G(s) = F(s)$, so

$$\mathcal{L}\left\{\int_0^t f(\tau)\,d\tau\right\} = \frac{F(s)}{s}.$$

There is also a useful companion rule in the opposite direction: multiplication by t in the time domain becomes differentiation with respect to s. This is especially helpful for signals such as te^{-at}, $t\sin(\omega t)$, and higher powers of t multiplying standard signals.

Result 2.2: Differentiation in the s-domain

If $\mathcal{L}\{f(t)\} = F(s)$, then

$$\mathcal{L}\{tf(t)\} = -\frac{dF(s)}{ds}.$$

More generally,

$$\mathcal{L}\{t^nf(t)\} = (-1)^n\frac{d^nF(s)}{ds^n}, \qquad n = 1, 2, \ldots.$$

Proof. Differentiate the definition of $F(s)$ with respect to s:

$$\frac{dF}{ds} = \frac{d}{ds}\int_0^\infty f(t)e^{-st}\,dt = \int_0^\infty f(t)(-t)e^{-st}\,dt = -\mathcal{L}\{tf(t)\}.$$

Repeated differentiation gives the general formula. □

Example 2.1: Using s-domain differentiation

Find the Laplace transforms of te^{-2t} and $t\sin(3t)$.

For the first signal, start from the basic pair

$$e^{-2t} \quad \longleftrightarrow \quad F(s) = \frac{1}{s+2}.$$

Using Result 2.2,

$$\mathcal{L}\{te^{-2t}\} = -\frac{d}{ds}\left(\frac{1}{s+2}\right) = \frac{1}{(s+2)^2}.$$

This is much quicker than evaluating the integral $\int_0^\infty te^{-2t}e^{-st}\,dt$ directly.

For the second signal, use

$$\sin(3t) \quad \longleftrightarrow \quad F(s) = \frac{3}{s^2+9}.$$

Therefore

$$\mathcal{L}\{t\sin(3t)\} = -\frac{d}{ds}\left(\frac{3}{s^2+9}\right) = \frac{6s}{(s^2+9)^2}.$$

In control problems, terms of this type appear when repeated poles or polynomially weighted transient responses occur.

2.3.3 Exponential shifting and trigonometric transforms

The exponential shifting property lets us reuse a known transform by shifting the variable s, rather than evaluating a new integral each time.

Result 2.3: Frequency shift (multiplication by an exponential)

If $\mathcal{L}\{f(t)\} = F(s)$, then

$$\mathcal{L}\{e^{at}f(t)\} = F(s-a).$$

Proof.

$$\mathcal{L}\{e^{at}f(t)\} = \int_0^\infty f(t)\,e^{at}\,e^{-st}\,dt = \int_0^\infty f(t)\,e^{-(s-a)t}\,dt = F(s-a). \qquad \square$$

As a direct application, taking $f(t) = u_0(t)$ with $F(s) = 1/s$ gives the transform of the exponential:

$$\mathcal{L}\{e^{at}\} = \frac{1}{s-a}, \qquad \Re(s) > \Re(a). \tag{2.2}$$

The same formula works for complex exponents: setting $a = \alpha + j\omega$ gives $\mathcal{L}\{e^{(\alpha+j\omega)t}\} = \frac{1}{(s-\alpha-j\omega)}$.

Trigonometric transforms. Using Euler's identities,

$$\cos\theta = \frac{e^{j\theta}+e^{-j\theta}}{2}, \qquad \sin\theta = \frac{e^{j\theta}-e^{-j\theta}}{2j},$$

we can derive the Laplace transforms of sine and cosine from (2.2). Oscillatory behaviour appears everywhere in systems and control — vibrating structures, electrical circuits, rotating machines, and underdamped closed-loop responses — so these results are fundamental.

Result 2.4: Laplace transform of cosine

$$\mathcal{L}\{\cos(\omega t)\} = \frac{s}{s^2 + \omega^2}.$$

Proof. By Euler's identity and linearity (Theorem 2.1),

$$\mathcal{L}\{\cos(\omega t)\} = \frac{1}{2}\mathcal{L}\{e^{j\omega t}\} + \frac{1}{2}\mathcal{L}\{e^{-j\omega t}\} = \frac{1}{2}\left(\frac{1}{s - j\omega} + \frac{1}{s + j\omega}\right) = \frac{s}{s^2 + \omega^2}.$$ □

The sine transform follows by the same approach:

Result 2.5: Laplace transform of sine

$$\mathcal{L}\{\sin(\omega t)\} = \frac{\omega}{s^2 + \omega^2}.$$

Proof. Using $\sin(\omega t) = \frac{1}{2j}\left(e^{j\omega t} - e^{-j\omega t}\right)$ and linearity,

$$\mathcal{L}\{\sin(\omega t)\} = \frac{1}{2j}\left(\frac{1}{s - j\omega} - \frac{1}{s + j\omega}\right) = \frac{1}{2j} \cdot \frac{2j\omega}{s^2 + \omega^2} = \frac{\omega}{s^2 + \omega^2}.$$ □

2.3.4 Initial and final value theorems

Often we only need to know how a response starts and where it settles, not the entire time history. The Initial value theorem and Final value theorem provide these limiting values directly from $F(s)$.

Theorem 2.2: Final value theorem

If $f(t)$ has Laplace transform $F(s)$, and all poles of $sF(s)$ have negative real parts, then

$$\lim_{t \to \infty} f(t) = \lim_{s \to 0} sF(s).$$

Proof. The differentiation property gives $\int_0^\infty f'(t)\, e^{-st}\, dt = sF(s) - f(0)$. Taking the limit as $s \to 0^+$ (permitted because the poles of $sF(s)$ lie in the open left half-plane):

$$\int_0^\infty f'(t)\, dt = \lim_{s \to 0}\big[sF(s) - f(0)\big].$$

The left side equals $\lim_{T \to \infty}[f(T) - f(0)]$. Cancelling $f(0)$ from both sides gives the result. □

Example 2.2: Final value theorem

Let $F(s) = \dfrac{5}{s(s^2+3s+2)}$. The poles of $sF(s) = \frac{5}{s^2+3s+2} = \frac{5}{(s+1)(s+2)}$ are at $s = -1$ and $s = -2$, both in the left half-plane. Therefore

$$\lim_{t\to\infty} f(t) = \lim_{s\to 0} sF(s) = \frac{5}{2}.$$

When the Final Value Theorem cannot be used

The Final Value Theorem is only valid when all poles of $sF(s)$ lie strictly in the left half-plane. If $sF(s)$ has poles on the imaginary axis or in the right half-plane, the time response does not settle to a finite constant, and the theorem must not be applied.

For example, let

$$F(s) = \frac{1}{s^2+1}.$$

Then $f(t) = \sin t$, which oscillates forever and has no final value. Although

$$\lim_{s\to 0} sF(s) = \lim_{s\to 0} \frac{s}{s^2+1} = 0,$$

this number is not a valid final value because $sF(s) = s/(s^2+1)$ has poles at $s = \pm j$.

Listing 2.1 shows how to apply the Final Value Theorem symbolically in MATLAB.

```
syms s
F = 5 / (s * (s^2 + 3*s + 2));
fv = limit(s * F, s, 0);
```

Listing 2.1: MATLAB code for applying the Final Value Theorem (Example 2.2).

Theorem 2.3: Initial value theorem

If $f(t)$ has Laplace transform $F(s)$ and $F(s)$ is strictly proper (degree of the numerator less than the degree of the denominator), then

$$\lim_{t\to 0^+} f(t) = \lim_{s\to\infty} sF(s).$$

Proof. Again from the differentiation property, $\int_0^\infty f'(t)\, e^{-st}\, dt = sF(s) - f(0)$. Now let $s \to \infty$: for $\Re(s)$ large, the factor e^{-st} forces the integral to zero. Hence

$$0 = \lim_{s\to\infty} \big[sF(s) - f(0)\big] \quad\Longrightarrow\quad f(0^+) = \lim_{s\to\infty} sF(s). \qquad \square$$

2.4 Common Laplace Transforms for Signals Used in Control

This section collects Laplace transforms of signals that recur throughout control engineering. The aim is to recognise these signal families and understand why they appear so often.

2.4.1 Standard test inputs: step, ramp, and parabolic

These signals represent progressively more demanding reference inputs and will be central to the tracking and steady-state error analysis in later chapters.

Step signal. A unit step is defined by

$$u_0(t) = \begin{cases} 1, & t \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

We use $u_0(t)$ for the unit step to avoid confusion with $u(t)$, which denotes the control input throughout this book. Some texts use $\mathbf{1}(t)$ or $\mathcal{H}(t)$ (Heaviside function) instead.

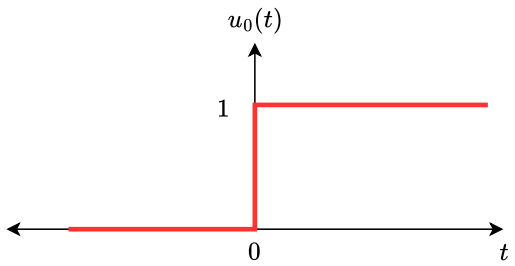


Figure 2.3: Unit-step signal $u_0(t)$.

Its Laplace transform follows directly from the definition:

$$\mathcal{L}\{u_0(t)\} = \int_0^\infty e^{-st} dt = \left[-\frac{e^{-st}}{s} \right]_0^\infty = \frac{1}{s}, \qquad \Re(s) > 0.$$

The unit-step signal can be generated in MATLAB as shown in Listing 2.2.

```
t = 0:0.01:10;
A = 1;
r = A*(t >= 0);
plot(t, r);
```

Listing 2.2: MATLAB code for the unit-step signal $r(t) = A\,u_0(t)$.

Ramp signal. Let $r(t) = At$ for $t \geq 0$. Then

$$\mathcal{L}\{r(t)\} = \int_0^\infty At\, e^{-st} dt = \frac{A}{s^2}, \qquad \Re(s) > 0.$$

The corresponding MATLAB code is given in Listing 2.3.

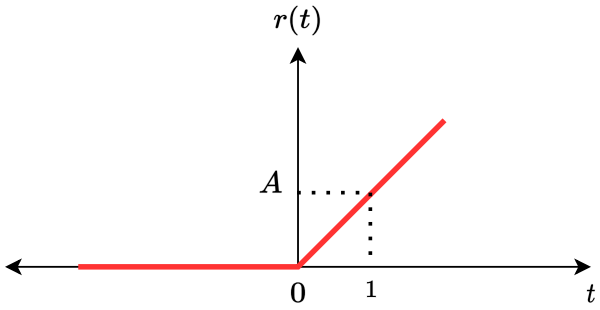


Figure 2.4: Ramp signal.

```
t = 0:0.01:10;
A = 1;
r = A*t;
plot(t,r);
```

Listing 2.3: MATLAB code for the ramp signal.

Parabolic signal. Let $r(t) = \frac{A}{2}t^2$ for $t \geq 0$. Then

$$\mathcal{L}\{r(t)\} = \frac{A}{s^3}, \qquad \Re(s) > 0.$$

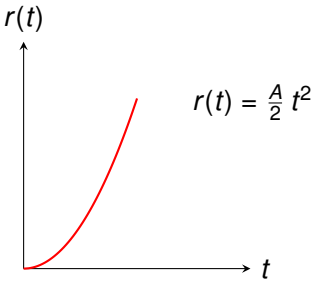


Figure 2.5: Parabolic signal $r(t) = \frac{A}{2}t^2$.

Listing 2.4 generates the parabolic signal in MATLAB.

```
t = 0:0.01:10;
A = 1;
r = (A/2).*t.^2;
plot(t,r);
```

Listing 2.4: MATLAB code for the parabolic signal.

These three signals reveal a useful pattern: each extra power of t in the time domain produces one extra factor of $1/s$ in the Laplace domain. The general formula is

$$\mathcal{L}\{t^n\} = \frac{n!}{s^{n+1}}, \qquad n = 0, 1, 2, \ldots$$

which gives $1/s$ for $n = 0$ (step), $1/s^2$ for $n = 1$ (ramp), and $2/s^3$ for $n = 2$ (parabolic with $A = 2$). This pattern will become important later when we interpret system type and tracking performance.

2.4.2 The Dirac delta (impulse) and the sifting property

The Dirac delta $\delta(t)$ represents an idealised very short input with finite total effect. Although no real actuator produces a perfect impulse, this idealisation is useful because it reveals the intrinsic dynamics of a system.

We can build intuition by starting with unit-area rectangular pulses:

$$r_\varepsilon(t) = \begin{cases} \dfrac{1}{2\varepsilon}, & -\varepsilon \le t < \varepsilon, \\ 0, & \text{otherwise}, \end{cases} \qquad \int_{-\infty}^{\infty} r_\varepsilon(t)\, dt = 1,$$

and letting $\varepsilon \to 0^+$, which yields $\delta(t)$ in the sense of distributions.

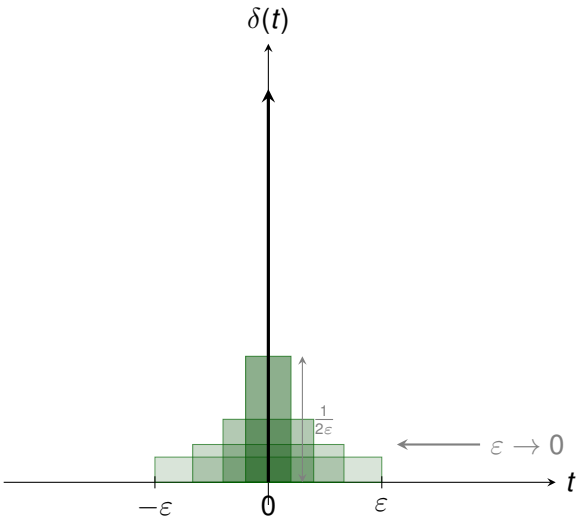


Figure 2.6: Obtaining the Dirac impulse $\delta(t)$ as the limit of unit-area rectangular pulses. Each pulse has width 2ε and height $1/(2\varepsilon)$, so the area is always 1. As $\varepsilon \to 0$, the pulses become narrower and taller, approaching $\delta(t)$.

Theorem 2.4: Sampling (sifting) property

For any continuous function $f(t)$,

$$\int_{-\infty}^{\infty} \delta(t - t_0)\, f(t)\, dt = f(t_0).$$

Informal justification. Replace $\delta(t - t_0)$ by the rectangular pulse $r_\varepsilon(t - t_0)$. For small ε, the function $f(t)$ is approximately constant over the narrow interval $[t_0 - \varepsilon,\ t_0 + \varepsilon]$, so

$$\int_{-\infty}^{\infty} r_\varepsilon(t - t_0)\, f(t)\, dt \ \approx\ f(t_0) \int_{-\infty}^{\infty} r_\varepsilon(t - t_0)\, dt = f(t_0) \cdot 1.$$

Taking $\varepsilon \to 0^+$ makes this exact. □

This property explains why the delta function is so powerful: it "picks out" the value of a signal at one instant. That is precisely why the impulse will later become the natural input for defining impulse response.

Using Theorem 2.4, the Laplace transform of the impulse is

$$\mathcal{L}\{\delta(t)\} = \int_0^{\infty} \delta(t) e^{-st} dt = e^{-s\cdot 0} = 1.$$

For a shifted impulse at $t = a > 0$, the sifting property gives $\mathcal{L}\{\delta(t-a)\} = e^{-sa}$. This is a special case of a more general result about time delay:

Result 2.6: Time-shifting theorem

If $\mathcal{L}\{f(t)\} = F(s)$ and $a > 0$, then

$$\mathcal{L}\{f(t-a)\,u_0(t-a)\} = e^{-as}\,F(s).$$

In words: delaying a signal by a seconds multiplies its transform by e^{-as}.

Proof. Substitute $\tau = t - a$ (so $t = \tau + a$, $dt = d\tau$):

$$\int_0^\infty f(t-a)\,u_0(t-a)\,e^{-st}\,dt = \int_0^\infty f(\tau)\,e^{-s(\tau+a)}\,d\tau = e^{-as}\int_0^\infty f(\tau)\,e^{-s\tau}\,d\tau = e^{-as}F(s). \qquad \square$$

This result is important whenever sensing, actuation, communication, or transport delays are present in a control system. Listing 2.5 illustrates how MATLAB's Symbolic Math Toolbox handles the Dirac delta and its Laplace transform.

```
syms t
d = dirac(t);
pretty(d)

L = laplace(d);
Lshift = laplace(dirac(t - 2));
```

Listing 2.5: MATLAB code to represent $\delta(t)$, $\delta(t-2)$, and their Laplace transforms.

2.4.3 Exponentially-modulated sinusoids

Exponentially-modulated sinusoids arise in underdamped responses, vibration, AC circuits, and modal analysis. The frequency-shift property (Result 2.3) gives:

$$\mathcal{L}\{e^{-at}\sin(\omega t)\} = \frac{\omega}{(s+a)^2+\omega^2}, \qquad \mathcal{L}\{e^{-at}\cos(\omega t)\} = \frac{s+a}{(s+a)^2+\omega^2}.$$

Example 2.3: Exponentially-modulated sinusoid

Consider

$$x(t) = e^{at}\sin(\omega t).$$

Using the frequency-shift property (Result 2.3) applied to Result 2.5, we obtain

$$X(s) = \frac{\omega}{(s-a)^2+\omega^2}.$$

For a decaying (stable) signal, $a < 0$. Example responses are shown in Figure 2.7.

Listing 2.6 reproduces these three cases in MATLAB.

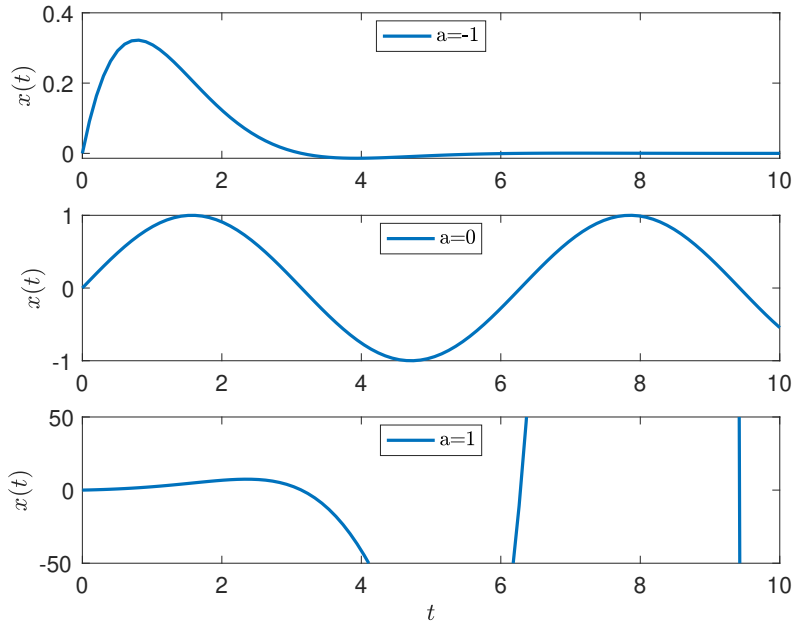


Figure 2.7: Top: exponentially decaying sinusoid ($a < 0$); middle: non-decaying sinusoid ($a = 0$); bottom: exponentially growing sinusoid ($a > 0$).

```
t = 0:0.01:10;
omega = 1;
a_values = [-1, 0, 1];
figure;

for i = 1:length(a_values)
  a = a_values(i);
  x = exp(a*t) .* sin(omega*t);
  subplot(3,1,i);
  plot(t, x, 'LineWidth', 1.5);
  xlabel('t'); ylabel('x(t)');
end

```

Listing 2.6: MATLAB code to display the exponentials in Example 2.3.

Listing 2.7 demonstrates how to construct a time-delayed signal in MATLAB using the unit-step factor to enforce causality.

```
t = 0:0.01:10;
original_function = exp(-t/2) .* sin(3*t);
tau = 2;
delayed_function = exp(-(t-tau)/2) .* sin(3*(t-tau)) .* (t>=tau);

```

Listing 2.7: MATLAB code to generate a time-delayed function $f(t-\tau)$ from $f(t)$.

When a time delay is present, the delayed signal must be interpreted carefully: the waveform is shifted to the right, and a unit-step factor is used to keep the signal causal. In control, that detail matters because a delayed response should not appear before the delay time has actually elapsed.

2.5 Solving ODEs with the Laplace Transform

The method follows a short recipe:

1. write the differential equation and the initial conditions clearly;
2. take the Laplace transform of each term, using Result 2.1 for derivatives;
3. solve the resulting algebraic equation for the unknown transform $X(s)$ or $Y(s)$;
4. rewrite the answer in a form that matches known transform pairs, then apply $\mathcal{L}^{-1}$.

Example 2.2: A first complete Laplace-solution workflow

Consider the initial-value problem

$$\dot{x}(t) + 2x(t) = u_0(t), \qquad x(0) = 1.$$

Taking the Laplace transform of both sides gives

$$\mathcal{L}\{\dot{x}(t)\} + 2\mathcal{L}\{x(t)\} = \mathcal{L}\{u_0(t)\}.$$

Using Result 2.1 and $\mathcal{L}\{u_0(t)\} = 1/s$,

$$\big(sX(s) - x(0)\big) + 2X(s) = \frac{1}{s}.$$

Substituting $x(0) = 1$ gives

$$(s+2)X(s) - 1 = \frac{1}{s},$$

so

$$(s+2)X(s) = 1 + \frac{1}{s} = \frac{s+1}{s} \quad \Longrightarrow \quad X(s) = \frac{s+1}{s(s+2)}.$$

Now rewrite $X(s)$ using simple partial fractions:

$$\frac{s+1}{s(s+2)} = \frac{A}{s} + \frac{B}{s+2}.$$

Matching coefficients gives $A = B = \frac{1}{2}$, hence

$$X(s) = \frac{1/2}{s} + \frac{1/2}{s+2}.$$

Applying the inverse Laplace transform,

$$x(t) = \frac{1}{2}u_0(t) + \frac{1}{2}e^{-2t}u_0(t).$$

For $t \geq 0$, we usually write this simply as

$$x(t) = \frac{1}{2} + \frac{1}{2}e^{-2t}.$$

This example is worth remembering because it shows the core pattern of the method: after the transform is taken, the differential equation disappears and we solve an ordinary algebraic equation instead.

The important point is that the answer above depends on both the input and the initial condition. In control, however, we often want a model that captures the system dynamics themselves, independently of a particular experiment. That leads naturally to the idea of a transfer function.

2.6 From Differential Equations to Transfer Functions

A transfer function describes the input–output dynamics of an LTI system itself, independent of any particular experiment. Transfer functions are defined under zero initial conditions so that the model reflects the system dynamics, not a particular starting state.

Definition 2.2: Transfer function

The transfer function of a linear time-invariant system is defined as

$$G(s) = \frac{Y(s)}{U(s)},$$

where $Y(s)$ is the Laplace transform of the output and $U(s)$ is the Laplace transform of the input, assuming zero initial conditions.

Once $G(s)$ is known, different inputs can be analysed by simple multiplication in the Laplace domain rather than by re-deriving the differential equation each time.

$$Y(s) = G(s)\,U(s)$$

Figure 2.8: Block-diagram representation of a SISO LTI system in the Laplace domain.

Example 2.4: Finding a transfer function

Consider the dynamic model

$$\ddot{y}(t) + 5\dot{y}(t) + 4y(t) = u(t), \qquad y(0) = \dot{y}(0) = 0.$$

where the input is $u(t) = 2e^{-2t}u_0(t)$.

Taking the Laplace transform of both sides and using Result 2.1 gives

$$(s^2 + 5s + 4)\,Y(s) = U(s).$$

Hence the transfer function (Definition 2.2) is

$$G(s) = \frac{Y(s)}{U(s)} = \frac{1}{s^2 + 5s + 4}.$$

Since $\mathcal{L}\{e^{-2t}\} = \frac{1}{s+2}$ and the input is $u(t) = 2e^{-2t}u_0(t)$, we have

$$U(s) = \frac{2}{s+2}.$$

Therefore

$$Y(s) = \frac{2}{(s^2 + 5s + 4)(s+2)}. \tag{2.3}$$

Listing 2.8 verifies the time-domain result using MATLAB's `ilaplace` command.

```
syms s t
Y   = 2 / ((s^2 + 5*s + 4) * (s + 2));
y_t = ilaplace(Y, s, t);

```

Listing 2.8: MATLAB code to compute the inverse Laplace transform $y(t)$ from $Y(s)$ in (2.3).

2.7 Inverse Laplace Transform via Partial Fraction Expansion

Partial fraction expansion (PFE) maps a rational $Y(s)$ to standard transform pairs. The key question is: *what kind of Poles does the expression have?* Distinct real poles, complex poles, and repeated poles each produce a standard type of time response.

2.7.1 Case 1: all poles are simple (real, distinct)

Suppose we can decompose (2.3) as

$$Y(s) = \frac{A}{s+4} + \frac{B}{s+1} + \frac{C}{s+2}.$$

For simple poles, the coefficients can be found using the **cover-up method** (also called the residue method): cover the factor associated with the coefficient you want, then substitute the corresponding pole value. For example:

$$A = \left.\frac{2}{(s+1)(s+2)}\right|_{s=-4} = \frac{1}{3}, \qquad B = \left.\frac{2}{(s+4)(s+2)}\right|_{s=-1} = \frac{2}{3},$$

$$C = \left.\frac{2}{(s+4)(s+1)}\right|_{s=-2} = -1.$$

The cover-up method works cleanly whenever all poles are simple. Each term in the expansion corresponds directly to one exponential in time, so the structure of the response is immediately visible:

$$Y(s) = \frac{1/3}{s+4} + \frac{2/3}{s+1} - \frac{1}{s+2}, \qquad y(t) = \left(\frac{1}{3}e^{-4t} + \frac{2}{3}e^{-t} - e^{-2t}\right) u_0(t).$$

These residues and poles can also be obtained numerically using the `residue` command, as shown in Listing 2.9.

```
num = [2];
den = conv([1 5 4], [1 2]); % (s^2+5s+4)(s+2)
[r, p, k] = residue(num, den);
```

Listing 2.9: MATLAB code for computing the partial fraction expansion using `residue`.

2.7.2 Case 2: complex poles

When a quadratic factor $s^2 + bs + c$ has complex roots, its inverse Laplace transform involves $e^{-\sigma t}\cos(\omega t)$ and $e^{-\sigma t}\sin(\omega t)$. The strategy is:

1. Keep the quadratic factor intact (do *not* split into complex conjugate fractions).

2. Complete the square in the denominator to get the form $(s+\sigma)^2+\omega^2$.
3. Rewrite the numerator as $\alpha(s+\sigma)+\beta\omega$ so that the fraction splits directly into α times the cosine pair plus β times the sine pair:

$$\frac{\alpha(s+\sigma)+\beta\omega}{(s+\sigma)^2+\omega^2} \longleftrightarrow e^{-\sigma t}\big[\alpha\cos(\omega t)+\beta\sin(\omega t)\big].$$

Example 2.5: Complex poles — step by step

Consider the ODE

$$\ddot{y}+\dot{y}+y=u(t),$$

with unit-step input $u(t)=u_0(t)$. Taking Laplace transforms (zero ICs):

$$Y(s)=\frac{1}{s(s^2+s+1)}.$$

Step 1: Partial fraction expansion.

$$\frac{1}{s(s^2+s+1)}=\frac{A}{s}+\frac{Bs+C}{s^2+s+1}.$$

The A term (real pole at $s=0$): use the cover-up method,

$$A=\left.\frac{1}{s^2+s+1}\right|_{s=0}=1.$$

Multiply both sides by $s(s^2+s+1)$:

$$1=(s^2+s+1)+(Bs+C)\,s.$$

Expanding: $1=s^2+s+1+Bs^2+Cs=(1+B)s^2+(1+C)s+1$.

Matching coefficients:

$$\begin{aligned} s^2: &\quad 1+B=0 \quad\Rightarrow\quad B=-1,\\ s^1: &\quad 1+C=0 \quad\Rightarrow\quad C=-1. \end{aligned}$$

So

$$Y(s)=\frac{1}{s}-\frac{s+1}{s^2+s+1}.$$

Step 2: Complete the square in the denominator.

$$s^2+s+1=\left(s+\tfrac{1}{2}\right)^2+\tfrac{3}{4}.$$

Identify: $\sigma=\frac{1}{2}$ and $\omega^2=\frac{3}{4}$, so $\omega=\frac{\sqrt{3}}{2}$.

Step 3: Rewrite the numerator to match $(s+\sigma)$ and ω.

We need the numerator $s+1$ expressed in terms of $(s+\frac{1}{2})$:

$$s+1=\underbrace{\left(s+\tfrac{1}{2}\right)}_{\text{matches cosine pair}}+\underbrace{\tfrac{1}{2}}_{\text{leftover}}.$$

Now write the leftover $\frac{1}{2}$ as a multiple of $\omega = \frac{\sqrt{3}}{2}$:

$$\frac{1}{2} = \frac{1}{\sqrt{3}} \cdot \frac{\sqrt{3}}{2} = \frac{1}{\sqrt{3}} \cdot \omega.$$

So

$$\frac{s+1}{(s+\frac{1}{2})^2 + \frac{3}{4}} = \frac{(s+\frac{1}{2})}{(s+\frac{1}{2})^2 + \omega^2} + \frac{1}{\sqrt{3}} \cdot \frac{\omega}{(s+\frac{1}{2})^2 + \omega^2}.$$

Step 4: Read off the inverse Laplace transform.

Using the standard pairs:

$$\frac{s+\sigma}{(s+\sigma)^2 + \omega^2} \longleftrightarrow e^{-\sigma t}\cos\omega t, \qquad \frac{\omega}{(s+\sigma)^2 + \omega^2} \longleftrightarrow e^{-\sigma t}\sin\omega t,$$

we get

$$y(t) = 1 - e^{-t/2}\cos\left(\frac{\sqrt{3}}{2}t\right) - \frac{1}{\sqrt{3}}e^{-t/2}\sin\left(\frac{\sqrt{3}}{2}t\right), \qquad t \geq 0.$$

Step 5: Compact single-sinusoid form.

The two oscillatory terms can be combined into a single sinusoid using the identity

$$A\cos\theta + B\sin\theta = C\sin(\theta + \phi), \qquad C = \sqrt{A^2 + B^2}, \quad \phi = \arctan\frac{A}{B}.$$

Here, with $\omega_d = \frac{\sqrt{3}}{2}$ and $\theta = \omega_d t$, the oscillatory part is

$$-\left[\cos(\omega_d t) + \frac{1}{\sqrt{3}}\sin(\omega_d t)\right] = -[A\cos(\omega_d t) + B\sin(\omega_d t)],$$

with $A = 1$ and $B = 1/\sqrt{3}$. Therefore

$$C = \sqrt{1 + \tfrac{1}{3}} = \frac{2}{\sqrt{3}}, \qquad \phi = \arctan\frac{A}{B} = \arctan\sqrt{3} = \frac{\pi}{3}.$$

So the step response can be written compactly as

$$\boxed{y(t) = 1 - \frac{2}{\sqrt{3}}e^{-t/2}\sin\left(\frac{\sqrt{3}}{2}t + \frac{\pi}{3}\right), \qquad t \geq 0.}$$

This single-sinusoid form makes it easy to read off the oscillation amplitude envelope $\frac{2}{\sqrt{3}}e^{-t/2}$ and the phase shift $\pi/3$ directly.

2.7.3 Case 3: repeated real poles

Repeated poles lead to one more standard pattern: powers of t multiplying exponentials. The extra factor of t is the time-domain signature of pole repetition.

Example 2.6: Inverse Laplace with a repeated pole

Find the inverse Laplace transform of

$$Y(s) = \frac{s-1}{(s+1)^2}.$$

For a double pole at $s = -1$, the partial fraction form is

$$\frac{s-1}{(s+1)^2} = \frac{A}{s+1} + \frac{B}{(s+1)^2}.$$

Multiplying both sides by $(s+1)^2$:

$$s - 1 = A(s+1) + B.$$

Setting $s = -1$ gives $B = -2$. Comparing the coefficient of s gives $A = 1$. Therefore

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

and using $\mathcal{L}^{-1}\{1/(s+a)^2\} = t\,e^{-at}$,

$$y(t) = e^{-t} - 2t\,e^{-t}, \qquad t \geq 0.$$

In general, a pole of multiplicity m at $s = -a$ contributes terms of the form $t^k e^{-at}$ for $k = 0, 1, \ldots, m-1$. The corresponding Laplace pair is

$$\mathcal{L}\{t^k e^{-at}\} = \frac{k!}{(s+a)^{k+1}}.$$

2.8 Impulse Response and Convolution

For LTI systems, two ideas connect the transfer function to time-domain behaviour: the *impulse response* and *convolution*.

2.8.1 Impulse response as the system fingerprint

The **impulse response** $g(t)$ is the output produced by the input $\delta(t)$. It captures the system's intrinsic dynamics in a single time-domain function.

If the transfer function of the system is $G(s)$, then an impulse input has transform $\mathcal{L}\{\delta(t)\} = 1$, so

$$Y(s) = G(s) \cdot 1 = G(s).$$

Therefore the impulse response is simply

$$g(t) = \mathcal{L}^{-1}\{G(s)\}.$$

The impulse response is the time-domain counterpart of the transfer function. In a mechanical system it shows how the structure vibrates after a sharp tap; in a circuit it shows the reaction to a short burst of

excitation. Once $g(t)$ is known, the output for any input can be computed without re-solving the differential equation.

2.8.2 Convolution in time, multiplication in the Laplace domain

Once the impulse response $g(t)$ is known, the output corresponding to a general input $x(t)$ can be written as the convolution of $x(t)$ with $g(t)$:

$$y(t) = (x \star g)(t) \triangleq \int_0^t g(t-\tau)\, x(\tau)\, d\tau. \tag{2.4}$$

Equation (2.4) says that the output at time t is built from past input values, weighted by how the system responds over time. The symbol $\star$ denotes convolution; it is not ordinary multiplication. Each small piece of the input excites the system, and the total output is the accumulation of all those shifted small responses.

$$y(t) = g(t) \star x(t) = \int_0^t g(t-\tau)x(\tau)\, d\tau$$

$$y(t) \neq g(t)x(t)$$

$$Y(s) = G(s)X(s)$$

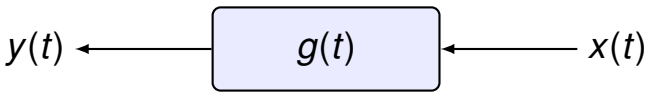


Figure 2.9: Convolution of input $x(t)$ with impulse response $g(t)$ in time corresponds to multiplication of $X(s)$ and $G(s)$ in the Laplace domain.

The great advantage of the Laplace transform is that it turns the integral in (2.4) into ordinary multiplication.

Theorem 2.5: Convolution theorem

If $\mathcal{L}\{x(t)\} = X(s)$ and $\mathcal{L}\{g(t)\} = G(s)$, then

$$\mathcal{L}\{x(t) \star g(t)\} = X(s)\, G(s).$$

Proof. Starting from the definition of the Laplace transform applied to the convolution integral:

$$\mathcal{L}\{x \star g\} = \int_0^\infty \left[\int_0^t g(t-\tau)\, x(\tau)\, d\tau \right] e^{-st}\, dt.$$

The double integral is over the region $0 \leq \tau \leq t < \infty$. Switching the order of integration to $0 \leq \tau < \infty$, $\tau \leq t < \infty$:

$$= \int_0^\infty x(\tau) \left[\int_\tau^\infty g(t-\tau)\, e^{-st}\, dt \right] d\tau.$$

In the inner integral, substitute $\sigma = t - \tau$:

$$= \int_0^\infty x(\tau)\, e^{-s\tau} \left[\int_0^\infty g(\sigma)\, e^{-s\sigma}\, d\sigma \right] d\tau = G(s) \int_0^\infty x(\tau)\, e^{-s\tau}\, d\tau = G(s)\, X(s). \qquad \square$$

Therefore, for an LTI system with transfer function $G(s)$ and input $X(s)$, we can write $Y(s) = G(s)X(s)$, which in the time domain means exactly that $y(t) = x(t) \star g(t)$. This theorem explains why transfer-function methods are so effective: in the time domain, interconnected system behaviour may require integration, but in the Laplace domain, the same relationship is captured by straightforward multiplication.

2.9 Laplace Transform of Periodic Signals*

The Laplace transform of a periodic signal can always be expressed in terms of the transform of a single period.

Result 2.7: Laplace transform of a periodic signal

Let $f(t)$ be periodic with period $T > 0$ for $t \geq 0$, i.e. $f(t+T) = f(t)$. Define $f_1(t)$ as one period of the signal:

$$f_1(t) = \begin{cases} f(t), & 0 \leq t < T, \\ 0, & \text{otherwise.} \end{cases}$$

If $F_1(s) = \mathcal{L}\{f_1(t)\}$, then

$$F(s) = \frac{F_1(s)}{1 - e^{-sT}}. \tag{2.5}$$

Proof. Write $f(t)$ as a sum of shifted copies of $f_1(t)$:

$$f(t) = \sum_{k=0}^{\infty} f_1(t - kT).$$

By the time-shift property (Result 2.6), $\mathcal{L}\{f_1(t-kT)\} = e^{-ksT}F_1(s)$, so

$$F(s) = \sum_{k=0}^{\infty} e^{-ksT}F_1(s) = F_1(s)\sum_{k=0}^{\infty}\left(e^{-sT}\right)^k = \frac{F_1(s)}{1 - e^{-sT}},$$

where the geometric series converges for $\Re(s) > 0$. □

The denominator $1 - e^{-sT}$ is common to all periodic signals with the same period; only the single-period transform $F_1(s)$ changes from one waveform to another. This makes it easy to analyse different periodic excitations once the one-period integral has been computed.

Example 2.7: Laplace transform of a symmetric square wave

Find the Laplace transform of a square wave with amplitude A and period T defined by

$$f(t) = \begin{cases} +A, & 0 \leq t < T/2, \\ -A, & T/2 \leq t < T, \end{cases} \qquad f(t+T) = f(t).$$

Step 1: Compute the one-period transform.

$$F_1(s) = \int_0^{T/2} A\,e^{-st}\,dt + \int_{T/2}^{T} (-A)\,e^{-st}\,dt = \frac{A}{s}\left(1 - e^{-sT/2}\right) - \frac{A}{s}\left(e^{-sT/2} - e^{-sT}\right) = \frac{A}{s}\left(1 - e^{-sT/2}\right)^2.$$

Step 2: Apply (2.5).

$$F(s) = \frac{F_1(s)}{1 - e^{-sT}} = \frac{A}{s}\cdot\frac{\left(1 - e^{-sT/2}\right)^2}{\left(1 - e^{-sT/2}\right)\left(1 + e^{-sT/2}\right)} = \frac{A}{s}\cdot\frac{1 - e^{-sT/2}}{1 + e^{-sT/2}}.$$

Multiplying numerator and denominator by $e^{sT/4}$ gives

$$\boxed{F(s) = \frac{A}{s}\tanh\left(\frac{sT}{4}\right).}$$

Why this matters for PID tuning

In Chapter 8, the relay (on–off) method for PID tuning drives the plant with a square-wave-like signal and measures the resulting oscillation period and amplitude. Understanding how periodic inputs are represented in the Laplace domain helps explain why the method works: the dominant harmonic of the relay output determines the critical gain and frequency.

Listing 2.10 verifies the square-wave result by comparing the analytical tanh form with a numerical partial-sum approximation.

```
A = 1;  T = 2;
s = 0.1:0.1:5;

% Analytical result: (A/s)*tanh(s*T/4)
F_exact = (A./s) .* tanh(s*T/4);

% Numerical check: sum N shifted copies of F1(s)
N = 200;
F_approx = zeros(size(s));
for k = 0:N-1
    F_approx = F_approx + (A./s) .* ...
        (1 - 2*exp(-s*T/2) + exp(-s*T)) .* exp(-k*s*T);
end

fprintf('Max error with %d terms: %.2e\n', N, max(abs(F_exact - F_approx)));
```

Listing 2.10: Verifying the periodic square-wave Laplace transform in MATLAB.

2.10 Useful MATLAB Commands for Laplace Transform Analysis

The following MATLAB commands are used throughout this chapter and the next few chapters:

- `residue(num, den)` — compute residues and poles for partial fraction expansion. The syntax is `[r, p, k] = residue(num, den)`, where `r` contains the residues, `p` the poles, and `k` the direct term (if any).
- `partfrac(F)` — symbolic partial fraction expansion (requires the Symbolic Math Toolbox).
- `laplace(f)` — Laplace transform of a symbolic expression $f(t)$.
- `ilaplace(F)` — inverse Laplace transform of $F(s)$.
- `limit(s*F, s, 0)` — apply the Final Value Theorem (Theorem 2.2).

Listing 2.11 combines `partfrac` and `ilaplace` in a short workflow.

```
  syms s t
  F = (s+1)/((s+2)*(s+3)^3);
  Factorized = partfrac(F);
  f_t = ilaplace(Factorized, s, t);
  disp(f_t)
```

Listing 2.11: Using `partfrac` and `ilaplace` to compute an inverse Laplace transform.

Common mistakes with Laplace methods

- Forgetting the initial-condition terms in derivative transforms, for example writing $\mathcal{L}\{\dot{x}\} = sX(s)$ when $x(0) \neq 0$.
- Applying the Final Value Theorem before checking the poles of $sF(s)$.
- Forgetting the unit-step factor in delayed signals: a causal delay should be written as $f(t-a)u_0(t-a)$, not just $f(t-a)$.
- Treating time-domain convolution as ordinary multiplication. In general, $y(t) = g(t) \star x(t)$, not $y(t) = g(t)x(t)$.
- Mixing the input notation $u(t)$ with the unit-step notation $u_0(t)$. In this book, $u(t)$ usually means a control input, while $u_0(t)$ denotes the unit step.

2.11 Summary

Chapter Summary

- The Laplace transform converts time-domain differential equations into algebraic equations in s, which is why it is so useful in control engineering.
- The most important beginner workflow is: transform the ODE, substitute the initial conditions, solve algebraically for $X(s)$ or $Y(s)$, and then invert the transform.
- Key properties include linearity, differentiation in time $\left(\mathcal{L}\{\dot{f}\} = sF(s) - f(0)\right)$, differentiation in the s-domain $\left(\mathcal{L}\{tf(t)\} = -dF/ds\right)$, integration, and shifting.
- Common signals such as the step, ramp, impulse, exponentials, and sinusoids have standard Laplace pairs that are used repeatedly in analysis and design.
- The Laplace transform of a periodic signal with period T is $F(s) = F_1(s)/(1 - e^{-sT})$, where $F_1(s)$ is the transform of one period.
- The transfer function $G(s) = Y(s)/U(s)$ captures the input–output dynamics of an LTI system under zero initial conditions.
- Partial fraction expansion is a practical technique for computing inverse Laplace transforms of rational functions and interpreting time responses.
- The impulse response $g(t)$ is the time-domain fingerprint of an LTI system, and convolution describes how a general input produces an output through that impulse response.
- The Initial and Final Value Theorems provide quick access to $f(0^+)$ and $f(\infty)$ directly from $F(s)$, without always performing the full inverse transform.
- MATLAB commands such as `laplace()`, `ilaplace()`, `partfrac()`, and `residue()` are useful for verifying hand calculations and exploring more complex expressions.

Summary Table of Common Laplace Transform Pairs

Signal type	Time domain $f(t),\ t \geq 0$	Laplace domain $F(s)$
Unit impulse	$\delta(t)$	1
Unit step	$u_0(t)$	$\dfrac{1}{s}$
Unit ramp	t	$\dfrac{1}{s^2}$
Powers of t	t^n	$\dfrac{n!}{s^{n+1}}$
Exponential	e^{-at}	$\dfrac{1}{s+a}$
$t\times$ exponential	$t\,e^{-at}$	$\dfrac{1}{(s+a)^2}$
Sine	$\sin(\omega t)$	$\dfrac{\omega}{s^2+\omega^2}$
Cosine	$\cos(\omega t)$	$\dfrac{s}{s^2+\omega^2}$
Damped sine	$e^{-at}\sin(\omega t)$	$\dfrac{\omega}{(s+a)^2+\omega^2}$
Damped cosine	$e^{-at}\cos(\omega t)$	$\dfrac{s+a}{(s+a)^2+\omega^2}$

Property	Result
Linearity	$\mathcal{L}\{\alpha f+\beta g\} = \alpha F(s)+\beta G(s)$
Differentiation	$\mathcal{L}\{\dot f\} = sF(s) - f(0)$
Second derivative	$\mathcal{L}\{\ddot f\} = s^2F(s) - sf(0) - \dot f(0)$
Integration	$\mathcal{L}\left\{\int_0^t f(\tau)\,d\tau\right\} = \dfrac{F(s)}{s}$
s-domain differentiation	$\mathcal{L}\{t^n f(t)\} = (-1)^n \dfrac{d^nF(s)}{ds^n}$
Frequency shift	$\mathcal{L}\{e^{at}f(t)\} = F(s-a)$
Time shift	$\mathcal{L}\{f(t-a)\,u_0(t-a)\} = e^{-as}F(s)$
Periodic signal	$f(t+T)=f(t) \Rightarrow F(s) = \dfrac{F_1(s)}{1-e^{-sT}}$
Final value	$\lim_{t\to\infty} f(t) = \lim_{s\to 0} sF(s)$
Initial value	$\lim_{t\to 0^+} f(t) = \lim_{s\to\infty} sF(s)$

A comprehensive table with waveform sketches and additional pairs is given in the Appendix.

2.12 End-of-Chapter Problems

1. **(Laplace Transform — Basic Signals)** Compute the Laplace transform of each signal from the definition or using the transform table and properties. State the region of convergence where applicable.

   (a) $f(t) = 3\,u_0(t) - 2\,e^{-4t}\,u_0(t)$

   (b) $f(t) = t\,e^{-2t}\,u_0(t)$

(c) $f(t) = e^{-t}\sin(3t)\,u_0(t)$

(d) $f(t) = 5\,u_0(t-2)$ (delayed unit step of magnitude 5)

2. **(Inverse Laplace Transform — Distinct Real Poles)** Find $f(t)$ for each $F(s)$ using partial fraction expansion:

(a) $F(s) = \dfrac{5}{s(s+3)}$

(b) $F(s) = \dfrac{2s+7}{(s+1)(s+4)}$

(c) $F(s) = \dfrac{10}{(s+1)(s+2)(s+5)}$

3. **(Inverse Laplace Transform — Complex Poles)** Find $f(t)$:

(a) $F(s) = \dfrac{4}{s^2+4s+13}$

(b) $F(s) = \dfrac{s+1}{s^2+2s+5}$

Hint: Complete the square in the denominator and use the $e^{-at}\sin(\omega t)$ and $e^{-at}\cos(\omega t)$ transform pairs.

4. **(Inverse Laplace Transform — Repeated Poles)** Find $f(t)$:

(a) $F(s) = \dfrac{3}{(s+2)^2}$

(b) $F(s) = \dfrac{s+5}{s(s+1)^2}$

5. **(Transfer Function from ODE)** A system is described by the differential equation:

$$\ddot{y} + 5\dot{y} + 6y = 2\dot{u} + 3u$$

(a) Assuming zero initial conditions, derive the transfer function $G(s) = Y(s)/U(s)$.

(b) Find the poles and zeros of $G(s)$.

(c) Is the system stable? Justify your answer.

(d) Use the Final Value Theorem to find the steady-state output for a unit step input.

6. **(Initial and Final Value Theorems)** For each $F(s)$, use the IVT to find $f(0^+)$ and the FVT to find $\lim_{t\to\infty} f(t)$ (if the limit exists). State any conditions required.

(a) $F(s) = \dfrac{3(s+2)}{s(s+1)(s+4)}$

(b) $F(s) = \dfrac{10}{s(s^2+2s+10)}$

(c) $F(s) = \dfrac{5}{s^2+4}$ (Does the FVT apply? Why or why not?)

7. **(Step Response by Hand)** Consider the transfer function:

$$G(s) = \frac{6}{(s+1)(s+3)}$$

(a) Find the unit step response $Y(s) = G(s)\cdot\dfrac{1}{s}$.

(b) Perform partial fraction expansion on $Y(s)$.

(c) Compute $y(t)$ using the inverse Laplace transform.

(d) What is the steady-state value? Verify using the Final Value Theorem.

(e) Verify your result in MATLAB using `step(tf([6],[1 4 3]))`.

8. **(MATLAB Verification)** Use MATLAB to verify your answers to Problems 2 and 4:

   (a) Use `residue(num, den)` to compute the partial fraction expansion for Problem 2(c). Compare with your hand calculation.

   (b) Use the Symbolic Math Toolbox: `syms s; F = ...; ilaplace(F)` to find the inverse transform for Problem 4(b). Does it match?

   (c) Create a transfer function object for Problem 7: `G = tf([6],[1 4 3])`. Use `step(G)` and `stepinfo(G)` to find the settling time and steady-state value.

9. **(Pole–Zero Map and Behaviour)** For each transfer function below, find the poles and zeros, plot the pole–zero map (by hand or using `pzmap()` in MATLAB), and describe the expected time-domain behaviour (decaying, oscillatory, growing, etc.):

   (a) $G_1(s) = \dfrac{1}{s+5}$

   (b) $G_2(s) = \dfrac{s+2}{s^2+2s+10}$

   (c) $G_3(s) = \dfrac{4}{s^2-1}$

10. **(Periodic Signals)** A sawtooth wave has amplitude A, period T, and is defined over one period by $f_1(t) = \frac{A}{T}\,t$ for $0 \le t < T$.

    (a) Show that the one-period Laplace transform is

    $$F_1(s) = \frac{A}{Ts^2}\big(1 - e^{-sT}(1+sT)\big).$$

    (b) Use the periodic-signal formula (2.5) to write $F(s)$.

    (c) A first-order system $G(s) = 1/(s+1)$ is driven by the sawtooth with $A = 1$ and $T = 2$. Use MATLAB to plot the output $y(t)$ for $0 \le t \le 10$ using `lsim` with a sawtooth input vector. Comment on whether $y(t)$ also becomes periodic after an initial transient.

11. **(Challenge)** A system has the transfer function:

    $$G(s) = \frac{s+3}{(s+1)(s^2+4s+8)}$$

    (a) Find all poles and zeros. Identify the damping ratio and natural frequency of the complex poles.

    (b) Perform full partial fraction expansion of $Y(s) = G(s)/s$.

    (c) Find $y(t)$ and sketch the expected step response shape.

    (d) Verify every step using MATLAB: `residue`, `ilaplace`, `step`, and `pzmap`.

Chapter 3

Modelling of dynamical systems and simulation

"Everything should be made as simple as possible, but not simpler."

— Albert Einstein

Learning Objectives

After completing this chapter, you should be able to:

- Derive transfer functions from physical laws for electrical (RLC), mechanical (translational and rotational), and thermal systems.
- Apply Kirchhoff's voltage and current laws to obtain circuit equations and convert them to the Laplace domain.
- Use the impedance method to derive transfer functions without writing differential equations explicitly.
- Apply Newton's second law to translational and rotational systems, including multi-body and geared configurations.
- Identify the electrical–mechanical–thermal analogies and use them to model systems across domains.
- Construct and simulate transfer function models in MATLAB using `tf()`, `step()`, and `lsim()`.

A **model** of a dynamical system is a mathematical description that links inputs, internal behaviour, and outputs in a way useful for prediction and control design.

3.1 Why do we need models of systems at all?

A mathematical model allows us to predict how a system will respond under different conditions *before* we build hardware or run expensive experiments. Once we have equations that describe the system, we can reason about stability, speed of response, and sensitivity to disturbances in a structured way — rather than relying on trial-and-error.

In safety-critical applications (aerospace, robotics, biomedical devices), models let us test disturbances, parameter variations, and candidate control laws through simulation before anything is deployed. Without a model, controller design becomes guesswork.

Remark 3.1

Models are abstractions of reality. They are not perfect copies of the physical world, but they are detailed enough to capture the essential dynamics needed for prediction, analysis, and design.

3.2 Why physics-based modelling still matters

If AI can learn from data, why do we still need mathematical models? AI methods work best when large amounts of reliable data are available and operating conditions match training. In safety-critical systems (aircraft, medical devices, autonomous vehicles), we cannot allow a controller to "learn by making mistakes." We need predictable behaviour from the start, and that comes from mathematical modelling. Furthermore, at the design stage a new system may have little or no data — physics-based models are then the only tools available.

The most effective solutions often combine both: physics-based models capture the dominant dynamics, while AI compensates for hard-to-model effects like friction or wear. In many industries, regulations require mathematical justification of system behaviour.

Why safety-critical industries demand white-box design

In domains where failure can cost lives — civil aviation, nuclear power, medical devices, autonomous vehicles — industry standards impose a non-negotiable requirement: every component in the control loop must be fully understood, mathematically justified, and independently verifiable. There is no room for black boxes.

Certification frameworks. In aviation, for example, standards such as ARP 4754A [20], DO-178C [19], and DO-254 [18] require the development process to be deterministic, auditable, and

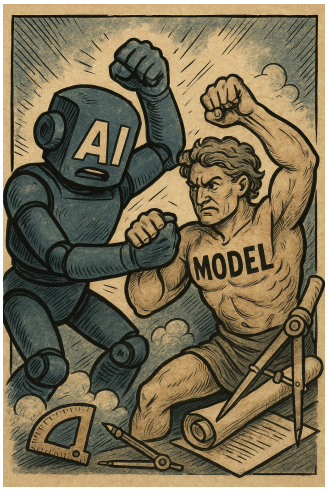


Figure 3.1: AI vs traditional modelling and control. Who do you think will win?

traceable. The authority reviewing the design must be able to inspect the equations, examine the control law, and verify why each design choice was made.

What "white-box" design means in practice. When a flight control law is submitted for certification, the manufacturer must provide a validated mathematical model of the aircraft, a controller whose gains can be traced back to explicit requirements, evidence that the closed-loop system remains stable throughout the operating envelope, and worst-case analysis for failures, uncertainty, saturation, and disturbances. A black-box controller cannot offer this kind of explanation, even if it performs well in limited testing.

The same logic appears elsewhere. Automotive standards such as ISO 26262 [8], medical-device standards such as IEC 62304 [7], nuclear standards such as IEC 61513 [6], and railway standards such as EN 50128/50129 [3, 4] all demand some form of transparent, verifiable, model-based reasoning.

The bottom line. In safety-critical systems the real question is not merely "does it work in this test?" but "can you explain why it works, prove that it is safe enough, and justify its behaviour beyond the exact situations you already observed?" Model-based design remains indispensable because it provides that line of reasoning.

Remark 3.2

Mathematical models are not outdated; they remain the foundation of control engineering. AI is a powerful companion, but it cannot replace the physical insight, safety guarantees, and predictive power that models provide.

3.3 Types of Models in Control Engineering

Models can be classified **by origin** (physical/first-principles vs. empirical/data-driven), **by domain** (time-domain differential equations vs. frequency-domain transfer functions), and **by mathematical form** (linear vs. nonlinear, continuous vs. discrete).

In this module we typically start from a physical continuous-time model, then simplify or linearise it into a linear time-invariant transfer function or state-space form suitable for analysis and control design.

3.4 What Makes a Good Control Model?

A good control model captures the dynamics that matter for the control task while remaining simple enough for analysis and design. The input, output, states, and disturbances should be identified explicitly, and the model should be accurate over the operating range where the controller will work. A model that is too simple misses essential behaviour; one that is too complex becomes impractical.

3.5 Linear Models

A large class of dynamical systems can be represented using linear constant-coefficient differential equations. The general input-output form is

$$y^{(n)}(t) + a_{n-1}y^{(n-1)}(t) + \cdots + a_1 y^{(1)}(t) + a_0 y(t) = \\ b_m u^{(m)}(t) + b_{m-1}u^{(m-1)}(t) + \cdots + b_1 u^{(1)}(t) + b_0 u(t), \quad (3.1)$$

Here, $u(t)$ is the input, $y(t)$ is the output, $y^{(i)}(t)$ denotes the i-th derivative of $y(t)$, and the coefficients a_i and b_j are constants. The condition $m \leq n$ is consistent with ordinary physical systems, where the output dynamics are not of lower order than the input-side dynamics.

Because the coefficients are constant, the system is **time-invariant**. In control calculations we often set the initial conditions to zero to isolate the pure input-output behaviour, and we assume Causality (the system cannot respond to future inputs).

Assuming zero initial conditions, taking the Laplace transform of (3.1) gives

$$(s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0)Y(s) = (b_m s^m + b_{m-1}s^{m-1} + \cdots + b_1 s + b_0)U(s).$$

We therefore define the **transfer function** as

$$G(s) \triangleq \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1}s^{m-1} + \cdots + b_1 s + b_0}{s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0}.$$

```
% Method 1: Using coefficient vectors
num = [1 3];           % Numerator: s + 3
den = [1 5 6];         % Denominator: s^2 + 5s + 6
sys1 = tf(num, den);

% Method 2: Using symbolic 's'
s = tf('s');
sys2 = (s + 3) / (s^2 + 5*s + 6);

disp(sys1);
disp(sys2);

```

Listing 3.1: Two MATLAB methods to define the transfer function $G(s) = \frac{s+3}{s^2+5s+6}$: (1) using numerator/denominator coefficients, (2) using the symbolic variable `s=tf('s')`.

Listing 3.1 shows two equivalent ways to define a transfer function in MATLAB. This ratio summarises how the system maps input to output in the Laplace domain when the initial state is zero. As a block diagram, it is represented in Figure 3.2.

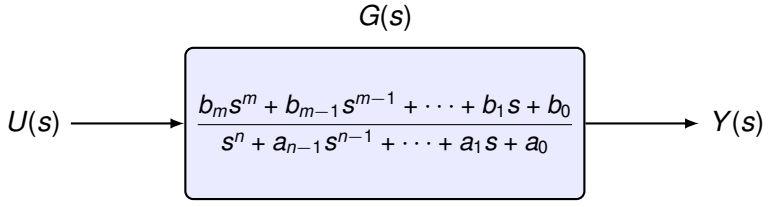


Figure 3.2: Transfer function representation as a block diagram.

Remark 3.3

A transfer function is called Proper transfer function when the degree of the denominator is greater than or equal to that of the numerator ($m \leq n$). It is called **strictly proper** when the denominator degree is strictly larger ($m < n$), in which case

$$G(\infty) = \lim_{s\to\infty} G(s) = 0.$$

If the numerator and denominator have the same degree, then $G(\infty) \neq 0$ in general and the transfer function can be separated into a constant high-frequency gain and a strictly proper remainder:

$$G(s) = G(\infty) + G_{sp}(s).$$

Example 3.1: Transfer Function of a System Represented by a Differential Equation

Find the transfer function of the system represented by the differential equation

$$\ddot{y} + 8\dot{y} + 15y \;=\; 3\ddot{u} + 10\dot{u} - 12u,$$

where $\dot{u} = u^{(1)}$, $\ddot{u} = u^{(2)}$, and so on. Determine if the transfer function is proper or strictly proper. If it is not strictly proper, find the strictly proper part.

The transfer function can be written by inspection as

$$G(s) = \frac{3s^2 + 10s - 12}{s^2 + 8s + 15}.$$

The second-order transfer function $G(s)$ is proper because the degrees of the numerator and denominator are the same. To check if it is strictly proper, we evaluate

$$G(\infty) = \lim_{s\to\infty} G(s).$$

Since the leading coefficients are 3 (numerator) and 1 (denominator), we have

$$G(\infty) = 3.$$

Thus, $G(s)$ is proper but not strictly proper. We can write

$$G(s) - G(\infty) = \frac{3s^2 + 10s - 12}{s^2 + 8s + 15} - 3.$$

Simplify the expression:

$$\begin{aligned} G(s) - G(\infty) &= \frac{3s^2 + 10s - 12 - 3(s^2 + 8s + 15)}{s^2 + 8s + 15}, \\ &= \frac{3s^2 + 10s - 12 - (3s^2 + 24s + 45)}{s^2 + 8s + 15}, \\ &= \frac{-14s - 57}{s^2 + 8s + 15}. \end{aligned}$$

Based on this, we define the strictly proper part as

$$G_{sp}(s) = \frac{-14s - 57}{s^2 + 8s + 15}.$$

Hence,

$$G(s) = 3 + \frac{-14s - 57}{s^2 + 8s + 15}.$$

```
% Define zeros, poles, and gain
z = [-2 -3];        % zeros
p = [-1 -5 -10];    % poles
k = 4;              % gain

% Create transfer function in ZPK form
sys = zpk(z,p,k);

% Display system
disp(sys)
```

Figure 3.3: Defining a transfer function in MATLAB using zero–pole–gain (ZPK) form.

Poles and zeros. Once a transfer function has been written down, the next quantities to identify are its zeros and poles. The Zeros are the values of s that make the numerator vanish. If the numerator factors as

$$b_m(s - z_1)(s - z_2) \cdots (s - z_m),$$

then $z_1, z_2, \ldots, z_m$ are the zeros. The **poles** are the values of s that make the denominator vanish. If the denominator factors as

$$(s - p_1)(s - p_2) \cdots (s - p_n),$$

then $p_1, p_2, \ldots, p_n$ are the poles.

This lets us rewrite the transfer function in the compact pole-zero form

$$G(s) = K \cdot \frac{(s - z_1)(s - z_2) \cdots (s - z_m)}{(s - p_1)(s - p_2) \cdots (s - p_n)},$$

where K is a constant gain factor (often b_m). Listing 3.3 shows how to define a transfer function using zero–pole–gain form in MATLAB.

The poles of $G(s)$ largely determine the stability and natural dynamic behaviour of the system. For example, if any pole lies in the right half of the complex plane, the system is unstable. The zeros influence how the input is shaped on its way to the output and therefore affect transient and frequency behaviour, but they do not by themselves determine stability. If the coefficients of the transfer function are real, then complex poles and zeros appear in conjugate pairs.

Remark 3.4

Poles shape the system's natural dynamics, while zeros affect how the input is filtered through the system.

Example 3.2: Identifying poles and zeros of a transfer function

Given the transfer function

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

find all poles and zeros, write $G(s)$ in pole-zero form, and determine whether the system is stable.

Solution:

We begin with the zeros, which are the values of s that make the numerator equal to zero:

$$2(s+2)(s+3) = 0.$$

Hence,

$$s = -2, \qquad s = -3.$$

Therefore, the system has two zeros located at

$$z_1 = -2, \qquad z_2 = -3.$$

Next we find the poles by setting the denominator equal to zero:

$$(s+1)(s+4)(s+5) = 0.$$

Hence,

$$s = -1, \qquad s = -4, \qquad s = -5.$$

Thus, the system has three poles located at

$$p_1 = -1, \qquad p_2 = -4, \qquad p_3 = -5.$$

The transfer function can therefore be written in pole-zero form as

$$G(s) = 2\frac{(s+2)(s+3)}{(s+1)(s+4)(s+5)} = K\frac{(s-z_1)(s-z_2)}{(s-p_1)(s-p_2)(s-p_3)},$$

where

$$K = 2, \quad z_1 = -2, \quad z_2 = -3, \quad p_1 = -1, \quad p_2 = -4, \quad p_3 = -5.$$

Since all poles lie in the left half-plane $(-1, -4, -5)$, the system is stable. The poles at -4 and -5 correspond to faster decaying modes, while the pole at -1 dominates the slower part of the response. The zeros at -2 and -3 influence how the input signal is filtered before it appears at the output. Figure 3.4 shows these pole and zero locations in the s-plane.

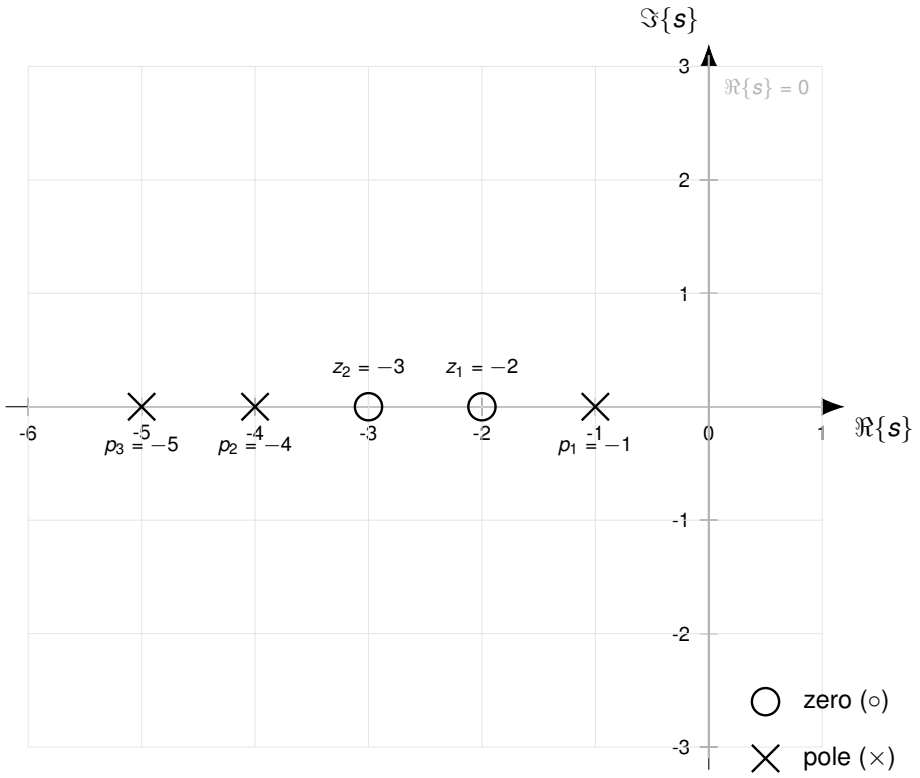


Figure 3.4: Pole–zero plot in the s-plane for $G(s) = \frac{2(s+2)(s+3)}{(s+1)(s+4)(s+5)}$. All poles lie in the left half-plane, so the system is stable.

3.6 Electrical Circuits

RLC circuit modelling starts from the constitutive equation of each component and applies Kirchhoff's laws to the connections. Figure 3.5 shows the basic elements.

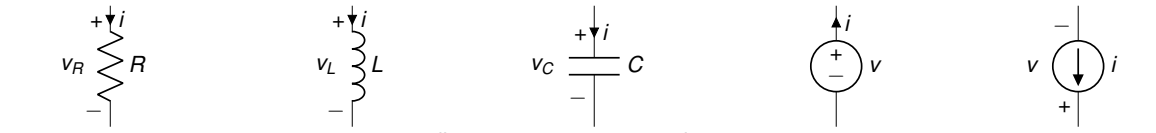


Figure 3.5: Basic components of an RLC circuit with increased spacing.

Each element has its own constitutive equation:

$$
\begin{aligned}
v_R(t) &= R\, i(t), \\
v_L(t) &= L\, \frac{di(t)}{dt}, \\
i_C(t) &= C\, \frac{dv(t)}{dt}.
\end{aligned}
$$

These equations are the building blocks of more complex circuits. Once the components are connected together, we use Kirchhoff's laws to combine them into an overall model.

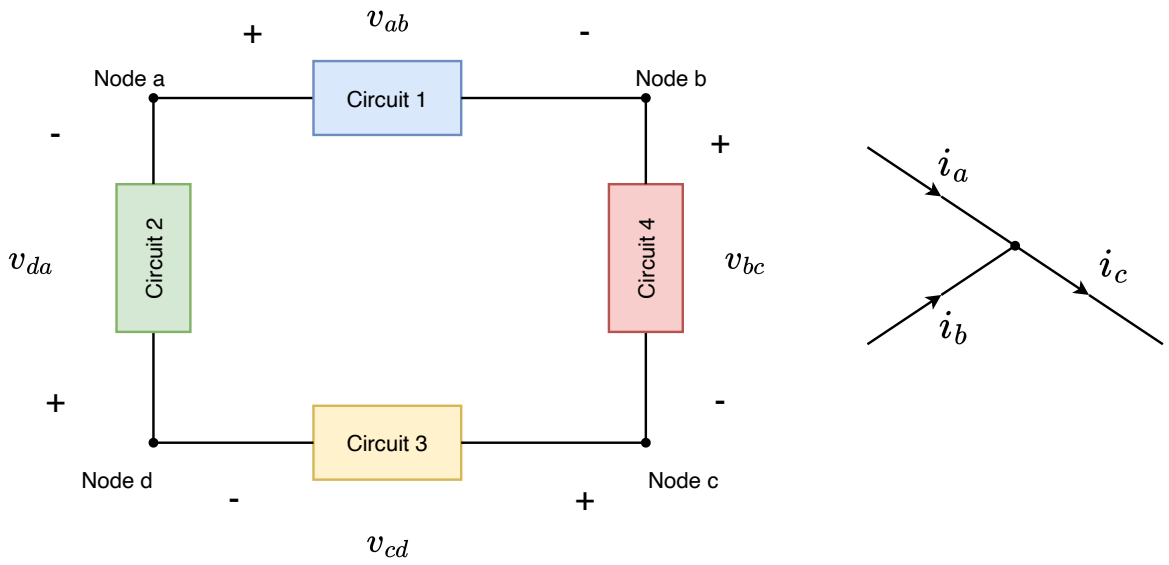


Figure 3.6: Kirchoff's voltage (a) and current laws (b).

Kirchhoff's laws. Figure 3.6 shows the two laws that organise circuit modelling. Kirchhoff's Voltage Law (KVL) states that the algebraic sum of voltages around a closed loop is zero. Kirchhoff's Current Law (KCL) states that the algebraic sum of currents at a node is zero, or simply that current flowing into a node must equal current flowing out.

Theorem 3.1: KVL and KCL

The algebraic sum of voltages around any closed loop is zero:

$$v_{ab} + v_{bc} + v_{cd} + v_{da} = 0$$

The algebraic sum of currents at any node is zero. In other words, the total current that goes into a node should leave the node:

$$i_a + i_b - i_c = 0$$

Example 3.3: Parallel RLC Circuit

Find the transfer function $\frac{V(s)}{I(s)}$ for the parallel RLC circuit shown in Fig. 3.7. Assume $v(t) = 0$ for $t < 0$.

Applying Kirchhoff's Current Law gives

$$i(t) = i_R(t) + i_L(t) + i_C(t).$$

Substituting the component relations gives

$$i(t) = \frac{v(t)}{R} + \frac{1}{L}\int_0^t v(\tau)\,d\tau + C\frac{dv(t)}{dt}.$$

Differentiating both sides with respect to t:

$$\frac{di(t)}{dt} = \frac{1}{R}\frac{dv(t)}{dt} + \frac{1}{L}v(t) + C\frac{d^2v(t)}{dt^2}.$$

Dividing by C:

$$\frac{d^2v(t)}{dt^2} + \frac{1}{CR}\frac{dv(t)}{dt} + \frac{1}{CL}v(t) = \frac{1}{C}\frac{di(t)}{dt}.$$

Taking the Laplace transform (zero initial conditions):

$$\left(s^2 + \frac{1}{CR}s + \frac{1}{CL}\right)V(s) = \frac{s}{C}I(s).$$

Hence, the transfer function is

$$G(s) = \frac{V(s)}{I(s)} = \frac{\frac{1}{C}s}{s^2 + \frac{1}{CR}s + \frac{1}{CL}}.$$

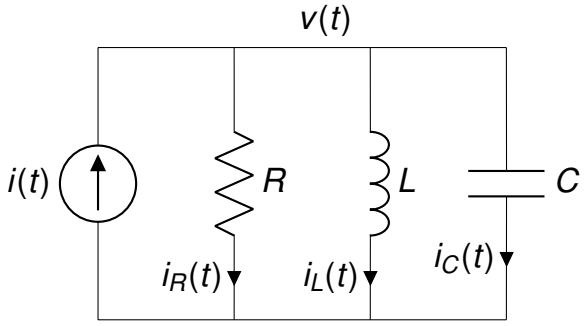


Figure 3.7: Parallel RLC circuit excited by a current source.

Example 3.4: Series RLC Circuit

Find the transfer function I/V of the series RLC circuit in Figure 3.8. Assume that no current was flowing through the loop prior to $t = 0$.

The current through all components is the same, so Kirchhoff's Voltage Law gives

$$\begin{aligned} v(t) &= v_R(t) + v_L(t) + v_C(t) \\ &= Ri(t) + L\frac{di(t)}{dt} + \frac{1}{C}\int_0^t i(\tau)\,d\tau + v_C(0). \end{aligned}$$

Differentiating both sides with respect to time and dividing by L, we obtain

$$\frac{d^2i(t)}{dt^2} + \frac{R}{L}\frac{di(t)}{dt} + \frac{1}{CL}i(t) = \frac{1}{L}\frac{dv(t)}{dt}.$$

Taking the Laplace transform under zero initial conditions,

$$\left(s^2 + \frac{R}{L}s + \frac{1}{CL}\right) I(s) = \frac{s}{L} V(s).$$

Thus, the transfer function is

$$G(s) = \frac{I(s)}{V(s)} = \frac{\frac{1}{L}s}{s^2 + \frac{R}{L}s + \frac{1}{CL}}.$$

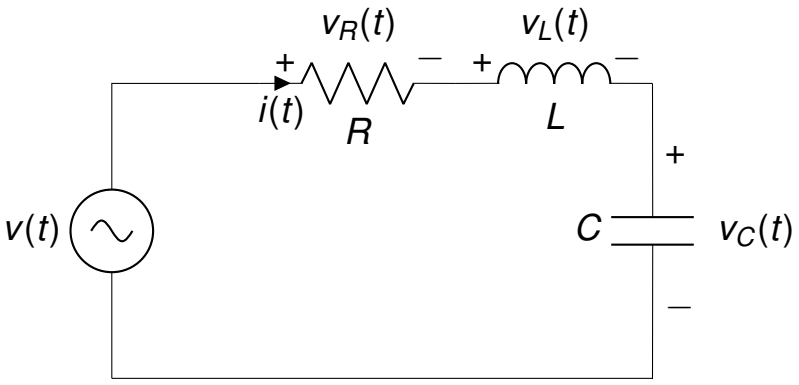


Figure 3.8: Series RLC circuit excited with a voltage source $v(t)$.

3.7 Impedance Method for Circuit Modelling

A quicker alternative to writing differential equations is to replace each element by its Impedance in the Laplace domain and analyse the circuit using series and parallel rules.

Impedance. For an electric circuit with input voltage $V(s)$ and input current $I(s)$, the impedance is defined as

$$Z(s) = \frac{V(s)}{I(s)}.$$

For the basic components, the impedances in the s–domain are

$$Z_R(s) = R, \qquad \text{for a resistor,}$$
$$Z_L(s) = sL, \qquad \text{for an inductor,}$$
$$Z_C(s) = \frac{1}{sC}, \qquad \text{for a capacitor.}$$

Series and parallel combinations. Once each element has been replaced by its impedance, ordinary circuit-combination rules apply. In series, impedances add directly:

$$Z = Z_1 + Z_2 + \cdots$$

In parallel, admittances add:

$$\frac{1}{Z} = \frac{1}{Z_1} + \frac{1}{Z_2} + \cdots$$

Figure 3.9 shows these two basic combinations.

Once an equivalent impedance is obtained, the transfer function follows from a voltage-divider or current-divider argument.

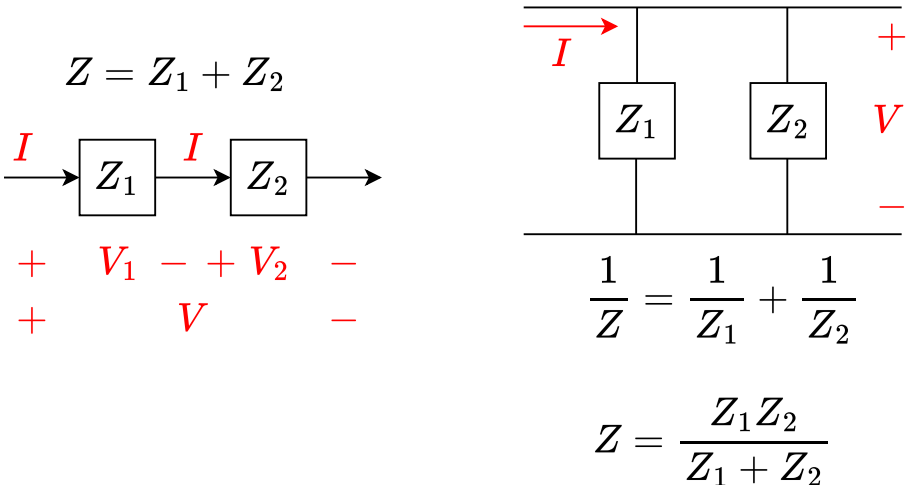


Figure 3.9: Series (left) and Parallel (right) connections of impedances.

Remark 3.5

This method avoids solving differential equations directly. Once the equivalent impedance $Z(s)$ is found, the transfer function follows immediately as

$$G(s) = \frac{V(s)}{I(s)} \quad \text{or} \quad G(s) = \frac{I(s)}{V(s)},$$

depending on the desired input-output relationship.

Example 3.5: Voltage divider with capacitor and inductor

Consider the two series networks in Fig. 3.10. Find the transfer function $G(s) = \dfrac{V_o(s)}{V_i(s)}$ where V_o is measured across the lower element (C_2 in (a), L_2 in (b)).

(a) Capacitive divider. The impedances are $Z_{C_1} = \dfrac{1}{sC_1}$ and $Z_{C_2} = \dfrac{1}{sC_2}$. By the voltage divider rule,

$$G(s) = \frac{V_o(s)}{V_i(s)} = \frac{Z_{C_2}}{Z_{C_1} + Z_{C_2}} = \frac{\dfrac{1}{sC_2}}{\dfrac{1}{sC_1} + \dfrac{1}{sC_2}} = \frac{C_1}{C_1 + C_2}.$$

Thus the gain is constant (independent of s): $G(s) = \dfrac{C_1}{C_1 + C_2}$.

(b) Inductive divider. The impedances are $Z_{L_1} = sL_1$ and $Z_{L_2} = sL_2$. Again using the divider rule,

$$G(s) = \frac{V_o(s)}{V_i(s)} = \frac{Z_{L_2}}{Z_{L_1} + Z_{L_2}} = \frac{sL_2}{s(L_1 + L_2)} = \frac{L_2}{L_1 + L_2}.$$

This gain is also constant: $G(s) = \dfrac{L_2}{L_1 + L_2}$.

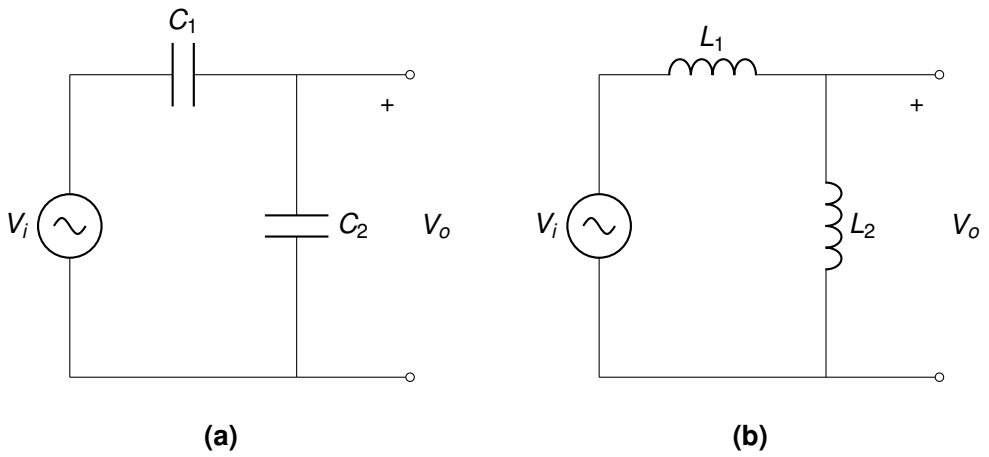


Figure 3.10: (a) Capacitive voltage divider. (b) Inductive voltage divider.

Example 3.6: Impedance Method for Transfer Function

Consider the circuits shown in Fig. 3.11. We wish to derive the transfer function

$$G(s) = \frac{V_o(s)}{V_i(s)}.$$

Using impedance equivalents, we identify

$$Z_1(s) = R_1 \left\| \frac{1}{sC_1}, \qquad Z_2(s) = R_2. \right.$$

Thus the transfer function is a simple voltage divider:

$$\frac{V_o(s)}{V_i(s)} = \frac{Z_2(s)}{Z_1(s) + Z_2(s)}.$$

Substituting the impedances:

$$G(s) = \frac{R_2}{\left(\dfrac{R_1}{1 + sR_1C_1}\right) + R_2}.$$

Example 3.7: A complex RLC circuit

Consider the circuit given in Fig 3.12. Considering the impedance equivalent of each component, derive the equations describing the system dynamics and obtain the transfer function from the input voltage source V_i to the output voltage V_o.

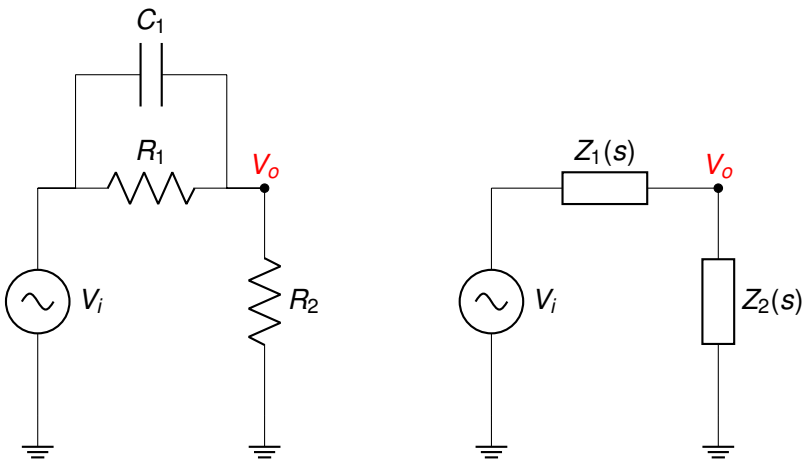


Figure 3.11: Comparison: physical RC circuit (left) and impedance model (right).

Consider the circuit in Figure 3.12. Using mesh currents I_1, I_2, I_3, I_4 and the impedances $Z_{L_k} = sL_k$, $Z_{C_k} = 1/(sC_k)$, the KVL equations are

$$
\begin{aligned}
sL_1 I_1 + \frac{1}{sC_1}(I_1 - I_2) &= V_i, \\
\frac{1}{sC_1}(I_2 - I_1) + R_1(I_2 - I_3) &= 0, \\
sL_2 I_3 + \frac{1}{sC_2}(I_3 - I_4) + R_1(I_3 - I_2) &= 0, \\
R_2 I_4 + \frac{1}{sC_2}(I_4 - I_3) &= 0.
\end{aligned}
$$

In matrix (mesh impedance) form,

$$
\underbrace{\begin{bmatrix}
Z_{L_1} + Z_{C_1} & -Z_{C_1} & 0 & 0 \\
-Z_{C_1} & Z_{C_1} + R_1 & -R_1 & 0 \\
0 & -R_1 & Z_{L_2} + R_1 + Z_{C_2} & -Z_{C_2} \\
0 & 0 & -Z_{C_2} & Z_{C_2} + R_2
\end{bmatrix}}_{Z(s)}
\underbrace{\begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix}}_{I(s)}
=
\underbrace{\begin{bmatrix} V_i \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{V(s)}.
$$

The currents can then be obtained with standard linear algebra techniques (e.g. $I(s) = Z(s)^{-1} V(s)$) or with the MATLAB Symbolic Toolbox as shown in code Listing 1. The output is $V_o(s) = R_2 I_4(s)$, hence $G(s) = V_o(s)/V_i(s) = R_2\, I_4(s)/V_i(s)$.

3.8 Ideal Operational Amplifier (Op-Amp)

An **operational amplifier (op-amp)** has two input terminals: the **inverting input** V_- and the **non-inverting input** V_+. Figure 3.13b shows a commercial op-amp (LM741) and its pin configuration.

The output voltage V_o is given by

$$V_o = K\,(V_+ - V_-),$$

where K is the open-loop gain of the amplifier.

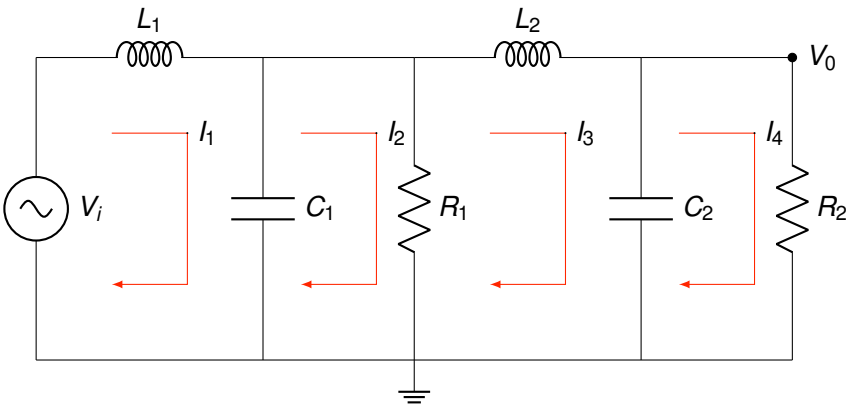


Figure 3.12: Example: RLC circuit

Listing 1 Symbolic computation of $G(s) = V_o(s)/V_i(s)$ for $L_1 = L_2 = 100\,\text{mH}$, $C_1 = C_2 = 10\,\text{mF}$, $R_1 = R_2 = 100\,\Omega$.

```
syms s Vi
L1=100e-3; L2=100e-3; C1=10e-3; C2=10e-3; R1=100; R2=100;

ZL1=s*L1; ZL2=s*L2;
ZC1=1/(s*C1); ZC2=1/(s*C2);

Z=[ ZL1+ZC1, -ZC1, 0, 0;
-ZC1, ZC1+R1, -R1, 0;
0, -R1, ZL2+R1+ZC2, -ZC2;
0, 0, -ZC2, ZC2+R2 ];

I = Z\[Vi;0;0;0];
G = simplify(R2*I(4)/Vi)
```

Ideal op-amp assumptions. We assume infinite gain ($K \to \infty$), infinite input impedance (zero input currents), and zero output impedance. With feedback, these assumptions imply

$$V_+ = V_-, \qquad i_+ = i_- = 0.$$

Example 3.8: Op-Amp Circuits

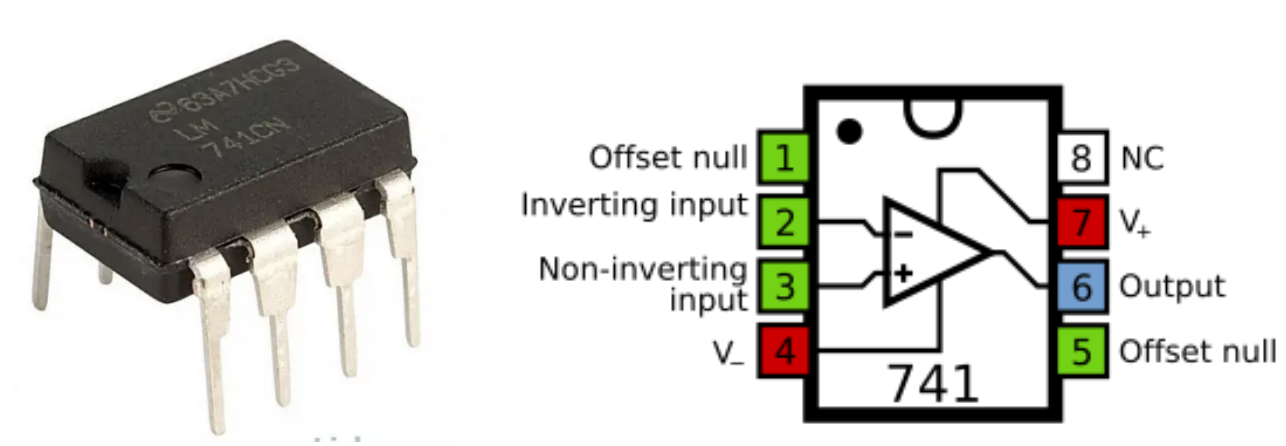
Find the transfer function V_o/V_i of the circuits shown in Fig. 3.15.

Solution:

Case (a): Non-inverting configuration. Here, $V_i(s) = V_+(s) = V_-(s)$. The input current into the op-amp is zero. By applying KCL at the inverting input:

$$\frac{V_i}{Z_1} = \frac{V_o - V_i}{Z_2} \quad \Rightarrow \quad \left(\frac{1}{Z_1} + \frac{1}{Z_2}\right) V_i = \frac{V_o}{Z_2}$$

$$\Rightarrow \quad \frac{V_o}{V_i} = \frac{Z_1 + Z_2}{Z_1}.$$



(a) An operational amplifier (LM741). (b) Pinout of LM741.

Figure 3.13: The LM741 operational amplifier and its pin configuration.

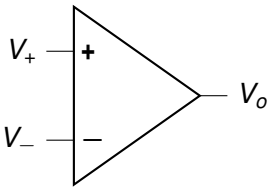


Figure 3.14: Symbol of the ideal operational amplifier.

Case (b): Inverting configuration. Here, $V_- = V_+ = 0$ (virtual ground). By KCL:

$$\frac{V_i}{Z_1} = -\frac{V_o}{Z_2} \quad \Rightarrow \quad \frac{V_o}{V_i} = -\frac{Z_2}{Z_1}.$$

Thus, the non-inverting case provides a positive gain greater than unity, while the inverting case provides a negative gain determined by the impedance ratio.

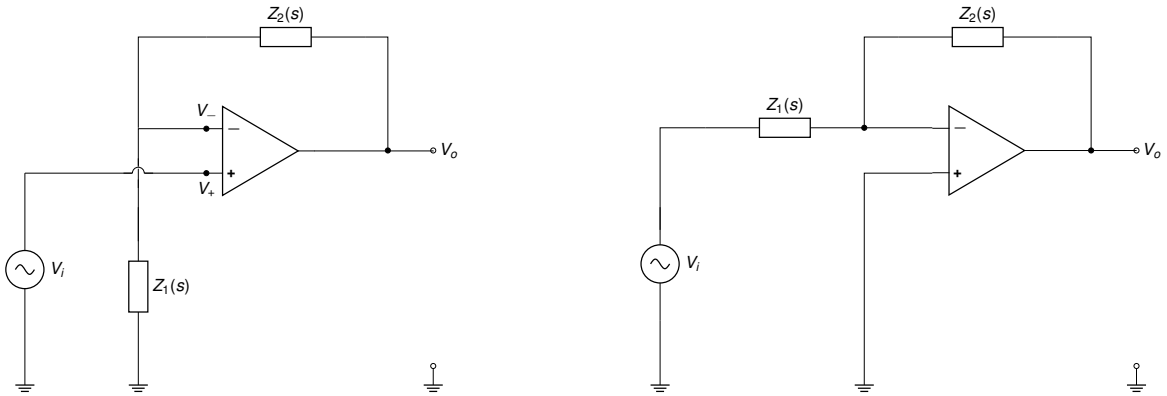


Figure 3.15: Op-amp circuits: (Left) non-inverting configuration and (Right) inverting configuration.

Example 3.9: A Circuit with Two Op-Amps

Find the overall transfer function $G(s) = V_o(s)/V_i(s)$ of the two-stage network shown in Figure 3.16. Assume zero initial conditions.

Solution. We split the analysis into two parts.

1. First stage: V_1/V_i

The first op-amp is the configuration of Example 2-7(b) with

$$Z_1(s) = \frac{1}{C_1 s} \Big\| R_1 = \frac{R_1}{C_1 R_1 s + 1},$$

$$Z_2(s) = \frac{1}{C_2 s} \Big\| R_2 = \frac{R_2}{C_2 R_2 s + 1}.$$

For an inverting op-amp,

$$\frac{V_1}{V_i} = -\frac{Z_2(s)}{Z_1(s)} = -\frac{\dfrac{R_2}{C_2 R_2 s + 1}}{\dfrac{R_1}{C_1 R_1 s + 1}} = -\frac{R_2}{R_1}\,\frac{C_1 R_1 s + 1}{C_2 R_2 s + 1}.$$

2. Second stage: V_o/V_1

Similarly, the second op-amp (also an inverting configuration) has

$$Z_1'(s) = \frac{R_3}{C_3 R_3 s + 1}, \qquad Z_2'(s) = \frac{R_4}{C_4 R_4 s + 1},$$

and

$$\frac{V_o}{V_1} = -\frac{Z_2'(s)}{Z_1'(s)} = -\frac{R_4}{R_3}\,\frac{C_3 R_3 s + 1}{C_4 R_4 s + 1}.$$

3. Overall transfer: V_o/V_i

The two stages cascade, so

$$G(s) = \frac{V_o}{V_i} = \frac{V_o}{V_1}\,\frac{V_1}{V_i} = \left(-\frac{R_4}{R_3}\,\frac{C_3 R_3 s + 1}{C_4 R_4 s + 1}\right) \times \left(-\frac{R_2}{R_1}\,\frac{C_1 R_1 s + 1}{C_2 R_2 s + 1}\right).$$

which simplifies to

$$G(s) = \frac{V_o(s)}{V_i(s)} = \frac{R_2\,R_4}{R_1\,R_3}\,\underbrace{\frac{C_1 R_1 s + 1}{C_2 R_2 s + 1}}_{\text{stage 1}} \times \underbrace{\frac{C_3 R_3 s + 1}{C_4 R_4 s + 1}}_{\text{stage 2}}.$$

3.9 Mechanical Systems : Translational motion

Translational motion is modelled with three basic elements: springs (react to displacement), dampers (react to velocity), and masses (react to acceleration). Figure 3.17 shows these elements and their constitutive relations.

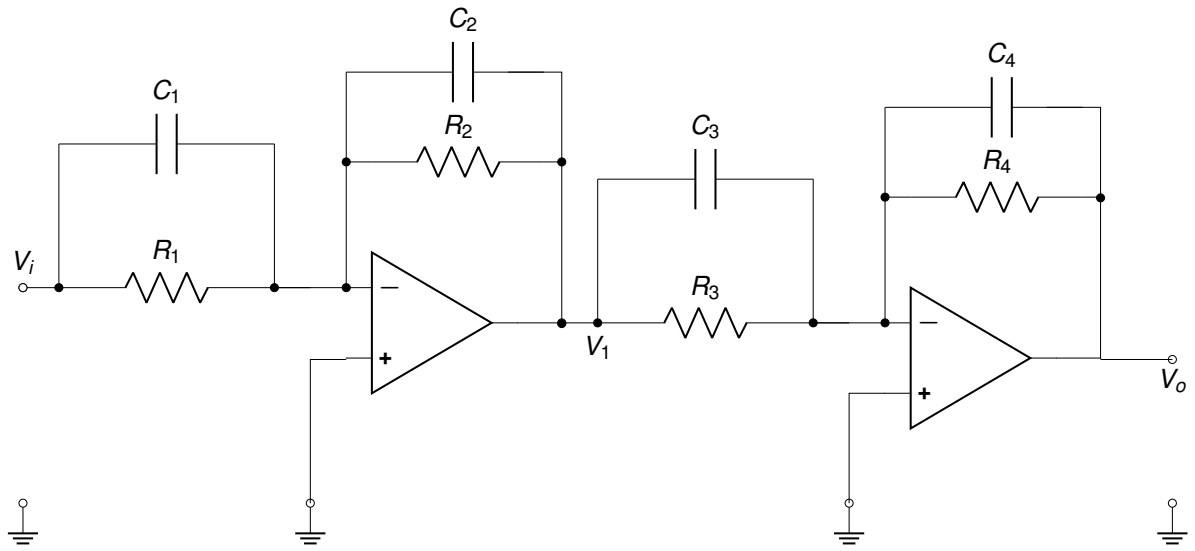


Figure 3.16: Cascade connection of 2-opamp filters.

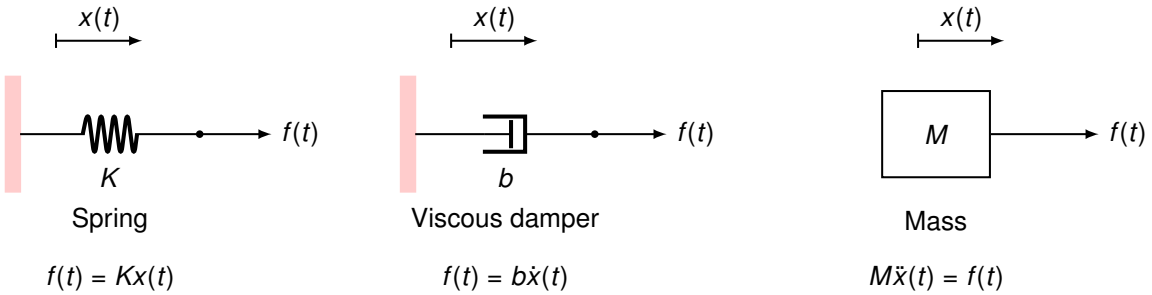


Figure 3.17: Basic mechanical components used in translational motion.

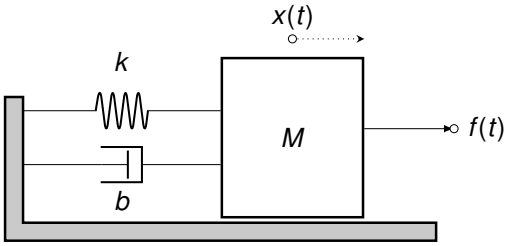


Figure 3.18: Simple mass-damper-spring system.

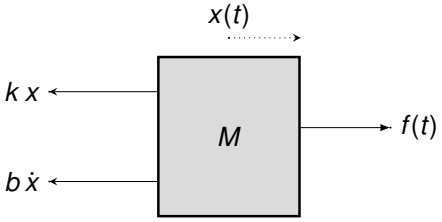


Figure 3.19: Simple mass-damper-spring system: Forces acting on mass M.

Spring. A linear spring stores potential energy and exerts a restoring force proportional to the displacement $x(t)$ from its equilibrium position. Its constitutive relation is

$$f(t) = K\,x(t), \qquad \underbrace{[\mathrm{N}]}_{f} = \underbrace{[\tfrac{\mathrm{N}}{\mathrm{m}}]}_{K} \times \underbrace{[\mathrm{m}]}_{x}.$$

Viscous damper. A viscous damper dissipates kinetic energy and produces a force proportional to relative velocity:

$$f(t) = b\,\dot{x}(t), \qquad \underbrace{[\mathrm{N}]}_{f} = \underbrace{[\tfrac{\mathrm{N{\cdot}s}}{\mathrm{m}}]}_{b} \times \underbrace{[\tfrac{\mathrm{m}}{\mathrm{s}}]}_{\dot{x}}.$$

Mass. A rigid mass resists acceleration according to Newton's second law:

$$M\,\ddot{x}(t) = f(t), \qquad \underbrace{[\mathrm{kg}]}_{M} \times \underbrace{[\tfrac{\mathrm{m}}{\mathrm{s}^2}]}_{\ddot{x}} = \underbrace{[\mathrm{N}]}_{f}.$$

Here $\dot{x}(t)$ denotes velocity and b is the damping coefficient, measured as force per unit velocity.

Example 3.10

Derive the equation of motion for the system shown in Figure 3.18. Assume that the system is initially at rest, so $x(0) = \dot{x}(0) = 0$.

We begin the modelling process by identifying all forces acting on the mass M and applying Newton's second law of motion:

$$\sum F = M\ddot{x},$$

where $\sum F$ denotes the net force acting on M.

Consider Figure 3.18. Since there is only one moving body, the analysis focuses on mass M and the forces applied to it. We choose the positive direction of motion to the right, with the external force $f(t)$ also acting in this direction.

The spring and damper, whose other ends are fixed to a rigid wall, oppose this motion. The spring exerts a restoring force proportional to displacement, $kx(t)$, and the damper exerts a resistive force

proportional to velocity, $b\dot{x}(t)$. Both of these forces act to the left, opposing the applied force. These forces are illustrated in Figure 3.19.

Thus, the equation of motion (EOM) is:

$$M\ddot{x}(t) = f(t) - kx(t) - b\dot{x}(t).$$

Taking the Laplace transform of both sides, and assuming zero initial displacement and velocity, we obtain:

$$(Ms^2 + bs + k)X(s) = F(s).$$

Therefore, the transfer function from the input force $f(t)$ to the displacement $x(t)$ is:

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

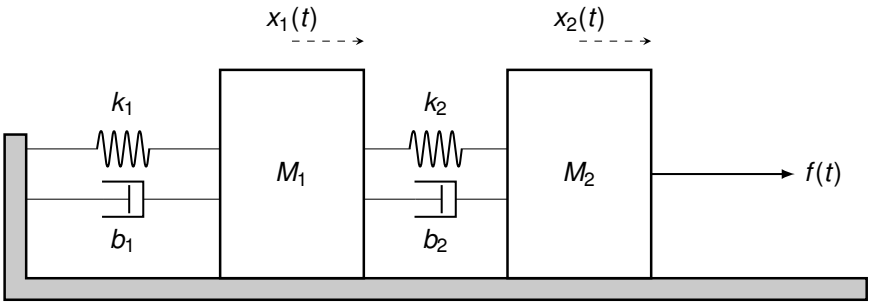


Figure 3.20: Two-mass system: M_1 attached to wall with spring and damper, M_2 connected to M_1 with another spring and damper, subject to external force $f(t)$.

Example 3.11

Derive the equations of motion for the two-mass system shown in Figure 3.20. Assume that the system is initially at rest, so $x_1(0) = \dot{x}_1(0) = x_2(0) = \dot{x}_2(0) = 0$.

We apply Newton's second law to each mass:

$$\sum F = M\ddot{x}.$$

For mass M_1, the wall spring and wall damper contribute the restoring and dissipative forces $-k_1x_1$ and $-b_1\dot{x}_1$. The coupling spring and damper contribute forces based on relative displacement and relative velocity, namely $-k_2(x_1 - x_2)$ and $-b_2(\dot{x}_1 - \dot{x}_2)$. Therefore

$$M_1\ddot{x}_1 = -k_1x_1 - b_1\dot{x}_1 - k_2(x_1 - x_2) - b_2(\dot{x}_1 - \dot{x}_2).$$

For mass M_2, the external force $f(t)$ acts to the right, while the coupling spring and damper oppose the relative motion. Hence

$$M_2\ddot{x}_2 = f(t) - k_2(x_2 - x_1) - b_2(\dot{x}_2 - \dot{x}_1).$$

These equations can be written compactly in matrix form as

$$\begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix} \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} + \begin{bmatrix} b_1 + b_2 & -b_2 \\ -b_2 & b_2 \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} + \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ f(t) \end{bmatrix}.$$

Assuming zero initial conditions and taking the Laplace transform gives

$$A(s) \begin{bmatrix} X_1(s) \\ X_2(s) \end{bmatrix} = \begin{bmatrix} 0 \\ F(s) \end{bmatrix},$$

where

$$A(s) = \begin{bmatrix} M_1 s^2 + (b_1 + b_2)s + (k_1 + k_2) & -(b_2 s + k_2) \\ -(b_2 s + k_2) & M_2 s^2 + b_2 s + k_2 \end{bmatrix}.$$

Solving this 2×2 system by matrix inversion gives

$$\begin{bmatrix} X_1(s) \\ X_2(s) \end{bmatrix} = \frac{1}{\Delta(s)} \begin{bmatrix} M_2 s^2 + b_2 s + k_2 & b_2 s + k_2 \\ b_2 s + k_2 & M_1 s^2 + (b_1 + b_2)s + (k_1 + k_2) \end{bmatrix} \begin{bmatrix} 0 \\ F(s) \end{bmatrix},$$

with

$$\Delta(s) = \left[M_1 s^2 + (b_1 + b_2)s + (k_1 + k_2)\right]\left[M_2 s^2 + b_2 s + k_2\right] - (b_2 s + k_2)^2.$$

Therefore,

$$X_1(s) = \frac{(b_2 s + k_2)F(s)}{\Delta(s)}, \quad X_2(s) = \frac{\left[M_1 s^2 + (b_1 + b_2)s + (k_1 + k_2)\right]F(s)}{\Delta(s)}.$$

Hence, the transfer function from the applied force to the displacement of M_2 is

$$\frac{X_2(s)}{F(s)} = \frac{M_1 s^2 + (b_1 + b_2)s + (k_1 + k_2)}{\Delta(s)}.$$

Listing 3.2 confirms the symbolic expressions for $X_1(s)$, $X_2(s)$, and the transfer function $X_2(s)/F(s)$ using MATLAB.

```
clear; clc;
syms M1 M2 b1 b2 k1 k2 s Fs

A_s = [M1*s^2+(b1+b2)*s+(k1+k2), -(b2*s+k2);
-(b2*s+k2), M2*s^2+b2*s+k2];
F_s = [0; Fs];

X_s = A_s \ F_s;
X1_s = simplify(X_s(1));
X2_s = simplify(X_s(2));

disp('X1(s):'); pretty(X1_s)
disp('X2(s):'); pretty(X2_s)

G2_s = simplify(X2_s/Fs);
disp('X2(s)/F(s):'); pretty(G2_s)

```

Listing 3.2: MATLAB verification of $X_1(s)$, $X_2(s)$, and transfer function $X_2(s)/F(s)$.

Example 3.12

Derive the equations of motion for the quarter-car suspension system shown in Figure 3.21. The model consists of a sprung mass M_s representing the vehicle body and an unsprung mass M_{us} representing the wheel assembly. The suspension is modelled by a spring k_s and a damper c_s, while the tire is modelled as a spring k_t. The road disturbance is $y_r(t)$. Assume initial rest conditions.

We apply Newton's second law to each mass. Let $y_s(t)$ denote the vertical displacement of the sprung mass and $y_{us}(t)$ the displacement of the unsprung mass. The positive direction is upward.

For the sprung mass, the only forces come from the suspension spring and damper, both of which depend on the relative motion between the body and the wheel. Therefore

$$M_s\ddot{y}_s = -k_s(y_s - y_{us}) - c_s(\dot{y}_s - \dot{y}_{us}).$$

For the unsprung mass, the suspension spring and damper act in the opposite direction, and the tire spring contributes an additional force based on the relative displacement between the wheel and the road. Hence

$$M_{us}\ddot{y}_{us} = -k_s(y_{us} - y_s) - c_s(\dot{y}_{us} - \dot{y}_s) - k_t(y_{us} - y_r).$$

The system can be written compactly as

$$\underbrace{\begin{bmatrix} M_s & 0 \\ 0 & M_{us} \end{bmatrix}}_{M} \begin{bmatrix} \ddot{y}_s \\ \ddot{y}_{us} \end{bmatrix} + \underbrace{\begin{bmatrix} c_s & -c_s \\ -c_s & c_s \end{bmatrix}}_{C} \begin{bmatrix} \dot{y}_s \\ \dot{y}_{us} \end{bmatrix} + \underbrace{\begin{bmatrix} k_s & -k_s \\ -k_s & k_s + k_t \end{bmatrix}}_{K} \begin{bmatrix} y_s \\ y_{us} \end{bmatrix} = \begin{bmatrix} 0 \\ k_t y_r \end{bmatrix}.$$

Assuming zero initial conditions and taking the Laplace transform gives

$$\left(Ms^2 + Cs + K\right) \begin{bmatrix} Y_s(s) \\ Y_{us}(s) \end{bmatrix} = \begin{bmatrix} 0 \\ k_t Y_r(s) \end{bmatrix}.$$

Let

$$A(s) = Ms^2 + Cs + K,$$

then

$$A(s) \begin{bmatrix} Y_s(s) \\ Y_{us}(s) \end{bmatrix} = \begin{bmatrix} 0 \\ k_t Y_r(s) \end{bmatrix}.$$

The solution is obtained by inverting $A(s)$:

$$\begin{bmatrix} Y_s(s) \\ Y_{us}(s) \end{bmatrix} = A(s)^{-1} \begin{bmatrix} 0 \\ k_t Y_r(s) \end{bmatrix}.$$

This system can be solved analytically by matrix inversion or numerically in MATLAB. For ride-comfort analysis, the transfer function from the road displacement $Y_r(s)$ to the body displacement $Y_s(s)$ is usually the quantity of greatest interest.

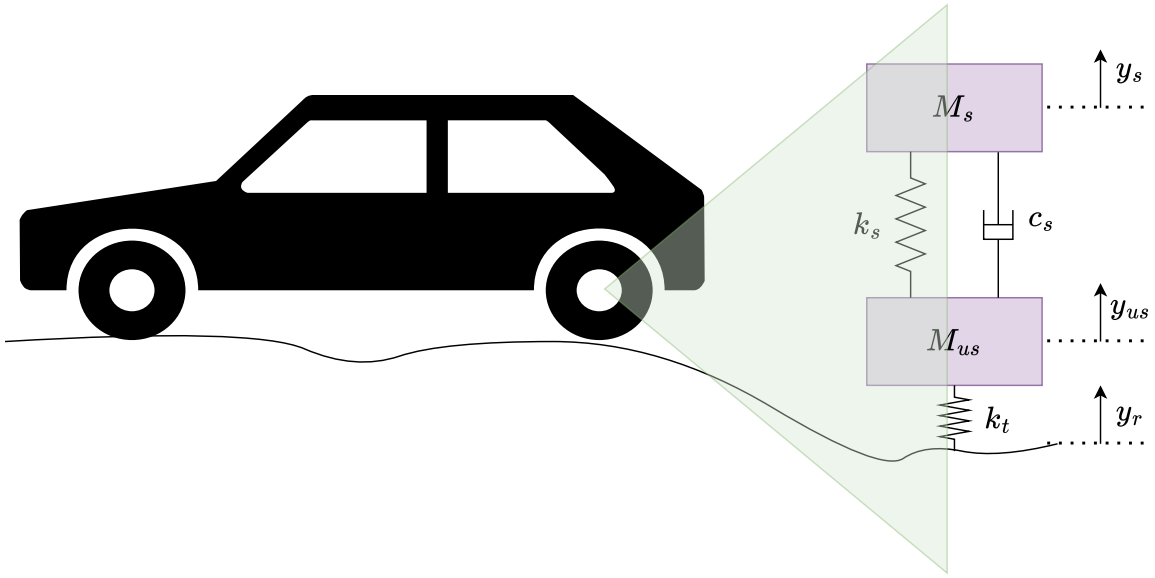
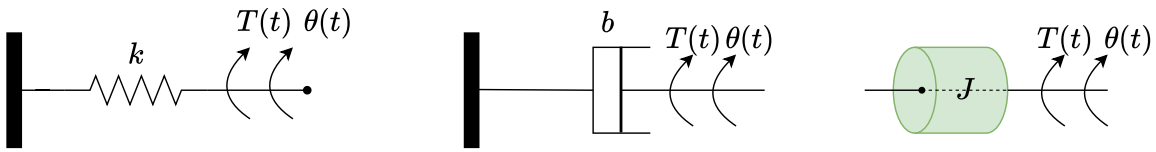


Figure 3.21: Suspension system and its simplified model.

Figure 3.22: A real suspension system used in cars.

3.10 Mechanical Systems : Rotational Systems

Rotational systems are modelled the same way as translational ones, with mass replaced by moment of inertia J, displacement by angle θ, velocity by $\dot{\theta}$, and force by torque T. A torsional spring obeys $T = k\,\theta$, a rotational damper obeys $T = b\,\dot{\theta}$, and Newton's second law for rotation gives $T = J\,\ddot{\theta}$.



Spring

$T(t) = k\,\theta(t)$

Viscous damper

$T(t) = b\,\dot{\theta}(t)$

Rotational inertial object

$J\,\ddot{\theta}(t) = T(t)$

Figure 3.23: Basic components of rotational mechanical systems.

Translational	**Rotational**	**Analogy**
Displacement x [m]	Angular displacement θ [rad]	Position
Velocity $\dot{x} = v$ [m/s]	Angular velocity $\dot{\theta} = \omega$ [rad/s]	Velocity
Acceleration $\ddot{x} = a$ [m/s^2]	Angular acceleration $\ddot{\theta} = \alpha$ [rad/s^2]	Acceleration
Mass m [kg]	Moment of inertia J [kg m^2]	Resistance to acceleration
Force F [N]	Torque T [N m]	Cause of motion
Spring: $F = k\,x$, k [N/m]	Spring: $T = k\,\theta$, k [N m/rad]	Restoring element
Damper: $F = b\,\dot{x}$, b [N s/m]	Damper: $T = b\,\dot{\theta}$, b [N m s/rad]	Energy dissipation
Inertia: $F = m\,\ddot{x}$, m [kg]	Inertia: $T = J\,\ddot{\theta}$, J [kg m^2]	Newton's law

Table 3.1: A comparison of basic components of translational and rotational mechanical systems.

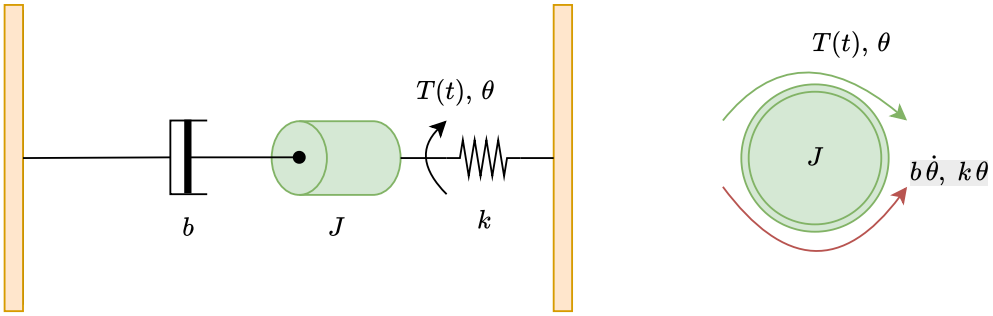


Figure 3.24: Example for a rotational system (left), torques acting on the inertial body J (right).

Example 3.13

Consider the system shown in Figure 3.24. The applied torque $T(t)$ causes the inertial body J to rotate in the clockwise direction, producing an angular displacement $\theta(t)$. However, the viscous damper (with damping constant b) and the torsional spring (with stiffness k) oppose this motion. The damper exerts a resisting torque of magnitude $b\,\dot{\theta}(t)$, while the spring produces a restoring torque of $k\,\theta(t)$, both acting in the opposite direction to $T(t)$.

Applying Newton's second law for rotation gives:

$$J\,\ddot{\theta}(t) = T(t) - b\,\dot{\theta}(t) - k\,\theta(t).$$

Taking the Laplace transform of both sides (assuming zero initial conditions) yields:

$$J\,s^2\Theta(s) = T(s) - b\,s\,\Theta(s) - k\,\Theta(s).$$

Rearranging terms:

$$\left(J\,s^2 + b\,s + k\right)\Theta(s) = T(s).$$

Thus, the transfer function from the input torque $T(s)$ to the angular displacement $\Theta(s)$ is:

$$\frac{\Theta(s)}{T(s)} = \frac{1}{J\,s^2 + b\,s + k}.$$

Example 3.14: Model of a torsional shaft

Consider the torsional system shown in Figure 3.25. In part (a), a torque $T(t)$ is applied to the left end of a shaft mounted on bearings at both ends. Because the shaft is elastic, it cannot be treated as a perfectly rigid body. Instead, it twists under the applied torque, so that the angular displacement at the input end differs from that at the output end. To capture this effect, the shaft is modeled in part (b) as two rotating inertias, J_1 and J_2, connected by a torsional spring of stiffness K. This spring accounts for the elastic twist of the shaft. The friction in the bearings is represented by viscous dampers b_1 and b_2, attached respectively to the two inertias, to model the energy dissipation due to bearing resistance.

Determine the transfer function between the input torque and the angular displacement of the second inertia, i.e.,

$$\frac{\Theta_2(s)}{T(s)}.$$

We write one rotational balance for J_1 and one for J_2. If J_2 is viewed temporarily as fixed at position θ_2, then a positive twist of J_1 by θ_1 produces a spring torque

$$k\big(\theta_1(t) - \theta_2(t)\big).$$

The damper b_1 contributes an opposing torque proportional to angular velocity:

$$b_1\dot{\theta}_1(t).$$

The dynamic equation for J_1 is therefore

$$J_1\,\ddot{\theta}_1(t) = T(t) - b_1\dot{\theta}_1(t) - k\big(\theta_1(t) - \theta_2(t)\big).$$

Now focus on J_2. When J_2 turns relative to J_1, the spring resists with a torque

$$k\big(\theta_2(t) - \theta_1(t)\big),$$

and the damper b_2 adds the torque

$$b_2\dot{\theta}_2(t).$$

Since no external torque is applied directly to J_2, its equation of motion is

$$J_2\,\ddot{\theta}_2(t) = -k\big(\theta_2(t) - \theta_1(t)\big) - b_2\dot{\theta}_2(t).$$

Taking the Laplace transform of both equations and assuming zero initial conditions gives

$$J_1\,s^2\,\Theta_1(s) = T(s) - b_1 s\Theta_1(s) - k\big(\Theta_1(s) - \Theta_2(s)\big), \tag{3.6}$$

$$J_2\,s^2\,\Theta_2(s) = -k\big(\Theta_2(s) - \Theta_1(s)\big) - b_2 s\Theta_2(s). \tag{3.7}$$

Rearranging (3.10) gives:

$$\big(J_1 s^2 + b_1 s + k\big)\Theta_1(s) - k\Theta_2(s) = T(s).$$

On the other hand, rearranging (3.7) gives:

$$-k\Theta_1(s) + \big(J_2 s^2 + b_2 s + k\big)\Theta_2(s) = 0.$$

Writing the equations in matrix form:

$$\begin{bmatrix} J_1 s^2 + b_1 s + k & -k \\ -k & J_2 s^2 + b_2 s + k \end{bmatrix} \begin{bmatrix} \Theta_1(s) \\ \Theta_2(s) \end{bmatrix} = \begin{bmatrix} T(s) \\ 0 \end{bmatrix}.$$

Inverting this system yields

$$\frac{\Theta_2(s)}{T(s)} = \frac{k}{(J_1 s^2 + b_1 s + k)(J_2 s^2 + b_2 s + k) - k^2}$$

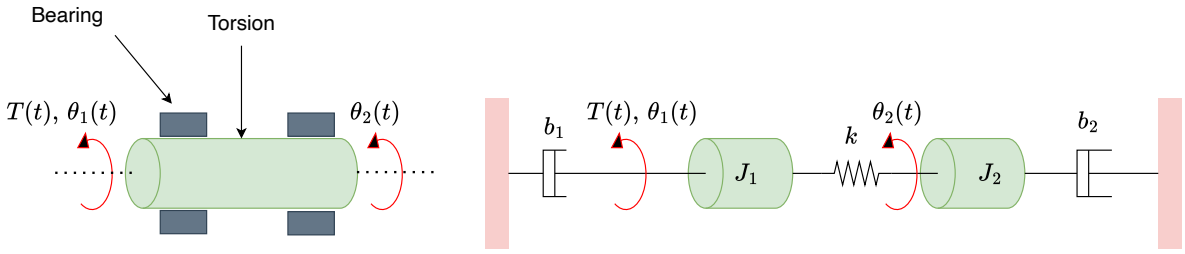


Figure 3.25: Model (b) of a torsional shaft(a). [10]

Example 3.15: A complex rotational system

Find the transfer function $\frac{\Theta_3(s)}{T(s)}$ for the rotational system shown in Figure 3.26.

Following the same procedure as in the previous examples, we write one equation of motion for each inertia J_i, $i = 1, \dots, 3$:

$$\begin{aligned} J_1\ddot{\theta}_1 &= T - k_1(\theta_1 - \theta_2) - b_1\dot{\theta}_1 \\ J_2\ddot{\theta}_2 &= -k_1(\theta_2 - \theta_1) - k_2(\theta_2 - \theta_3) - b_3(\dot{\theta}_2 - \dot{\theta}_3) - b_2\dot{\theta}_2 \\ J_3\ddot{\theta}_3 &= -k_2(\theta_3 - \theta_2) - b_3(\dot{\theta}_3 - \dot{\theta}_2) - b_4\dot{\theta}_3 \end{aligned}$$

Taking Laplace transforms (zero initial conditions assumed):

$$\begin{aligned} (J_1 s^2 + b_1 s + k_1)\Theta_1 - k_1\Theta_2 &= T(s) \\ -k_1\Theta_1 + (J_2 s^2 + (b_2 + b_3)s + (k_1 + k_2))\Theta_2 - (k_2 + b_3 s)\Theta_3 &= 0 \\ -(k_2 + b_3 s)\Theta_2 + (J_3 s^2 + (b_3 + b_4)s + k_2)\Theta_3 &= 0 \end{aligned}$$

These equations can then be written in matrix form as

$$\underbrace{\begin{bmatrix} J_1 s^2 + b_1 s + k_1 & -k_1 & 0 \\ -k_1 & J_2 s^2 + (b_2 + b_3)s + (k_1 + k_2) & -(k_2 + b_3 s) \\ 0 & -(k_2 + b_3 s) & J_3 s^2 + (b_3 + b_4)s + k_2 \end{bmatrix}}_{\triangleq A(s)} \begin{bmatrix} \Theta_1 \\ \Theta_2 \\ \Theta_3 \end{bmatrix} = \begin{bmatrix} T(s) \\ 0 \\ 0 \end{bmatrix}.$$

Letting $\boldsymbol{\Theta}(s) = [\Theta_1, \Theta_2, \Theta_3]^\top$, we compactly have

$$A(s)\,\boldsymbol{\Theta}(s) = \begin{bmatrix} T(s) \\ 0 \\ 0 \end{bmatrix}.$$

Solving for $\boldsymbol{\Theta}(s)$ gives

$$\boldsymbol{\Theta}(s) = A(s)^{-1} \begin{bmatrix} T(s) \\ 0 \\ 0 \end{bmatrix}.$$

Therefore, the transfer function is obtained from the $(3, 1)$ element of $A(s)^{-1}$:

$$\frac{\Theta_3(s)}{T(s)} = \left[A(s)^{-1}\right]_{31}.$$

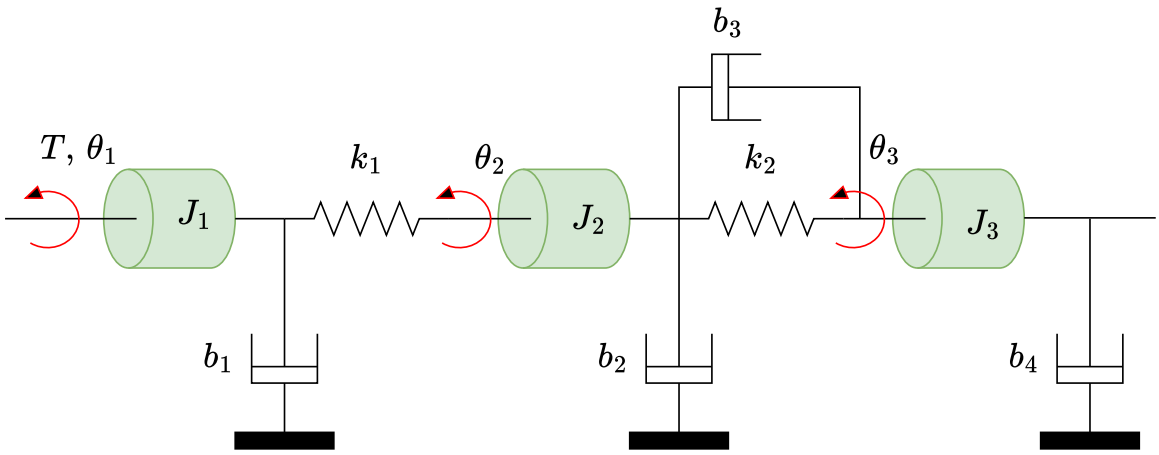


Figure 3.26: A rotational system with three inertial bodies.

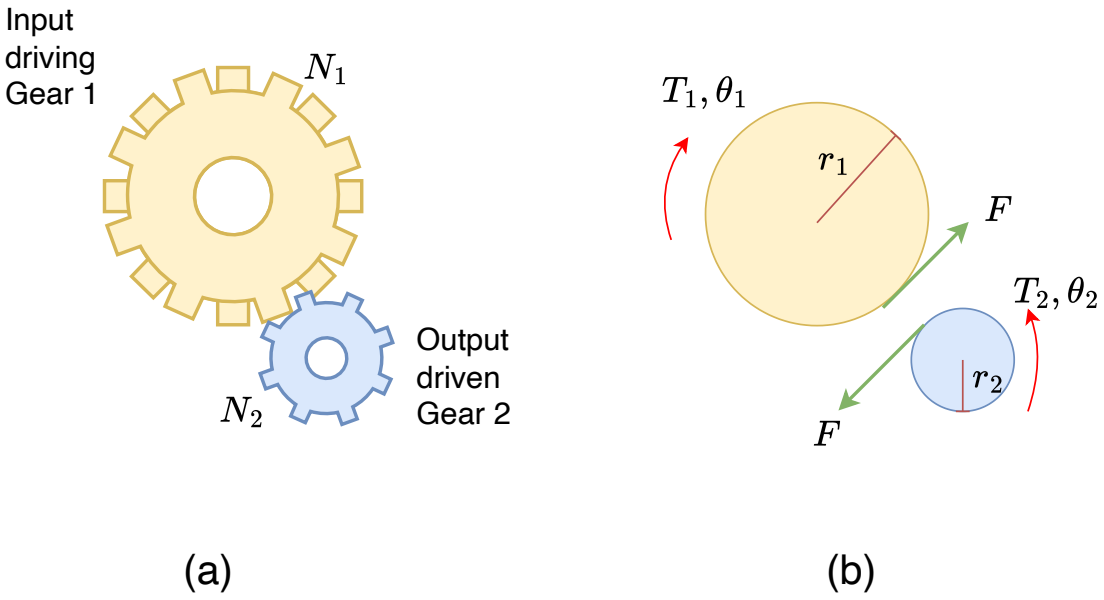


Figure 3.27: Gear system (a) and the torques, forces and geometry of the gears (b)

3.11 Gears in mechanical systems

Gears trade speed for torque and impose geometric constraints between angular displacement, velocity, and torque.

Consider the gear pair in Figure 3.27. Gear 1 is the input gear, with radius r_1, tooth count N_1, and angular displacement θ_1. Gear 2 is the output gear, with radius r_2, tooth count N_2, and angular displacement θ_2.

At the point of contact, assuming no slipping, the arc lengths must be equal:

$$r_1\theta_1 = r_2\theta_2$$

Taking time derivatives yields the angular velocity relationship:

$$r_1\omega_1 = r_2\omega_2$$

The number of teeth on each gear is proportional to its radius, hence:

$$\frac{\theta_2}{\theta_1} = \frac{\omega_2}{\omega_1} = \frac{r_1}{r_2} = \frac{N_1}{N_2}$$

where N_1/N_2 is the **gear ratio**.

At the contact point, the tangential force F is equal and opposite on both gears. Since torque is the product of force and radius:

$$T_1 = Fr_1 \qquad T_2 = Fr_2$$

Thus, the torque relationship becomes:

$$\frac{T_1}{r_1} = \frac{T_2}{r_2} \quad \Rightarrow \quad \frac{T_1}{T_2} = \frac{r_1}{r_2}$$

Therefore, the torque relationship between the two gears becomes:

$$\frac{T_2}{T_1} = \frac{r_2}{r_1} = \frac{N_2}{N_1}.$$

Assuming there is no frictional loss,

$$T_2 = J_2 s^2 \Theta_2$$

Since $T_1 = \frac{r_1}{r_2} T_2$

$$T_1 = \frac{r_1}{r_2} T_2 = \frac{r_1}{r_2} J_2 s^2 \Theta_2 = \left(\frac{r_1}{r_2}\right)^2 J_2 s^2 \Theta_1$$

Remark 3.6

Secondary side inertias and damping coefficients translate to the primary side by multiplying the term by the square of the gear ratio, $\left(\frac{r_1}{r_2}\right)^2$.

Example 3.16: Motor driving a load through gears

Consider the motor-load system shown in Figure 3.28. A motor with moment of inertia J_m and viscous damping b_m drives a load with inertia J_L and damping b_L through an ideal gear train with gear ratio $\frac{N_1}{N_2}$, where N_1 and N_2 are the number of teeth on the motor-side and load-side gears respectively. The motor applies torque T_m, and rotates with angular displacement θ_m, while the load rotates with θ_L. Derive the transfer functions $\frac{\theta_m(s)}{T_m(s)}$ and $\frac{\theta_L(s)}{T_m(s)}$.

In an ideal gear, angular displacements and torques are related by

$$\frac{N_1}{N_2}\theta_m = \theta_L, \qquad T_1 = \frac{N_1}{N_2} T_2,$$

where T_1 is the torque exerted by the motor gear on the load gear, and T_2 is the torque acting on the load. T_1 appears as a resistive torque in the motor's dynamics, while T_2 drives the load.

The equation of motion for the load is

$$J_L s^2 \theta_L = T_2 - b_L s \theta_L \Rightarrow T_2 = (J_L s^2 + b_L s)\theta_L.$$

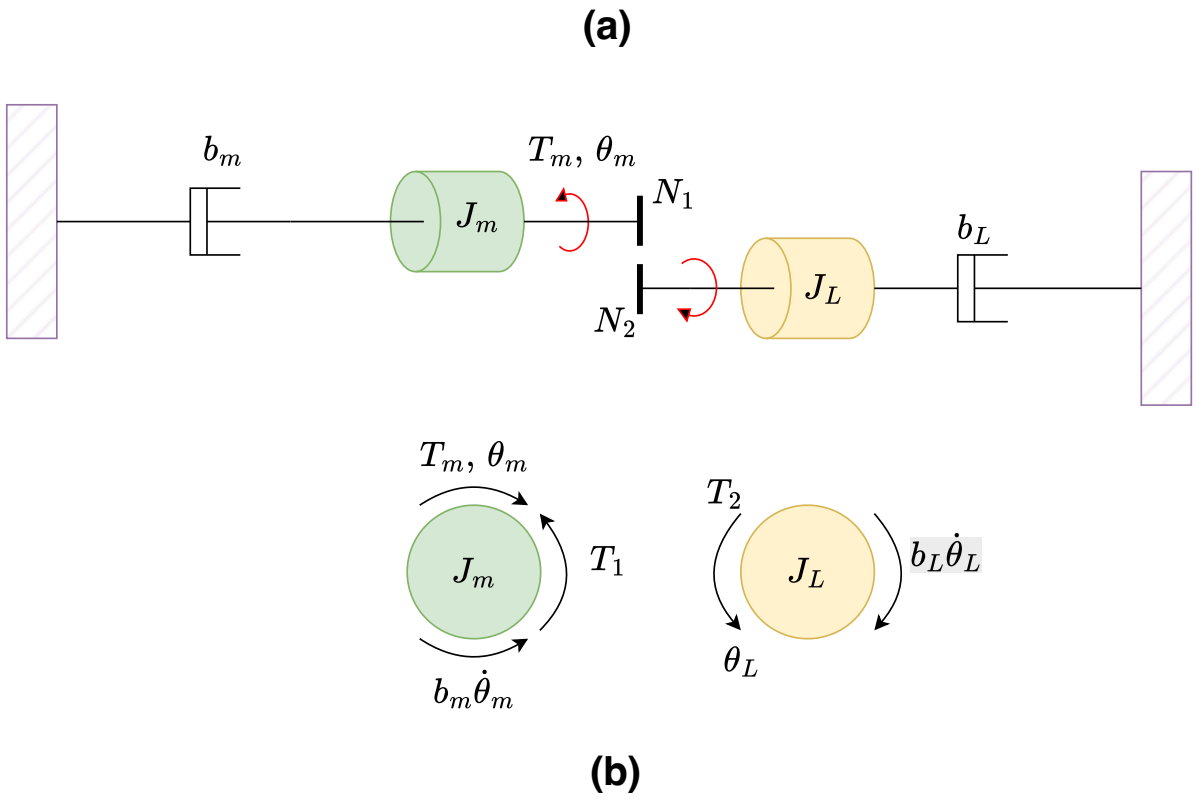


Figure 3.28: (a) Motor–load system (b) torques and forces acting on the motor and load.

Substituting $\theta_L = \frac{N_1}{N_2}\theta_m$ gives

$$T_2 = \left(J_L s^2 + b_L s\right) \frac{N_1}{N_2}\theta_m.$$

Then

$$T_1 = \frac{N_1}{N_2} T_2 = \left(\frac{N_1}{N_2}\right)^2 \cdot \left(J_L s^2 + b_L s\right) \cdot \theta_m$$

The motor's equation of motion is

$$J_m s^2\theta_m = T_m - b_m s\theta_m - T_1.$$

Substituting T_1,

$$J_m s^2\theta_m = T_m - b_m s\theta_m - \left(\frac{N_1}{N_2}\right)^2 \left(J_L s^2 + b_L s\right)\theta_m,$$

In terms of the gear ratio, the load inertia and damping reflected to the motor side are $\left(\frac{N_1}{N_2}\right)^2 J_L$ and $\left(\frac{N_1}{N_2}\right)^2 b_L$, so the correct reflected equation is

$$\left(J_m + \left(\frac{N_1}{N_2}\right)^2 J_L\right) s^2\theta_m + \left(b_m + \left(\frac{N_1}{N_2}\right)^2 b_L\right) s\theta_m = T_m.$$

Define

$$J_e = J_m + \left(\frac{N_1}{N_2}\right)^2 J_L, \qquad b_e = b_m + \left(\frac{N_1}{N_2}\right)^2 b_L,$$

then

$$J_e s^2 \theta_m + b_e s \theta_m = T_m.$$

The transfer functions are

$$\frac{\theta_m(s)}{T_m(s)} = \frac{1}{s(J_e s + b_e)}, \qquad \frac{\theta_L(s)}{T_m(s)} = \frac{N_1}{N_2} \cdot \frac{1}{s(J_e s + b_e)}.$$

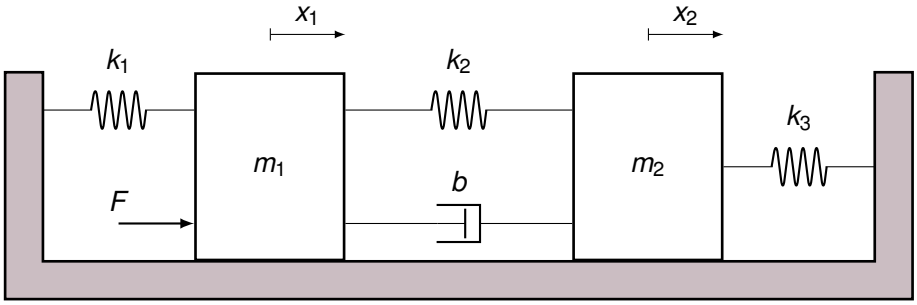


Figure 3.29: Mechanical system consisting of two masses m_1 and m_2, connected by springs and a viscous damper, subject to an external force F.

Example 3.17: Electrical-analogue solution via node equations

Consider the two mass system and its electrical analogue in Fig. 3.29.

Collecting terms gives the linear system

$$\underbrace{\begin{bmatrix} k_1 + s^2 m_1 + k_2 + sb & -(k_2 + sb) \\ -(k_2 + sb) & k_3 + s^2 m_2 + k_2 + sb \end{bmatrix}}_{A(s)} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}.$$

Let $a_{11} = k_1 + s^2 m_1 + k_2 + sb$, $a_{22} = k_3 + s^2 m_2 + k_2 + sb$, and $a_{12} = a_{21} = -(k_2 + sb)$. With $\Delta(s) = a_{11}a_{22} - a_{12}a_{21} = (k_1 + s^2 m_1 + k_2 + sb)(k_3 + s^2 m_2 + k_2 + sb) - (k_2 + sb)^2$, the inverse of $A(s)$ yields

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \frac{1}{\Delta(s)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \begin{bmatrix} F \\ 0 \end{bmatrix} = \frac{1}{\Delta(s)} \begin{bmatrix} a_{22} F \\ -(-(k_2 + sb))\, F \end{bmatrix}.$$

Hence the transfer function from force input to the displacement of the second mass is

$$\frac{X_2(s)}{F(s)} = \frac{k_2 + b\,s}{\Delta(s)}$$

where

$$\Delta(s) = m_1 m_2 s^4 + b\,(m_1 + m_2)\, s^3 + \left[m_1(k_2 + k_3) + m_2(k_1 + k_2)\right] s^2 + b\,(k_1 + k_3)\, s + (k_1 k_2 + k_2 k_3 + k_3 k_1)$$

The MATLAB code in Listing 3.3 verifies this result symbolically.

```
syms s k1 k2 k3 m1 m2 c F real

A = [k1 + s^2*m1 + k2 + s*c, -(k2 + s*c);
-(k2 + s*c), k3 + s^2*m2 + k2 + s*c];

b = [F; 0];

X = A\b;     % X=[X1; X2]
X1 = simplify(X(1));
X2 = simplify(X(2));

G = simplify(X2/F);
```

Listing 3.3: Symbolic verification of $X_2(s)/F(s)$ for the two-mass electrical-analogue system.

Example 3.18: Matrix solution via impedance analogy

Consider the mechanical system in Figure 3.18. The node equations are

$$ms^2X_1 + k(X_1 - X_2) = F,$$
$$k(X_2 - X_1) + bs\,X_2 = 0.$$

These can be written compactly as a 2×2 matrix equation

$$\underbrace{\begin{bmatrix} ms^2 + k & -k \\ -k & k + bs \end{bmatrix}}_{A(s)} \underbrace{\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}}_{X(s)} = \underbrace{\begin{bmatrix} F \\ 0 \end{bmatrix}}_{B(s)}.$$

Therefore, $X(s) = A^{-1}(s)B(s)$ The determinant of $A(s)$ is

$$\begin{aligned} \Delta(s) = \det A(s) &= (ms^2 + k)(k + bs) - k^2 \\ &= ms^2(k + bs) + kbs \\ &= mb\,s^3 + mk\,s^2 + kb\,s. \end{aligned}$$

Using $A(s)^{-1} = \frac{1}{\Delta(s)} \begin{bmatrix} k + bs & k \\ k & ms^2 + k \end{bmatrix}$,

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \frac{1}{\Delta(s)} \begin{bmatrix} k + bs & k \\ k & ms^2 + k \end{bmatrix} \begin{bmatrix} F \\ 0 \end{bmatrix} = \frac{1}{\Delta(s)} \begin{bmatrix} (k + bs)\,F \\ k\,F \end{bmatrix}.$$

Hence the transfer functions are

$$\boxed{\frac{X_1(s)}{F(s)} = \frac{k + bs}{mb\,s^3 + mk\,s^2 + kb\,s} = \frac{k + bs}{s\,(mb\,s^2 + mk\,s + kb)}}$$

$$\boxed{\frac{X_2(s)}{F(s)} = \frac{k}{mb\,s^3 + mk\,s^2 + kb\,s} = \frac{k}{s\,(mb\,s^2 + mk\,s + kb)}}.$$

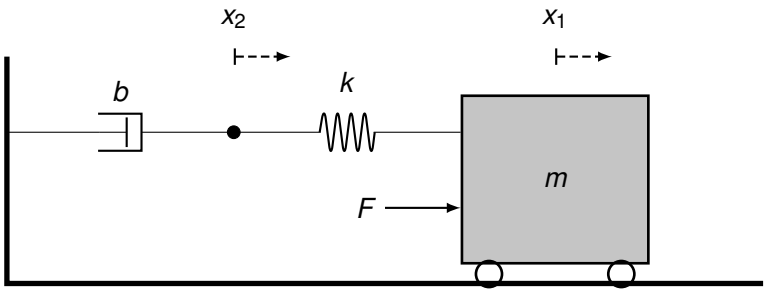


Figure 3.30: Mechanical system with a damper and spring

Example 3.19

Consider the three-mass mechanical system shown in Figure 3.31. Obtain the equations of motion using the electrical analogy and determine the transfer function from the applied force F to the displacement x_3.

Let us consider the forces on each mass m_i, $i = 1, \dots, 3$, and apply Newton's second law to each component. Assume m_2 and m_3 are fixed at x_2 and x_3, respectively, while m_1 moves to the right by an amount x_1. Then Newton's law gives

$$m_1\ddot{x}_1 = F - k_1x_1 - b_1\dot{x}_1 - b_2(\dot{x}_1 - \dot{x}_3) - k_2(x_1 - x_2)$$

Similarly, for m_2 and m_3, one can write:

$$\begin{aligned} m_2\ddot{x}_2 &= -k_2(x_2 - x_1) - k_3(x_2 - x_3) \\ m_3\ddot{x}_3 &= -b_2(\dot{x}_3 - \dot{x}_1) - k_3(x_3 - x_2) \end{aligned}$$

Taking the Laplace Transform, the equation of motion can be writen in matrix form as follows:

$$\underbrace{\begin{bmatrix} s^2m_1 + sb_1 + k_1 + k_2 + sb_2 & -k_2 & -sb_2 \\ -k_2 & s^2m_2 + k_2 + k_3 & -k_3 \\ -sb_2 & -k_3 & s^2m_3 + k_3 + sb_2 \end{bmatrix}}_{A(s)} \begin{bmatrix} X_1(s) \\ X_2(s) \\ X_3(s) \end{bmatrix} = \begin{bmatrix} F(s) \\ 0 \\ 0 \end{bmatrix}.$$

Let $X(s) = [X_1(s)\ X_2(s)\ X_3(s)]^\top$ and $A(s)$ be as above. Solving $A(s)X(s) = [F(s)\ 0\ 0]^\top$ gives $G(s) = \dfrac{X_3(s)}{F(s)}$. The MATLAB code in Listing 3.4 constructs $A(s)$ symbolically and computes $G(s)$, together with its collected numerator and denominator polynomials in s.

For reference, $G(s)$ can be written compactly as

$$G(s) = \frac{X_3(s)}{F(s)} = \frac{k_2k_3 + s\,b_2(k_2 + k_3) + s^3b_2m_2}{\det A(s)}$$

with $\det A(s)$ being the sixth-order polynomial obtained from expanding the 3×3 determinant. Because $\det A(s)$ is lengthy, it is best computed symbolically; the MATLAB code below does exactly that.

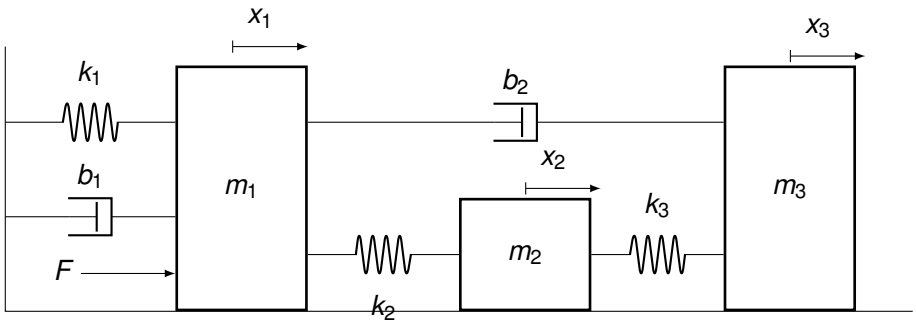


Figure 3.31: Triple mass system.

```
syms s m1 m2 m3 b1 b2 k1 k2 k3 F real
assumeAlso([m1 m2 m3 b1 b2 k1 k2 k3] > 0)

A = [ s^2*m1 + s*b1 + k1 + k2 + s*b2,    -k2,      -s*b2;
-k2,       s^2*m2 + k2 + k3,    -k3;
-s*b2,      -k3,       s^2*m3 + k3 + s*b2 ];

rhs = [F; 0; 0];

X   = simplify(A\rhs);        % [X1; X2; X3]
G   = simplify(X(3)/F);       % Transfer function X3/F

[num, den] = numden(G);
num = collect(expand(num), s);
den = collect(expand(den), s);

disp('G(s) = X3(s)/F(s) =');   pretty(G)
disp('Numerator:');            pretty(num)
disp('Denominator:');          pretty(den)

```

Listing 3.4: Symbolic computation of $G(s) = X_3(s)/F(s)$ for the triple-mass system.

3.12 Utilisation of RC Circuits for Thermal Modelling

The electrical–thermal analogy maps temperatures to voltages, heat flows to currents, thermal capacitance to electrical capacitance, and thermal resistance to electrical resistance.

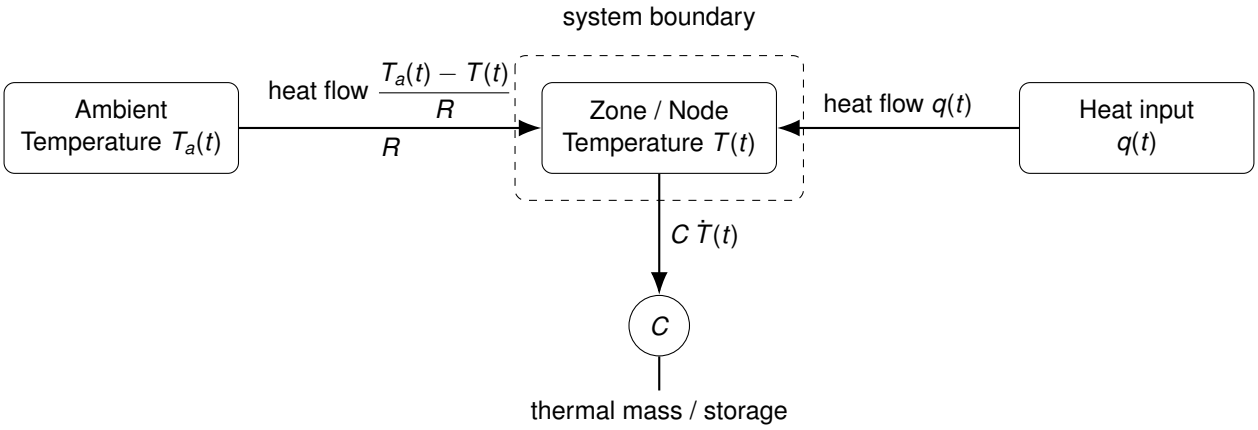
Consider a single-zone enclosure with temperature $T(t)$, ambient temperature $T_a(t)$, heat input $q(t)$ (e.g. from a heater or solar gain), thermal capacitance C and an overall thermal resistance R between the enclosure and the ambient environment. The heat-energy balance is

$$C\,\dot{T}(t) \;=\; \frac{T_a(t) - T(t)}{R} \;+\; q(t),$$

which has exactly the same mathematical form as the first-order RC equation in the electrical domain.

Taking the Laplace transform (assuming zero initial conditions) gives

$$(Cs + 1/R)\,T(s) \;=\; \frac{T_a(s)}{R} \;+\; Q(s).$$



$$C\,\dot{T}(t) = \frac{T_a(t) - T(t)}{R} + q(t)$$

Figure 3.32: Lumped thermal RC model. The zone temperature $T(t)$ stores heat in thermal capacitance C and exchanges heat with the ambient $T_a(t)$ through thermal resistance R, while receiving an external heat input $q(t)$.

Using the electrical thermal analogy (voltage $\leftrightarrow$ temperature, current $\leftrightarrow$ heat flow, resistance $\leftrightarrow$ thermal resistance, capacitance $\leftrightarrow$ thermal capacitance), the house in Fig. 3.33 is modelled as an *RC* ladder with two thermal nodes: the *envelope* temperature T_e (walls/roof/fabric) and the *indoor air* temperature T_i. Heat inputs are represented by current sources: the heater Q_h acts directly on the indoor air node, while solar gains Q_s mainly warm the envelope. The ambient temperature T_o is a voltage source connected to the envelope through the external resistance R_e; the indoor air exchanges heat with the envelope through the internal resistance R_i. The envelope and indoor air store heat in the capacitances C_e and C_i respectively. This structure captures the thermal inertia and time lag of the building fabric and is the thermal counterpart of a two-capacitor, two-resistor *RC* network.

Applying energy balance at each node gives the differential equations

$$C_i\,\dot{T}_i(t) = \frac{T_e(t) - T_i(t)}{R_i} \;+\; Q_h(t), \tag{3.8}$$

$$C_e\,\dot{T}_e(t) = \frac{T_i(t) - T_e(t)}{R_i} \;+\; \frac{T_o(t) - T_e(t)}{R_e} \;+\; Q_s(t). \tag{3.9}$$

Equations (3.8)–(3.9) are first-order *RC* balances with units of watts on both sides.

Taking Laplace transforms with zero initial conditions (standard transfer-function assumption) gives

$$\left(C_i s + \tfrac{1}{R_i}\right) T_i(s) - \tfrac{1}{R_i}\,T_e(s) = Q_h(s), \tag{3.10}$$

$$-\tfrac{1}{R_i}\,T_i(s) + \left(C_e s + \tfrac{1}{R_i} + \tfrac{1}{R_e}\right) T_e(s) = \tfrac{1}{R_e}\,T_o(s) + Q_s(s). \tag{3.11}$$

Solving (3.10) and (3.11) for $T_i(s)$ yields

$$T_i(s) = \frac{\left(C_e s + \tfrac{1}{R_i} + \tfrac{1}{R_e}\right) Q_h(s) + \tfrac{1}{R_i}\,Q_s(s) + \tfrac{1}{R_i R_e}\,T_o(s)}{\left(C_i s + \tfrac{1}{R_i}\right)\left(C_e s + \tfrac{1}{R_i} + \tfrac{1}{R_e}\right) - \left(\tfrac{1}{R_i}\right)^2}.$$

Equivalently, the indoor temperature is the superposition of three inputs,

$$T_i(s) = G_h(s)\,Q_h(s) + G_s(s)\,Q_s(s) + G_o(s)\,T_o(s),$$

with transfer functions

$$G_h(s) = \frac{C_e s + \tfrac{1}{R_i} + \tfrac{1}{R_e}}{D(s)}, \qquad G_s(s) = \frac{\tfrac{1}{R_i}}{D(s)}, \qquad G_o(s) = \frac{\tfrac{1}{R_i R_e}}{D(s)},$$

where

$$D(s) = \left(C_i s + \frac{1}{R_i}\right)\left(C_e s + \frac{1}{R_i} + \frac{1}{R_e}\right) - \left(\frac{1}{R_i}\right)^2.$$

These expressions provide $T_i(s)$ directly in the s-domain without resorting to a state-space formulation.

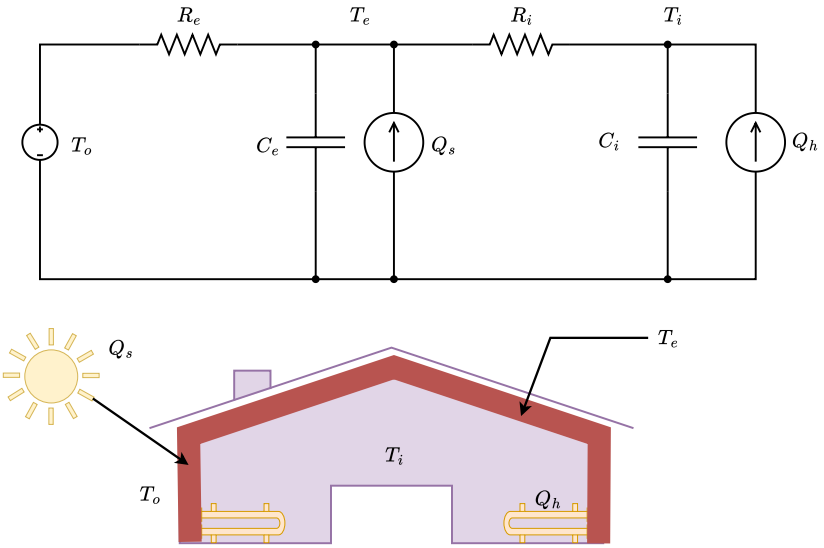


Figure 3.33: Electrical *analogy* of the thermodynamic model of a house.

3.13 DC Motor System

DC motors are common actuators in control: armature current determines torque, input voltage influences speed. They are an ideal example of coupled electrical and mechanical dynamics.

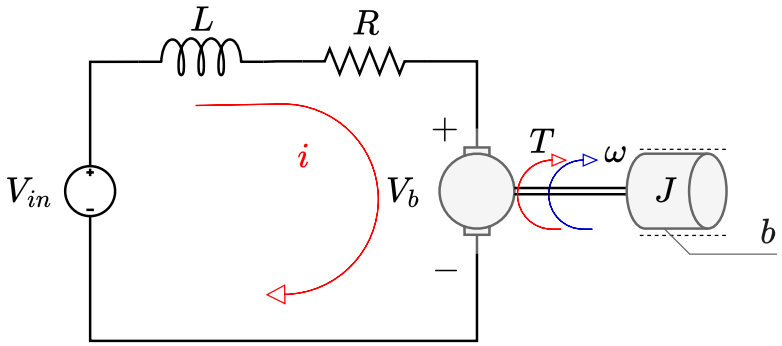


Figure 3.34: Electro-mechanical Structure of a Permanent Magnet DC motor

The system shown in Figure 3.34 represents a permanent-magnet DC motor connected to a mechanical load. It is helpful to think of the model as two coupled parts: an electrical subsystem that governs the armature current, and a mechanical subsystem that governs the shaft speed.

The key quantities are the input voltage V_{in}, armature resistance R, armature inductance L, armature current i, back electromotive force V_b, torque constant K_m, back-EMF constant K_b, motor torque T, combined inertia J, viscous friction coefficient b, and shaft angular velocity ω.

Figure 3.35: A commercial PMDC motor.

Applying KVL to the armature circuit:

$$V_{in} = L\frac{di}{dt} + Ri + V_b$$

Newton's second law for the shaft rotation gives:

$$T = J\dot{\omega} + b\,\omega$$

The electrical and mechanical subsystems are linked by two constitutive laws. The motor torque is proportional to the armature current:

$$T = K_m i.$$

The back EMF is proportional to the shaft speed:

$$V_b = K_b\,\omega.$$

In SI units, the torque constant and back EMF constant are numerically equal: $K_m = K_b$.

Taking the Laplace transform (zero initial conditions):

The E.O.M.s become:

$$\begin{aligned}(Ls + R)I(s) + V_b(s) &= V_{in}(s)\\ T(s) &= K_m I(s)\\ (Js + b)\Omega(s) &= T(s)\\ V_b(s) &= K_b\Omega(s)\end{aligned}$$

where $I(s)$, $V_{in}(s)$, $V_b(s)$, $T(s)$, and $\Omega(s)$ are the Laplace transforms of the corresponding time-domain variables.

From these algebraic equations, we can read off several useful intermediate transfer relationships:

$$\frac{I(s)}{V_{in}(s) - V_b(s)} = \frac{1}{Ls + R}, \qquad \frac{T(s)}{I(s)} = K_m,$$
$$\frac{\Omega(s)}{T(s)} = \frac{1}{Js + b}, \qquad \frac{V_b(s)}{\Omega(s)} = K_b.$$

We eliminate $I(s)$, $T(s)$, and $V_b(s)$ to obtain the overall transfer function $\Omega(s)/V_{in}(s)$.

Expressing the current as a function of speed:

$$\begin{aligned}K_m I(s) &= (Js + b)\Omega(s)\\ \Longrightarrow\ I(s) &= \frac{Js + b}{K_m}\Omega(s)\end{aligned}$$

Hence

$$V_{in}(s) = (Ls + R)\left(\frac{Js + b}{K_m}\Omega(s)\right) + K_b\Omega(s)$$

Factoring out $\Omega(s)$:

$$V_{in}(s) = \left[\frac{LJs^2 + (Lb + RJ)s + Rb + K_mK_b}{K_m}\right]\Omega(s)$$

The transfer function from input voltage to angular velocity is:

$$G(s) = \frac{\Omega(s)}{V_{in}(s)} = \frac{K_m}{LJs^2 + (Lb + RJ)s + (Rb + K_mK_b)}$$

The denominator is the characteristic polynomial; its roots determine the damping ratio and natural frequency of the motor's response.

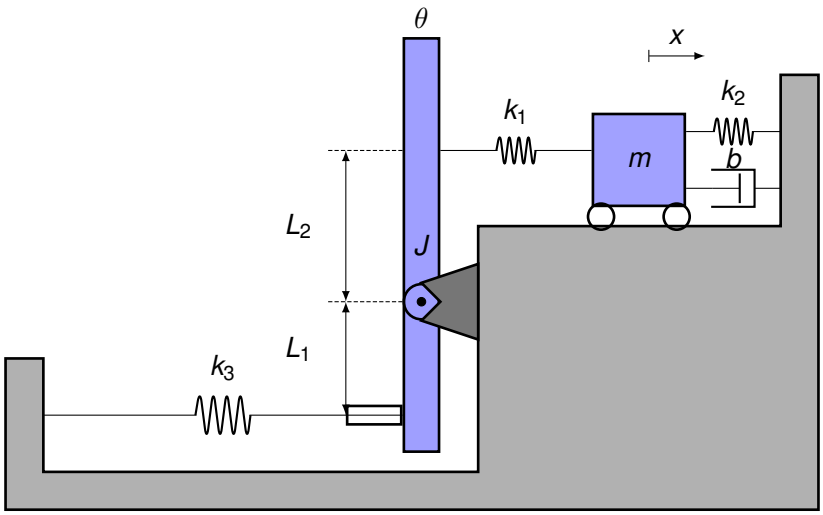


Figure 3.36: Coupled mechanical system

3.14 Examples on Modelling

Example 3.20: Coupled mechanical system

This example demonstrates the derivation of the equations of motion for the coupled mechanical system shown in Figure 3.36. We define the translational coordinate of the cart as x (positive to the right) and the rotational coordinate of the link as θ (positive clockwise).

Applying Newton's Second Law for the translation of mass m yields:

$$\sum F_x = m\ddot{x}$$
$$-k_1(x - L_1\sin\theta) - k_2x - b\dot{x} = m\ddot{x}$$

Rearranging gives the first equation of motion:

$$m\ddot{x} + b\dot{x} + (k_1 + k_2)x - k_1L_1\sin\theta = 0$$

For the rotation of the link with moment of inertia J about its pivot, the sum of torques gives:

$$\sum \tau = J\ddot{\theta}$$
$$k_1(x - L_1 \sin\theta)L_1 \cos\theta - k_3(L_2 \sin\theta)L_2 \cos\theta = J\ddot{\theta}$$

Rearranging gives the second equation of motion:

$$J\ddot{\theta} + (k_1 L_1^2 + k_3 L_2^2)\sin\theta\cos\theta - k_1 L_1 x \cos\theta = 0$$

For control system analysis, it is often useful to linearize the system about its equilibrium position ($\theta = 0$). Using the small angle approximations, $\sin\theta \approx \theta$ and $\cos\theta \approx 1$, the equations become:

$$m\ddot{x} + b\dot{x} + (k_1 + k_2)x - k_1 L_1 \theta = 0$$
$$J\ddot{\theta} - k_1 L_1 x + (k_1 L_1^2 + k_3 L_2^2)\theta = 0$$

These linearized equations can be conveniently expressed in the standard matrix form $M\ddot{q} + C\dot{q} + Kq = 0$, where the generalized-coordinate vector is $q = \begin{bmatrix} x & \theta \end{bmatrix}^T$.

$$\begin{bmatrix} m & 0 \\ 0 & J \end{bmatrix}\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} b & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \dot{x} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} k_1 + k_2 & -k_1 L_1 \\ -k_1 L_1 & k_1 L_1^2 + k_3 L_2^2 \end{bmatrix}\begin{bmatrix} x \\ \theta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

To obtain a first-order linear state-space model, define the state vector

$$z = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}.$$

From the linearized equations,

$$\ddot{x} = -\frac{b}{m}\dot{x} - \frac{k_1 + k_2}{m}x + \frac{k_1 L_1}{m}\theta, \qquad \ddot{\theta} = \frac{k_1 L_1}{J}x - \frac{k_1 L_1^2 + k_3 L_2^2}{J}\theta.$$

Therefore, the linear state-space equations are

$$\dot{z} = Az,$$

with

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\dfrac{k_1 + k_2}{m} & -\dfrac{b}{m} & \dfrac{k_1 L_1}{m} & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{k_1 L_1}{J} & 0 & -\dfrac{k_1 L_1^2 + k_3 L_2^2}{J} & 0 \end{bmatrix}.$$

If we choose the cart displacement and link angle as outputs, then

$$y = Cz, \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \qquad D = 0.$$

MATLAB Companion: Simulating a Derived Model

Try it yourself

Once you have derived a transfer function from physical laws, MATLAB lets you simulate its behaviour in just a few lines. Listing 3.5 creates a mass-spring-damper system and plots its step response and response to a custom force input.

```
% Mass-spring-damper: m*x'' + b*x' + k*x = F(t)
% Transfer function: G(s) = 1 / (m*s^2 + b*s + k)
m = 1;  b = 3;  k = 2;            % system parameters
G = tf(1, [m b k]);               % create transfer function

% Step response: apply a unit step force and see displacement
figure;
step(G, 10);                      % simulate for 10 seconds
title('Mass-Spring-Damper Step Response');
grid on;

% Custom input: ramp force F(t) = 2t for 5 seconds
t = 0:0.01:5;                     % time vector
F = 2*t;                          % ramp force input
figure;
lsim(G, F, t);                    % simulate with custom input
title('Response to Ramp Force F(t) = 2t');
grid on;

% Display poles to check system behaviour
disp('System poles:');
pole(G)
```

Listing 3.5: Simulating a mass-spring-damper transfer function in MATLAB.

Key Takeaways

- A transfer function $G(s) = Y(s)/U(s)$ captures the input–output dynamics of an LTI system under zero initial conditions.
- Electrical circuits are modelled using KVL/KCL with component relations, or more efficiently via the impedance method ($Z_R = R$, $Z_L = sL$, $Z_C = 1/(sC)$).
- Translational mechanical systems use $F = ma$ with springs (Kx), dampers ($b\dot{x}$), and masses ($m\ddot{x}$); rotational systems follow the same pattern with torque, angle, and inertia.
- Gears impose the constraint $\theta_L/\theta_m = N_1/N_2$, and load-side inertia and damping reflect to the motor side scaled by the square of the gear ratio.
- The DC motor transfer function $\Omega(s)/V_{in}(s) = K_m/[LJs^2 + (Lb + RJ)s + (Rb + K_mK_b)]$ couples electrical and mechanical dynamics.
- The electrical–thermal analogy allows RC-network methods to model heat flow in buildings, devices, and thermal systems.
- Multi-body systems lead to matrix equations $A(s)\,X(s) = B(s)$ that can be solved symbolically or numerically in MATLAB.

End-of-Chapter Exercises

1. **Series RLC circuit.** For a series RLC circuit with $R = 4\ \Omega$, $L = 1$ H, $C = \frac{1}{3}$ F, and the output taken across the capacitor:
   (a) Write the differential equation relating $v_C(t)$ to the input voltage $v(t)$.
   (b) Derive the transfer function $G(s) = V_C(s)/V(s)$.
   (c) Find the poles of the system and classify them (real distinct, repeated, or complex).
   (d) Verify your transfer function in MATLAB using `tf()` and `pole()`.

2. **Mass–spring–damper.** A mass $m = 2$ kg is attached to a spring with stiffness $k = 8$ N/m and a damper with coefficient $b = 4$ Ns/m. An external force $F(t)$ is applied to the mass. Taking displacement $x(t)$ as the output:
   (a) Draw a free-body diagram and write the equation of motion.
   (b) Derive $G(s) = X(s)/F(s)$.
   (c) Use MATLAB to plot the step response. What is the steady-state displacement for a unit step force?

3. **Two-mass system.** Two masses $m_1 = 1$ kg and $m_2 = 2$ kg are connected by a spring $k = 5$ N/m. A force $F(t)$ is applied to m_1 only. Write the equations of motion for both masses and express them in Laplace-domain matrix form $A(s)\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}$.

4. **Gear ratio.** A motor with inertia $J_m = 0.01\ \text{kg}\,\text{m}^2$ drives a load with inertia $J_L = 1\ \text{kg}\,\text{m}^2$ through a gear train with $N_1 = 10$ teeth (motor side) and $N_2 = 50$ teeth (load side). Neglect damping.
   (a) Compute the equivalent inertia J_e referred to the motor side.
   (b) Find the transfer function $\theta_m(s)/T_m(s)$.
   (c) Find the transfer function $\theta_L(s)/T_m(s)$.

5. **Rotational system.** A disc with moment of inertia $J = 0.5\ \text{kg}\,\text{m}^2$ and rotational damping $b = 2$ Nm s/rad is driven by a torque $T(t)$. Derive the transfer function $\Theta(s)/T(s)$ and find the time constant.

6. **Thermal system.** A room has thermal capacitance $C = 5000$ J/°C and thermal resistance $R = 0.2$ °C/W to the outside environment. A heater supplies heat $q(t)$ watts.
   (a) Write the first-order ODE governing the room temperature deviation from ambient.
   (b) Derive the transfer function $T(s)/Q(s)$ and identify the time constant.
   (c) If the heater applies a constant 500 W step input, what is the steady-state temperature rise?

7. **Op-amp integrator.** An inverting op-amp circuit has input resistor $R = 10\ \text{k}\Omega$ and feedback capacitor $C = 1\ \mu\text{F}$. Derive the transfer function $V_{out}(s)/V_{in}(s)$ and verify that it acts as an integrator.

8. **MATLAB verification.** For the mass–spring–damper in Exercise 2, use `lsim()` to simulate the response to a ramp input $F(t) = 3t$ over the interval $[0, 10]$ seconds. Plot the result and comment on the steady-state behaviour.

Chapter 4

State-Space Modelling

Learning Objectives

After completing this chapter, you should be able to:

- Explain why state-space models are used instead of (or alongside) transfer functions
- Identify state variables for mechanical, electrical, and thermal systems
- Convert higher-order ODEs into first-order state-space form $\dot{x} = Ax + Bu$, $y = Cx + Du$
- Derive state-space models from physical system equations, including coupled and nonlinear systems
- Compute the transfer function from a state-space model using $G(s) = C(sI - A)^{-1}B + D$
- Create, simulate, and convert state-space models in MATLAB using `ss`, `tf`, `lsim`, and related commands

4.1 Introduction to State-Space Modelling

Earlier chapters described dynamic systems using differential equations and transfer functions. Transfer functions are a powerful tool for analysing linear time-invariant systems, but they emphasise input–output behaviour and can be awkward to use for systems with multiple inputs and outputs or when the internal dynamics matter.

Modern control therefore often uses the State-space representation, which models a system through a set of State variables that summarise its internal condition (or memory) at a given time. The collection of these variables forms the State vector

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}.$$

A key idea is that, given the current state $x(t_0)$ and the input $u(t)$ for $t \geq t_0$, the future behaviour of the system is determined.

For a linear time-invariant system, the state-space model is written as two matrix equations. The State equation describes how the state evolves,

$$\dot{x}(t) = Ax(t) + Bu(t),$$

and the *output equation* relates the state to the measured output,

$$y(t) = Cx(t) + Du(t).$$

The matrices A, B, C, and D capture the system dynamics and how the input influences both the state and the output.

4.2 Motivation for the State-Space Approach

Transfer functions are particularly convenient for frequency-domain input–output analysis. However, they do not explicitly encode the system's internal dynamics, and extending the framework beyond SISO systems can be cumbersome. Many modelling and control tasks are also most naturally stated in terms of time-domain state trajectories and constraints.

The state-space approach resolves these issues by introducing a vector of state variables and rewriting the dynamics as a set of coupled first-order differential equations. This formulation

- makes the internal behaviour explicit,
- extends naturally to multi-input multi-output systems,
- is well suited to simulation and digital implementation, and
- underpins modern analysis and design techniques such as linear quadratic control, model predictive control, and sliding mode control.

4.3 State Variables and the State Vector

Definition 4.1: State of a dynamical system

More formally, the *state* of a system at time t_0 is defined as the minimum set of variables such that knowledge of these variables at t_0, together with the input $u(t)$ for $t \geq t_0$, uniquely determines the system's future response.

If n state variables are required, they are typically denoted by $x_1(t), x_2(t), \ldots, x_n(t)$ and collected into the *state vector*

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} \in \mathbb{R}^n.$$

The number of state variables n is called the *order* of the system. For a single-input single-output linear system described by an nth-order ordinary differential equation, the state dimension is typically n.

4.3.1 Choosing states

Different (but equivalent) state choices are possible for the same physical system. A good choice typically:

- uses physically meaningful variables (often energy-storing quantities),
- leads to a compact first-order model,
- aligns with what can be measured or estimated in practice.

Remark 4.1

In practice, when you write the dynamic (motion/energy-balance) equations of a physical system, the variables whose derivatives appear are natural candidates for the state variables. The state vector is then formed by collecting these variables (often energy-storing quantities) and rewriting the model as a set of *first-order* differential equations.

4.3.2 From higher-order dynamics to first-order state equations

A recurring step in state–space modelling is to convert higher-order dynamics into an equivalent first-order form by defining appropriate states. The next three examples illustrate the process for mechanical, electrical, and thermal systems.

Mechanical example: mass–spring–damper

Consider a translational mass–spring–damper system shown in Figure 4.1 driven by an external force $u(t)$, with displacement $y(t)$:

$$m\ddot{y}(t) + c\dot{y}(t) + ky(t) = u(t). \tag{4.1}$$

The derivatives in (4.1) suggest selecting displacement and velocity as states:

$$x_1(t) = y(t), \qquad x_2(t) = \dot{y}(t).$$

Then $\dot{x}_1(t) = x_2(t)$ and, rearranging (4.1),

$$\dot{x}_2(t) = \ddot{y}(t) = -\frac{k}{m}x_1(t) - \frac{c}{m}x_2(t) + \frac{1}{m}u(t).$$

Hence the mechanical dynamics can be written in state form as

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t), \qquad y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t),$$

where $x = [x_1 \; x_2]^\top$.

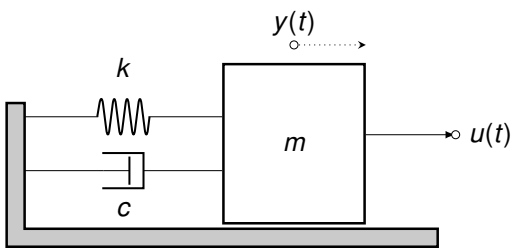


Figure 4.1: Simple mass-damper-spring system.

Electrical example: series RLC circuit

Consider a series RLC circuit shown in Figure 4.2 driven by an input voltage $u(t)$, and let the output be the capacitor voltage $y(t) = v_C(t)$.

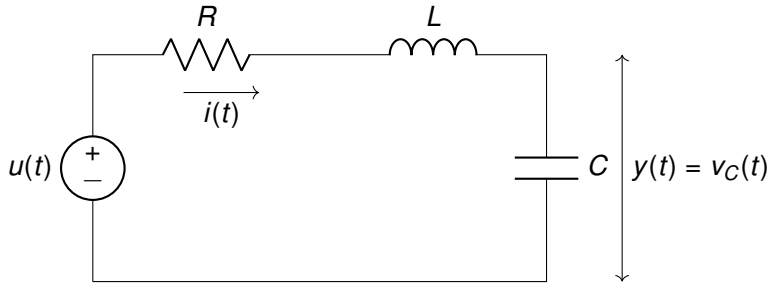


Figure 4.2: Series RLC circuit with input voltage $u(t)$ and output capacitor voltage $y(t) = v_C(t)$.

With series current $i(t)$, KVL gives

$$u(t) = v_R(t) + v_L(t) + v_C(t) = Ri(t) + L\dot{i}(t) + v_C(t), \tag{4.2}$$

and the capacitor relation is

$$i(t) = C\dot{v}_C(t). \tag{4.3}$$

Here the energy-storing elements are the inductor and capacitor, so a standard choice of states is

$$x_1(t) = v_C(t), \qquad x_2(t) = i(t).$$

From (4.3) we obtain

$$\dot{x}_1(t) = \dot{v}_C(t) = \frac{1}{C} i(t) = \frac{1}{C} x_2(t).$$

From (4.2) we solve for $\dot{i}(t)$:

$$L\dot{i}(t) = u(t) - Ri(t) - v_C(t) \quad \Rightarrow \quad \dot{x}_2(t) = -\frac{1}{L} x_1(t) - \frac{R}{L} x_2(t) + \frac{1}{L} u(t).$$

Thus the electrical dynamics are

$$\dot{x}(t) = \begin{bmatrix} 0 & \frac{1}{C} \\ -\frac{1}{L} & -\frac{R}{L} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} u(t), \qquad y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t),$$

with $x = [x_1 \; x_2]^\top = [v_C \; i]^\top$.

Thermal example: lumped thermal mass

Consider a single lumped thermal mass with temperature $T(t)$, thermal capacity C_{th}, and heat loss to ambient temperature T_a through thermal resistance R_{th}. Let the input be applied heat rate $u(t)$ (W). An energy balance gives

$$C_{\text{th}} \dot{T}(t) = -\frac{1}{R_{\text{th}}} \big(T(t) - T_a \big) + u(t). \tag{4.4}$$

This model is already first-order, so a natural state choice is the temperature:

$$x_1(t) = T(t).$$

Then (4.4) becomes

$$\dot{x}_1(t) = -\frac{1}{C_{\text{th}} R_{\text{th}}} x_1(t) + \frac{1}{C_{\text{th}} R_{\text{th}}} T_a + \frac{1}{C_{\text{th}}} u(t).$$

If T_a is constant, it can be treated as a known bias term; alternatively, one may model $(T - T_a)$ as the state. With output $y(t) = T(t)$, we have $y(t) = x_1(t)$.

4.4 State-Space Representation of Dynamic Systems

The *state-space representation* expresses a dynamic system as a set of first-order differential equations together with an algebraic output relation. It extends naturally to multiple-input multiple-output (MIMO) systems.

Throughout this chapter we consider continuous-time linear time-invariant (LTI) systems and use the notation

$$x(t) \in \mathbb{R}^n, \qquad u(t) \in \mathbb{R}^m, \qquad y(t) \in \mathbb{R}^p.$$

In this module we will mainly focus on the single-input single-output (SISO) case, i.e. $m = 1$ and $p = 1$, but the formulas are presented in the general form to keep the notation consistent with modern control texts.

4.4.1 State Equation

The *state equation* describes how the internal state evolves over time. For a continuous-time LTI system it is written as

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{4.5}$$

where

$$A \in \mathbb{R}^{n\times n}, \qquad B \in \mathbb{R}^{n\times m}.$$

The matrix A is called the State matrix (or *system* matrix) and captures the intrinsic dynamics of the system (how the state evolves when $u(t) = 0$). The matrix B is the *input* matrix and determines how the input drives the state.

Remark 4.2

For a continuous-time model, the state equation is always *first order* in time. Higher-order dynamics are represented by increasing the dimension n of the state vector.

In practice, (4.5) is obtained by selecting states and rewriting the original governing equations (often higher-order ODEs) as a set of coupled first-order equations, as shown in the earlier mechanical, electrical, and thermal examples.

4.4.2 Output Equation

The *output equation* specifies which variables are measured or of interest as outputs, and how they depend on the state and input:

$$y(t) = Cx(t) + Du(t), \tag{4.6}$$

where

$$C \in \mathbb{R}^{p\times n}, \qquad D \in \mathbb{R}^{p\times m}.$$

The matrix C is the *output* matrix: it selects and combines state variables to form the outputs. The matrix D is the Feedthrough matrix (or *direct transmission*) matrix: it represents any instantaneous effect of the input on the output.

Remark 4.3

If $D \neq 0$, then the output depends *instantaneously* on the input. Many physical systems have $D = 0$, but direct feedthrough can arise when the measured output includes a component of the input (for example, certain sensor/actuator models).

For the SISO case used most often in this module, $y(t) \in \mathbb{R}$ and (4.6) becomes

$$y(t) = Cx(t) + Du(t), \qquad C \in \mathbb{R}^{1\times n}, \;\; D \in \mathbb{R}.$$

4.4.3 Compact Matrix Form

Equations (4.5)–(4.6) together define the state-space model:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t). \end{aligned} \tag{4.7}$$

We state the general multi-input multi-output (MIMO) case first:

$$x(t) \in \mathbb{R}^n, \qquad u(t) \in \mathbb{R}^m, \qquad y(t) \in \mathbb{R}^p,$$

with

$$A \in \mathbb{R}^{n\times n}, \quad B \in \mathbb{R}^{n\times m}, \quad C \in \mathbb{R}^{p\times n}, \quad D \in \mathbb{R}^{p\times m}.$$

To make the structure explicit, write the vectors and matrices in terms of their entries:

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}, \qquad u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_m(t) \end{bmatrix}, \qquad y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_p(t) \end{bmatrix}.$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{bmatrix},$$

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{p1} & c_{p2} & \cdots & c_{pn} \end{bmatrix}, \qquad D = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1m} \\ d_{21} & d_{22} & \cdots & d_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ d_{p1} & d_{p2} & \cdots & d_{pm} \end{bmatrix}.$$

With this notation, the ith component of the state equation is

$$\dot{x}_i(t) = \sum_{j=1}^{n} a_{ij}\, x_j(t) \;+\; \sum_{\ell=1}^{m} b_{i\ell}\, u_\ell(t), \qquad i = 1, \ldots, n,$$

and the rth component of the output equation is

$$y_r(t) = \sum_{j=1}^{n} c_{rj}\, x_j(t) \;+\; \sum_{\ell=1}^{m} d_{r\ell}\, u_\ell(t), \qquad r = 1, \ldots, p.$$

Remark 4.4

We present the state-space equations in the general MIMO form because the notation is standard in modern control. However, in this module we will primarily focus on the single-input single-output (SISO) case, where $m = 1$ and $p = 1$.

For SISO, $u(t) \in \mathbb{R}$ and $y(t) \in \mathbb{R}$, so $B \in \mathbb{R}^{n\times 1}$, $C \in \mathbb{R}^{1\times n}$, and $D \in \mathbb{R}$, and (4.7) becomes

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad y(t) = Cx(t) + Du(t).$$

Figure 4.3 shows the signal flow diagram for a state-space system.

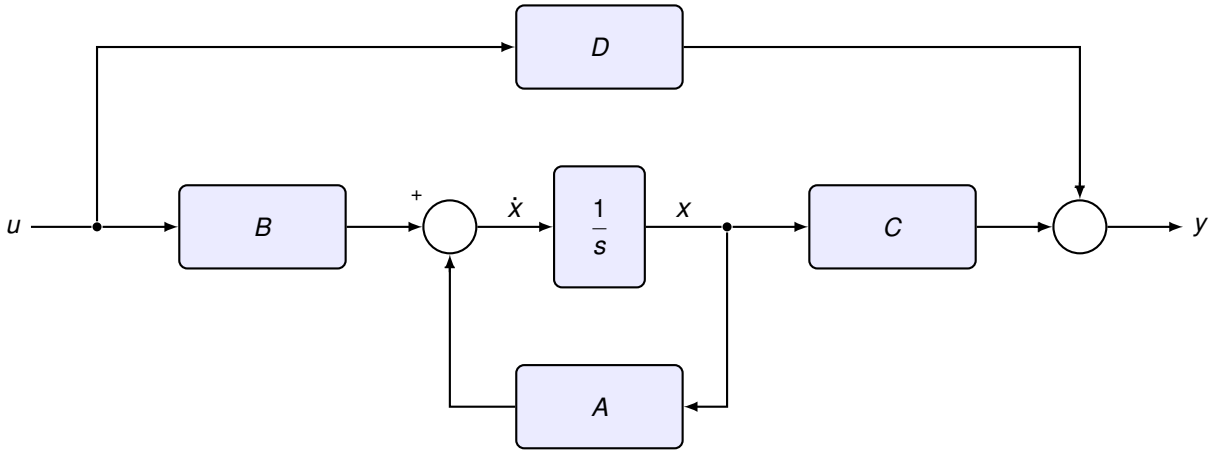


Figure 4.3: Block diagram of a continuous-time state-space realisation. Thick lines denote vector (bus) signals.

4.5 Deriving State-Space Models from Physical Systems: Complex Examples

Example 4.1: Two-mass spring–damper system: state-space derivation

System and modelling assumptions. Consider the two-degree-of-freedom translational system in Fig. 4.4. Let $x_1(t)$ and $x_2(t)$ denote the displacements of masses m_1 and m_2 from their equilibrium positions (positive to the right). The input is the external force $u(t) = F(t)$ applied to m_1. Springs are linear with stiffness k_1, k_2, k_3, and the damper between the masses is viscous with coefficient b.

Step 1: Write Newton's second law for each mass. Forces on m_1 (all terms with negative sign are forces opposing the positive direction of motion trying to slow down the mass):

- Spring to the left wall: $-k_1 x_1$
- Coupling spring: $-k_2(x_1 - x_2)$
- Coupling damper: $-b(\dot{x}_1 - \dot{x}_2)$
- External input: $+u$

Hence,

$$m_1\ddot{x}_1 = -k_1x_1 - k_2(x_1 - x_2) - b(\dot{x}_1 - \dot{x}_2) + u. \tag{4.8}$$

Forces on m_2:

- Spring to the right wall: $-k_3x_2$
- Coupling spring: $-k_2(x_2 - x_1)$
- Coupling damper: $-b(\dot{x}_2 - \dot{x}_1)$

Hence,

$$m_2\ddot{x}_2 = -k_3x_2 - k_2(x_2 - x_1) - b(\dot{x}_2 - \dot{x}_1). \tag{4.9}$$

Step 2: Choose states suggested by the derivatives. A standard choice is displacement and velocity for each mass:

$$z_1 = x_1, \quad z_2 = \dot{x}_1, \quad z_3 = x_2, \quad z_4 = \dot{x}_2, \qquad z = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 \end{bmatrix}^\top.$$

Step 3: Rewrite the dynamics as first-order equations. By definition,

$$\dot{z}_1 = z_2, \qquad \dot{z}_3 = z_4.$$

From (4.8)–(4.9),

$$\dot{z}_2 = \ddot{x}_1 = -\frac{k_1 + k_2}{m_1}z_1 - \frac{b}{m_1}z_2 + \frac{k_2}{m_1}z_3 + \frac{b}{m_1}z_4 + \frac{1}{m_1}u, \tag{4.10}$$

$$\dot{z}_4 = \ddot{x}_2 = \frac{k_2}{m_2}z_1 + \frac{b}{m_2}z_2 - \frac{k_2 + k_3}{m_2}z_3 - \frac{b}{m_2}z_4. \tag{4.11}$$

Step 4: Assemble the state-space model. Collecting (4.10)–(4.11) with $\dot{z}_1 = z_2$ and $\dot{z}_3 = z_4$ gives

$$\dot{z} = Az + Bu,$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{k_1+k_2}{m_1} & -\frac{b}{m_1} & \frac{k_2}{m_1} & \frac{b}{m_1} \\ 0 & 0 & 0 & 1 \\ \frac{k_2}{m_2} & \frac{b}{m_2} & -\frac{k_2+k_3}{m_2} & -\frac{b}{m_2} \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ \frac{1}{m_1} \\ 0 \\ 0 \end{bmatrix}.$$

Step 5: Define the output equation. The measured output depends on the application. Two common choices are:

- displacement of the first mass: $y = x_1 = z_1$, giving

$$y = Cz + Du, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}, \quad D = 0;$$

- both displacements (a MIMO output): $y = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\top = \begin{bmatrix} z_1 & z_3 \end{bmatrix}^\top$, giving

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Remark 4.5

For mechanical systems, the state variables are typically the quantities whose derivatives appear in the equations of motion. Practically, once you have the dynamic equations, a reliable default is: *choose each displacement as a state, and each corresponding velocity as its derivative state*, then rewrite the model as a first-order system.

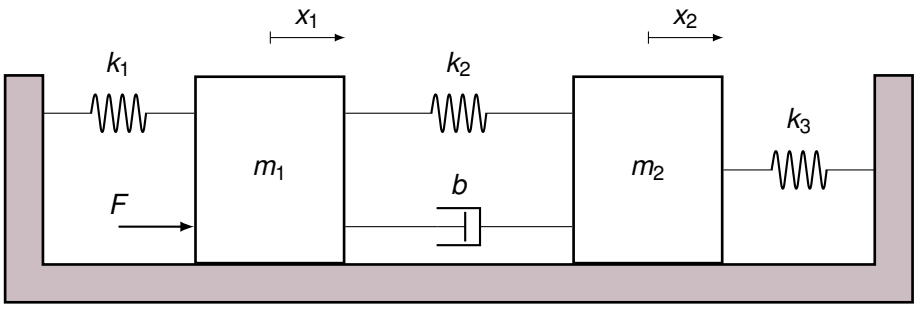


Figure 4.4: Mechanical system consisting of two masses m_1 and m_2, connected by springs and a viscous damper, subject to an external force F.

Example 4.2: Inverted pendulum on a cart (nonlinear, coupled dynamics)

System description. Consider a cart of mass M moving horizontally with position $x(t)$, carrying a rigid pendulum of mass m and length ℓ (distance from pivot to centre of mass). Let $\theta(t)$ be the pendulum angle measured from the upright vertical (so $\theta = 0$ is the unstable upright equilibrium). The input is a horizontal force $u(t)$ applied to the cart. Viscous friction on the cart is modelled by a coefficient $d \geq 0$.

Step 1: Equations of motion (coupled second-order ODEs). A standard nonlinear model is

$$(M + m)\ddot{x} + d\,\dot{x} + m\ell\ddot{\theta}\cos\theta - m\ell\dot{\theta}^2\sin\theta = u, \tag{4.12}$$

$$m\ell^2\ddot{\theta} + m\ell\ddot{x}\cos\theta = mg\ell\sin\theta. \tag{4.13}$$

Step 2: Choose states suggested by the derivatives. Select cart position/velocity and pendulum angle/angular velocity:

$$z_1 = x, \quad z_2 = \dot{x}, \quad z_3 = \theta, \quad z_4 = \dot{\theta}, \qquad z = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 \end{bmatrix}^\top.$$

Then $\dot{z}_1 = z_2$ and $\dot{z}_3 = z_4$. It remains to express $\dot{z}_2 = \ddot{x}$ and $\dot{z}_4 = \ddot{\theta}$.

Step 3: Solve (4.12)–(4.13) for $\ddot{x}$ and $\ddot{\theta}$. Rewrite the dynamics as a linear system in the unknown accelerations:

$$\underbrace{\begin{bmatrix} M + m & m\ell\cos z_3 \\ m\ell\cos z_3 & m\ell^2 \end{bmatrix}}_{=:\,\mathcal{M}(z_3)} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} u - dz_2 + m\ell z_4^2\sin z_3 \\ mg\ell\sin z_3 \end{bmatrix}.$$

Define the determinant (non-zero for typical parameter values and away from singular configurations)

$$\Delta(z_3) = \det\mathcal{M}(z_3) = m\ell^2\left(M + m - m\cos^2 z_3\right).$$

Using the 2×2 inverse formula,

$$\mathcal{M}(z_3)^{-1} = \frac{1}{\Delta(z_3)} \begin{bmatrix} m\ell^2 & -m\ell\cos z_3 \\ -m\ell\cos z_3 & M+m \end{bmatrix}.$$

Hence,

$$\ddot{x} = \frac{1}{\Delta(z_3)} \Big(m\ell^2 \big[u - dz_2 + m\ell z_4^2 \sin z_3 \big] - m\ell\cos z_3 \cdot mg\ell \sin z_3 \Big), \tag{4.14}$$

$$\ddot{\theta} = \frac{1}{\Delta(z_3)} \Big(-m\ell\cos z_3 \big[u - dz_2 + m\ell z_4^2 \sin z_3 \big] + (M+m)\, mg\ell \sin z_3 \Big). \tag{4.15}$$

Step 4: State-space form (nonlinear). The nonlinear state-space model $\dot{z} = f(z, u)$ is

$$\begin{aligned} \dot{z}_1 &= z_2, \\ \dot{z}_2 &= \ddot{x} \text{ given by (4.14)}, \\ \dot{z}_3 &= z_4, \\ \dot{z}_4 &= \ddot{\theta} \text{ given by (4.15)}. \end{aligned}$$

Step 5: Output equation. Depending on sensing, common choices include:

$$y = x \quad (\Rightarrow\ y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} z), \qquad \text{or} \qquad y = \begin{bmatrix} x \\ \theta \end{bmatrix} \quad (\Rightarrow\ y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} z).$$

Step 6: Small-angle approximation and linear state-space model

The equations of motion obtained above are nonlinear because they contain $\sin\theta(t)$ and $\cos\theta(t)$. To obtain a linear model, we assume the pendulum remains close to the upright position so that $|\theta(t)| \ll 1$ (in radians). Using the small-angle approximations

$$\sin\theta(t) \approx \theta(t), \qquad \cos\theta(t) \approx 1,$$

(and neglecting higher-order terms), the nonlinear dynamics reduce to a linear set of differential equations.

With the state choice

$$x_1(t) = x(t), \qquad x_2(t) = \dot{x}(t), \qquad x_3(t) = \theta(t), \qquad x_4(t) = \dot{\theta}(t),$$

we have

$$\dot{x}_1(t) = x_2(t), \qquad \dot{x}_3(t) = x_4(t),$$

and the remaining two state derivatives take the linear form

$$\dot{x}_2(t) = -\frac{d}{M} x_2(t) - \frac{mg}{M} x_3(t) + \frac{1}{M} u(t),$$

$$\dot{x}_4(t) = \frac{d}{M\ell} x_2(t) + \frac{(M+m)g}{M\ell} x_3(t) - \frac{1}{M\ell} u(t).$$

Hence the linearised state-space model can be written compactly as

$$\dot{x}(t) = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{d}{M} & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{d}{M\ell} & \frac{(M+m)g}{M\ell} & 0 \end{bmatrix}}_{A} x(t) + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ -\frac{1}{M\ell} \end{bmatrix}}_{B} u(t), \qquad y(t) = Cx(t) + Du(t),$$

with $x = [x_1\ x_2\ x_3\ x_4]^\top$.

Remark 4.6

Even for strongly coupled nonlinear dynamics, the workflow is the same: (i) write the coupled equations of motion, (ii) choose states as the variables whose derivatives appear, and (iii) solve the equations for the highest derivatives to obtain a first-order state model $\dot{z} = f(z, u)$.

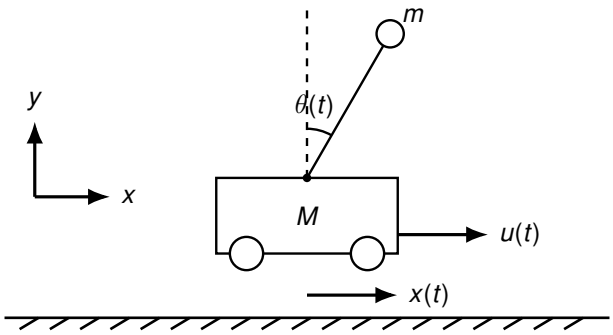
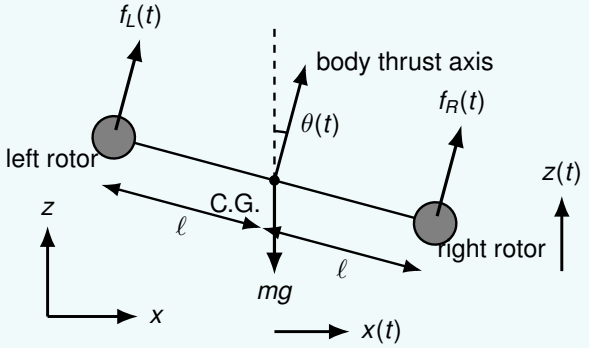


Figure 4.5: Inverted pendulum on a cart: input $u(t)$, cart position $x(t)$, and pendulum angle $\theta(t)$ (measured from the upright vertical).

Example 4.3: Planar duorotor model and hover linearisation

System description. Consider a planar duorotor vehicle moving in the vertical plane. Let $x(t)$ be the horizontal position, $z(t)$ the vertical position, and $\theta(t)$ the pitch angle measured clockwise from the vertical. The left and right rotors generate thrusts $f_L(t)$ and $f_R(t)$ along the body vertical axis. The vehicle has mass m, pitch moment of inertia J, and half-arm length ℓ.



Define the total thrust magnitude and pitch torque as

$$F(t) = f_L(t) + f_R(t), \qquad \tau(t) = \ell\big(f_L(t) - f_R(t)\big).$$

Here $F(t)$ is the scalar sum of the two rotor thrust magnitudes, acting along the body thrust axis. With the sign convention above, positive θ tilts the thrust vector towards positive x, and positive τ increases θ. Thus $f_L > f_R$ gives a positive pitch torque. If the opposite angle convention is used, the signs of the horizontal and torque equations must be changed consistently.

Step 1: Write the nonlinear equations of motion. Resolving the thrust vector into inertial x and z components gives

$$m\ddot{x} = F\sin\theta, \tag{4.16}$$
$$m\ddot{z} = F\cos\theta - mg,$$
$$J\ddot{\theta} = \tau. \tag{4.17}$$

These equations are nonlinear because the translational accelerations depend on $\sin\theta$ and $\cos\theta$.

Step 2: Choose states. Select each configuration variable and its velocity as a state:

$$x_1 = x, \quad x_2 = \dot{x}, \quad x_3 = z, \quad x_4 = \dot{z}, \quad x_5 = \theta, \quad x_6 = \dot{\theta}, \qquad \mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{bmatrix}^\top.$$

Step 3: Rewrite as a first-order nonlinear state model. Using (4.16)–(4.17),

$$\begin{aligned} \dot{x}_1 &= x_2, & \dot{x}_2 &= \frac{F}{m}\sin x_5, \\ \dot{x}_3 &= x_4, & \dot{x}_4 &= \frac{F}{m}\cos x_5 - g, \\ \dot{x}_5 &= x_6, & \dot{x}_6 &= \frac{\tau}{J}. \end{aligned} \tag{4.18}$$

This is a nonlinear state-space model $\dot{\mathbf{x}} = f(\mathbf{x}, F, \tau)$.

Step 4: Linearise about hover. At hover, the vehicle is level and has zero acceleration, so

$$\theta_0 = 0, \qquad \dot{\mathbf{x}}_0 = 0, \qquad F_0 = mg, \qquad \tau_0 = 0.$$

Introduce small input deviations from hover:

$$u_1(t) = F(t) - mg, \qquad u_2(t) = \tau(t).$$

These are *virtual inputs*: u_1 represents the change in total thrust from the hover value, while u_2 represents the applied pitch torque. For small pitch angles, use

$$\sin\theta \approx \theta, \qquad \cos\theta \approx 1.$$

Substituting $F = mg + u_1$ into (4.18) and neglecting products of small quantities such as $u_1\theta$ gives

$$\ddot{x} \approx g\theta, \qquad \ddot{z} \approx \frac{1}{m}u_1, \qquad \ddot{\theta} = \frac{1}{J}u_2.$$

Therefore the hover-linearised state-space model is

$$\dot{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{A} \mathbf{x} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{m} & 0 \\ 0 & 0 \\ 0 & \frac{1}{J} \end{bmatrix}}_{B} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}.$$

If the measured outputs are horizontal position, altitude, and pitch angle, then

$$y = \begin{bmatrix} x \\ z \\ \theta \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{C} \mathbf{x}, \qquad D = 0.$$

Remark 4.7

The hover model separates altitude and attitude inputs at first order: u_1 changes vertical acceleration, while u_2 changes angular acceleration. Horizontal acceleration appears indirectly through pitch, since $\ddot{x} \approx g\theta$.

Example 4.4: DC–DC buck converter (averaged model in CCM)

A highly practical state-space example is the buck (step-down) converter, used in laptops, EV auxiliary power, robotics, and embedded systems to regulate a DC output voltage from a higher DC supply.

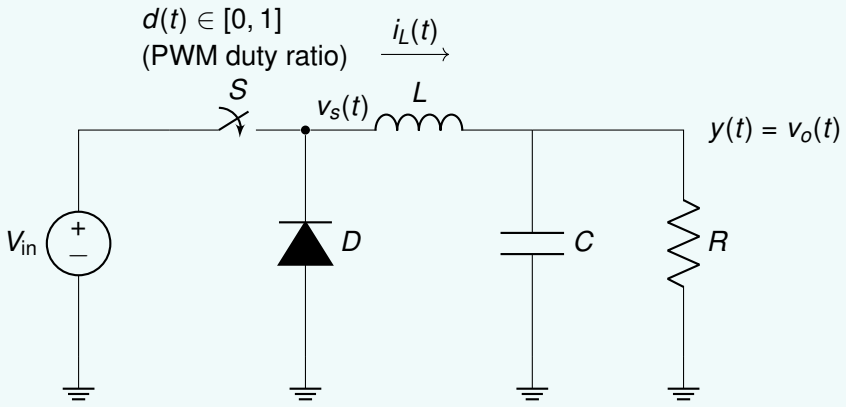


Figure 4.6: Buck converter. In continuous conduction mode (CCM), an averaged model yields a linear state-space form in terms of $d(t)$. Assume voltage drop across the diode $V_D = 0$ when it conducts.

Step 1: Choose states (energy-storing variables). A natural choice is

$$x_1(t) = i_L(t), \qquad x_2(t) = v_o(t) = v_C(t).$$

Step 2: Write the switching (piecewise) dynamics. Let $v_s(t)$ be the switch-node voltage applied to the inductor. For an ideal buck:

$$v_s(t) = \begin{cases} V_{in}, & \text{switch ON} \\ 0, & \text{switch OFF (diode conducts)} \end{cases}$$

The inductor and capacitor relations give

$$L\dot{i}_L(t) = v_s(t) - v_o(t), \qquad C\dot{v}_o(t) = i_L(t) - \frac{v_o(t)}{R}.$$

Step 3: Average the model over one PWM period (CCM). With duty ratio $d(t)$, the averaged switch-node voltage is

$$\overline{v_s}(t) = d(t)\, V_{\text{in}}.$$

Hence the averaged (continuous-time) state equations become

$$\dot{x}_1(t) = \frac{1}{L}\Big(\overline{v_s}(t) - x_2(t)\Big) = -\frac{1}{L}x_2(t) + \frac{V_{\text{in}}}{L}d(t),$$

$$\dot{x}_2(t) = \frac{1}{C}\Big(x_1(t) - \frac{x_2(t)}{R}\Big) = \frac{1}{C}x_1(t) - \frac{1}{RC}x_2(t).$$

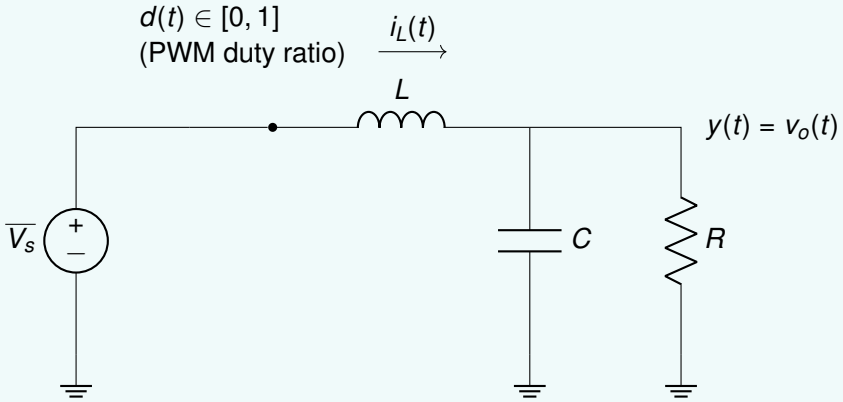


Figure 4.7: Averaged buck converter model used for state-space derivation. The switching network is replaced by the averaged source $\bar{v}_s(t) = d(t)V_{\text{in}}$.

Step 4: State-space form (MIMO-ready notation; SISO used here). Define input $u(t) = d(t)$ and output $y(t) = v_o(t) = x_2(t)$. Then

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad y(t) = Cx(t) + Du(t),$$

with

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix}, \qquad B = \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix} = \begin{bmatrix} \frac{V_{\text{in}}}{L} \\ 0 \end{bmatrix}.$$

For the output,

$$C = \begin{bmatrix} c_{11} & c_{12} \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix}, \qquad D = \begin{bmatrix} d_{11} \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}.$$

Remark 4.8

Power-electronics circuits are naturally modelled in state space because they are *switching systems*. In Continuous Conduction Mode (CCM), an averaged model gives a compact continuous-time state-space description with states (i_L, v_o) and control input $d(t)$ (the duty ratio).

Example 4.1: Multi-loop electrical network with two inputs

Derive the state-space model for the electrical network shown in Figure 4.8, where the inputs are the voltage source $v(t)$ and the current source $i_s(t)$, and the output is the capacitor voltage v_2.

Solution:

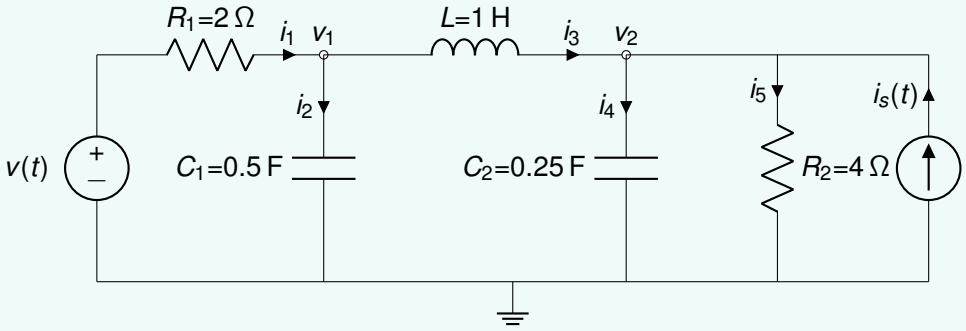


Figure 4.8: Two-loop electrical network with voltage source $v(t)$ and current source $i_s(t)$.

Step 1 — Identify energy-storing elements and state variables. There are three independent energy-storing elements: C_1, C_2, and L. Therefore three state variables are needed:

$$x_1 = v_1, \qquad x_2 = v_2, \qquad x_3 = i_3.$$

Step 2 — KCL at node v_1. The current entering node v_1 through R_1 equals the sum of the current into C_1 and the current into L:

$$\frac{v - v_1}{R_1} = C_1 \frac{dv_1}{dt} + i_3.$$

Solving for $\dot{x}_1$:

$$\dot{x}_1 = -\frac{1}{R_1 C_1} x_1 - \frac{1}{C_1} x_3 + \frac{1}{R_1 C_1} v.$$

Step 3 — KCL at node v_2. The current from L plus the current source equals the current into C_2 plus the current through R_2:

$$i_3 + i_s = C_2 \frac{dv_2}{dt} + \frac{v_2}{R_2}.$$

Solving for $\dot{x}_2$:

$$\dot{x}_2 = -\frac{1}{R_2 C_2} x_2 + \frac{1}{C_2} x_3 + \frac{1}{C_2} i_s.$$

Step 4 — Voltage across L.

$$L \frac{di_3}{dt} = v_1 - v_2 \qquad \Longrightarrow \qquad \dot{x}_3 = \frac{1}{L} x_1 - \frac{1}{L} x_2.$$

Step 5 — Assemble the parametric state-space model. Combining the three state equations with the output $y = v_2 = x_2$:

$$\dot{x}(t) = \begin{bmatrix} -\dfrac{1}{R_1 C_1} & 0 & -\dfrac{1}{C_1} \\ 0 & -\dfrac{1}{R_2 C_2} & \dfrac{1}{C_2} \\ \dfrac{1}{L} & -\dfrac{1}{L} & 0 \end{bmatrix} x(t) + \begin{bmatrix} \dfrac{1}{R_1 C_1} & 0 \\ 0 & \dfrac{1}{C_2} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v(t) \\ i_s(t) \end{bmatrix}, \qquad y(t) = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} x(t).$$

Remark 4.9

The two zeros in the third row of B reflect the fact that neither input connects directly to the inductor — the inductor current changes only through the voltage difference $v_1 - v_2$ across it.

Step 6 — Substitute the numerical values. With $R_1 = 2\,\Omega$, $C_1 = 0.5\,\text{F}$, $L = 1\,\text{H}$, $C_2 = 0.25\,\text{F}$, $R_2 = 4\,\Omega$:

$$\dot{x}(t) = \begin{bmatrix} -1 & 0 & -2 \\ 0 & -1 & 4 \\ 1 & -1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 1 & 0 \\ 0 & 4 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v(t) \\ i_s(t) \end{bmatrix}, \qquad y(t) = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} x(t).$$

4.6 Solution of State-Space Equations

Consider the continuous-time LTI state-space model

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad x(0) = x_0, \tag{4.19}$$

$$y(t) = Cx(t) + Du(t). \tag{4.20}$$

4.6.1 Time-domain solution and the state transition matrix

The solution of (4.19) is most naturally expressed using the State transition matrix

$$\Phi(t) \triangleq e^{At}.$$

With this notation, the state trajectory is

$$x(t) = e^{At} x_0 + \int_0^t e^{A(t-\tau)} B\, u(\tau)\, d\tau. \tag{4.21}$$

Substituting (4.21) into (4.20) gives the output

$$y(t) = Ce^{At} x_0 + \int_0^t Ce^{A(t-\tau)} B\, u(\tau)\, d\tau + Du(t).$$

The integral term is called the convolution integral and it is generally very difficult to calculate even for simple input signals $u(t)$.

Definition of the matrix exponential

The Matrix exponential is defined by the convergent power series

$$e^{At} \triangleq \sum_{k=0}^{\infty} \frac{(At)^k}{k!} = I + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \cdots,$$

where I is the $n \times n$ identity matrix. In practice, e^{At} is computed numerically (for example, using `expm` in MATLAB).

4.6.2 Laplace-domain solution and obtaining transfer functions from state-space

Taking the Laplace transform of (4.19) and using $\mathcal{L}\{\dot{x}(t)\} = sX(s) - x(0)$ yields

$$sX(s) - x_0 = AX(s) + BU(s).$$

Rearranging gives

$$(sI - A)X(s) = x_0 + BU(s), \qquad X(s) = (sI - A)^{-1}x_0 + (sI - A)^{-1}B\,U(s). \tag{4.22}$$

Taking the Laplace transform of (4.20) gives

$$Y(s) = CX(s) + DU(s). \tag{4.23}$$

Substituting (4.22) into (4.23),

$$Y(s) = C(sI - A)^{-1}x_0 + \Big(C(sI - A)^{-1}B + D\Big)U(s).$$

Transfer function from state-space

For zero initial conditions ($x_0 = 0$), the input–output relationship becomes

$$Y(s) = G(s)U(s),$$

where the transfer function is

$$G(s) = C(sI - A)^{-1}B + D.$$

This formula is particularly useful for converting state-space models to transfer functions for frequency-domain analysis and for checking consistency with models obtained via differential equations.

Connection to the time-domain solution

The Laplace-domain expressions are consistent with the time-domain solution:

$$\mathcal{L}\{e^{At}\} = (sI - A)^{-1},$$

so $(sI - A)^{-1}$ may be viewed as the Laplace transform of the state transition matrix. In many problems, solving (4.22) and then applying an inverse Laplace transform is the quickest way to obtain $x(t)$ and $y(t)$, especially when $u(t)$ is a standard input (step, impulse, sinusoid, or an exponential) and partial fraction expansion can be used.

Example 4.5: State-space solution via Laplace transform and MATLAB verification

Question. Consider the state-space model

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad y(t) = Cx(t) + Du(t),$$

with

$$A = \begin{bmatrix} -2 & 1 \\ 0 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad D = 0,$$

initial condition

$$x(0) = x_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

and input

$$u(t) = 1 \quad \text{(unit step)}.$$

(a) Solve for $X(s)$ using the Laplace transform and obtain closed-form expressions for $x_1(t)$ and $x_2(t)$.

(b) Compute $Y(s)$ and obtain $y(t)$ in the time domain.

(c) Verify your result using MATLAB by simulating the state response and output for $0 \leq t \leq 5$.

Solution:

(a) Laplace-domain state solution. Taking the Laplace transform of $\dot{x}(t) = Ax(t) + Bu(t)$ gives

$$sX(s) - x_0 = AX(s) + BU(s) \quad \Rightarrow \quad (sI - A)X(s) = x_0 + BU(s).$$

For the unit step input, $U(s) = \dfrac{1}{s}$. Hence

$$X(s) = (sI - A)^{-1}x_0 + (sI - A)^{-1}B\frac{1}{s}.$$

Compute

$$sI - A = \begin{bmatrix} s+2 & -1 \\ 0 & s+3 \end{bmatrix}, \qquad (sI - A)^{-1} = \begin{bmatrix} \dfrac{1}{s+2} & \dfrac{1}{(s+2)(s+3)} \\ 0 & \dfrac{1}{s+3} \end{bmatrix}.$$

Now evaluate each term.

Term 1: $(sI - A)^{-1}x_0$:

$$(sI - A)^{-1}x_0 = \begin{bmatrix} \dfrac{1}{s+2} & \dfrac{1}{(s+2)(s+3)} \\ 0 & \dfrac{1}{s+3} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \dfrac{1}{(s+2)(s+3)} \\ \dfrac{1}{s+3} \end{bmatrix}.$$

Term 2: $(sI - A)^{-1}B\frac{1}{s}$: first compute $(sI - A)^{-1}B$:

$$(sI - A)^{-1}B = \begin{bmatrix} \dfrac{1}{s+2} & \dfrac{1}{(s+2)(s+3)} \\ 0 & \dfrac{1}{s+3} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} \dfrac{1}{s+2} + \dfrac{2}{(s+2)(s+3)} \\ \dfrac{2}{s+3} \end{bmatrix}.$$

Therefore

$$(sI - A)^{-1}B\frac{1}{s} = \begin{bmatrix} \dfrac{1}{s(s+2)} + \dfrac{2}{s(s+2)(s+3)} \\ \dfrac{2}{s(s+3)} \end{bmatrix}.$$

Combine the terms to obtain $X(s) = [X_1(s)\ X_2(s)]^\top$:

$$X_2(s) = \frac{1}{s+3} + \frac{2}{s(s+3)} = \frac{s}{s(s+3)} + \frac{2}{s(s+3)} = \frac{s+2}{s(s+3)}.$$

Partial fractions:

$$\frac{s+2}{s(s+3)} = \frac{A}{s} + \frac{B}{s+3} \Rightarrow s+2 = A(s+3) + Bs \Rightarrow \begin{cases} A+B=1 \\ 3A=2 \end{cases} \Rightarrow A = \frac{2}{3},\ B = \frac{1}{3}.$$

Thus

$$x_2(t) = \frac{2}{3} + \frac{1}{3}e^{-3t}.$$

For $X_1(s)$:

$$X_1(s) = \frac{1}{(s+2)(s+3)} + \frac{1}{s(s+2)} + \frac{2}{s(s+2)(s+3)}.$$

Write the third term using partial fractions:

$$\frac{2}{s(s+2)(s+3)} = \frac{A}{s} + \frac{B}{s+2} + \frac{C}{s+3}.$$

Solving

$$2 = A(s+2)(s+3) + Bs(s+3) + Cs(s+2)$$

by substitution gives:

$$\begin{aligned} s = 0: &\quad 2 = 6A \Rightarrow A = \frac{1}{3}, \\ s = -2: &\quad 2 = -2B \Rightarrow B = -1, \\ s = -3: &\quad 2 = 3C \Rightarrow C = \frac{2}{3}. \end{aligned}$$

Hence

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

Also

$$\frac{1}{(s+2)(s+3)} = \frac{1}{s+2} - \frac{1}{s+3}, \qquad \frac{1}{s(s+2)} = \frac{1}{2}\frac{1}{s} - \frac{1}{2}\frac{1}{s+2}.$$

Therefore

$$X_1(s) = \left(\frac{1}{2} + \frac{1}{3}\right)\frac{1}{s} + \left(1 - \frac{1}{2} - 1\right)\frac{1}{s+2} + \left(-1 + \frac{2}{3}\right)\frac{1}{s+3}.$$

So

$$X_1(s) = \frac{5}{6}\frac{1}{s} - \frac{1}{2}\frac{1}{s+2} - \frac{1}{3}\frac{1}{s+3}.$$

Taking the inverse Laplace transform gives

$$x_1(t) = \frac{5}{6} - \frac{1}{2}e^{-2t} - \frac{1}{3}e^{-3t}. \tag{4.24}$$

(b) Output. Since $y(t) = Cx(t) = x_1(t)$ (because $C = [1\ 0]$ and $D = 0$),

$$y(t) = \frac{5}{6} - \frac{1}{2}e^{-2t} - \frac{1}{3}e^{-3t}. \tag{4.25}$$

(c) MATLAB verification. Use `lsim` with a unit-step input and the given initial condition, as shown in Listing 4.1.

```
clear; clc; close all;

A = [-2  1; 0 -3];
B = [1; 2];
C = [1 0];
D = 0;

x0 = [0; 1];

t = linspace(0,5,1001);
u = ones(size(t));  % unit step

sys = ss(A,B,C,D);

% Simulate
[y,tout,x] = lsim(sys,u,t,x0);

% Closed-form expressions
x1_ana = 5/6 - 0.5*exp(-2*t) - (1/3)*exp(-3*t);
x2_ana = 2/3 + (1/3)*exp(-3*t);
y_ana  = x1_ana;

% Plot comparisons
figure;
plot(tout, x(:,1), 'LineWidth', 1.5); hold on;
plot(t,   x1_ana, '--', 'LineWidth', 1.5);
grid on;
xlabel('Time (s)'); ylabel('x_1(t)');
legend('MATLAB lsim','Analytical','Location','best');
title('State x_1(t): MATLAB vs analytical');

figure;
plot(tout, x(:,2), 'LineWidth', 1.5); hold on;
plot(t,   x2_ana, '--', 'LineWidth', 1.5);
grid on;
xlabel('Time (s)'); ylabel('x_2(t)');
legend('MATLAB lsim','Analytical','Location','best');
title('State x_2(t): MATLAB vs analytical');

figure;
plot(tout, y, 'LineWidth', 1.5); hold on;
plot(t,   y_ana, '--', 'LineWidth', 1.5);
grid on;
xlabel('Time (s)'); ylabel('y(t)');
legend('MATLAB lsim','Analytical','Location','best');
title('Output y(t): MATLAB vs analytical');

```

Listing 4.1: MATLAB verification of the Laplace-domain solution.

The MATLAB simulation traces should overlap the analytical curves (4.24)–(4.25), confirming the Laplace-domain solution.

4.6.3 Eigenvalues of A and system poles

There is a direct connection between the state matrix A and the poles of the transfer function $G(s)$. The poles are the values of s where $(sI - A)$ is singular, i.e. the roots of

$$\det(sI - A) = 0.$$

These are exactly the **eigenvalues** of A. In other words:

Result 4.1: Eigenvalues = system poles

The poles of the transfer function $G(s) = C(sI - A)^{-1}B + D$ are a subset of the eigenvalues of A. If there are no pole-zero cancellations, they are identical.

Example 4.6: Poles from eigenvalues and from the transfer function

A mass–spring–damper with $m = 1\,\text{kg}$, $b = 3\,\text{Ns/m}$, $k = 2\,\text{N/m}$ has the state-space model

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad D = 0.$$

Find the system poles (a) from the eigenvalues of A, and (b) from the transfer function.

Solution:

(a) Eigenvalue approach. Solve $\det(sI - A) = 0$:

$$\det \begin{bmatrix} s & -1 \\ 2 & s+3 \end{bmatrix} = s(s+3) + 2 = s^2 + 3s + 2 = (s+1)(s+2) = 0.$$

The eigenvalues are $\lambda_1 = -1$ and $\lambda_2 = -2$.

(b) Transfer function approach. Compute $G(s) = C(sI - A)^{-1}B$:

$$(sI - A)^{-1} = \frac{1}{s^2 + 3s + 2} \begin{bmatrix} s+3 & 1 \\ -2 & s \end{bmatrix},$$

so

$$G(s) = \begin{bmatrix} 1 & 0 \end{bmatrix} \frac{1}{s^2 + 3s + 2} \begin{bmatrix} s+3 & 1 \\ -2 & s \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{s^2 + 3s + 2} = \frac{1}{(s+1)(s+2)}.$$

The transfer function poles are $s = -1$ and $s = -2$, matching the eigenvalues exactly.

Both poles are in the left-half plane, so the system is stable. The pole at -1 is the slower mode (time constant $\tau = 1\,\text{s}$) and dominates the transient response.

Stability, natural frequency, and damping can all be read directly from the eigenvalues of A, without first converting to a transfer function. Listing 4.2 verifies this in MATLAB.

```
A = [0 1; -2 -3]; B = [0; 1]; C = [1 0]; D = 0;
sys = ss(A, B, C, D);

eig(A)           % eigenvalues of A: -1, -2
pole(tf(sys))    % TF poles: -1, -2 (should match)
```

Listing 4.2: Comparing eigenvalues of A with transfer-function poles.

Remark 4.10

If a mode is uncontrollable or unobservable (Chapter 10), the corresponding eigenvalue will not appear as a transfer function pole — it is "hidden" by a pole-zero cancellation. The eigenvalues of A always reveal the full picture.

4.7 Implementation in MATLAB

This section shows how to work with state-space models in MATLAB in a way that matches the notation used throughout the chapter. We use the Control System Toolbox functions `ss`, `tf`, `ss2tf`, `step`, `impulse`, and `lsim`. Unless stated otherwise, we assume continuous-time LTI systems.

4.7.1 Creating State-Space Models

A continuous-time state-space model has the form

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad y(t) = Cx(t) + Du(t),$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, and $D \in \mathbb{R}^{p\times m}$.

Defining an `ss` model from matrices

In MATLAB, the most direct way to create a state-space model is shown in Listing 4.3.

```
A = [-2  1; 0 -3];
B = [1; 2];
C = [1 0];
D = 0;

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

Listing 4.3: Creating a continuous-time state-space model in MATLAB.

You can check the dimensions and inspect the model using Listing 4.4.

```
size(A), size(B), size(C), size(D)
sys
```

Listing 4.4: Inspecting a state-space model.

Naming inputs, outputs, and states

For clarity in plots and model inspection, assign names as in Listing 4.5.

```
sys.InputName  = "u";
sys.OutputName = "y";
sys.StateName  = ["x1"; "x2"];
```

Listing 4.5: Naming states, inputs, and outputs.

Simulating time responses

Step and impulse responses. For an LTI system, the standard responses are obtained as in Listing 4.6.

```
figure; step(sys); grid on;
figure; impulse(sys); grid on;
```

Listing 4.6: Step and impulse responses.

Response to an arbitrary input with initial conditions. To simulate with a given input signal $u(t)$ and an initial condition $x(0) = x_0$, use `lsim` as in Listing 4.7.

```
t  = linspace(0,5,1001);
u  = ones(size(t));   % unit step input
x0 = [0; 1];

[y,tout,x] = lsim(sys,u,t,x0);

figure; plot(tout,y,'LineWidth',1.5); grid on;
xlabel('Time (s)'); ylabel('y(t)'); title('Output response');
```

Listing 4.7: Simulation with lsim and nonzero initial condition.

Discrete-time state-space models

A discrete-time state-space model has the form

$$x_{k+1} = Ax_k + Bu_k, \qquad y_k = Cx_k + Du_k.$$

In MATLAB, specify the sample time T_s as in Listing 4.8.

```
Ts  = 0.05;               % sample time
sysd = ss(A,B,C,D,Ts);
```

Listing 4.8: Creating a discrete-time state-space model.

You may discretise a continuous-time model using `c2d` (Listing 4.9).

```
Ts   = 0.05;
sysd = c2d(sys,Ts,'zoh');   % zero-order hold
```

Listing 4.9: Discretising a continuous-time model.

4.7.2 Converting State-Space Models to Transfer Functions

Although state-space is the preferred representation for modern control design, it is often useful to obtain transfer functions for classical analysis (poles/zeros, frequency response, Bode/Nyquist plots, etc.).

Using `tf(sys)`

The simplest conversion is shown in Listing 4.10.

```
G = tf(sys)
```

Listing 4.10: State-space to transfer function using tf(sys).

If `sys` is MIMO, then `G` is a transfer-function matrix. Individual channels are indexed as `G(i,j)` (Listing 4.11).

```
% G is p-by-m (outputs-by-inputs)
G11 = G(1,1);
```

Listing 4.11: Selecting a SISO channel from a MIMO transfer function matrix.

Using `ss2tf`

The command `ss2tf` returns numerator and denominator polynomials (Listing 4.12).

```
[num,den] = ss2tf(A,B,C,D);   % SISO: num is 1-by-(n+1)
G2 = tf(num,den);
```

Listing 4.12: State-space to transfer function using ss2tf.

For MIMO systems, `ss2tf` can be used per input channel by selecting a particular input index (Listing 4.13).

```
inputIndex = 1;
[num,den] = ss2tf(A,B,C,D,inputIndex);
```

Listing 4.13: Using ss2tf for a chosen input channel in a MIMO model.

Poles, zeros, and DC gain

Once you have `sys` (or `G`), typical checks are shown in Listing 4.14.

```
pole(sys)
tzero(sys)
dcgain(sys)

% Frequency response plots
figure; bode(sys); grid on;
figure; nyquist(sys); grid on;
```

Listing 4.14: Basic analysis: poles, zeros, and DC gain.

4.7.3 Converting Transfer Functions to State-Space: Canonical Forms

The choice of state variables is not unique — different choices give different A, B, C, D matrices for the same input–output behaviour. Two standard choices are particularly useful:

Controllable canonical form (CCF)

Given a transfer function

$$G(s) = \frac{b_1 s^{n-1} + b_2 s^{n-2} + \cdots + b_n}{s^n + a_1 s^{n-1} + \cdots + a_n},$$

the controllable canonical form is

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} b_n & b_{n-1} & \cdots & b_1 \end{bmatrix}.$$

This form is always controllable (the controllability matrix has full rank).

Observable canonical form (OCF)

The observable canonical form is the transpose of the CCF: $A_o = A_c^T$, $B_o = C_c^T$, $C_o = B_c^T$. This form is always observable.

In MATLAB, `tf2ss` returns a form similar to the CCF (with coefficients in the first row rather than the last), as illustrated in Listing 4.15.

```
G = tf([1 3], [1 5 6]);        % G(s) = (s+3)/(s^2+5s+6)
[A,B,C,D] = tf2ss([1 3], [1 5 6]);
sys_ss = ss(A,B,C,D);
```

Listing 4.15: Converting a transfer function to state-space using tf2ss.

Remark 4.11

MATLAB's `tf2ss` places the denominator coefficients in the *first* row of A, not the last. This is a transposed convention compared to many textbooks. Use `canon(sys,'companion')` to get the standard companion form.

4.8 Simulink Representation of State-Space Systems

Simulink provides a natural environment for implementing and simulating state-space models, especially when block-diagram integration with nonlinearities, saturations, disturbances, measurement noise, or multi-rate components is required.

4.8.1 State-Space block

The most direct representation is the **State-Space** block:

- Library: `Simulink` → `Continuous` → `State-Space`
- Parameters: matrices `A`, `B`, `C`, `D` entered in the block dialog

- Input: $u(t)$, Output: $y(t)$, with internal state $x(t)$ managed by the solver

Figure 4.9 shows a minimal Simulink model using the State-Space block.

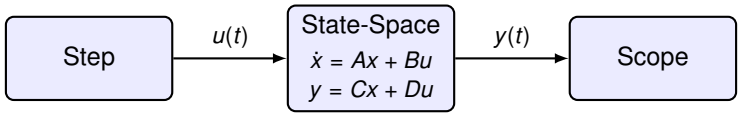


Figure 4.9: Minimal Simulink model: a Step input drives the State-Space block, and the output is displayed on a Scope.

A standard workflow is:

Step 1: Define the matrices A, B, C, D in the MATLAB workspace (script or Live Script).

Step 2: Insert a **State-Space** block and set its parameters to `A`, `B`, `C`, `D`.

Step 3: Provide an input signal (e.g., **Step**, **Signal Builder**, **From Workspace**).

Step 4: Add a **Scope** (or **To Workspace**) to view/log outputs.

Step 5: Configure solver settings: **Model Settings** $\rightarrow$ **Solver**.

4.8.2 Realising state-space with integrators and gains

Sometimes it is helpful pedagogically (or for custom architectures) to implement

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

using basic blocks:

- **Integrator** blocks to obtain x from $\dot{x}$,
- **Gain** blocks to implement Ax, Bu, Cx, and Du,
- a **Sum** block to form $\dot{x} = Ax + Bu$ (and another to form $y = Cx + Du$).

This construction makes the signal flow explicit: the integrator output is the state vector $x(t)$, which feeds back through A and forward through C, while the input $u(t)$ is injected through B and D.

4.8.3 Logging states and setting initial conditions

Two common tasks in Simulink are:

- **Initial conditions:** set the initial condition of the state vector either in the **State-Space** block (if available) or by specifying the initial condition inside the integrator blocks when using an integrator realisation.
- **State logging:** use **To Workspace** blocks (or enable signal logging) to record $x(t)$ and $y(t)$ for post-processing in MATLAB.

Key Takeaways

- The state-space model $\dot{x} = Ax + Bu$, $y = Cx + Du$ describes a system's internal dynamics, not just its input–output behaviour.
- State variables are typically energy-storing quantities (displacement, velocity, voltage, current, temperature).
- Higher-order ODEs are converted to state-space form by defining each derivative as a new state variable.
- The transfer function is obtained from state-space via $G(s) = C(sI - A)^{-1}B + D$.
- The eigenvalues of A are the system poles (assuming no pole-zero cancellations).
- Nonlinear systems can be linearised about an operating point to obtain a linear state-space model.
- MATLAB commands `ss`, `tf`, `ss2tf`, `tf2ss`, and `lsim` handle all standard conversions and simulations.

End-of-Chapter Exercises

1. A mass-spring-damper system is described by the ODE

$$m\ddot{y} + b\dot{y} + ky = f(t),$$

where $m = 2\,\text{kg}$, $b = 3\,\text{Ns/m}$, and $k = 5\,\text{N/m}$. Choose $x_1 = y$ and $x_2 = \dot{y}$ as state variables and write the system in standard state-space form $\dot{x} = Ax + Bu$, $y = Cx + Du$. Clearly state the matrices A, B, C, and D.

2. Consider a second-order ODE

$$\ddot{y} + 4\dot{y} + 3y = 2u(t).$$

   (a) Write this as a state-space model using the standard choice $x_1 = y$, $x_2 = \dot{y}$.

   (b) Compute the eigenvalues of the A matrix by hand.

   (c) What do these eigenvalues tell you about the system's natural behaviour?

3. A system has the transfer function

$$G(s) = \frac{5}{s^2 + 6s + 8}.$$

   (a) Convert this to a state-space model by hand using the controllable canonical form.

   (b) Verify your answer in MATLAB using `[A,B,C,D] = tf2ss([5],[1 6 8])`.

4. Given the state-space model

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad D = 0,$$

   (a) Find the transfer function $G(s) = C(sI - A)^{-1}B + D$ by hand.

   (b) Verify your result in MATLAB using `sys = ss(A,B,C,D)` followed by `tf(sys)`.

5. An RLC series circuit with input voltage $v_{in}(t)$ and output voltage $v_C(t)$ across the capacitor satisfies

$$L\ddot{v}_C + R\dot{v}_C + \frac{1}{C}v_C = \frac{1}{C}v_{in}.$$

With $L = 1\,\text{H}$, $R = 4\,\Omega$, $C = 0.5\,\text{F}$, choose appropriate state variables and write the state-space matrices A, B, C, D.

6. Write a short MATLAB script that:

   (a) defines the state-space model from Exercise 1 using `sys = ss(A,B,C,D);`

   (b) converts it to a transfer function using `tf(sys);`

   (c) computes the eigenvalues of A using `eig(A);`

   (d) plots the unit step response using `step(sys)`.

   Run your script and briefly describe the shape of the step response.

7. A system is described by two coupled first-order ODEs:

$$\dot{x}_1 = -x_1 + 2x_2 + u, \qquad \dot{x}_2 = 3x_1 - 4x_2,$$

with output $y = x_1$.

   (a) Write down the matrices A, B, C, D.

   (b) Find the eigenvalues of A. Is the system stable?

   (c) Use MATLAB to plot the step response and confirm your stability prediction.

8. A third-order system has the transfer function

$$G(s) = \frac{10}{s^3 + 5s^2 + 8s + 4}.$$

Use MATLAB to:

   (a) convert this to state-space form using `tf2ss`;

   (b) find the eigenvalues of A;

   (c) convert back to a transfer function using `ss2tf` and verify you recover the original $G(s)$.

Chapter 5

Block Diagrams and Computer Simulations

Learning Objectives

After completing this chapter, you should be able to:

- Identify the fundamental elements of a block diagram: blocks, summing junctions, and pick-off points
- Reduce block diagrams to a single transfer function using series, parallel, and feedback rules
- Apply Mason's Gain Formula to find transfer functions from forward paths and loop gains
- Derive the transfer function of a DC motor from its block diagram
- Use superposition to analyse systems with multiple inputs (reference, disturbance, noise)
- Implement block diagrams in MATLAB (`series`, `feedback`, `connect`) and Simulink

Block diagrams represent interconnected subsystems visually and provide a systematic way to derive the overall transfer function. Once drawn, tools such as MATLAB and SIMULINK can simulate the system under different conditions.

5.1 Fundamental Elements of Block Diagrams

In block diagrams, a subsystem with transfer function

$$Y(s) = G(s)U(s)$$

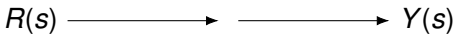
is represented by a rectangular block. The signal $U(s)$ is the input to the block and $Y(s)$ is the output.

Definition 5.1: Basic Elements of a Block Diagram

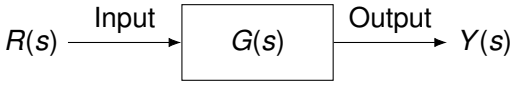
A block diagram is composed of the following fundamental components:

- **Signal lines** representing system variables.
- **System blocks** representing transfer functions.
- **Summing junctions** used to combine signals algebraically.
- **Pick-off (branch) points** used to distribute a signal to multiple locations.

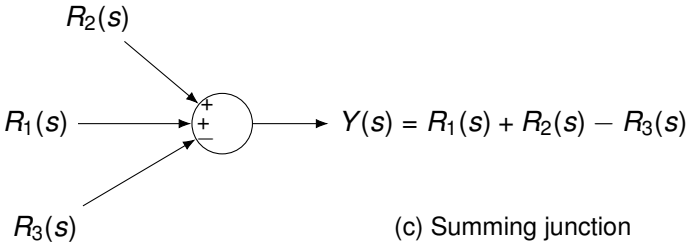
Figure 5.1 illustrates the basic building blocks used in block diagram representations.



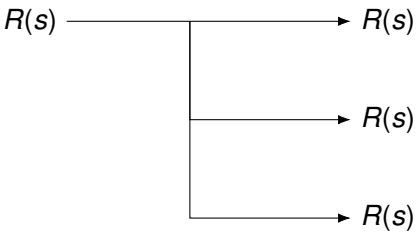
(a) Signals



(b) System



(c) Summing junction



(d) Pickoff (departure) point

Figure 5.1: Fundamental components of block diagrams: (a) representation of signals, (b) system block with transfer function $G(s)$, (c) summing junction for combining multiple input signals algebraically, and (d) pickoff point for distributing a signal to multiple paths.

5.2 Block Diagram Reduction Techniques

To determine the overall transfer function, a block diagram can be simplified using reduction rules.

Definition 5.2: Block Diagram Reduction

Block diagram reduction is the process of simplifying a complex block diagram by systematically combining blocks into an equivalent single transfer function.

Common reduction techniques include:

- Series connection
- Parallel connection
- Feedback connection
- Moving blocks across summing junctions

The three most common reductions are

$$G_{\text{series}}(s) = G_1(s)G_2(s), \qquad G_{\text{parallel}}(s) = G_1(s) + G_2(s),$$

and

$$G_{\text{feedback}}(s) = \frac{G(s)}{1 + G(s)H(s)} \quad \text{for negative feedback.}$$

For positive feedback, the sign in the denominator changes:

$$G_{\text{feedback}}(s) = \frac{G(s)}{1 - G(s)H(s)}.$$

These rules are shown in Figure 5.2.

Example 5.1: Block diagram reduction with multiple feedback paths

Question. Consider the block diagram below. Use block diagram reduction to find the closed-loop transfer function

$$T(s) = \frac{C(s)}{R(s)}.$$

Assume all blocks are proper transfer functions, and the summing junction signs are exactly as shown.

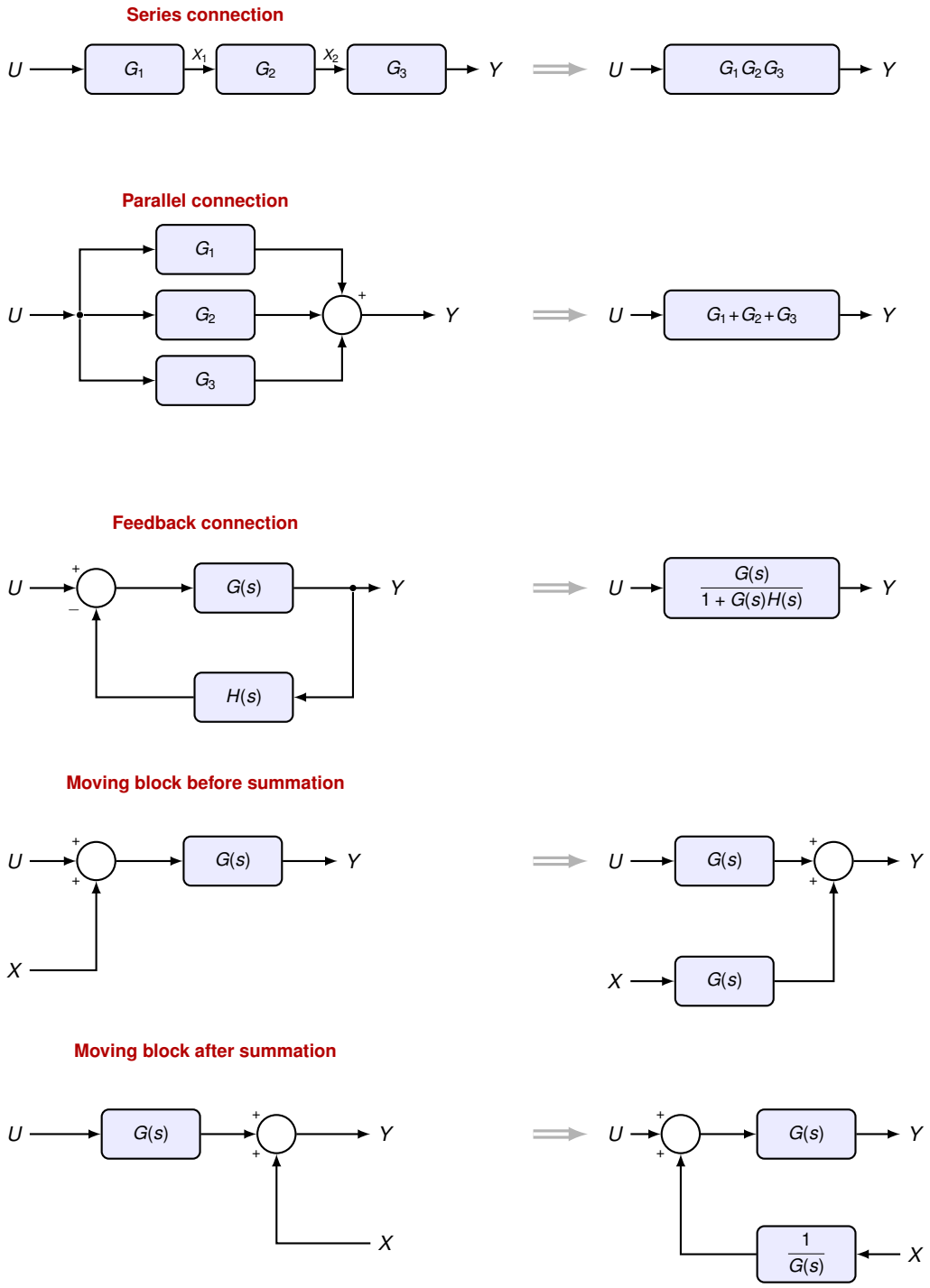
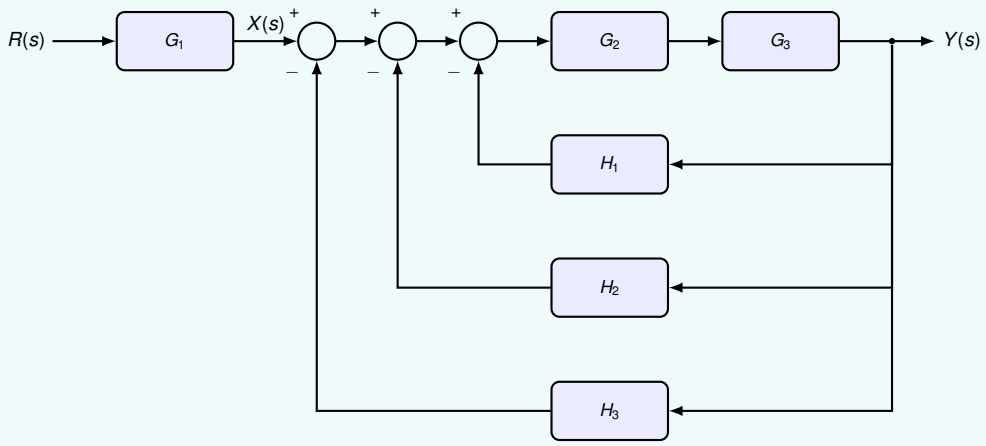


Figure 5.2: Common block diagram reduction rules: series, parallel, and feedback connections, plus moving a block across a summation point.



Solution:

Step 1: Identify the forward path. From the summing junction output to the plant output,

$$G(s) = G_2(s)\,G_3(s).$$

The signal into the summing junction from the reference is

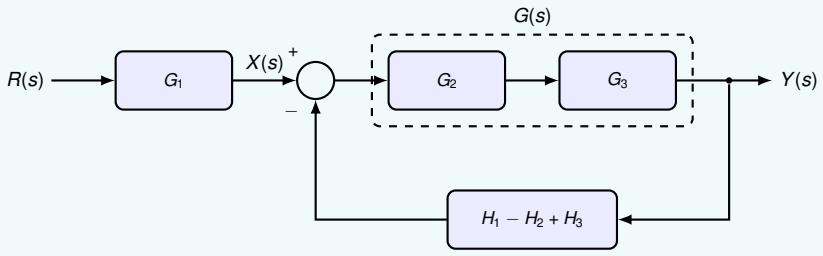
$$X(s) = G_1(s)R(s).$$

Step 2: Combine the feedback paths into an equivalent feedback signal. All three feedback blocks sense the same output $Y(s)$ and feed the summing junction in parallel (with signs). Thus the feedback contribution added at the summing node is

$$\underbrace{\big(-H_1(s)Y(s)\big)}_{\text{negative input}} + \underbrace{\big(+H_2(s)Y(s)\big)}_{\text{positive input}} + \underbrace{\big(-H_3(s)Y(s)\big)}_{\text{negative input}}.$$

So the summing junction output (call it $E(s)$) is

$$E(s) = X(s) - H_1(s)Y(s) + H_2(s)Y(s) - H_3(s)Y(s) = X(s) + \Big(-H_1(s) + H_2(s) - H_3(s)\Big)Y(s).$$



Step 3: Close the loop using the forward dynamics. Since $Y(s) = G(s)E(s) = G_2(s)G_3(s)\,E(s)$,

$$\begin{aligned} Y(s) &= G_2(s)G_3(s)\Big[X(s) + \big(-H_1(s) + H_2(s) - H_3(s)\big)Y(s)\Big] \\ &= G_2(s)G_3(s)X(s) + G_2(s)G_3(s)\big(-H_1(s) + H_2(s) - H_3(s)\big)Y(s). \end{aligned}$$

Move all terms in $Y(s)$ to the left:

$$Y(s)\Big[1 - G_2(s)G_3(s)\big(-H_1(s) + H_2(s) - H_3(s)\big)\Big] = G_2(s)G_3(s)X(s).$$

Step 4: Substitute $X(s) = G_1(s)R(s)$ and simplify. Note that

$$1 - G_2G_3(-H_1 + H_2 - H_3) = 1 + G_2G_3(H_1 - H_2 + H_3).$$

Therefore,

$$\frac{Y(s)}{R(s)} = \frac{G_2(s)G_3(s)\,G_1(s)}{1 + G_2(s)G_3(s)\Big(H_1(s) - H_2(s) + H_3(s)\Big)}.$$

5.3 Direct Transfer Function Calculation

A faster and more systematic alternative to manual reduction is based on identifying **forward paths** and **loop gains**. This approach uses Mason's gain formula, applied here directly to block diagrams without signal-flow graph notation.

Definition 5.3: Forward Path and Loop Gain

A **forward path** is a path from the input to the output that does not pass through any node more than once. The gain of the k-th forward path is denoted by $P_k(s)$.

A **loop** is a closed path that starts and ends at the same node without passing through any other node more than once. The loop gain $L_m(s)$ is the product of all block gains and summing-junction signs around that loop.

For example, a negative feedback loop with forward path $G(s)$ and feedback path $H(s)$ has signed loop gain

$$L(s) = -G(s)H(s).$$

Mason's Gain Formula gives the overall transfer function as

$$T(s) = \frac{Y(s)}{U(s)} = \frac{\displaystyle\sum_{k=1}^{N_f} P_k(s)\Delta_k(s)}{\Delta(s)},$$

where

$$\Delta(s) = 1 - \sum L_m(s) + \sum L_i(s)L_j(s) - \sum L_i(s)L_j(s)L_k(s) + \cdots.$$

The products in $\Delta(s)$ are formed only from **non-touching loops**. The factor $\Delta_k(s)$ is found from $\Delta(s)$ after removing all loops that touch the k-th forward path. However, in this class, we will not use the full generality of Mason's formula.

Two loops are said to **touch** if they have at least one common node. They are **non-touching** if they have no node in common. This distinction matters because non-touching loops can act independently, and their products must be included in $\Delta(s)$. For two non-touching loops $L_1(s)$ and $L_2(s)$,

$$\Delta(s) = 1 - \big(L_1(s) + L_2(s)\big) + L_1(s)L_2(s).$$

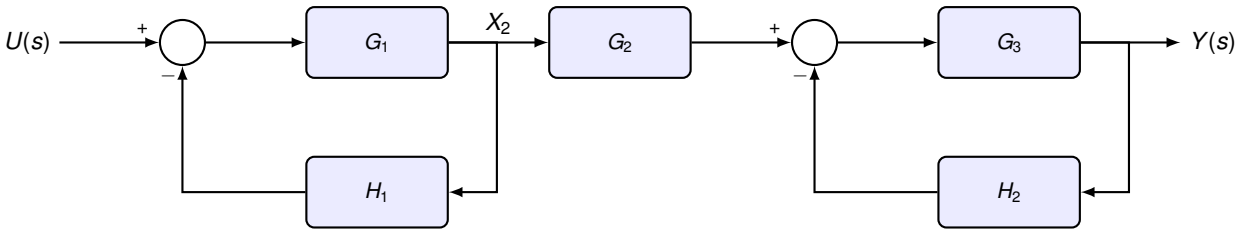


Figure 5.3: A block diagram with two non-touching feedback loops.

The last term is present only because the two loops are non-touching.

Example 5.2: Mason's formula with non-touching loops

Question. For the block diagram in Figure 5.3, find the transfer function

$$T(s) = \frac{Y(s)}{U(s)}$$

using Mason's Gain Formula.

Solution:

Step 1: Identify the forward path. There is only one forward path from $U(s)$ to $Y(s)$:

$$P_1(s) = G_1(s)G_2(s)G_3(s).$$

Step 2: Identify the loop gains. The first feedback loop contains $G_1(s)$ and $H_1(s)$. Since the feedback enters the summing junction with a negative sign,

$$L_1(s) = -G_1(s)H_1(s).$$

The second feedback loop contains $G_3(s)$ and $H_2(s)$, and it is also negative:

$$L_2(s) = -G_3(s)H_2(s).$$

These two loops are non-touching because they do not share any node.

Step 3: Form $\Delta(s)$. Because $L_1(s)$ and $L_2(s)$ are non-touching, their product must be included:

$$\begin{aligned}\Delta(s) &= 1 - \big(L_1(s) + L_2(s)\big) + L_1(s)L_2(s) \\ &= 1 + G_1(s)H_1(s) + G_3(s)H_2(s) + G_1(s)G_3(s)H_1(s)H_2(s).\end{aligned}$$

Step 4: Form $\Delta_1(s)$. The forward path $P_1(s)$ touches both loops, so both loops are removed when calculating $\Delta_1(s)$. Therefore,

$$\Delta_1(s) = 1.$$

Step 5: Apply Mason's formula.

$$T(s) = \frac{P_1(s)\Delta_1(s)}{\Delta(s)}.$$

Hence,

$$\frac{Y(s)}{U(s)} = \frac{G_1(s)G_2(s)G_3(s)}{1 + G_1(s)H_1(s) + G_3(s)H_2(s) + G_1(s)G_3(s)H_1(s)H_2(s)}$$

Remark 5.1

For many simple teaching examples, all loops touch each other and each loop touches every forward path. In that special case, $\Delta_k(s) = 1$ and

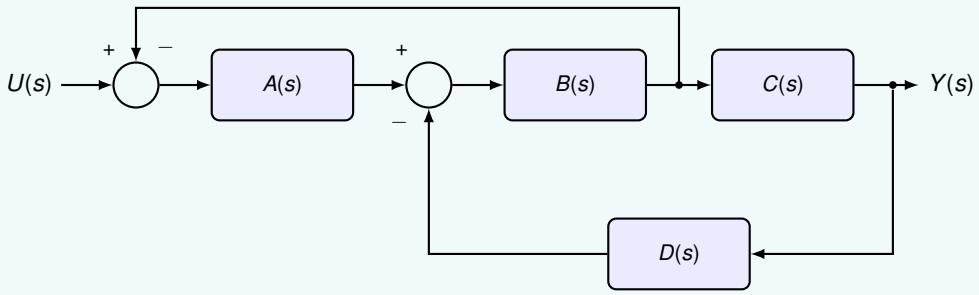
$$T(s) = \frac{\sum P_k(s)}{1 - \sum L_m(s)}.$$

This is a useful shortcut, but it should not be used blindly when a diagram contains feedforward branches or non-touching loops.

Example 5.3: Fast method to find transfer function

Question. Consider the block diagram in the figure. Using the *fast approach* (forward paths + loop gains), find the transfer function

$$T(s) = \frac{Y(s)}{U(s)}.$$



Solution:

Step 1 — Forward paths (loops opened). Open all feedback connections conceptually. There is only *one* independent forward path from input to output:

$$N_f = 1, \qquad P_1(s) = A(s)\,B(s)\,C(s).$$

Step 2 — Identify the feedback loops and their signed loop gains.

- **Loop 1 (upper loop, negative feedback at S_1):** Traverse the loop starting at the signal after B and returning to S_1:

  $$S_1 \;\to\; A(s) \;\to\; S_2 \;\to\; B(s) \;\to\; (\text{after } B) \;\to\; S_1.$$

  The feedback signal enters S_1 with a negative sign. Hence the signed loop gain is

  $$L_1(s) = -A(s)\,B(s).$$

- **Loop 2 (lower loop, negative feedback at S_2):** Traverse from S_2 through the forward blocks to the output, then through $D(s)$ back to S_2:

  $$S_2 \;\to\; B(s) \;\to\; C(s) \;\to\; Y(s) \;\to\; D(s) \;\to\; S_2.$$

  The signal enters S_2 with a negative sign, therefore

  $$L_2(s) = -B(s)\,C(s)\,D(s).$$

Step 3 — Check the shortcut condition. The two loops *touch* (they share nodes/blocks around S_2 and B), so there are **no non-touching loops**. Both loops also touch the forward path, so $\Delta_1(s) = 1$. Mason's formula reduces to

$$T(s) = \frac{P_1(s)}{\Delta(s)}, \qquad \Delta(s) = 1 - \big(L_1(s) + L_2(s)\big).$$

Step 4 — Substitute.

$$\begin{aligned} T(s) &= \frac{A(s)B(s)C(s)}{1 - \big[-A(s)B(s) - B(s)C(s)D(s)\big]} \\ &= \frac{A(s)B(s)C(s)}{1 + A(s)B(s) + B(s)C(s)D(s)}. \end{aligned}$$

Example 5.4: Transfer function using Mason's formula

Consider the block diagram shown in Figure 5.4. Using the direct method based on forward paths and loop gains, determine the overall transfer function

$$T(s) = \frac{Y(s)}{X(s)}.$$

All summing junction inputs are positive except those explicitly marked with a negative sign.

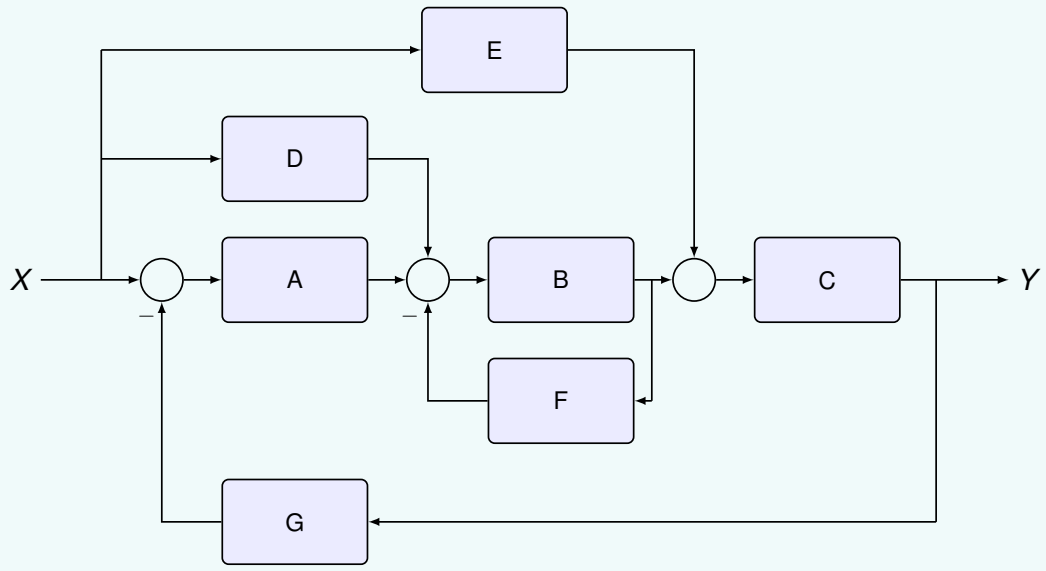


Figure 5.4: Block diagram with multiple forward paths and feedback loops.

Step 1: Identify forward paths

When all feedback loops are opened, there are three independent forward paths from the input $X(s)$ to the output $Y(s)$:

$$P_1(s) = A(s)B(s)C(s)$$
$$P_2(s) = D(s)B(s)C(s)$$
$$P_3(s) = E(s)C(s)$$

Hence

$$N_f = 3$$

and

$$\sum_{k=1}^{3} P_k(s) = ABC + DBC + EC$$

Step 2: Identify loop gains

There are two feedback loops.

Loop 1 (inner loop through F)

Because this signal enters the lower input of the summing junction with a negative sign,

$$L_1(s) = -B(s)F(s).$$

Loop 2 (global feedback through G)

Traversing the loop through the lower feedback block gives

$$A(s)B(s)C(s)G(s)$$

Again the returned signal enters the summing junction with a negative sign, therefore

$$L_2(s) = -A(s)B(s)C(s)G(s).$$

The two loops touch because both contain block $B(s)$. Therefore there are no products of non-touching loop gains in $\Delta(s)$, and

$$\Delta(s) = 1 - \big(L_1(s) + L_2(s)\big) = 1 + B(s)F(s) + A(s)B(s)C(s)G(s).$$

Step 3: Determine the path cofactors

The first two forward paths pass through block $B(s)$, so they touch both loops:

$$\Delta_1(s) = 1, \qquad \Delta_2(s) = 1.$$

The third path $P_3(s) = E(s)C(s)$ touches the global loop through $G(s)$, but it does not touch the inner loop through $F(s)$. Hence

$$\Delta_3(s) = 1 - L_1(s) = 1 + B(s)F(s).$$

Step 4: Apply Mason's formula

$$T(s) = \frac{P_1\Delta_1 + P_2\Delta_2 + P_3\Delta_3}{\Delta}.$$

Therefore,

$$T(s) = \frac{ABC + DBC + EC(1 + BF)}{1 + BF + ABCG}$$

5.4 Block Diagram Representation of a DC Motor

The DC motor model derived earlier can be represented as a block diagram by converting each governing equation into a block.

Example 5.5: Block diagram of a DC motor

The electrical equation of the motor is

$$V_{in}(t) = L\frac{di}{dt} + Ri(t) + K_b\omega(t).$$

Taking the Laplace transform gives

$$I(s) = \frac{V_{in}(s) - K_b\Omega(s)}{Ls + R}.$$

The mechanical equation is

$$K_m i(t) = J\frac{d\omega}{dt} + b\omega(t)$$

which yields

$$\Omega(s) = \frac{K_m I(s)}{Js + b}.$$

Finally, the position is obtained from

$$\Theta(s) = \frac{\Omega(s)}{s}.$$

Combining these relations produces the block diagram shown in Figure 5.5.

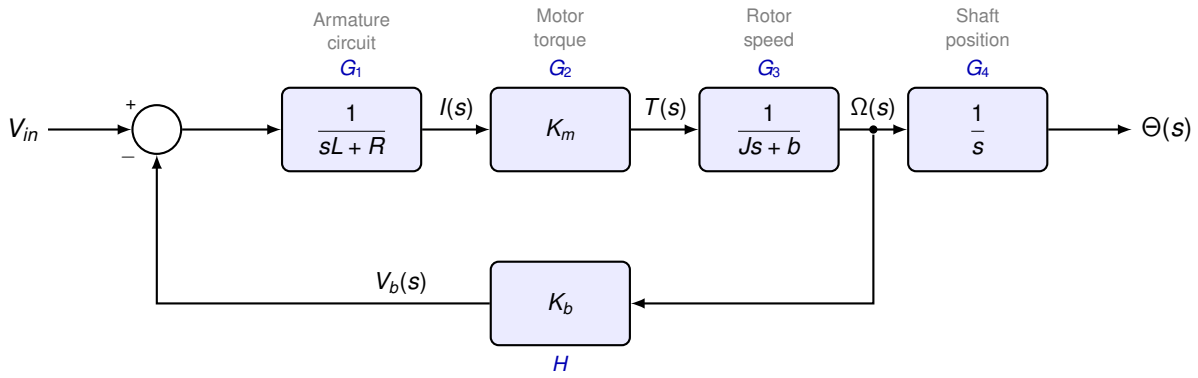


Figure 5.5: Block diagram representation of a DC motor. The back-EMF voltage $V_b(s) = K_b\,\Omega(s)$ closes the feedback loop.

Example 5.6: DC motor transfer function using block diagram reduction

Consider the motor block diagram shown earlier.

Define the blocks

$$G_1(s) = \frac{1}{Ls + R}, \quad G_2(s) = K_m$$

$$G_3(s) = \frac{1}{Js + b}, \quad G_4(s) = \frac{1}{s}, \quad H(s) = K_b.$$

The transfer function from input voltage to motor speed is

$$\frac{\Omega(s)}{V_{in}(s)} = \frac{G_1 G_2 G_3}{1 + G_1 G_2 G_3 H}.$$

The position output is obtained by including the integrator

$$\frac{\Theta(s)}{V_{in}(s)} = \frac{G_1 G_2 G_3 G_4}{1 + G_1 G_2 G_3 H}.$$

Substituting the transfer functions gives

$$\frac{\Omega(s)}{V_{in}(s)} = \frac{K_m}{(Ls + R)(Js + b) + K_b K_m}.$$

The corresponding position transfer function is

$$\frac{\Theta(s)}{V_{in}(s)} = \frac{K_m}{s\left[(Ls + R)(Js + b) + K_b K_m\right]}.$$

Example 5.7: Applying the direct method to the DC motor

Since the motor diagram contains

- one forward path
- one feedback loop

the forward path from voltage to position is

$$P_1(s) = G_1 G_2 G_3 G_4.$$

The back-emf loop is negative and returns from the speed signal, before the position integrator. Hence its signed loop gain is

$$L_1(s) = -G_1 G_2 G_3 H.$$

Therefore,

$$G(s) = \frac{P_1(s)}{1 - L_1(s)} = \frac{G_1 G_2 G_3 G_4}{1 + G_1 G_2 G_3 H}.$$

5.5 Block Diagrams with Multiple Inputs

A closed-loop system typically contains a reference input $R(s)$, a disturbance $D(s)$, and measurement noise $N(s)$. Because these systems are linear, the effect of each input can be found separately and added by superposition.

Consider the standard negative-feedback system

$$E(s) = R(s) - Y_m(s), \qquad Y_m(s) = Y(s) + N(s),$$

with controller $C(s)$, plant $G_p(s)$, and a disturbance $D(s)$ added at the plant input. Then

$$Y(s) = G_p(s)\left[C(s)E(s) + D(s)\right].$$

Substituting the feedback relation gives

$$Y(s) = G_p\, C\big(R(s) - Y(s) - N(s)\big) + G_p D(s).$$

Therefore,

$$\big(1 + C\, G_p\big)\, Y(s) = C\, G_p R(s) + G_p D(s) - C\, G_p N(s).$$

The output can be written as

$$Y(s) = \underbrace{\frac{C\, G_p}{1 + C\, G_p}}_{\text{reference response}} R(s) + \underbrace{\frac{G_p}{1 + C\, G_p}}_{\text{disturbance response}} D(s) - \underbrace{\frac{C\, G_p}{1 + C\, G_p}}_{\text{noise response}} N(s).$$

Remark 5.2

When a block diagram has several independent inputs, set all other inputs to zero and calculate one transfer function at a time. The complete response is the sum of the individual responses.

5.6 Computer Simulation and Simulink Implementation

In MATLAB, transfer functions can be combined using `series`, `parallel`, and `feedback`. In SIMULINK, the same model is built graphically using source blocks, transfer-function blocks, summing junctions, gain blocks, integrators, and scope blocks.

MATLAB Implementation

Example 5.8: MATLAB simulation of the DC motor speed response

For the DC motor model,

$$\frac{\Omega(s)}{V_{in}(s)} = \frac{K_m}{(Ls+R)(Js+b)+K_bK_m},$$

the following MATLAB script creates the transfer function and plots the step response (Listing 5.1). The numerical values are typical Quanser SRV02 motor parameters.

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

R  = 2.6;          % Armature resistance (ohm)
L  = 0.18e-3;      % Armature inductance (H)
J  = 3.90e-7;      % Motor rotor inertia (kg m^2)
b  = 0;            % Motor viscous friction neglected (N m s/rad)
Km = 7.68e-3;      % Motor torque constant (N m/A)
Kb = 7.68e-3;      % Back-emf constant (V s/rad)

Gspeed = Km/((L*s + R)*(J*s + b) + Kb*Km);

step(Gspeed)
grid on
title('DC motor speed response')
xlabel('Time (s)')
ylabel('Angular speed (rad/s)')
```

Listing 5.1: MATLAB simulation of the DC motor speed response.

The same model can also be assembled from smaller blocks, as shown in Listing 5.2:

```
G1 = 1/(L*s + R);      % Electrical dynamics
G2 = Km;               % Torque constant
G3 = 1/(J*s + b);      % Mechanical dynamics
H  = Kb;               % Back-emf feedback

Gspeed = feedback(G1*G2*G3, H);
Gpos   = Gspeed/s;
```

Listing 5.2: Constructing the DC motor block diagram in MATLAB.

Example 5.9: MATLAB verification of the transfer function in Figure 5.4

We now return to the block diagram in Figure 5.4. For that example, Mason's formula gave

$$T(s) = \frac{ABC + DBC + EC(1 + BF)}{1 + BF + ABCG}.$$

The following two methods verify the same result using numerical transfer functions. The first method follows the equations of the diagram directly. This is often the clearest way to debug the calculation before building the same interconnection with named MATLAB signals.

Method 1: Verify from the signal equations (Listing 5.3).

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

% Example transfer functions
A0 = 1/(s + 1);
B0 = 2/(s + 2);
C0 = 3/(s + 3);
D0 = 1/(s + 4);
E0 = 2/(s + 5);
F0 = 1/(s + 6);
G0 = 1/(s + 7);

% Formula obtained using Mason's Gain Formula
% minreal() cancels common pole-zero factors to give a minimal realisation
T_mason = minreal((A0*B0*C0 + D0*B0*C0 + E0*C0*(1 + B0*F0)) ...
    /(1 + B0*F0 + A0*B0*C0*G0));

% Equation-based check:
% b = B0*(a + d - F0*b), so b/(a+d) = B0/(1+B0*F0)
B_inner = feedback(B0,F0);

% y = C0*( B_inner*(A0*(x-G0*y) + D0*x) + E0*x )
% Therefore:
% y/x = (C0*B_inner*(A0+D0) + C0*E0)/(1 + C0*B_inner*A0*G0)
T_equations = minreal((C0*B_inner*(A0 + D0) + C0*E0) ...
    /(1 + C0*B_inner*A0*G0));

% This should be close to zero
err1 = minreal(tf(T_equations - T_mason),1e-7)
```

Listing 5.3: Verification from the signal equations.

Method 2: Build the interconnection using `connect` and `sumblk` (Listing 5.4).

```
% Give each block input and output signal names
A = A0; A.InputName = 'e1'; A.OutputName = 'a';
B = B0; B.InputName = 'e2'; B.OutputName = 'b';
C = C0; C.InputName = 'e3'; C.OutputName = 'y';
D = D0; D.InputName = 'x';  D.OutputName = 'd';
E = E0; E.InputName = 'x';  E.OutputName = 'e';
F = F0; F.InputName = 'b';  F.OutputName = 'fb';
G = G0; G.InputName = 'y';  G.OutputName = 'gy';

% Summing junctions from the block diagram
S1 = sumblk('e1 = x - gy');
S2 = sumblk('e2 = a + d - fb');
S3 = sumblk('e3 = b + e');

T_connect = minreal(tf(connect(A,B,C,D,E,F,G,S1,S2,S3,'x','y')),1e-7);

% Compare the two frequency responses.
```

```
w = logspace(-2,2,200);
figure
bode(T_mason, 'b', T_connect, 'r--', w)
grid on
legend('Mason formula', 'MATLAB interconnection')

err_freq = squeeze(freqresp(T_connect - T_mason,w));
max_freq_error = max(abs(err_freq))
```

Listing 5.4: Verification using connect and sumblk.

If the calculation is correct, `err1` should simplify to zero and the two Bode plots should overlap. For the numerical values used above, MATLAB gives approximately

$$\texttt{max\_freq\_error} = 3.9 \times 10^{-9},$$

which is small compared with the size of the frequency response.

Optional check: compare time responses (Listing 5.5).

```
t = 0:0.01:20;

y_mason  = step(T_mason, t);
y_connect = step(T_connect, t);

plot(t, y_mason, 'b', t, y_connect, 'r--', 'LineWidth', 1.5)
grid on
legend('Mason formula', 'MATLAB interconnection')
xlabel('Time (s)')
ylabel('Output y(t)')

max_error = max(abs(y_mason - y_connect))
```

Listing 5.5: Verification by comparing time responses.

Simulink Implementation

Example 5.10: Simulink implementation of the DC motor model

We now return to the DC motor model and implement the same speed block diagram in SIMULINK. Use the following procedure.

1. Open a new Simulink model.
2. Add a `Step` block to represent the input voltage $V_{in}(t)$.
3. Add a `Sum` block with signs + and − to subtract the back-emf voltage $K_b\omega(t)$.
4. Add a `Transfer Fcn` block with numerator 1 and denominator $[L \;\; R]$ to model $1/(Ls + R)$.
5. Add a `Gain` block with gain K_m to convert current into torque.
6. Add a second `Transfer Fcn` block with numerator 1 and denominator $[J \;\; b]$ to model $1/(Js+b)$.

7. Feed the speed output $\omega(t)$ back through a `Gain` block with gain K_b to the negative input of the first summing junction.
8. Add a `Scope` block to display $\omega(t)$. To obtain position $\theta(t)$, add an `Integrator` block after the speed signal.

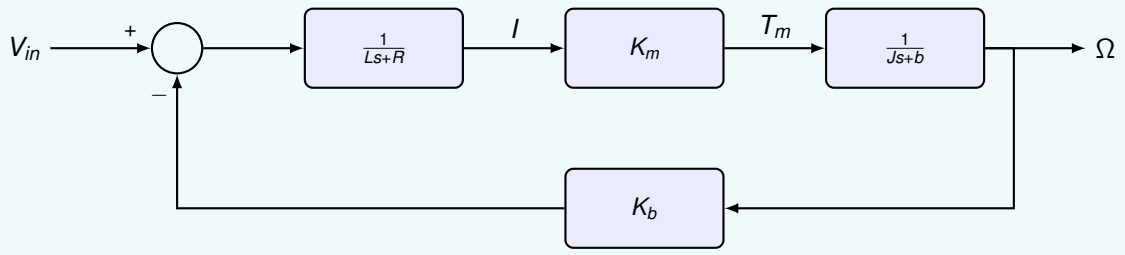


Figure 5.6: Simulink-style implementation of the DC motor speed model.

Remark 5.3

The Simulink diagram should match the mathematical block diagram. Each block represents one algebraic or dynamic relation, and each feedback signal must enter the correct sign of the summing junction.

Key Takeaways

- A block diagram represents system relationships using blocks, summing junctions, and pick-off points.
- Series blocks multiply, parallel blocks add, and feedback connections produce denominators of the form $1 \pm GH$.
- Block diagram reduction must preserve the signs and locations of all feedback signals.
- Mason's Gain Formula computes transfer functions from forward paths and loop gains — non-touching loops contribute product terms in $\Delta(s)$.
- For multiple inputs, use superposition: find the transfer function from each input separately, then sum.
- The DC motor block diagram couples electrical and mechanical dynamics with back-emf feedback.
- MATLAB (`feedback`, `connect`, `sumblk`) and Simulink can implement and verify any block diagram.

5.7 Exercises

1. Two blocks $G_1(s) = \frac{2}{s+1}$ and $G_2(s) = \frac{5}{s+3}$ are connected in series. Find the equivalent transfer function.
2. Two blocks $G_1(s) = \frac{1}{s+2}$ and $G_2(s) = \frac{3}{s+4}$ are connected in parallel. Find the equivalent transfer function.
3. A system has forward path $G(s) = \frac{10}{s(s+2)}$ and unity negative feedback. Find the closed-loop transfer function.
4. Repeat Exercise 3 for positive feedback. Comment on the difference in the denominator.
5. For a negative-feedback system with controller $C(s)$, plant $G_p(s)$, and disturbance $D(s)$ added at the plant input, derive $Y(s)/D(s)$ when $R(s) = 0$.
6. A block diagram has one forward path $P_1(s) = G_1 G_2 G_3$ and two touching negative feedback loops with unsigned gains $G_1 G_2 H_1$ and $G_2 G_3 H_2$. Use Mason's formula to find $Y(s)/R(s)$.
7. For the DC motor model, derive $\Omega(s)/V_{in}(s)$ from the electrical and mechanical equations.
8. Using the DC motor parameters in the MATLAB listing, simulate the speed response and record the steady-state speed.
9. Draw the Simulink diagram for the DC motor position model $\Theta(s)/V_{in}(s)$. Indicate where the additional integrator should be placed.
10. Explain why the sign of the back-emf feedback in a DC motor model is negative.

Chapter 6

Time Response of Systems

Learning Objectives

After completing this chapter, you should be able to:

- Derive and sketch the step response of first-order systems, identifying gain K and time constant τ
- Compute rise time and settling time for first-order systems from τ
- Classify second-order systems by damping ratio: underdamped, critically damped, overdamped
- Compute ω_n, ζ, ω_d, peak time, percentage overshoot, rise time, and settling time for underdamped second-order systems
- Relate pole locations on the s-plane to transient response characteristics
- Determine steady-state error for step, ramp, and parabolic inputs using the final value theorem and system type

A stable control system must also respond in a predictable, satisfactory way. By studying how outputs evolve after standard test inputs, we quantify performance: speed of response, overshoot, oscillation, and settling behaviour. This chapter focuses primarily on the step response, which reveals the transient characteristics most important for control design.

6.1 First-Order Systems

First-order system systems arise in thermal processes, liquid-level systems, RC circuits, and low-pass filtering.

Definition 6.1: General first-order system

A linear time-invariant first-order system can be written in the differential-equation form

$$\tau \frac{dy(t)}{dt} + y(t) = K\,u(t), \tag{6.1}$$

where

- $u(t)$ is the input,
- $y(t)$ is the output,
- K is the steady-state gain,
- $\tau > 0$ is the Time constant.

Taking the Laplace transform of (6.1) under zero initial conditions gives

$$\tau s Y(s) + Y(s) = KU(s),$$

hence the transfer function is

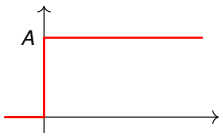
$$G(s) = \frac{Y(s)}{U(s)} = \frac{K}{\tau s + 1}. \tag{6.2}$$

Many physical systems have one dominant storage mechanism (e.g. thermal energy in a heated body, charge in a capacitor), and a first-order model often captures the main transient behaviour well, even as a first approximation for higher-order systems.

6.1.1 Step response of a first-order system

To understand the meaning of the time constant, consider the response of (6.2) to a step input of magnitude A, namely

$$u(t) = A\,u_0(t)$$



where $u_0(t)$ denotes the unit step.

The Laplace transform of the input is

$$U(s) = \frac{A}{s}.$$

Therefore,

$$Y(s) = G(s)U(s) = \frac{K}{\tau s + 1} \cdot \frac{A}{s} = \frac{KA}{s(\tau s + 1)}.$$

Using partial fractions,

$$\frac{KA}{s(\tau s + 1)} = \frac{KA}{s} - \frac{KA\tau}{\tau s + 1}.$$

Since

$$\frac{1}{\tau s + 1} = \frac{1}{\tau} \frac{1}{s + \frac{1}{\tau}},$$

the inverse Laplace transform gives

$$y(t) = KA\left(1 - e^{-t/\tau}\right) u_0(t). \tag{6.3}$$

Thus the response starts from zero and approaches KA. The quantity K sets the final value for a unit input, while τ controls how quickly the exponential term decays.

6.1.2 Meaning of the time constant

Evaluate (6.3) at $t = \tau$:

$$y(\tau) = KA\left(1 - e^{-1}\right).$$

Since

$$e^{-1} \approx 0.3679,$$

we obtain

$$y(\tau) \approx 0.632\,KA.$$

Definition 6.2: Time constant

The **time constant** τ of a first-order system is the time required for the step response to complete approximately 63.2% of its total change from the initial value to the final value. For a zero-initial-condition response, this is the same as reaching 63.2% of the final value.

A small τ means the output reaches its final value quickly; a large τ means it takes longer.

Interpretation 6.1: How to estimate τ from a response plot

If the step response of a first-order system is known experimentally, the time constant can be estimated by:

1. determining the initial value y_0 and final value $y(\infty)$,
2. computing the output change $\Delta y = y(\infty) - y_0$,
3. finding the time at which $y(t) = y_0 + 0.632\,\Delta y$.

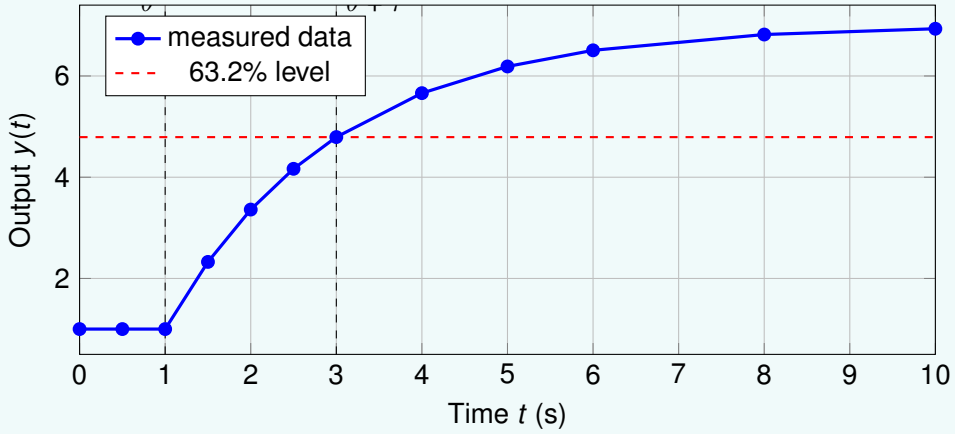
If the response has no delay, that time is the estimated time constant τ. If the response has a visible delay θ, then the time constant is measured from the end of the delay, not from the instant at which the input was changed.

Example 6.1: Estimating a delayed first-order model from step-test data

A process is initially at steady state with input $u_0 = 2$ and output $y_0 = 1$. At $t = 0$, the input is changed to $u_1 = 5$. The following stored data are obtained from the output response.

Time t (s)	Input $u(t)$	Output $y(t)$
−0.5	2	1.000
0.0	5	1.000
0.5	5	1.000
1.0	5	1.000
1.5	5	2.327
2.0	5	3.361
2.5	5	4.166
3.0	5	4.793
4.0	5	5.661
5.0	5	6.188
6.0	5	6.507
8.0	5	6.819
10.0	5	6.933

The response plot is shown below. The dashed horizontal line marks the 63.2% point of the output change. Obtain the mathematical model of the process as a first-order system with time delay, and write the time-domain response for this particular step test.



Solution:

The input step size is

$$\Delta u = u_1 - u_0 = 5 - 2 = 3.$$

The output starts at $y_0 = 1$ and settles close to

$$y(\infty) \approx 7.$$

Hence the output change is

$$\Delta y = y(\infty) - y_0 = 7 - 1 = 6.$$

The steady-state gain is therefore

$$K = \frac{\Delta y}{\Delta u} = \frac{6}{3} = 2.$$

From the data, the output does not begin to change until approximately $t = 1$ s. Thus the process delay is estimated as

$$\theta \approx 1 \text{ s}.$$

The 63.2% point must be computed from the output change, not from zero:

$$y_{63} = y_0 + 0.632\,\Delta y = 1 + 0.632(6) = 4.792.$$

From the stored data, $y(t) \approx 4.792$ at $t = 3$ s. Since the response starts after the delay,

$$\tau = 3 - 1 = 2 \text{ s}.$$

The identified first-order plus time-delay model is therefore

$$G(s) = \frac{\Delta Y(s)}{\Delta U(s)} \approx \frac{2e^{-s}}{2s+1}.$$

In time-domain form, for this particular step test,

$$y(t) = \begin{cases} 1, & 0 \le t < 1, \\ 1 + 6\left(1 - e^{-(t-1)/2}\right), & t \ge 1. \end{cases}$$

6.1.3 Rise time and settling time

Two additional measures of response speed are commonly used.

Definition 6.3: Rise time

The Rise time is the time required for the response to rise from a lower percentage to a higher percentage of its final value.

For first-order systems, rise time is commonly defined as the time taken to go from 10% to 90% of the final value.

Let t_{10} denote the time at which the response reaches 10% of the final value, and t_{90} the time at which it reaches 90%.

From

$$\frac{y(t)}{KA} = 1 - e^{-t/\tau},$$

for 10% of final value,

$$0.1 = 1 - e^{-t_{10}/\tau} \quad \Rightarrow \quad e^{-t_{10}/\tau} = 0.9 \quad \Rightarrow \quad t_{10} = -\tau \ln(0.9).$$

Similarly, for 90% of final value,

$$0.9 = 1 - e^{-t_{90}/\tau} \quad \Rightarrow \quad e^{-t_{90}/\tau} = 0.1 \quad \Rightarrow \quad t_{90} = -\tau \ln(0.1).$$

Hence the rise time is

$$\begin{aligned} t_r &= t_{90} - t_{10} \\ &= -\tau \ln(0.1) + \tau \ln(0.9) \\ &= \tau \ln\left(\frac{0.9}{0.1}\right) = \tau \ln 9. \end{aligned}$$

Numerically,

$$t_r \approx 2.2\tau.$$

For the 10%–90% convention, the result is therefore $t_r = \tau \ln 9 \approx 2.2\tau$. Other conventions exist, so any quoted rise time should be interpreted together with the percentage levels used to define it.

Definition 6.4: Settling time

The Settling time is the time required for the response to enter and remain within a specified band around the final value.

Common engineering definitions use either the $\pm 2\%$ band or the $\pm 5\%$ band.

For a first-order system, the error from final value is

$$e(t) = KA - y(t) = KAe^{-t/\tau}.$$

For the 2% settling-time criterion,

$$\frac{|e(t_s)|}{|KA|} \leq 0.02 \quad \Rightarrow \quad e^{-t_s/\tau} \leq 0.02.$$

Therefore,

$$t_s \geq -\tau \ln(0.02).$$

Numerically,

$$t_s \approx 3.9\tau \approx 4\tau.$$

For the 5% settling-time criterion,

$$e^{-t_s/\tau} \leq 0.05 \quad \Rightarrow \quad t_s \geq -\tau \ln(0.05),$$

hence

$$t_s \approx 3\tau.$$

For a stable first-order system with $\tau > 0$, the step response is monotonic. It does not oscillate, it does not overshoot, and it has no peak time in the sense used for underdamped second-order systems. The useful rules of thumb are

$$t_r \approx 2.2\tau \quad \text{(10\%–90\% rise time)}, \qquad t_s \approx 4\tau \quad \text{(2\% settling)}, \qquad t_s \approx 3\tau \quad \text{(5\% settling)}.$$

6.1.4 Graphical interpretation of the response

The step response

$$y(t) = KA\left(1 - e^{-t/\tau}\right)$$

starts at zero, rises rapidly at first, and then gradually approaches the final value KA. The slope is largest at the beginning because the output is still far from its final value. As the output gets closer to KA, the remaining error $KA - y(t)$ becomes smaller, so the curve flattens. Figure 6.1 marks the most commonly used time-domain measurements: one time constant, the 10%–90% rise time, and the 2% settling time.

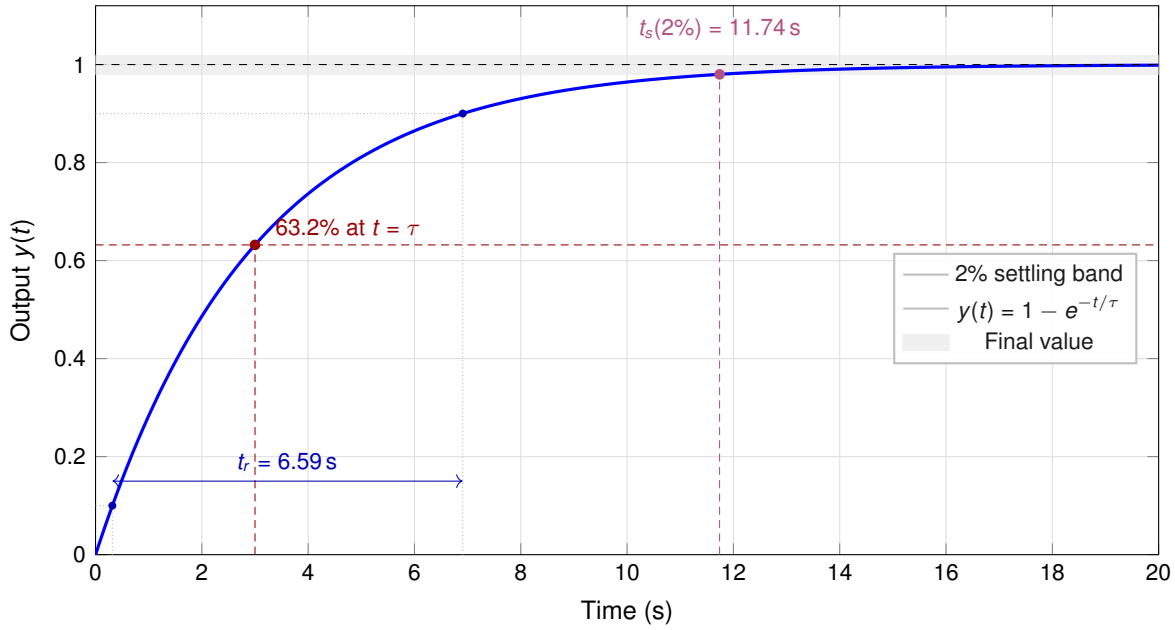


Figure 6.1: Step response of a first-order system ($\tau = 3\,\mathrm{s}$, $K = 1$) showing the time constant, 10%–90% rise time, and 2% settling time.

6.1.5 Pole location of first-order systems on the s-plane

The transfer function of a standard first-order system is

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

The pole is obtained by setting the denominator equal to zero:

$$\tau s + 1 = 0 \qquad \Rightarrow \qquad s = -\frac{1}{\tau}.$$

Therefore, a first-order system has a single pole, and its location is determined entirely by the time constant τ. This simple relation is important because it connects the algebraic form of the transfer function with both stability and response speed.

1. Stability from pole location

For a standard first-order system, the time constant is defined with $\tau > 0$, so the pole

$$s = -\frac{1}{\tau}$$

lies in the left-half plane and the system is stable. More generally, any first-order transfer function with a pole in the right-half plane is unstable, because its response grows with time.

2. Response speed from pole location

Since the pole is located at

$$s = -\frac{1}{\tau},$$

large values of τ place the pole close to the origin, while small values of τ move the pole further to the left. This agrees with the time-domain result: a smaller time constant produces a faster response. Conversely, as τ becomes larger, the pole approaches the origin and the response becomes slower.

Interpretation 6.2: Geometric meaning on the s-plane

For first-order systems, the distance of the pole from the origin along the negative real axis gives a direct indication of the speed of response:

- poles far to the left correspond to fast responses,
- poles near the origin correspond to slow responses.

For example, consider several values of τ:

$$\tau = 4, \qquad \tau = 2, \qquad \tau = 1, \qquad \tau = 0.5.$$

The corresponding poles are

$$s = -\frac{1}{4} = -0.25, \qquad s = -\frac{1}{2} = -0.5, \qquad s = -1, \qquad s = -2.$$

These poles all lie on the negative real axis, but the one corresponding to $\tau = 0.5$ is furthest to the left. Hence that system is the fastest among them.

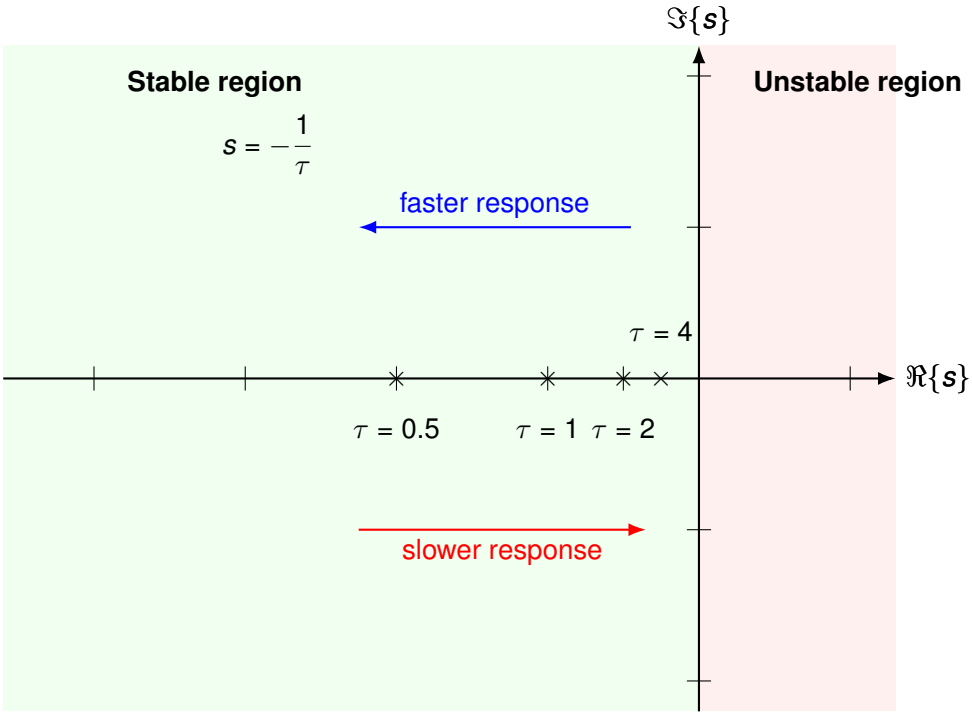


Figure 6.2: Pole locations of first-order systems for different time constants on the s-plane

Example 6.2: Pole location and speed of first-order systems

Consider the following first-order transfer functions:

$$G_1(s) = \frac{1}{4s+1}, \qquad G_2(s) = \frac{1}{2s+1}, \qquad G_3(s) = \frac{1}{s+1}, \qquad G_4(s) = \frac{1}{0.5s+1}.$$

Determine the poles and compare the speed of response.

Solution:

The poles are found from the denominators:

$$4s+1 = 0 \Rightarrow s = -0.25,$$

$$2s+1 = 0 \Rightarrow s = -0.5,$$

$$s+1 = 0 \Rightarrow s = -1,$$

$$0.5s+1 = 0 \Rightarrow s = -2.$$

Thus the poles are:

$$-0.25, \quad -0.5, \quad -1, \quad -2.$$

All poles are in the left-half plane, so all four systems are stable.

Since the pole at -2 is furthest to the left, the system

$$G_4(s) = \frac{1}{0.5s+1}$$

has the fastest response.

Since the pole at -0.25 is closest to the origin, the system

$$G_1(s) = \frac{1}{4s+1}$$

has the slowest response.

Example 6.3: Step response of a first-order system

Consider the system

$$G(s) = \frac{5}{2s+1}.$$

Find the response to a unit-step input and determine:

a) the final value,

b) the time constant,

c) the approximate rise time,

d) the approximate settling time using the 2% criterion.

Solution:

Comparing

$$G(s) = \frac{5}{2s+1}$$

with the standard form

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

we identify

$$K = 5, \qquad \tau = 2.$$

For a unit-step input,

$$U(s) = \frac{1}{s}.$$

Hence

$$Y(s) = \frac{5}{2s+1} \cdot \frac{1}{s} = \frac{5}{s(2s+1)}.$$

The time-domain response is

$$y(t) = 5\left(1 - e^{-t/2}\right), \qquad t \geq 0.$$

Therefore:

a) The final value is

$$y(\infty) = 5.$$

b) The time constant is

$$\tau = 2 \text{ s}.$$

c) The rise time is approximately

$$t_r \approx 2.2\tau = 2.2(2) = 4.4 \text{ s}.$$

d) The 2% settling time is approximately

$$t_s \approx 4\tau = 4(2) = 8 \text{ s}.$$

Example 6.4: Determining the time constant from the 63% criterion

A first-order system has a unit-step response with final value equal to 10. From an experimental plot, the output reaches 6.32 at $t = 1.8$ s. Determine the time constant and estimate the 2% settling time.

Solution:

Since

$$6.32 = 0.632 \times 10,$$

the given output corresponds to 63.2% of the final value. Therefore, by definition of the time constant,

$$\tau = 1.8 \text{ s}.$$

The 2% settling time is approximately

$$t_s \approx 4\tau = 4(1.8) = 7.2 \text{ s}.$$

Hence,

$$\boxed{\tau = 1.8 \text{ s}}, \qquad \boxed{t_s \approx 7.2 \text{ s}}.$$

Example 6.5: Response to a non-unit step

Consider the first-order system

$$G(s) = \frac{3}{4s+1}.$$

Find the response to an input step of magnitude 2.

Solution:

Here,

$$K = 3, \qquad \tau = 4, \qquad A = 2.$$

The general step-response formula is

$$y(t) = KA\left(1 - e^{-t/\tau}\right).$$

Substituting the values gives

$$y(t) = 3 \times 2\left(1 - e^{-t/4}\right) = 6\left(1 - e^{-t/4}\right), \qquad t \geq 0.$$

The final value is

$$y(\infty) = 6.$$

At $t = \tau = 4$ s,

$$y(4) = 6\left(1 - e^{-1}\right) \approx 3.79.$$

So after 4 seconds the response has reached approximately 63.2% of the final value.

Example 6.6: Finding system parameters from the response expression

A system has the unit-step response

$$y(t) = 8\left(1 - e^{-t/5}\right), \qquad t \geq 0.$$

Find:

a) the steady-state gain,

b) the time constant,

c) the transfer function.

Solution:

The standard unit-step response of a first-order system is

$$y(t) = K\left(1 - e^{-t/\tau}\right).$$

Comparing with

$$y(t) = 8\left(1 - e^{-t/5}\right),$$

we identify

$$K = 8, \qquad \tau = 5.$$

Therefore:

a) The steady-state gain is

$$K = 8.$$

b) The time constant is

$$\tau = 5 \text{ s}.$$

c) The transfer function is

$$G(s) = \frac{K}{\tau s + 1} = \frac{8}{5s + 1}.$$

6.2 Second-Order Systems

Mechanical vibration systems, RLC circuits, servo mechanisms, and many closed-loop control systems require a *second-order model*. Such systems can exhibit oscillation, overshoot, and varying degrees of damping.

The transient-specification formulae below are exact for the standard second-order model. For higher-order systems, they serve as approximations based on dominant poles.

6.2.1 General second-order model

A general linear time-invariant Second-order system may be written as

$$a_2\ddot{y}(t) + a_1\dot{y}(t) + a_0 y(t) = b_0 u(t),$$

where $a_2 > 0$, a_1, a_0, and b_0 are constants.

Under zero initial conditions, the transfer function becomes

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_0}{a_2 s^2 + a_1 s + a_0}.$$

In control engineering, it is customary to express a second-order system in the *standard form*

$$G(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \tag{6.4}$$

where:

- K is the steady-state gain,
- ω_n is the undamped Natural frequency,
- ζ is the Damping ratio.

ω_n, measured in radians per second, is the frequency at which the system would oscillate if no damping were present. The damping ratio ζ is dimensionless and describes how strongly the transient oscillation is suppressed. Small ζ gives a lightly damped, oscillatory response; larger ζ produces less overshoot and eventually removes oscillation altogether.

Definition 6.5: Damped natural frequency

For an underdamped second-order system ($0 < \zeta < 1$), the oscillation occurs at the Damped natural frequency

$$\omega_d = \omega_n\sqrt{1 - \zeta^2},$$

measured in radians per second.

The symbol ω_n is often called simply the natural frequency, but in the underdamped case the visible oscillation occurs at ω_d, not at ω_n. Damping reduces the oscillation frequency because $\omega_d = \omega_n\sqrt{1 - \zeta^2} < \omega_n$ when $0 < \zeta < 1$.

6.2.2 Poles of the standard second-order system

The poles of (6.4) are the roots of

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0.$$

Using the quadratic formula,

$$s = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}.$$

The parameter ζ mainly determines the shape of the transient response, while ω_n scales it in time.

6.2.3 Classification according to the damping ratio

Undamped system ($\zeta = 0$)

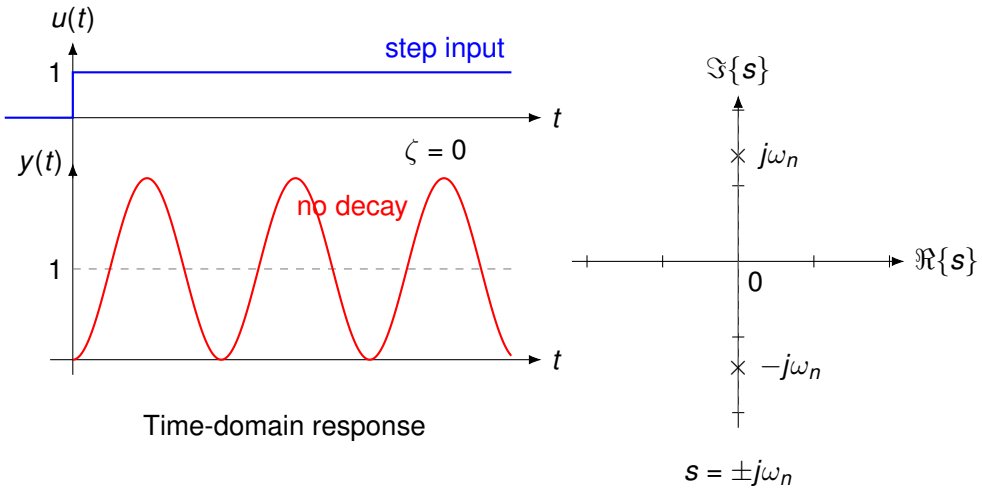
If $\zeta = 0$, then

$$G(s) = \frac{K\omega_n^2}{s^2 + \omega_n^2},$$

and the poles are

$$s = \pm j\omega_n.$$

The poles lie on the imaginary axis and the system oscillates indefinitely. An undamped second-order system does not settle to its final value in the usual transient-response sense. Instead, it exhibits sustained oscillation. Figure 6.3 shows the step response of the undamped second-order system, which continuously oscillates about the final value of the step response.



Time-domain response

$s = \pm j\omega_n$

Poles on the imaginary axis

Figure 6.3: Undamped second-order system: the step response oscillates indefinitely and does not settle, while the poles lie on the imaginary axis at $s = \pm j\omega_n$.

Underdamped system ($0 < \zeta < 1$)

If $0 < \zeta < 1$, then

$$s = -\zeta\omega_n \pm j\omega_n\sqrt{1 - \zeta^2} = -\zeta\omega_n \pm j\omega_d.$$

The poles are complex conjugates in the left-half plane. The response is oscillatory, but the oscillations decay with time. Figure 6.4 shows the unit-step response of an underdamped second-order system and the corresponding pole locations on the s-plane.

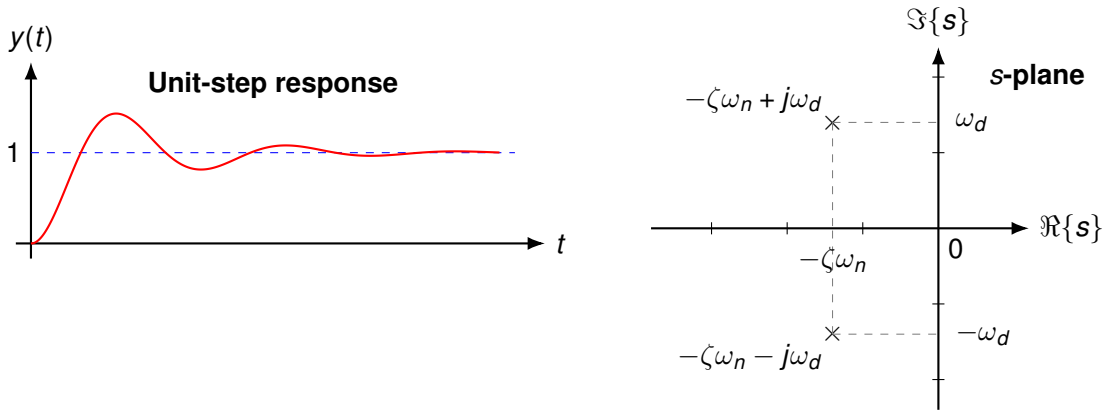


Figure 6.4: Underdamped second-order system: decaying oscillatory step response and complex-conjugate poles in the left-half plane

Critically damped system ($\zeta = 1$)

If $\zeta = 1$, then the characteristic equation has a repeated real root at

$$s = -\omega_n.$$

Thus, both poles coincide on the negative real axis. The response is non-oscillatory and, among the standard second-order cases, gives the fastest approach to the steady-state value without overshoot. Figure 6.5 illustrates the corresponding unit-step response and pole locations.

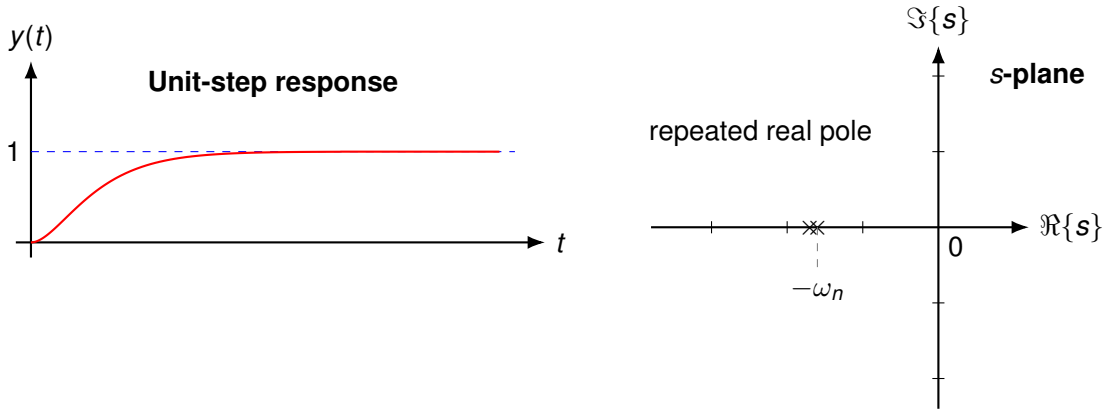


Figure 6.5: Critically damped second-order system: non-oscillatory step response and repeated real pole at $s = -\omega_n$

Overdamped system ($\zeta > 1$)

If $\zeta > 1$, then the characteristic equation has two distinct real negative roots. More explicitly, the poles are

$$p_{1,2} = -\omega_n \left(\zeta \mp \sqrt{\zeta^2 - 1} \right), \qquad \zeta > 1.$$

Hence, the poles lie at separate locations on the negative real axis. The response is non-oscillatory because there is no imaginary part, but it is usually slower than the critically damped response because the slower real pole dominates the tail of the transient. Figure 6.6 shows the unit-step response and the corresponding pole locations.

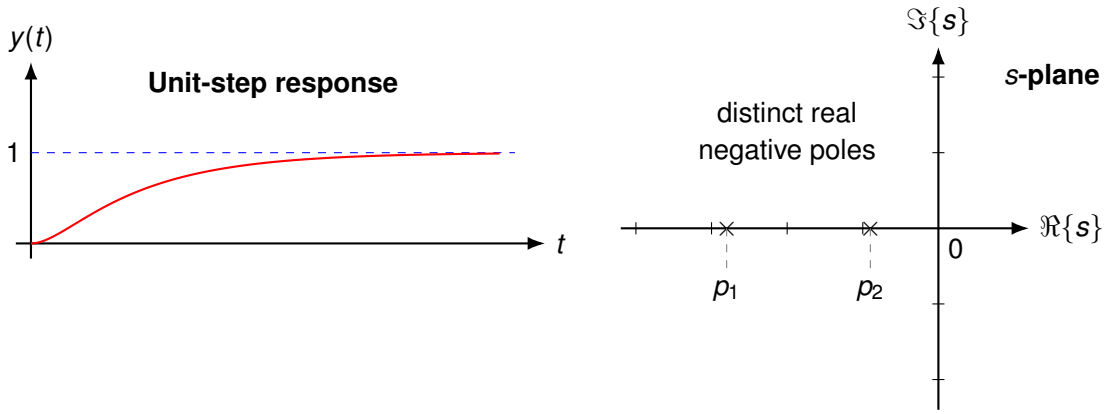


Figure 6.6: Overdamped second-order system: non-oscillatory unit-step response and two distinct real negative poles

Unstable system ($\zeta < 0$)

Within the standard second-order form with $\omega_n > 0$, a negative damping ratio gives poles with positive real part, so the response grows with time and the system is unstable.

Interpretation 6.3: Effect of damping ratio on pole locations

Changing ζ moves the poles through a sequence of qualitatively different cases. At $\zeta = 0$, the poles lie on the imaginary axis and the response does not decay. For $0 < \zeta < 1$, the poles are complex conjugates in the left-half plane and the response oscillates with decaying amplitude. At $\zeta = 1$, the poles meet as a repeated real pole. For $\zeta > 1$, they split into two distinct negative real poles and the response remains non-oscillatory.

6.2.4 Step response of the standard second-order system

Consider the unit-step input

$$u(t) = u_0(t), \qquad U(s) = \frac{1}{s}.$$

For the system (6.4),

$$Y(s) = \frac{K\omega_n^2}{s\left(s^2 + 2\zeta\omega_n s + \omega_n^2\right)}.$$

The exact time-domain expression depends on ζ, but the final value is simple: for a stable standard second-order system and a unit-step input, $y(\infty) = K$.

Undamped case ($\zeta = 0$)

When $\zeta = 0$, the standard second-order transfer function

$$G(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

reduces to

$$G(s) = \frac{K\omega_n^2}{s^2 + \omega_n^2}.$$

For a unit-step input,

$$U(s) = \frac{1}{s},$$

so the output in the Laplace domain is

$$Y(s) = G(s)U(s) = \frac{K\omega_n^2}{s(s^2 + \omega_n^2)}.$$

Using partial fractions, write

$$\frac{K\omega_n^2}{s(s^2 + \omega_n^2)} = \frac{A}{s} + \frac{Bs + C}{s^2 + \omega_n^2}.$$

Multiplying through by $s(s^2 + \omega_n^2)$ gives

$$K\omega_n^2 = A(s^2 + \omega_n^2) + (Bs + C)s.$$

Expanding,

$$K\omega_n^2 = (A + B)s^2 + Cs + A\omega_n^2.$$

Equating coefficients yields

$$A + B = 0, \qquad C = 0, \qquad A\omega_n^2 = K\omega_n^2.$$

Hence

$$A = K, \qquad B = -K, \qquad C = 0.$$

Therefore,

$$Y(s) = \frac{K}{s} - \frac{Ks}{s^2 + \omega_n^2}.$$

Taking the inverse Laplace transform,

$$y(t) = K\left(1 - \cos\omega_n t\right), \qquad t \geq 0.$$

This response oscillates indefinitely because there is no damping term to dissipate energy. For a unit-step input, the output oscillates about the level K, but it never settles to that value.

Underdamped case $(0 < \zeta < 1)$

This is the most important case in control applications. The unit-step response is

$$y(t) = K\left[1 - e^{-\zeta\omega_n t}\left(\cos(\omega_d t) + \frac{\zeta}{\sqrt{1 - \zeta^2}}\sin(\omega_d t)\right)\right], \qquad t \geq 0,$$

where

$$\omega_d = \omega_n\sqrt{1 - \zeta^2}.$$

An equivalent form is

$$y(t) = K\left[1 - \frac{e^{-\zeta\omega_n t}}{\sqrt{1 - \zeta^2}}\sin(\omega_d t + \phi)\right], \qquad \phi = \arccos(\zeta).$$

The expression has two parts: an oscillatory term at frequency ω_d, and an exponential envelope $e^{-\zeta\omega_n t}$ that gradually removes the oscillation. This is why the response can overshoot and still settle to K in the long run.

Critically damped case ($\zeta = 1$)

For $\zeta = 1$, the unit-step response is

$$y(t) = K\left[1 - e^{-\omega_n t}(1 + \omega_n t)\right], \qquad t \geq 0.$$

Overdamped case ($\zeta > 1$)

For $\zeta > 1$, the poles are real and distinct: $p_{1,2} = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}$. The unit-step response is a sum of two decaying exponentials:

$$y(t) = K\left[1 - c_1 e^{p_1 t} - c_2 e^{p_2 t}\right], \qquad t \geq 0,$$

where the constants are

$$c_1 = \frac{p_2}{p_2 - p_1}, \qquad c_2 = \frac{-p_1}{p_2 - p_1}.$$

The response is non-oscillatory. The pole closer to the imaginary axis (the "slow" pole) dominates the settling behaviour.

The most widely used transient-response formulae for overshoot, peak time, rise time, and settling time are normally stated for the underdamped case $0 < \zeta < 1$, because that is the case in which these quantities are most informative.

6.2.5 Transient-response characteristics

For underdamped second-order systems, the following quantities are especially important.

Peak time

The Peak time is the time taken for the response to reach its first maximum value. It is given by

$$t_p = \frac{\pi}{\omega_d} = \frac{\pi}{\omega_n\sqrt{1 - \zeta^2}}.$$

Percentage overshoot

The Percentage overshoot is the amount by which the response exceeds the final value, expressed as a percentage. For an underdamped second-order system ($0 < \zeta < 1$), it is given by

$$\%OS = 100\, e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}. \tag{6.5}$$

This expression shows that the overshoot depends only on the damping ratio ζ. In particular, it decreases monotonically as ζ increases, as shown in Figure 6.7.

In design problems, however, the situation is often reversed. A designer may begin with a transient-response specification such as

$$\%OS \leq 10\%$$

or

$$\%OS \leq 5\%,$$

and must then determine the damping ratio required to satisfy that specification. Rearranging (6.5) gives the corresponding formula for ζ.

Let

$$M_p = \frac{\%OS}{100}.$$

Then

$$M_p = e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}.$$

Taking natural logarithms,

$$\ln(M_p) = -\frac{\zeta\pi}{\sqrt{1-\zeta^2}}.$$

Solving for ζ yields

$$\zeta = \frac{-\ln\left(\frac{\%OS}{100}\right)}{\sqrt{\pi^2 + \left[\ln\left(\frac{\%OS}{100}\right)\right]^2}}.$$

Remark 6.1

The formula

$$\%OS = 100\, e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}$$

is mainly used for **analysis**, whereas

$$\zeta = \frac{-\ln\left(\frac{\%OS}{100}\right)}{\sqrt{\pi^2 + \left[\ln\left(\frac{\%OS}{100}\right)\right]^2}}$$

is especially useful for **design**, when the overshoot specification is known in advance.

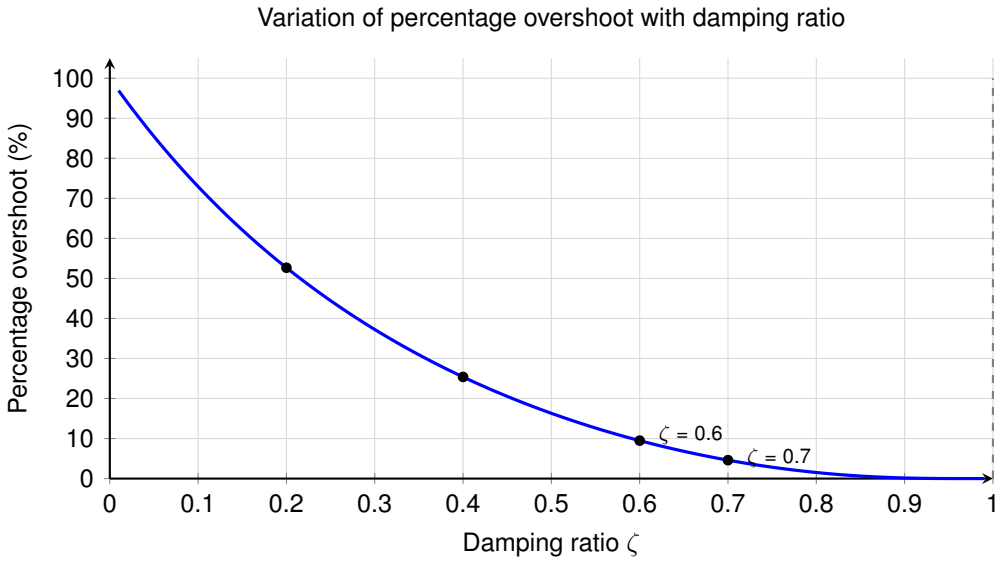


Figure 6.7: Percentage overshoot as a function of damping ratio for an underdamped second-order system

Figure 6.7 shows that the percentage overshoot drops rapidly as the damping ratio increases. Small values of ζ lead to highly oscillatory responses with large overshoot, whereas moderate damping significantly

reduces the first peak. For example, $\zeta \approx 0.6$ gives an overshoot of less than 10%, while for ζ close to 1 the overshoot approaches zero.

Remark 6.2

As ζ increases, the overshoot decreases. When $\zeta \geq 1$, the standard second-order step response does not overshoot, because the response is no longer underdamped and becomes non-oscillatory.

Rise time

The **rise time** is taken as the time required for the response to rise from 10% to 90% of its final value. For a standard underdamped second-order system, an exact closed-form expression for this 10%–90% rise time is not especially convenient. Different textbooks use different empirical approximations. One commonly used approximation, valid reasonably well for

$$0.3 \leq \zeta \leq 0.8,$$

is

$$t_r \approx \frac{2.16\zeta + 0.6}{\omega_n}.$$

This approximation is particularly useful in preliminary design and analysis, since it relates the rise time directly to the damping ratio ζ and the undamped natural frequency ω_n.

For fixed ζ, a larger ω_n reduces the rise time and makes the response faster. For fixed ω_n, a smaller ζ usually gives a faster initial rise, but this comes at the cost of more oscillation and overshoot.

Settling time

Using the 2% criterion, the settling time is commonly approximated by

$$t_s \approx \frac{4}{\zeta\omega_n}.$$

For the 5% criterion, a common approximation is

$$t_s \approx \frac{3}{\zeta\omega_n}.$$

For an underdamped second-order system, the main approximations used in this chapter are

$$\omega_d = \omega_n\sqrt{1-\zeta^2}, \qquad t_p = \frac{\pi}{\omega_d}, \qquad \%OS = 100\, e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}},$$

$$t_r \approx \frac{2.16\zeta + 0.6}{\omega_n} \quad \text{for } 0.3 \leq \zeta \leq 0.8, \qquad t_s \approx \frac{4}{\zeta\omega_n} \quad \text{(2\% criterion)}.$$

Figure 6.8 illustrates the main transient-response characteristics of an underdamped second-order system. In particular, it shows the rise time t_r, the peak time t_p, the percentage overshoot, and the settling time t_s based on the 2% criterion.

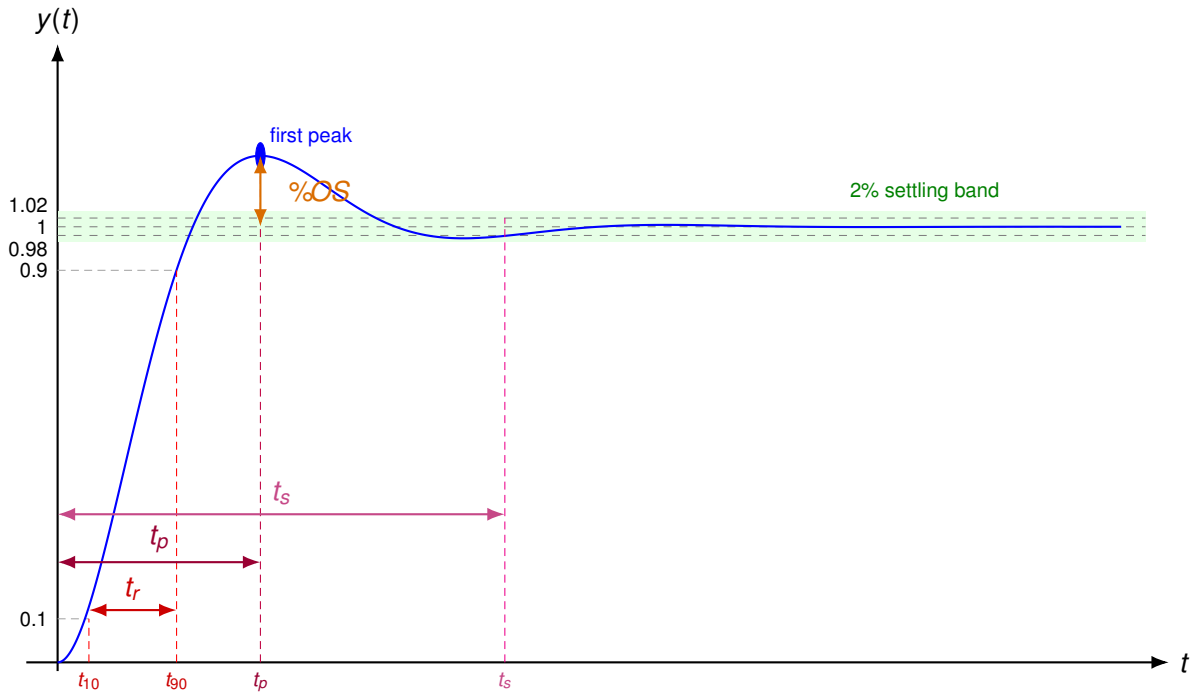


Figure 6.8: Transient-response characteristics of an underdamped second-order step response, showing the rise time $t_r = t_{90} - t_{10}$, peak time t_p, settling time t_s, and percentage overshoot $\%OS$. The 2% settling band is visually enlarged for clarity.

6.2.6 Effect of pole locations on transient response

The poles provide immediate insight into the response characteristics.

For underdamped systems, poles are located at

$$s = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2}.$$

Hence:

- the real part $-\zeta\omega_n$ governs the rate of decay,
- the imaginary part ω_d governs the oscillation frequency.

Interpretation 6.4: How ζ and ω_n affect the response

Increasing ω_n generally makes the response faster because the poles move further from the origin. Increasing ζ reduces oscillation and overshoot because the poles move toward the negative real axis. Very small ζ gives a lightly damped response with large overshoot, while $\zeta = 1$ gives the fastest non-oscillatory response in the standard second-order family.

Figure 6.9 shows the pole locations and their geometric relations with transient-response characteristics. Figure 6.10 compares these effects for three different families of pole motions. When the poles move horizontally with a constant imaginary part, the oscillation frequency remains unchanged because ω_d is fixed, and therefore the peak time stays the same, while the settling time and overshoot vary according to the changing real part.

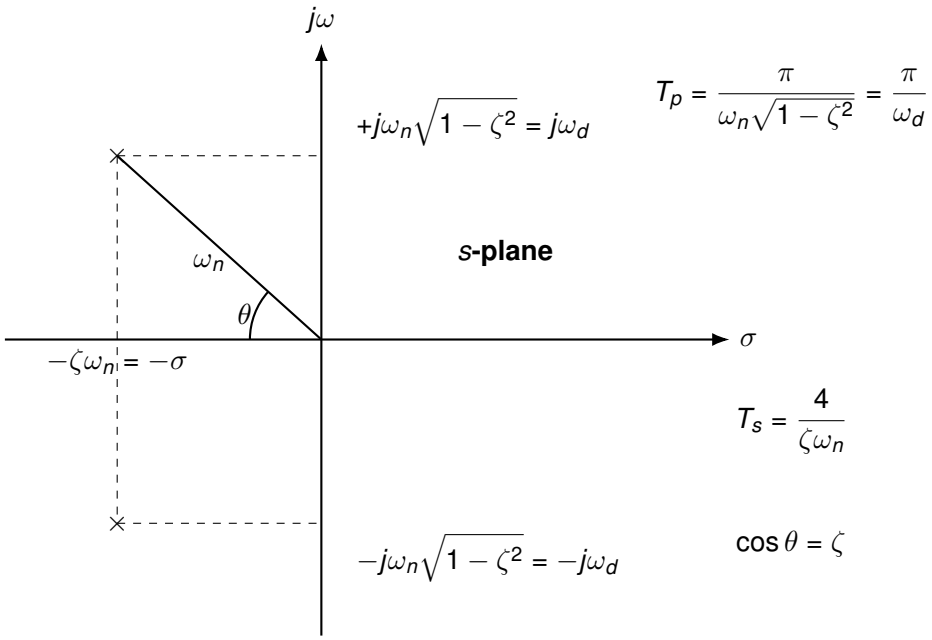


Figure 6.9: Geometric interpretation of the dominant poles of an underdamped second-order system on the *s*-plane

When the poles move vertically with a constant real part, the exponential decay rate remains unchanged, so the systems have essentially the same settling behaviour, whereas the oscillation frequency, peak time, and overshoot change as the imaginary part varies.

Finally, when the poles move along a ray from the origin corresponding to constant damping ratio, the systems have the same damping ratio and therefore exhibit similar overshoot and overall oscillatory character, while their speed changes with distance from the origin: moving further from the origin increases ω_n, making both the peak time and settling time smaller.

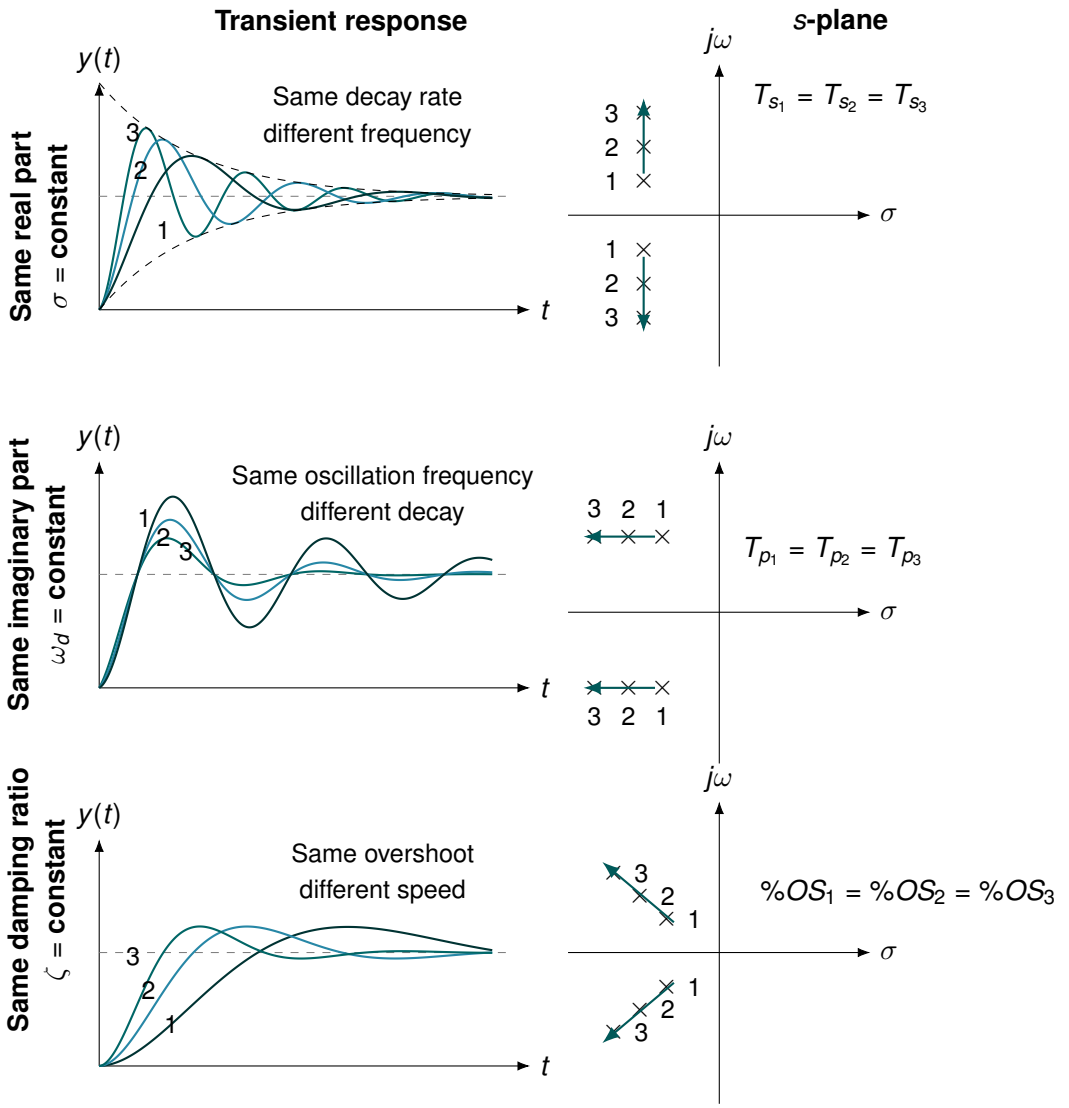


Figure 6.10: Comparison of transient responses and corresponding pole motions for second-order systems: same real part, same imaginary part, and same damping ratio.

Example 6.7: Identifying ζ, ω_n, and the poles

Consider the system

$$G(s) = \frac{25}{s^2 + 4s + 25}.$$

Find:

a) the damping ratio,

b) the undamped natural frequency,

c) the damped natural frequency,

d) the poles,

e) the type of transient response.

Solution:

Compare the denominator

$$s^2 + 4s + 25$$

with the standard form

$$s^2 + 2\zeta\omega_n s + \omega_n^2.$$

Equating coefficients gives

$$\omega_n^2 = 25 \quad \Rightarrow \quad \omega_n = 5 \text{ rad/s},$$

and

$$2\zeta\omega_n = 4.$$

Substituting $\omega_n = 5$,

$$2\zeta(5) = 4 \quad \Rightarrow \quad \zeta = 0.4.$$

The damped natural frequency is

$$\omega_d = \omega_n\sqrt{1-\zeta^2} = 5\sqrt{1-0.4^2} = 5\sqrt{0.84} \approx 4.583 \text{ rad/s}.$$

The poles are

$$s = -\zeta\omega_n \pm j\omega_d = -2 \pm j4.583.$$

Since $0 < \zeta < 1$, the system is underdamped. Therefore the response is oscillatory with decaying amplitude.

Example 6.8: Computing percentage overshoot

A second-order system has damping ratio $\zeta = 0.5$. Determine the percentage overshoot.

Solution:

Using

$$\%OS = 100\, e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}},$$

we obtain

$$\%OS = 100\, e^{-\frac{0.5\pi}{\sqrt{1-0.5^2}}}.$$

Since

$$\sqrt{1-0.25} = 0.8660,$$

$$\%OS \approx 100\, e^{-\frac{0.5\pi}{0.8660}} = 100\, e^{-1.8138} = 16.3\%.$$

Example 6.9: Computing peak time, rise time, and settling time

Consider the standard second-order system with $\zeta = 0.3$, and $\omega_n = 8$ rad/s.

Find:

a) the damped natural frequency,

b) the peak time,

c) the approximate rise time,

d) the approximate settling time using the 2% criterion.

Solution:

First,

$$\omega_d = \omega_n\sqrt{1-\zeta^2} = 8\sqrt{1-0.3^2} = 8\sqrt{0.91} \approx 7.631 \text{ rad/s}.$$

Thus,

$$\boxed{\omega_d \approx 7.631 \text{ rad/s}}.$$

The peak time is

$$t_p = \frac{\pi}{\omega_d} = \frac{\pi}{7.631} \approx 0.412 \text{ s}.$$

Since $0.3 \leq \zeta \leq 0.8$, we may use the rise-time approximation

$$t_r \approx \frac{2.16\zeta + 0.6}{\omega_n}.$$

Substituting $\zeta = 0.3$ and $\omega_n = 8$,

$$t_r \approx \frac{2.16(0.3) + 0.6}{8} = \frac{0.648 + 0.6}{8} = \frac{1.248}{8} = 0.156 \text{ s}.$$

Hence,

$$\boxed{t_r \approx 0.156 \text{ s}}.$$

The settling time is

$$t_s \approx \frac{4}{\zeta\omega_n} = \frac{4}{0.3 \times 8} = \frac{4}{2.4} \approx 1.667 \text{ s}.$$

Therefore,

$$\boxed{t_p \approx 0.412 \text{ s}}, \quad \boxed{t_r \approx 0.156 \text{ s}}, \quad \boxed{t_s \approx 1.667 \text{ s}}.$$

Example 6.10: Finding ζ from a specified overshoot

A designer requires that the percentage overshoot of a closed-loop second-order system should not exceed 10%. Estimate the required damping ratio.

Solution:

We use

$$10 = 100\, e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}.$$

Thus

$$0.1 = e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}.$$

Taking natural logarithms,

$$\ln(0.1) = -2.3026.$$

Hence

$$-\frac{\zeta\pi}{\sqrt{1-\zeta^2}} = -2.3026.$$

Solving gives approximately

$$\zeta \approx 0.591.$$

Hence a damping ratio of about

$$\boxed{\zeta \approx 0.59}$$

is required to keep the overshoot near or below 10%.

Example 6.11: Classifying the response from pole locations

The poles of a second-order system are

$$s = -3 \pm j4.$$

Discuss the likely transient response.

Solution:

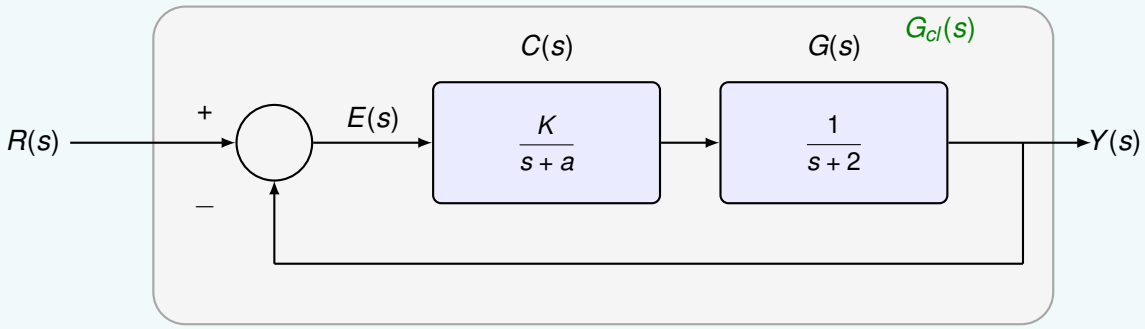
The poles are complex conjugates in the left-half plane. Therefore the system is stable and underdamped. The real part -3 indicates exponential decay, while the imaginary part 4 indicates oscillation. Hence the transient response will:

- oscillate,
- exhibit overshoot,
- decay to the steady-state value as time increases.

Example 6.12: Controller design from settling-time and overshoot specifications

Consider the unity-feedback system shown below, where

$$C(s) = \frac{K}{s+a}, \qquad G(s) = \frac{1}{s+2}.$$



Determine the controller parameters K and a such that the closed-loop system satisfies

$$T_s^{5\%} \leq 1 \text{ s}, \qquad \%OS \leq 10\%.$$

Then verify the design in MATLAB.

Solution:

The open-loop transfer function is

$$C(s)G(s) = \frac{K}{(s+a)(s+2)}.$$

With unity feedback, the closed-loop transfer function becomes

$$G_{cl}(s) = \frac{C(s)G(s)}{1 + C(s)G(s)} = \frac{\dfrac{K}{(s+a)(s+2)}}{1 + \dfrac{K}{(s+a)(s+2)}}$$
$$= \frac{K}{s^2 + (a+2)s + (2a+K)}. \tag{6.6}$$

We compare the *denominator* with the standard second-order characteristic polynomial

$$s^2 + 2\zeta\omega_n s + \omega_n^2.$$

Hence, by equating coefficients,

$$2\zeta\omega_n = a + 2, \qquad \omega_n^2 = 2a + K. \tag{6.7}$$

Step 1: Use the settling-time specification.

For the 5% settling-time criterion,

$$T_s^{5\%} \approx \frac{3}{\zeta\omega_n}.$$

The requirement $T_s{}^{5\%} \leq 1$ gives

$$\frac{3}{\zeta\omega_n} \leq 1 \qquad \Rightarrow \qquad \zeta\omega_n \geq 3.$$

For a design on the specification boundary, we choose

$$\zeta\omega_n = 3. \tag{6.8}$$

Step 2: Use the overshoot specification.

For a standard underdamped second-order system,

$$\%OS = 100\, e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}.$$

Setting $\%OS = 10\%$ gives the minimum damping ratio required:

$$\zeta = \frac{-\ln(0.1)}{\sqrt{\pi^2 + \ln^2(0.1)}} \approx 0.591.$$

Thus any

$$\zeta \geq 0.591$$

satisfies the overshoot requirement. For convenience, choose

$$\zeta = 0.6.$$

Step 3: Determine ω_n.

From (6.8),

$$\zeta\omega_n = 3.$$

With $\zeta = 0.6$,

$$0.6\,\omega_n = 3 \qquad \Rightarrow \qquad \omega_n = 5 \text{ rad/s}.$$

Step 4: Match coefficients to find a and K.

From (6.7),

$$a + 2 = 2\zeta\omega_n = 2(0.6)(5) = 6.$$

Hence

$$a = 4.$$

Also,

$$2a + K = \omega_n^2 = 25.$$

Substituting $a = 4$,

$$2(4) + K = 25 \qquad \Rightarrow \qquad K = 17.$$

Therefore, the controller parameters are

$$\boxed{a = 4}, \qquad \boxed{K = 17}.$$

The resulting controller is

$$\boxed{C(s) = \frac{17}{s+4}}.$$

Step 5: Write the closed-loop transfer function.

Substituting $a = 4$ and $K = 17$ into (6.6),

$$G_{cl}(s) = \frac{17}{s^2 + 6s + 25}.$$

The denominator matches the standard second-order characteristic polynomial with

$$\omega_n = 5, \qquad \zeta = 0.6.$$

Therefore, the pole-based transient characteristics are expected to be

$$T_s{}^{5\%} \approx \frac{3}{\zeta\omega_n} = \frac{3}{0.6 \times 5} = 1 \text{ s},$$

and

$$\%OS = 100\, e^{-\frac{0.6\pi}{\sqrt{1-0.6^2}}} \approx 9.48\%.$$

So the settling-time and overshoot specifications are satisfied.

However, note that the numerator is not equal to ω_n^2, so the closed-loop transfer function is not exactly the canonical standard second-order form with unit DC gain. In fact,

$$G_{cl}(0) = \frac{17}{25},$$

so a unit-step input produces a final value of $17/25$, not 1.

MATLAB verification

The MATLAB script Listing 2 can be used to verify the design. The MATLAB results should confirm that the closed-loop system has a 5% settling time close to 1 second and an overshoot below 10%.

Listing 2 MATLAB verification of the designed controller

```
clc; clear; close all;

s = tf('s');

K = 17;
a = 4;

Gc = K/(s+a);
G  = 1/(s+2);

T = feedback(Gc*G,1);

disp('Closed-loop transfer function:')
T

figure;
step(T);
grid on;
title('Closed-Loop Step Response');
xlabel('Time (s)');
ylabel('Output');

info = stepinfo(T,'SettlingTimeThreshold',0.05);
disp('Step response information (5% settling criterion):')
disp(info)

zeta = 0.6;
OS = 100*exp(-(zeta*pi)/sqrt(1-zeta^2));
fprintf('Predicted overshoot = %.2f %%\n', OS);

```

Example 6.13: Design of a rotational second-order system

Consider the rotational mechanical system shown in Figure 6.11, where the torsional stiffness is

$$K = 5\ \text{N m/rad}.$$

Determine the inertia J and damping coefficient D such that the response to a unit-step torque input $T(t)$ has

$$\%OS = 20\%, \qquad T_s^{2\%} = 2\ \text{s}.$$

Solution:

Let $\theta(t)$ denote the angular displacement. The equation of motion of the rotational spring–mass–damper system is

$$J\ddot{\theta}(t) + D\dot{\theta}(t) + K\theta(t) = T(t).$$

Taking the Laplace transform under zero initial conditions gives

$$(Js^2 + Ds + K)\Theta(s) = T(s).$$

Hence the transfer function from input torque to angular displacement is

$$\frac{\Theta(s)}{T(s)} = \frac{1}{Js^2 + Ds + K}.$$

Dividing numerator and denominator by J,

$$\frac{\Theta(s)}{T(s)} = \frac{1/J}{s^2 + \frac{D}{J}s + \frac{K}{J}}.$$

Comparing the denominator with the standard second-order form

$$s^2 + 2\zeta\omega_n s + \omega_n^2,$$

we identify

$$2\zeta\omega_n = \frac{D}{J}, \qquad \omega_n^2 = \frac{K}{J}. \tag{6.9}$$

Step 1: Find the damping ratio from the overshoot specification.

For an underdamped second-order system,

$$\%OS = 100\, e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}.$$

With $\%OS = 20\%$,

$$\zeta = \frac{-\ln(0.2)}{\sqrt{\pi^2 + \ln^2(0.2)}}.$$

Since

$$\ln(0.2) \approx -1.6094,$$

we obtain

$$\zeta \approx \frac{1.6094}{\sqrt{\pi^2 + 1.6094^2}} \approx 0.456.$$

Thus,

$$\boxed{\zeta \approx 0.456}.$$

Step 2: Use the settling-time specification to find ω_n.

For the 2% settling-time criterion,

$$T_s^{2\%} \approx \frac{4}{\zeta\omega_n}.$$

Given that $T_s^{2\%} = 2$ s,

$$2 = \frac{4}{\zeta\omega_n} \qquad \Rightarrow \qquad \zeta\omega_n = 2.$$

Hence

$$\omega_n = \frac{2}{\zeta} = \frac{2}{0.456} \approx 4.386 \text{ rad/s}.$$

Therefore,

$$\boxed{\omega_n \approx 4.386 \text{ rad/s}}.$$

Step 3: Determine the inertia J.

From (6.9),

$$\omega_n^2 = \frac{K}{J} \qquad \Rightarrow \qquad J = \frac{K}{\omega_n^2}.$$

Substituting $K = 5$ and $\omega_n \approx 4.386$,

$$J = \frac{5}{(4.386)^2} = 0.260 \text{ kg m}^2.$$

Step 4: Determine the damping coefficient D.

Again from (6.9),

$$2\zeta\omega_n = \frac{D}{J}.$$

Since $\zeta\omega_n = 2$,

$$2\zeta\omega_n = 4.$$

Thus $D = 4J$.

Substituting $J = 0.260$,

$$D = 4(0.260) = 1.040 \text{ N m s/rad}.$$

Hence,

$$\boxed{D = 1.04 \text{ N m s/rad}}.$$

Check of the design

Using these values,

$$\omega_n = \sqrt{\frac{K}{J}} = \sqrt{\frac{5}{0.260}} \approx 4.386 \text{ rad/s},$$

and

$$\zeta = \frac{D}{2\sqrt{KJ}} = \frac{1.04}{2\sqrt{5 \times 0.260}} \approx 0.456.$$

Therefore,

$$\%OS \approx 20\%, \qquad T_s^{2\%} \approx \frac{4}{\zeta\omega_n} = \frac{4}{2} = 2 \text{ s},$$

which confirms that the specifications are satisfied.

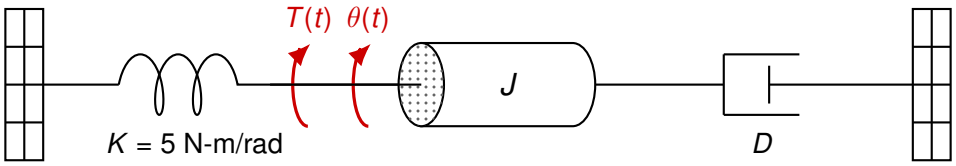


Figure 6.11: Rotational spring–inertia–damper system driven by the input torque $T(t)$

6.3 Steady-State Error

The transient-response measures above describe how a system reaches its final value. Equally important is how *accurate* that final value is. The Steady-state error measures the remaining difference between the reference input and the output after transients have died away.

6.3.1 Why steady-state error matters

A system may be stable with acceptable transient behaviour, yet settle with a significant offset from the desired value.

- In a temperature-control system, the user may request 22°C, but the room may settle at 20°C. The system is stable, yet it has a steady-state error.
- In a cruise-control system, the vehicle may settle below the commanded speed when travelling uphill if the controller cannot remove the offset.
- In a position-control system, a motor may move close to the demanded angle but stop short of the exact target.

6.3.2 Definition of steady-state error in the time domain

For a unity-feedback system, the tracking error is

$$e(t) = r(t) - y(t),$$

where $r(t)$ is the reference input and $y(t)$ is the output. For a non-unity feedback system, the signal entering the summing junction is instead

$$e(t) = r(t) - y_m(t),$$

where $y_m(t)$ is the measured output returned by the feedback path. These two definitions coincide only when the feedback path has unity gain.

Definition 6.6: Steady-state error

The **steady-state error** is the limiting value of the error signal as time tends to infinity:

$$e_{ss} = \lim_{t\to\infty} e(t),$$

provided this limit exists.

If $e_{ss} = 0$, the system tracks the input exactly in steady state. If $e_{ss} \neq 0$, the system settles with a residual offset. If the limit does not exist, then a constant steady-state error cannot be defined.

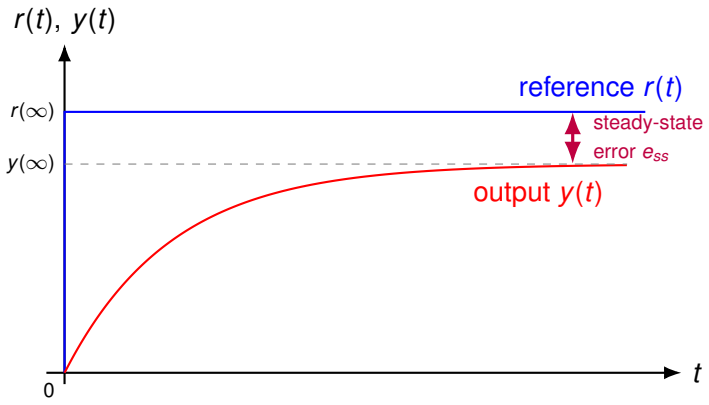


Figure 6.12: Illustration of the tracking error and steady-state error. The reference $r(t)$ is the desired signal, the output $y(t)$ is the actual system response, and the vertical gap between their final values is the steady-state error e_{ss}.

Figure 6.12 illustrates the tracking error for the unity-feedback case. The steady-state error is meaningful only if the error signal approaches a constant value; if the system is unstable or oscillates indefinitely, e_{ss} does not exist.

6.3.3 Steady-state error in the Laplace domain

If $E(s)$ is the Laplace transform of $e(t)$, the final value theorem gives

$$e_{ss} = \lim_{t \to \infty} e(t) = \lim_{s \to 0} sE(s),$$

provided the final value theorem is applicable.

The final value theorem requires that all poles of $sE(s)$ lie in the open left half-plane — in particular, the closed-loop system must be stable.

6.3.4 Error transfer function for a feedback system

Consider the standard feedback system shown in Figure 6.13, with forward transfer function $G(s)$ and feedback transfer function $H(s)$. Here $E(s)$ denotes the error at the summing junction, namely the difference between the reference and the measured feedback signal $B(s)$.

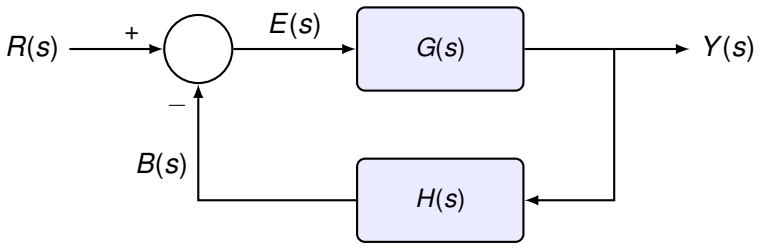


Figure 6.13: Standard feedback system showing the reference $R(s)$, error $E(s)$, output $Y(s)$, and feedback signal $B(s)$. Note that when $H(s) = 1$, the feedback connection is called *unity-feedback system*.

Since

$$Y(s) = G(s)E(s), \qquad B(s) = H(s)Y(s), \qquad E(s) = R(s) - B(s),$$

we obtain

$$E(s) = R(s) - H(s)Y(s) = R(s) - H(s)G(s)E(s).$$

Hence,

$$E(s)\big(1 + G(s)H(s)\big) = R(s),$$

so

$$E(s) = \frac{R(s)}{1 + G(s)H(s)}.$$

For the important special case of unity feedback, $H(s) = 1$, this becomes

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

Therefore, for a stable unity-feedback system,

$$e_{ss} = \lim_{s \to 0} s \, \frac{R(s)}{1 + G(s)}.$$

6.3.5 Practical meaning for standard test inputs

The steady-state error depends on both the system and the applied input. The standard test signals are:

$$\text{step: } R(s) = \frac{A}{s}, \qquad \text{ramp: } R(s) = \frac{A}{s^2}, \qquad \text{parabolic: } R(s) = \frac{A}{s^3}.$$

Equivalently, in the time domain these may be written as

$$r(t) = A\,u_0(t), \qquad r(t) = At\,u_0(t), \qquad r(t) = \frac{A}{2}t^2\,u_0(t).$$

- A **step input** models a sudden change in the desired value, such as a new temperature set-point or a new speed command.
- A **ramp input** models a command that changes at a constant rate, such as a position reference generated by constant desired velocity.
- A **parabolic input** models an accelerating command.

Example 6.1: Finite steady-state error to a step input

Consider a unity-feedback system with forward transfer function

$$G(s) = \frac{10}{s + 2}.$$

Find the steady-state error at the summing junction for a unit-step input.

Solution:

For a unit-step input,

$$R(s) = \frac{1}{s}.$$

Since the system has unity feedback,

$$E(s) = \frac{R(s)}{1 + G(s)} = \frac{\frac{1}{s}}{1 + \frac{10}{s+2}}.$$

Simplifying,

$$E(s) = \frac{1}{s} \cdot \frac{s+2}{s+12} = \frac{s+2}{s(s+12)}.$$

Using the final value theorem,

$$e_{ss} = \lim_{s \to 0} sE(s) = \lim_{s \to 0} \frac{s+2}{s+12} = \frac{2}{12} = \frac{1}{6}.$$

Hence,

$$e_{ss} = \frac{1}{6}.$$

The system is stable, but it does not remove the final offset completely. In practice, this means that the output settles near the demanded value, but not exactly at it.

Example 6.2: Zero steady-state error to a step input

Consider a unity-feedback system with forward transfer function

$$G(s) = \frac{10}{s(s+2)}.$$

Find the steady-state error for a unit-step input.

Solution:

For a unit-step input,

$$R(s) = \frac{1}{s}.$$

Then

$$E(s) = \frac{R(s)}{1 + G(s)} = \frac{\frac{1}{s}}{1 + \frac{10}{s(s+2)}}.$$

Simplifying,

$$E(s) = \frac{1}{s} \cdot \frac{s(s+2)}{s(s+2) + 10} = \frac{s+2}{s^2 + 2s + 10}.$$

Therefore,

$$e_{ss} = \lim_{s \to 0} sE(s) = \lim_{s \to 0} \frac{s(s+2)}{s^2 + 2s + 10} = 0.$$

Hence,

$$e_{ss} = 0.$$

The extra pole at the origin in the forward path allows the system to eliminate the steady-state error for a step input.

Example 6.3: Finite steady-state error to a ramp input

Consider again the unity-feedback system

$$G(s) = \frac{10}{s(s+2)}.$$

Find the steady-state error for a unit-ramp input.

Solution:

For a unit-ramp input,

$$R(s) = \frac{1}{s^2}.$$

Then

$$E(s) = \frac{R(s)}{1+G(s)} = \frac{\frac{1}{s^2}}{1+\frac{10}{s(s+2)}}.$$

Simplifying,

$$E(s) = \frac{1}{s^2} \cdot \frac{s(s+2)}{s(s+2)+10} = \frac{s+2}{s\left(s^2+2s+10\right)}.$$

Using the final value theorem,

$$e_{ss} = \lim_{s\to 0} sE(s) = \lim_{s\to 0} \frac{s+2}{s^2+2s+10} = \frac{2}{10} = \frac{1}{5}.$$

Hence,

$$e_{ss} = \frac{1}{5}.$$

This means that the system cannot follow a ramp input perfectly. It tracks the ramp with a constant long-term lag.

Example 6.4: Zero steady-state error to a ramp input

Consider a unity-feedback system with forward transfer function

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

Find the steady-state error for a unit-ramp input.

Solution:

For a unit-ramp input,

$$R(s) = \frac{1}{s^2}.$$

Hence,

$$E(s) = \frac{R(s)}{1+G(s)} = \frac{\frac{1}{s^2}}{1+\frac{10}{s^2(s+2)}}.$$

Simplifying,

$$E(s) = \frac{1}{s^2} \cdot \frac{s^2(s+2)}{s^2(s+2)+10} = \frac{s+2}{s^2(s+2)+10}.$$

Therefore,

$$e_{ss} = \lim_{s\to 0} sE(s) = \lim_{s\to 0} \frac{s(s+2)}{s^2(s+2)+10} = 0.$$

Hence,

$$\boxed{e_{ss} = 0.}$$

With two poles at the origin in the forward path, the system can track a ramp input without steady-state error.

Example 6.5: Unbounded steady-state error for an input that is too demanding

Consider the unity-feedback system

$$G(s) = \frac{10}{s(s+2)}.$$

Find the steady-state error for a unit-parabolic input.

Solution:

For a unit-parabolic input,

$$r(t) = \frac{1}{2}t^2\, u_0(t), \qquad R(s) = \frac{1}{s^3}.$$

Thus,

$$E(s) = \frac{R(s)}{1+G(s)} = \frac{\frac{1}{s^3}}{1+\frac{10}{s(s+2)}}.$$

Simplifying,

$$E(s) = \frac{1}{s^3} \cdot \frac{s(s+2)}{s(s+2)+10} = \frac{s+2}{s^2\left(s^2+2s+10\right)}.$$

Now,

$$sE(s) = \frac{s+2}{s\left(s^2+2s+10\right)}.$$

As $s \to 0$, this expression becomes unbounded. Therefore, the steady-state error does not approach a finite constant; instead, it grows without bound.

Hence the system cannot track a parabolic input satisfactorily. In practical terms, the reference changes too aggressively for the available low-frequency dynamics of the forward path.

Example 6.6: Steady-state error for a system with sensor dynamics

Consider the feedback system shown in Figure 6.13, where the forward transfer function is

$$G(s) = \frac{15}{s+3},$$

and the feedback path contains sensor dynamics described by

$$H(s) = \frac{2}{s+4}.$$

This is therefore **not** a unity-feedback system, because the measured output is modified by the sensor before it is fed back.

Find the steady-state error for a unit-step input.

Solution:

Since the feedback path contains sensor dynamics, we must use the general error transfer function

$$E(s) = \frac{R(s)}{1 + G(s)H(s)}.$$

For a unit-step input,

$$R(s) = \frac{1}{s}.$$

Hence

$$E(s) = \frac{\frac{1}{s}}{1 + \frac{15}{s+3} \cdot \frac{2}{s+4}}.$$

First simplify the loop term:

$$G(s)H(s) = \frac{30}{(s+3)(s+4)}.$$

Therefore,

$$E(s) = \frac{1}{s} \cdot \frac{(s+3)(s+4)}{(s+3)(s+4) + 30}.$$

Expanding,

$$(s+3)(s+4) = s^2 + 7s + 12,$$

so

$$E(s) = \frac{1}{s} \cdot \frac{s^2 + 7s + 12}{s^2 + 7s + 42} = \frac{s^2 + 7s + 12}{s\left(s^2 + 7s + 42\right)}.$$

Using the final value theorem,

$$e_{ss} = \lim_{s \to 0} sE(s) = \lim_{s \to 0} \frac{s^2 + 7s + 12}{s^2 + 7s + 42} = \frac{12}{42} = \frac{2}{7}.$$

Hence,

$$e_{ss} = \frac{2}{7}.$$

This example shows that when the sensor has its own dynamics, the feedback path is no longer unity. The sensor affects the loop transfer function through $H(s)$, and therefore it also affects the error signal seen at the summing junction. In such cases, the shortcut

$$E(s) = \frac{R(s)}{1 + G(s)}$$

cannot be used; instead, the full expression

$$E(s) = \frac{R(s)}{1 + G(s)H(s)}$$

must be applied.

6.3.6 An important observation about input poles and zero steady-state error

The worked examples reveal an important pattern. For the standard polynomial test signals,

$$R(s) = \frac{A}{s}, \qquad \frac{A}{s^2}, \qquad \frac{A}{s^3},$$

the poles of the input all lie at the origin. Therefore, what matters is how many poles at the origin appear in the forward transfer function.

Remark 6.3

To obtain zero steady-state error, the forward path must contain sufficient copies of the pole of the input signal. For the standard step, ramp, and parabolic inputs, this means sufficient poles at the origin in the forward transfer function.

In particular:

- to obtain zero steady-state error for a step input, at least one pole at the origin is required in the forward path;
- to obtain zero steady-state error for a ramp input, at least two poles at the origin are required;
- to obtain zero steady-state error for a parabolic input, at least three poles at the origin are required.

Interpretation 6.5: Physical meaning of poles at the origin

pole at the origin corresponds to an integrating effect. Integrating action accumulates error over time and forces the controller or plant to continue acting until the offset is removed. More poles at the origin provide stronger low-frequency tracking capability, but they also make the system more demanding to stabilise.

6.3.7 A compact procedure for finding steady-state error

For a stable LTI feedback system, the steady-state error can be found systematically as follows.

1. Write down the reference input $R(s)$.
2. Derive the error transfer function $E(s)$.
3. Compute
$$e_{ss} = \lim_{s\to 0} sE(s).$$
4. Interpret the result:
   - $e_{ss} = 0$: exact tracking in steady state,
   - $e_{ss} \neq 0$: finite offset remains,
   - e_{ss} unbounded or limit does not exist: the input cannot be tracked with a finite constant steady-state error.

6.3.8 System type and static error constants

Steady-state error results can be summarised using System type and Static error constants.

Definition 6.7: System Type

The **type number** of a unity-feedback system is the number of free integrators (poles at $s = 0$) in the open-loop transfer function $G(s)$.

- Type 0: no integrator in $G(s)$,
- Type 1: one integrator ($1/s$ factor),
- Type 2: two integrators ($1/s^2$ factor).

The **static error constants** are defined as:

$$K_p = \lim_{s\to 0} G(s), \qquad K_v = \lim_{s\to 0} s\,G(s), \qquad K_a = \lim_{s\to 0} s^2 G(s),$$

where K_p is the *position error constant*, K_v is the *velocity error constant*, and K_a is the *acceleration error constant*.

The steady-state errors for standard inputs in a stable unity-feedback system are summarised in Table 6.1.

Table 6.1: Steady-state error e_{ss} for unity-feedback systems by system type and input class

Input	Type 0	Type 1	Type 2
Step $(1/s)$	$\dfrac{1}{1+K_p}$	0	0
Ramp $(1/s^2)$	∞	$\dfrac{1}{K_v}$	0
Parabolic $(1/s^3)$	∞	∞	$\dfrac{1}{K_a}$

Remark 6.4

Increasing the system type (adding integrators) improves tracking accuracy for polynomial inputs, but makes stability harder to achieve. This is a fundamental trade-off in controller design.

Example 6.7: Rate feedback design with steady-state error analysis

Consider the rate feedback system shown in Figure 6.14. The plant transfer function is $\dfrac{K}{s(s+5)}$ and the inner (rate) feedback gain is $K_t\, s$.

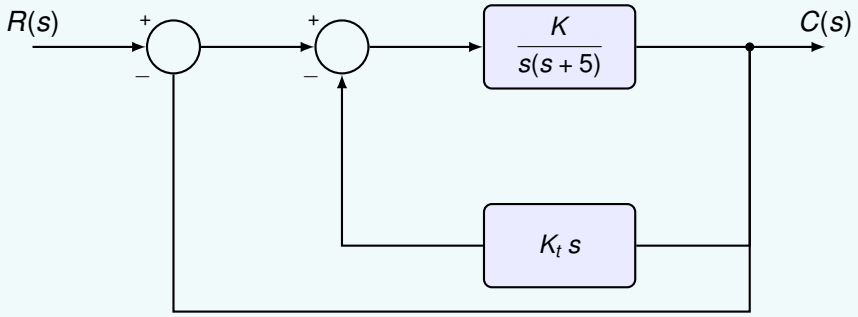


Figure 6.14: Rate feedback system with plant $K/[s(s+5)]$ and derivative feedback $K_t\, s$.

(a) Find the values of K and K_t so that the closed-loop poles are at $s_{1,2} = -3 \pm j4$.

(b) Determine the system type and order.

(c) Find the steady-state error for a unit ramp input.

(d) Verify the design in MATLAB by plotting the step response, ramp response, tracking error, and pole locations.

Solution:

(a) Finding K and K_t.

First, reduce the inner loop. The forward path of the inner loop is $G(s) = K/[s(s+5)]$ and its feedback is $H(s) = K_t\, s$, so the inner closed-loop transfer function is

$$G_{\text{inner}}(s) = \frac{K/[s(s+5)]}{1 + K_t\, s \cdot K/[s(s+5)]} = \frac{K}{s^2 + (5 + KK_t)\, s}.$$

The outer loop has unity feedback, so the overall closed-loop transfer function is

$$T(s) = \frac{G_{\text{inner}}(s)}{1 + G_{\text{inner}}(s)} = \frac{K}{s^2 + (5 + KK_t)\, s + K}.$$

The desired characteristic equation from the specified poles is

$$(s + 3 - j4)(s + 3 + j4) = s^2 + 6s + 25 = 0.$$

Comparing coefficients:

$$K = 25, \qquad 5 + KK_t = 6 \quad \Longrightarrow \quad K_t = \frac{1}{25} = 0.04.$$

(b) System type and order.

The open-loop transfer function of the outer loop is

$$G_{\text{ol}}(s) = G_{\text{inner}}(s) = \frac{25}{s(s+6)}.$$

There is one free integrator $(1/s)$ in the forward path, so the system is **type 1** and **order 2**.

(c) Steady-state error.

For a type-1 system, the steady-state error to a unit ramp is $e_{ss} = 1/K_v$, where

$$K_v = \lim_{s \to 0} s\, G_{\text{ol}}(s) = \lim_{s \to 0} s \cdot \frac{25}{s(s+6)} = \frac{25}{6} \approx 4.17.$$

Therefore

$$e_{ss} = \frac{1}{K_v} = \frac{6}{25} = 0.24.$$

As a consistency check, the closed-loop transfer function is $T(s) = 25/(s^2 + 6s + 25)$, which is a standard second-order system with $\omega_n = 5$ rad/s and $\zeta = 0.6$, confirming the poles at $s = -3 \pm j4$.

(d) MATLAB verification.

Write a MATLAB script to verify the design. The code should construct the closed-loop transfer function, confirm the pole locations, plot the step and ramp responses, and read off the steady-state error.

```
% Rate feedback design verification
K  = 25;
Kt = 0.04;

% Inner-loop reduction
G      = tf(K, [1 5 0]);         % K / [s(s+5)]
H_inner = tf([Kt 0], 1);        % Kt*s
G_inner = feedback(G, H_inner);

```

```
% Outer closed-loop (unity feedback)
T = feedback(G_inner, 1);
disp('Closed-loop TF:'); T

% Verify pole locations
p = pole(T);
fprintf('Poles: %.2f +/- j%.2f\n', real(p(1)), abs(imag(p(1))));

% Step response
figure;
subplot(2,2,1);  step(T, 3);  grid on;
title('Step response');
info = stepinfo(T);
fprintf('Overshoot = %.1f%%, Ts = %.2f s\n', info.Overshoot, info.SettlingTime
    );

% Ramp response
subplot(2,2,2);
t = 0:0.01:5;  r = t;
y = lsim(T, r, t);
plot(t, r, 'r--', t, y, 'b', 'LineWidth', 1.5);
legend('Reference','Output'); grid on;
title('Ramp response');

% Ramp tracking error
subplot(2,2,3);
plot(t, r - y, 'b', 'LineWidth', 1.5); grid on;
yline(6/25, 'r--', 'LineWidth', 1.5);
title('Ramp tracking error');
legend('e(t)', 'e_{ss} = 6/25');

% Pole-zero map
subplot(2,2,4);  pzmap(T);  grid on;
title('Pole-zero map');

```

Listing 6.1: MATLAB verification of the rate feedback design.

Figure 6.15 shows the results. The step response confirms $\%OS = 9.5\%$ and $T_s \approx 1.2\,\text{s}$, consistent with $\zeta = 0.6$. The ramp error settles to $e_{ss} = 6/25 = 0.24$, matching the analytical prediction.

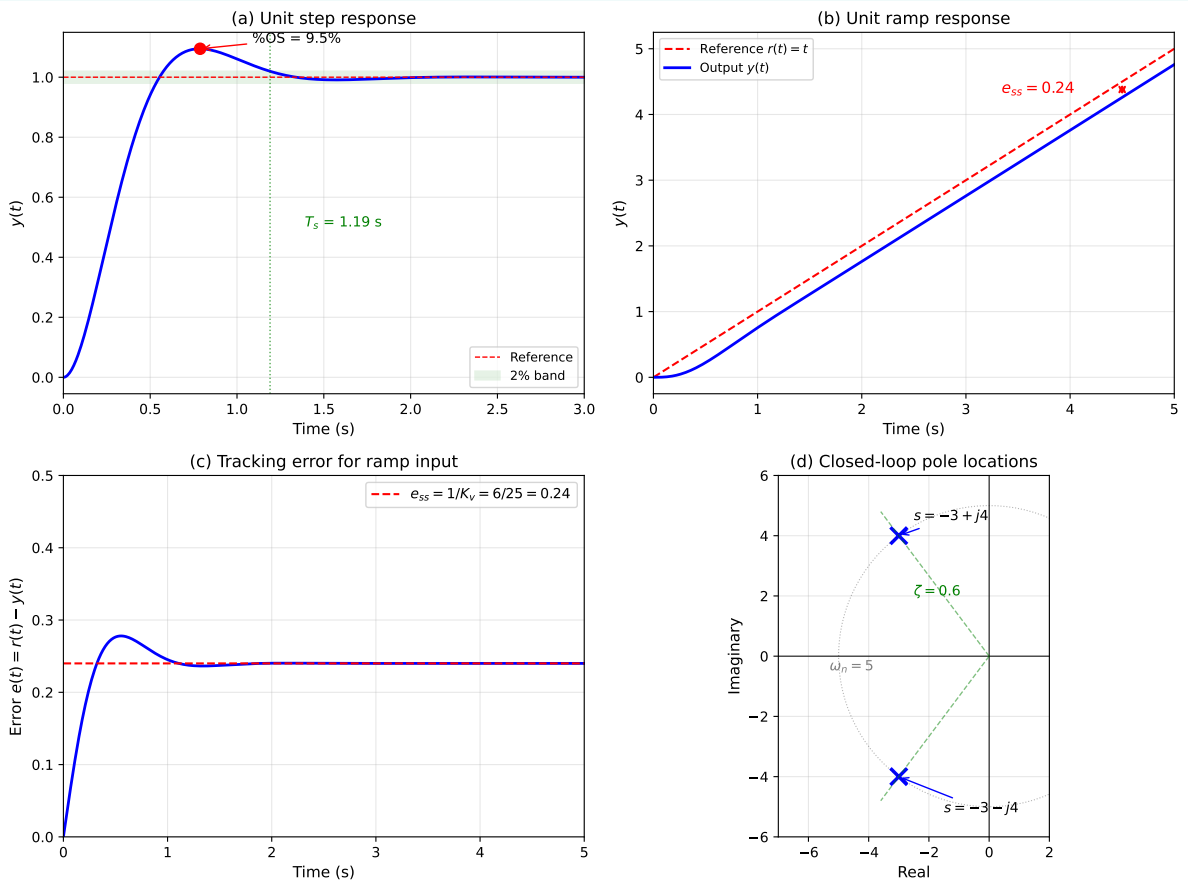


Figure 6.15: MATLAB verification of the rate feedback design: (a) unit step response with 9.5% overshoot, (b) unit ramp tracking showing the steady-state lag, (c) ramp error converging to $e_{ss} = 0.24$, (d) closed-loop poles at $s = -3 \pm j4$.

MATLAB Companion: Analysing First- and Second-Order Systems

Try it yourself

MATLAB can extract all the key time-response specifications automatically. Listing 6.2 creates a first-order and a second-order system, plots their step responses, and reads off rise time, settling time, overshoot, damping ratio, and natural frequency.

```
% --- First-order system: G1 = 1 / (tau*s + 1) ---
tau = 0.5;
G1 = tf(1, [tau 1]);
figure;
step(G1, 5);  title('First-Order Step Response');  grid on;
info1 = stepinfo(G1);
fprintf('First-order: Rise time = %.2f s, Settling time = %.2f s\n', ...
        info1.RiseTime, info1.SettlingTime);

% --- Second-order system: G2 = wn^2 / (s^2 + 2*zeta*wn*s + wn^2) ---
wn = 4;  zeta = 0.3;
```

```
G2 = tf(wn^2, [1  2*zeta*wn  wn^2]);
figure;
step(G2, 5);  title('Second-Order Step Response');  grid on;
info2 = stepinfo(G2);
fprintf('Second-order: Overshoot = %.1f%%, Settling time = %.2f s\n', ...
         info2.Overshoot, info2.SettlingTime);

% --- Extract damping ratio and natural frequency from any system ---
damp(G2)   % prints zeta, wn, and poles in a table
```

Listing 6.2: Extracting time-response specifications in MATLAB.

Key Takeaways

- A first-order system $G(s) = K/(\tau s + 1)$ has rise time $t_r \approx 2.2\tau$ and settling time $t_s \approx 4\tau$ (2% criterion).
- Second-order systems are classified by damping ratio: underdamped ($0 < \zeta < 1$), critically damped ($\zeta = 1$), overdamped ($\zeta > 1$).
- For underdamped systems: $\%OS = 100\, e^{-\zeta\pi/\sqrt{1-\zeta^2}}$, $t_p = \pi/\omega_d$, $t_s \approx 4/(\zeta\omega_n)$.
- Pole locations on the s-plane directly determine response character: real part controls decay rate, imaginary part controls oscillation frequency.
- Constant-ζ lines are rays from the origin; constant-σ lines are vertical; constant-ω_d lines are horizontal.
- Steady-state error depends on both the system type (number of integrators in $G(s)$) and the input signal.
- Each additional integrator in the forward path eliminates SSE for one higher power of s in the input, but makes stability harder to achieve.

End-of-Chapter Exercises

1. A first-order system has the transfer function

$$G(s) = \frac{8}{s+4}.$$

   (a) Identify the time constant τ and the DC gain.

   (b) What is the value of the output at $t = \tau$ when a unit step is applied?

   (c) How long does it take for the output to reach 98% of its final value?

2. A second-order system has the transfer function

$$G(s) = \frac{25}{s^2 + 6s + 25}.$$

   (a) Find the natural frequency ω_n and damping ratio ζ.

   (b) Classify the system as underdamped, critically damped, or overdamped.

   (c) Calculate the percentage overshoot and the approximate settling time (2% criterion).

3. For the standard second-order transfer function

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2},$$

a step response is observed with 16% overshoot and a peak time of $t_p = 0.5\,\text{s}$.

   (a) Determine the damping ratio ζ from the overshoot.

   (b) Determine the natural frequency ω_n from the peak time.

   (c) Write the transfer function with the numerical values of ζ and ω_n.

4. Consider a unity-feedback system with forward transfer function

$$G(s) = \frac{K}{s(s+5)}.$$

   (a) Find the closed-loop transfer function.

   (b) For $K = 20$, determine the damping ratio, natural frequency, overshoot, and settling time of the closed-loop step response.

   (c) Verify your answers using the MATLAB commands `step()` and `stepinfo()`.

5. A unity-feedback system has forward transfer function

$$G(s) = \frac{50}{s+10}.$$

   (a) Compute the steady-state error for a unit step input.

   (b) Compute the steady-state error for a unit ramp input.

   (c) Explain physically why the ramp error is larger (or infinite).

6. A unity negative-feedback system has the open-loop transfer function

$$G(s) = \frac{100}{s(s+4)(s+10)}.$$

   (a) What is the system type?

   (b) Find the steady-state error for a unit step input.

   (c) Find the steady-state error for a unit ramp input.

7. Use MATLAB to investigate the effect of the damping ratio on step response. Write a short script that plots the unit step response of

$$G(s) = \frac{16}{s^2 + 2\zeta(4)s + 16}$$

for $\zeta = 0.1,\ 0.3,\ 0.5,\ 0.7,\ 1.0$ on a single figure. Comment on how overshoot and settling time change as ζ increases.

8. A system has the following step response characteristics: the output reaches a peak of 1.25 at $t = 1\,\text{s}$ and settles to a final value of 1.0.

   (a) Estimate the percentage overshoot.

   (b) Estimate the damping ratio ζ.

   (c) If the response appears to settle (within 2%) by $t = 4\,\text{s}$, estimate the natural frequency ω_n.

Chapter 7

Stability

Learning Objectives

After completing this chapter, you should be able to:

- Classify a continuous-time LTI system as asymptotically stable, marginally stable, or unstable from its pole locations.
- Explain why poles must lie in the open left half-plane for asymptotic stability.
- Construct a Routh array from a characteristic polynomial and count right-half-plane roots.
- Determine the range of a parameter (e.g. gain K) for which a feedback system remains stable using the Routh–Hurwitz criterion.
- Compute eigenvalues of a state matrix A and relate them to system stability.
- Verify stability in MATLAB using `roots()`, `pole()`, `eig()`, and `isstable()`.

Stability is one of the central ideas in control engineering. Before discussing speed of response, overshoot, steady-state error, or robustness, we must first determine whether the system behaves in a physically acceptable manner over time. A system that produces an ever-growing output in response to a bounded command or a small disturbance cannot be regarded as satisfactory, regardless of how attractive its other specifications may appear.

For linear time-invariant systems, stability is answered by the location of the poles of the transfer function, or equivalently by the roots of the Characteristic equation.

7.1 Why Stability Comes First

If the response of a dynamic system grows without bound, it may saturate actuators, excite structural vibration, damage hardware, or simply fail to perform its intended task. Only once stability has been established does it make sense to ask whether the response is sufficiently fast or sufficiently accurate.

The standard decomposition of the output into a forced part and a natural part makes this precise:

$$y(t) = y_{\text{forced}}(t) + y_{\text{natural}}(t).$$

The forced response is associated with the applied input, whereas the natural response is determined by the internal dynamics of the system. The study of stability focuses primarily on the natural response, because it reveals whether the system's own modes decay or grow with time.

Remark 7.1

In continuous-time LTI systems, stability is fundamentally a question about the natural response, and the natural response is determined by the poles of the system.

7.2 Basic Stability Definitions for LTI Systems

Definition 7.1: Asymptotic, Marginal Stability and Instability

A continuous-time LTI system is said to be **asymptotically stable** if every natural response approaches zero as $t \to \infty$.

A continuous-time LTI system is said to be **marginally stable** if every natural response remains bounded for all time, but at least one natural response does not approach zero as $t \to \infty$.

A continuous-time LTI system is said to be **unstable** if at least one natural response becomes unbounded as $t \to \infty$.

Definition 7.2: BIBO Stability

A system is said to be **bounded-input bounded-output** stable if every bounded input produces a bounded output.

Remark 7.2

In introductory control analysis, the pole test is usually presented in terms of the natural response of the transfer function model. This chapter follows that convention. For standard rational LTI systems without problematic cancellations, the familiar left-half-plane pole condition also gives the usual BIBO stability test.

7.3 Transfer Function, Characteristic Equation, and Poles

Consider a continuous-time LTI system with transfer function

$$G(s) = \frac{N(s)}{D(s)},$$

where $N(s)$ and $D(s)$ are polynomials in s. The roots of the denominator polynomial $D(s)$ are called the **poles** of the system. These poles determine the form of the natural response.

In feedback control, it is usually the *closed-loop* poles that matter. If the closed-loop transfer function is written as

$$T(s) = \frac{N_{cl}(s)}{D_{cl}(s)},$$

then the equation

$$D_{cl}(s) = 0$$

is called the **characteristic equation**. The roots of this equation determine the stability of the closed-loop system.

Suppose the denominator factors as

$$D(s) = \prod_{i=1}^{r} (s - p_i)^{m_i},$$

where p_i are the distinct poles and m_i are their multiplicities. The inverse Laplace transform then shows that the natural response is built from terms of the form

$$t^k e^{p_i t}, \qquad k = 0, 1, \ldots, m_i - 1.$$

Therefore, the behaviour of the response depends directly on the location of each pole p_i in the complex plane.

7.4 Why Stability Requires Poles in the Open Left Half-Plane

Let a pole be located at

$$p = \sigma + j\omega.$$

The corresponding modal term in the time response contains the factor

$$e^{pt} = e^{(\sigma + j\omega)t} = e^{\sigma t}\left(\cos \omega t + j \sin \omega t\right).$$

This expression immediately reveals the role of the real part σ:

- if $\sigma < 0$, the exponential factor $e^{\sigma t}$ decays to zero;
- if $\sigma = 0$, the exponential factor remains constant in magnitude;
- if $\sigma > 0$, the exponential factor grows without bound.

Hence, a mode decays only when the corresponding pole lies strictly to the left of the imaginary axis. If even one pole lies on the imaginary axis, that mode does not decay. If a pole lies in the right half-plane, the corresponding mode grows exponentially. This is the reason for the classical stability condition:

$$\Re(p_i) < 0 \qquad \text{for all poles } p_i.$$

The phrase *open left half-plane* is essential. It means that the imaginary axis itself is excluded. A pole exactly on the imaginary axis does not produce exponential decay, so Asymptotic stability is lost.

Result 7.1: Pole Location Test for Continuous-Time LTI Systems

For a continuous-time rational LTI system, the stability classification is determined by the poles of the transfer function as follows:

1. The system is **asymptotically stable** if all poles lie in the open left half-plane.
2. The system is **marginally stable** if there are no poles in the right half-plane, the only poles on the imaginary axis are simple, and all remaining poles lie in the left half-plane.
3. The system is **unstable** if at least one pole lies in the right half-plane, or if there is any repeated pole on the imaginary axis.

7.5 Interpretation of the Different Pole Locations

7.5.1 All poles in the open left half-plane

If every pole satisfies $\Re(p_i) < 0$, then every modal term decays with time. Even if a pole is repeated, the response may contain factors such as te^{-at} or t^2e^{-at}, but these still tend to zero because the exponential decay dominates the polynomial growth. Consequently, the natural response vanishes, and the system is asymptotically stable.

Interpretation 7.1: Physical meaning

Poles in the left half-plane correspond to modes that dissipate energy. The system eventually forgets its initial condition and settles to its forced behaviour.

Example 7.1: Step response of a stable system

Consider the transfer function

$$G(s) = \frac{2}{(s+1)(s+2)}.$$

Since the poles are at $s = -1$ and $s = -2$, both poles lie in the open left half-plane. We now determine the response to a unit step input

$$r(t) = u(t), \qquad R(s) = \frac{1}{s}.$$

Solution:

The output in the Laplace domain is

$$Y(s) = G(s)R(s) = \frac{2}{(s+1)(s+2)} \cdot \frac{1}{s} = \frac{2}{s(s+1)(s+2)}.$$

Using partial fraction expansion,

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

Taking the inverse Laplace transform gives

$$y(t) = 1 - 2e^{-t} + e^{-2t}, \qquad t \geq 0.$$

This expression shows that the step response consists of a constant steady-state term and two decaying exponential terms. As $t \to \infty$,

$$e^{-t} \to 0, \qquad e^{-2t} \to 0,$$

so that

$$y(t) \to 1.$$

The natural part of the response is

$$y_{\text{natural}}(t) = -2e^{-t} + e^{-2t},$$

and it vanishes with time because both poles are in the open left half-plane. Hence the system is asymptotically stable.

7.5.2 Simple poles on the imaginary axis

If a pole lies at $p = j\omega$ or $p = -j\omega$, with no repetition, then the associated response contains sustained sinusoidal terms such as

$$A\cos\omega t + B\sin\omega t.$$

These oscillations remain bounded, but they do not decay. The same idea applies to a simple pole at the origin, which produces a constant mode. In either case, the natural response persists indefinitely, so the system is marginally stable rather than asymptotically stable.

Example 7.2: Step response of a system with simple poles on the imaginary axis

Consider the transfer function

$$G(s) = \frac{1}{s^2 + 4}.$$

The poles are at

$$s = \pm j2,$$

so they are simple poles on the imaginary axis. We determine the response to a unit step input

$$r(t) = u(t), \qquad R(s) = \frac{1}{s}.$$

Solution:

The output in the Laplace domain is

$$Y(s) = G(s)R(s) = \frac{1}{s(s^2 + 4)}.$$

Using partial fractions,

$$\frac{1}{s(s^2+4)} = \frac{1}{4s} - \frac{s}{4(s^2+4)}.$$

Taking the inverse Laplace transform gives

$$y(t) = \frac{1}{4} - \frac{1}{4}\cos(2t), \qquad t \geq 0.$$

This response remains bounded for all time, but it does not settle to a fixed value in the usual decaying sense, because the oscillatory term

$$-\frac{1}{4}\cos(2t)$$

persists indefinitely.

The non-decaying oscillation is a direct consequence of the simple poles at $\pm j2$ on the imaginary axis, so this is a marginally stable case.

Remark 7.3

Marginally stable systems are neither asymptotically stable nor unstable in the strict natural-response sense. Nevertheless, they are generally not desirable in control engineering.

The main reason is that poles on the imaginary axis lie exactly on the boundary between stability and instability. Therefore, even a small modelling error, parameter variation, neglected dynamic effect, or unmodelled disturbance may shift such a pole into the right half-plane. If this happens, the corresponding mode grows with time and the system becomes unstable. For this reason, Marginal stability offers no real robustness margin.

By contrast, if all poles lie strictly in the open left half-plane, then the system is asymptotically stable, and for a standard rational LTI system every bounded input produces a bounded output. In other words, asymptotically stable systems are BIBO stable.

Marginally stable systems do not enjoy this property in general: a bounded input at an imaginary-axis pole frequency can make

$$Y(s) = G(s)U(s)$$

contain repeated imaginary-axis poles, producing terms whose amplitude grows with time.

Example 7.3: Marginally stable system

Consider the transfer function

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

The poles are at

$$s = \pm j,$$

so the system has simple poles on the imaginary axis and is marginally stable in the natural-response sense.

Now apply the bounded input

$$u(t) = \sin t, \qquad t \geq 0.$$

Its Laplace transform is

$$U(s) = \frac{1}{s^2 + 1}.$$

Solution:

The output in the Laplace domain is

$$Y(s) = G(s)U(s) = \frac{1}{s^2 + 1} \cdot \frac{1}{s^2 + 1} = \frac{1}{(s^2 + 1)^2}.$$

Hence the poles of $Y(s)$ are now repeated at $s = \pm j$. Taking the inverse Laplace transform,

$$y(t) = \frac{1}{2} \sin t - \frac{1}{2} t \cos t, \qquad t \geq 0.$$

The first term is bounded, but the second term,

$$-\frac{1}{2} t \cos t,$$

has an amplitude that grows linearly with time. Therefore, although the input $u(t) = \sin t$ is bounded, the output $y(t)$ is unbounded.

7.5.3 Repeated poles on the imaginary axis

A repeated pole on the imaginary axis changes the situation completely. If $p = j\omega$ is repeated, then the natural response includes terms such as

$$te^{j\omega t}, \qquad t^2 e^{j\omega t},$$

whose real and imaginary parts are of the form

$$t \cos \omega t, \qquad t \sin \omega t, \qquad t^2 \cos \omega t, \qquad t^2 \sin \omega t.$$

These terms grow in amplitude with time, even though the oscillation frequency remains fixed. Therefore, the response becomes unbounded, and the system is unstable.

A repeated pole at the origin is a special case of the same phenomenon. For example, a double pole at $s = 0$ produces ramp-type terms, which also become unbounded.

7.5.4 Poles in the right half-plane

If a pole lies in the right half-plane, then its real part is positive. The corresponding mode contains the factor $e^{\sigma t}$ with $\sigma > 0$, so the response grows exponentially. If the pole is complex, the response is an exponentially growing oscillation; if the pole is real, the response grows monotonically. In either case, the system is unstable.

Interpretation 7.2: Why right-half-plane poles are unacceptable

A right-half-plane pole represents an internally amplifying mode. Even if the initial condition is small, that mode grows rather than decays, so the system cannot be regarded as dynamically well behaved.

Example 7.4: Step response of a system with a right-half-plane pole

Consider the transfer function

$$G(s) = \frac{1}{s-2}.$$

The system has a pole at

$$s = 2,$$

which lies in the right half-plane. We determine the response to a unit step input

$$r(t) = u(t), \qquad R(s) = \frac{1}{s}.$$

Solution:

The output in the Laplace domain is

$$Y(s) = G(s)R(s) = \frac{1}{s(s-2)}.$$

Using partial fractions,

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

Taking the inverse Laplace transform gives

$$y(t) = -\frac{1}{2} + \frac{1}{2}e^{2t}, \qquad t \geq 0.$$

The response contains the term

$$\frac{1}{2}e^{2t},$$

and since the exponent is positive, this term grows without bound as $t \to \infty$. Therefore,

$$y(t) \to \infty.$$

The right-half-plane pole therefore makes the system unstable.

7.6 A Useful Summary of the Cases

Stability classification in the s-plane

For a continuous-time LTI system:

- **All poles in the open left half-plane** $\Rightarrow$ asymptotically stable.
- **Simple isolated poles on the imaginary axis, all other poles in the left half-plane** $\Rightarrow$ marginally stable.
- **Any repeated pole on the imaginary axis** $\Rightarrow$ unstable.
- **Any pole in the right half-plane** $\Rightarrow$ unstable.

7.7 Worked Examples

Example 7.5: Classifying systems from their poles

Determine the stability of the following transfer functions:

$$G_1(s) = \frac{4}{(s+2)(s+5)},$$
$$G_2(s) = \frac{3}{s^2+9},$$
$$G_3(s) = \frac{1}{s^2(s+1)},$$
$$G_4(s) = \frac{2}{(s-1)(s+4)}.$$

Solution:

For

$$G_1(s) = \frac{4}{(s+2)(s+5)},$$

the poles are at $s = -2$ and $s = -5$. Both lie in the open left half-plane, so the system is asymptotically stable.

For

$$G_2(s) = \frac{3}{s^2+9},$$

the poles are at $s = \pm j3$. These are simple poles on the imaginary axis. The response remains oscillatory and bounded, but it does not decay. Hence the system is marginally stable.

For

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

there is a repeated pole at the origin and one pole at $s = -1$. Because the pole on the imaginary axis is repeated, the response contains unbounded polynomial terms in time. Therefore, the system is unstable.

For

$$G_4(s) = \frac{2}{(s-1)(s+4)},$$

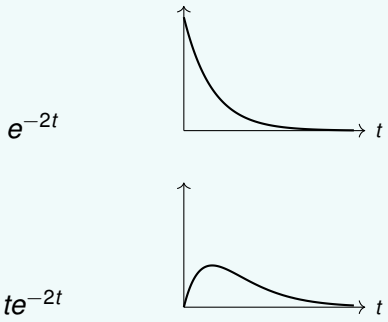
the poles are at $s = 1$ and $s = -4$. Since one pole lies in the right half-plane, the system is unstable.

Example 7.6: Why repeated left-half-plane poles are still acceptable

Consider

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

The system has a repeated pole at $s = -2$. Its natural response contains terms proportional to



Although the factor t grows, the exponential decay dominates, and both terms tend to zero as $t \to \infty$. In general, an exponential decay term dominates any polynomial growth term in a product of the form $t^n e^{-at}$ with $a > 0$. Therefore, repeated left-half-plane poles are still asymptotically stable.

7.8 The Routh–Hurwitz Criterion

For low-order systems, the poles can be found directly by factoring the characteristic polynomial. For higher-order systems, explicit factorisation may be tedious or impossible by hand. The Routh–Hurwitz criterion provides a test that determines how many roots lie in the right half-plane using only the coefficients of the characteristic polynomial, without computing the roots themselves.

The criterion is named after **Edward John Routh** (1876) and **Adolf Hurwitz** (1895). They did not produce the result jointly. Routh developed a recursive tabular procedure for testing stability, whereas Hurwitz developed an equivalent formulation based on determinants. In control engineering, the tabular form is usually preferred for hand calculations, and this is the form referred to as the *Routh criterion* or the *Routh–Hurwitz criterion*.

Instead of calculating all poles of

$$P(s) = a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0,$$

we construct an array from the coefficients and examine the signs of the entries in its first column.

This is useful for three reasons: it allows stability to be checked quickly even for higher-order systems; it tells us *how many* roots lie in the right half-plane, not merely whether the system is stable or unstable; and it is especially valuable in design problems where coefficients depend on a controller parameter such as a gain K, because the Routh array can determine the range of K for which the system remains stable.

Remark 7.4

The Routh–Hurwitz criterion determines the number of right-half-plane roots of the characteristic polynomial directly from its coefficients, without explicitly finding the roots.

Remark 7.5

Before constructing the array, one simple observation is worth making. If all roots of the polynomial are to lie in the open left half-plane, then all coefficients of the characteristic polynomial must be present and must have the same sign. For first- and second-order polynomials this condition is also sufficient. For third-order and higher polynomials, however, it is only necessary. A polynomial may have all positive coefficients and still possess right-half-plane roots, so the Routh array is needed to complete the test.

Result 7.2: Routh–Hurwitz criterion

Consider the characteristic polynomial

$$P(s) = a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0, \qquad a_n > 0.$$

After constructing the Routh array, the number of sign changes in the first column is equal to the number of roots of $P(s)$ that lie in the right half-plane.

Consequently, the system is asymptotically stable if and only if there are no sign changes in the first column and no row of zeros indicating poles on the imaginary axis.

7.8.1 Construction of the Routh array

The Routh array is formed from the coefficients of the characteristic polynomial. The first two rows are written directly from the coefficients by alternating powers of s:

$$\begin{array}{c|cccc} s^n & a_n & a_{n-2} & a_{n-4} & \cdots \\ s^{n-1} & a_{n-1} & a_{n-3} & a_{n-5} & \cdots \end{array}$$

where any missing coefficients are taken as zero.

The remaining rows are generated recursively. If two consecutive rows begin as

$$\begin{array}{c|cccc} s^k & a & b & c & \cdots \\ s^{k-1} & d & e & f & \cdots \end{array}$$

then the next row begins with

$$s^{k-2} \;\Big|\; \frac{d\cdot b - a\cdot e}{d} \quad \frac{d\cdot c - a\cdot f}{d} \quad \cdots$$

and the process continues until the s^0 row is reached.

Once the table has been completed, the first column is inspected.

- If all entries in the first column are positive, there are no right-half-plane roots.
- Each sign change in the first column corresponds to one right-half-plane root.

7.8.2 Special cases

Two special cases can arise during the construction.

1. **A zero appears as the first element of a row, but the row is not entirely zero.** In this case, the zero is replaced temporarily by a small positive number ε, and the sign changes are determined in the limit as $\varepsilon \to 0^+$.
2. **An entire row becomes zero.** This indicates a symmetric root pattern with respect to origin and usually signals the presence of roots on the imaginary axis (If the system is stable). In this case, an **auxiliary polynomial** is formed from the row above the zero row. The zero row is then replaced by the coefficients of the derivative of that auxiliary polynomial with respect to s, after which the Routh process continues.

Important interpretation

The Routh array is not merely a computational trick. A row of zeros is highly informative: it indicates that the polynomial contains a factor with roots symmetrically located about the origin, and in control problems this commonly reveals poles on the imaginary axis.

Example 7.7: Using the Routh array to confirm stability

Consider the characteristic polynomial

$$P(s) = s^3 + 6s^2 + 11s + 6.$$

Determine whether all roots lie in the open left half-plane.

Solution:

The Routh array begins with

$$\begin{array}{c|cc} s^3 & 1 & 11 \\ s^2 & 6 & 6 \end{array}$$

The next row is

$$s^1 \left| \frac{6\cdot 11 - 1\cdot 6}{6} \quad 0 \right. = s^1 \left| \begin{array}{cc} 10 & 0 \end{array} \right.$$

and the final row is

$$\begin{array}{c|cc} s^0 & 6 & 0 \end{array}$$

Hence the full array is

$$\begin{array}{c|cc} s^3 & 1 & 11 \\ s^2 & 6 & 6 \\ s^1 & 10 & 0 \\ s^0 & 6 & 0 \end{array}$$

The first column is

$$1, \quad 6, \quad 10, \quad 6$$

and all entries are positive. Therefore, there are no sign changes, so there are no right-half-plane roots. The polynomial is stable, and the corresponding system is asymptotically stable.

Example 7.8: Finding the range of K for stability

Consider the characteristic polynomial

$$P(s) = s^3 + 3s^2 + 2s + K.$$

Determine the values of K for which the system is asymptotically stable.

Solution:

The first two rows of the Routh array are

$$\begin{array}{c|cc} s^3 & 1 & 2 \\ s^2 & 3 & K \end{array}$$

The next row is

$$s^1 \left| \frac{3\cdot 2 - 1\cdot K}{3} \quad 0 \right. = s^1 \left| \frac{6-K}{3} \quad 0 \right.$$

and the final row is

$$s^0 \mid K \quad 0$$

Thus the first column is

$$\left\{1, \quad 3, \quad \frac{6-K}{3}, \quad K\right\}.$$

For asymptotic stability, every element in the first column must be positive. Therefore,

$$\begin{aligned} K &> 0, \\ \frac{6-K}{3} &> 0. \end{aligned}$$

The second inequality gives $K < 6$. Hence the stable range is

$$0 < K < 6.$$

It is also worth noting that when $K = 6$, the s^1 row begins with zero, which indicates that the system lies on the boundary of stability.

Example 7.9: A Routh array with a zero as the first element of a row

Consider the characteristic polynomial

$$P(s) = s^6 + 2s^5 + 5s^4 + 6s^3 + 7s^2 + 2s + 1.$$

Use the Routh–Hurwitz criterion to determine the number of right-half-plane roots.

Solution:

The first two rows of the Routh array are formed directly from the coefficients:

$$\begin{array}{c|cccc} s^6 & 1 & 5 & 7 & 1 \\ s^5 & 2 & 6 & 2 & 0 \end{array}$$

The s^4 row is

$$s^4 \left| \frac{2\cdot 5 - 1\cdot 6}{2} \quad \frac{2\cdot 7 - 1\cdot 2}{2} \quad \frac{2\cdot 1 - 1\cdot 0}{2} \right. = s^4 \mid 2 \quad 6 \quad 1$$

Now form the s^3 row:

$$s^3 \left| \frac{2\cdot 6 - 2\cdot 6}{2} \quad \frac{2\cdot 2 - 2\cdot 1}{2} \quad 0 \right. = s^3 \mid 0 \quad 1 \quad 0$$

At this stage, the first element of the row is zero, but the row is not entirely zero. This is a special case. We replace the zero temporarily by a small positive number ε, where

$$\varepsilon > 0, \qquad \varepsilon \to 0^+.$$

Hence we write the s^3 row as

$$s^3 \mid \varepsilon \quad 1 \quad 0$$

The next row is then computed in the usual way:

$$s^2 \left| \frac{\varepsilon\cdot 6 - 2\cdot 1}{\varepsilon} \quad \frac{\varepsilon\cdot 1 - 2\cdot 0}{\varepsilon} \quad 0 \right. = s^2 \left| 6 - \frac{2}{\varepsilon} \right. \quad 1 \quad 0$$

The s^1 row becomes

$$s^1 \left| \frac{\left(6 - \frac{2}{\varepsilon}\right)\cdot 1 - \varepsilon\cdot 1}{6 - \frac{2}{\varepsilon}} \quad 0 \right. = s^1 \left| \frac{6 - \frac{2}{\varepsilon} - \varepsilon}{6 - \frac{2}{\varepsilon}} \right. \quad 0$$

and the last row is

$$s^0 \mid 1 \quad 0$$

Thus the first column is

$$1, \quad 2, \quad 2, \quad \varepsilon, \quad 6 - \frac{2}{\varepsilon}, \quad \frac{6 - \frac{2}{\varepsilon} - \varepsilon}{6 - \frac{2}{\varepsilon}}, \quad 1.$$

Now consider the signs as $\varepsilon \to 0^+$:

$$1 > 0, \qquad 2 > 0, \qquad 2 > 0, \qquad \varepsilon > 0,$$

while

$$6 - \frac{2}{\varepsilon} < 0$$

for sufficiently small positive ε, and

$$\frac{6 - \dfrac{2}{\varepsilon} - \varepsilon}{6 - \dfrac{2}{\varepsilon}} > 0.$$

Therefore, the signs in the first column are

$$+, \quad +, \quad +, \quad +, \quad -, \quad +, \quad +.$$

The two sign changes show that the polynomial has

$$\boxed{\text{2 roots in the right half-plane.}}$$

Hence the system is **unstable**.

Example 7.10: Detecting imaginary-axis poles from a row of zeros

Consider the characteristic polynomial

$$P(s) = s^4 + 2s^3 + 2s^2 + 2s + 1.$$

Use the Routh criterion to investigate its stability.

Solution:

The first two rows are

$$\begin{array}{c|ccc} s^4 & 1 & 2 & 1 \\ s^3 & 2 & 2 & 0 \end{array}$$

The next row is

$$s^2 \left| \frac{2\cdot 2 - 1\cdot 2}{2} \quad \frac{2\cdot 1 - 1\cdot 0}{2} \quad 0 \right. = s^2 \left| 1 \quad 1 \quad 0 \right.$$

The next row becomes

$$s^1 \left| \frac{1\cdot 2 - 2\cdot 1}{1} \quad \frac{1\cdot 0 - 2\cdot 0}{1} \quad 0 \right. = s^1 \left| 0 \quad 0 \quad 0 \right.$$

so an entire row of zeros appears.

We therefore form the auxiliary polynomial from the row above, namely

$$A(s) = s^2 + 1.$$

Differentiating with respect to s gives

$$A'(s) = 2s.$$

The zero row is replaced by the coefficients of $A'(s)$, so the array continues as

s^4	1	2	1
s^3	2	2	0
s^2	1	1	0
s^1	2	0	0
s^0	1	0	0

The first column now contains no sign change. Hence there are no right-half-plane roots. However, the row of zeros revealed the presence of imaginary-axis roots. Indeed, the auxiliary polynomial

$$A(s) = s^2 + 1$$

has roots at

$$s = \pm j.$$

These simple imaginary-axis roots make the system marginally stable.

Example 7.11: Finding the range of gain K using the Routh–Hurwitz criterion

For the unity-feedback system shown in Figure 7.1, determine the range of K for which the closed-loop system is stable, unstable, and marginally stable. Assume $K > 0$.

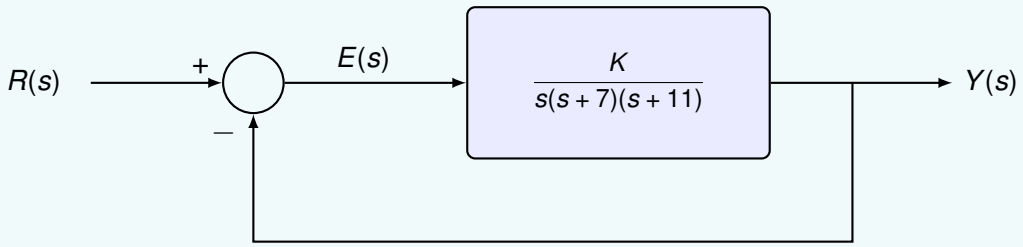


Figure 7.1: Unity-feedback system for gain selection using the Routh–Hurwitz criterion

Solution:

The open-loop transfer function is

$$G(s) = \frac{K}{s(s+7)(s+11)}.$$

Since the feedback is unity and negative, the closed-loop characteristic equation is

$$1 + G(s) = 0.$$

Hence,

$$1 + \frac{K}{s(s+7)(s+11)} = 0.$$

Multiplying through by $s(s+7)(s+11)$ gives

$$s(s+7)(s+11)+K=0.$$

Expanding the polynomial leads to

$$s^3+18s^2+77s+K=0.$$

We now form the Routh array:

$$\begin{array}{c|cc} s^3 & 1 & 77 \\ s^2 & 18 & K \\ s^1 & \dfrac{18\cdot 77-K}{18} & 0 \\ s^0 & K & 0 \end{array} = \begin{array}{c|cc} s^3 & 1 & 77 \\ s^2 & 18 & K \\ s^1 & \dfrac{1386-K}{18} & 0 \\ s^0 & K & 0 \end{array}$$

For asymptotic stability, every entry in the first column must be positive. Therefore,

$$1>0, \qquad 18>0, \qquad \frac{1386-K}{18}>0, \qquad K>0.$$

Since $K>0$ is already assumed, the remaining condition is

$$1386-K>0,$$

which gives

$$K<1386.$$

Therefore, the closed-loop system is stable for

$$\boxed{0<K<1386.}$$

Now consider the boundary case $K=1386$. Then the s^1 row becomes zero, so the system lies on the boundary of stability. Substituting $K=1386$ into the characteristic polynomial gives

$$s^3+18s^2+77s+1386=0.$$

This factors as

$$(s+18)(s^2+77)=0.$$

Hence the poles are

$$s=-18, \qquad s=\pm j\sqrt{77}.$$

The closed-loop system therefore has a simple pair of poles on the imaginary axis, so it is marginally stable when

$$\boxed{K=1386.}$$

Finally, if

$$K>1386,$$

then

$$\frac{1386-K}{18}<0,$$

so the first column contains the sign pattern

$$+, \quad +, \quad -, \quad +,$$

which has two sign changes. Therefore, there are two poles in the right half-plane, and the system is unstable for

$$\boxed{K > 1386.}$$

Thus, the complete answer is

Stable:	$0 < K < 1386,$
Marginally stable:	$K = 1386,$
Unstable:	$K > 1386.$

Example 7.12: Using the Routh test for variation of a system parameter

Consider an industrial servo positioning table that moves a payload along a rail. The feedback controller is fixed after tuning, but the payload mass varies from job to job. As the payload becomes heavier, the dynamics of the plant change.

A simplified normalised closed-loop characteristic equation for the position loop is

$$P(s) = s^3 + (4 + m)s^2 + (6 - m)s + 2,$$

where $m > 0$ is an effective payload parameter.

Use the Routh–Hurwitz criterion to determine the range of m for which the closed-loop system is stable. Explain what happens when this range is exceeded.

Solution:

The characteristic polynomial is

$$P(s) = s^3 + (4 + m)s^2 + (6 - m)s + 2.$$

We now form the Routh array:

$$\begin{array}{c|cc} s^3 & 1 & 6 - m \\ s^2 & 4 + m & 2 \\ s^1 & \dfrac{(4 + m)(6 - m) - 2}{4 + m} & 0 \\ s^0 & 2 & 0 \end{array}$$

Simplifying the s^1 entry gives

$$\frac{(4 + m)(6 - m) - 2}{4 + m} = \frac{24 + 2m - m^2 - 2}{4 + m} = \frac{22 + 2m - m^2}{4 + m}.$$

Hence the first column is

$$1, \qquad 4+m, \qquad \frac{22+2m-m^2}{4+m}, \qquad 2.$$

For asymptotic stability, all entries in the first column must be positive. Therefore,

$$1>0, \qquad 4+m>0, \qquad \frac{22+2m-m^2}{4+m}>0, \qquad 2>0.$$

Since $m>0$, the terms 1, $4+m$, and 2 are automatically positive. The only nontrivial condition is

$$22+2m-m^2>0.$$

Rearranging,

$$m^2-2m-22<0.$$

The roots of the quadratic equation

$$m^2-2m-22=0$$

are

$$m=1\pm\sqrt{23}.$$

Therefore,

$$1-\sqrt{23}<m<1+\sqrt{23}.$$

Since $m>0$, the admissible range is

$$\boxed{0<m<1+\sqrt{23}.}$$

Numerically,

$$1+\sqrt{23}\approx 5.80.$$

Hence the closed-loop system is **stable** for

$$\boxed{0<m<1+\sqrt{23}\approx 5.80.}$$

Boundary of stability. When

$$m=1+\sqrt{23},$$

the s^1 row becomes zero, so the system is on the boundary of stability. At this value,

$$P(s)=s^3+(5+\sqrt{23})s^2+(5-\sqrt{23})s+2.$$

Because

$$(5+\sqrt{23})(5-\sqrt{23})=2,$$

the polynomial factors as

$$P(s)=\left(s+5+\sqrt{23}\right)\left(s^2+5-\sqrt{23}\right).$$

Thus the poles are

$$s=-(5+\sqrt{23}), \qquad s=\pm j\sqrt{5-\sqrt{23}}\,.$$

So the system has a simple pair of poles on the imaginary axis and is therefore **marginally stable** at

$$\boxed{m = 1 + \sqrt{23}.}$$

When the payload exceeds the safe range. If

$$m > 1 + \sqrt{23},$$

then

$$\frac{22 + 2m - m^2}{4 + m} < 0,$$

so the first column has the sign pattern

$$+, \quad +, \quad -, \quad +.$$

This gives **two sign changes**, so the system is **unstable** for

$$\boxed{m > 1 + \sqrt{23}.}$$

Physical interpretation. This example represents a practical limitation of a fixed controller. The table may behave well for light and moderate payloads, but if the payload becomes too heavy, the closed-loop dynamics change enough to destroy stability. In practice, the motion would become increasingly oscillatory, and beyond the stability limit the table would fail to settle at the commanded position. This is why control systems are usually designed not only for nominal parameters, but also for the expected range of plant variation.

Example 7.13: Maximum allowable measurement delay for stability

Consider the standard negative-feedback system in Figure 7.2

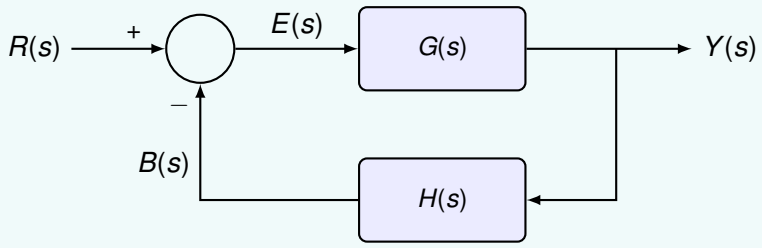


Figure 7.2: Standard feedback system showing the reference $R(s)$, error $E(s)$, output $Y(s)$, and feedback signal $B(s)$

where

$$G(s) = \frac{1}{s(s+1)}, \qquad H(s) = e^{-Ts},$$

where $T > 0$ is a time delay in the feedback path. The delay may represent sensing, communication, or computation lag.

Using the first two terms of the Taylor expansion of the delay,

$$e^{-Ts} \approx 1 - Ts,$$

find an approximate value of the maximum allowable delay T for stability. For this value of T, determine the frequency of the sustained oscillations. Finally, confirm the result in MATLAB (Listing 3).

Solution:

For a standard negative-feedback system, the closed-loop characteristic equation is

$$1 + G(s)H(s) = 0.$$

Substituting the given transfer functions gives

$$1 + \frac{e^{-Ts}}{s(s+1)} = 0.$$

Multiplying through by $s(s+1)$,

$$s(s+1) + e^{-Ts} = 0.$$

Using the first-order Taylor approximation

$$e^{-Ts} \approx 1 - Ts,$$

we obtain the approximate characteristic equation

$$s(s+1) + 1 - Ts = 0.$$

Expanding,

$$s^2 + s + 1 - Ts = 0,$$

so

$$s^2 + (1 - T)s + 1 = 0. \tag{7.1}$$

We now apply the Routh–Hurwitz criterion. The Routh array for is

$$\begin{array}{c|cc} s^2 & 1 & 1 \\ s^1 & 1-T & 0 \\ s^0 & 1 & 0 \end{array}$$

For stability, all entries in the first column must be positive. Therefore,

$$1 > 0, \qquad 1 - T > 0, \qquad 1 > 0.$$

Hence the approximate stability condition is

$$T < 1.$$

Therefore, the **maximum allowable delay** predicted by the linear approximation is

$$\boxed{T_{\max} \approx 1 \text{ s.}}$$

At the boundary of stability, $T = 1$. Substituting this into (7.1) gives

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

Hence the poles are

$$s = \pm j.$$

These poles lie on the imaginary axis, so the system is marginally stable and exhibits sustained oscillation. The angular frequency of oscillation is therefore

$$\boxed{\omega = 1 \text{ rad/s}.}$$

The corresponding oscillation period is

$$\boxed{T_{\text{osc}} = \frac{2\pi}{\omega} = 2\pi \text{ s}.}$$

Listing 3 MATLAB verification of the delay-margin example

```
clc; clear; close all;

s = tf('s');
G = 1/(s*(s+1));

% Approximate critical delay from Taylor expansion
Tcrit_approx = 1;

% Approximate feedback path: exp(-Ts) ~ 1 - Ts
Hlin = 1 - Tcrit_approx*s;

% Closed-loop system using the approximate delay model
Tlin = feedback(G,Hlin);

disp('Approximate closed-loop transfer function:')
Tlin

disp('Poles of the approximate closed-loop system:')
pole(Tlin)

figure;
step(Tlin,25);
grid on;
title('Approximate model at T = 1 s');
xlabel('Time (s)');
ylabel('Output');

% Exact delayed system for comparison
Hexact = tf(1,1,'InputDelay',Tcrit_approx);
Texact = feedback(G,Hexact);
M = allmargin(G*Hexact);

figure;
step(Texact,40);
grid on;
title('Exact delayed system at T = 1 s');
xlabel('Time (s)');
ylabel('Output');

fprintf('Approximate delay margin from Taylor model = %.4f s\n', Tcrit_approx);
fprintf('Exact delay margin from MATLAB            = %.4f s\n', M.DelayMargin(1))
;
fprintf('Exact oscillation frequency               = %.4f rad/s\n', M.DMFrequency
(1));

```

7.9 Stability of State-Space Models

Many practical systems are more conveniently described in **state-space form** than by a transfer function. A continuous-time LTI system in state-space form is written as

$$\dot{x}(t) = Ax(t) + Bu(t),$$
$$y(t) = Cx(t) + Du(t),$$

where $x(t) \in \mathbb{R}^n$ is the state vector, $u(t)$ is the input, $y(t)$ is the output, and A, B, C, D are constant matrices of appropriate dimensions.

The stability of the state-space model is determined by the eigenvalues of the system matrix A.

7.9.1 Why the matrix A determines stability

Consider the unforced system

$$\dot{x}(t) = Ax(t).$$

Its solution is

$$x(t) = e^{At}x(0),$$

where e^{At} is the matrix exponential. The behaviour of $x(t)$ therefore depends on how the matrix exponential evolves with time, and this in turn is determined by the eigenvalues of A.

If all eigenvalues of A have negative real part, then every natural mode decays with time, and the state tends to zero. If any eigenvalue has positive real part, then at least one mode grows exponentially, and the system is unstable. If eigenvalues lie on the imaginary axis, then marginal or unstable behaviour may arise depending on whether those modes are repeated.

Definition 7.3: Asymptotic stability of a state-space model

The continuous-time LTI system

$$\dot{x}(t) = Ax(t) + Bu(t)$$

is said to be **asymptotically stable** if the state of the unforced system

$$\dot{x}(t) = Ax(t)$$

satisfies

$$x(t) \to 0 \qquad \text{as } t \to \infty$$

for every initial condition $x(0)$.

7.9.2 Eigenvalues and the characteristic equation

Let λ be an eigenvalue of the matrix A. By definition, λ satisfies

$$Av = \lambda v$$

for some nonzero vector v, called an eigenvector. Rearranging gives

$$(\lambda I - A)v = 0.$$

For a nonzero vector v to exist, the matrix $\lambda I - A$ must be singular, so

$$\det(\lambda I - A) = 0.$$

In control engineering, it is common to use the variable s instead of λ, so the characteristic equation is written as

$$\det(sI - A) = 0.$$

The roots of this equation are the eigenvalues of A.

Result 7.3: State-space stability test

A continuous-time LTI system

$$\dot{x}(t) = Ax(t) + Bu(t)$$

is asymptotically stable if and only if all eigenvalues of A lie in the open left half-plane, that is,

$$\Re(\lambda_i) < 0 \qquad \text{for all eigenvalues } \lambda_i \text{ of } A.$$

7.9.3 Connection with transfer-function poles

The poles of the transfer function are precisely the roots of

$$\det(sI - A) = 0,$$

provided the state-space realisation is minimal. The transfer function of the state-space model is

$$G(s) = C(sI - A)^{-1}B + D.$$

The inverse $(sI - A)^{-1}$ exists only when $\det(sI - A) \neq 0$. Therefore, the values of s for which $\det(sI - A) = 0$ are exactly the singularities of the transfer function, that is, the poles.

Remark 7.6

For a minimal realisation, the eigenvalues of the matrix A are exactly the poles of the transfer function. Thus, studying the eigenvalues of A is equivalent to studying the poles of the system.

7.9.4 How to find eigenvalues by hand

For small systems, the eigenvalues can be found directly from the characteristic equation

$$\det(sI - A) = 0.$$

Example 7.14: Finding eigenvalues from $\det(sI - A) = 0$

Consider the system matrix

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}.$$

To determine stability, we form

$$sI - A = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} = \begin{bmatrix} s & -1 \\ 2 & s+3 \end{bmatrix}.$$

Hence,

$$\det(sI - A) = \begin{vmatrix} s & -1 \\ 2 & s+3 \end{vmatrix} = s(s+3) + 2.$$

Therefore, the characteristic equation is

$$s^2 + 3s + 2 = 0.$$

Factoring gives

$$(s+1)(s+2) = 0,$$

so the eigenvalues are

$$\lambda_1 = -1, \qquad \lambda_2 = -2.$$

Both eigenvalues lie in the open left half-plane. Therefore, the state-space model is asymptotically stable.

Example 7.15: A state-space model with an unstable eigenvalue

Consider

$$A = \begin{bmatrix} 0 & 1 \\ 2 & -1 \end{bmatrix}.$$

Then

$$sI - A = \begin{bmatrix} s & -1 \\ -2 & s+1 \end{bmatrix},$$

and

$$\det(sI - A) = \begin{vmatrix} s & -1 \\ -2 & s+1 \end{vmatrix} = s(s+1) - 2.$$

Thus the characteristic equation is

$$s^2 + s - 2 = 0.$$

Factoring gives

$$(s+2)(s-1) = 0.$$

Hence the eigenvalues are

$$\lambda_1 = -2, \qquad \lambda_2 = 1.$$

Since one eigenvalue lies in the right half-plane, the system is unstable.

7.9.5 Finding eigenvalues in MATLAB

In practice, eigenvalues are usually computed numerically. MATLAB provides a direct command for this. If the state matrix is A, then the eigenvalues are found using

```
eig(A)
```

This command returns all eigenvalues of the matrix A, as shown in Listing 4.

Listing 4 Finding eigenvalues of a state matrix in MATLAB

```
A = [0 1;
-2 -3];

lambda = eig(A)

```

The output is

$$\lambda = \begin{bmatrix} -1 \\ -2 \end{bmatrix}.$$

Since both eigenvalues are negative, the system is asymptotically stable.

If a full state-space model is defined in MATLAB, the poles can also be obtained directly using the `pole` command (Listing 5).

Listing 5 Finding poles from a state-space model in MATLAB

```
A = [0 1;
-2 -3];
B = [0; 1];
C = [1 0];
D = 0;

sys = ss(A,B,C,D);

eig_A = eig(A)
p = pole(sys)

```

For a minimal realisation, the vectors `eig(A)` and `pole(sys)` are identical.

Example 7.16: A marginal stability case in MATLAB

What does MATLAB return when eigenvalues lie on the imaginary axis? See Listing 6.

Listing 6 MATLAB example with imaginary-axis eigenvalues

```
A = [0 1;
-4 0];

lambda = eig(A)

```

The characteristic equation is

$$\det(sI - A) = s^2 + 4 = 0,$$

so the eigenvalues are

$$\lambda = \pm 2j.$$

These imaginary-axis eigenvalues make the system marginally stable in the natural-response sense.

When a system is given in state-space form, its stability can be checked systematically as follows:

1. write down the state matrix A;
2. form the characteristic equation $\det(sI - A) = 0$;
3. solve for the eigenvalues of A;

4. inspect the real parts of the eigenvalues.

If all eigenvalues have negative real part, the system is asymptotically stable. If any eigenvalue has positive real part, the system is unstable. If eigenvalues lie on the imaginary axis, the system lies on the boundary of stability.

Stability of state-space models

For a continuous-time LTI state-space model, stability is determined by the eigenvalues of the system matrix A. These eigenvalues are found from the characteristic equation

$$\det(sI - A) = 0.$$

For a minimal realisation, they are exactly the poles of the transfer function. Therefore, the pole-based and state-space stability tests are completely consistent.

In practice, the eigenvalues can be found by hand for small matrices or numerically in MATLAB using the command `eig(A)`. This makes state-space stability analysis both conceptually clear and computationally straightforward.

MATLAB Companion: Checking Stability

Try it yourself

MATLAB offers several quick ways to check whether a system is stable. The script in Listing 7.1 demonstrates the most useful commands: `roots()`, `pole()`, `eig()`, and `isstable()`.

```
% --- Method 1: Find poles from a characteristic polynomial ---
% Example: s^3 + 6s^2 + 11s + 6
p = roots([1 6 11 6])        % returns [-3; -2; -1] => stable

% --- Method 2: Find poles from a transfer function ---
G = tf(2, [1 -1 4]);         % G(s) = 2 / (s^2 - s + 4)
pole(G)                       % shows the poles directly

% --- Method 3: Find eigenvalues from a state matrix ---
A = [0 1; -2 -3];
eig(A)                        % returns [-1; -2] => stable

% --- Method 4: One-line stability check ---
sys = ss(A, [0;1], [1 0], 0);
isstable(sys)                 % returns 1 (true) if stable, 0 if not
```

Listing 7.1: MATLAB methods for checking stability.

Key Takeaways

- A continuous-time LTI system is asymptotically stable if and only if all poles lie in the open left half-plane ($\Re(p_i) < 0$ for all i).
- Simple poles on the imaginary axis give marginal stability (bounded but non-decaying oscillations); repeated imaginary-axis poles or any right-half-plane pole means instability.

- The Routh–Hurwitz criterion counts right-half-plane roots from the polynomial coefficients alone — no root-finding needed.
- The Routh array is especially powerful for finding the range of a design parameter (e.g. gain K) that keeps the system stable.
- A row of zeros in the Routh array signals poles symmetrically placed about the origin, often indicating imaginary-axis roots.
- For state-space models, stability is determined by the eigenvalues of the system matrix A, found from $\det(sI - A) = 0$. For a minimal realisation, these are exactly the transfer-function poles.
- MATLAB commands `roots()`, `pole()`, `eig()`, and `isstable()` provide quick numerical verification.

End-of-Chapter Exercises

1. A system has the characteristic equation

$$s^3 + 4s^2 + 5s + 2 = 0.$$

   (a) Find the roots (poles) by factoring or using MATLAB `roots()`.

   (b) State whether the system is stable, unstable, or marginally stable, and explain why.

2. For each of the following sets of pole locations, classify the system as asymptotically stable, marginally stable, or unstable. Briefly justify each answer.

   (a) $s = -1,\ s = -3$

   (b) $s = -2 \pm j3$

   (c) $s = +1,\ s = -5$

   (d) $s = \pm j2$

   (e) $s = -1,\ s = -1,\ s = 0$

3. Construct the Routh array for the characteristic equation

$$s^3 + 3s^2 + 3s + 1 = 0.$$

   Determine the number of roots in the right-half plane. Is the system stable?

4. A unity negative-feedback system has the open-loop transfer function

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

   (a) Write the closed-loop characteristic equation.

   (b) Construct the Routh array in terms of K.

   (c) Find the range of $K > 0$ for which the closed-loop system is stable.

5. Consider the characteristic equation

$$s^4 + 2s^3 + 3s^2 + 2s + 1 = 0.$$

   (a) Build the Routh array. You will encounter a special case (row of zeros). Show how to handle it.

(b) How many roots are in the right-half plane?

(c) Verify your result using `roots([1 2 3 2 1])` in MATLAB.

6. A system has the characteristic equation

$$s^3 + 2s^2 + s + K = 0.$$

(a) Use the Routh criterion to find the maximum value of K for stability.

(b) For K equal to that maximum value, find the frequency of the sustained oscillation.

(c) Verify with MATLAB by computing `roots([1 2 1 K])` for your critical K.

7. A state-space model has the system matrix

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix}.$$

(a) Compute the eigenvalues of A by hand (hint: expand $\det(sI - A)$ and factor).

(b) Is the system stable?

(c) Verify using `eig(A)` in MATLAB.

8. Use MATLAB to check the stability of the system with transfer function

$$G(s) = \frac{10(s+1)}{s^4 + 3s^3 + 5s^2 + 4s + 2}.$$

Write a short script that creates the transfer function, finds its poles, and uses `isstable()` to confirm. Are any poles close to the imaginary axis?

Chapter 8

PID Control

Learning Objectives

After completing this chapter, you should be able to:

- Explain the physical meaning of proportional, integral, and derivative control actions.
- Derive the effect of each PID term on tracking error and disturbance rejection.
- Predict qualitatively how changing each PID gain affects the closed-loop response.
- Design a PID controller using Ziegler–Nichols tuning rules (reaction curve and sustained oscillation methods).
- Explain Åström's relay-feedback auto-tuning method and compute K_u and P_u from relay data.
- Implement and tune PID controllers in MATLAB and Simulink using `pidtune()`.
- Recognise when PID control is insufficient and state-space methods are needed.

> *"More than 90% of all control loops in industry use PID controllers. Understanding PID is not optional — it is the foundation of practical control engineering."*

The Proportional–Integral–Derivative (PID controller) controller is by far the most widely used controller in industry. From regulating the temperature of a chemical reactor to controlling the speed of an electric motor, from maintaining water level in a tank to positioning a robot arm — PID controllers are everywhere. Their popularity comes from simplicity, effectiveness, and the fact that they can be tuned without a detailed mathematical model of the plant.

Mass-produced electronic PID controllers have been accessible since the 1940s, initially as analogue op-amp circuits. Starting in the 1980s, microprocessor-based digital technology emerged, leading to Distributed Control Systems (DCS) and SCADA systems. Today, PID controllers appear in diverse configurations across all industries.

8.1 The Idea Behind PID Control

Consider the everyday task of driving a car at a desired speed on a motorway. You look at the speedometer (measurement), compare the current speed with the speed limit (reference), and adjust the accelerator pedal (control action). How you adjust the pedal depends on several factors:

- **How far are you from the desired speed right now?** If you are 20 mph below the limit, you press the pedal harder than if you are only 2 mph below. This is Proportional action.
- **How long have you been below the desired speed?** If you have been consistently slow for a long time (perhaps going uphill), you press the pedal even more. This is Integral action — it accumulates past errors.
- **How fast is the error changing?** If your speed is rising quickly towards the target, you ease off the pedal slightly to avoid overshooting. This is Derivative action — it anticipates the future.

A PID controller does exactly this, automatically and continuously.

Remark 8.1

The PID controller combines three actions:

- **P (Proportional):** Reacts to the *present* error.
- **I (Integral):** Reacts to the *accumulated past* error.
- **D (Derivative):** Reacts to the *rate of change* (prediction of future) error.

8.2 The PID Controller — Mathematical Formulation

Let $r(t)$ be the reference (desired output) and $y(t)$ be the measured output. The tracking error is

$$e(t) = r(t) - y(t).$$

The PID controller generates a control signal $u(t)$ given by:

Definition 8.1: PID Controller (Time Domain — Textbook Form)

$$u(t) = K_p\, e(t) \;+\; K_I \int_0^t e(\tau)\, d\tau \;+\; K_D \frac{de(t)}{dt}$$

where K_p is the proportional gain, K_I is the integral gain, and K_D is the derivative gain.

Taking the Laplace transform (with zero initial conditions), the PID transfer function is:

Definition 8.2: PID Controller (Transfer Function — Textbook Form)

$$C(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_I}{s} + K_D s$$

8.2.1 Industrial (Standard) Form

An alternative and very common parameterisation used in industry is obtained by factoring out K_p:

$$u(t) = K_p \left(e(t) + \frac{1}{\tau_I} \int_0^t e(\tau)\, d\tau + \tau_D \frac{de(t)}{dt} \right)$$

In the Laplace domain this becomes:

$$C(s) = K_p \left(1 + \frac{1}{\tau_I s} + \tau_D s \right)$$

where $\tau_I = K_p/K_I$ is the *integral time constant* and $\tau_D = K_D/K_p$ is the *derivative time constant*.

Remark 8.2

The two parameterisations are completely equivalent:

$$K_I = \frac{K_p}{\tau_I}, \qquad K_D = K_p \tau_D.$$

The textbook form is convenient for analysis; the industrial form is what you will see on real controller hardware and in tuning tables.

8.3 Analogue Realisation of PID Using Operational Amplifiers

Before the digital era, PID controllers were built entirely from analogue electronic circuits. Understanding these realisations provides physical insight into what each term does.

8.3.1 Proportional Controller

A proportional controller is simply an inverting amplifier:

The transfer function is $V_{out}(s)/V_{in}(s) = -R_2/R_1$. The negative sign can be removed by cascading with a unity-gain inverter.

8.3.2 Integral Controller

Replacing the feedback resistor with a capacitor gives an integrator:

In the time domain: $V_{out}(t) = -\frac{1}{RC} \int V_{in}(t)\, dt$. In the Laplace domain: $V_{out}(s) = -\frac{K_I}{s} V_{in}(s)$ where $K_I = 1/(RC)$.

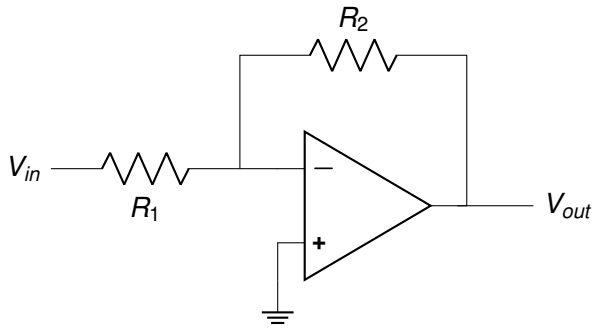


Figure 8.1: Proportional controller realised with an operational amplifier. The gain is $V_{out}/V_{in} = -R_2/R_1 = -K_p$.

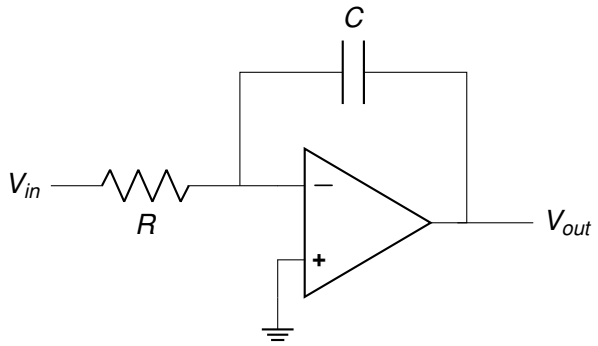


Figure 8.2: Integrator controller using an operational amplifier. The transfer function is $V_{out}(s)/V_{in}(s) = -1/(RCs) = -K_I/s$.

8.3.3 Derivative Controller

Swapping the resistor and capacitor positions gives a differentiator:

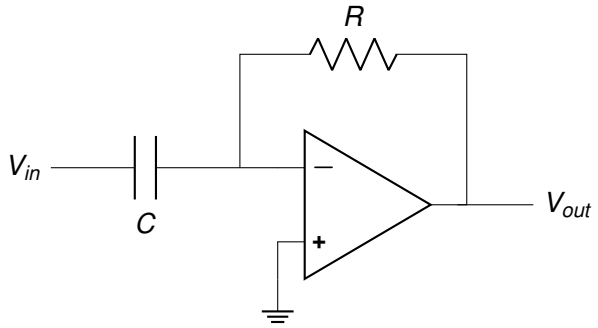


Figure 8.3: Derivative controller using an operational amplifier. The transfer function is $V_{out}(s)/V_{in}(s) = -RCs = -K_D s$.

In the time domain: $V_{out}(t) = -RC\,\dfrac{dV_{in}(t)}{dt}$. In the Laplace domain: $V_{out}(s) = -K_D s\, V_{in}(s)$ where $K_D = RC$.

Remark 8.3

A complete analogue PID controller is constructed by connecting the three op-amp circuits (P, I, D) in parallel, summing their outputs. Each term takes the error signal $e(t)$ as input and generates its own control contribution.

8.4 PID Controller in a Feedback Loop

The general structure of a feedback control system with a PID controller is shown in Figure 8.4. The controller $C(s)$ acts on the error signal $E(s) = R(s) - Y(s)$ to produce the control signal $U(s)$, which drives the plant $G(s)$.

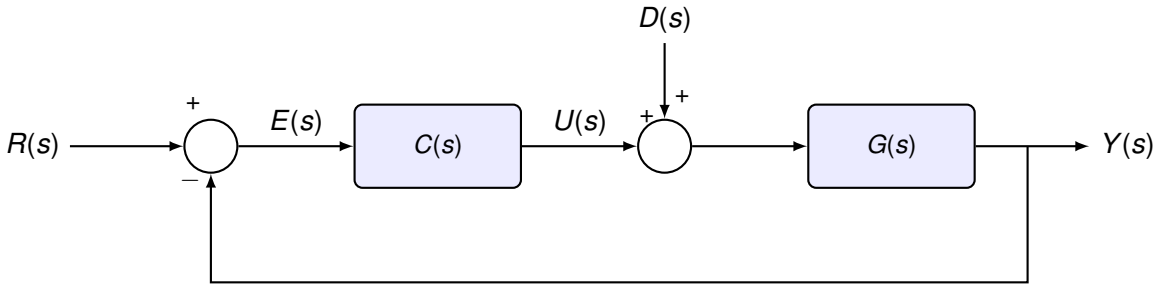


Figure 8.4: Feedback control system with controller $C(s)$, plant $G(s)$, reference $R(s)$, and disturbance $D(s)$.

From Figure 8.4, using the superposition principle (since the system is linear), the output can be written as:

$$Y(s) = \frac{G(s)C(s)}{1 + G(s)C(s)}\,R(s) + \frac{G(s)}{1 + G(s)C(s)}\,D(s) \tag{8.1}$$

The tracking error is:

$$E(s) = R(s) - Y(s) = \frac{1}{1 + G(s)C(s)}\,R(s) - \frac{G(s)}{1 + G(s)C(s)}\,D(s)$$

8.5 Effects of P, I, and D Actions on Performance

8.5.1 Proportional Control

Let $C(s) = K_p$ and consider the plant $G(s) = \dfrac{1}{s+2}$. From (8.1):

$$Y(s) = \frac{K_p}{s + 2 + K_p}\,R(s) + \frac{1}{s + 2 + K_p}\,D(s)$$

The tracking error becomes:

$$E(s) = \frac{s+2}{s + 2 + K_p}\,R(s) - \frac{1}{s + 2 + K_p}\,D(s)$$

For a unit step reference $R(s) = 1/s$ and a step disturbance $D(s) = d/s$, the final value theorem gives:

$$e_{ss} = \lim_{s \to 0} sE(s) = \frac{2}{2 + K_p} - \frac{d}{2 + K_p}$$

Interpretation 8.1: What does this tell us?

- As K_p increases, both the tracking error and the effect of the disturbance decrease.
- However, the tracking error *cannot be reduced to zero*. Even with $K_p = 100$, the error is $2/102 \approx 2\%$.
- Using very high K_p is impractical: it produces large control signals that may saturate actuators.

We can also rewrite the error as:

$$E(s) = \frac{\frac{s+2}{2+K_p}}{\frac{1}{2+K_p}s + 1} R(s) - \frac{\frac{1}{2+K_p}}{\frac{1}{2+K_p}s + 1} D(s)$$

Each component is a first-order transfer function with time constant $\tau_{cl} = \dfrac{1}{2 + K_p}$. As K_p increases, this time constant decreases, meaning faster response (shorter rise time and settling time).

Example 8.1: Proportional Control — Effect of Gain on Step Response

Consider $G(s) = \dfrac{1}{s+2}$ with $C(s) = K_p$ in a unity-feedback loop. Compare the step responses for $K_p = 1, 5, 20$.

Solution:

The closed-loop transfer function is:

$$T(s) = \frac{K_p}{s + 2 + K_p}$$

	$K_p = 1$	$K_p = 5$	$K_p = 20$
Closed-loop pole	-3	-7	-22
Time constant τ_{cl}	0.33 s	0.14 s	0.045 s
DC gain	$1/3$	$5/7$	$20/22$
Steady-state error	67%	29%	9%

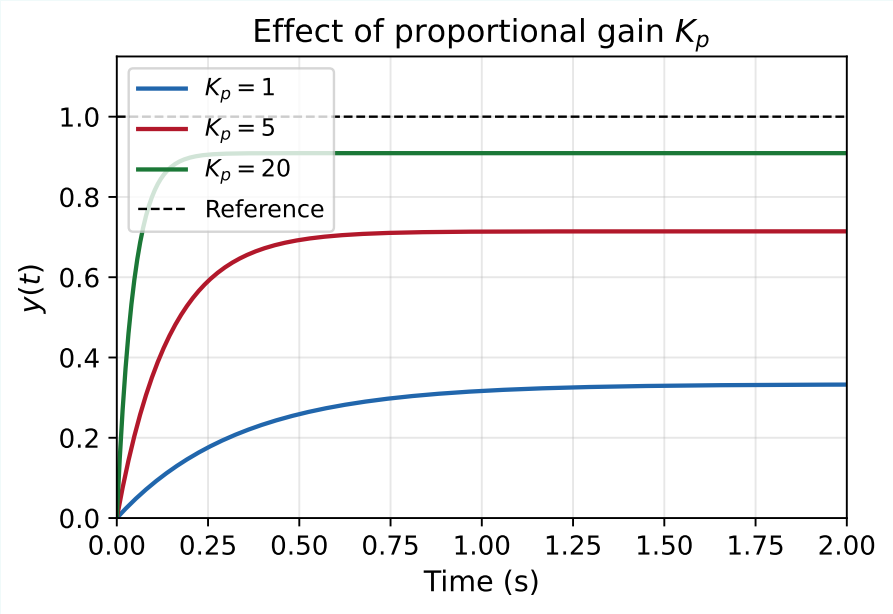


Figure 8.5: Effect of proportional gain on step response. Increasing K_p makes the response faster and reduces steady-state error, but the error never reaches zero.

The MATLAB code in Listing 8.1 generates this comparison.

```
G = tf(1, [1 2]);
Kp_values = [1, 5, 20];
figure; hold on;
for Kp = Kp_values
    T = feedback(Kp*G, 1);
    step(T, 2);
end
yline(1, '--k', 'Reference');
legend('Kp=1','Kp=5','Kp=20','Reference');
title('Effect of proportional gain');
xlabel('Time (s)'); ylabel('y(t)');
grid on;

```

Listing 8.1: Step response comparison for proportional control

Remark 8.4

Proportional control:

- Faster response with higher gain (pole moves further left).
- Reduces steady-state error but *cannot eliminate it* for type-0 systems.
- Reduces the effect of disturbances.
- Too much gain may saturate actuators or destabilise higher-order systems.

8.5.2 Integral Control

Now consider a pure integral controller $C(s) = K_I/s$ applied to the same plant $G(s) = 1/(s+2)$. The output becomes:

$$Y(s) = \frac{K_I}{s^2 + 2s + K_I} R(s) + \frac{s}{s^2 + 2s + K_I} D(s)$$

The error signal is:

$$E(s) = R(s) - Y(s) = \frac{s^2 + 2s}{s^2 + 2s + K_I} \frac{1}{s} - \frac{s}{s^2 + 2s + K_I} \frac{d}{s}$$

When $d = 0$ and $R(s) = 1/s$:

$$e_{ss} = \lim_{s \to 0} sE(s) = \lim_{s \to 0} \frac{s^2 + 2s}{s^2 + 2s + K_I} = \frac{0}{K_I} = 0$$

Interpretation 8.2: Why integral action eliminates steady-state error

The integrator adds a pole at $s = 0$ to the forward path. This increases the system type by one, which is precisely what is needed to track a step input with zero steady-state error. Physically, even if the error is very small, the integral keeps accumulating until the error is driven to exactly zero.

However, examine the characteristic equation: $s^2 + 2s + K_I = 0$. Comparing with the standard second-order form $s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$:

$$\omega_n = \sqrt{K_I}, \qquad 2\zeta\omega_n = 2 \quad \Rightarrow \quad \zeta = \frac{1}{\sqrt{K_I}}$$

The price of integral action

As K_I increases:

- ω_n increases (faster response).
- ζ *decreases* (more oscillatory, larger overshoot).
- The system can become unstable if ζ drops too low in higher-order systems.

This is why pure integral control is rarely used alone — it should be combined with proportional action.

Example 8.2: Integral Control — Eliminating Steady-State Error

Using $G(s) = 1/(s+2)$ and $C(s) = K_I/s$, plot the step response for $K_I = 1, 4, 10$.

Solution:

The closed-loop transfer function is:

$$T(s) = \frac{K_I}{s^2 + 2s + K_I}$$

	$K_I = 1$	$K_I = 4$	$K_I = 10$
ω_n	1	2	3.16
ζ	1	0.5	0.316
Overshoot	0%	16%	35%
e_{ss}	0	0	0

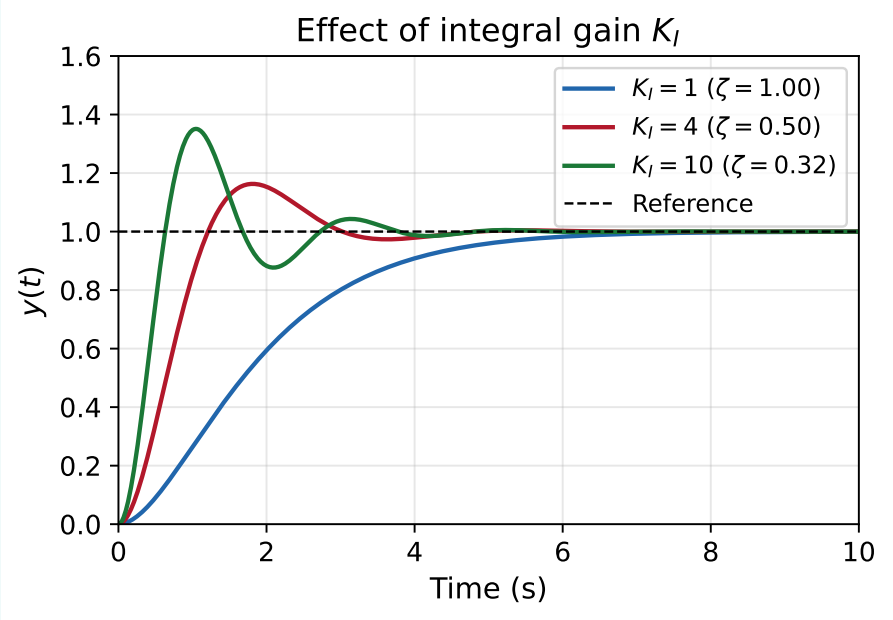


Figure 8.6: Effect of integral gain on step response. All three reach $y = 1$ (zero steady-state error), but higher K_I causes excessive oscillation due to reduced damping.

The MATLAB code in Listing 8.2 generates this comparison.

```
G = tf(1, [1 2]);
KI_values = [1, 4, 10];
figure; hold on;
for Ki = KI_values
    Gc = tf(Ki, [1 0]);  % Ki/s
    T = feedback(Gc*G, 1);
    step(T, 8);
end
yline(1, '--k');
legend('K_I=1','K_I=4','K_I=10','Reference');
title('Effect of integral gain');
grid on;

```

Listing 8.2: Step response comparison for integral control

8.5.3 Derivative Control

Now consider a pure derivative controller $C(s) = K_D s$ applied to a second-order plant $G(s) = \dfrac{1}{s^2 + 2s + 1}$. The error signal (with no disturbance) is:

$$E(s) = \frac{s^2 + 2s + 1}{s^2 + (2 + K_D)s + 1} R(s)$$

For a unit step input $R(s) = 1/s$:

$$e_{ss} = \lim_{s \to 0} sE(s) = \frac{1}{1} = 1$$

The steady-state error is 100%! The output always returns to zero regardless of the reference. Derivative control alone is useless for tracking.

However, comparing the characteristic equations:

- Without D: $s^2 + 2s + 1 = 0 \Rightarrow \zeta = 1$
- With D: $s^2 + (2 + K_D)s + 1 = 0 \Rightarrow \zeta = (2 + K_D)/2$

Derivative action *increases damping*. It acts as a brake — anticipating where the output is heading and applying a corrective force proportionally to how fast the error is changing.

Derivative control cannot be used alone

- It produces 100% steady-state error for step inputs.
- Since $|C(j\omega)| = K_D\omega$, the gain increases with frequency. This amplifies high-frequency measurement noise.
- Derivative action is always used *in combination* with P or PI control.

Derivative kick

When the reference $r(t)$ changes as a step, the error $e(t) = r(t) - y(t)$ has a step discontinuity. The derivative term $K_D \dfrac{de}{dt}$ produces a very large spike at that instant (theoretically infinite for an ideal step). This spike drives the actuator momentarily to its limits — a phenomenon called Derivative kick.

The standard remedy is to differentiate only the *measurement* $y(t)$ instead of the error:

$$u_D(t) = -K_D \frac{dy(t)}{dt} \qquad \text{instead of} \qquad u_D(t) = K_D \frac{de(t)}{dt}.$$

Since $r(t)$ is removed from the derivative calculation, step changes in the reference no longer cause a spike. This is called *derivative on measurement* (or *derivative on process variable*). Most industrial PID controllers — and the MATLAB `pid` block — use this form by default.

8.5.4 PD Control

Derivative control is combined with proportional action to give PD control:

$$C(s) = K_p + K_D s$$

Using $G(s) = \dfrac{1}{s^2 + 2s + 1}$, the error becomes:

$$E(s) = \frac{s^2 + 2s + 1}{s^2 + (2 + K_D)s + (1 + K_p)} R(s)$$

For a unit step:

$$e_{ss} = \frac{1}{1 + K_p}$$

The characteristic equation is $s^2 + (2 + K_D)s + (1 + K_p) = 0$, giving:

$$\omega_n = \sqrt{1 + K_p}, \qquad \zeta = \frac{2 + K_D}{2\sqrt{1 + K_p}}$$

Interpretation 8.3: Physical meaning of PD control

- K_p determines the natural frequency (speed of response) and reduces steady-state error.
- K_D determines the damping ratio (reduces overshoot and oscillation).
- Unlike integral control which *destabilises*, derivative control has a *stabilising* effect.
- However, PD control cannot eliminate steady-state error for type-0 systems.

The settling time is approximately:

$$T_s \approx \frac{4}{\zeta \omega_n} = \frac{4}{(2 + K_D)/2} = \frac{8}{2 + K_D}$$

So increasing K_D also reduces settling time.

8.5.5 PI Control

The PI controller $C(s) = K_p + K_I/s$ combines the speed benefit of proportional action with the zero-error guarantee of integral action. This is the most commonly used controller in process industries (temperature, flow, level, pressure control).

Example 8.3: PI Control — Speed and Accuracy

Using $G(s) = \dfrac{1}{s+2}$ and $C(s) = K_p + \dfrac{K_I}{s} = \dfrac{K_p s + K_I}{s}$, find the closed-loop transfer function and steady-state error.

Solution:

$$T(s) = \frac{C(s)G(s)}{1 + C(s)G(s)} = \frac{K_p s + K_I}{s^2 + (2 + K_p)s + K_I}$$

The characteristic equation is $s^2 + (2 + K_p)s + K_I = 0$, giving:

$$\omega_n = \sqrt{K_I}, \qquad \zeta = \frac{2 + K_p}{2\sqrt{K_I}}$$

For a unit step input:

$$e_{ss} = \lim_{s\to 0} \frac{s}{1 + C(s)G(s)} \cdot \frac{1}{s} = \lim_{s\to 0} \frac{s^2}{s^2 + (2 + K_p)s + K_I} = 0$$

Now we have two tuning knobs:

- K_I controls the speed (ω_n) and guarantees zero error.
- K_p controls the damping (ζ), reducing overshoot.

For example, with $K_I = 4$ and $K_p = 2$: $\omega_n = 2$, $\zeta = 1$ (critically damped, no overshoot, zero steady-state error).

8.5.6 Full PID Control

The full PID controller $C(s) = K_p + K_I/s + K_D s$ provides three independent tuning knobs:

- K_p: sets the overall gain level and response speed.
- K_I: eliminates steady-state error.
- K_D: adds damping, reducing overshoot and improving stability.

Remark 8.5

Summary of controller effects:

Controller	Rise time	Overshoot	Settling time	SS error
Increase K_p	Decreases	Increases	Small change	Decreases
Increase K_I	Decreases	Increases	Increases	Eliminates
Increase K_D	Small change	Decreases	Decreases	No effect

These are general guidelines. In practice, increasing any gain too much will destabilise the system.

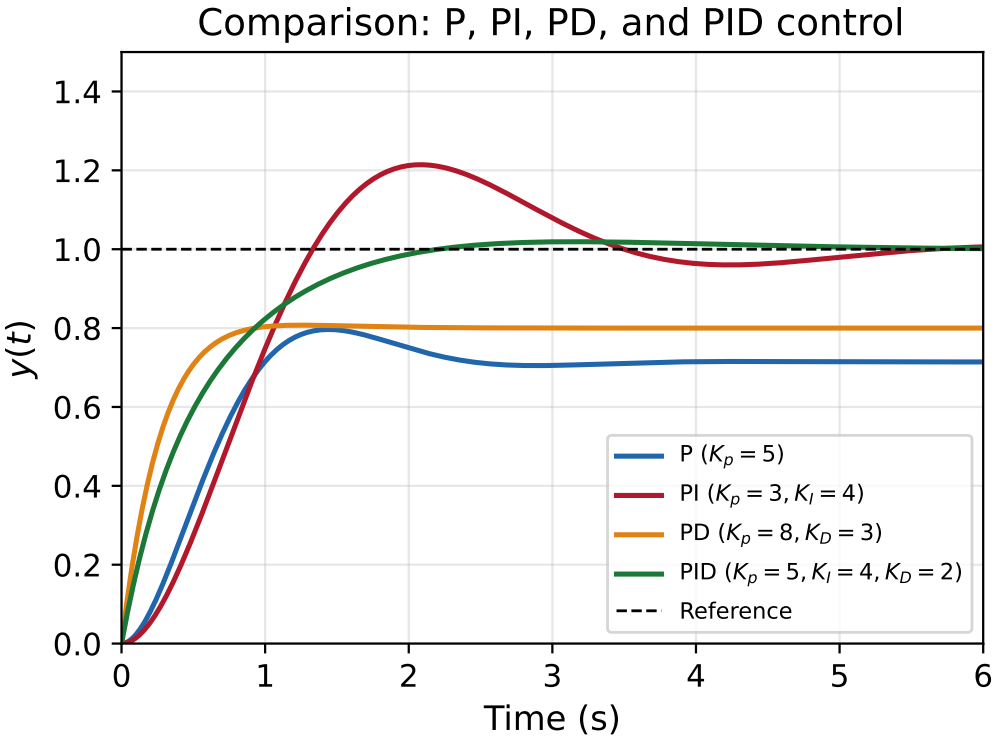


Figure 8.7: Step responses comparing different controller structures applied to $G(s) = 1/((s+1)(s+2))$. P and PD cannot eliminate steady-state error. PI eliminates it but may oscillate. PID achieves both zero error and good transient response.

8.6 PID Tuning Methods

Large industrial plants may have thousands of control loops, making individual mathematical optimisation impractical. Generic tuning methods have been developed — mainly through empirical results — to quickly tune PID controllers to an acceptable level. We study three methods:

1. Ziegler–Nichols tuning reaction curve method (open-loop experiment)
2. Ziegler–Nichols sustained oscillation method (closed-loop experiment)
3. Åström's relay-feedback method (automated tuning)

8.6.1 Ziegler–Nichols Reaction Curve Method

This method does not require a mathematical model of the plant. It works for systems whose open-loop step response has an S-shaped curve (typical of many industrial processes).

Procedure:

1. Break the feedback loop (disconnect the controller).
2. With the system at a settled condition, apply a small step change Δu (typically around 10% of the operating range) to the plant input.
3. Record the output response (the *reaction curve*).
4. Draw a tangent line at the steepest part of the response curve.
5. From the tangent, measure:
   - L = time delay (where the tangent intersects the initial value line)
   - N = normalised reaction rate, defined as $N = \dfrac{\Delta y}{\Delta u \cdot \Delta T}$, where Δy is the total output change, Δu is the step size applied, and ΔT is the time interval over which the tangent rises by Δy. Equivalently, N is the slope of the tangent line divided by the input step size.

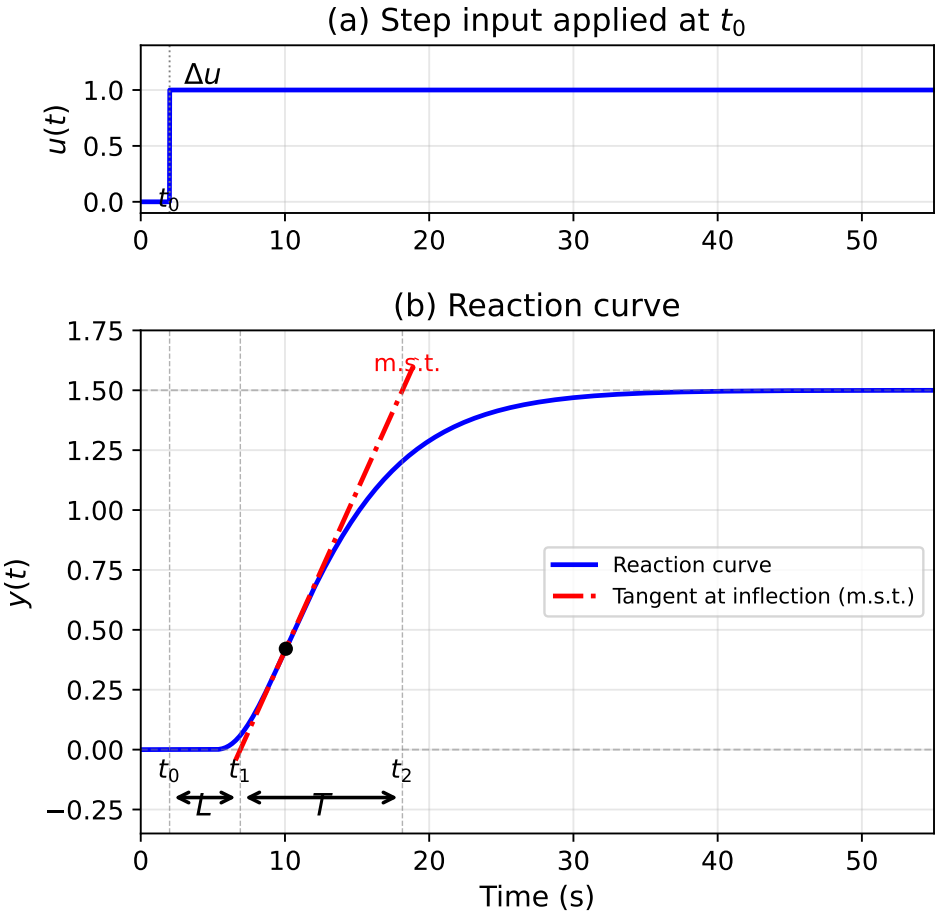


Figure 8.8: Ziegler–Nichols reaction curve method. (a) A step change Δu is applied. (b) The reaction curve is recorded; a tangent at the inflection point gives the delay L and normalised reaction rate $N = \Delta y/(\Delta u \cdot \Delta T)$.

The recommended controller parameters are:

Table 8.1: Ziegler–Nichols reaction curve tuning rules. The controller is in standard form: $C(s) = K_p\left(1 + \dfrac{1}{\tau_I s} + \tau_D s\right)$.

Controller	K_p	τ_I	τ_D
P	$\dfrac{1}{NL}$	∞	0
PI	$\dfrac{0.9}{NL}$	$3L$	0
PID	$\dfrac{1.2}{NL}$	$2L$	$0.5L$

Remark 8.6

The Ziegler–Nichols design provides only a *starting point*. The resulting response typically has 25–60% overshoot (quarter-amplitude decay criterion). Further tuning is almost always needed. However, it gives a systematic first attempt without requiring a mathematical model.

8.6.2 Ziegler–Nichols Sustained Oscillation Method

In many industrial settings, operators are hesitant to open the feedback loop or to apply large step inputs to the process. The sustained oscillation method works with the loop *closed* and uses only proportional control.

The idea is simple: for many industrial systems, increasing the proportional gain eventually causes sustained oscillations (the system reaches the boundary of stability). This critical gain value and the corresponding oscillation period contain enough information to tune a PID controller.

Procedure:

1. **Process setup:** Ensure the system is in steady state under closed-loop control. Disengage integral and derivative actions (set τ_I very large, $\tau_D = 0$).
2. **Experiment:** Carefully increment K_p until sustained oscillations are observed in the output. These should be very slowly decaying (not growing!) to maintain safe operation.
3. **Record:**
   - K_u = the ultimate gain (gain at which sustained oscillation occurs)
   - P_u = the ultimate period (period of the sustained oscillation)
4. **Calculate PID parameters** from Table 8.2.

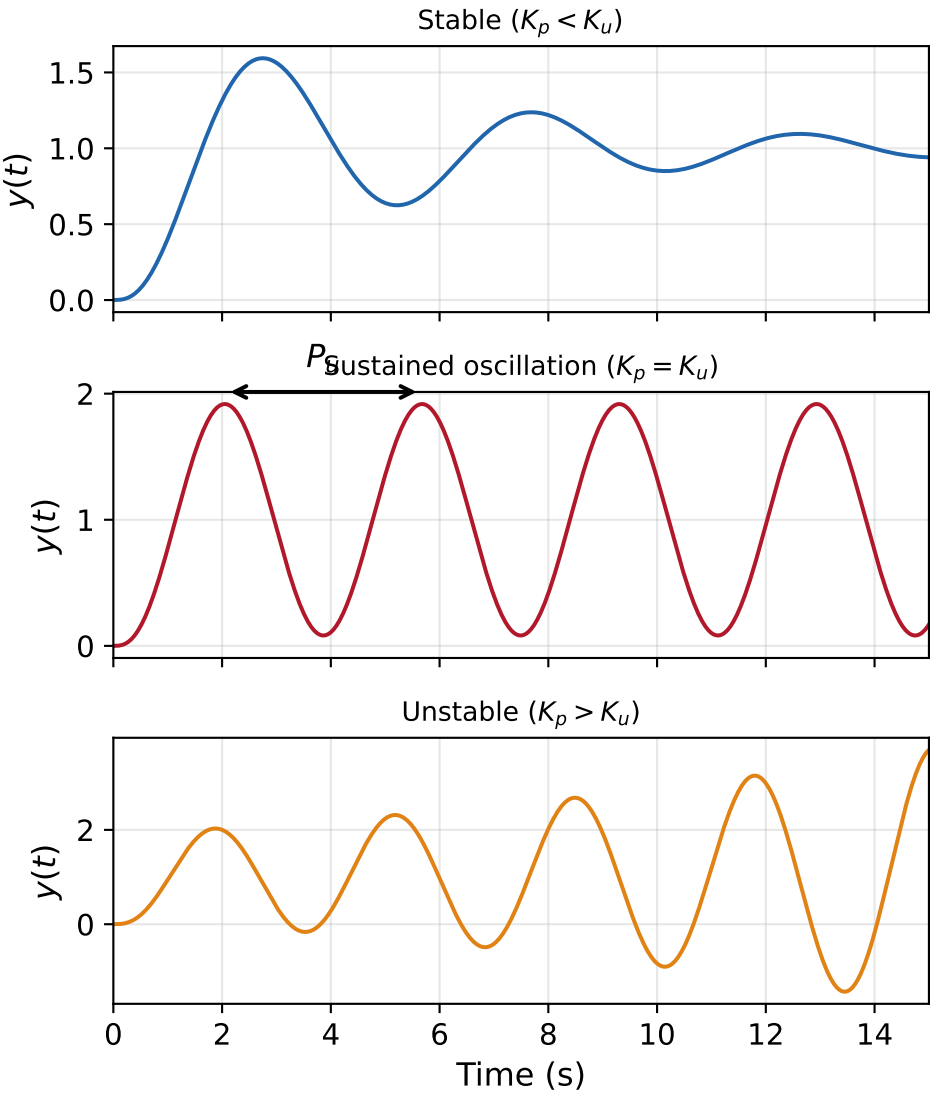


Figure 8.9: Sustained oscillation method applied to $G(s) = 1/(s(s+1)(s+3))$ with $K_u = 12$: as K_p increases from below K_u (stable), through K_u (sustained oscillation), to above K_u (unstable). We record K_u and P_u from the middle case.

Table 8.2: Ziegler–Nichols sustained oscillation tuning rules.

Controller	K_p	τ_I	τ_D
P	$0.5\,K_u$	∞	0
PI	$0.45\,K_u$	$P_u/1.2$	0
PID	$0.6\,K_u$	$P_u/2$	$P_u/8$

Example 8.4: Sustained Oscillation Method — Complete Design

A process has transfer function $G(s) = \dfrac{1}{s(s+1)(s+3)}$. Design a PID controller using the sustained oscillation method.

Solution:

Step 1: Find the ultimate gain K_u.

The closed-loop characteristic equation with $C(s) = K_p$ is:

$$1 + K_p G(s) = 0 \quad \Rightarrow \quad s(s+1)(s+3) + K_p = 0$$

$$s^3 + 4s^2 + 3s + K_p = 0$$

For sustained oscillation, substitute $s = j\omega$:

$$-j\omega^3 - 4\omega^2 + 3j\omega + K_p = 0$$

Separating real and imaginary parts:

$$\begin{aligned} \text{Real:} \quad & K_p - 4\omega^2 = 0 \quad \Rightarrow \quad K_p = 4\omega^2 \\ \text{Imaginary:} \quad & 3\omega - \omega^3 = 0 \quad \Rightarrow \quad \omega(\omega^2 - 3) = 0 \end{aligned}$$

Since $\omega \neq 0$: $\omega = \sqrt{3}$ rad/s. Therefore:

$$K_u = 4 \times 3 = 12, \qquad P_u = \frac{2\pi}{\omega} = \frac{2\pi}{\sqrt{3}} = 3.63 \text{ s}$$

Step 2: Apply Z–N PID rules:

$$\begin{aligned} K_p &= 0.6 \times K_u = 0.6 \times 12 = 7.2 \\ \tau_I &= P_u/2 = 3.63/2 = 1.815 \text{ s} \\ \tau_D &= P_u/8 = 3.63/8 = 0.454 \text{ s} \end{aligned}$$

Converting to textbook form:

$$K_I = K_p/\tau_I = 7.2/1.815 = 3.97, \qquad K_D = K_p \cdot \tau_D = 7.2 \times 0.454 = 3.27$$

The MATLAB verification is shown in Listing 8.3.

```
G = tf(1, [1 4 3 0]);
Kp = 7.2; Ki = 3.97; Kd = 3.27;
C = pid(Kp, Ki, Kd);
T = feedback(C*G, 1);
step(T, 10);
title('Z-N PID Design via Sustained Oscillation Method');
grid on;
stepinfo(T)

```

Listing 8.3: Z–N sustained oscillation PID design and verification

8.7 Åström's Relay-Feedback Method (Auto-Tuning)

The sustained oscillation method requires manually adjusting K_p to the stability boundary — a delicate and potentially dangerous procedure. Karl Johan Åström's insight was to replace the controller with an ON–OFF relay, which *automatically* generates sustained oscillations at the critical frequency without ever making the system unstable.

8.7.1 How It Works

The relay controller is:

$$u(t) = \begin{cases} +M & \text{if } e(t) \geq 0 \\ -M & \text{if } e(t) < 0 \end{cases}$$

where M is the relay amplitude (chosen within the actuator's safe range).

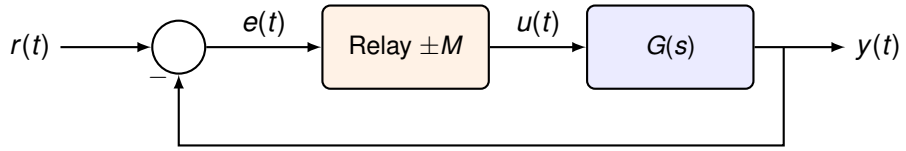


Figure 8.10: Relay-feedback auto-tuning. The PID controller is temporarily replaced by an ON–OFF relay. The relay produces bounded sustained oscillations from which K_u and P_u are measured. After tuning, the relay is switched back to the PID controller.

When the relay is connected in the feedback loop and a brief disturbance is applied (a small step in $r(t)$), the system naturally develops sustained oscillations. These oscillations are *stable by construction* — the relay keeps them bounded regardless of the plant dynamics. The amplitude never diverges.

8.7.2 Extracting K_u and P_u from the Relay Experiment

From the sustained oscillation produced by the relay:

- Measure the **ultimate period** P_u from the error signal $e(t)$ (time between successive zero crossings in the same direction).
- Measure the **oscillation amplitude** A_o of the output.

The ultimate gain is then calculated from the describing function of the relay:

$$K_u = \frac{4M}{\pi A_o} \tag{8.2}$$

where M is the relay height.

Once K_u and P_u are obtained, the standard Ziegler–Nichols sustained oscillation rules (Table 8.2) are applied to calculate the PID parameters.

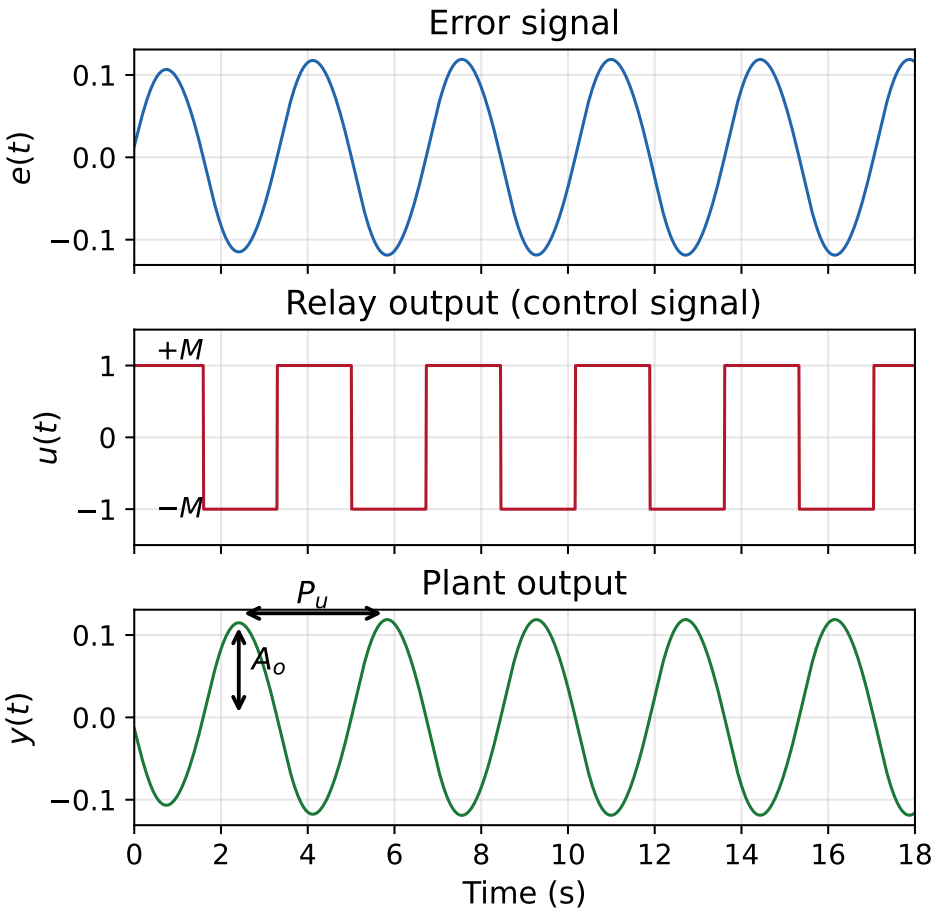


Figure 8.11: Signals during the relay-feedback experiment on $G(s) = 1/((2s+1)(s+1)(0.5s+1))$ with relay amplitude $M = 1$. The relay output $u(t)$ is a square wave; the plant output $y(t)$ is approximately sinusoidal. Measure A_o and P_u from the output to compute $K_u = 4M/(\pi A_o)$.

Remark 8.7

Advantages of the relay method over manual sustained oscillation:

- The oscillation amplitude is bounded by the relay height M — no risk of instability.
- The experiment is fast (typically 3–5 oscillation periods are sufficient).
- It can be fully automated — modern process controllers have an "Auto-Tune" button that implements exactly this procedure.
- The relay amplitude M can be chosen within the actuator's safe operating range.

Example 8.5: Relay Auto-Tuning in MATLAB

Demonstrate the relay-feedback experiment for $G(s) = \dfrac{1}{(2s+1)(s+1)(0.5s+1)}$.

Solution:

Listing 8.4 demonstrates the relay-feedback procedure.

```
% Plant
G = tf(1, conv(conv([2 1],[1 1]),[0.5 1]));

% Simulate relay feedback using Simulink or manual code:
% For illustration, find Ku and Pu analytically/numerically:
% Use allmargin to find the gain crossover
[Gm, Pm, Wcg, Wcp] = margin(G);
Ku = Gm;            % Ultimate gain = gain margin
Pu = 2*pi/Wcg;      % Ultimate period

fprintf('Ku = %.3f, Pu = %.3f s\n', Ku, Pu);

% Z-N PID tuning from sustained oscillation rules
Kp = 0.6*Ku;
Ti = Pu/2;
Td = Pu/8;
Ki = Kp/Ti;
Kd = Kp*Td;

C = pid(Kp, Ki, Kd);
T = feedback(C*G, 1);
step(T, 20);
title('PID controller designed via relay method');
grid on;

```

Listing 8.4: Relay auto-tuning: finding K_u, P_u and designing a PID controller

In practice with a real relay experiment, K_u would be computed from (8.2) using the measured A_o and M.

8.8 PID Tuning with MATLAB

Modern software tools can directly optimise PID gains for specified performance criteria, providing a more systematic alternative to hand-tuning methods.

8.8.1 The `pidtune()` Function

MATLAB's `pidtune()` automatically designs a PID controller that balances performance and robustness. Listing 8.5 shows how to compare P, PI, and PID designs.

```
G = tf(1, [1 4 3 0]);    % Plant

% Design different controller types
C_P   = pidtune(G, 'P');   % P-only
C_PI  = pidtune(G, 'PI');  % PI
C_PID = pidtune(G, 'PID'); % Full PID

% Compare closed-loop responses
T_P   = feedback(C_P*G, 1);
T_PI  = feedback(C_PI*G, 1);
T_PID = feedback(C_PID*G, 1);

figure;
step(T_P, T_PI, T_PID, 10);
legend('P','PI','PID');
title('Comparison of controller types (pidtune)');
grid on;
```

Listing 8.5: Comparing P, PI, and PID controllers using `pidtune()`

8.8.2 Adjusting Performance with Bandwidth

The third argument of `pidtune()` specifies the desired crossover frequency (bandwidth). Higher bandwidth means faster response but less robustness, as shown in Listing 8.6.

```
G = tf(5, [1 6 5]);

% Conservative (slow, robust)
C_slow = pidtune(G, 'PID', 2);

% Aggressive (fast, less robust)
C_fast = pidtune(G, 'PID', 15);

% Default
C_default = pidtune(G, 'PID');

T_slow = feedback(C_slow*G, 1);
T_fast = feedback(C_fast*G, 1);
T_def  = feedback(C_default*G, 1);

step(T_slow, T_def, T_fast, 5);
legend('Slow (wc=2)','Default','Fast (wc=15)');
title('Effect of target bandwidth on PID design');
```

Listing 8.6: Effect of target bandwidth on PID design

8.8.3 PID Tuning in Simulink

Simulink provides a dedicated PID Controller block (in the *Continuous* library) that includes:

- Derivative filtering (adjustable filter coefficient N)
- Anti-windup (back-calculation or clamping)
- Output saturation limits
- Setpoint weighting

Building a PID loop in Simulink:

1. Add a **Step** block (reference input).
2. Add a **Sum** block (error = reference $-$ output).
3. Add a **PID Controller** block.
4. Add a **Transfer Fcn** block (your plant).
5. Connect in a feedback loop.
6. Add **Scope** blocks to observe signals.

Double-click the PID Controller block to enter gains manually, or click **"Tune..."** to open the interactive PID Tuner, which lets you adjust the speed–robustness tradeoff using a graphical slider.

8.9 Practical Considerations

8.9.1 Derivative Filtering

Pure differentiation ($K_D s$) has infinite gain at high frequencies, making it impractical due to measurement noise. In all real implementations, the derivative term includes a low-pass filter:

$$D(s) = \frac{K_D s}{1 + s/N}$$

where N is the filter coefficient (typically $N = 10$ to 100). This limits the high-frequency gain to $K_D N$ rather than infinity.

The complete *practical PID* transfer function is:

$$C(s) = K_p + \frac{K_I}{s} + \frac{K_D s}{1 + s/N}$$

MATLAB's `pid()` object uses this form by default with $N = 100$.

8.9.2 Integral Windup and Anti-Windup

In real systems, actuators have physical limits (e.g., a motor can supply at most ± 12 V, a valve opens between 0% and 100%). When the controller output saturates at these limits, the integrator continues to accumulate error, growing to a very large value. When the error finally changes sign, the large integral prevents the controller from responding quickly — this is called Integral windup.

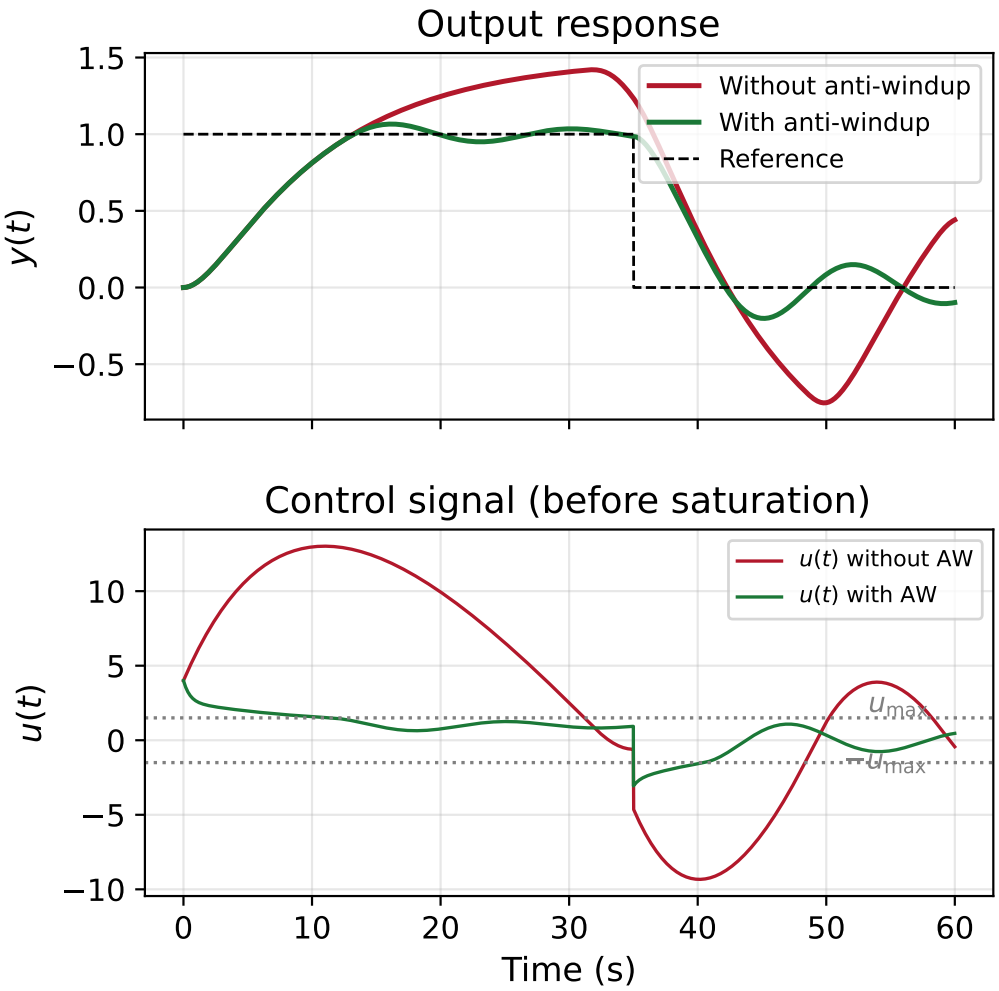


Figure 8.12: Integral windup demonstration: PI control of $G(s) = 1/((10s+1)(2s+1))$ with $K_p = 4$, $K_i = 2$, and actuator limit $u_{\max} = 1.5$. A step reference is applied at $t = 0$ and removed at $t = 35\,\text{s}$. Without anti-windup (red), the integrator winds up during prolonged saturation, causing 42% overshoot and a large undershoot on the step-down. With back-calculation anti-windup (green), the overshoot is reduced to under 7% and the step-down response is much cleaner.

Remedies (Anti-windup):

- **Clamping:** Stop integrating when the controller output hits the saturation limit.
- **Back-calculation:** Feed back the difference between the saturated and unsaturated controller output to "unwind" the integrator.

Both methods are available in Simulink's PID Controller block, as illustrated in Listing 8.7.

```
% Continuous-time PID controller
C = pid(Kp, Ki, Kd);

% In Simulink: double-click PID block -> Anti-Windup -> Enable
```

```
% Choose method: back-calculation or clamping
% Set upper/lower saturation limits in the block parameters
```

Listing 8.7: PID with anti-windup in Simulink

8.10 Limitations of PID Control — Motivation for State-Space Methods

PID control has fundamental limitations that motivate the state-space techniques studied in the remaining chapters:

1. **Single-loop only:** PID is a single-input, single-output (SISO) controller. It cannot coordinate multiple actuators or handle strong coupling between variables.
2. **Non-minimum phase systems:** When a system has right-half-plane (RHP) zeros, the output initially moves in the *opposite* direction to the desired response. PID reacts to the error and cannot anticipate or compensate for this inverse-response behaviour. Increasing the gains makes the initial undershoot worse.
3. **Only uses the output:** PID acts on the measured output error. If important internal states are not directly measured, PID cannot address their dynamics.
4. **No systematic pole placement:** With PID, we have only 3 parameters (K_p, K_I, K_D). For an n-th order system with $n > 3$, we cannot place all closed-loop poles freely.

Example 8.6: When PID Struggles: A Non-Minimum Phase System

Consider a system with a right-half-plane zero:

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

This is a **non-minimum phase** system. The zero at $s = +1$ causes the step response to initially dip *below zero* before rising toward the reference — an **inverse response**. Physically, such behaviour arises in systems where the initial reaction opposes the final steady-state direction (e.g., certain chemical processes, boiler water level dynamics, some aircraft manoeuvres).

Solution:

A PID controller only reacts to the error signal. It does not "understand" that the initial wrong-way response is temporary. As the gains are increased to improve tracking speed, the inverse-response dip becomes deeper and the overshoot grows larger, as shown in Listing 8.8.

```
G = tf([-1 1],[1 2 1]);  % G(s) = (1-s)/(s+1)^2

% Conservative PI
C1 = pid(0.5, 0.1);
T1 = feedback(C1*G, 1);

% Moderate PI
C2 = pid(1.0, 0.3);
T2 = feedback(C2*G, 1);

% Aggressive PI
```

```
C3 = pid(1.5, 0.5);
T3 = feedback(C3*G, 1);

step(T1, T2, T3, 25);
legend('Conservative', 'Moderate', 'Aggressive');

```

Listing 8.8: PI control of a non-minimum phase plant

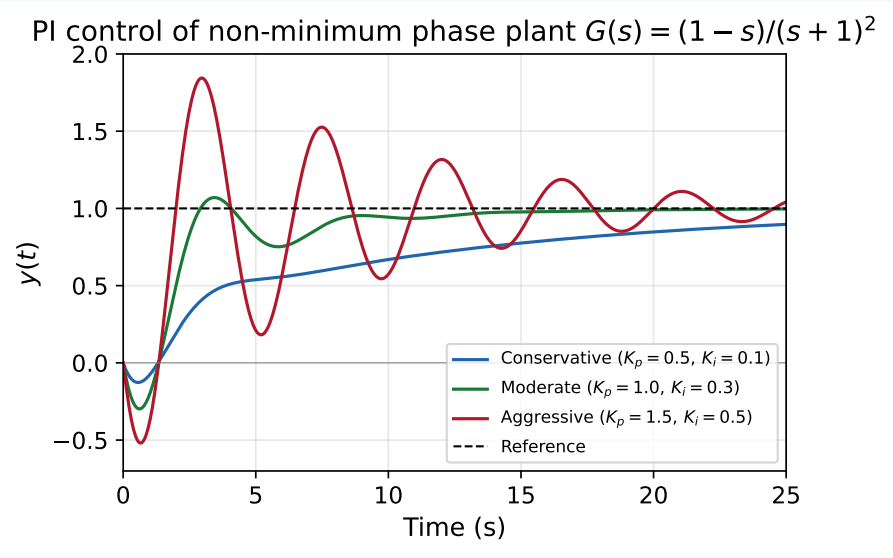


Figure 8.13: PI control of a non-minimum phase plant $G(s) = (1 - s)/(s + 1)^2$. Conservative tuning (blue) gives a small inverse dip but large steady-state error. Moderate tuning (green) achieves good tracking but with a deeper dip. Aggressive tuning (red) produces a large inverse dip and 84% overshoot. There is no PI/PID tuning that eliminates both the inverse response and the overshoot.

The fundamental trade-off is clear: *increasing the gains to reduce the steady-state error unavoidably deepens the inverse response and increases the overshoot*. A model-based controller (Chapter 9) that knows about the internal plant structure can handle this situation far more effectively.

Remark 8.8

PID control is highly effective for many low-order, well-damped SISO systems, but it is not universal. Systems with:

- Multiple inputs/outputs (MIMO)
- Right-half-plane zeros (non-minimum phase / inverse response)
- Important unmeasured internal states
- Actuator constraints (saturation, rate limits)
- More poles than tuning parameters

require controllers that explicitly account for the model structure and constraints. The state-space control methods developed in Chapters 8–11 provide this capability.

8.11 Summary

Chapter Summary

- The PID controller generates control action based on the present error (K_p), accumulated past error (K_I), and rate of change of error (K_D).
- Two equivalent forms: textbook ($K_p + K_I/s + K_D s$) and industrial ($K_p(1 + 1/\tau_I s + \tau_D s)$).
- Proportional action provides speed but cannot eliminate steady-state error.
- Integral action guarantees zero steady-state error but reduces damping (destabilising effect).
- Derivative action improves damping (stabilising effect) but amplifies noise and cannot be used alone.
- The Ziegler–Nichols reaction curve method uses an open-loop step experiment.
- The Ziegler–Nichols sustained oscillation method uses the critical gain K_u and period P_u.
- Åström's relay method automates the sustained oscillation experiment safely.
- MATLAB `pidtune()` and Simulink PID Tuner provide modern alternatives.
- Practical issues include derivative filtering (coefficient N) and integral windup (anti-windup schemes).
- PID has limitations for non-minimum phase, MIMO, and high-order systems — motivating state-space methods.

8.12 End-of-Chapter Problems

1. **(Conceptual)** Explain in your own words why:
   (a) Proportional control cannot eliminate steady-state error for a type-0 plant.
   (b) Integral action guarantees zero steady-state error but may destabilise the system.
   (c) Derivative action has a stabilising effect but cannot be used alone.
2. **(Analysis)** Consider the system in Figure 8.4 with $G(s) = \dfrac{1}{s+2}$ and no disturbance ($D = 0$).
   (a) With $C(s) = K_p$: derive the closed-loop transfer function, find the steady-state error for a unit step, and determine how large K_p must be to achieve $e_{ss} < 5\%$.
   (b) With $C(s) = K_p + K_I/s$ (PI control): show that $e_{ss} = 0$ for any positive K_I.
   (c) For the PI case with $K_p = 4$ and $K_I = 4$: find the closed-loop poles, damping ratio, and natural frequency. Is this design satisfactory?

3. **(Analysis with disturbance)** For the same plant $G(s) = 1/(s+2)$ with $C(s) = K_p$, a unit step reference and a step disturbance of magnitude $d = 0.5$ are applied simultaneously.
   (a) Derive the steady-state error as a function of K_p and d.
   (b) What value of K_p is needed to keep the total steady-state error below 10%?
   (c) Explain why PI control would be superior for disturbance rejection.

4. **(PD Control)** A second-order plant $G(s) = \dfrac{1}{s^2 + 2s + 1}$ is controlled by $C(s) = K_p + K_D s$.
   (a) Derive the closed-loop characteristic equation.
   (b) Express ω_n and ζ in terms of K_p and K_D.
   (c) Design K_p and K_D to achieve $\omega_n = 4$ rad/s and $\zeta = 0.7$.
   (d) What is the steady-state error? Why can't PD control eliminate it here?

5. **(Ziegler–Nichols Step Response)** A process has an S-shaped open-loop step response with the following measurements: step input magnitude $\Delta u = 2$, final output change $\Delta y = 3$, delay $L = 1.5$ s, and the tangent intersects the final value at $t = 7.5$ s (so $\Delta T = 6$ s).
   (a) Calculate the reaction rate N.
   (b) Use the Z–N reaction curve rules to find P, PI, and PID controller parameters.
   (c) Implement all three in MATLAB using the approximate model $G(s) = \dfrac{1.5\, e^{-1.5s}}{6s + 1}$ and compare step responses.

6. **(Ziegler–Nichols Sustained Oscillation)** A wastewater treatment plant has the transfer function:
$$G(s) = \frac{1}{5s^3 + 15.5s^2 + 11.5s + 1}$$
   (a) Find the ultimate gain K_u and ultimate period P_u using the Routh–Hurwitz criterion or by substituting $s = j\omega$ into $1 + K_p G(s) = 0$.
   (b) Design a PI controller using the Z–N sustained oscillation rules.
   (c) Implement in MATLAB and assess the tracking response. Is it acceptable? If not, how would you modify the gains?

7. **(Relay Method)** For the same plant as Problem 6:
   (a) If a relay experiment with $M = 1$ produces output oscillations of amplitude $A_o = 0.8$ and period $P_u = 5.2$ s, calculate K_u.
   (b) Design a PID controller using Z–N rules with these values.
   (c) Compare this design with the one from Problem 6 in MATLAB.

8. **(MATLAB)** Consider the plant $G(s) = \dfrac{10}{(s+2)(s+5)}$.
   (a) Use `pidtune(G, 'P')`, `pidtune(G, 'PI')`, and `pidtune(G, 'PID')` to design three controllers.
   (b) Plot all three closed-loop step responses on one figure. Use `stepinfo()` to compare rise time, settling time, overshoot, and steady-state error.
   (c) For the PID case, vary the target bandwidth ($\omega_c = 2, 5, 10, 20$ rad/s) and observe the tradeoff between speed and robustness.

9. **(Integral Windup)** A motor speed controller uses PI control with $K_p = 2$, $K_I = 10$. The motor input is limited to ± 12 V.
   (a) Build a Simulink model with a saturation block after the controller.

(b) Apply a step reference from 0 to 100 rad/s. Observe the output with and without anti-windup.

(c) Explain the difference in transient response.

10. **(Limitations — Non-Minimum Phase)** Consider the system $G(s) = \dfrac{1 - s}{(s + 1)^2}$.

(a) Where is the right-half-plane zero? What does it imply about the step response?

(b) Design PI controllers with $K_p = 0.5, 1.0, 1.5$ (with $K_I = 0.1, 0.3, 0.5$ respectively). Plot the closed-loop step responses and comment on the trade-off between tracking speed and the initial inverse response.

(c) Explain why increasing the gains makes the inverse response *worse*, and discuss what type of controller might handle this system better.

Chapter 9

Frequency-Domain Analysis and Synthesis

Learning Objectives

After completing this chapter, you should be able to:

- Compute the sinusoidal steady-state output of a stable LTI system from $G(j\omega)$.
- Convert between magnitude ratios and decibels, and use decades and octaves correctly.
- Sketch asymptotic Bode magnitude and phase plots from first-order and second-order building blocks.
- Read gain margin, phase margin, crossover frequencies, and bandwidth from a Bode plot.
- Translate time-domain specifications (overshoot, settling time, steady-state error) into frequency-domain targets (PM, ω_{gc}, K_v).
- Design lead, lag, and lag–lead compensators using the systematic Bode-plot procedure.
- Verify frequency-domain designs in MATLAB using `margin()`, `bode()`, and step response simulation.

Frequency-domain methods show how a system treats slow, medium, and fast signals. That viewpoint is one of the most useful ways to understand feedback.

9.1 Why Frequency Methods Are Needed

In practice, signals and disturbances span a wide range of frequencies: a temperature sensor carries high-frequency noise, a vehicle sees low-frequency road undulations and higher-frequency vibration, and a motor drive contains periodic ripple from power electronics. Frequency-domain analysis asks: *how does the system respond at each frequency?*

Good tracking of slowly varying references requires large loop gain at low frequencies, while high-frequency sensor noise should not be amplified, so the loop gain must be small at high frequencies. Between these two extremes lies the crossover region, where gain and phase margins indicate how much modelling error the closed loop can tolerate before instability. The frequency-domain viewpoint makes all of these trade-offs visible on a single diagram.

Remark 9.1

Frequency-domain methods do not replace time-domain methods. They provide a different view of the same dynamics. Good control design usually requires both views.

9.2 Sinusoidal Steady-State Response

Consider a stable continuous-time LTI system with transfer function $G(s)$. If the input is a sinusoid

$$u(t) = A\sin(\omega t),$$

then, after transients have died away, the output settles to a sinusoid of the *same* frequency ω, but generally with a different amplitude and a phase shift:

$$y_{\text{ss}}(t) = A\,|G(j\omega)|\,\sin\!\big(\omega t + \angle G(j\omega)\big).$$

9.2.1 Derivation for a first-order system

To see why, consider the simplest possible stable system,

$$G(s) = \frac{1}{s+a}, \qquad a > 0.$$

The Laplace transform of $u(t) = A\sin(\omega t)$ is $U(s) = A\omega/(s^2+\omega^2)$, so

$$Y(s) = G(s)\,U(s) = \frac{A\omega}{(s+a)(s^2+\omega^2)}.$$

Expanding in partial fractions:

$$Y(s) = \frac{c_1}{s+a} + \frac{Bs + C}{s^2+\omega^2}.$$

Setting $s = -a$ gives $c_1 = A\omega/(a^2+\omega^2)$. Comparing coefficients of s^2 and s^0 yields

$$B = -\frac{A\omega}{a^2+\omega^2}, \qquad C = \frac{A\omega\, a}{a^2+\omega^2}.$$

The first term inverts to $c_1\, e^{-at}$, which decays to zero because $a > 0$. The second term inverts using the standard pairs $\mathcal{L}^{-1}\{s/(s^2+\omega^2)\} = \cos\omega t$ and $\mathcal{L}^{-1}\{\omega/(s^2+\omega^2)\} = \sin\omega t$:

$$y_{\text{ss}}(t) = B\cos\omega t + \frac{C}{\omega}\sin\omega t = \frac{A}{a^2+\omega^2}\big(a\sin\omega t - \omega\cos\omega t\big).$$

The expression in brackets is a sum of a sine and a cosine at the same frequency. Any such combination can be written as a single sinusoid using the identity

$$\alpha\sin\theta - \beta\cos\theta = \sqrt{\alpha^2+\beta^2}\;\sin\!\big(\theta - \tan^{-1}(\beta/\alpha)\big).$$

With $\alpha = a$ and $\beta = \omega$:

$$y_{\text{ss}}(t) = \frac{A}{\cancel{a^2+\omega^2}}\cdot\sqrt{\cancel{a^2+\omega^2}}\;\sin\!\big(\omega t - \tan^{-1}(\omega/a)\big) = \frac{A}{\sqrt{a^2+\omega^2}}\;\sin\!\big(\omega t - \tan^{-1}(\omega/a)\big).$$

Recognising that $|G(j\omega)| = 1/\sqrt{a^2+\omega^2}$ and $\angle G(j\omega) = -\tan^{-1}(\omega/a)$, this is

$$\boxed{y_{\text{ss}}(t) = A\,|G(j\omega)|\,\sin\!\big(\omega t + \angle G(j\omega)\big).}$$

The same argument extends to any stable rational transfer function: every pole of $G(s)$ in the left half-plane produces a decaying transient, and the only terms that persist come from the input poles at $s = \pm j\omega$.

Sinusoidal steady-state rule

For a stable LTI system, a sinusoidal input produces a sinusoidal output at the same frequency. The system changes only the amplitude (by the factor $|G(j\omega)|$) and the phase (by the angle $\angle G(j\omega)$). The function $G(j\omega)$ is called the Frequency response of the system.

Example 9.1: Sinusoidal response of a first-order system

Consider

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

At frequency ω,

$$G(j\omega) = \frac{1}{1+j\omega}, \qquad |G(j\omega)| = \frac{1}{\sqrt{1+\omega^2}}, \qquad \angle G(j\omega) = -\tan^{-1}(\omega).$$

If $u(t) = 2\sin(3t)$, then $A = 2$ and $\omega = 3$. We compute

$$|G(j3)| = \frac{1}{\sqrt{10}} \approx 0.316, \qquad \angle G(j3) = -\tan^{-1}(3) \approx -71.6^\circ.$$

The steady-state output is therefore

$$y_{ss}(t) = 2 \times 0.316 \, \sin(3t - 71.6^\circ) = 0.632 \sin(3t - 71.6^\circ).$$

The amplitude has been reduced to about one-third of the input, and the output lags behind by nearly a quarter period. This is the filtering action of the first-order low-pass system. Listing 9.1 shows how to verify this in MATLAB.

```
s = tf('s');
G = 1/(s+1);
w = 3;
[mag, phase] = bode(G, w);
mag = squeeze(mag);        % magnitude ratio
phase = squeeze(phase);    % phase in degrees
fprintf('Amplitude = %.3f, Phase = %.1f deg\n', 2*mag, phase)
```

Listing 9.1: MATLAB code for checking sinusoidal steady-state response.

Example 9.2: Second-order system at resonance

Consider

$$G(s) = \frac{100}{s^2 + 2s + 100},$$

which has $\omega_n = 10$ rad/s and $\zeta = 0.1$. At the natural frequency,

$$G(j10) = \frac{100}{(j10)^2 + 2(j10) + 100} = \frac{100}{-100 + j20 + 100} = \frac{100}{j20} = -j5.$$

Hence $|G(j10)| = 5$ and $\angle G(j10) = -90^\circ$. Although the DC gain of the system is equal to 1, a unit-amplitude sinusoid at $\omega = 10$ rad/s produces a steady-state output with amplitude 5 — the system *amplifies* inputs near its resonant frequency. This resonance amplification is one of the most important features that Bode plots reveal.

9.3 How a Frequency Response Plot Is Generated

Conceptually, a frequency response plot is produced by applying many separate sinusoidal inputs, one frequency at a time, and recording the steady-state response.

For a stable LTI system, the experiment at each test frequency ω is:

$$u(t) = A\sin(\omega t) \quad \Longrightarrow \quad y_{ss}(t) = B(\omega)\sin(\omega t + \phi(\omega)).$$

We then record the gain and phase:

$$|G(j\omega)| = \frac{B(\omega)}{A}, \qquad \angle G(j\omega) = \phi(\omega).$$

Repeating this at many frequencies gives a table of magnitude and phase values. Plotting the magnitude (in dB) and phase (in degrees) against frequency gives the Bode plot.

Example 9.3: Generating frequency-response data point by point

Suppose a first-order system $G(s) = 1/(s+1)$ is tested using sinusoidal inputs of amplitude $A = 1$. The measured steady-state output amplitudes and phase shifts are:

ω (rad/s)	0.1	0.3	1	3	10
$B(\omega)$	0.995	0.958	0.707	0.316	0.100
$\phi(\omega)$	-5.7°	-16.7°	-45°	-71.6°	-84.3°

The Bode magnitude values are not the amplitudes directly. They are the amplitudes converted to dB:

$$20\log_{10} B(\omega).$$

For example, at $\omega = 3$ rad/s,

$$20\log_{10}(0.316) \approx -10 \text{ dB}.$$

At $\omega = 1$ rad/s the magnitude is -3 dB and the phase is -45°. This frequency is the Break frequency (or corner frequency) of the first-order system, and it marks the boundary between the low-frequency pass-band and the high-frequency roll-off.

Figure 9.1 summarises the entire process: test the system at one frequency, record the steady-state gain and phase, then repeat at many frequencies to build the Bode plot.

Figure 9.2 shows the resulting Bode plot built from these five measured points together with the exact analytical curve. This experimental view is important. When MATLAB draws `bode(G)`, it is doing the same idea analytically: it evaluates $G(j\omega)$ at many frequencies and plots the resulting magnitude and phase.

9.4 Decades, Octaves, Decibels, and Semi-Log Graphs

Frequency responses often cover several orders of magnitude, so a linear frequency axis would either waste space at low frequencies or compress the high-frequency behaviour. Bode plots use a logarithmic frequency axis.

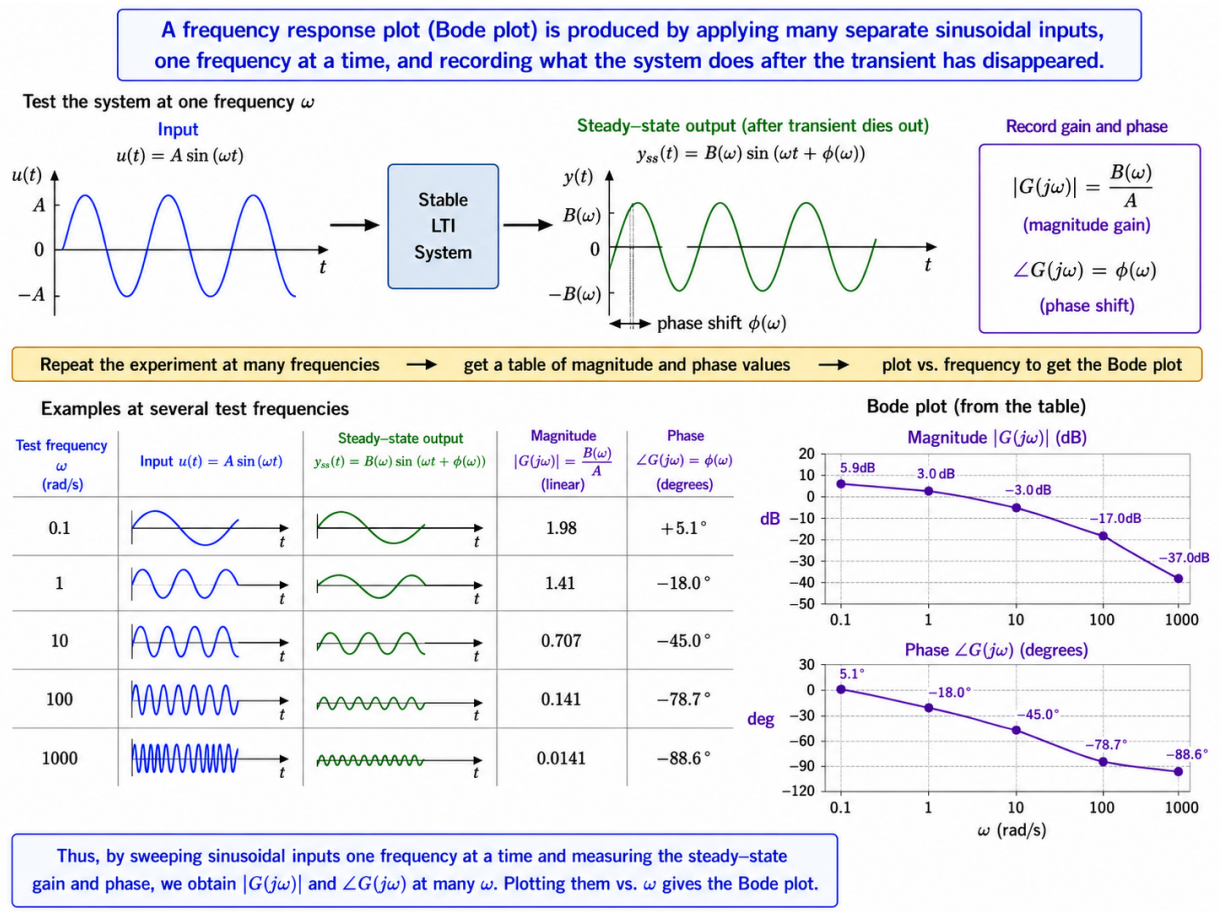


Figure 9.1: Overview of the frequency-response experiment. A sinusoidal input at frequency ω produces a sinusoidal steady-state output with modified amplitude and phase. Repeating at many frequencies and plotting the results gives the Bode plot.

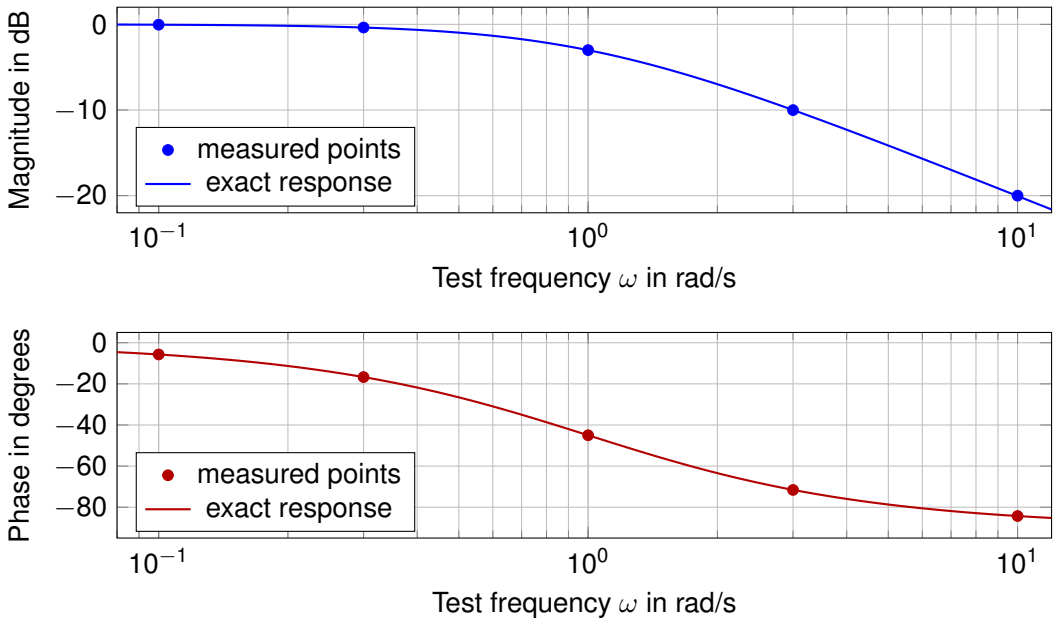


Figure 9.2: A Bode plot can be understood as a frequency sweep: apply one sinusoid, measure gain and phase, then repeat at many frequencies.

Definition 9.1: Decade and octave

A **decade** is a factor of 10 in frequency. Moving from 1 rad/s to 10 rad/s is one decade; moving from 10 rad/s to 1000 rad/s is two decades.

An **octave** is a factor of 2 in frequency. Moving from 5 rad/s to 10 rad/s is one octave.

Slopes in Bode magnitude plots are often described in dB/decade or dB/octave. A first-order pole contributes -20 dB/decade after its break frequency. Since one decade is about 3.32 octaves, this is equivalently -6 dB/octave.

Definition 9.2: Decibel scale for Bode plots

In a Bode magnitude plot, the transfer-function magnitude is an amplitude ratio:

$$M = |G(j\omega)| = \frac{\text{output amplitude}}{\text{input amplitude}}.$$

This amplitude ratio is usually expressed in Decibel (dB) units as

$$M_{\text{dB}} = 20 \log_{10} M.$$

Thus:

$$M = 1 \quad \Longrightarrow \quad 0 \text{ dB},$$

$$M = 10 \quad \Longrightarrow \quad 20 \text{ dB},$$

and

$$M = 0.1 \quad \Longrightarrow \quad -20 \text{ dB}.$$

The factor 20 appears because power is proportional to amplitude squared. The original decibel definition for power ratios uses

$$10 \log_{10} \left(\frac{P_2}{P_1} \right).$$

If $P \propto A^2$, then an amplitude ratio M corresponds to a power ratio M^2, so

$$10 \log_{10}(M^2) = 20 \log_{10} M.$$

Some useful conversions worth remembering:

M	0.01	0.1	0.5	1	2	10	100
M_{dB}	-40	-20	-6	0	+6	+20	+40

9.5 Bode Plots

A **Bode plot** consists of two panels drawn on the same logarithmic frequency axis. The upper panel shows the magnitude $20 \log_{10} |G(j\omega)|$ in decibels. The lower panel shows the phase $\angle G(j\omega)$ in degrees.

To read the magnitude plot, choose a frequency on the horizontal axis and move vertically to the curve. A value of 0 dB means the output sinusoid has the same amplitude as the input sinusoid. A value of +20 dB

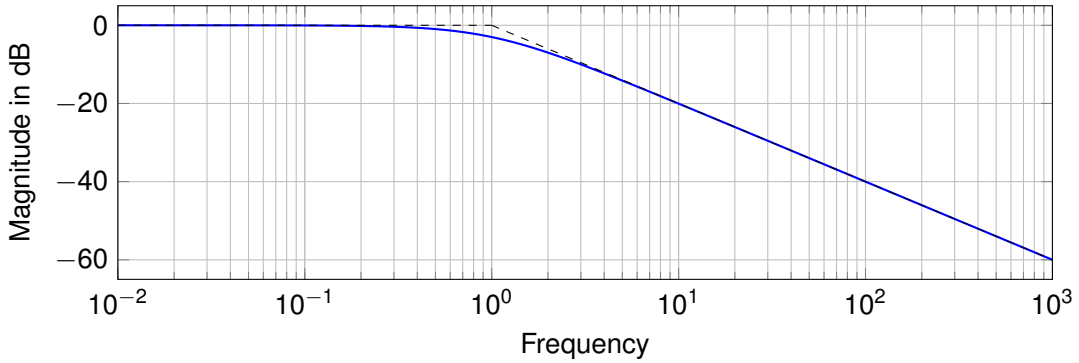


Figure 9.3: A semi-log magnitude plot for a first-order low-pass response. The dashed lines show the asymptotic approximation.

means the output amplitude is 10 times larger. A value of -20 dB means the output amplitude is 10 times smaller. Figure 9.3 shows a typical example: the magnitude of a first-order low-pass system plotted on a semi-log frequency axis, together with its straight-line asymptotic approximation.

To read the phase plot, use the same frequency and read the phase angle. A phase of -90° means the output sinusoid lags the input by one quarter of a period. The time lag corresponding to a phase angle ϕ (in degrees, negative for lag) at frequency ω is

$$\Delta t = \frac{|\phi|}{360^\circ} \cdot \frac{2\pi}{\omega}.$$

The same phase angle corresponds to a different time lag at a different frequency.

Example 9.4: Reading a Bode plot point

Suppose a Bode plot gives $|G(j5)|_{\text{dB}} = -6$ dB and $\angle G(j5) = -60^\circ$. The amplitude ratio is

$$|G(j5)| = 10^{-6/20} \approx 0.50.$$

Thus an input $u(t) = 4\sin(5t)$ gives the steady-state output

$$y_{\text{ss}}(t) \approx 4 \times 0.50 \, \sin(5t - 60^\circ) = 2\sin(5t - 60^\circ).$$

The time delay at this frequency is

$$\Delta t = \frac{60}{360} \cdot \frac{2\pi}{5} \approx 0.209 \text{ s}.$$

Figure 9.4 summarises this four-step procedure visually: locate the input frequency on the horizontal axis, read the magnitude and phase from the two panels, then construct the steady-state output.

9.5.1 Exact and asymptotic Bode plots

The **exact Bode plot** is obtained by evaluating $G(j\omega)$ directly at every frequency — MATLAB does this. The **asymptotic Bode plot** is a hand-sketch approximation built from straight-line magnitude segments and approximate phase transitions. Its purpose is to reveal the structure of the transfer function quickly: where the break frequencies are, what the slopes are, and roughly what the margins look like. After drawing the asymptotic sketch, always check the exact response near break frequencies and lightly damped second-order modes where the approximation can be poor.

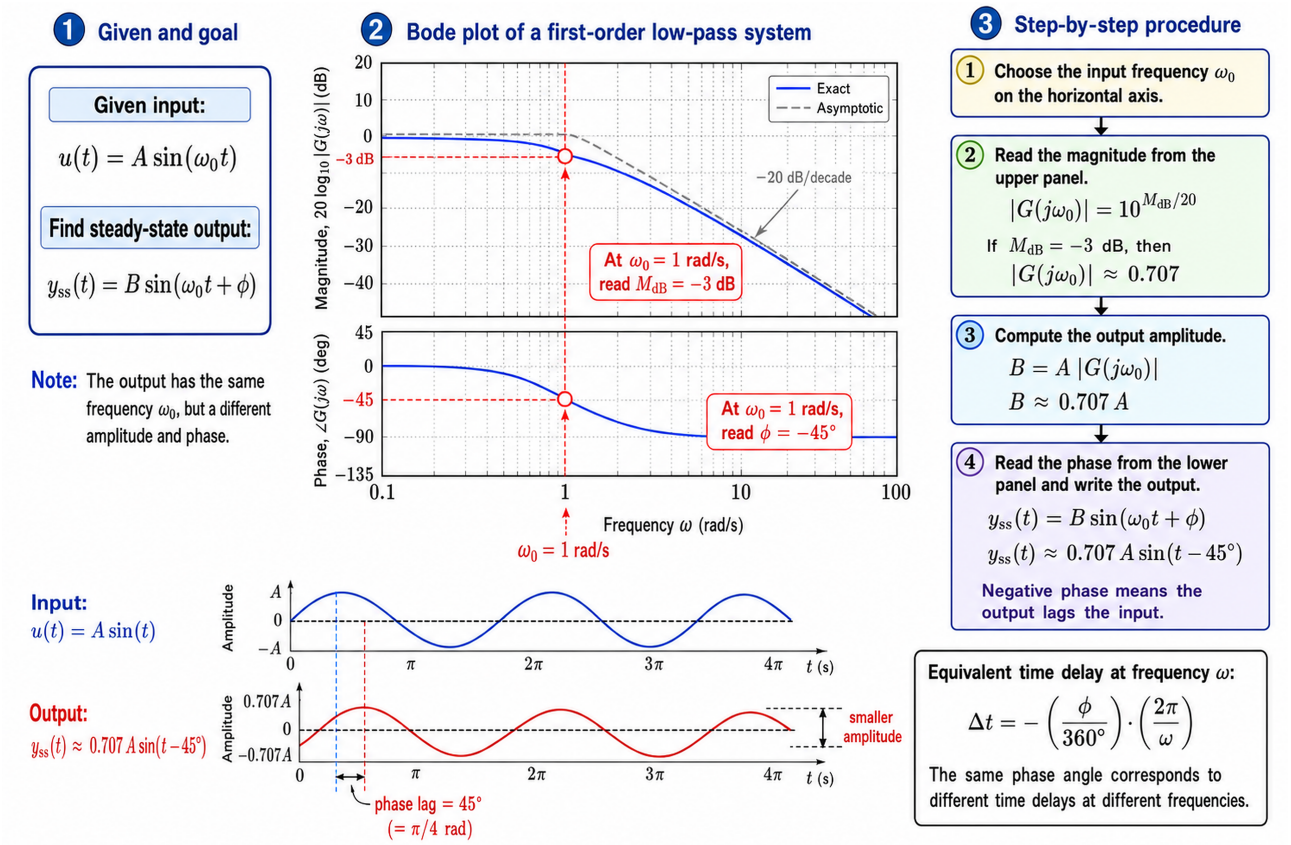


Figure 9.4: Step-by-step procedure for using a Bode plot to determine the sinusoidal steady-state output. The input and output share the same frequency; only the amplitude and phase change.

9.5.2 Putting a transfer function into Bode form

Before drawing a Bode plot, write the transfer function using factors of the form

$$1 + \frac{s}{a}, \qquad 1 + 2\zeta\frac{s}{\omega_n} + \left(\frac{s}{\omega_n}\right)^2, \qquad s.$$

The constants a and ω_n are the **break frequencies**. This conversion is essential because the Bode sketch is drawn from the break frequencies, and each factor contributes independently to the overall magnitude and phase.

Example 9.5: Converting polynomials into Bode form

Consider

$$G(s) = \frac{5(s+2)}{s(s+10)(s+50)}.$$

Convert each polynomial factor by extracting the constant:

$$s+2 = 2\left(1+\frac{s}{2}\right), \quad s+10 = 10\left(1+\frac{s}{10}\right), \quad s+50 = 50\left(1+\frac{s}{50}\right).$$

Therefore

$$G(s) = \frac{5\cdot 2}{10\cdot 50}\cdot\frac{1+s/2}{s\,(1+s/10)(1+s/50)} = 0.02\,\frac{1+s/2}{s\,(1+s/10)(1+s/50)}.$$

The Bode-form gain is $K_B = 0.02$, giving a low-frequency starting magnitude of $20\log_{10}(0.02) = -34$ dB (before accounting for the integrator slope). The zero break frequency is 2 rad/s, and the pole break frequencies are 10 and 50 rad/s.

Common Bode-form mistake

A common mistake is to sketch from $s + 10$ as if the gain were 1. The factor $s + 10$ contains the constant 10, and ignoring it shifts the entire magnitude plot by 20 dB. Always extract the constant from every factor before sketching.

9.5.3 Basic Bode building blocks

Multiplication of transfer functions becomes addition in logarithmic magnitude, so complicated Bode plots are obtained by adding the dB magnitudes and phases of simple factors:

$$20\log_{10}|G_1(j\omega)\,G_2(j\omega)| = 20\log_{10}|G_1(j\omega)| + 20\log_{10}|G_2(j\omega)|.$$

Every rational transfer function is a product of a few standard building blocks.

Constant gain

For $G(s) = K$, the magnitude is $20\log_{10}|K|$ dB at all frequencies. If $K > 0$, the phase is 0°. If $K < 0$, the phase is -180°.

Integrator and differentiator

For an integrator $G(s) = 1/s$, we have $G(j\omega) = 1/(j\omega)$, so

$$20\log_{10}|G(j\omega)| = -20\log_{10}\omega, \qquad \angle G(j\omega) = -90^\circ.$$

The magnitude is a straight line with slope -20 dB/decade passing through 0 dB at $\omega = 1$ rad/s. The phase is a constant -90° at all frequencies.

For a differentiator $G(s) = s$, the magnitude slope is +20 dB/decade and the phase is $+90^\circ$ (Figure 9.5). Each additional integrator or differentiator adds another ± 20 dB/decade to the slope and another $\pm 90^\circ$ to the phase.

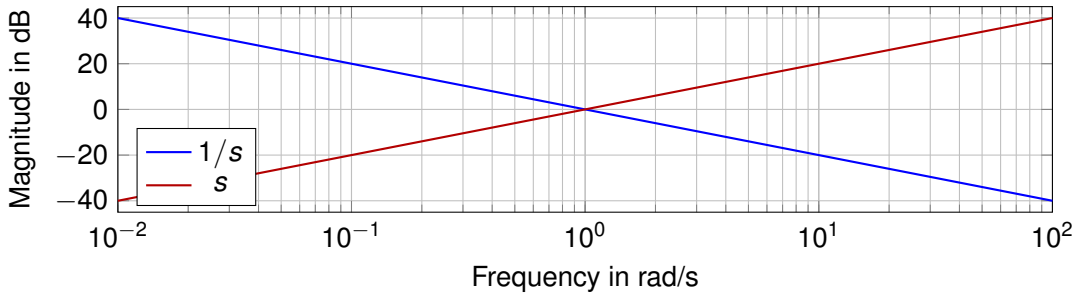


Figure 9.5: Magnitude plots of an integrator and a differentiator. These are already straight-line asymptotes, so the exact and asymptotic plots are identical.

First-order pole

A first-order pole in Bode form is

$$G_p(s) = \frac{1}{1 + s/a}.$$

The break frequency is a. Its exact frequency response is

$$G_p(j\omega) = \frac{1}{1 + j\omega/a},$$

giving

$$20\log_{10}|G_p(j\omega)| = -10\log_{10}\left(1 + \left(\frac{\omega}{a}\right)^2\right), \qquad \angle G_p(j\omega) = -\tan^{-1}\left(\frac{\omega}{a}\right).$$

The asymptotic magnitude approximation is straightforward: the magnitude is 0 dB for $\omega \ll a$ (the factor has not yet "turned on"), and it drops at -20 dB/decade for $\omega \gg a$. The transition between these two regimes happens at the break frequency $\omega = a$, where the exact magnitude is -3 dB while the asymptote still reads 0 dB. This 3 dB error at the corner is the largest discrepancy between the exact and asymptotic curves.

The phase transitions more gradually. A useful hand-sketch rule places the phase at 0° until one decade below the break frequency ($0.1a$), draws a straight line from 0° to -90° between $0.1a$ and $10a$, and holds -90° after $10a$. The exact phase at the break frequency itself is -45°. Figure 9.6 illustrates both the magnitude and phase plots for a pole at $a = 10$ rad/s, showing how the asymptotic straight-line approximation compares with the exact curves.

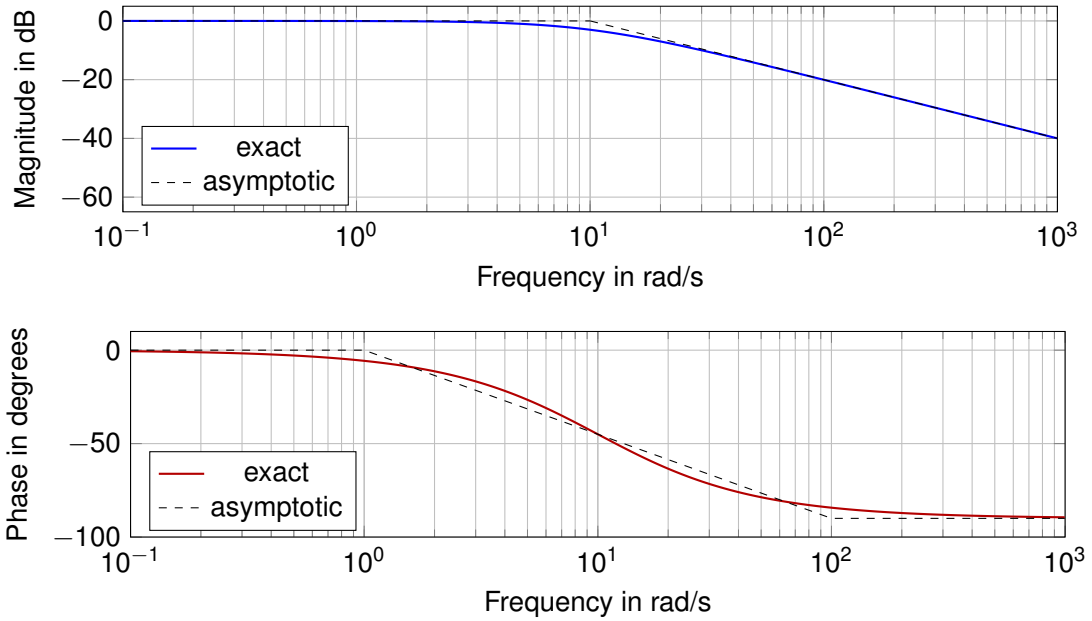


Figure 9.6: Exact and asymptotic Bode plots for a first-order pole $1/(1 + s/10)$. The break frequency is $a = 10$ rad/s. At that frequency, the exact magnitude is -3 dB and the exact phase is -45°. The phase settles to -90° well above $10a = 100$ rad/s.

First-order zero

A first-order zero in Bode form is $G_z(s) = 1 + s/a$. Its magnitude and phase are the mirror images of the first-order pole:

$$20\log_{10}|G_z(j\omega)| = 10\log_{10}\left(1+\left(\frac{\omega}{a}\right)^2\right), \qquad \angle G_z(j\omega) = +\tan^{-1}\left(\frac{\omega}{a}\right).$$

The asymptotic magnitude is 0 dB before the break frequency and +20 dB/decade after it. The phase rises from 0° to +90° between $0.1a$ and $10a$. Figure 9.7 confirms the mirror-image relationship with the first-order pole.

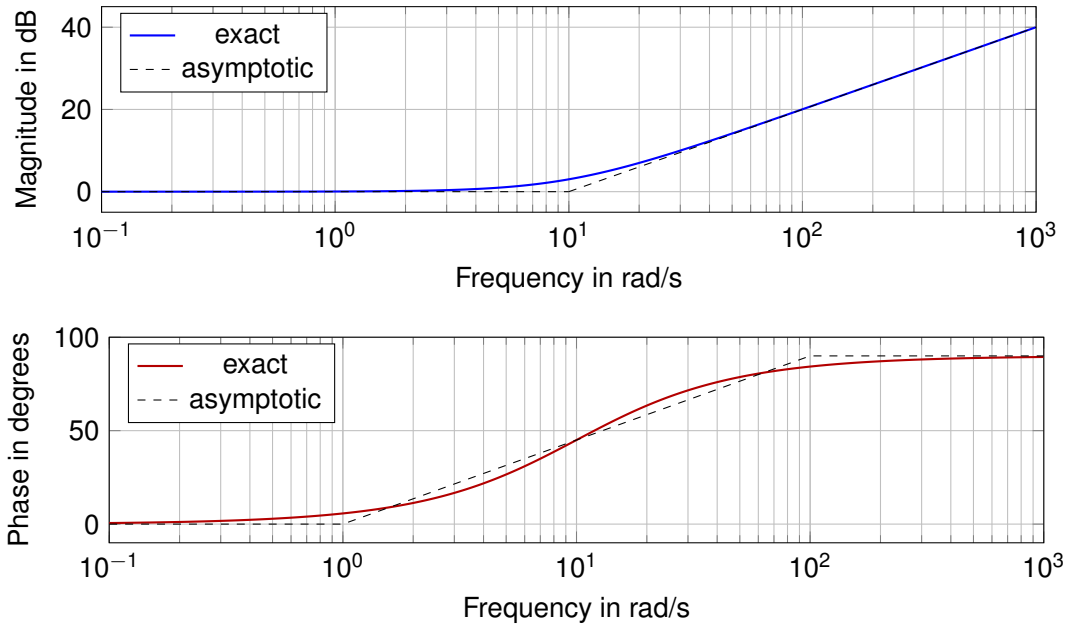


Figure 9.7: Exact and asymptotic Bode plots for a first-order zero $1 + s/10$. The phase settles to +90° above $10a = 100$ rad/s.

Second-order pole pair

A standard second-order pole pair is written as

$$G_{2p}(s) = \frac{1}{1 + 2\zeta(s/\omega_n) + (s/\omega_n)^2}.$$

The break frequency is the natural frequency ω_n. The exact magnitude is

$$20\log_{10}|G_{2p}(j\omega)| = -10\log_{10}\left[\left(1-\left(\frac{\omega}{\omega_n}\right)^2\right)^2 + \left(2\zeta\frac{\omega}{\omega_n}\right)^2\right].$$

The asymptotic magnitude is 0 dB for $\omega < \omega_n$ and drops at -40 dB/decade for $\omega > \omega_n$. The phase moves from 0° to $-180°$, passing through $-90°$ at ω_n.

The critical feature is the Resonant peak. When the damping ratio ζ is small, the exact magnitude can rise well above 0 dB near ω_n. The peak magnitude is approximately $1/(2\zeta)$, or $-20\log_{10}(2\zeta)$ dB. For example, $\zeta = 0.1$ gives a peak of about 14 dB. The asymptotic plot does *not* show this peak, so the exact response must always be checked for lightly damped systems. Figure 9.8 shows an example with $\zeta = 0.4$, where a moderate peak is visible near ω_n.

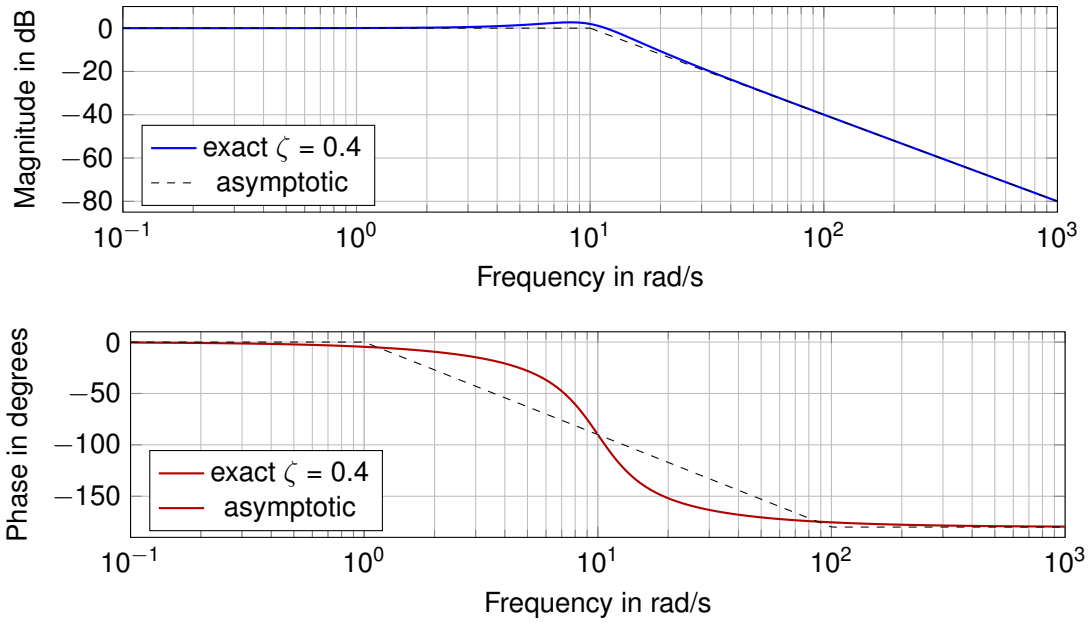


Figure 9.8: Exact and asymptotic Bode plots for a second-order pole pair with $\omega_n = 10$ rad/s and $\zeta = 0.4$. The phase settles to -180° well above ω_n.

Figure 9.9 shows how both the magnitude and phase change as ζ varies from 0.05 to 1.0. Two trends are immediately visible. First, the resonant peak grows dramatically as ζ decreases: at $\zeta = 0.05$ the peak is about 20 dB, while at $\zeta = 0.7$ it has essentially disappeared. Second, the phase transition near ω_n becomes increasingly abrupt for small ζ — at $\zeta = 0.05$ the phase drops almost vertically through -90°, whereas at $\zeta = 1.0$ the transition is gradual and spread over several decades. In all cases, the asymptotic approximation (dashed black) ignores these damping-dependent effects entirely.

Second-order zero pair

A second-order zero pair $G_{2z}(s) = 1 + 2\zeta(s/\omega_n) + (s/\omega_n)^2$ has the opposite frequency-response characteristics of a second-order pole pair: its magnitude and phase contributions are equal in size but opposite in sign. The asymptotic magnitude is 0 dB before ω_n and +40 dB/decade after ω_n. The phase moves from 0° to $+180^\circ$. If the damping is light, there is a notch (dip) instead of a peak near ω_n. Figure 9.10 shows the magnitude plot for this case.

Figure 9.11 shows the mirror-image behaviour across a range of damping ratios: the notch deepens and the phase transition sharpens as ζ decreases, exactly the inverse of the pole pair.

Listing 9.2 shows how to generate the exact Bode plots for all four standard factors in MATLAB.

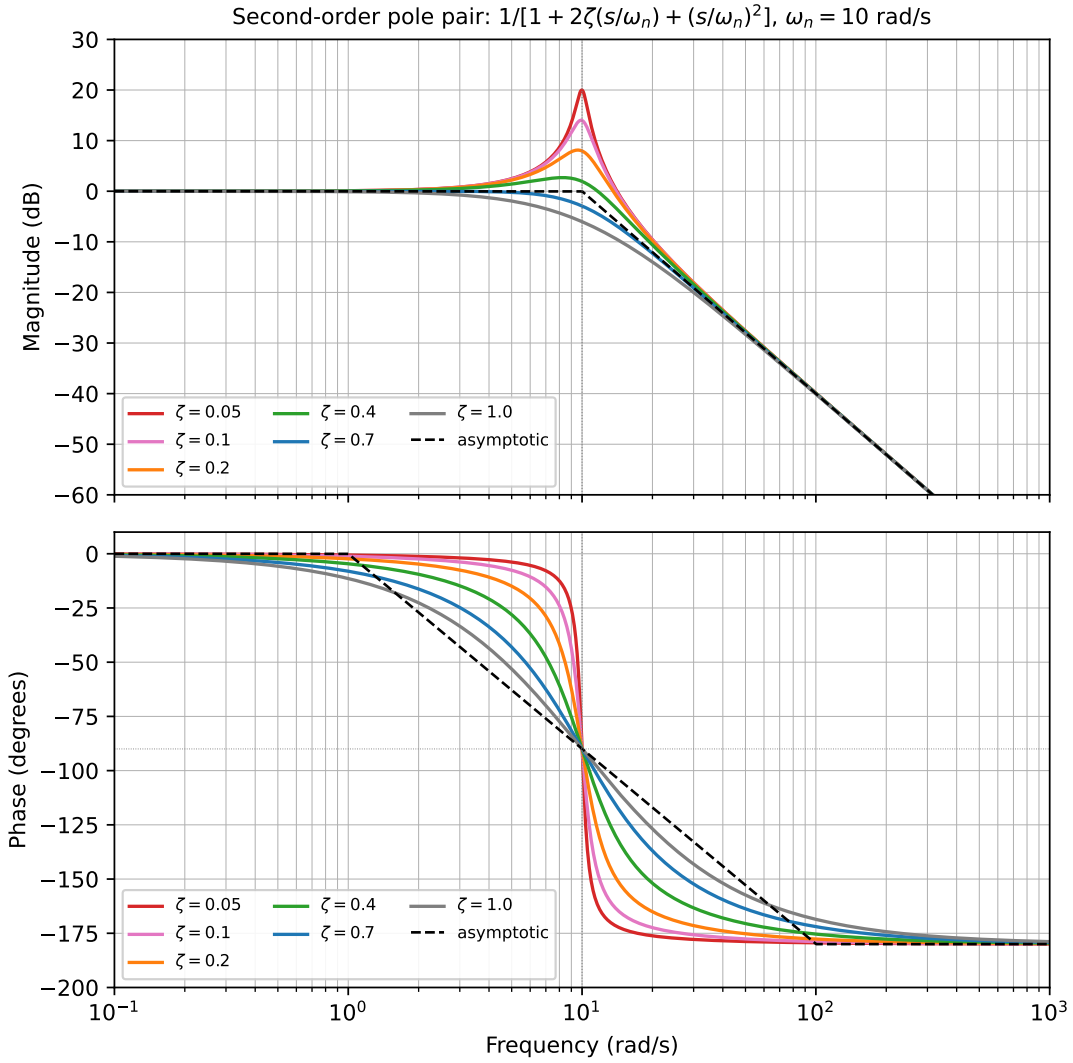


Figure 9.9: Effect of damping ratio on the Bode plot of a second-order pole pair with ω_n = 10 rad/s. As ζ decreases, the resonant peak grows and the phase transition near ω_n becomes steeper. The asymptotic approximation (dashed) is independent of ζ.

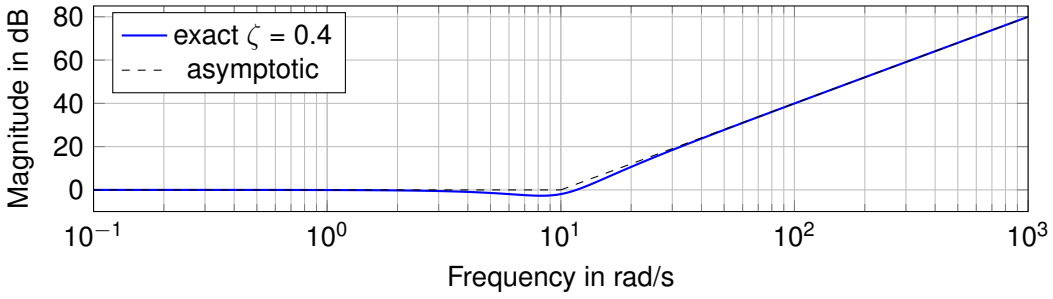


Figure 9.10: Exact and asymptotic magnitude plots for a second-order zero pair.

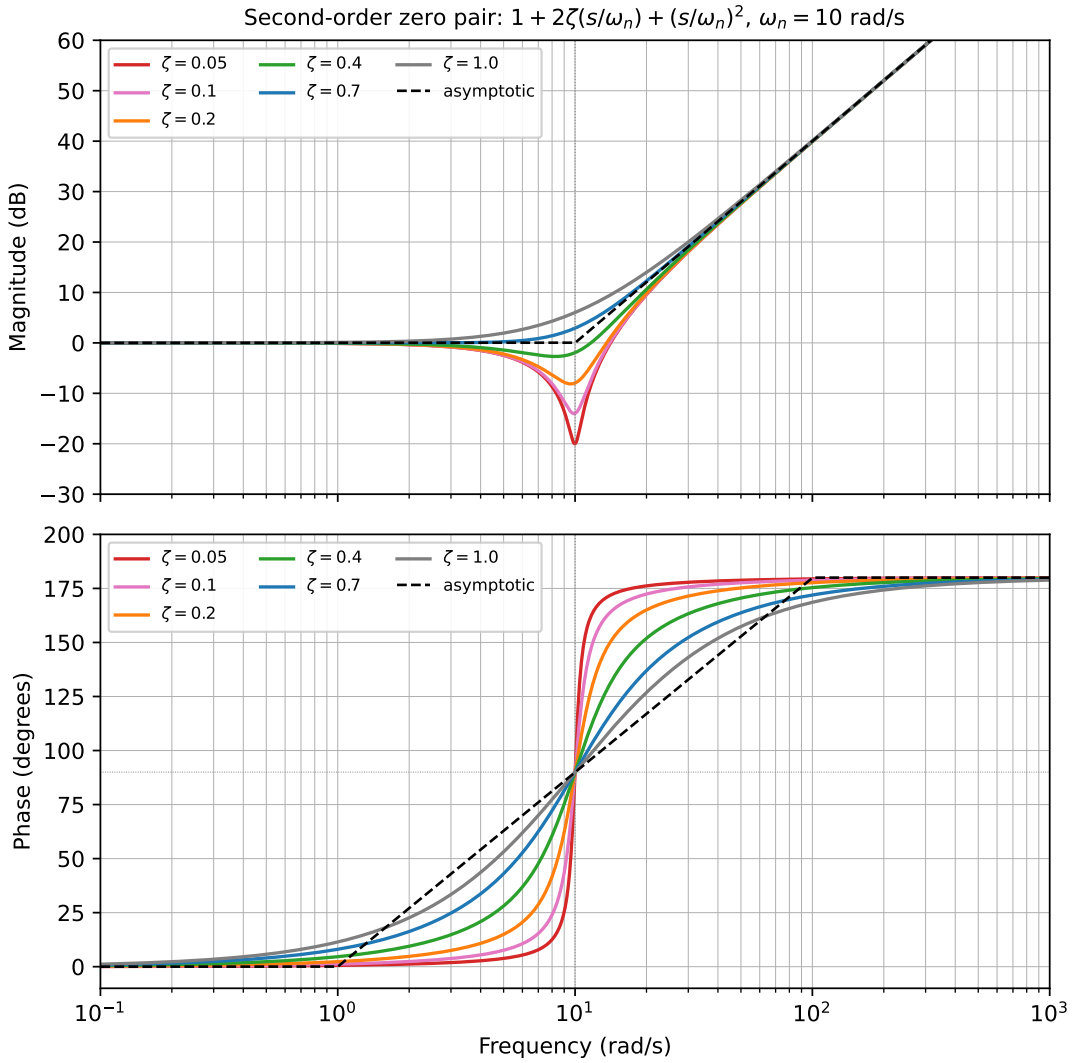


Figure 9.11: Effect of damping ratio on the Bode plot of a second-order zero pair with ω_n = 10 rad/s. As ζ decreases, the anti-resonance notch deepens and the phase transition near ω_n becomes steeper.

```
s = tf('s');
a = 10;
wn = 10;
zeta = 0.4;

Gp1 = 1/(1+s/a);                          % first-order pole
Gz1 = 1+s/a;                              % first-order zero
Gp2 = 1/(1+2*zeta*(s/wn)+(s/wn)^2);       % second-order pole pair
Gz2 = 1+2*zeta*(s/wn)+(s/wn)^2;           % second-order zero pair

figure
bode(Gp1, Gz1, Gp2, Gz2), grid on
legend('1st-order pole','1st-order zero', ...
       '2nd-order pole pair','2nd-order zero pair')
```

Listing 9.2: MATLAB code for exact Bode plots of the standard factors.

9.5.4 Step-by-step asymptotic Bode construction

The practical hand-sketch procedure is as follows.

1. Convert the transfer function into Bode form.
2. Mark every break frequency on the logarithmic axis.
3. Start the magnitude at the low-frequency value, including the constant gain and any poles or zeros at the origin.
4. Move from left to right. At each first-order zero, add +20 dB/decade to the slope. At each first-order pole, add -20 dB/decade. At each second-order zero, add +40 dB/decade. At each second-order pole, add -40 dB/decade.
5. Sketch the phase by adding the phase contributions of all factors.
6. Compare the final hand sketch with the exact Bode plot from MATLAB.

Example 9.1: Complete asymptotic Bode sketch

Consider

$$G(s) = \frac{100(1 + s/2)}{s(1 + s/10)(1 + s/50)}.$$

Step 1: Identify the components. The Bode-form gain is $K_B = 100$ (40 dB). The factors are:

Factor	Type	Break frequency
$K_B = 100$	constant gain	—
$1/s$	integrator	—
$1 + s/2$	zero	2 rad/s
$1/(1 + s/10)$	pole	10 rad/s
$1/(1 + s/50)$	pole	50 rad/s

Step 2: Determine the slope in each frequency range. Starting from the integrator slope of -20 dB/decade:

Range	Slope change	Net slope	Reason
$\omega < 2$	$---$	-20 dB/dec	integrator only
$2 < \omega < 10$	$+20$	0 dB/dec	zero at 2 cancels slope
$10 < \omega < 50$	-20	-20 dB/dec	pole at 10
$\omega > 50$	-20	-40 dB/dec	pole at 50

Step 3: Fix the vertical position. At $\omega = 1$ rad/s (before any break frequency), the magnitude is

$$20 \log_{10}\left(\frac{100}{1}\right) = 40 \text{ dB}.$$

From this anchor point, we draw the asymptotic segments with the slopes determined above.

Step 4: Sketch the phase. The integrator contributes -90° at all frequencies. The zero at 2 adds positive phase between 0.2 and 20 rad/s. The poles at 10 and 50 each subtract phase in their respective transition bands. The total phase starts near -90° and eventually approaches -180° at high frequencies. Figure 9.12 compares the resulting asymptotic sketch with the exact Bode plot, and Figure 9.13 shows the individual factor contributions that build up the cumulative asymptote.

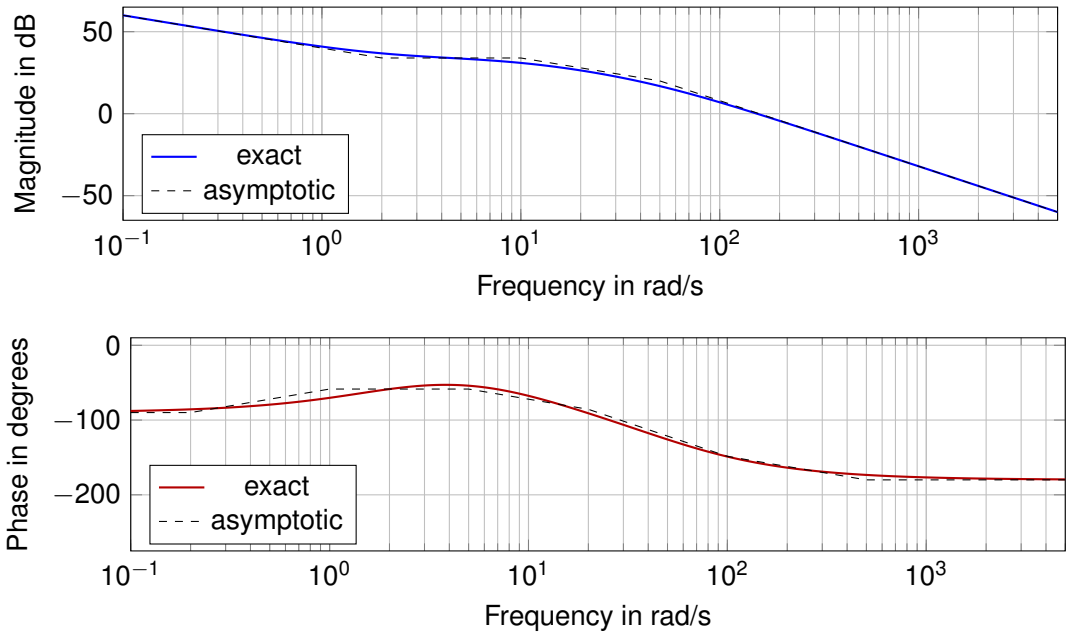


Figure 9.12: Exact and asymptotic Bode plots for $G(s) = 100(1 + s/2)/(s(1 + s/10)(1 + s/50))$.

Figure 9.13 shows how each individual factor contributes its own asymptotic magnitude and phase segment, and how these are summed to produce the cumulative asymptotic Bode plot. Comparing the cumulative asymptote (dashed black) with the exact response confirms that the hand-sketch procedure captures the essential shape of the frequency response.

The exact Bode plot can be verified with Listing 9.3.

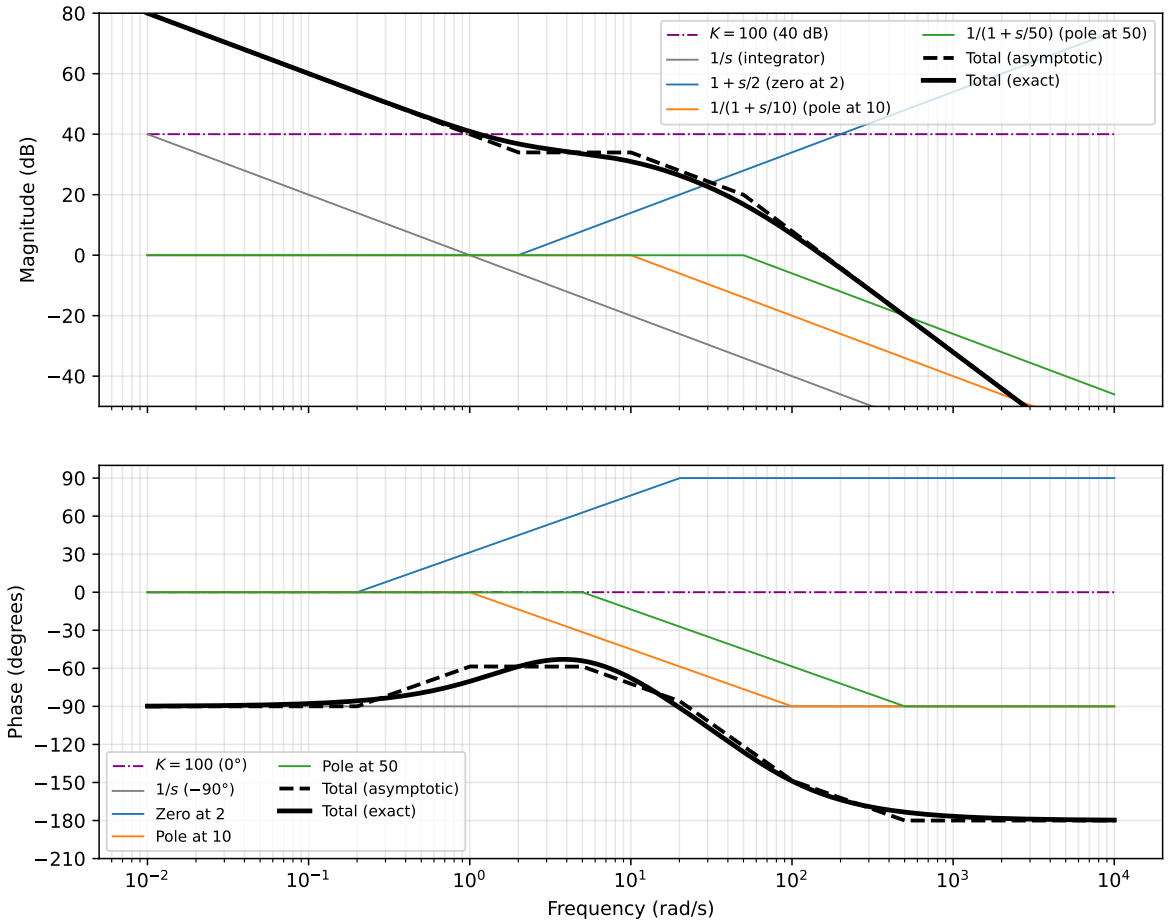


Figure 9.13: Asymptotic Bode construction for $G(s) = 100(1 + s/2)/(s(1 + s/10)(1 + s/50))$. Thin coloured lines show the individual factor contributions; the dashed black line is the cumulative asymptotic approximation; the solid blue/red line is the exact response.

```
s = tf('s');
G = 100*(1+s/2)/(s*(1+s/10)*(1+s/50));
bode(G), grid on
```

Listing 9.3: MATLAB code for the exact Bode plot of the complete example.

Example 9.6: A higher-order Bode plot

Consider

$$G(s) = \frac{50(s+3)}{s(s+1)(s^2+4s+100)}.$$

To sketch by hand, first convert to Bode form. The numerator factor is $s+3 = 3(1+s/3)$. The denominator quadratic is $s^2+4s+100 = 100(1+0.4\cdot s/10+(s/10)^2)$, giving $\omega_n = 10$ and $\zeta = 0.2$. The denominator constants are 1 (from $s+1$) and 100 (from the quadratic), so the Bode-form gain is $K_B = 50\cdot 3/(1\cdot 100) = 1.5$, or 3.5 dB.

The slope pattern is: -20 dB/dec (integrator) up to $\omega = 1$, then -40 dB/dec (pole at 1), then -20 dB/dec (zero at 3), then -60 dB/dec (second-order poles at 10). The lightly damped poles at $\omega_n = 10$ will produce a resonant peak that the asymptotic sketch misses.

Figure 9.14 shows each factor's individual contribution (coloured curves) together with the cumulative asymptotic and exact Bode plots (black dashed and solid). The resonant peak near $\omega_n = 10$ rad/s in the exact total is clearly absent from the asymptotic total, illustrating why the exact plot must always be checked for lightly damped modes. Listing 9.4 reproduces the exact plot and computes the stability margins.

```
s = tf('s');
G = 50*(s+3)/(s*(s+1)*(s^2 + 4*s + 100));
figure
margin(G), grid on
[Gm, Pm, Wcg, Wcp] = margin(G);
fprintf('Gain margin = %.1f dB, Phase margin = %.1f deg\n', ...
        20*log10(Gm), Pm)
```

Listing 9.4: MATLAB code for a higher-order Bode plot with stability margins.

9.6 Interpreting Bode Plots

The **low-frequency gain** tells us how strongly the system responds to slowly varying signals — high loop gain at low frequencies improves tracking and disturbance rejection. The **high-frequency gain** tells us how much high-frequency content passes through — high gain at high frequencies amplifies sensor noise, so a fast roll-off is usually desirable.

The Bandwidth is the frequency range over which the closed-loop system responds effectively. For many low-pass closed-loop systems, the bandwidth is approximately the -3 dB frequency of the closed-loop magnitude. A wider bandwidth means faster response, but also greater sensitivity to high-frequency noise.

A **resonant peak** is a bump in the magnitude plot, usually caused by lightly damped poles. In the time domain, a resonant peak corresponds to overshoot and oscillation. The peak height is related to the damping ratio: roughly $M_p \approx 1/(2\zeta)$ for a standard second-order system.

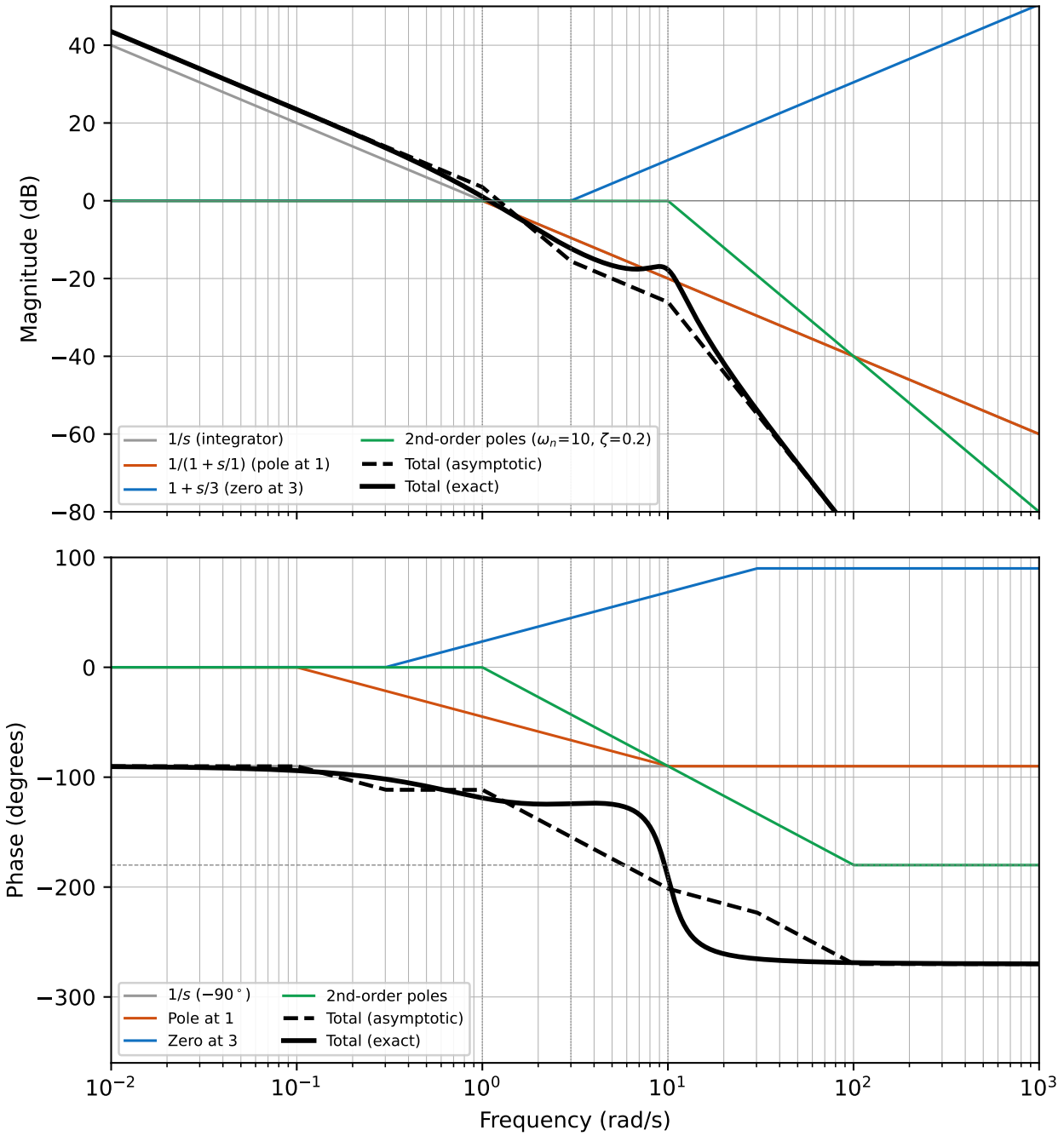


Figure 9.14: Construction of the Bode plot for $G(s) = 50(s+3)/[s(s+1)(s^2+4s+100)]$. Each factor's magnitude and phase contribution is shown individually (coloured curves), with asymptotic approximations as dashed lines. The thick black curves are the total exact and asymptotic Bode plots. Vertical dotted lines mark the break frequencies at 1, 3, and 10 rad/s.

The Gain crossover frequency ω_{gc} is where the open-loop magnitude passes through 0 dB. The Phase crossover frequency ω_{pc} is where the open-loop phase reaches -180°. These two frequencies are central to stability margin analysis, which is covered in the next section.

Example 9.7: Low-pass and resonant interpretation

The standard second-order system

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} = \frac{1}{1 + 2\zeta(s/\omega_n) + (s/\omega_n)^2}$$

has a static gain of 1 (0 dB) and rolls off at -40 dB/decade above ω_n. When $\zeta = 0.7$, the exact magnitude decreases monotonically — there is no resonant peak and the time-domain step response has about 5% overshoot. When $\zeta = 0.2$, a sharp resonant peak of about 8 dB appears near ω_n, and the step response oscillates significantly. The Bode plot makes this difference immediately visible (see Figure 9.15).

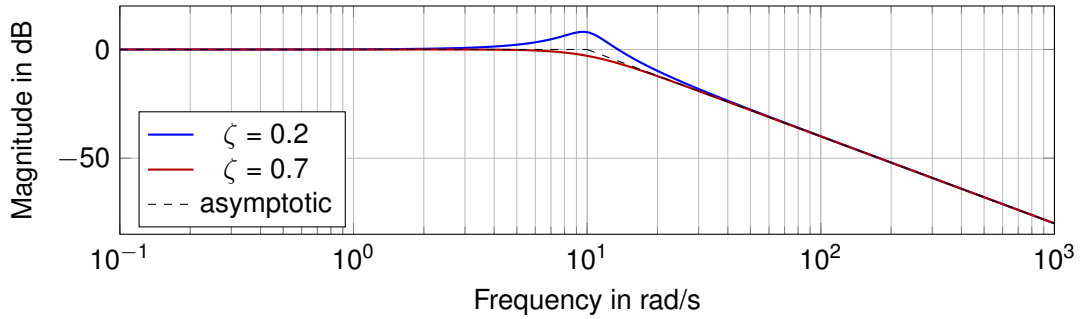


Figure 9.15: Exact second-order magnitude plots compared with the same asymptotic sketch. The asymptote gives the slope, but it does not reveal the resonant peak.

Figure 9.16 shows the corresponding closed-loop step responses. With $\zeta = 0.2$, the resonant peak in the Bode plot translates directly into large overshoot and prolonged oscillation. With $\zeta = 0.7$, the monotonic magnitude roll-off corresponds to a well-damped response with only about 5% overshoot.

Listing 9.5 reproduces these plots in MATLAB, making it easy to experiment with other damping values.

```
s = tf('s');
wn = 10;
G1 = wn^2/(s^2 + 2*0.2*wn*s + wn^2);
G2 = wn^2/(s^2 + 2*0.7*wn*s + wn^2);
bode(G1, G2), grid on
legend('zeta = 0.2','zeta = 0.7')
```

Listing 9.5: MATLAB code for comparing second-order Bode plots with different damping.

9.7 Stability Margins from Bode Plots

Frequency-domain stability analysis uses the Loop transfer function

$$L(s) = C(s)\,G(s).$$

For unity negative feedback, the closed loop becomes unstable if $L(j\omega) = -1$ for some real ω — that is, when the loop gain has magnitude 1 (0 dB) and phase -180° simultaneously. The stability margins measure how far the system is from this condition.

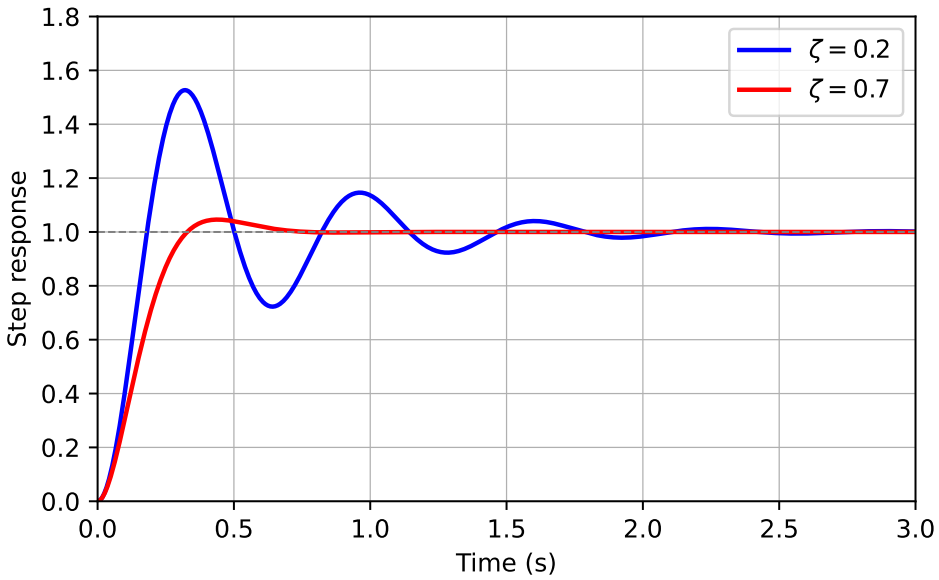


Figure 9.16: Step responses of the standard second-order system for $\zeta = 0.2$ and $\zeta = 0.7$ ($\omega_n = 10$ rad/s). The lightly damped case exhibits significant overshoot and oscillation, consistent with the resonant peak in Figure 9.15.

Definition 9.3: Gain crossover and phase margin

The **gain crossover frequency** ω_{gc} is the frequency where

$$|L(j\omega_{gc})| = 1 \quad \text{(i.e. 0 dB)}.$$

The Phase margin is

$$PM = 180^\circ + \angle L(j\omega_{gc}).$$

It measures how much additional phase lag would be needed to reach -180° at the frequency where the gain is unity.

Definition 9.4: Phase crossover and gain margin

The **phase crossover frequency** ω_{pc} is the frequency where

$$\angle L(j\omega_{pc}) = -180^\circ.$$

The Gain margin is

$$GM = \frac{1}{|L(j\omega_{pc})|}, \qquad GM_{\text{dB}} = -20\log_{10}|L(j\omega_{pc})|.$$

It measures how much the loop gain could increase before the magnitude reaches 0 dB at the frequency where the phase is already -180°.

Figure 9.17 shows both margins on a single Bode plot. The phase margin is read at the gain crossover frequency ω_{gc}: it is the vertical distance between the phase curve and the -180° line on the phase plot. The gain margin is read at the phase crossover frequency ω_{pc}: it is the vertical distance between the magnitude curve and the 0 dB line on the magnitude plot. A positive phase margin and a positive gain margin (in dB) both indicate stability; either one reaching zero means the system is on the boundary of instability.

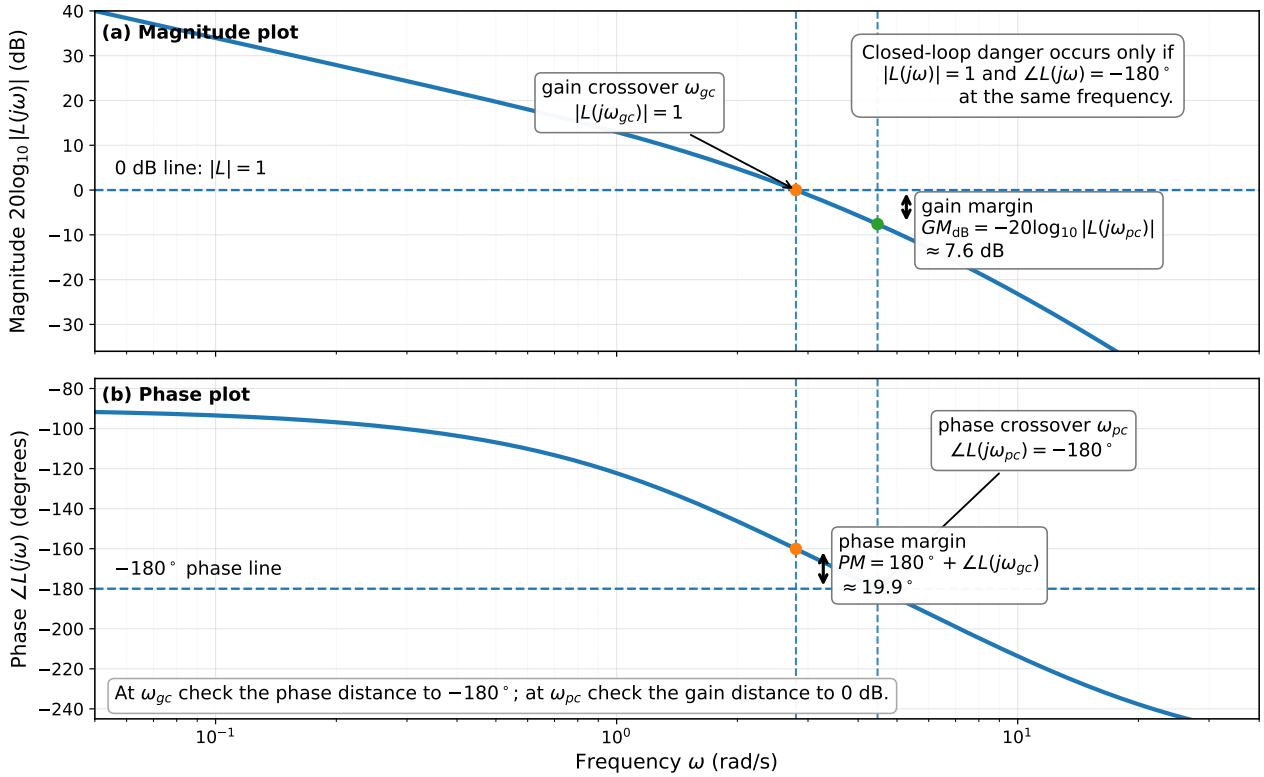


Figure 9.17: Gain and phase margins on the Bode plot of $L(s)$. At ω_{gc}, the magnitude is 0 dB and the phase margin measures how far the phase is from $-180°$. At ω_{pc}, the phase is $-180°$ and the gain margin measures how far the magnitude is below 0 dB. Both margins must be positive for a stable closed loop.

Typical engineering targets are a phase margin of 30° to 60° and a gain margin of at least 6 dB.

9.7.1 Physical interpretation of the crossover frequencies

The gain crossover frequency ω_{gc} determines the *speed* of the closed-loop system. Below ω_{gc}, the loop gain is much greater than one, so feedback forces the output to follow the reference. Above ω_{gc}, the loop gain is much less than one, so feedback is essentially inactive. The closed-loop bandwidth is therefore closely related to ω_{gc}: roughly $\omega_{BW} \approx (1\text{–}2)\,\omega_{gc}$.

A higher ω_{gc} means a faster response over a wider frequency band, but also more high-frequency sensor noise passing into the control signal. This is the fundamental trade-off of feedback: speed versus noise sensitivity and robustness.

At ω_{pc}, the open-loop phase is exactly $-180°$ — the feedback signal arrives in phase with the reference (positive feedback). If the magnitude were also 0 dB at that frequency, the system would sustain oscillations. The gain margin tells us how far the magnitude is below 0 dB at ω_{pc}.

9.7.2 What the margins tell you about closed-loop behaviour

A small phase margin ($PM < 20°$) means the system is "almost oscillating" — the step response will show large overshoot and poor damping. At $PM = 0°$, the system oscillates indefinitely. A phase margin of 60°–70° gives a well-damped response, while $PM > 80°$ usually means the system responds too slowly.

The gain margin provides robustness against plant gain uncertainty. A gain margin of 6 dB means the actual gain can be up to twice the nominal value before instability — useful insurance against modelling errors. Table 9.1 summarises the practical guidelines.

Margin range	Closed-loop character	Typical use
$PM < 20^\circ$	Highly oscillatory, fragile	Unacceptable
$20^\circ < PM < 30^\circ$	Oscillatory, marginal robustness	Sometimes acceptable
$30^\circ < PM < 60^\circ$	Moderate damping, good robustness	Recommended range
$PM > 60^\circ$	Well damped, possibly slow	Good if speed is adequate
$GM < 3$ dB	Almost no gain tolerance	Unacceptable
$GM = 6$ dB	Factor-of-two gain tolerance	Minimum target
$GM > 12$ dB	Very robust to gain changes	Conservative

Table 9.1: Practical guidelines for phase margin and gain margin.

9.7.3 Phase margin and gain margin calculations

The following two examples illustrate both the phase margin and the gain margin calculations in detail.

Example 9.2: Phase margin calculation

Let

$$L(s) = \frac{10}{s(s+1)}.$$

Step 1: Find the gain crossover frequency. We need $|L(j\omega)| = 1$:

$$|L(j\omega)| = \frac{10}{\omega\sqrt{1+\omega^2}} = 1 \quad \Longrightarrow \quad \omega^2(1+\omega^2) = 100.$$

Let $x = \omega^2$. Then $x^2 + x - 100 = 0$, giving $x = (-1 + \sqrt{401})/2 \approx 9.51$, so

$$\omega_{gc} = \sqrt{9.51} \approx 3.08 \text{ rad/s}.$$

Step 2: Compute the phase at ω_{gc}.

$$\angle L(j3.08) = -90^\circ - \tan^{-1}(3.08) = -90^\circ - 72.0^\circ = -162.0^\circ.$$

Step 3: Calculate the phase margin.

$$PM = 180^\circ + (-162.0^\circ) = 18.0^\circ.$$

This is a small phase margin. The closed-loop step response will be highly oscillatory, with significant overshoot and slow settling. Figure 9.18 shows the Bode plot with the phase margin annotated.

Example 9.3: Gain margin calculation

Using the same loop transfer function $L(s) = 10/(s(s+1))$:

Step 1: Find the phase crossover frequency. We need $\angle L(j\omega) = -180^\circ$:

$$\angle L(j\omega) = -90^\circ - \tan^{-1}(\omega) = -180^\circ \quad \Longrightarrow \quad \tan^{-1}(\omega) = 90^\circ.$$

This requires $\omega \to \infty$, so strictly speaking there is no finite phase crossover frequency. The phase approaches -180° but never reaches it. Therefore the gain margin is infinite:

$$GM = \infty.$$

This makes physical sense: the system has only two poles (a type-1 system), so the phase saturates at -180° asymptotically. No finite gain increase can make the loop unstable. The stability limitation comes entirely from the phase margin.

Contrast: If the plant had a third pole, e.g. $L(s) = 10/(s(s+1)(s/5+1))$, the phase would reach -180° at a finite frequency, and the gain margin would be finite and important.

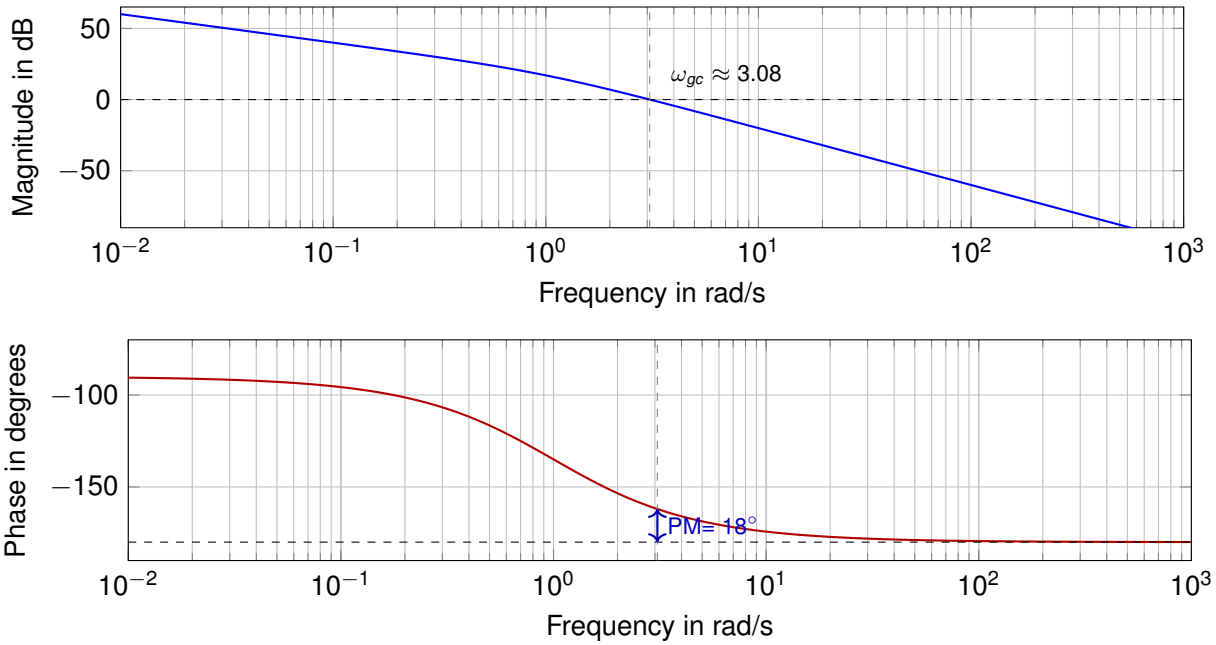


Figure 9.18: Bode plot of $L(s) = 10/(s(s+1))$ showing the phase margin of 18° at $\omega_{gc} \approx 3.08$ rad/s.

Listing 9.6 shows how to compute these margins in MATLAB using the `margin` command.

```
s = tf('s');
L = 10/(s*(s+1));
margin(L), grid on
[Gm, Pm, Wcg, Wcp] = margin(L);
fprintf('GM = %.1f dB, PM = %.1f deg at wgc = %.2f rad/s\n', ...
        20*log10(Gm), Pm, Wcp)
```

Listing 9.6: MATLAB code for gain and phase margins.

Remark 9.2

The Bode margins are computed from the *open-loop* transfer function $L(s) = C(s)G(s)$, not from the closed-loop transfer function. The closed-loop is what we want to stabilise; the open-loop Bode plot is the diagnostic tool.

9.8 MATLAB for Bode Analysis

The most commonly used MATLAB commands for Bode analysis are summarised in Table 9.2.

Command	Purpose
`bode(G)`	Plot magnitude and phase
`margin(G)`	Plot Bode diagram with gain and phase margins
`[Gm,Pm,Wcg,Wcp] = margin(G)`	Compute margins numerically
`bandwidth(T)`	Estimate closed-loop bandwidth
`allmargin(G)`	Detailed margin information

Table 9.2: Key MATLAB commands for frequency-domain analysis.

A typical analysis workflow is shown in Listing 9.7.

```
s = tf('s');
G = 5/((s+1)*(s+4));

figure
bode(G), grid on
title('Open-loop Bode plot')

figure
margin(G), grid on
title('Open-loop margins')

T = feedback(G, 1);
bw = bandwidth(T);
info = stepinfo(T);
fprintf('Bandwidth = %.2f rad/s, Settling time = %.2f s\n', ...
        bw, info.SettlingTime)
```

Listing 9.7: MATLAB workflow for frequency-domain analysis.

9.9 Connecting Frequency Domain and Time Domain

Control specifications are often stated in the time domain (overshoot, settling time, steady-state error), but frequency-domain tools work with bandwidth, crossover frequency, phase margin, and loop gain. The designer must translate between the two domains. This translation relies on second-order approximations — for higher-order systems, MATLAB verification of the final design is essential.

9.9.1 The second-order bridge

As we saw in Section 6.2, the standard second-order closed-loop transfer function is

$$T(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2},$$

where ζ is the damping ratio and ω_n is the natural frequency. The key time-domain specifications for a step input are:

$$\text{overshoot:} \quad M_p = e^{-\pi\zeta/\sqrt{1-\zeta^2}} \times 100\%, \qquad \text{settling time (2\%):} \quad t_s \approx \frac{4}{\zeta\omega_n}.$$

These formulas provide the link between the time and frequency domains.

9.9.2 Phase margin and damping ratio

For a second-order system in a unity-feedback configuration, there is an approximate relationship between phase margin and damping ratio.

Engineering rule (sufficient for design):

$$PM \approx 100\,\zeta \quad \text{(degrees)}, \qquad \text{for } 0 < \zeta < 0.7.$$

Exact second-order formula (used when higher accuracy is needed):

$$PM = \tan^{-1}\left(\frac{2\zeta}{\sqrt{\sqrt{1 + 4\zeta^4} - 2\zeta^2}}\right).$$

For design purposes, the simple rule $PM \approx 100\zeta$ is usually sufficient. For example, a specification of no more than 16% overshoot corresponds to $\zeta \geq 0.5$ (from the overshoot formula), which translates to a phase margin target of about 50°. Figure 9.19 illustrates this relationship visually. The left panel shows a family of unit step responses sweeping from low phase margin (red, highly oscillatory) to high phase margin (blue, well damped). The right panel maps phase margin to overshoot percentage, with shaded regions indicating design quality: margins below 25° are generally unacceptable, 45°–70° is the preferred design region, and values above 70° are conservative.

9.9.3 Crossover frequency, bandwidth, and speed

The gain crossover frequency ω_{gc} is closely related to the closed-loop bandwidth ω_{BW}. For a second-order system with moderate damping ($0.3 \leq \zeta \leq 0.7$), the approximation

$$\omega_{\text{BW}} \approx \omega_{gc} \cdot \left(1 + 1.4\zeta + \zeta^2\right)^{1/2}$$

holds reasonably well. For $\zeta \approx 0.5$, this gives $\omega_{\text{BW}} \approx 1.4\,\omega_{gc}$. As a rough engineering rule:

$$\boxed{\omega_{\text{BW}} \approx (1 \text{ to } 2)\,\omega_{gc}.}$$

Once the bandwidth is known, the rise time can be estimated:

$$t_r \approx \frac{1.8}{\omega_{\text{BW}}} \quad (\text{for } 0.4 \leq \zeta \leq 0.8).$$

The settling time is best estimated from the time-domain formula $t_s \approx 4/(\zeta\,\omega_n)$, using $\omega_n \approx \omega_{\text{BW}}/1.4$ for $\zeta \approx 0.5$.

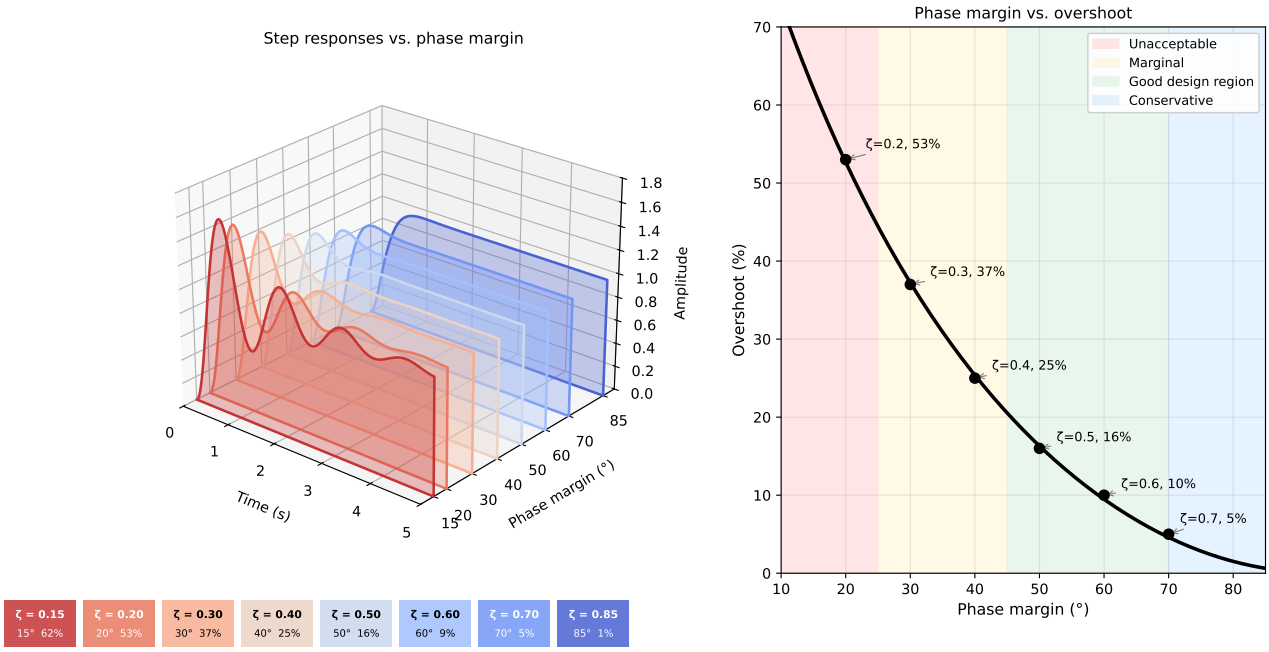


Figure 9.19: Phase margin, damping ratio, and step response character for second-order systems. Left: step responses evolve from oscillatory (red, low *PM*) to well-damped (blue, high *PM*). Right: the overshoot–phase margin curve with design-quality zones.

9.9.4 Steady-state error and low-frequency gain

The steady-state error to polynomial reference inputs depends on the **system type** (the number of free integrators in the loop transfer function $L(s)$) and the corresponding **error constant**. Three error constants cover the most common reference signals, as summarised in Table 9.3.

System type	Integrators	Error constant	Definition	Steady-state error
Type 0	0	K_p (position)	$\lim_{s\to 0} L(s)$	Step: $e_{ss} = \dfrac{1}{1+K_p}$
Type 1	1	K_v (velocity)	$\lim_{s\to 0} s\,L(s)$	Ramp: $e_{ss} = \dfrac{1}{K_v}$
Type 2	2	K_a (acceleration)	$\lim_{s\to 0} s^2 L(s)$	Parabola: $e_{ss} = \dfrac{1}{K_a}$

Table 9.3: System type, error constants, and steady-state error for polynomial reference inputs.

A type-0 system has finite step error and infinite ramp error. A type-1 system has zero step error, finite ramp error, and infinite parabolic error. A type-2 system has zero error for both steps and ramps, and finite parabolic error. The designer must identify the system type and the relevant error constant before setting the low-frequency gain target.

On the Bode plot, the error constants can be read from the low-frequency asymptote of the open-loop magnitude:

- **Type 0:** K_p is the DC gain — read directly from the low-frequency flat region of the magnitude plot.
- **Type 1:** K_v equals $|L(j\omega)|$ extrapolated along the -20 dB/dec integrator slope back to $\omega = 1$ rad/s.
- **Type 2:** K_a equals the magnitude of $|L(j\omega)|$ extrapolated along the -40 dB/dec double-integrator slope back to $\omega = 1$ rad/s.

Example 9.4: Reading error constants from the Bode magnitude plot

Problem: For each of the following unity-feedback loop transfer functions, identify the system type, read the error constant from the Bode magnitude plot, and compute the steady-state error to the appropriate polynomial input.

1. $L_0(s) = \dfrac{20}{(s+1)(s+5)}$
2. $L_1(s) = \dfrac{50}{s(s+10)}$
3. $L_2(s) = \dfrac{900}{s^2(s+30)}$

Solution:

(a) Type 0 — position constant K_p. There are no free integrators, so this is a type-0 system. The relevant reference is a unit step. At very low frequencies, all poles and zeros are inactive and the magnitude settles to a flat value:

$$|L_0(j\omega)|\big|_{\omega\to 0} = \frac{20}{1\cdot 5} = 4 \quad\Longrightarrow\quad K_p = 4 \ (12\ \text{dB}).$$

On the Bode plot (Figure 9.20), K_p is simply the height of the low-frequency flat region (shaded). The steady-state step error is

$$e_{ss} = \frac{1}{1+K_p} = \frac{1}{1+4} = 0.2 \quad (20\%).$$

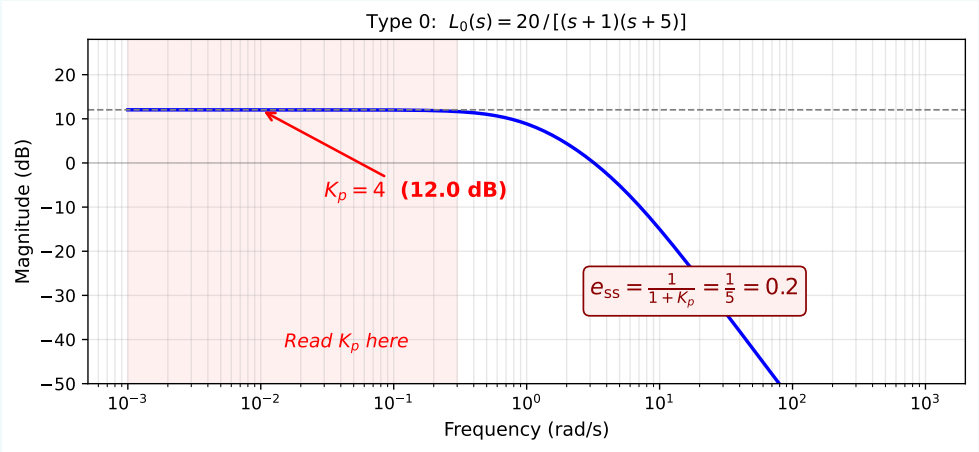


Figure 9.20: Type-0 system: K_p is the DC gain, read from the low-frequency flat region of the magnitude plot.

(b) Type 1 — velocity constant K_v. One free integrator $1/s$ is present, so this is a type-1 system. The relevant reference is a unit ramp. At low frequencies the magnitude plot has a -20 dB/dec slope due to the integrator. Extrapolating this slope back to $\omega = 1$ rad/s gives the intercept:

$$K_v = \lim_{s\to 0} s\,L_1(s) = \frac{50}{10} = 5 \quad\Longrightarrow\quad 20\log_{10}(5) = 14.0\ \text{dB at } \omega = 1.$$

On the Bode plot (Figure 9.21), follow the -20 dB/dec asymptote (dashed red) and read off the value where it crosses $\omega = 1$ rad/s. The steady-state ramp error is

$$e_{ss} = \frac{1}{K_v} = \frac{1}{5} = 0.2.$$

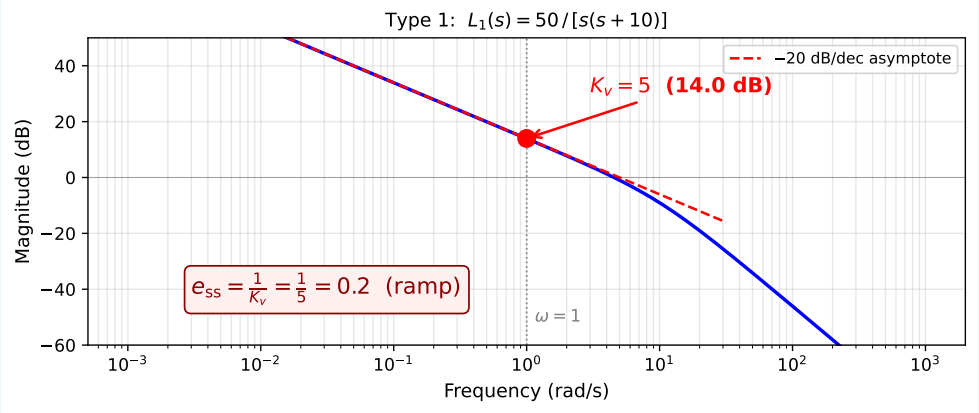


Figure 9.21: Type-1 system: K_v is the value of the -20 dB/dec asymptote extrapolated to $\omega = 1$ rad/s (red dot).

(c) Type 2 — acceleration constant K_a. Two free integrators $1/s^2$ are present, so this is a type-2 system. The relevant reference is a unit parabola. At low frequencies the magnitude plot has a -40 dB/dec slope. Extrapolating this slope back to $\omega = 1$ rad/s:

$$K_a = \lim_{s\to 0} s^2 L_2(s) = \frac{900}{30} = 30 \quad \Longrightarrow \quad 20\log_{10}(30) = 29.5 \text{ dB at } \omega = 1.$$

On the Bode plot (Figure 9.22), follow the -40 dB/dec asymptote (dashed red) to $\omega = 1$ rad/s. The steady-state parabolic error is

$$e_{ss} = \frac{1}{K_a} = \frac{1}{30} \approx 0.033.$$

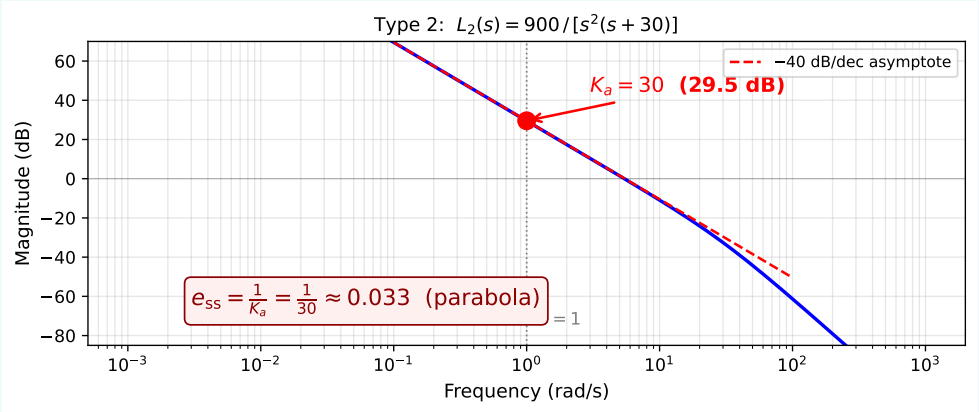


Figure 9.22: Type-2 system: K_a is the value of the -40 dB/dec asymptote extrapolated to $\omega = 1$ rad/s (red dot).

Example 9.5: Asymptotic Bode magnitude sketch for a type-2 system

Consider the open-loop transfer function

$$G(s) = \frac{5(s+5)(s+10)}{s^2(s+20)}.$$

Sketch the asymptotic Bode magnitude plot. Determine the acceleration error constant K_a and the gain crossover frequency ω_{gc}.

Solution:

Step 1 — Convert to Bode form. Factor each term so that the frequency-dependent parts have the form $(1 + s/a)$:

$$G(s) = \frac{5 \cdot 5\left(1+\frac{s}{5}\right) \cdot 10\left(1+\frac{s}{10}\right)}{s^2 \cdot 20\left(1+\frac{s}{20}\right)} = \frac{12.5\left(1+\frac{s}{5}\right)\left(1+\frac{s}{10}\right)}{s^2\left(1+\frac{s}{20}\right)}.$$

The Bode gain is $K_B = 12.5$, i.e. $20\log_{10}(12.5) = 21.9$ dB.

Step 2 — Identify the components and break frequencies.

Factor	Type	Break frequency
$K_B = 12.5$	constant gain	—
$1/s^2$	double integrator	—
$1 + s/5$	zero	5 rad/s
$1 + s/10$	zero	10 rad/s
$1/(1 + s/20)$	pole	20 rad/s

Step 3 — Determine the slope in each frequency range. The double integrator gives a starting slope of -40 dB/dec. At each break frequency the slope changes by +20 (zero) or -20 (pole):

Range	Slope change	Net slope	Reason
$\omega < 5$	$---$	-40 dB/dec	double integrator only
$5 < \omega < 10$	+20	-20 dB/dec	zero at 5
$10 < \omega < 20$	+20	0 dB/dec	zero at 10
$\omega > 20$	-20	-20 dB/dec	pole at 20

Step 4 — Fix the vertical position (anchor point). At $\omega = 1$ rad/s (below all break frequencies), the magnitude is determined by K_B/ω^2:

$$20\log_{10}\left(\frac{12.5}{1^2}\right) = 21.9 \text{ dB}.$$

From this anchor, draw the -40 dB/dec line until $\omega = 5$:

$$\text{At } \omega = 5: \quad 21.9 - 40\log_{10}(5) = 21.9 - 28.0 = -6.0 \text{ dB}.$$

Continue with -20 dB/dec from 5 to 10:

$$\text{At } \omega = 10: \quad -6.0 - 20\log_{10}\left(\frac{10}{5}\right) = -6.0 - 6.0 = -12.0 \text{ dB}.$$

The slope is 0 dB/dec from 10 to 20, so the magnitude remains at -12.0 dB. After the pole at 20, the slope returns to -20 dB/dec.

The resulting asymptotic sketch is compared with the exact Bode magnitude plot in Figure 9.23.

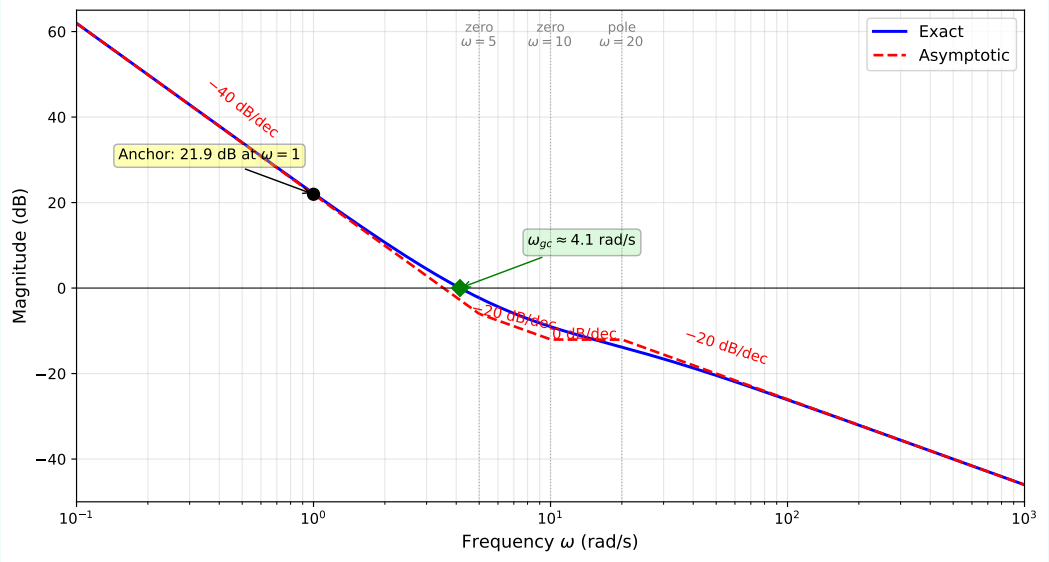


Figure 9.23: Asymptotic (dashed) and exact (solid) Bode magnitude plots for $G(s) = 12.5(1 + s/5)(1 + s/10)/[s^2(1 + s/20)]$. The anchor point at $\omega = 1$ is marked, along with the gain crossover frequency.

Step 5 — Acceleration error constant. This is a type-2 system (two integrators), so the relevant error constant is

$$K_a = \lim_{s\to 0} s^2\, G(s) = \frac{5 \times 5 \times 10}{20} = 12.5.$$

On the Bode plot, K_a equals the value of the -40 dB/dec asymptote at $\omega = 1$ rad/s, which is indeed 21.9 dB = 12.5 in linear scale. The steady-state error to a unit parabolic input is

$$e_{ss} = \frac{1}{K_a} = \frac{1}{12.5} = 0.08.$$

Step 6 — Gain crossover frequency. The gain crossover is where $|G(j\omega)| = 0$ dB. From the asymptotic plot, this occurs on the -40 dB/dec segment:

$$21.9 - 40\log_{10}(\omega_{gc}) = 0 \quad \Longrightarrow \quad \log_{10}(\omega_{gc}) = \frac{21.9}{40} = 0.549 \quad \Longrightarrow \quad \omega_{gc} \approx 3.5 \text{ rad/s}.$$

The exact value (from the plot or MATLAB) is $\omega_{gc} \approx 4.2$ rad/s — the asymptotic estimate is conservative because it does not account for the nearby zero at $\omega = 5$ which lifts the magnitude slightly above the asymptote.

9.9.5 Translation recipe: time-domain specs to frequency-domain targets

Given time-domain specifications, the designer translates them into frequency-domain targets using the following procedure:

1. **Overshoot → phase margin.** Use $M_p = e^{-\pi\zeta/\sqrt{1-\zeta^2}}$ to find the required ζ, then set the phase margin target as $PM \approx 100\zeta$ degrees.
2. **Settling time → crossover frequency.** Use $t_s \approx 4/(\zeta\omega_n)$ together with $\omega_{gc} \approx \omega_n$ (for moderate damping) to estimate the required crossover frequency: $\omega_{gc} \gtrsim 4/(\zeta\, t_s)$.
3. **Steady-state error → low-frequency gain.** Identify the system type and the reference signal, then compute the required error constant: K_p for step tracking (type 0), K_v for ramp tracking (type 1), or K_a for parabolic tracking (type 2). For example, a ramp error of $e_{ss} \leq 0.05$ requires $K_v \geq 1/0.05 = 20$.
4. **Gain margin.** Unless stated otherwise, target $GM \geq 6$ dB (a factor of 2 in gain).

Example 9.6: Translating time-domain specifications to frequency-domain targets

Problem: A motor speed control system must have:

- overshoot $\leq 20\%$ to a step command,
- settling time ≤ 2 s (2% criterion),
- steady-state error $\leq 5\%$ to a ramp input.

Translate these into frequency-domain design targets.

Step 1 (Overshoot → ζ → PM). From $M_p = 0.20$:

$$0.20 = e^{-\pi\zeta/\sqrt{1-\zeta^2}} \quad\Longrightarrow\quad \ln 0.20 = \frac{-\pi\zeta}{\sqrt{1-\zeta^2}} \quad\Longrightarrow\quad \zeta \approx 0.456.$$

Phase margin target: $PM \geq 100 \times 0.456 \approx 46^\circ$. Rounding up for safety: $PM \geq 50^\circ$.

Step 2 (Settling time → ω_{gc}).

$$\omega_n \geq \frac{4}{\zeta\, t_s} = \frac{4}{0.456 \times 2} = 4.39 \text{ rad/s}.$$

Since $\omega_{gc} \approx \omega_n$ for moderate damping, we target $\omega_{gc} \geq 4.4$ rad/s.

Step 3 (Ramp error → K_v).

$$K_v \geq \frac{1}{e_{ss}} = \frac{1}{0.05} = 20.$$

Summary of frequency-domain targets:

Parameter	Target
Phase margin	$\geq 50^\circ$
Gain crossover frequency	≥ 4.4 rad/s
Velocity error constant	$K_v \geq 20$
Gain margin	≥ 6 dB

Remark 9.3

The second-order approximations are most accurate when the closed-loop system is dominated by a pair of complex poles. For higher-order plants where additional poles are not far from the dominant pair, the approximations underestimate the overshoot and settling time. Always verify the final design with a MATLAB step response.

9.10 Frequency-Domain Compensator Design

The goal is to choose a controller $C(s)$ so that the loop transfer function $L(s) = C(s)G(s)$ has: large low-frequency gain (tracking and disturbance rejection), an acceptable crossover frequency (speed), adequate phase margin (damping and robustness), and low high-frequency gain (noise rejection). No controller improves everything at once, so frequency-domain design is about shaping the loop gain in a controlled way.

9.10.1 Which compensator? — A design decision chart

The time-domain specifications translate into three frequency-domain questions, as shown in Table 9.4.

Time-domain spec	Frequency-domain target	Bode-plot quantity
Overshoot M_p	Phase margin PM	Phase at ω_{gc}
Rise time t_r, settling time t_s	Crossover frequency ω_{gc}	Magnitude = 0 dB frequency
Steady-state error e_{ss}	Error constant $K_p/K_v/K_a$	Low-frequency gain level

Table 9.4: Translation of time-domain specifications to frequency-domain design targets.

PM and ω_{gc} both concern the *crossover region* and are addressed by a Lead compensator (adds positive phase and raises gain near crossover). Steady-state accuracy concerns the *low-frequency* region and is addressed by a Lag compensator (boosts low-frequency gain without affecting crossover). The compensator selection involves three checks:

1. **Phase margin:** Is PM adequate (from the overshoot spec)? If not, a lead section is needed to add phase near crossover.
2. **Crossover frequency:** Is ω_{gc} high enough (from the settling/rise time spec)? If not, a lead section is again the remedy — it raises the magnitude near crossover, pushing ω_{gc} higher.
3. **Steady-state accuracy:** Is the error constant (K_p, K_v, or K_a) large enough? If not, a lag section is needed to boost low-frequency gain.

Checks 1 and 2 both lead to a lead compensator, so both “No” paths merge in the flowchart (Figure 9.24).

Table 9.5 shows what each compensator does to the Bode plot.

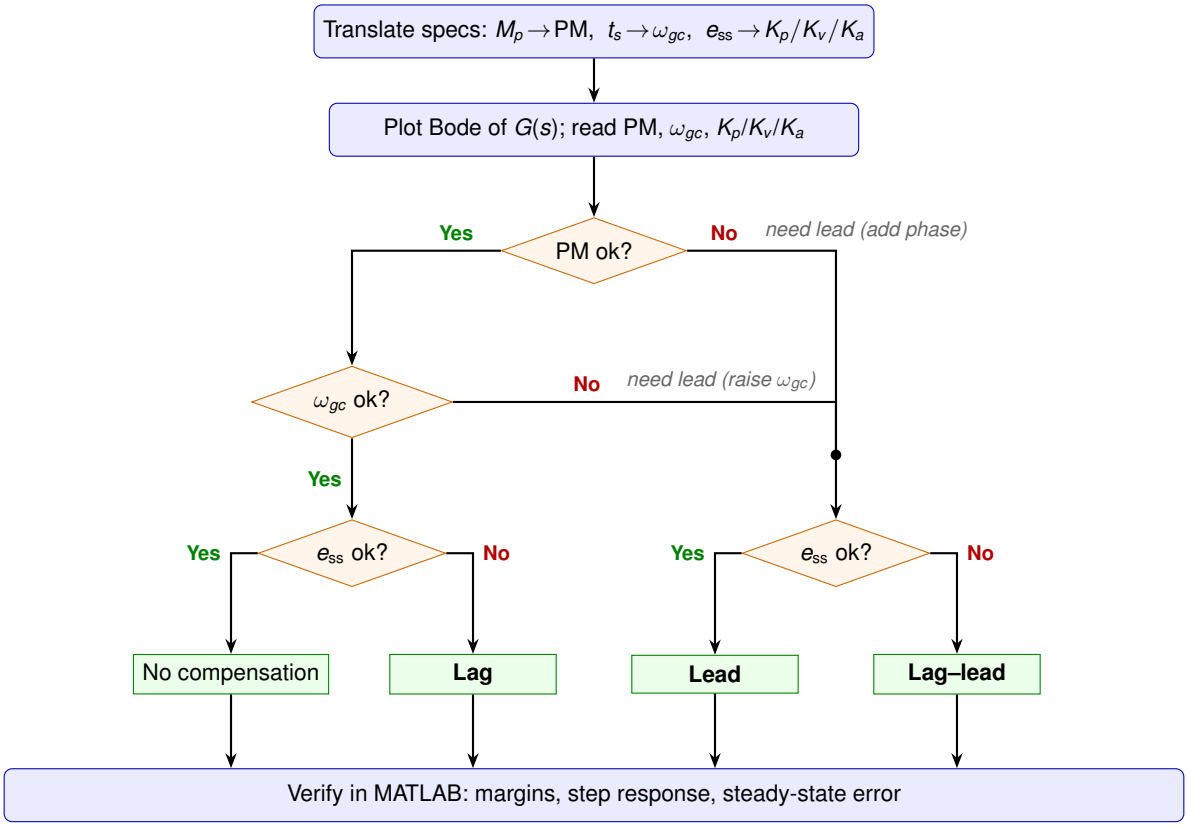


Figure 9.24: Compensator selection flowchart. PM and ω_{gc} are checked separately, but both "No" paths merge because a lead compensator addresses both (it adds phase *and* tends to raise the crossover frequency). The final decision checks whether the low-frequency gain also needs boosting.

Compensator	**Raises ω_{gc}?**	**Adds PM?**	**Boosts low-freq gain?**
Lead	Yes (moderate)	Yes	No
Lag	No (slight decrease)	No (slight decrease)	Yes
Lag–lead	Yes	Yes	Yes

Table 9.5: Compensator capabilities: effect on the three key Bode-plot properties.

9.10.2 General design algorithm

The frequency-domain design process follows a common structure:

1. **Translate specifications.** Convert each time-domain requirement into a frequency-domain target:
   - Overshoot $\rightarrow$ phase margin (via $PM \approx 100\zeta$).
   - Settling time or rise time $\rightarrow$ crossover frequency (via $\omega_{gc} \gtrsim 4/(\zeta\, t_s)$).
   - Steady-state error $\rightarrow$ error constant K_p, K_v, or K_a (depending on system type and reference).
   - Gain margin $\rightarrow$ default $GM \geq 6$ dB unless specified otherwise.
2. **Plot and diagnose.** Plot the uncompensated Bode diagram. Read ω_{gc}, PM, ω_{pc}, GM, and the relevant error constant. Compare each with the targets from Step 1.
3. **Select compensator type.** Use the decision chart (Figure 9.24): if the crossover region (PM and/or ω_{gc}) is deficient, a lead section is needed; if the low-frequency gain is deficient, a lag section is needed; if both, use lag–lead.
4. **Compute compensator parameters.** Follow the step-by-step algorithm for the chosen compensator type (detailed in the subsections that follow).
5. **Verify.** Plot the compensated Bode diagram and the closed-loop step response in MATLAB. Check *all* specifications: PM, ω_{gc}, e_{ss}, GM, overshoot, and settling time. If any target is missed, iterate — typically by adjusting the safety margin or the placement of the compensator zero and pole.

9.10.3 Lead compensation

A **lead compensator** adds positive phase near the gain crossover frequency, improving damping and speed of response.

When is a lead compensator needed?

After plotting the Bode diagram of the uncompensated loop $L_0(s) = K\,G(s)$, check the phase margin at the gain crossover frequency. A lead compensator is the right tool when:

- The **phase margin is too small** (the closed loop is too oscillatory or close to instability), *or*
- The **crossover frequency is too low** (the closed-loop response is too sluggish).

A lead section addresses *both* problems simultaneously: it adds positive phase at crossover (raising PM) and adds gain in that band (pushing ω_{gc} higher).

A lead compensator is **not** the right choice when the main deficiency is steady-state accuracy (low-frequency gain too small). That situation calls for a lag compensator (Section 9.10.4).

The standard lead compensator

The transfer function is

$$C_{\text{lead}}(s) = K_c\,\frac{Ts+1}{\alpha Ts+1}, \qquad 0 < \alpha < 1.$$

It has:

- a **zero** at $\omega_z = 1/T$ (lower frequency),
- a **pole** at $\omega_p = 1/(\alpha T)$ (higher frequency, since $\alpha < 1$).

Between these two frequencies the compensator contributes positive phase. The **maximum phase lead** and the frequency at which it occurs are

$$\phi_{\max} = \sin^{-1}\left(\frac{1-\alpha}{1+\alpha}\right), \qquad \omega_m = \frac{1}{T\sqrt{\alpha}}.$$

At ω_m the compensator magnitude is $10\log_{10}(1/\alpha)$ dB, exactly half the high-frequency asymptotic gain of $20\log_{10}(1/\alpha)$ dB. Figure 9.25 summarises all these relationships on a parametric Bode plot.

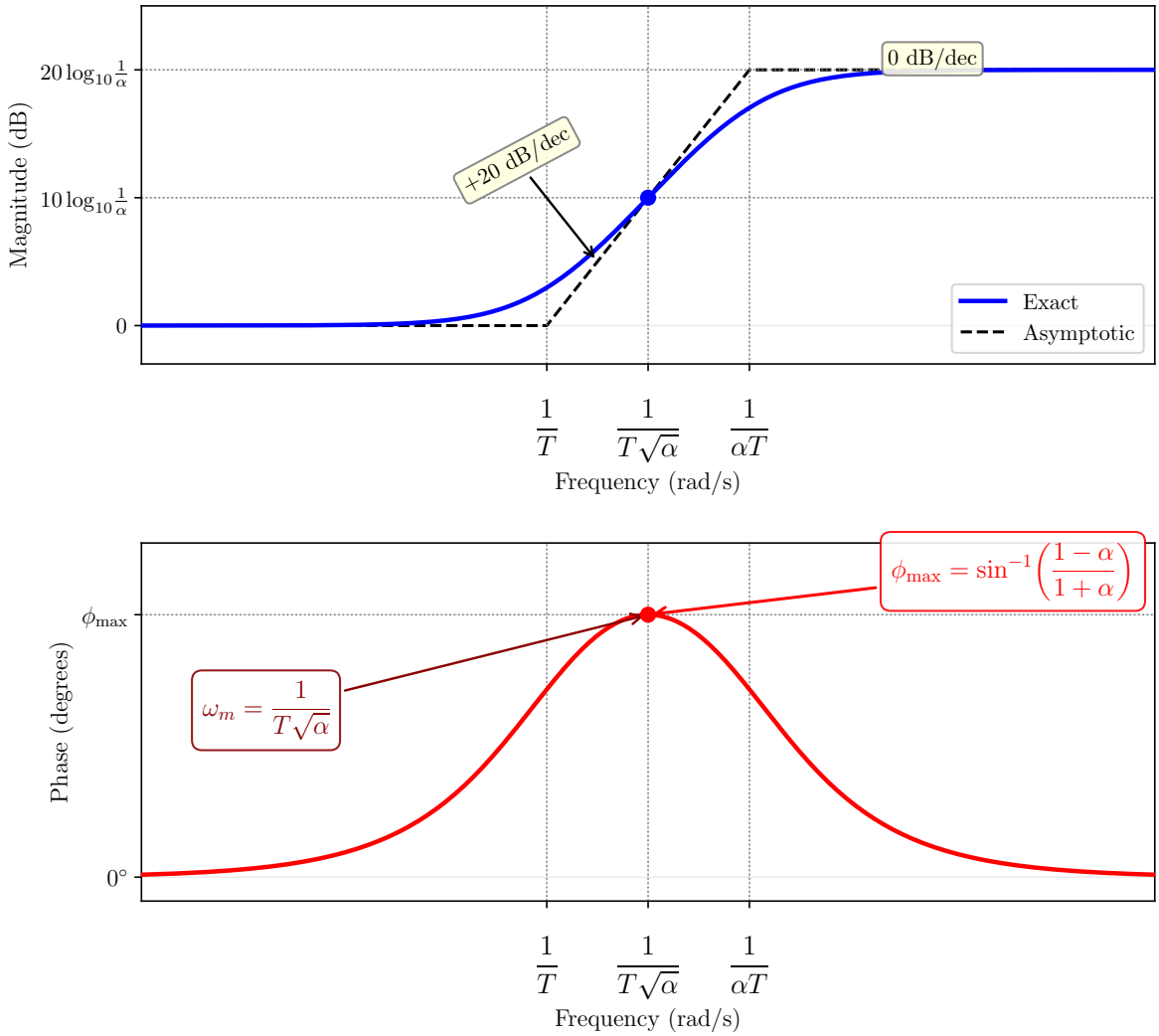


Figure 9.25: Parametric Bode plot of a lead compensator $C(s) = K_c(Ts+1)/(\alpha Ts+1)$ with $0 < \alpha < 1$. Key design quantities — the corner frequencies $1/T$ and $1/(\alpha T)$, the peak frequency $\omega_m = 1/(T\sqrt{\alpha})$, and the maximum phase lead $\phi_{\max}$ — are shown in terms of the compensator parameters.

Design procedure

Lead Compensator Design Procedure

1. **Analyse the uncompensated loop.** Plot the Bode diagram of the plant $G(s)$ (or $K\,G(s)$ if a gain has already been chosen). Read off the current phase margin PM_0 and gain crossover frequency $\omega_{gc,0}$.

2. **Determine the required phase lead.** Compute the phase deficit:

$$\phi_{\text{need}} = \mathrm{PM}_{\text{des}} - \mathrm{PM}_0.$$

   Because the lead section raises the gain near crossover, the crossover shifts to a higher frequency where the plant phase is more negative. To compensate, add a safety margin of 5°–15°:

$$\phi_{\max} = \phi_{\text{need}} + \underbrace{(5^\circ\text{–}15^\circ)}_{\text{safety margin}}.$$

   Rule of thumb: use 5° if the plant phase curve is fairly flat near crossover, and up to 15° if it drops steeply.

3. **Compute α.** Invert the $\phi_{\max}$ formula:

$$\alpha = \frac{1 - \sin\phi_{\max}}{1 + \sin\phi_{\max}}.$$

   Smaller α gives more phase lead but spreads the zero and pole further apart, which amplifies high-frequency noise. In practice $\alpha \geq 0.05$ (i.e. $\phi_{\max} \leq 65^\circ$); if more lead is needed, cascade two lead sections.

4. **Find the new crossover frequency ω_m.** At ω_m the compensator adds $10\log_{10}(1/\alpha)$ dB of gain. For the *compensated* crossover to fall at ω_m, the *uncompensated* magnitude must equal minus this value:

$$|G(j\omega_m)|_{\text{dB}} = -10\log_{10}\frac{1}{\alpha}.$$

   Read ω_m from the Bode magnitude plot (or solve numerically).

5. **Compute T.**

$$T = \frac{1}{\omega_m\sqrt{\alpha}}.$$

   This places the zero at $\omega_z = 1/T = \omega_m\sqrt{\alpha}$ and the pole at $\omega_p = 1/(\alpha T) = \omega_m/\sqrt{\alpha}$, symmetrically around ω_m on a log scale.

6. **Set the gain K_c.** If a specific low-frequency gain or error constant is required, choose K_c to meet it. Otherwise set $K_c = 1$ (the lead section alone does not change the DC gain when $K_c = 1$, since $C_{\text{lead}}(0) = 1$).

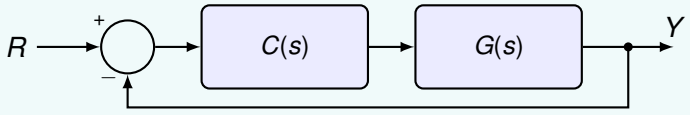
7. **Verify.** Plot the compensated Bode diagram $C_{\text{lead}}(j\omega)\,G(j\omega)$ and confirm that PM and ω_{gc} meet the specifications. Simulate the closed-loop step response to check overshoot and settling time. If the margin is insufficient, increase the safety margin in Step 2 and repeat.

Design tricks and practical tips

- **Reading the Bode plot is the key skill.** Steps 1 and 4 both require reading values from the magnitude and phase curves. Practise locating the frequency where the magnitude equals a given dB value.
- **Why the peak goes at ω_m.** The compensator contributes its maximum phase exactly at the geometric mean of its zero and pole. By forcing the compensated crossover to coincide with ω_m, we extract every degree of phase lead that the compensator can provide.
- **Zero below, pole above.** The zero $1/T$ always sits below the new crossover ω_m, and the pole $1/(\alpha T)$ always sits above it. This is what creates the positive phase hump at crossover.
- **Noise amplification trade-off.** A smaller α gives more phase lead but also more high-frequency gain ($20\log_{10}(1/\alpha)$ dB). This amplifies sensor noise. Keep $\alpha \geq 0.05$; if more lead is needed, use two cascaded lead sections rather than one aggressive one.
- **The safety margin is not a guess.** It accounts for the fact that the plant phase at the *new* crossover is more negative than at the *old* one. A steep phase roll-off (e.g. multiple poles near crossover) demands a larger safety margin.

Example 9.7: Lead design to improve phase margin — complete worked example

Problem: Consider the unity-feedback system shown below, with plant $G(s) = \frac{10}{s(s+1)}$ and compensator $C(s)$ to be designed. The specification is PM $\geq 50^\circ$.



We follow the design procedure step by step.

Solution:

Step 1 — Analyse the uncompensated loop. With $C = 1$, the Bode plot of $G(j\omega)$ gives $\text{PM}_0 \approx 18^\circ$ at $\omega_{gc,0} \approx 3.1$ rad/s. The phase margin is far below the target of 50°, so a lead compensator is needed to add phase near crossover.

Step 2 — Determine the required phase lead. The phase deficit is

$$\phi_{\text{need}} = 50^\circ - 18^\circ = 32^\circ.$$

The plant phase rolls off at moderate rate near crossover (one integrator plus one real pole), so we add a 10° safety margin:

$$\phi_{\max} = 32^\circ + 10^\circ = 42^\circ.$$

Step 3 — Compute α.

$$\alpha = \frac{1 - \sin 42^\circ}{1 + \sin 42^\circ} = \frac{1 - 0.669}{1 + 0.669} = \frac{0.331}{1.669} \approx 0.198.$$

Since $\alpha \geq 0.05$, a single lead section is sufficient.

Step 4 — Find the new crossover frequency ω_m. The compensator adds $10\log_{10}(1/0.198) \approx 7.0$ dB at ω_m. For the compensated crossover to land at ω_m, the uncompensated magnitude must equal -7.0 dB there. From the Bode plot (or solving $|G(j\omega_m)|_{\text{dB}} = -7.0$ numerically):

$$\omega_m \approx 4.7 \text{ rad/s}.$$

Step 5 — Compute T and locate the zero and pole.

$$T = \frac{1}{\omega_m\sqrt{\alpha}} = \frac{1}{4.7 \times \sqrt{0.198}} = \frac{1}{4.7 \times 0.445} \approx 0.478 \text{ s}.$$

The compensator zero and pole are:

$$\omega_z = \frac{1}{T} = 2.09 \text{ rad/s} \quad \text{(below } \omega_m\text{)}, \qquad \omega_p = \frac{1}{\alpha T} = 10.6 \text{ rad/s} \quad \text{(above } \omega_m\text{)}.$$

Step 6 — Set the gain and write the compensator. No low-frequency gain adjustment is required ($K_c = 1$):

$$C_{\text{lead}}(s) = \frac{0.478\,s + 1}{0.0946\,s + 1}.$$

Step 7 — Verify. Figure 9.26 confirms PM $\approx 54°$ at $\omega_{gc} \approx 4.7$ rad/s — the $10°$ safety margin was well chosen. Figure 9.27 shows the closed-loop step response: about 18% overshoot and 1.3 s settling time, a large improvement over the uncompensated 60%+ overshoot.

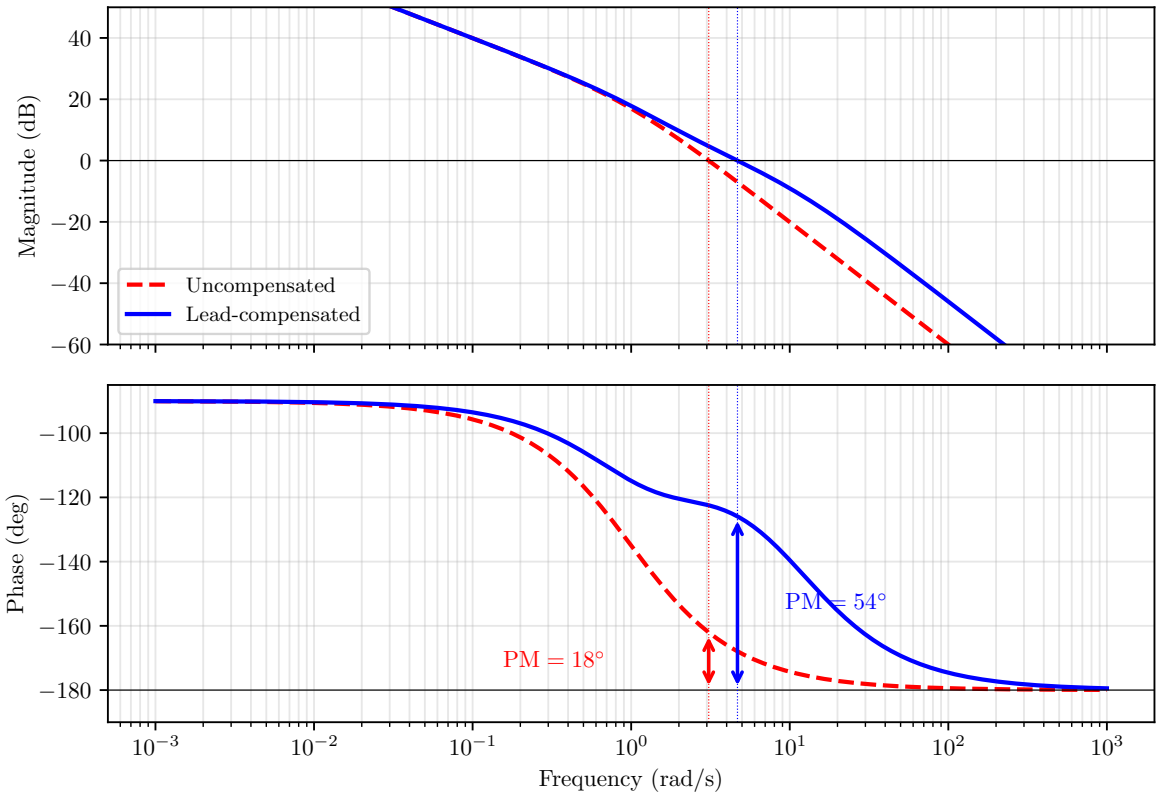


Figure 9.26: Bode plots for the lead design example. The compensator shifts the crossover from 3.1 to 4.7 rad/s while lifting the phase from $-162°$ to $-126°$ at the new crossover.

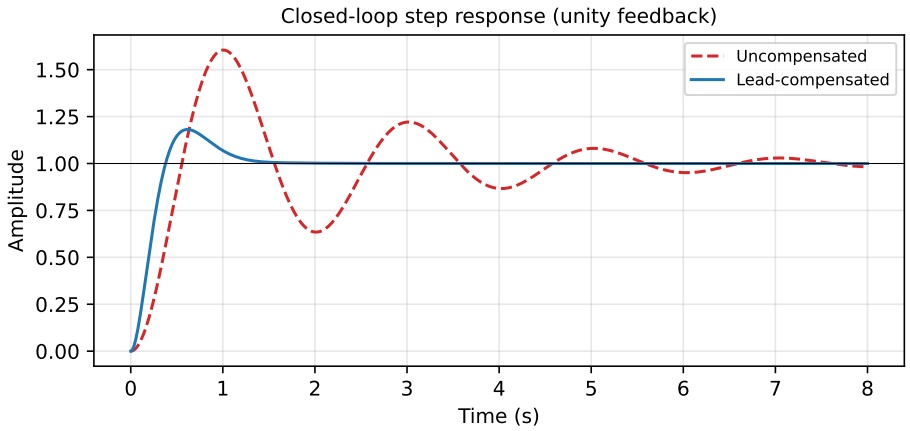


Figure 9.27: Closed-loop step responses for the lead design example (unity feedback).

The complete MATLAB implementation is given in Listing 9.8.

```
s = tf('s');
G = 10/(s*(s+1));

% Lead compensator
T_lead = 0.478;
alpha = 0.198;
Clead = (T_lead*s + 1)/(alpha*T_lead*s + 1);

% Bode comparison
figure
margin(G), grid on, title('Uncompensated')
figure
margin(Clead*G), grid on, title('Lead-compensated')

% Closed-loop step response (unity feedback: H=1)
Tcl_uncomp = feedback(G, 1);
Tcl_lead   = feedback(Clead*G, 1);
figure
step(Tcl_uncomp, Tcl_lead), grid on
legend('Uncompensated','Lead-compensated')
```

Listing 9.8: MATLAB code for the lead compensator design and verification.

Example 9.8: Lead compensator design for a double-integrator plant

Problem. Consider the unity-feedback system in Figure 9.28. The plant is a double integrator $G(s) = K/s^2$, where K is a gain to be chosen. Design a lead compensator

$$C(s) = \frac{Ts + 1}{\alpha Ts + 1}, \qquad 0 < \alpha < 1,$$

and choose K so that the closed-loop system meets:

- settling time $t_s \leq 4$ s (2% criterion),
- damping ratio $\zeta \geq 0.45$ (equivalently, overshoot $\leq 25\%$).

Assume the closed-loop behaviour is dominated by a second-order pair.

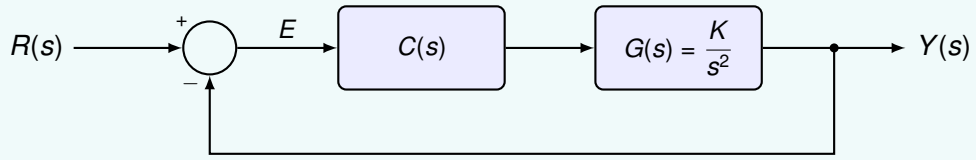


Figure 9.28: Unity-feedback system with a lead compensator and double-integrator plant.

Solution:

Step 1 — Translate time-domain specs to frequency domain (preliminary step).

For a dominant second-order system with $\zeta \geq 0.45$, the approximate relation $\text{PM} \approx 100\zeta$ gives

$$\text{PM}_{\text{des}} \geq 100 \times 0.45 = 45^\circ.$$

The settling-time requirement $t_s \leq 4$ s imposes a minimum bandwidth. Using $t_s \approx 4/(\zeta\omega_n)$ with $\zeta = 0.45$:

$$\omega_n \geq \frac{4}{0.45 \times 4} \approx 2.22 \text{ rad/s}.$$

As a rule of thumb, the gain crossover frequency should satisfy $\omega_{gc} \gtrsim \omega_n$, so we target $\omega_{gc} \geq 2.2$ rad/s.

Step 2 — Analyse the uncompensated loop (Procedure Step 1).

The uncompensated plant K/s^2 is a double integrator: magnitude drops at -40 dB/dec and the phase is a constant -180° at all frequencies. Therefore $\text{PM}_0 = 0^\circ$ — the system is marginally unstable without compensation. A lead compensator is essential to inject positive phase at crossover.

Step 3 — Determine the required phase lead (Procedure Step 2).

The entire 45° of phase margin must come from the compensator. The double integrator has a flat -180° phase, so the crossover shift has little effect on the phase, but we still add a 10° safety margin:

$$\phi_{\max} = 45^\circ + 10^\circ = 55^\circ.$$

Step 4 — Compute α (Procedure Step 3).

$$\alpha = \frac{1 - \sin 55^\circ}{1 + \sin 55^\circ} = \frac{1 - 0.8192}{1 + 0.8192} = \frac{0.1808}{1.8192} \approx 0.1.$$

Since $\alpha = 0.1 \geq 0.05$, a single lead section is sufficient.

Step 5 — Find the new crossover frequency ω_m (Procedure Step 4).

At ω_m, the compensator adds $10\log_{10}(1/\alpha) = 10\log_{10} 10 = 10$ dB. For the compensated crossover to fall at ω_m, the uncompensated magnitude must equal -10 dB there:

$$20\log_{10}\left(\frac{K}{\omega_m^2}\right) = -10 \quad \Longrightarrow \quad \frac{K}{\omega_m^2} = 10^{-1/2} = 0.3162.$$

Setting $K = 2$:

$$\omega_m = \sqrt{\frac{2}{0.3162}} = \sqrt{6.325} \approx 2.51 \text{ rad/s},$$

which exceeds the minimum 2.2 rad/s target from Step 1.

Step 6 — Compute T and locate the zero and pole (Procedure Step 5).

$$T = \frac{1}{\omega_m\sqrt{\alpha}} = \frac{1}{2.51 \times \sqrt{0.1}} = \frac{1}{2.51 \times 0.3162} \approx 1.26 \text{ s}.$$

The compensator zero and pole are:

$$\omega_z = \frac{1}{T} = 0.80 \text{ rad/s} \quad (\text{below } \omega_m), \qquad \omega_p = \frac{1}{\alpha T} = 7.95 \text{ rad/s} \quad (\text{above } \omega_m).$$

The lead compensator is therefore

$$\boxed{C_{\text{lead}}(s) = \frac{1.26\,s + 1}{0.126\,s + 1}}.$$

Step 7 — Verify (Procedure Step 7).

Figure 9.29 shows that the compensated loop achieves PM = 55° at $\omega_{gc} \approx 2.5$ rad/s. Figure 9.30 confirms a step-response overshoot of 23% ($< 25\%$) and settling time $t_s \approx 3.3$ s (< 4 s) — both specifications are satisfied.

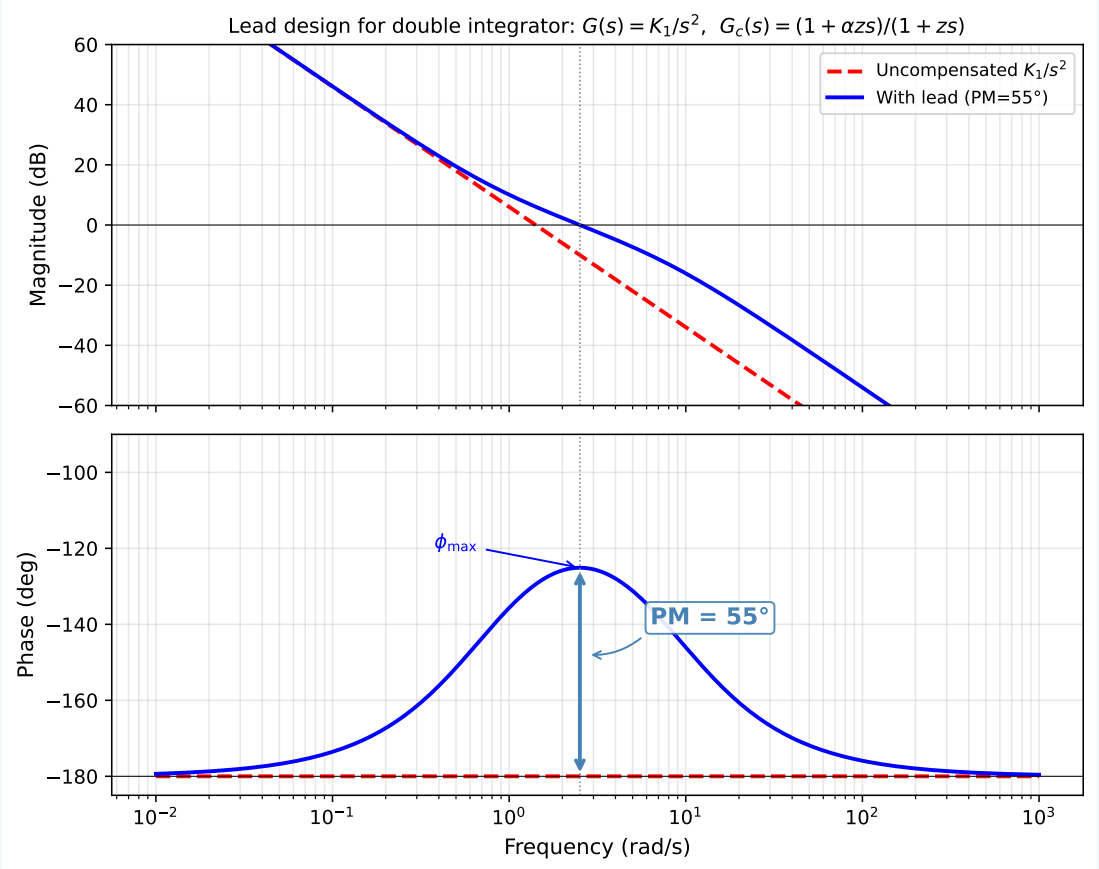


Figure 9.29: Bode plots for the double-integrator lead design. The compensator lifts the phase at the new crossover to $-125°$, giving PM = 55°.

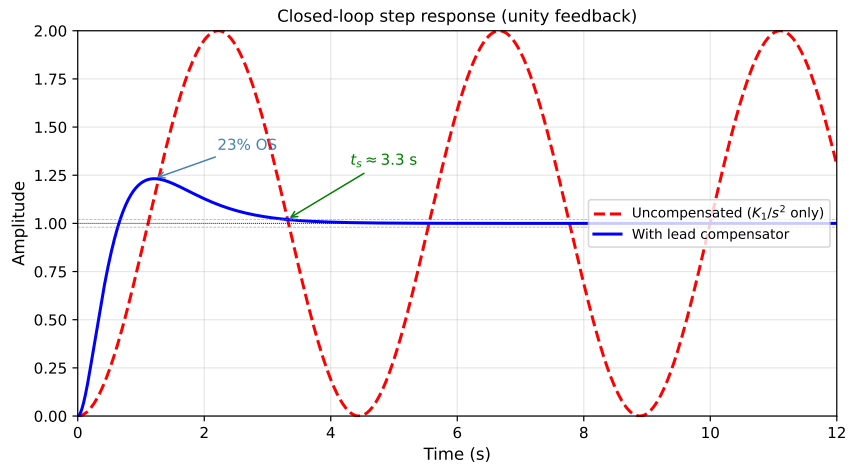


Figure 9.30: Closed-loop step response of the double-integrator lead design: 23% overshoot and $t_s \approx 3.3$ s.

The MATLAB code in Listing 9.9 reproduces these results.

```
s  = tf('s');
K = 2;
G = K/s^2;                              % double-integrator plant

alpha  = 0.1;                            % from phi_max = 55 deg
T_lead = 1.26;                           % 1/(wm*sqrt(alpha))
Clead = (T_lead*s + 1)/(alpha*T_lead*s + 1);  % lead compensator

L = Clead*G;                             % open-loop
figure, margin(L), grid on               % Bode plot with PM annotation

Tcl = feedback(L, 1);                    % closed-loop (unity feedback)
figure, step(Tcl, 8), grid on            % step response
```

Listing 9.9: MATLAB code for the double-integrator lead compensator design.

9.10.4 Lag compensation

A **lag compensator** increases the low-frequency loop gain without greatly changing the crossover frequency, thereby improving steady-state accuracy.

When is a lag compensator needed?

After plotting the Bode diagram of the uncompensated loop, check the steady-state error performance. A lag compensator is the right tool when:

- The **phase margin is already adequate** (the transient response is acceptable), *but*
- The **low-frequency gain is too small** (the steady-state error for step, ramp, or parabolic inputs is too large — i.e. K_p, K_v, or K_a is insufficient).

The lag section boosts the low-frequency gain by a factor β while leaving the crossover region almost untouched, so the transient response is approximately preserved.

A lag compensator is **not** the right choice when the phase margin itself is deficient. That situation calls for a lead compensator (Section 9.10.3). If *both* PM and low-frequency gain are inadequate, a lag–lead compensator is needed (Section 9.10.5).

The standard lag compensator

The transfer function is

$$C_{\text{lag}}(s) = K_c \frac{Ts + 1}{\beta Ts + 1}, \qquad \beta > 1.$$

It has:

- a **zero** at $\omega_z = 1/T$ (higher frequency),
- a **pole** at $\omega_p = 1/(\beta T)$ (lower frequency, closer to the origin, since $\beta > 1$).

Below both frequencies, the magnitude is $20 \log_{10} \beta$ dB higher than above both — this is the low-frequency gain boost. Between the pole and zero, the magnitude transitions at -20 dB/dec. Both the zero and pole are placed *well below* the crossover frequency, so the lag section boosts low-frequency gain by β while leaving the crossover region essentially unchanged.

The lag section also introduces **negative phase** between its pole and zero, with a maximum lag of

$$\phi_{\text{min}} = -\sin^{-1}\left(\frac{\beta - 1}{\beta + 1}\right),$$

occurring at the geometric mean frequency $\omega_m = 1/(T\sqrt{\beta})$. This negative phase must be kept well away from the crossover region — otherwise it will reduce the phase margin. Figure 9.31 summarises all these relationships on a parametric Bode plot.

Design procedure

Lag Compensator Design Procedure

1. **Analyse the uncompensated loop.** Plot the Bode diagram of $G(s)$ (or $K\,G(s)$). Read off the current phase margin PM_0, gain crossover frequency $\omega_{gc,0}$, and compute the relevant error constant (K_p, K_v, or K_a) from the low-frequency magnitude. Confirm that PM is already acceptable — if not, a lag compensator alone is not enough.
2. **Determine β.** The gain boost required is

   $$\beta = \frac{K_{\text{des}}}{K_{\text{uncomp}}},$$

   where K is the relevant error constant (K_p, K_v, or K_a). This is the single most important parameter — it determines how much the low-frequency gain is raised.
3. **Place the lag zero.** Choose the zero frequency $\omega_z = 1/T$ at least one decade below the current crossover:

   $$\omega_z \leq \frac{\omega_{gc,0}}{10}.$$

   This ensures that the negative phase contribution of the lag section has returned close to 0° by the time the crossover region is reached.

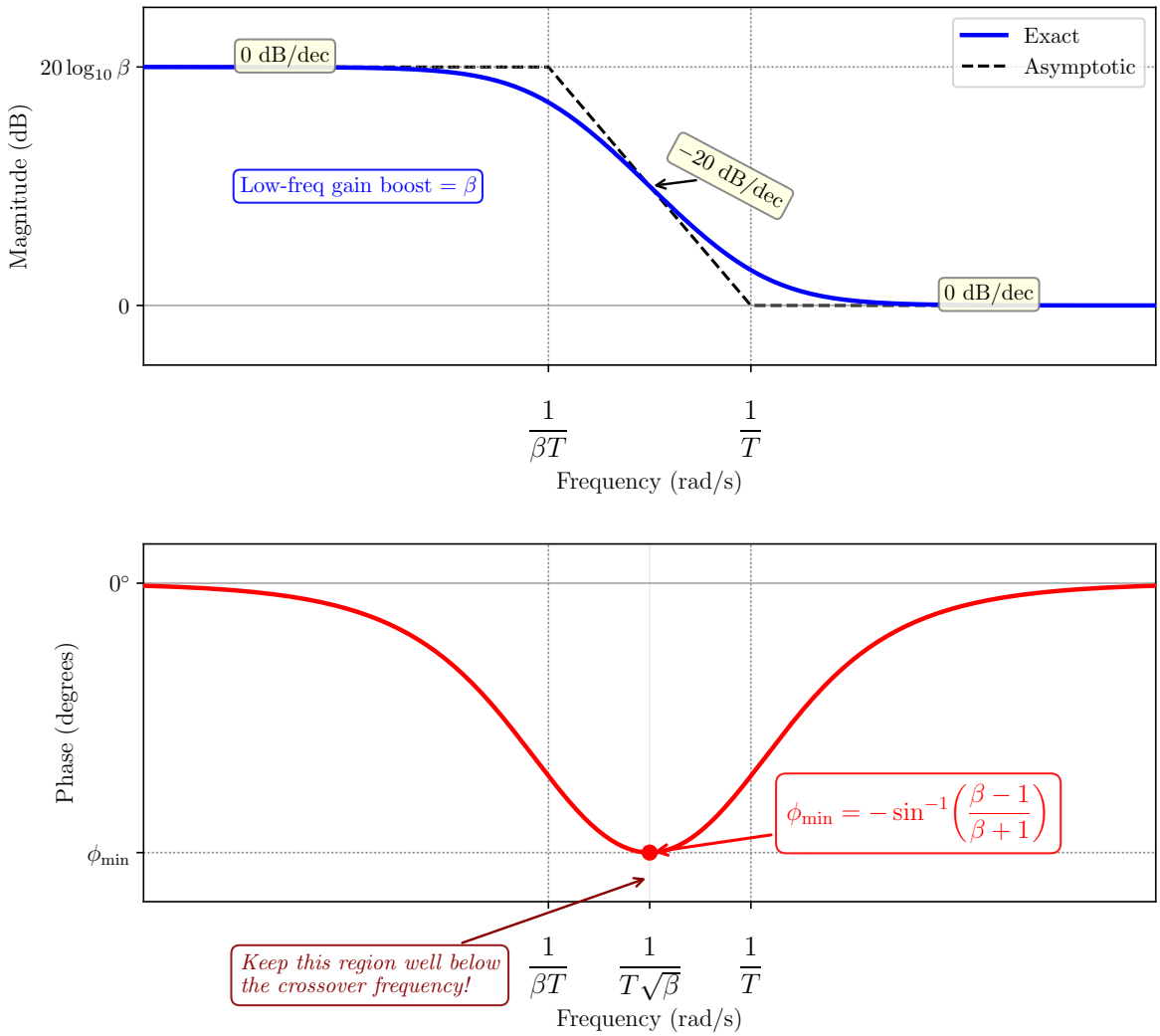


Figure 9.31: Parametric Bode plot of a lag compensator $C(s) = K_c(Ts+1)/(\beta Ts+1)$ with $\beta > 1$. The low-frequency gain is boosted by $20\log_{10}\beta$ dB. The negative phase dip between the pole $1/(\beta T)$ and zero $1/T$ must be kept well below the crossover frequency to avoid eroding PM.

Rule of thumb: a factor of 8–10 below ω_{gc} is typical. Placing it even lower reduces the phase margin loss but slows down the closed-loop settling (the lag pole is closer to the origin).

4. **Compute T and locate the pole.**

$$T = \frac{1}{\omega_z}, \qquad \omega_p = \frac{\omega_z}{\beta} = \frac{1}{\beta T}.$$

The pole sits β times lower than the zero on the frequency axis.

5. **Set the gain and write the compensator.** A common choice is $K_c = \beta$, which puts the entire low-frequency boost into the compensator:

$$C_{\text{lag}}(s) = \beta \frac{Ts + 1}{\beta Ts + 1}, \qquad C_{\text{lag}}(0) = \beta.$$

Alternatively, part of the boost can come from a separate loop gain; the key requirement is that the total DC loop gain meets the error-constant specification.

6. **Verify.** Plot the compensated Bode diagram and confirm that:
   - the low-frequency gain has increased by $20 \log_{10} \beta$ dB (error constant met),
   - the crossover frequency is approximately unchanged,
   - the phase margin has dropped by no more than a few degrees.

   Simulate the closed-loop step response to confirm the transient is still acceptable. If the phase margin drops too much, move the zero lower (increase T).

Design tricks and practical tips

- **What the lag section does to the magnitude curve.** It raises the gain at low frequencies (where accuracy matters) without raising it near crossover (where PM matters), by placing a pole–zero pair that tilts the magnitude curve downward between the two break frequencies.
- **Phase margin will decrease slightly.** The negative phase between the pole and zero always extends partially into the crossover region. Typically PM drops by 3°–8°; if this is unacceptable, push the zero further below crossover.
- **Slow closed-loop pole.** The pole at $1/(\beta T)$ is very close to the origin. This adds a slow component to the closed-loop response — the step response may appear settled but continue creeping for tens of seconds before reaching its final value.
- **Don't use lag for PM improvement.** Lag adds negative phase, not positive. It cannot fix a system whose PM is already too low — that requires lead compensation.
- **β is rarely larger than 20.** Very large β pushes the pole too close to the origin, making the closed-loop settling unacceptably slow. If more gain boost is needed, redesign the system (e.g. add an integrator or use lag–lead).

Example 9.9: Lag design for steady-state accuracy — complete worked example

Problem: A unity-feedback system (same structure as the lead examples above) has plant $G(s) = 1/(s(s+1))$. The uncompensated step response is acceptable ($\text{PM}_0 \approx 52°$), but the velocity error constant is only

$$K_v = \lim_{s \to 0} s\, G(s) = 1,$$

giving a steady-state ramp error of $e_{ss} = 1/K_v = 1$. The specification requires $e_{ss} \leq 0.1$, i.e. $K_v \geq 10$.

We follow the design procedure step by step.

Solution:

Step 1 — Analyse the uncompensated loop. The Bode plot of $G(j\omega)$ gives $\mathrm{PM}_0 \approx 52^\circ$ at $\omega_{gc,0} \approx 0.78$ rad/s. The phase margin is adequate, but $K_v = 1$ is far below the required 10. Since PM is already good and only the low-frequency gain needs improvement, a lag compensator is the right choice.

Step 2 — Determine β.

$$\beta = \frac{K_v^{\text{des}}}{K_v^{\text{uncomp}}} = \frac{10}{1} = 10.$$

Step 3 — Place the lag zero. The guideline says $\omega_z \leq \omega_{gc}/10 = 0.078$ rad/s. We relax this slightly to $\omega_z = 0.1$ rad/s ($\approx \omega_{gc}/8$), because pushing the zero to 0.078 would place the pole at 0.0078 rad/s — unacceptably close to the origin, causing extremely slow settling. With this choice:

$$\omega_z = 0.1 \text{ rad/s}, \qquad T = 1/\omega_z = 10 \text{ s}.$$

Step 4 — Locate the pole.

$$\omega_p = \frac{\omega_z}{\beta} = \frac{0.1}{10} = 0.01 \text{ rad/s}, \qquad \beta T = 100 \text{ s}.$$

Step 5 — Write the compensator. With $K_c = \beta = 10$:

$$C_{\text{lag}}(s) = 10 \cdot \frac{10s + 1}{100s + 1}.$$

The new velocity error constant is $K_v = \beta \times K_v^{\text{uncomp}} = 10 \times 1 = 10$, meeting the specification.

Step 6 — Verify. Figure 9.32 shows that the low-frequency magnitude is raised by 20 dB (the factor of 10) while the crossover region is barely affected. The phase margin drops from 52° to 46° — a small price for a tenfold improvement in K_v. Figure 9.33 (left) confirms the step response remains well-damped, and Figure 9.33 (right) shows the ramp response: the steady-state ramp error drops from 1 to 0.1, exactly as predicted by the new $K_v = 10$.

The MATLAB implementation is given in Listing 9.10.

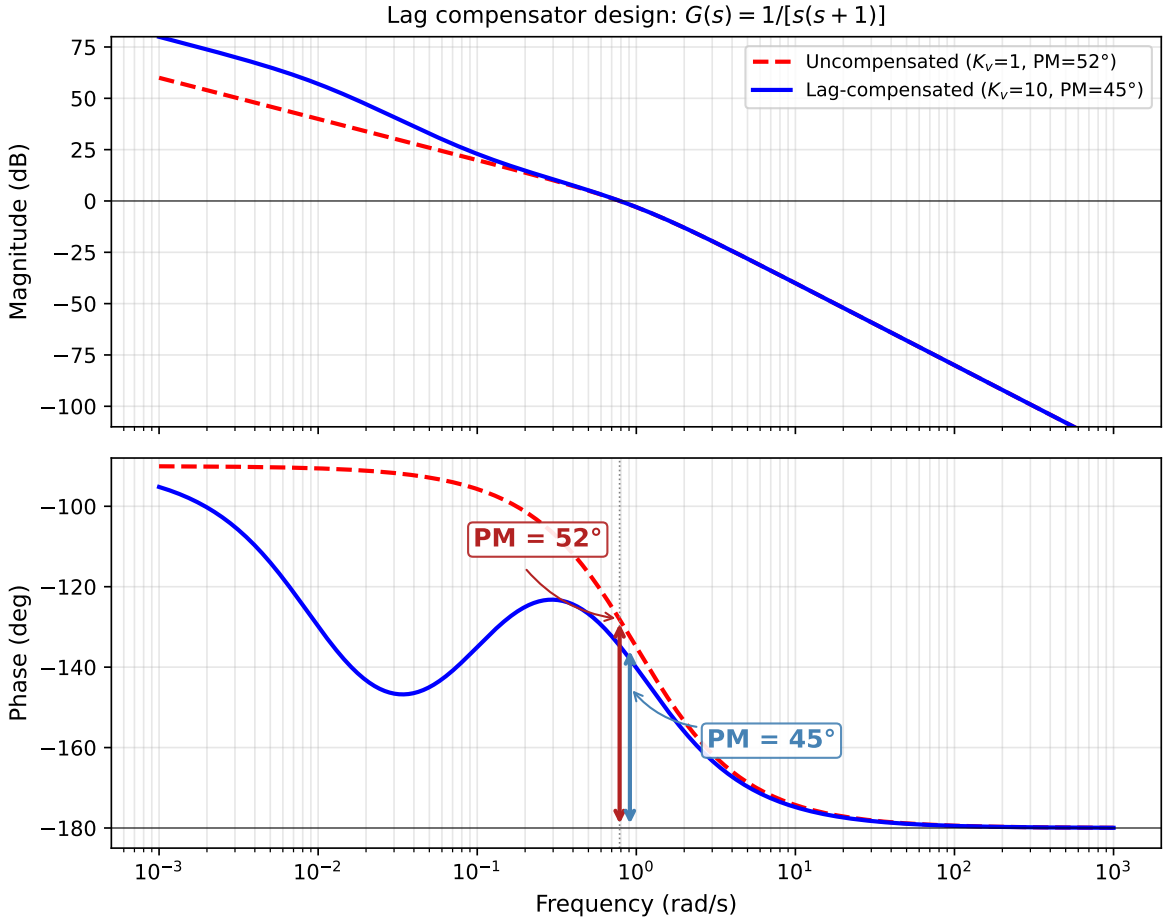


Figure 9.32: Bode plots for the lag design example. The 20 dB gain boost below 0.1 rad/s is clearly visible, while the crossover region (near 0.8 rad/s) is almost unchanged.

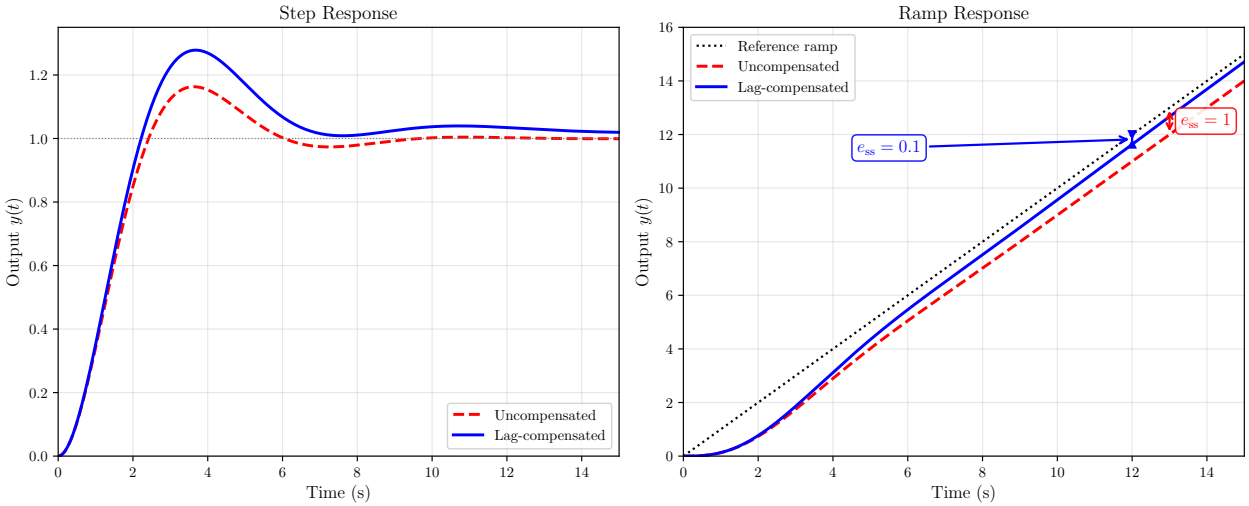


Figure 9.33: Closed-loop responses for the lag design example (unity feedback). Left: step response — the lag-compensated system is slightly more oscillatory due to the 6° PM reduction but remains well-damped. Right: ramp response — the steady-state ramp error drops from $e_{ss} = 1/K_v = 1$ (uncompensated) to 0.1 (lag-compensated), confirming the tenfold K_v improvement.

```
s = tf('s');
G = 1/(s*(s+1));

% Lag compensator
beta = 10;
T_lag = 10;
Clag = beta*(T_lag*s + 1)/(beta*T_lag*s + 1);

% Bode comparison
figure
margin(G), grid on, title('Uncompensated')
figure
margin(Clag*G), grid on, title('Lag-compensated')

% Verify Kv
Kv = dcgain(s*Clag*G);
fprintf('Kv = %.1f, ramp error = %.2f\n', Kv, 1/Kv)

% Closed-loop step response (unity feedback)
Tcl = feedback(Clag*G, 1);
figure
step(Tcl), grid on, title('Step response')
```

Listing 9.10: MATLAB code for the lag compensator design and verification.

9.10.5 Lag–lead compensation

When a plant has both poor steady-state accuracy *and* poor phase margin, a **lag–lead compensator** combines both sections:

$$C(s) = K_c \underbrace{\frac{T_1 s + 1}{\beta T_1 s + 1}}_{\text{lag section}} \underbrace{\frac{T_2 s + 1}{\alpha T_2 s + 1}}_{\text{lead section}}, \qquad \beta > 1, \quad 0 < \alpha < 1.$$

Example 9.10: Lag–lead design — complete worked example

Problem: In the same unity-feedback configuration, the plant is

$$G_p(s) = \frac{280(s + 0.5)}{s(s + 0.2)(s + 5)(s + 70)} = \frac{2\,(1 + s/0.5)}{s\,(1 + s/0.2)(1 + s/5)(1 + s/70)}.$$

This is a type-1 system (one free integrator), so there is zero steady-state error for a step and finite error for a ramp. The time-domain specifications are:

- overshoot $\leq$ 12% to a step command,
- settling time $t_s \leq 1.5$ s (2% criterion),
- steady-state ramp-tracking error $e_{ss} \leq 2\%$.

Solution:

Step 1: Translate to frequency-domain targets.

1. **Overshoot → phase margin.** From $M_p = e^{-\pi\zeta/\sqrt{1-\zeta^2}} \le 0.12$ we obtain $\zeta \ge 0.56$, giving
$$PM \ge 100\zeta \approx 56^\circ.$$
We round to $PM \ge 55^\circ$ as our design target.
2. **Settling time → crossover frequency.** Using $\omega_{gc} \gtrsim 4/(\zeta\, t_s)$:
$$\omega_{gc} \ge \frac{4}{0.56 \times 1.5} \approx 4.8 \text{ rad/s}.$$
We round up to target $\omega_{gc} = 5$ rad/s.
3. **Ramp error → velocity error constant.** For a type-1 system, $e_{ss} = 1/K_v \le 0.02$, so $K_v \ge 50$.
4. **Gain margin.** We adopt the standard default $GM \ge 6$ dB.

Step 2: Analyse the uncompensated plant. The velocity error constant of the plant alone is
$$K_{v,\text{plant}} = \lim_{s\to 0} s\, G_p(s) = \frac{280 \times 0.5}{0.2 \times 5 \times 70} = 2.$$
We need $K_v = 50$, so an overall gain increase of $50/2 = 25$ is required. From the uncompensated Bode plot (dashed red in Figure 9.34): $\omega_{gc} \approx 0.9$ rad/s and $PM \approx 63^\circ$. The phase margin is adequate, but ω_{gc} is far below the 5 rad/s target and K_v is far too low. Neither lag nor lead alone can satisfy all three specifications simultaneously, so a lag–lead compensator is required.

Step 3: Determine the gain distribution. At $\omega = 5$ rad/s the uncompensated magnitude is -18.9 dB, so the total gain needed at crossover is +18.9 dB to bring $|L(j5)| = 0$ dB. The lead section will contribute some of this gain at its peak; the remainder must come from the explicit gain K_c. We design the lead first, then allocate K_c and β accordingly.

Step 4: Design the lead section. At $\omega = 5$ rad/s, the uncompensated phase is -142.5°, giving a "would-be" phase margin of only 37.5°. The deficit is $55 - 37.5 = 17.5^\circ$. Adding a 5° safety margin for the negative phase contributed by the lag section near crossover, we target $\phi_{\max} = 22.5^\circ$:
$$\alpha = \frac{1 - \sin 22.5^\circ}{1 + \sin 22.5^\circ} \approx 0.447.$$
The lead section adds $10\log_{10}(1/0.447) \approx 3.5$ dB at its peak. We place this peak at the desired crossover $\omega_m = 5$ rad/s:
$$T_2 = \frac{1}{\omega_m\sqrt{\alpha}} = \frac{1}{5 \times 0.668} \approx 0.299 \text{ s}.$$
$$C_{\text{lead}}(s) = \frac{0.299s + 1}{0.134s + 1} \quad \text{(zero at 3.34, pole at 7.48 rad/s)}.$$

Step 5: Set K_c and design the lag section. Of the 18.9 dB needed at crossover, the lead provides 3.5 dB, so the explicit gain must supply $18.9 - 3.5 = 15.4$ dB, giving $K_c = 10^{15.4/20} \approx 5.89$. With this gain alone, $K_v = K_c \times 2 = 11.8$. The lag section must boost this to 50:
$$\beta = \frac{50}{11.8} \approx 4.24.$$

We place the lag zero two decades below crossover at $\omega_z = 0.05$ rad/s so that the negative phase of the lag section is concentrated well below ω_{gc}:

$$C_{\text{lag}}(s) = 4.24 \cdot \frac{20s+1}{84.8s+1} \quad \text{(zero at 0.05, pole at 0.012 rad/s).}$$

Step 6: Combine and verify.

$$C(s) = K_c \cdot C_{\text{lag}}(s) \cdot C_{\text{lead}}(s) = 5.89 \times 4.24 \cdot \frac{20s+1}{84.8s+1} \cdot \frac{0.299s+1}{0.134s+1}.$$

Figure 9.34 confirms: the intermediate loop $K_c C_{\text{lag}} G_p$ (orange dotted) raises K_v to 50 and moves ω_{gc} to 3.8 rad/s with $PM \approx 45^\circ$. Adding the lead section (blue solid) pushes ω_{gc} to exactly 5 rad/s with $PM \approx 60^\circ$ and $GM \approx 22$ dB — all specifications are met with margin.

Figure 9.35 compares the time responses. The compensated step response reaches steady state in about 3 s with only 11% overshoot ($\leq$ 12% spec met), versus the sluggish uncompensated response that takes over 8 s. The ramp response confirms the steady-state tracking: the uncompensated system lags the ramp by $1/K_v = 0.5$, while the compensated system tracks with an error of only $1/50 = 0.02$ ($\leq$ 2% spec met).

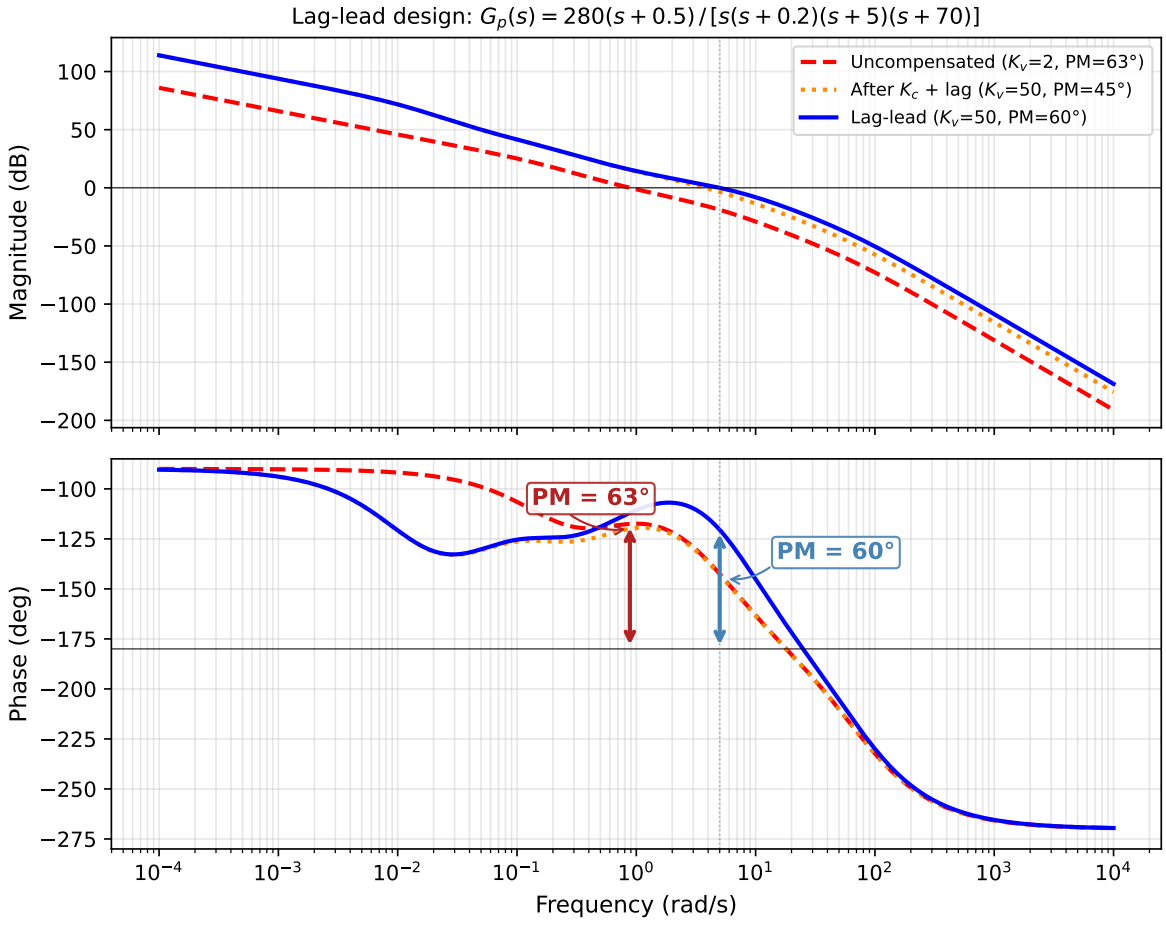


Figure 9.34: Bode plots for the lag–lead design. The lag section and gain K_c (orange, dotted) boost the low-frequency gain by 28 dB and push ω_{gc} from 0.9 to 3.8 rad/s. The lead section then moves the crossover to 5 rad/s while lifting the phase margin from 45° to 60°.

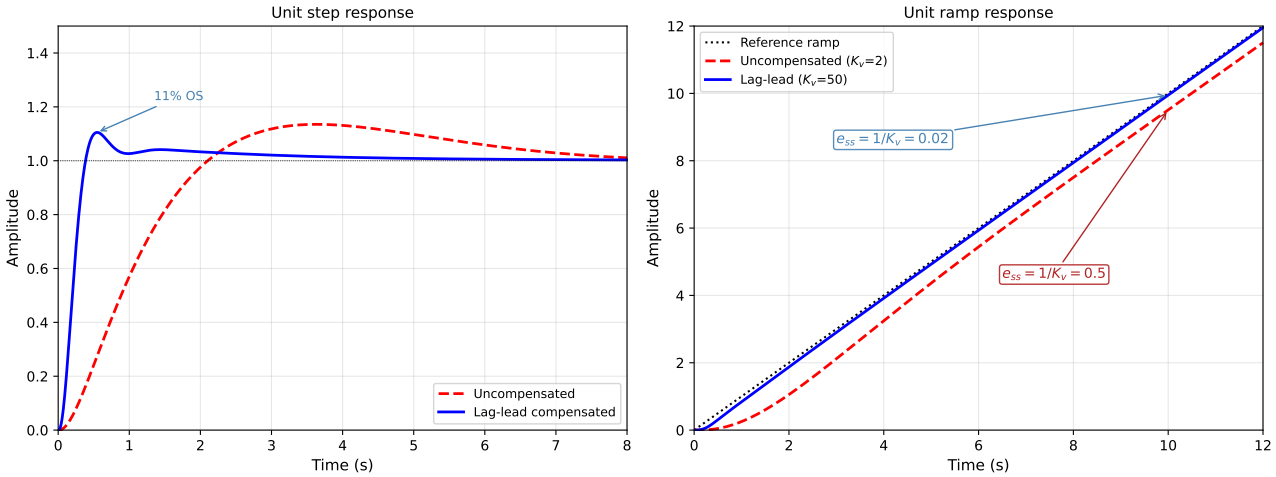


Figure 9.35: Closed-loop time responses for the lag–lead design (unity feedback). Left: step response — the compensated system settles in about 3 s with 11% overshoot. Right: ramp response — the uncompensated system ($K_v = 2$) lags the reference by $1/K_v = 0.5$, while the compensated system ($K_v = 50$) tracks with only 0.02 steady-state error.

The complete MATLAB code for this design is given in Listing 9.11.

```
s = tf('s');
Gp = 280*(s+0.5) / (s*(s+0.2)*(s+5)*(s+70));

% Uncompensated analysis
Kv_plant = dcgain(s*Gp);            % = 2
fprintf('Kv_plant = %.1f\n', Kv_plant)

% Lead section: phi_max = 22.5 deg at wm = 5 rad/s
alpha = 0.447;
T2 = 1/(5*sqrt(alpha));             % = 0.299 s
Clead = (T2*s + 1)/(alpha*T2*s + 1);

% Gain Kc to bring magnitude to 0 dB at wgc = 5
Kc = 5.89;

% Lag section: beta to raise Kv from Kc*2 to 50
beta = 50/(Kc*Kv_plant);            % = 4.24
Clag = beta*(20*s + 1)/(beta*20*s + 1);

% Combined compensator
C = Kc*Clag*Clead;

% Verify
Kv_final = dcgain(s*C*Gp);
fprintf('Kv = %.1f\n', Kv_final)

figure
margin(Gp), grid on, title('Uncompensated')
figure
margin(Kc*Clag*Gp), grid on, title('After Kc + lag')
figure
margin(C*Gp), grid on, title('Lag-lead compensated')

Tcl = feedback(C*Gp, 1);
figure, step(Tcl, 8), grid on
title('Closed-loop step response')
stepinfo(Tcl)
```

Listing 9.11: MATLAB code for the lag–lead compensator design and verification.

9.10.6 Complete design example: satellite antenna positioning

Example 9.11: Satellite antenna positioning system — from specs to compensator

A satellite ground-station antenna must track a slowly moving satellite. The unity-feedback system has plant transfer function (motor voltage to antenna angle)

$$G(s) = \frac{10}{s(s+2)(s+5)}.$$

Time-domain specifications: overshoot $\leq 25\%$; steady-state ramp error $\leq 10\%$.

Step 1: Translate to frequency-domain targets.

- Overshoot $\leq 25\% \Rightarrow \zeta \geq 0.40 \Rightarrow PM \geq 40^\circ$. With safety: target $PM \geq 45^\circ$.
- Ramp error $\leq 10\% \Rightarrow K_v \geq 10$. Uncompensated $K_v = 10/(2\times 5) = 1$, so we need a tenfold gain increase.

Step 2: Diagnose the uncompensated loop. From Figure 9.36 (dashed red): $PM \approx 56^\circ$ at $\omega_{gc} \approx 0.9$ rad/s, $GM \approx 17$ dB. Phase margin is adequate; K_v is not. The decision chart says: *try lag*.

Step 3: Attempt a lag compensator. Set $\beta = 10/1 = 10$. We place the lag zero at $\omega_z = 0.2$ rad/s rather than the usual "one decade below crossover" rule of thumb ($\omega_{gc}/10 \approx 0.09$), because the crossover is already low at 0.9 rad/s and pushing the zero to 0.09 would give an impractically slow pole at 0.009 rad/s. With $\omega_z = 0.2$:

$$C_{\text{lag}}(s) = 10 \cdot \frac{5s+1}{50s+1} \quad \text{(zero at 0.2, pole at 0.02 rad/s).}$$

Figure 9.36 (orange dotted) shows the intermediate result: $PM \approx 44^\circ$. The lag zero at only $\omega_{gc}/4.5$ — closer than ideal — contributes about 12° of negative phase near the crossover region, dropping PM below the 45° target. Lag alone is insufficient; we add a lead section.

Step 4: Add a lead section. The shortfall is only 1°, but a margin of 1° is unacceptable in practice (component tolerances, model uncertainty). We target $\phi_{\max} = 25^\circ$ to give a robust margin:

$$\alpha = \frac{1-\sin 25^\circ}{1+\sin 25^\circ} \approx 0.406.$$

The lead peak adds 3.9 dB. We place it where the intermediate loop magnitude equals -3.9 dB ($\omega_m \approx 1.3$ rad/s), so the lead pushes the crossover to that frequency:

$$T_2 = \frac{1}{1.3\times\sqrt{0.406}} \approx 1.21 \text{ s.} \qquad C_{\text{lead}}(s) = \frac{1.21s+1}{0.49s+1} \quad \text{(zero at 0.82, pole at 2.03).}$$

Step 5: Combine and verify.

$$C(s) = 10 \cdot \frac{5s+1}{50s+1} \cdot \frac{1.21s+1}{0.49s+1}.$$

Figure 9.36 (blue solid) confirms $PM \approx 60^\circ$ at $\omega_{gc} \approx 1.3$ rad/s, $GM \approx 14$ dB, $K_v = 10$. Figure 9.37 shows about 13% overshoot (well within the 25% limit) and a ramp tracking error that converges to $1/K_v = 10\%$.

The full MATLAB implementation is given in Listing 9.12.

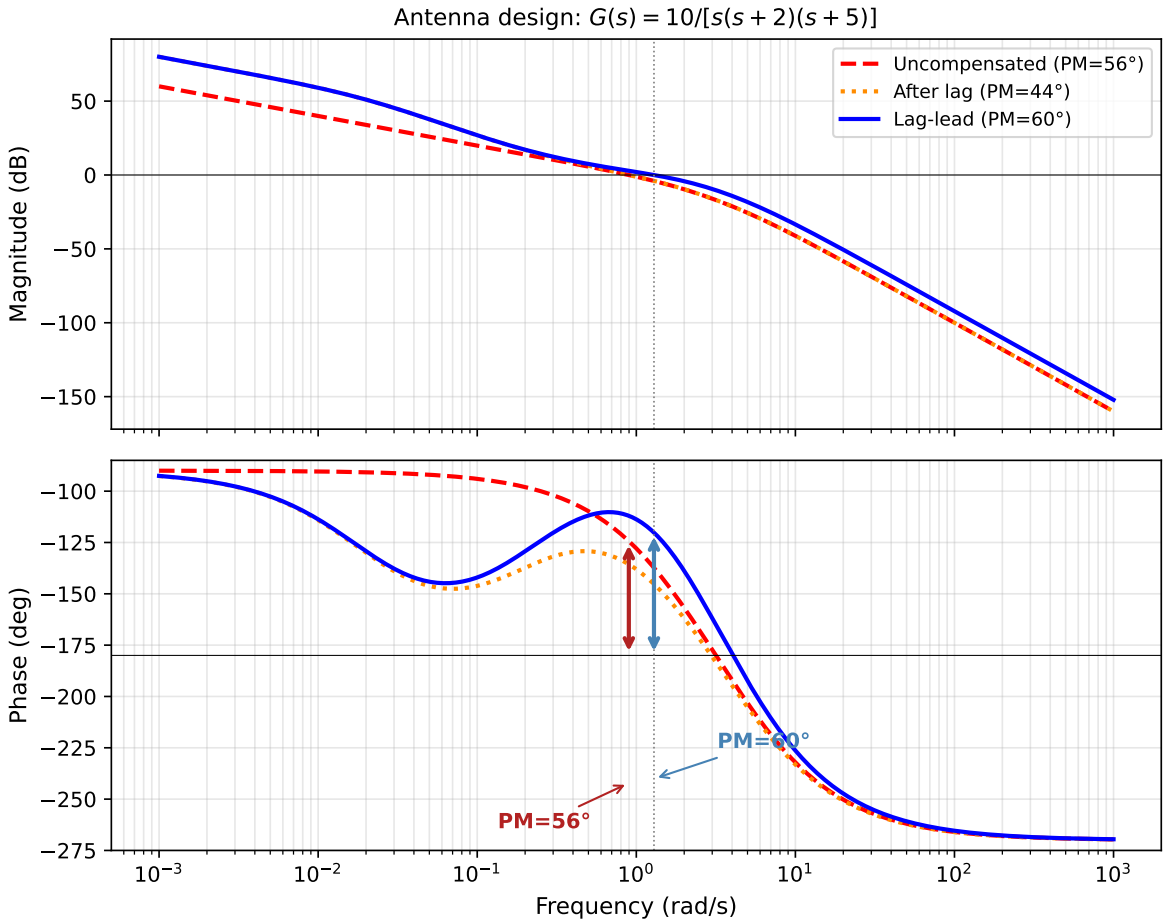


Figure 9.36: Bode plots for the antenna design. The lag section raises the low-frequency gain by 20 dB but costs 12° of PM because the lag zero is relatively close to crossover. The lead section restores the phase and pushes ω_{gc} from 0.9 to 1.3 rad/s.

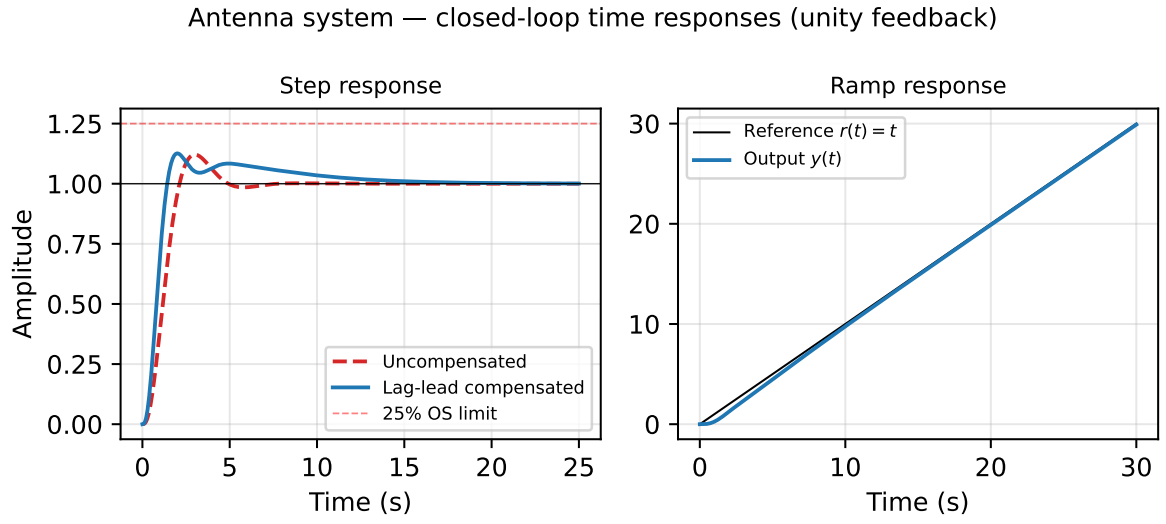


Figure 9.37: Closed-loop time responses for the antenna system (unity feedback). Left: step response with 13% overshoot. Right: ramp tracking — the output follows the ramp with a small steady-state lag of $1/K_v = 0.1$ s.

```
s = tf('s');
G = 10/(s*(s+2)*(s+5));

% Diagnose uncompensated loop
figure, margin(G), grid on, title('Uncompensated')
Kv_uncomp = dcgain(s*G);
fprintf('Uncompensated Kv = %.1f\n', Kv_uncomp)

% Lag section
beta = 10;
Clag_section = (5*s+1)/(50*s+1);
figure, margin(beta*Clag_section*G), grid on
title('After lag section')

% Lead section
alpha = 0.406;
T2 = 1.21;
Clead = (T2*s+1)/(alpha*T2*s+1);

% Combined lag-lead compensator
C = beta*Clag_section*Clead;
figure, margin(C*G), grid on, title('Lag-lead compensated')

% Closed-loop verification (unity feedback)
Tcl = feedback(C*G, 1);
figure, step(Tcl, 25), grid on
title('Closed-loop step response')
info = stepinfo(Tcl);
Kv = dcgain(s*C*G);
fprintf('Kv = %.1f, OS = %.1f%%, Ts = %.1f s\n', ...
        Kv, info.Overshoot, info.SettlingTime)
```

Listing 9.12: MATLAB code for the antenna positioning system design.

9.11 Summary

Chapter Summary

- Frequency-domain analysis studies how LTI systems respond to sinusoidal inputs at different frequencies. A stable LTI system changes the amplitude and phase of a sinusoid but not its frequency.
- Bode plots show magnitude in dB and phase in degrees against a logarithmic frequency axis. They are constructed by adding the contributions of basic building blocks: constant gain, integrators, first-order poles/zeros, and second-order pairs.
- The **gain crossover frequency** ω_{gc} (where $|L| = 0$ dB) determines the speed of the closed-loop response. The **phase crossover frequency** ω_{pc} (where $\angle L = -180^\circ$) marks the onset of potential positive-feedback instability.
- Gain margin and phase margin measure robustness against modelling errors. Typical targets are $PM \in [30^\circ, 60^\circ]$ and $GM \geq 6$ dB.
- The second-order approximation $PM \approx 100\zeta$ connects frequency-domain margins to time-domain behaviour. The crossover frequency is closely related to bandwidth ($\omega_{BW} \approx 1$–$2\ \omega_{gc}$), which determines rise time and settling time.

- Time-domain specifications (overshoot, settling time, steady-state error) are translated into frequency-domain targets (PM, ω_{gc}, K_v) before design, and verified in the time domain after design.
- A lead compensator adds phase near crossover and is used to improve phase margin. A lag compensator boosts low-frequency gain and is used to improve steady-state accuracy. A lag–lead compensator combines both when both are needed.
- The compensator selection follows a systematic decision chart based on two questions: is the phase margin adequate? is the steady-state accuracy adequate?

9.12 End-of-Chapter Problems

1. **(Decibel conversions)** Convert the following magnitude ratios into dB: 0.01, 0.1, 0.5, 1, 2, 10, 100. Then convert -14 dB and +26 dB back to magnitude ratios.
2. **(Sinusoidal steady state)** For $G(s) = 1/(s+2)$, compute $|G(j\omega)|$ and $\angle G(j\omega)$ at $\omega = 0.2$, 2, and 20 rad/s. For each frequency, write the steady-state output if $u(t) = 3\sin(\omega t)$.
3. **(SSS of a second-order system)** For $G(s) = 25/(s^2 + 4s + 25)$:
   (a) Find ω_n and ζ.
   (b) Compute $|G(j5)|$ and $\angle G(j5)$.
   (c) If $u(t) = \sin(5t)$, what is the steady-state output? Comment on the amplitude.
4. **(Asymptotic Bode sketch)** Sketch the asymptotic Bode magnitude plot of
$$G(s) = \frac{20}{s(1 + s/5)}.$$
State the slope in each frequency range. What is the magnitude at $\omega = 1$ rad/s and at $\omega = 50$ rad/s?
5. **(Bode form conversion)** Put the following transfer function into Bode form, identifying the Bode-form gain and all break frequencies:
$$G(s) = \frac{200(s+5)}{(s+2)(s^2 + 10s + 100)}.$$
6. **(Phase margin by hand)** For
$$L(s) = \frac{5}{s(s+1)},$$
find the gain crossover frequency and estimate the phase margin. Is the closed-loop step response expected to be oscillatory?
7. **(Gain margin by hand)** For
$$L(s) = \frac{K}{s(s+1)(s+4)},$$
with $K = 20$:
   (a) Find the phase crossover frequency ω_{pc} by solving $\angle L(j\omega) = -180^\circ$.
   (b) Compute $|L(j\omega_{pc})|$ and the gain margin in dB. What does this value of GM tell you about the closed-loop stability?

(c) What is the maximum value of K for which the closed loop remains stable?

8. **(MATLAB Bode analysis)** Use MATLAB to:

   (a) Plot the Bode diagram and margins of $G(s) = 10/((s+1)(s+5))$ using `margin()`.

   (b) Compute the closed-loop bandwidth using `bandwidth(feedback(G,1))`.

   (c) Plot the step response of the closed-loop system and compare the settling time with the bandwidth.

9. **(Lead compensator design)** For the plant $G(s) = 10/(s(s+2))$:

   (a) Find the uncompensated phase margin.

   (b) Design a lead compensator to achieve $PM \geq 50^\circ$. Show all steps: compute α, find ω_m, compute T, and write $C_{\text{lead}}(s)$.

   (c) Verify using MATLAB that the compensated phase margin meets the specification.

   (d) Compare the uncompensated and compensated step responses.

10. **(Lag compensator design)** For a unity-feedback system with plant $G(s) = 2/(s(s+1))$:

    (a) What is the current velocity error constant K_v?

    (b) Design a lag compensator to increase K_v to at least 20 while keeping the phase margin above 35°. *Hint:* the margin is tight — you may need to place the lag zero well below crossover and iterate.

    (c) Verify using MATLAB.

11. **(Lag–lead design)** For the plant $G(s) = 1/(s(s+1)(s/10+1))$, design a lag–lead compensator to achieve $K_v \geq 10$ and $PM \geq 45^\circ$. Document all design steps and verify in MATLAB with both Bode plots and a step response.

12. **(Bode plot interpretation)** The Bode magnitude plot of an unknown stable system shows the following features: a flat region at +6 dB for low frequencies, a break at $\omega = 2$ rad/s where the slope changes to -20 dB/decade, and another break at $\omega = 20$ rad/s where the slope changes to -40 dB/decade. Estimate the transfer function.

13. **(Time-to-frequency translation)** A servo system must have: overshoot $\leq 10\%$, settling time ≤ 1 s, and zero steady-state error to a step.

    (a) Find the required ζ and compute the phase margin target.

    (b) Estimate the required gain crossover frequency ω_{gc}.

    (c) What system type (number of integrators) is needed for zero step error?

    (d) If the plant is $G(s) = K/(s+4)(s+10)$, what value of K is needed, and is a compensator necessary? Verify with MATLAB.

14. **(Crossover frequency and speed)** Two unity-feedback systems have the same plant $G(s) = 1/(s(s+1))$ but different proportional gains: $K_1 = 2$ and $K_2 = 10$.

    (a) Find ω_{gc} for each case.

    (b) Predict which system responds faster and which has more overshoot, based on the crossover frequencies and phase margins.

    (c) Verify your predictions using MATLAB step responses.

15. **(Design decision)** For each of the following situations, state which compensator type (gain only, lead, lag, or lag–lead) you would choose and why:

    (a) $PM = 55^\circ$, $K_v = 2$, required $K_v \geq 20$.

(b) $PM = 15^\circ$, $K_v = 30$, required $PM \geq 45^\circ$ and $K_v \geq 20$.

(c) $PM = 12^\circ$, $K_v = 0.5$, required $PM \geq 40^\circ$ and $K_v \geq 10$.

(d) $PM = 50^\circ$, $K_v = 8$, required $PM \geq 45^\circ$ and $K_v \geq 5$.

16. **(Full design from time specs)** A position control system with plant

$$G(s) = \frac{5}{s(s+1)(s+10)}$$

must achieve: overshoot $\leq 20\%$, settling time ≤ 3 s, and ramp tracking error $\leq 4\%$.

(a) Translate the specifications into frequency-domain targets.

(b) Plot the uncompensated Bode diagram and identify the deficiencies.

(c) Select a compensator type using the decision chart.

(d) Design the compensator, showing all steps.

(e) Verify in MATLAB with Bode plot, margins, and step response.

Chapter 10

Controllability and Observability

> *"Before you try to control a system, first ask: can you actually control it? And before you try to estimate its state, ask: can you actually observe it?"*

State feedback control (Chapter 9) places closed-loop poles by feeding back the full state vector. But two fundamental questions must be answered first: *is it even possible to steer every state of the system using the available input?* And *can we determine all the internal states from the output measurements?*

These questions lead to **Controllability** and **Observability** — structural properties that determine whether a control or estimation design is feasible at all.

Learning Objectives

After completing this chapter, you should be able to:

- Explain what controllability and observability mean physically.
- Construct the controllability matrix and apply the rank test.
- Construct the Observability matrix and apply the rank test.
- Recognise uncontrollable and unobservable modes from block diagrams.
- Convert a transfer function to controllable and observable canonical forms.
- Use MATLAB to check controllability and observability.

10.1 Controllability

10.1.1 What does controllability mean?

Consider a state-space system

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad y(t) = Cx(t).$$

We say this system is **controllable** if, starting from *any* initial state $x(0)$, there exists an input $u(t)$ that can drive the state to *any* desired final state in finite time.

Definition 10.1: Controllability

The pair (A, B) is *controllable* if for any initial state x_0 and any target state x_f, there exists an input $u(t)$ defined over a finite interval $[0, t_f]$ that drives $x(0) = x_0$ to $x(t_f) = x_f$.

Think of it this way: controllability tells you whether the input has a "path" to influence every part of the system's internal state. If some state variable is completely disconnected from the input — no matter what you do with $u(t)$ — you cannot change it. That state is *uncontrollable*.

Example 10.1: Two-tank system

Consider two water tanks connected by a pipe (Figure 10.1). Water flows in through $u(t)$ to Tank 1, which feeds Tank 2 through the pipe. By adjusting $u(t)$, you can control the level in Tank 1 directly, and the level in Tank 2 indirectly (through Tank 1). The input can influence both states — the system is **controllable**.

Now imagine the pipe between the tanks is removed. Tank 2 just drains on its own. No matter how you adjust $u(t)$, you cannot change the level in Tank 2. The state x_2 is *uncontrollable*.

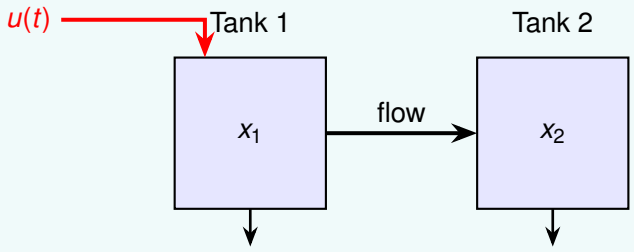


Figure 10.1: Two-tank system: the input $u(t)$ flows into Tank 1, which feeds Tank 2 through a pipe.

Example 10.2: The toothpaste analogy

Consider squeezing toothpaste from a tube onto a toothbrush (Figure 10.2). By squeezing the tube (your input u), you can move paste from the tube to the brush — you can drive the "paste on brush" state to any desired amount. But can you reverse the process? Can you put the toothpaste *back* into the tube?

No. Once the paste is out, you cannot return it. The "paste inside the tube" state is *not controllable* in the reverse direction — your input (squeezing) can only move paste one way. This is a system where the state space is only partially reachable from a given initial condition.

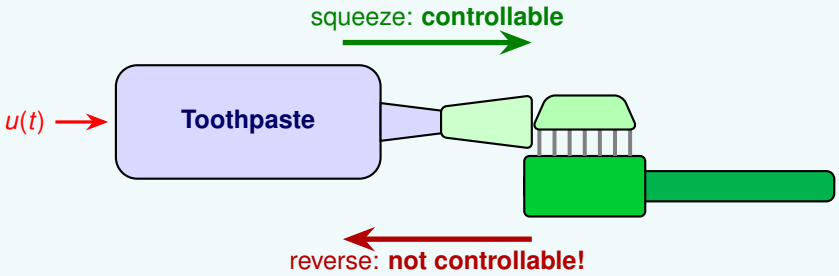


Figure 10.2: The toothpaste analogy: squeezing moves paste from tube to brush (controllable), but you cannot push it back (uncontrollable).

Analogy vs. formal definition

The toothpaste analogy is a loose illustration of irreversibility and reachability in *nonlinear*, *constrained* systems — it is **not** the formal definition of controllability for linear systems. In a controllable linear system $\dot{x} = Ax + Bu$, the input $u(t)$ can drive the state in *both* directions along every state dimension (positive and negative). There is no "one-way" restriction. The analogy is useful for building initial intuition, but do not carry its limitations into the mathematics.

10.1.2 The controllability matrix

How do we test controllability mathematically? For an n-th order system with state matrix $A \in \mathbb{R}^{n\times n}$ and input matrix $B \in \mathbb{R}^{n\times 1}$, we form the **Controllability matrix**:

Definition 10.2: Controllability Matrix

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$$

This is an $n \times n$ matrix (for a single-input system). The system is controllable if and only if

$$\text{rank}(\mathcal{C}) = n.$$

Remark 10.1

For a single-input system, $\mathcal{C}$ is an $n \times n$ square matrix. The rank test simplifies to: the system is controllable if and only if $\det(\mathcal{C}) \neq 0$.

Where does this matrix come from?

The solution of $\dot{x} = Ax + Bu$ is

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

The integral term represents what the input can contribute. By the Cayley–Hamilton theorem [5], e^{At} can be written as a polynomial in A of degree at most $n-1$. So the input can push the state in the directions $B, AB, A^2B, \ldots, A^{n-1}B$. If these n vectors span all of $\mathbb{R}^n$, the input can reach any point — the system is controllable.

Example 10.3: Second-order controllable system

Consider

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The controllability matrix is

$$\mathcal{C} = \begin{bmatrix} B & AB \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -3 \end{bmatrix}.$$

We compute $\det(\mathcal{C}) = (0)(-3) - (1)(1) = -1 \neq 0$, so the system is **controllable**.

Example 10.4: Second-order uncontrollable system

Now consider

$$A = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

This is a diagonal system with two decoupled modes. The input enters only through x_1:

$$\dot{x}_1 = -x_1 + u, \qquad \dot{x}_2 = -2x_2.$$

The controllability matrix is

$$\mathcal{C} = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}.$$

The second row is all zeros, so $\det(\mathcal{C}) = 0$ and $\operatorname{rank}(\mathcal{C}) = 1 < 2$. The system is **not controllable**. The state x_2 decays on its own and cannot be influenced by u.

The block diagram in Figure 10.3 shows this uncontrollable system. The second mode (x_2) is completely disconnected from the input.

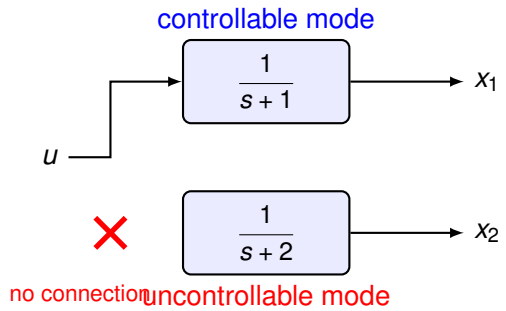


Figure 10.3: Block diagram of an uncontrollable system. The mode at $s = -2$ (state x_2) has no path from the input u.

Example 10.5: Third-order system

Consider a system with

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

We compute:

$$AB = \begin{bmatrix} 0 \\ 1 \\ -6 \end{bmatrix}, \qquad A^2B = A(AB) = \begin{bmatrix} 1 \\ -6 \\ 25 \end{bmatrix}.$$

So

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & -6 \\ 1 & -6 & 25 \end{bmatrix}.$$

Computing the determinant (expand along the first column):

$$\det(\mathcal{C}) = 1 \cdot \det \begin{bmatrix} 0 & 1 \\ 1 & -6 \end{bmatrix} = 1 \cdot (0 - 1) = -1 \neq 0.$$

The system is controllable.

10.1.3 MATLAB commands for controllability

Listing 10.1 shows how to form the controllability matrix and test its rank in MATLAB.

```
A = [0 1 0; 0 0 1; -6 -11 -6];
B = [0; 0; 1];

% Form controllability matrix
Co = ctrb(A, B);

% Check rank
n = size(A,1);
r = rank(Co);
fprintf('System order = %d, rank(C) = %d\n', n, r);
if r == n
    fprintf('System is controllable.\n');
else
    fprintf('System is NOT controllable.\n');
end
```

Listing 10.1: Checking controllability in MATLAB

Common Mistake

MATLAB's `rank()` uses a numerical tolerance. For a matrix that is *nearly* singular, the result depends on rounding. Always check the singular values (`svd(Co)`) if you are unsure — a very small singular value suggests the system is "almost uncontrollable".

10.2 Observability

10.2.1 What does observability mean?

Observability is the dual of controllability: can we *determine* every state by observing the output?

Definition 10.3: Observability

The pair (A, C) is *observable* if the initial state $x(0)$ can be uniquely determined from the output $y(t)$ and input $u(t)$ measured over a finite time interval $[0, t_f]$.

If a system is observable, then every internal state leaves a "fingerprint" on the output. If it is not, some internal dynamics are invisible — they evolve without affecting the output at all.

Example 10.6: Building with a hidden room

Imagine a building with temperature sensors only in certain rooms. If Room A and Room B are connected (heat flows between them), a sensor in Room A can detect temperature changes in Room B indirectly. But if Room C is completely insulated from all monitored rooms, its temperature is invisible to the sensors — it is an *unobservable* state.

10.2.2 The observability matrix

Definition 10.4: Observability Matrix

For a system with $A \in \mathbb{R}^{n\times n}$ and $C \in \mathbb{R}^{1\times n}$, the observability matrix is

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}.$$

The system is observable if and only if $\text{rank}(\mathcal{O}) = n$.

The logic mirrors controllability. The output $y(t) = Ce^{At}x(0) + \ldots$ contains information about $x(0)$ through the terms $Cx(0)$, $CAx(0)$, $CA^2x(0)$, and so on. If these row vectors span all of $\mathbb{R}^n$, we can reconstruct $x(0)$ uniquely from $y(t)$.

Example 10.7: Observable system

Take

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

The observability matrix is

$$\mathcal{O} = \begin{bmatrix} C \\ CA \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Since $\det(\mathcal{O}) = 1 \neq 0$, the system is **observable**. We can determine both states from the output $y = x_1$.

Example 10.8: Unobservable system

Consider the diagonal system

$$A = \begin{bmatrix} -1 & 0 \\ 0 & -3 \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

The output is $y = x_1$. We cannot see x_2 at all:

$$\mathcal{O} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}.$$

$\det(\mathcal{O}) = 0$, so the system is **not observable**. The mode x_2 is hidden from the output.

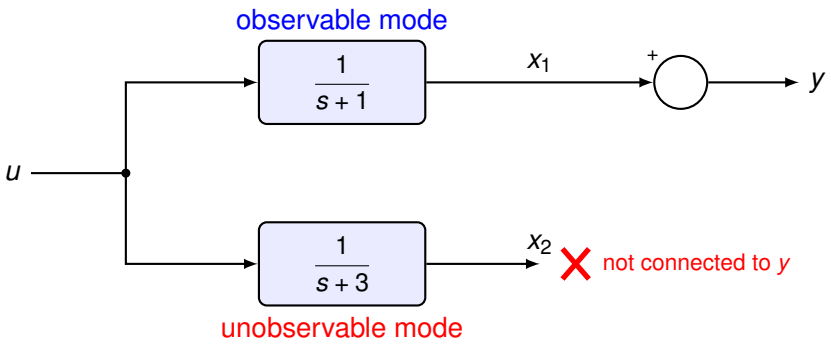


Figure 10.4: Block diagram of an unobservable system. Both modes are excited by the input, but only x_1 contributes to the output y. The mode at $s = -3$ is unobservable.

10.2.3 MATLAB commands for observability

Listing 10.2 shows the corresponding procedure for observability.

```
A = [-1 0; 0 -3];
C = [1 0];

% Form observability matrix
Ob = obsv(A, C);

% Check rank
n = size(A,1);
r = rank(Ob);
fprintf('System order = %d, rank(O) = %d\n', n, r);
if r == n
    fprintf('System is observable.\n');
else
    fprintf('System is NOT observable.\n');
end
```

Listing 10.2: Checking observability in MATLAB

10.3 Duality Between Controllability and Observability

Controllability and observability are directly connected.

Theorem 10.1: Duality

The pair (A, B) is **controllable** if and only if the pair $(A^\top, B^\top)$ is **observable**.

To see why, note that the controllability matrix of (A, B) is

$$\mathcal{C} = \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix}$$

while the observability matrix of $(A^\top, B^\top)$ is

$$\mathcal{O}_{\text{dual}} = \begin{bmatrix} B^\top \\ B^\top A^\top \\ \vdots \\ B^\top (A^\top)^{n-1} \end{bmatrix} = \mathcal{C}^\top.$$

Since transposing does not change the rank, the two conditions are equivalent.

Remark 10.2

Duality means that any theorem or method for controllability has a mirror version for observability (and vice versa). In Chapter 10, we will use this: observer design is "dual" to state feedback design.

10.4 Transfer Function and Pole-Zero Cancellation

Consider a state-space model (A, B, C, D) with transfer function

$$G(s) = C(sI - A)^{-1}B + D.$$

The eigenvalues of A are the poles of the state-space model. However, the transfer function may have *fewer* poles than A has eigenvalues. Why? Because uncontrollable or unobservable modes *cancel* in the transfer function.

Example 10.9: Pole-zero cancellation from uncontrollable mode

Consider the system

$$A = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad D = 0.$$

The eigenvalues of A are -1 and -2. But the transfer function is

$$G(s) = C(sI - A)^{-1}B = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{s+1} & 0 \\ 0 & \frac{1}{s+2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{s+1}.$$

The pole at $s = -2$ has disappeared! It corresponds to the uncontrollable state x_2 (which B does not reach). The transfer function only "sees" the controllable and observable part of the system.

Hidden Dynamics

If a system has uncontrollable or unobservable modes, the transfer function hides them. This is dangerous: an unstable hidden mode would not appear in the transfer function, but the system would still be internally unstable. Always check controllability and observability before relying solely on transfer function analysis.

Listing 10.3 compares the state-space poles with the transfer function poles.

```
A = [-1 0; 0 -2]; B = [1; 0]; C = [1 1]; D = 0;
sys_ss = ss(A, B, C, D);
sys_tf = tf(sys_ss);      % Transfer function: 1/(s+1)

% The pole at s=-2 is missing from the TF!
fprintf('State-space poles: '); disp(eig(A)');
fprintf('Transfer function poles: '); disp(pole(sys_tf)');

% Check controllability
```

```
Co = ctrb(A, B);
fprintf('rank(Co) = %d (need %d for controllability)\n', rank(Co), size(A,1));
```

Listing 10.3: Detecting hidden modes

10.5 Canonical Forms

A transfer function can be realised in many different state-space forms. Two are particularly important: the **Controllable canonical form** (CCF) and the **Observable canonical form** (OCF), which provide systematic realisations guaranteed to be controllable or observable, respectively.

10.5.1 Controllable canonical form (CCF)

Given a transfer function

$$G(s) = \frac{b_1 s^{n-1} + b_2 s^{n-2} + \cdots + b_n}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_n}$$

the controllable canonical form is:

Definition 10.5: Controllable Canonical Form

$$A_c = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$$C_c = \begin{bmatrix} b_n & b_{n-1} & \cdots & b_1 \end{bmatrix}, \quad D_c = 0$$

Notice the structure: the denominator coefficients $a_1, \ldots, a_n$ appear in the last row of A_c, and B_c has a 1 only in the last entry. This structure guarantees controllability.

Example 10.10: CCF for a second-order system

Given

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

we identify $a_1 = 4$, $a_2 = 3$, $b_1 = 3$, $b_2 = 5$. The CCF is:

$$A_c = \begin{bmatrix} 0 & 1 \\ -3 & -4 \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C_c = \begin{bmatrix} 5 & 3 \end{bmatrix}, \quad D_c = 0.$$

Let us verify controllability:

$$\mathcal{C} = \begin{bmatrix} B_c & A_c B_c \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -4 \end{bmatrix}, \quad \det(\mathcal{C}) = -1 \neq 0. \quad \checkmark$$

10.5.2 Observable canonical form (OCF)

The observable canonical form is the *dual* of the CCF. For the same transfer function $G(s)$:

Definition 10.6: Observable Canonical Form

$$A_o = \begin{bmatrix} 0 & 0 & \cdots & 0 & -a_n \\ 1 & 0 & \cdots & 0 & -a_{n-1} \\ 0 & 1 & \cdots & 0 & -a_{n-2} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & -a_1 \end{bmatrix}, \quad B_o = \begin{bmatrix} b_n \\ b_{n-1} \\ \vdots \\ b_1 \end{bmatrix}$$

$$C_o = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix}, \quad D_o = 0$$

Notice that $A_o = A_c^\top$, $B_o = C_c^\top$, $C_o = B_c^\top$. This is the duality in action. The OCF guarantees observability.

Example 10.11: OCF for the same second-order system

For $G(s) = \dfrac{3s+5}{s^2+4s+3}$, the OCF is:

$$A_o = \begin{bmatrix} 0 & -3 \\ 1 & -4 \end{bmatrix}, \quad B_o = \begin{bmatrix} 5 \\ 3 \end{bmatrix}, \quad C_o = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad D_o = 0.$$

Verify observability:

$$\mathcal{O} = \begin{bmatrix} C_o \\ C_o A_o \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -4 \end{bmatrix}, \quad \det(\mathcal{O}) = -1 \neq 0. \quad \checkmark$$

10.5.3 Diagonal canonical form

When the system has distinct (non-repeated) poles $p_1, p_2, \ldots, p_n$, we can also write the state-space model in **diagonal form** by performing a partial fraction expansion of $G(s)$:

$$G(s) = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + \cdots + \frac{r_n}{s-p_n}$$

where r_i are the residues. The diagonal realisation is

$$A_d = \begin{bmatrix} p_1 & & & \\ & p_2 & & \\ & & \ddots & \\ & & & p_n \end{bmatrix}, \quad B_d = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \quad C_d = \begin{bmatrix} r_1 & r_2 & \cdots & r_n \end{bmatrix}.$$

In this form, each state corresponds directly to one pole. This makes it easy to see which modes are controllable (nonzero entry in B_d) and observable (nonzero entry in C_d).

Example 10.12: Diagonal form

For $G(s) = \dfrac{3s+5}{s^2+4s+3} = \dfrac{3s+5}{(s+1)(s+3)}$, partial fractions give

$$G(s) = \frac{1}{s+1} + \frac{2}{s+3}.$$

So

$$A_d = \begin{bmatrix} -1 & 0 \\ 0 & -3 \end{bmatrix}, \quad B_d = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad C_d = \begin{bmatrix} 1 & 2 \end{bmatrix}.$$

Both entries of B_d are nonzero (controllable) and both entries of C_d are nonzero (observable).

10.5.4 MATLAB: canonical form conversions

Listing 10.4 demonstrates how to obtain different canonical forms and verify them using partial fractions.

```
% Define transfer function
num = [3 5]; den = [1 4 3];
sys = tf(num, den);

% Convert to state-space (MATLAB uses CCF by default for SISO)
[A, B, C, D] = tf2ss(num, den);
fprintf('tf2ss result (CCF-like):\n');
disp(A); disp(B'); disp(C);

% Convert to different canonical forms using canon()
sys_ss = ss(sys);
sys_modal = canon(sys_ss, 'modal');   % diagonal/modal form
fprintf('Modal form A matrix:\n');
disp(sys_modal.A);

% Alternative: use residue for partial fractions
[r, p, k] = residue(num, den);
fprintf('Residues: '); disp(r');
fprintf('Poles: '); disp(p');
```

Listing 10.4: Canonical forms in MATLAB

MATLAB Note

MATLAB's `tf2ss()` returns a form where the denominator coefficients are in the *first* row of A (not the last row as in our textbook CCF). The two forms are related by reversing the order of states. Both are valid controllable canonical forms.

10.6 Controllability and Observability of Specific Forms

The canonical forms highlight an important pattern:

Controllability/Observability of Canonical Forms

Form	Controllable?	Observable?
Controllable canonical form (CCF)	Always	Depends on numerator
Observable canonical form (OCF)	Depends on numerator	Always
Diagonal form (distinct poles)	Iff all B_d entries $\neq 0$	Iff all C_d entries $\neq 0$

For state feedback design (Chapter 9), we need the system to be *controllable*. For observer design (Chapter 10), we need it to be *observable*. For a combined observer-based state feedback controller, we need *both*.

Example 10.13: DC motor: controllability and observability analysis

A simplified DC motor has the transfer function

$$G(s) = \frac{5}{s^2 + 6s + 5} = \frac{5}{(s+1)(s+5)}.$$

Step 1: Controllable canonical form.

With $a_1 = 6$, $a_2 = 5$, $b_1 = 0$, $b_2 = 5$:

$$A_c = \begin{bmatrix} 0 & 1 \\ -5 & -6 \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C_c = \begin{bmatrix} 5 & 0 \end{bmatrix}.$$

Step 2: Controllability check.

$$\mathcal{C} = \begin{bmatrix} 0 & 1 \\ 1 & -6 \end{bmatrix}, \quad \det(\mathcal{C}) = -1 \neq 0 \quad \Rightarrow \quad \text{controllable.}$$

Step 3: Observability check.

$$\mathcal{O} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}, \quad \det(\mathcal{O}) = 25 \neq 0 \quad \Rightarrow \quad \text{observable.}$$

Step 4: Diagonal form. Partial fractions:

$$G(s) = \frac{5}{(s+1)(s+5)} = \frac{5/4}{s+1} + \frac{-5/4}{s+5}.$$

So $r_1 = 5/4$, $r_2 = -5/4$, and

$$A_d = \begin{bmatrix} -1 & 0 \\ 0 & -5 \end{bmatrix}, \quad B_d = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad C_d = \begin{bmatrix} 5/4 & -5/4 \end{bmatrix}.$$

Both B_d entries are nonzero $\Rightarrow$ controllable. Both C_d entries are nonzero $\Rightarrow$ observable.

Step 5: MATLAB verification. Listing 10.5 confirms both properties.

```
G = tf(5, [1 6 5]);
sys = ss(G);
fprintf('Contr: %d\n', rank(ctrb(sys.A, sys.B)) == size(sys.A,1));
fprintf('Observ: %d\n', rank(obsv(sys.A, sys.C)) == size(sys.A,1));

```

Listing 10.5: DC motor controllability and observability check

Both tests return 1 (true). This motor can be controlled by state feedback and its states can be estimated by an observer.

10.6.1 Kalman Decomposition (Conceptual Overview)

A result due to Rudolf Kalman states that *any* state-space system can be decomposed — via a change of coordinates — into four subsystems:

1. **Controllable and observable** — this part fully appears in the input–output transfer function.
2. **Controllable but unobservable** — the input can reach these states, but they do not affect the measured output.
3. **Uncontrollable but observable** — these states influence the output, but the input cannot steer them.
4. **Uncontrollable and unobservable** — completely hidden from both input and output.

Only subsystem (1) — the controllable *and* observable part — appears in the transfer function $G(s) = C(sI - A)^{-1}B$. The remaining modes are "invisible" to input–output experiments. This is precisely why *pole–zero cancellations* in a transfer function can hide unstable or uncontrollable modes.

Remark 10.3

The practical lesson is clear: **never rely on the transfer function alone** to judge a system's behaviour. A transfer function that looks perfectly stable and well-behaved may conceal uncontrollable or unobservable modes that cause problems in practice (e.g. internal instability). Always verify controllability and observability from the state-space matrices (A, B, C).

10.7 Summary

Chapter Summary

- **Controllability** answers: can the input influence every state? Test: $\text{rank}(\mathcal{C}) = n$ where $\mathcal{C} = [B\ AB\ \cdots\ A^{n-1}B]$.

- **Observability** answers: can we determine every state from the output? Test: $\text{rank}(\mathcal{O}) = n$ where $\mathcal{O} = [C;\ CA;\ \cdots;\ CA^{n-1}]$.
- **Duality**: (A, B) controllable $\Leftrightarrow$ $(A^\top, B^\top)$ observable.
- **Hidden modes**: uncontrollable or unobservable poles cancel in the transfer function. Always check state-space properties, not just the transfer function.
- **Controllable canonical form (CCF)**: denominator coefficients in the last row of A; always controllable.
- **Observable canonical form (OCF)**: transpose of CCF; always observable.
- **Diagonal form**: each state maps to one pole; controllability and observability are visible from B and C entries.
- MATLAB: use `ctrb()`, `obsv()`, `rank()`, `canon()`, `tf2ss()`.

10.8 End-of-Chapter Problems

1. **(Controllability — 2nd order)** Given

$$A = \begin{bmatrix} 0 & 1 \\ -5 & -6 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

   (a) Compute the controllability matrix $\mathcal{C}$.

   (b) Determine whether the system is controllable.

   (c) If you change B to $\begin{bmatrix} 1 \\ -5 \end{bmatrix}$, is the system still controllable? Explain.

2. **(Observability — 2nd order)** For the system in Problem 1 (original B) with $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$:

   (a) Compute the observability matrix $\mathcal{O}$.

   (b) Is the system observable?

   (c) Find a C matrix that makes the system *not* observable.

3. **(Diagonal system)** Consider

$$A = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -5 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}.$$

   (a) Which modes are controllable? Which are uncontrollable?

   (b) Which modes are observable? Which are unobservable?

   (c) What is the transfer function $G(s) = C(sI - A)^{-1}B$? How many poles does it have?

4. **(CCF construction)** The transfer function of a system is

$$G(s) = \frac{2s + 7}{s^3 + 5s^2 + 8s + 4}.$$

   (a) Write the CCF matrices A_c, B_c, C_c.

(b) Write the OCF matrices A_o, B_o, C_o.

(c) Verify that $A_o = A_c^\top$, $B_o = C_c^\top$, $C_o = B_c^\top$.

5. (Pole-zero cancellation) Consider

$$A = \begin{bmatrix} -1 & 0 \\ 0 & -4 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 \end{bmatrix}.$$

(a) Find the eigenvalues of A.

(b) Compute the transfer function $G(s) = C(sI - A)^{-1}B$.

(c) Explain why both eigenvalues vanish from $G(s)$. Is this system controllable? Observable?

6. (Third-order system) A system has

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -4 & -8 & -5 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 4 & 0 & 0 \end{bmatrix}.$$

(a) Verify that this is in CCF. What is the corresponding transfer function?

(b) Check controllability and observability using the rank test.

(c) Find the diagonal form using partial fraction expansion of $G(s)$.

7. (Mass–spring–damper revisited) A mass–spring–damper with $m = 1$, $c = 3$, $k = 2$ has state-space matrices

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

(a) Show the system is controllable for $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$ (position output). Is it observable?

(b) Repeat for $C = \begin{bmatrix} 0 & 1 \end{bmatrix}$ (velocity output).

(c) Can we design both a state feedback controller and an observer using position measurement only? Justify.

8. (MATLAB) For the system

$$G(s) = \frac{s + 3}{s^3 + 6s^2 + 11s + 6}$$

(a) Use MATLAB to obtain the CCF (`tf2ss`) and the modal form (`canon`).

(b) Use `ctrb()`, `obsv()`, and `rank()` to check controllability and observability of each form.

(c) Factor the denominator and find the partial fraction expansion using `residue()`. Verify that the modal form matches.

Chapter 11

State Feedback Control

> *"With PID, you tune three knobs and hope for the best. With state feedback, you choose exactly where the closed-loop poles go."*

Instead of feeding back only the output error, State feedback feeds back the *entire state vector* $x(t)$. This gives n gains to choose (one per state variable), which is exactly enough to place all n closed-loop poles at any desired locations — provided the system is **controllable** (Chapter 8).

Learning Objectives

After completing this chapter, you should be able to:

- Explain the state feedback control law $u = -Kx + r$ and derive the closed-loop system matrix $A - BK$.
- Translate time-domain specifications (settling time, overshoot) into desired pole locations.
- Compute the feedback gain K by direct comparison of characteristic polynomials.
- Apply Ackermann's formula to find K for single-input systems.
- Use MATLAB's `place()` and `acker()` commands for pole placement.
- Design state feedback with integral action to achieve zero steady-state tracking error.

11.1 The State Feedback Control Law

Consider a controllable system in state-space form:

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad y(t) = Cx(t).$$

The **state feedback** control law is:

Definition 11.1: State Feedback Control Law

$$u(t) = -Kx(t) + r(t)$$

where $K \in \mathbb{R}^{1\times n}$ is the *feedback gain vector* and $r(t)$ is the reference input.

Substituting into the state equation:

$$\dot{x} = Ax + B(-Kx + r) = (A - BK)x + Br.$$

The closed-loop system matrix is $A_{cl} = A - BK$. The closed-loop poles are the eigenvalues of $A - BK$.

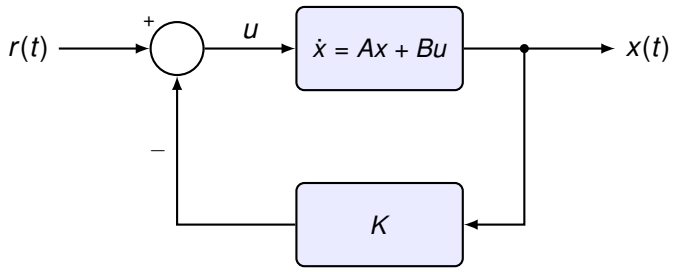


Figure 11.1: State feedback control: the full state vector $x(t)$ is fed back through gain K.

Remark 11.1

The key result: the closed-loop characteristic polynomial is

$$\det(sI - A + BK) = 0.$$

By choosing K, we can place the roots of this polynomial (the closed-loop poles) at any desired locations, **if and only if** (A, B) is controllable.

11.1.1 Why does controllability matter?

If the system is not controllable, some modes cannot be influenced by the input. No choice of K can move those poles. Controllability (Chapter 8) is therefore a prerequisite for Pole placement.

11.2 Pole Placement by Direct Comparison

The simplest method to find K is to compare the closed-loop characteristic polynomial with the desired one.

11.2.1 Design procedure

1. **Choose desired poles** $p_1, p_2, \dots, p_n$ based on performance specifications (settling time, damping, etc.).
2. **Form the desired characteristic polynomial:**
$$\alpha_d(s) = (s - p_1)(s - p_2)\cdots(s - p_n) = s^n + \alpha_1 s^{n-1} + \cdots + \alpha_n.$$
3. **Compute the closed-loop characteristic polynomial** $\det(sI - A + BK)$ symbolically in terms of $K = [k_1 \;\; k_2 \;\; \cdots \;\; k_n]$.
4. **Match coefficients:** equate the coefficients of each power of s to get n equations in n unknowns $k_1, \dots, k_n$.

Example 11.1: Second-order pole placement

Consider the mass–spring–damper from Chapter 3:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

The open-loop poles are the eigenvalues of A:

$$\det(sI - A) = s^2 + 3s + 2 = (s + 1)(s + 2).$$

So the open-loop poles are at $s = -1$ and $s = -2$ (stable, but slow).

Specification: We want the closed-loop system to have $\zeta = 0.7$ and $\omega_n = 5$ rad/s. The desired poles are:

$$p_{1,2} = -\zeta\omega_n \pm j\omega_n\sqrt{1 - \zeta^2} = -3.5 \pm j3.57.$$

The desired characteristic polynomial is:

$$\alpha_d(s) = s^2 + 2\zeta\omega_n s + \omega_n^2 = s^2 + 7s + 25.$$

Find K: Let $K = [k_1 \;\; k_2]$. Then

$$A - BK = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} k_1 & k_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -2 - k_1 & -3 - k_2 \end{bmatrix}.$$

The closed-loop characteristic polynomial is:

$$\det(sI - A + BK) = s^2 + (3 + k_2)s + (2 + k_1).$$

Matching with $s^2 + 7s + 25$:

$$\begin{aligned} 3 + k_2 = 7 \quad &\Rightarrow \quad k_2 = 4, \\ 2 + k_1 = 25 \quad &\Rightarrow \quad k_1 = 23. \end{aligned}$$

Therefore $K = \begin{bmatrix} 23 & 4 \end{bmatrix}$.

Verification:

$$A_{cl} = A - BK = \begin{bmatrix} 0 & 1 \\ -25 & -7 \end{bmatrix}.$$

Eigenvalues: $s^2 + 7s + 25 = 0$ gives $s = -3.5 \pm j3.57$. ✓

11.2.2 Choosing desired poles from specifications

How do we translate time-domain specifications into pole locations? For a dominant second-order system:

Pole Location from Specifications

- **Settling time** (2% criterion): $t_s \approx \dfrac{4}{\zeta\omega_n} \quad \Rightarrow \quad \sigma = \zeta\omega_n \approx \dfrac{4}{t_s}$
- **Overshoot:** $M_p = e^{-\pi\zeta/\sqrt{1-\zeta^2}} \times 100\% \quad \Rightarrow \quad \zeta = \dfrac{-\ln(M_p/100)}{\sqrt{\pi^2 + \ln^2(M_p/100)}}$
- **Desired poles:** $p_{1,2} = -\sigma \pm j\omega_d$ where $\omega_d = \sigma\dfrac{\sqrt{1-\zeta^2}}{\zeta}$

Example 11.2: Design from specifications

A system has

$$A = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Specification: settling time $t_s \leq 1$ s, overshoot $M_p \leq 10\%$.

From $M_p = 10\%$: $\zeta = \dfrac{-\ln(0.1)}{\sqrt{\pi^2 + \ln^2(0.1)}} \approx 0.59$.

From $t_s = 1$ s: $\sigma = 4/1 = 4$, so $\omega_n = \sigma/\zeta = 4/0.59 \approx 6.78$ rad/s.

Desired poles: $p_{1,2} = -4 \pm j5.47$.

Desired characteristic polynomial: $s^2 + 8s + 46$.

With $K = [k_1 \; k_2]$:

$$\det(sI - A + BK) = s^2 + (1 + k_2)s + (1 + k_1).$$

Matching: $k_2 = 7$, $k_1 = 45$. So $K = [45 \; 7]$.

11.3 Pole Placement Using Controllable Canonical Form

When the system is already in controllable canonical form (CCF), the gain computation becomes trivial. Recall from Chapter 8 that the CCF has

$$A_c = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & \cdots & -a_1 \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

With $K_c = [k_1 \; k_2 \; \cdots \; k_n]$, only the last row of A_c changes:

$$A_c - B_c K_c = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ -a_n - k_1 & -a_{n-1} - k_2 & \cdots & -a_1 - k_n \end{bmatrix}.$$

The closed-loop characteristic polynomial is

$$s^n + (a_1 + k_n)s^{n-1} + \cdots + (a_n + k_1).$$

Comparing with the desired polynomial $s^n + \alpha_1 s^{n-1} + \cdots + \alpha_n$, each gain is simply:

Remark 11.2

In controllable canonical form, the feedback gains are

$$k_i = \alpha_{n+1-i} - a_{n+1-i}, \qquad i = 1, \ldots, n.$$

In other words, subtract the original coefficients from the desired ones.

Example 11.3: Third-order system in CCF

A system in CCF has

$$A_c = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

The open-loop characteristic polynomial is $s^3 + 6s^2 + 11s + 6 = (s+1)(s+2)(s+3)$.

Desired poles: $s = -5, -5, -5$ (all poles at -5 for fast decay).

Desired polynomial: $(s+5)^3 = s^3 + 15s^2 + 75s + 125$.

The gains are:

$$\begin{aligned} k_1 &= 125 - 6 = 119, \\ k_2 &= 75 - 11 = 64, \\ k_3 &= 15 - 6 = 9. \end{aligned}$$

So $K_c = \begin{bmatrix} 119 & 64 & 9 \end{bmatrix}$.

For systems not in CCF, you can first transform to CCF, find K_c there, and then transform K_c back to the original coordinates. However, Ackermann's formula (next section) does this automatically.

11.4 Ackermann's Formula

For a single-input system, **Ackermann's formula** gives the feedback gain directly, without needing to transform to CCF:

Definition 11.2: Ackermann's Formula

$$K = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \mathcal{C}^{-1} \alpha_d(A)$$

where $\mathcal{C} = [B \; AB \; \cdots \; A^{n-1}B]$ is the controllability matrix and

$$\alpha_d(A) = A^n + \alpha_1 A^{n-1} + \cdots + \alpha_n I$$

is the desired characteristic polynomial evaluated at the matrix A.

The formula requires $\mathcal{C}$ to be invertible, which is guaranteed if and only if the system is controllable.

Example 11.4: Ackermann's formula for 2nd-order system

Revisit Example 11.1: $A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, desired polynomial $\alpha_d(s) = s^2 + 7s + 25$.

Step 1: Controllability matrix:

$$\mathcal{C} = \begin{bmatrix} 0 & 1 \\ 1 & -3 \end{bmatrix}, \quad \mathcal{C}^{-1} = \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix}.$$

Step 2: Evaluate $\alpha_d(A)$:

$$\alpha_d(A) = A^2 + 7A + 25I = \begin{bmatrix} -2 & -3 \\ 6 & 7 \end{bmatrix} + \begin{bmatrix} 0 & 7 \\ -14 & -21 \end{bmatrix} + \begin{bmatrix} 25 & 0 \\ 0 & 25 \end{bmatrix} = \begin{bmatrix} 23 & 4 \\ -8 & 11 \end{bmatrix}.$$

Step 3: Apply the formula:

$$K = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 23 & 4 \\ -8 & 11 \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 61 & 23 \\ 23 & 4 \end{bmatrix} = \begin{bmatrix} 23 & 4 \end{bmatrix}.$$

This matches the result from direct comparison. ✓

11.5 MATLAB Implementation

MATLAB provides two commands for pole placement, as shown in Listing 11.1.

```
A = [0 1; -2 -3];
B = [0; 1];

% Desired poles
p = [-3.5+3.57j, -3.5-3.57j];

% Method 1: place() - robust, works for repeated poles (approximately)
K = place(A, B, p);
fprintf('K (place) = [%.4f  %.4f]\n', K);

% Method 2: acker() - Ackermann's formula, exact for SISO
K2 = acker(A, B, p);
fprintf('K (acker) = [%.4f  %.4f]\n', K2);

% Verify closed-loop poles
A_cl = A - B*K;
fprintf('Closed-loop poles: '); disp(eig(A_cl)');
```

Listing 11.1: Pole placement in MATLAB

place() vs acker()

- `acker()` implements Ackermann's formula exactly. It works only for single-input systems and can be numerically sensitive for high-order systems.
- `place()` uses a more robust algorithm and also works for multi-input systems. It does not allow repeated poles at the exact same location (use slightly separated poles instead, e.g. -5.0 and -5.01).
- For most coursework problems, both give the same answer.

11.5.1 Complete design and simulation in MATLAB

```
% Plant
A = [0 1; -2 -3]; B = [0; 1]; C = [1 0]; D = 0;
sys_ol = ss(A, B, C, D);

% Design: desired poles at -3.5 +/- j3.57
p = [-3.5+3.57j, -3.5-3.57j];
K = place(A, B, p);

% Closed-loop system
A_cl = A - B*K;
sys_cl = ss(A_cl, B, C, D);

% Compare unit-step responses
t = 0:0.01:8;
[y_ol, t] = step(sys_ol, t);
[y_cl, ~] = step(sys_cl, t);

figure;
subplot(2,1,1);
plot(t, ones(size(t)), 'k--', t, y_ol, 'b', t, y_cl, 'r', 'LineWidth', 1.5);
legend('Reference r = 1', 'Open-loop', 'State feedback');
title('Pole placement improves the transient but not the tracking');
```

```
ylabel('Output y');
grid on;

subplot(2,1,2);
plot(t, 1-y_ol, 'b', t, 1-y_cl, 'r', 'LineWidth', 1.5);
legend('Open-loop error', 'State feedback error');
xlabel('Time (s)');
ylabel('Tracking error e = r-y');
grid on;

% Check performance
info = stepinfo(sys_cl);
fprintf('Settling time: %.2f s\n', info.SettlingTime);
fprintf('Overshoot: %.1f%%\n', info.Overshoot);
fprintf('Open-loop final value: %.4f\n', dcgain(sys_ol));
fprintf('State-feedback final value: %.4f\n', dcgain(sys_cl));
```

Listing 11.2: Full state feedback design and simulation

Figure 11.2 shows the responses. The pole-placement design gives a faster transient, but the tracking error does not go to zero — motivating the integral action introduced later in this chapter.

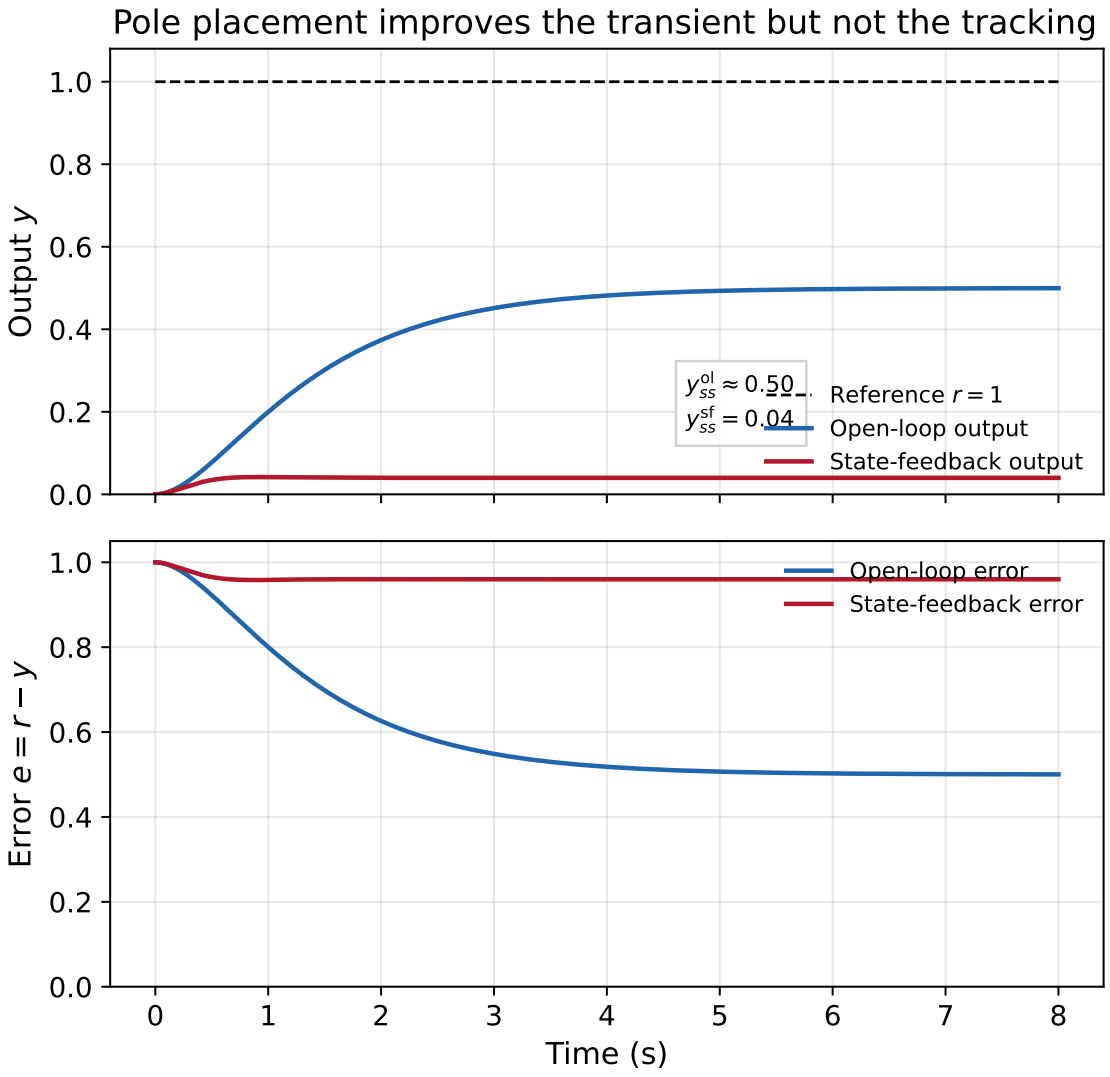


Figure 11.2: Unit-step response of the open-loop plant and the pole-placed state-feedback system from Listing 11.2. State feedback improves the transient response, but the output still settles far below the reference because there is no integral action.

11.6 The Tracking Problem

State feedback gives us the desired transient behaviour, but if we apply a step reference $r(t) = 1$, the output generally does *not* settle at $y = 1$.

The reason: $u = -Kx + r$ has no mechanism to accumulate past errors. If the output is below the reference, nothing pushes it up over time. PID solves this with the integral term; state feedback needs the same idea.

Example 11.5: State feedback without integral action

Consider $A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$ with $K = \begin{bmatrix} 23 & 4 \end{bmatrix}$ (poles at $-3.5 \pm j3.57$).

For a unit step reference $r = 1$, the closed-loop system is

$$\dot{x} = (A - BK)x + Br, \qquad y = Cx.$$

So the transfer function from r to y is

$$G_{ry}(s) = C\big(sI - (A - BK)\big)^{-1}B = \frac{1}{s^2 + 7s + 25}.$$

Using the final value theorem with $R(s) = 1/s$:

$$y_{ss} = \lim_{s\to 0} sY(s) = \lim_{s\to 0} s\,G_{ry}(s)\frac{1}{s} = \lim_{s\to 0} \frac{1}{s^2 + 7s + 25} = \frac{1}{25} = 0.04.$$

Equivalently, at steady state

$$0 = (A - BK)x_{ss} + B, \qquad y_{ss} = Cx_{ss} = -C(A - BK)^{-1}B = 0.04.$$

The output settles at 0.04 instead of 1. The system has a massive steady-state error of 96%.

State feedback alone is a *regulator* — it drives the state to zero, not to a nonzero reference.

11.6.1 Why not just scale the reference?

A tempting fix is to multiply the reference by a gain $\bar{N}$ so that $u = -Kx + \bar{N}r$, choosing $\bar{N}$ to make $y_{ss} = 1$. This requires exact knowledge of A, B, C. Any modelling error or disturbance produces a permanent steady-state offset. The robust solution is **integral action**.

11.7 Integral Action in State Feedback

11.7.1 The idea

To achieve zero steady-state error, integrate the tracking error and use that integral as an additional feedback signal. The integral accumulates until the error reaches zero — then it holds the control signal at the right value.

Define the tracking error:

$$e(t) = r(t) - y(t) = r(t) - Cx(t)$$

and introduce a new state variable — the integral of this error:

$$x_I(t) = \int_0^t e(\tau)\, d\tau, \qquad \dot{x}_I(t) = e(t) = r(t) - Cx(t).$$

The control law becomes:

Definition 11.3: State Feedback with Integral Action

$$u(t) = -Kx(t) + K_I\, x_I(t)$$

where $K \in \mathbb{R}^{1\times n}$ is the state feedback gain and $K_I \in \mathbb{R}$ is the integral gain.

At steady state, $\dot{x}_I = 0$ forces $e = r - y = 0$, so $y = r$ regardless of modelling errors or constant disturbances.

11.7.2 The augmented system

To design the gains K and K_I systematically, we combine the original states x with the new integral state x_I into an **augmented state vector**:

$$\bar{x}(t) = \begin{bmatrix} x(t) \\ x_I(t) \end{bmatrix}.$$

The augmented system dynamics are:

$$\underbrace{\begin{bmatrix} \dot{x} \\ \dot{x}_I \end{bmatrix}}_{\dot{\bar{x}}} = \underbrace{\begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}}_{\bar{A}} \underbrace{\begin{bmatrix} x \\ x_I \end{bmatrix}}_{\bar{x}} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{\bar{B}} u + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{\bar{B}_r} r.$$

The control law $u = -Kx + K_I x_I$ can be written as:

$$u = -\underbrace{\begin{bmatrix} K & -K_I \end{bmatrix}}_{\bar{K}} \bar{x}.$$

This is standard state feedback on the Augmented system. The closed-loop matrix is $\bar{A} - \bar{B}\bar{K}$, and we can place its $n+1$ eigenvalues at any desired locations — provided $(\bar{A}, \bar{B})$ is controllable.

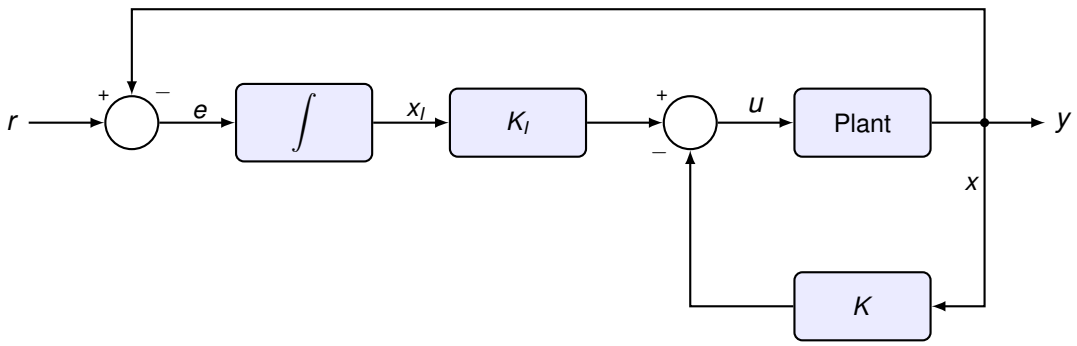


Figure 11.3: State feedback with integral action. The integrator accumulates the tracking error $e = r - y$ and drives it to zero at steady state.

Remark 11.3

The augmented system has $n+1$ states (the original n plant states plus the integral state x_I). We need to place $n+1$ poles, so we have $n+1$ gains to choose: $K = [k_1 \ \cdots \ k_n]$ and K_I.

11.7.3 Controllability of the augmented system

For the design to work, $(\bar{A}, \bar{B})$ must be controllable. A sufficient condition is that (A, B) is controllable (which we already check) and A has no eigenvalue at $s = 0$. If A does have a zero eigenvalue (e.g. a free integrator in the plant), the augmented system may still be controllable — check with MATLAB.

11.7.4 Design procedure

1. **Form the augmented system** $(\bar{A}, \bar{B})$ by appending the integral state.
2. **Check controllability** of $(\bar{A}, \bar{B})$.
3. **Choose** $n+1$ **desired poles.** The extra pole (from the integrator) is typically placed on the real axis, further left than the dominant poles, so it does not slow down the response.
4. **Compute** $\bar{K} = \begin{bmatrix} K & -K_I \end{bmatrix}$ using `place()` or `acker()` on the augmented system.
5. **Extract** K and K_I from $\bar{K}$.

11.8 Worked Example: Mass–Spring–Damper with Integral Action

Example 11.6: State feedback with integral action

Consider our familiar system:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

Step 1: Augmented system.

$$\bar{A} = \begin{bmatrix} 0 & 1 & 0 \\ -2 & -3 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

Step 2: Controllability.

$$\bar{\mathcal{C}} = \begin{bmatrix} \bar{B} & \bar{A}\bar{B} & \bar{A}^2\bar{B} \end{bmatrix} = \begin{bmatrix} 0 & 1 & -3 \\ 1 & -3 & 7 \\ 0 & 0 & -1 \end{bmatrix}.$$

$\det(\bar{\mathcal{C}}) = 1 \neq 0$. Therefore the augmented system is Controllable. ✓

Step 3: Desired poles. We want the original two poles to be the roots of

$$s^2 + 7s + 25 = 0,$$

which are approximately $-3.5 \pm j3.57$, and we place the integral pole at -8 (fast, so it doesn't affect the transient):

$$p = [-3.5 + j3.5707, \quad -3.5 - j3.5707, \quad -8].$$

Step 4: Compute gains. The desired characteristic polynomial is

$$(s^2 + 7s + 25)(s + 8) = s^3 + 15s^2 + 81s + 200.$$

With $\bar{K} = [k_1 \; k_2 \; -K_I]$, the closed-loop characteristic polynomial of $\bar{A} - \bar{B}\bar{K}$ is

$$\det(sI - \bar{A} + \bar{B}\bar{K}) = s^3 + (3 + k_2)s^2 + (2 + k_1)s + K_I.$$

Rather than expanding by hand for a 3×3 system, we use MATLAB (Listing 11.3):

```
A = [0 1; -2 -3]; B = [0; 1]; C = [1 0];

% Augmented system
Abar = [A zeros(2,1); -C 0];
Bbar = [B; 0];

  % Desired poles: original pair from s^2 + 7s + 25, plus integral pole
  wd = sqrt(25 - 3.5^2);
  p = [-3.5+1j*wd, -3.5-1j*wd, -8];

% Compute augmented gain
Kbar = place(Abar, Bbar, p);

% Extract gains
K  = Kbar(1:2);        % state feedback gains
KI = -Kbar(3);         % integral gain (note the sign)

fprintf('K  = [%.4f  %.4f]\n', K);
fprintf('KI = %.4f\n', KI);

```

Listing 11.3: Integral action design via augmented system.

This gives

$$K = \begin{bmatrix} 79 & 12 \end{bmatrix}, \qquad K_I = 200.$$

If you instead use the rounded pole pair $-3.5 \pm j3.57$ directly, MATLAB returns values extremely close to these.

Step 5: Simulate (Listing 11.4).

```
  % Closed-loop system with integral action
  Acl = Abar - Bbar*Kbar;
  Br  = [0; 0; 1];           % reference enters through integral state
  sys_int = ss(Acl, Br, [C 0], 0);

  % Compare with state feedback without integral action
    wd = sqrt(25 - 3.5^2);
    Ksf = place(A, B, [-3.5+1j*wd, -3.5-1j*wd]);
  sys_sf = ss(A - B*Ksf, B, C, 0);
```

```
10
11  t = 0:0.01:8;
12  [y_sf, t] = step(sys_sf, t);
13  [y_int, ~] = step(sys_int, t);
14
15  figure;
16  subplot(2,1,1);
17  plot(t, ones(size(t)), 'k--', t, y_sf, 'r', t, y_int, 'g', 'LineWidth', 1.5)
      ;
18  legend('Reference r = 1', 'Without integral action', 'With integral action')
      ;
19  title('Integral action removes the steady-state tracking error');
20  ylabel('Output y');
21  grid on;
22
23  subplot(2,1,2);
24  plot(t, 1-y_sf, 'r', t, 1-y_int, 'g', 'LineWidth', 1.5);
25  legend('Error without K_I', 'Error with K_I');
26  xlabel('Time (s)');
27  ylabel('Tracking error e = r-y');
28  grid on;
29
```

Listing 11.4: Simulating state feedback with and without integral action.

Figure 11.4 shows the responses produced by this simulation. The upper plot makes the practical benefit immediate: with integral action, the output converges to the unit-step reference instead of stalling at a large offset. The lower plot confirms the same point in error coordinates: $e(t)$ goes to zero only when the integral state is included.

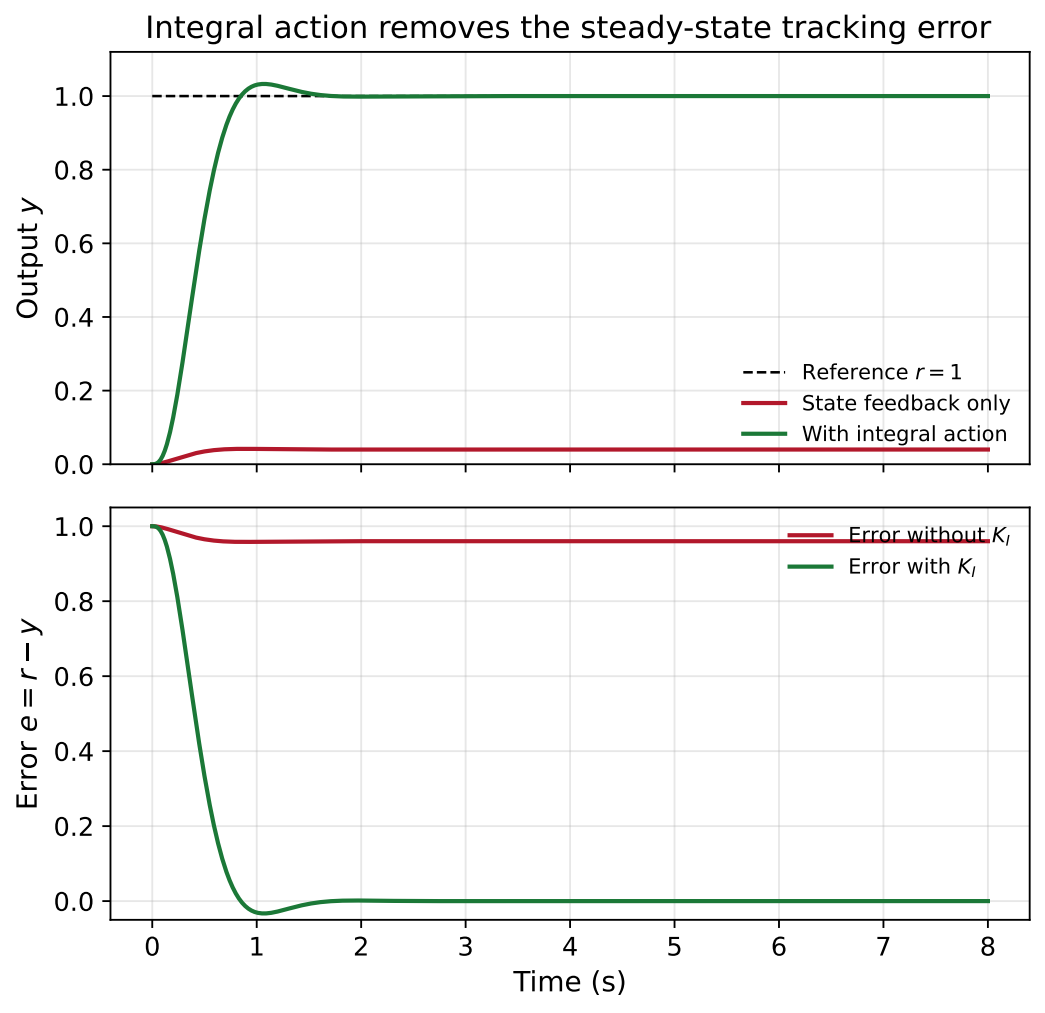


Figure 11.4: Comparison of state feedback with and without integral action for the same mass–spring–damper system. The integral-action design tracks the unit-step reference and drives the tracking error to zero.

The output now tracks the step reference with **zero steady-state error** for this nominal design, and integral action also improves robustness to constant disturbances and modest modelling errors provided the closed loop remains stable.

11.9 Why Integral Action Works: An Intuitive View

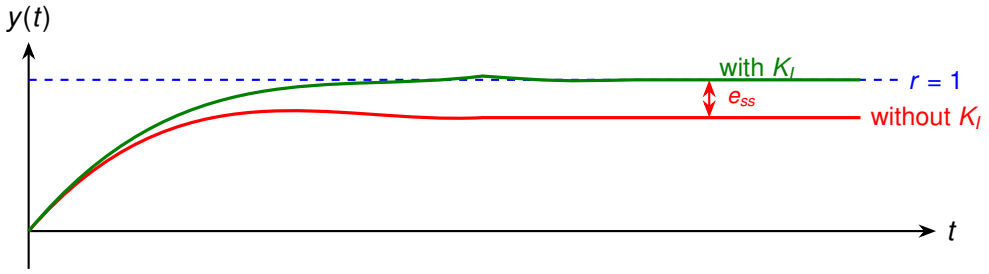


Figure 11.5: Comparison of state feedback with and without integral action. Without K_I, the output settles below the reference. With K_I, the integrator drives the error to zero.

Think of the integral state x_I as a "memory of dissatisfaction". Every instant that $y < r$, the error $e > 0$ adds to x_I, which increases $K_I x_I$ in the control signal, pushing the output harder towards the reference. The process stops only when $e = 0$ and x_I holds steady. This is exactly what the "I" term does in PID — here we have embedded it into the state feedback framework.

Remark 11.4

At steady state: $\dot{x}_I = 0 \Rightarrow e = r - y = 0 \Rightarrow y = r$. This holds regardless of the actual plant parameters, as long as the closed-loop system is stable. This is what makes integral action **robust**.

11.10 A Complete Design Example: Rotary Inverted Pendulum Tracking

This example uses the linearised Quanser rotary inverted pendulum model and the balance-design specifications given in the Quanser Controls Board lab manual [16]. We assume all four states are measurable, so no observer is needed at this stage.

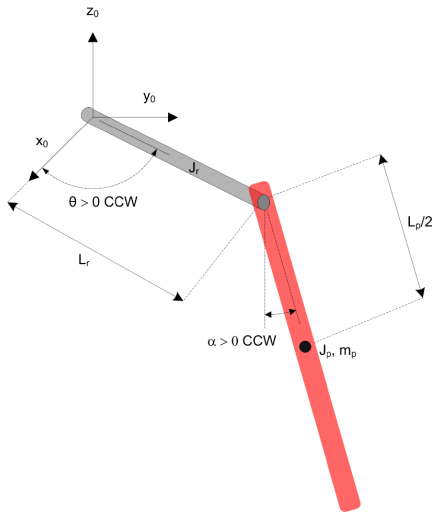


Figure 11.6: Rotary inverted pendulum geometry used in the Quanser lab manual. Adapted from the Quanser Controls Board lab manual [16].

Example 11.7: Rotary inverted pendulum tracking with integral action

Problem formulation.

Using the model in Figure 11.6, let the state be

$$x = \begin{bmatrix} \theta & \tilde{\alpha} & \dot{\theta} & \dot{\tilde{\alpha}} \end{bmatrix}^\top,$$

where θ is the rotary arm angle. In the Quanser geometry, the physical pendulum angle α is measured from the hanging-down position, so the upright equilibrium corresponds to $\alpha = \pi$. The linear balance model is therefore written in terms of the small deviation

$$\tilde{\alpha} = \alpha - \pi.$$

The control objective is to move the arm to a constant command $\theta_r = 20^\circ$ while keeping the pendulum close to the upright equilibrium, i.e. keeping $\tilde{\alpha}$ small. To make the tracking problem realistic, we also include an unknown constant actuator bias $d = 0.30$ V, representing effects such as amplifier offset or unmodelled friction.

The controller must:

- stabilise the upright pendulum,
- track the arm-angle command θ_r,
- remove the steady-state error caused by the constant bias.

Step 1: State-space model from the Quanser manual.

The representative linearised model given in the Quanser lab manual is [16]

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

with

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 149.23 & -0.0104 & 0 \\ 0 & 261.53 & -0.0103 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 49.72 \\ 49.13 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Only θ and the physical pendulum angle are measured directly; in this worked example we assume $\dot{\theta}$ and $\dot{\tilde{\alpha}}$ are also available, for example from filtered differentiation of encoder signals.

The open-loop poles are approximately

$$0, \quad -0.00452, \quad -16.1748, \quad +16.1690.$$

Because one pole is in the right-half plane, this is the linearisation about the *upright* equilibrium, not the hanging-down equilibrium. That is exactly what we need for balance control.

Step 2: Check controllability.

Before placing poles, verify that the pair (A, B) is controllable (Listing 11.5):

```
A = [0 0 1 0;
     0 0 0 1;
     0 149.23 -0.0104 0;
     0 261.53 -0.0103 0];

B = [0; 0; 49.72; 49.13];

rank(ctrb(A,B))   % returns 4

```

Listing 11.5: RIP controllability check.

So full-state feedback can assign the four closed-loop poles arbitrarily.

Step 3: First design a balance controller.

We start from the Quanser manual specifications for the balance problem [16]:

$$\zeta = 0.7, \qquad \omega_n = 4 \text{ rad/s}.$$

This gives the dominant pair

$$p_{1,2} = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2} = -2.8 \pm j2.8566.$$

As in the manual, place the remaining poles at

$$p_3 = -30, \qquad p_4 = -40.$$

In MATLAB (Listing 11.6):

```
p = [-2.8+2.8566j, -2.8-2.8566j, -30, -40];
K = place(A, B, p)

```

Listing 11.6: RIP balance controller pole placement.

which gives

$$K = \begin{bmatrix} -3.3853 & 41.4787 & -1.3825 & 2.9377 \end{bmatrix}.$$

If the model were exact and no disturbance were present, a reference prefilter

$$N_r = -\frac{1}{C_r(A-BK)^{-1}B}, \qquad C_r = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix},$$

makes the arm angle track a constant command. For this model,

$$N_r = -3.3853.$$

Step 4: Why the balance controller is not enough for robust tracking.

With a constant input bias d, the practical control law becomes

$$u = -Kx + N_r r, \qquad \dot{x} = (A-BK)x + BN_r r + Bd.$$

At steady state,

$$0 = (A-BK)x_{ss} + B(N_r r + d),$$

so the tracked arm angle is

$$\theta_{ss} = -C_r(A - BK)^{-1}B(N_r r + d).$$

Now substitute the prefilter chosen in Step 3:

$$N_r = -\frac{1}{C_r(A - BK)^{-1}B}.$$

Then

$$\theta_{ss} = -C_r(A - BK)^{-1}B\,N_r\,r - C_r(A - BK)^{-1}B\,d = r - C_r(A - BK)^{-1}B\,d.$$

Hence the steady-state tracking error is

$$e_{ss} = r - \theta_{ss} = C_r(A - BK)^{-1}B\,d.$$

This shows the key point: the prefilter N_r cancels the nominal steady-state gain only when $d = 0$. A constant bias adds an extra term that is not cancelled, so the steady-state error is generally nonzero. Therefore, even though state feedback stabilises the pendulum, it does *not* guarantee zero steady-state tracking error in the presence of constant disturbances.

Step 5: Add integral action on the arm-angle error.

Define the integral state

$$x_I = \int (r - \theta)\,dt, \qquad \dot{x}_I = r - C_r x.$$

The augmented model is

$$\dot{\bar{x}} = \bar{A}\bar{x} + \bar{B}u + B_r r, \qquad \bar{x} = \begin{bmatrix} x \\ x_I \end{bmatrix},$$

with

$$\bar{A} = \begin{bmatrix} A & 0 \\ -C_r & 0 \end{bmatrix}, \qquad \bar{B} = \begin{bmatrix} B \\ 0 \end{bmatrix}, \qquad B_r = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

We keep the four balance poles and add one extra pole for the integral state:

$$p_5 = -6.$$

This pole is fast enough to remove the bias promptly, but not so fast that it makes the control unnecessarily aggressive.

Step 6: Compute the augmented gain.

The augmented pair $(\bar{A}, \bar{B})$ must also be controllable (Listing 11.7):

```
Cr   = [1 0 0 0];
Abar = [A zeros(4,1);
       -Cr 0];
Bbar = [B; 0];

rank(ctrb(Abar,Bbar))   % returns 5

pbar = [-2.8+2.8566j, -2.8-2.8566j, -30, -40, -6];
Kbar = place(Abar, Bbar, pbar)

```

Listing 11.7: RIP augmented system with integral action.

This gives

$$\bar{K} = \begin{bmatrix} -11.6792 & 59.1050 & -3.2617 & 4.9616 & 20.3117 \end{bmatrix}.$$

Using the chapter convention $\bar{K} = \begin{bmatrix} K & -K_I \end{bmatrix}$, this means

$$K_I = -20.3117.$$

The negative sign is not a mistake. For this particular linearised model, a positive arm-angle command requires a negative steady-state motor voltage, so both the reference prefilter and the integral gain are negative in the convention $u = -Kx + K_I x_I$.

Step 7: Simulate the tracking problem.

The following MATLAB script compares:

- state feedback with reference prefilter only, and
- state feedback with integral action,

for the same 20° command and the same constant bias $d = 0.30$ V (Listing 11.8).

```
A = [0 0 1 0;
     0 0 0 1;
     0 149.23 -0.0104 0;
     0 261.53 -0.0103 0];
B = [0; 0; 49.72; 49.13];
Cr = [1 0 0 0];

% State feedback without integral action
p = [-2.8+2.8566j, -2.8-2.8566j, -30, -40];
K = place(A, B, p);
Nr = -1/(Cr * ((A - B*K)\B));

Acl_sf = A - B*K;
Bsf = [B*Nr  B];            % inputs [r; d]
sys_sf = ss(Acl_sf, Bsf, eye(4), zeros(4,2));

% State feedback with integral action
Abar = [A zeros(4,1);
       -Cr 0];
Bbar = [B; 0];
Br = [0; 0; 0; 0; 1];
Bd = [B; 0];

pbar = [-2.8+2.8566j, -2.8-2.8566j, -30, -40, -6];
Kbar = place(Abar, Bbar, pbar);

Acl_int = Abar - Bbar*Kbar;
sys_int = ss(Acl_int, [Br Bd], eye(5), zeros(5,2));

t = 0:0.005:8;
r = deg2rad(20) * ones(size(t));
d = 0.30 * ones(size(t));
U = [r(:) d(:)];

xsf = lsim(sys_sf, U, t);
zint = lsim(sys_int, U, t);

theta_sf = rad2deg(xsf(:,1));
```

```
alpha_tilde_sf = rad2deg(xsf(:,2));
theta_int = rad2deg(zint(:,1));
alpha_tilde_int = rad2deg(zint(:,2));

u_sf = -xsf*K.' + Nr*deg2rad(20);
u_int = -zint*Kbar.';

figure;
subplot(2,2,1);
plot(t, 20*ones(size(t)), 'k--', t, theta_sf, 'r', t, theta_int, 'g', '
    LineWidth', 1.5);
legend('Reference', 'State feedback + N_r', 'With integral action');
ylabel('\theta (deg)'); grid on;

subplot(2,2,2);
plot(t, 20-theta_sf, 'r', t, 20-theta_int, 'g', 'LineWidth', 1.5);
ylabel('Tracking error (deg)'); grid on;

subplot(2,2,3);
plot(t, alpha_tilde_sf, 'r', t, alpha_tilde_int, 'g', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('\\alpha~from upright (deg)'); grid on;

subplot(2,2,4);
plot(t, u_sf, 'r', t, u_int, 'g', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('u (V)'); grid on;

```

Listing 11.8: RIP tracking simulation: prefilter vs integral action.

Figure 11.7 shows the result. The red curves correspond to the balance controller with a reference prefilter only; the green curves include the integral state. Without the integrator, the arm settles at about 14.9° instead of 20°, so the steady-state tracking error is about 5.1°. With integral action, the arm reaches the command and the error goes to zero, while the pendulum deviation from upright remains small.

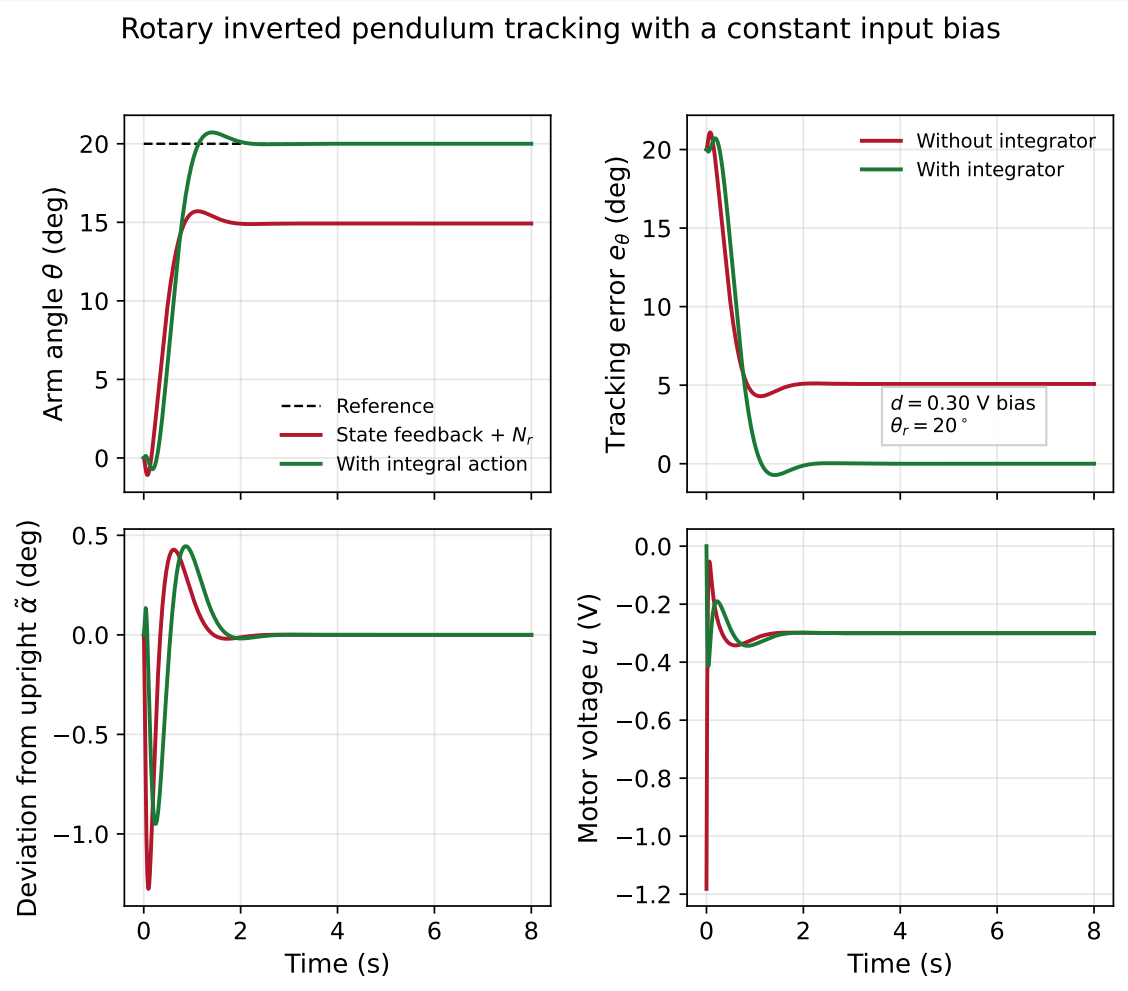


Figure 11.7: Rotary inverted pendulum tracking with a constant actuator bias of 0.30 V. Integral action removes the arm-angle offset while keeping the pendulum deviation from upright small.

From the simulation:

$$\theta_{ss,\text{no int}} \approx 14.9^\circ, \qquad \theta_{ss,\text{int}} \approx 20.0^\circ,$$

and the integral-action design keeps

$$\max|\tilde{\alpha}(t)| \approx 0.95^\circ, \qquad \max|u(t)| \approx 0.42 \text{ V},$$

which is comfortably within the additional Quanser implementation limits of pendulum deflection and control effort quoted in the same lab manual [16].

Conclusion.

For the rotary inverted pendulum, ordinary state feedback is excellent for *balancing*, but integral action is what makes the design a reliable *tracking controller* in the presence of constant disturbances and modelling errors.

11.11 State Feedback vs PID: When to Use Which?

State Feedback (with Integral Action) vs PID

PID	State Feedback + Integral
Feeds back the output error only	Feeds back all states + integral of error
Three gains to tune	$n+1$ gains (systematic)
Limited pole placement capability	Places *all* $n+1$ poles exactly
Does not need a model	Requires a state-space model
Built-in integral action	Integral action added explicitly
SISO only	Extends naturally to MIMO
Industry standard, widely understood	Foundation for modern control

Remark 11.5

State feedback requires measurement of all states $x(t)$. In practice, we often cannot measure all states directly. Chapter 12 solves this problem by introducing an *observer* — a system that estimates $x(t)$ from the measured output $y(t)$.

11.12 Summary

Chapter Summary

- State feedback uses $u = -Kx$ to place all closed-loop poles at desired locations. The closed-loop system matrix is $A_{cl} = A - BK$.
- Pole placement requires **controllability** of (A, B).
- K can be found by direct comparison of characteristic polynomials, via the CCF shortcut, or by Ackermann's formula.
- MATLAB: `place(A,B,p)` or `acker(A,B,p)`.
- State feedback alone does not guarantee zero steady-state tracking error.
- **Integral action** solves this: augment the state with $x_I = \int (r - y)\, dt$, then design $\bar{K} = [K \quad -K_I]$ on the augmented system. The result is robust zero steady-state error.
- State feedback requires full state measurement — Chapter 12 introduces observers to estimate unmeasured states.

11.13 End-of-Chapter Problems

1. **(Direct comparison — 2nd order)** A system has
$$A = \begin{bmatrix} 0 & 1 \\ -3 & -4 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$
   (a) Find the open-loop poles.
   (b) Design K to place the closed-loop poles at $s = -5 \pm j5$.
   (c) Verify by computing $\det(sI - A + BK)$.

2. **(Specifications to poles)** For the system in Problem 1 with $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$:
   (a) Find the desired pole locations for $t_s \leq 0.8$ s and $M_p \leq 15\%$.
   (b) Design K for these specifications.
   (c) Apply a unit step reference using $u = -Kx + r$ (no integral action). What is y_{ss}?

3. **(Ackermann's formula)** For the system
$$A = \begin{bmatrix} -1 & 2 \\ 0 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}:$$
   (a) Verify controllability.
   (b) Use Ackermann's formula to find K that places the poles at $s = -4, -6$.
   (c) Verify using direct comparison.

4. **(CCF pole placement)** A third-order system in CCF has
$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -5 & -4 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$
   (a) Find the open-loop poles.
   (b) Place the poles at $s = -3, -3, -3$ using the CCF shortcut (subtract coefficients).
   (c) Verify with MATLAB.

5. **(Integral action — 2nd order)** For the system in Problem 1 with $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$:
   (a) Form the augmented system $(\bar{A}, \bar{B})$ by adding the integral state.
   (b) Verify that $(\bar{A}, \bar{B})$ is controllable.
   (c) Place the three closed-loop poles at $s = -5 \pm j5$ and $s = -12$. Find K and K_I.
   (d) Simulate the step response in MATLAB and verify $y_{ss} = 1$.

6. **(Uncontrollable system)** Consider
$$A = \begin{bmatrix} -2 & 0 \\ 0 & -5 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$
   (a) Show that the system is not controllable.
   (b) Try to place both poles at $s = -10$. What happens?
   (c) Which pole can be moved by state feedback? Which cannot?

7. **(Robustness of integral action)** For the system $A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$:

(a) Design state feedback with integral action (place poles at $-3.5 \pm j3.57$ and -8).

(b) Simulate with the nominal plant. Verify $y_{ss} = 1$.

(c) Now change the plant to $A_{\text{true}} = \begin{bmatrix} 0 & 1 \\ -2.5 & -3.5 \end{bmatrix}$ (a 25% parameter error) but keep the same controller. Simulate again. Does y_{ss} still equal 1?

(d) Repeat part (c) but without integral action (use state feedback with a fixed gain only). Compare.

8. (MATLAB — complete design) A system has

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -10 & -17 & -8 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}.$$

(a) Check controllability of the original and augmented systems.

(b) Design state feedback with integral action. Place the original poles at $s = -6 \pm j4$ and $s = -8$, and the integral pole at $s = -15$.

(c) Simulate the step response and verify zero steady-state error.

(d) Add a step disturbance of magnitude 0.5 at $t = 5$ s (entering at the plant input). Show that the integral action rejects it.

Chapter 12

Observer Design and Observer-Based Control

"You cannot control what you cannot see — but you can build a system that sees for you."

State feedback requires measuring the full state vector $x(t)$, but in practice this is rarely possible. A DC motor controller may measure shaft position but not armature current; a robot may measure joint angles but not velocities.

The solution is an **observer** (state estimator): a dynamic system that runs in parallel with the plant and produces an estimate $\hat{x}(t)$ from the measured output $y(t)$ and known input $u(t)$.

Learning Objectives

After completing this chapter, you should be able to:

- Explain why observers are needed and state the observability condition for their existence.
- Derive the Luenberger observer equation and its error dynamics.
- Compute the observer gain L by pole placement using the duality with state feedback.
- Apply the separation principle to combine observer and state feedback independently.
- Design an Observer-based controller with integral action for zero steady-state error.
- Implement and simulate the complete design in MATLAB and Simulink.

12.1 The Observer Problem

Consider a plant in state-space form:

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad y(t) = Cx(t). \tag{12.1}$$

We know the matrices A, B, C (from our model) and we can measure $u(t)$ and $y(t)$. We want to reconstruct $x(t)$.

12.1.1 A naive approach: open-loop simulation

The simplest idea is to build a copy of the plant and run it in parallel:

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t). \tag{12.2}$$

If we knew the exact initial condition $\hat{x}(0) = x(0)$ and the model were perfect, then $\hat{x}(t) = x(t)$ for all time.

We never know the exact initial condition, and models are never perfect. Define the estimation error:

$$e(t) = x(t) - \hat{x}(t).$$

Subtracting (12.2) from (12.1):

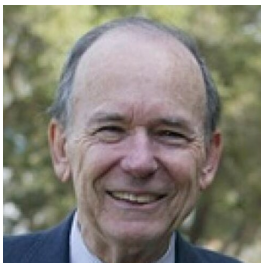
$$\dot{e}(t) = Ae(t).$$

The error follows the plant's own dynamics. If the plant is stable, the error decays — but only as fast as the plant's slowest pole. If the plant is unstable, the error grows without bound.

Remark 12.1

The open-loop estimator works only if the plant is stable and we are patient. In control engineering, we need the estimate to converge *fast* — much faster than the plant itself. This is what the Luenberger observer achieves.

12.2 The Luenberger Observer



David G. Luenberger. Photo: Stanford Engineering.

The observer is named after David G. Luenberger, whose early work helped establish the modern state-observer framework. In his 1964 paper, he showed how the full state vector of a linear system can be reconstructed from measured inputs and outputs; his 1971 survey then organised the main observer ideas in a form that became standard in control engineering [11, 12].

The key insight is to use the measured output $y(t)$ to *correct* the estimate. At each instant, we compare the actual output $y(t)$ with the predicted output $\hat{y}(t) = C\hat{x}(t)$. The difference $y - \hat{y}$ tells us how wrong our estimate is. We feed this correction back through a gain matrix L:

Definition 12.1: Luenberger Observer

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L\big(y(t) - C\hat{x}(t)\big) \tag{12.3}$$

where $L \in \mathbb{R}^{n\times 1}$ is the *observer gain vector*.

The observer equation has three terms:

- $A\hat{x} + Bu$: the model prediction (same as the open-loop estimator),
- $L(y - C\hat{x})$: the correction term, which pushes $\hat{x}$ towards x whenever the output prediction is wrong.

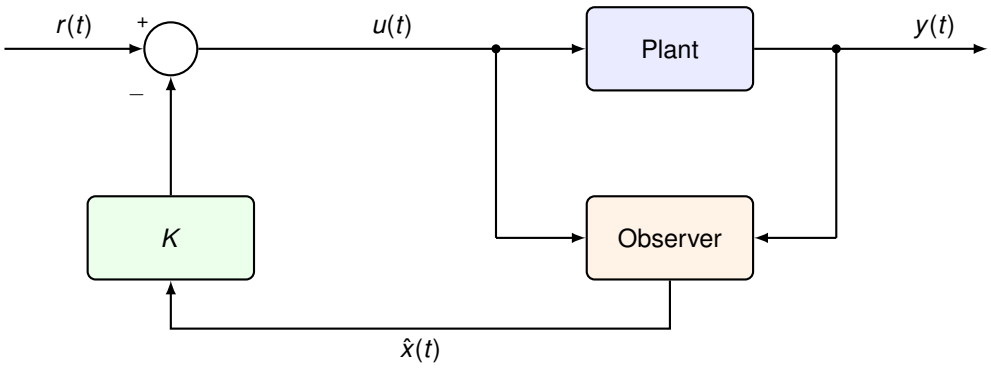


Figure 12.1: Observer-based state feedback controller. The observer estimates $\hat{x}(t)$ from $u(t)$ and $y(t)$. The feedback gain K uses $\hat{x}$ instead of x.

12.2.1 Error dynamics

The estimation error $e(t) = x(t) - \hat{x}(t)$ satisfies:

$$\begin{aligned}\dot{e}(t) &= \dot{x}(t) - \dot{\hat{x}}(t) \\ &= \big[Ax + Bu\big] - \big[A\hat{x} + Bu + L(Cx - C\hat{x})\big] \\ &= (A - LC)\, e(t).\end{aligned}$$

The error dynamics are governed by the eigenvalues of $A - LC$, which determine how fast the estimation error decays.

Result 12.1: Observer Error Convergence

If the eigenvalues of $A - LC$ all have negative real parts, then $e(t) \to 0$ as $t \to \infty$, regardless of the initial error $e(0) = x(0) - \hat{x}(0)$.

12.2.2 Choosing the observer gain L

Choose L so that $A - LC$ has eigenvalues at desired locations — the same type of pole placement problem we solved for K in Chapter 11.

Recall from Chapter 10 that a system is **observable** if and only if the observability matrix

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

has full rank. The following result is the observer counterpart of the controllability condition for pole placement:

Result 12.2: Observer Pole Placement

The eigenvalues of $A - LC$ can be placed at arbitrary locations if and only if the pair (A, C) is **observable**.

Remark 12.2

Compare with state feedback: K places eigenvalues of $A - BK$ (requires controllability of (A, B)), while L places eigenvalues of $A - LC$ (requires observability of (A, C)). The duality between controllability and observability (Chapter 10) is at work here.

12.2.3 How fast should the observer be?

A practical rule of thumb:

Observer pole placement guideline

Place the observer poles 3–5 times faster (further left in the s-plane) than the controller poles.

The observer must converge before the controller can act effectively. Too slow, and the controller acts on a poor estimate. Too fast, and the large gains in L amplify sensor noise.

12.3 Computing the Observer Gain

12.3.1 Direct comparison method

1. Choose the desired observer poles $\lambda_1, \lambda_2, \ldots, \lambda_n$.
2. Form the desired characteristic polynomial: $\alpha_d(s) = (s - \lambda_1)(s - \lambda_2)\cdots(s - \lambda_n)$.
3. Compute $\det(sI - A + LC)$ symbolically in terms of the unknown entries of L.
4. Match coefficients with $\alpha_d(s)$ and solve for L.

12.3.2 Using duality

By duality (Chapter 10), (A, C) observable means $(A^\top, C^\top)$ is controllable, so we can compute L via:

$$L = \texttt{place}(A^\top, C^\top, p_{\text{obs}})^\top$$

where p_{obs} is the vector of desired observer poles.

12.3.3 MATLAB implementation

Listing 12.1 shows how to compute the observer gain using the duality trick.

```
% Plant matrices
A = [0 1; -2 -3];
B = [0; 1];
C = [1 0];

% Controller poles (desired closed-loop)
pc = [-3.5+3.57j, -3.5-3.57j];
K = place(A, B, pc);

% Observer poles (3-5x faster)
po = [-15, -16];
L = place(A', C', po)';

% Verify eigenvalues
eig(A - L*C)
```

Listing 12.1: Computing the observer gain in MATLAB

12.4 Worked Example: Observer Design for a Second-Order System

Example 12.1: Luenberger observer for a mass–spring–damper

Consider the mass–spring–damper system from Chapter 11:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

Only the position $y = x_1$ is measured. The velocity x_2 is not directly available.

Step 1: Check observability.

$$\mathcal{O} = \begin{bmatrix} C \\ CA \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \operatorname{rank}(\mathcal{O}) = 2 = n. \quad \checkmark$$

Step 2: Choose observer poles. Suppose the controller poles are at $s = -3.5 \pm j3.57$ (from Chapter 11). We choose observer poles $4\times$ faster:

$$\lambda_{1,2} = -15, \; -16.$$

Step 3: Compute L. The characteristic polynomial of $A - LC$ is:

$$\det(sI - A + LC) = s^2 + (3 + l_1)s + (2 + l_2 + 3l_1).$$

The desired polynomial is:

$$(s + 15)(s + 16) = s^2 + 31s + 240.$$

Matching coefficients:

$$3 + l_1 = 31 \implies l_1 = 28, \qquad 2 + l_2 + 3 \times 28 = 240 \implies l_2 = 154.$$

Therefore:

$$L = \begin{bmatrix} 28 \\ 154 \end{bmatrix}.$$

Step 4: Verify.

$$A - LC = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} - \begin{bmatrix} 28 \\ 154 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} -28 & 1 \\ -156 & -3 \end{bmatrix}.$$

Eigenvalues: -15 and -16. ✓

12.5 The Separation Principle

Can we simply replace x with $\hat{x}$ in the feedback law $u = -Kx + r$?

$$u(t) = -K\hat{x}(t) + r(t) \tag{12.4}$$

Yes, and this is guaranteed by the **Separation principle**:

Theorem 12.1: The Separation Principle

When the observer-based control law (12.4) is applied to the plant (12.1), the closed-loop characteristic polynomial is:

$$\det(sI - A + BK) \cdot \det(sI - A + LC).$$

The closed-loop poles are the union of the controller poles and the observer poles. The two designs do not interact.

Proof sketch. Define $e = x - \hat{x}$. The closed-loop system in the coordinates (x, e) is:

$$\begin{bmatrix} \dot{x} \\ \dot{e} \end{bmatrix} = \underbrace{\begin{bmatrix} A - BK & BK \\ 0 & A - LC \end{bmatrix}}_{\text{block upper-triangular}} \begin{bmatrix} x \\ e \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} r.$$

Because the matrix is block upper-triangular, its eigenvalues are those of $A - BK$ together with those of $A - LC$. The controller poles and observer poles are completely decoupled.

Remark 12.3

The separation principle means we can design the controller and observer independently, and the combined system will have exactly the poles we intended.

Interpretation 12.1: What does the separation principle mean physically?

The observer error $e(t)$ converges to zero with its own dynamics $(A - LC)$. Once $\hat{x} \approx x$, the controller behaves as if it had full state measurement. The transient caused by the initial estimation error affects performance briefly, but the steady-state behaviour is identical to the full-state-feedback case.

12.6 Observer-Based Controller: Complete Design Procedure

The complete design procedure is:

1. **Model**: obtain the state-space model (A, B, C).
2. **Check controllability**: verify $\operatorname{rank}(\mathcal{C}) = n$.
3. **Check observability**: verify $\operatorname{rank}(\mathcal{O}) = n$.
4. **Design state feedback**: choose desired controller poles p_c, then compute the gain
$$K = \text{place}(A, B, p_c).$$
5. **Design observer**: choose desired observer poles p_o (3–5$\times$ faster than p_c) and compute $L = \texttt{place}(A', C', p_o)'$.

6. **Combine**: implement the control law $u = -K\hat{x} + r$ with the observer running in parallel.
7. **Simulate and verify**.

Example 12.2: Complete observer-based controller for the mass–spring–damper

Consider the mass–spring–damper from Chapter 11 with the transfer function $G(s) = \dfrac{1}{s^2 + 3s + 2}$.

State-space form (controllable canonical form):

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

Specifications: settling time $t_s \leq 2\,\text{s}$, overshoot $\leq 10\%$.

From Chapter 6, these translate to $\zeta \geq 0.59$ and $\sigma \geq 2$. We choose controller poles at $s = -3 \pm j3$ (giving $\zeta = 0.71$, $\sigma = 3$).

Solution:

Controller gain:

$$K = \texttt{place}(A, B, [-3 + 3j,\ -3 - 3j]) = \begin{bmatrix} 16 & 3 \end{bmatrix}.$$

Observer gain (poles $4\times$ faster, at -12 and -13):

$$L = \texttt{place}(A', C', [-12, -13])' = \begin{bmatrix} 22 \\ 88 \end{bmatrix}.$$

MATLAB simulation: Listing 12.2 builds the composite closed-loop system and plots the step response.

```
A = [0 1; -2 -3];  B = [0; 1];  C = [1 0];  D = 0;

K  = place(A, B, [-3+3j, -3-3j]);
L  = place(A', C', [-12, -13])';

% Closed-loop system with observer
Acl = [A-B*K, B*K; zeros(2), A-L*C];
Bcl = [B; zeros(2,1)];
Ccl = [C, zeros(1,2)];

sys_cl = ss(Acl, Bcl, Ccl, 0);
step(sys_cl);
title('Observer-based state feedback');
```

Listing 12.2: Observer-based state feedback: closed-loop simulation

12.7 Observer-Based Controller with Integral Action

Without an integrator in the loop, the observer-based controller does not guarantee zero steady-state error for step references. The solution combines three ingredients:

1. **Observer**: estimates the state $\hat{x}$ from y and u.
2. **State feedback**: stabilises the system and shapes the transient response.
3. **Integral action**: eliminates steady-state error.

12.7.1 The augmented observer-based controller

Recall from Chapter 11 that integral action introduces an additional state:

$$\dot{x}_I(t) = r(t) - y(t) = r(t) - Cx(t).$$

The control law is:

$$u(t) = -K\hat{x}(t) + K_I\, x_I(t)$$

where $\hat{x}$ comes from the observer and x_I is computed directly from the measured output (no estimation needed — y is measured).

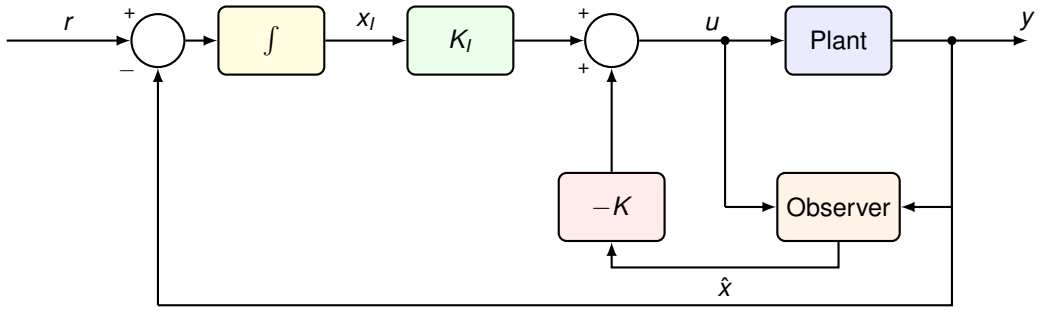


Figure 12.2: Observer-based state feedback with integral action. The integrator accumulates the tracking error $r - y$, the observer estimates $\hat{x}$, and both signals are combined to form the control input u.

12.7.2 Design procedure

1. **Design the augmented state feedback** (as in Chapter 11):
   - Form the augmented system with states $\bar{x} = [x^\top \;\; x_I]^\top$.
   - Choose $n+1$ desired controller poles.
   - Compute $\bar{K} = [K \;\; -K_I]$ using `place()` on the augmented system.
2. **Design the observer** (on the original, non-augmented system):
   - Choose n observer poles (3–5$\times$ faster).
   - Compute $L = \texttt{place}(A', C', p_o)'$.
3. **Implement**: the observer estimates $\hat{x}$; the integrator uses the measured y directly; the control law is $u = -K\hat{x} + K_I x_I$.

Remark 12.4

The integral state x_I does not need to be estimated — it is computed from the measured output $y(t)$. Therefore, the observer is designed for the original n-state system, not the augmented $(n+1)$-state system.

Example 12.3: Observer-based controller with integral action

Continuing the DC motor example with $A = \left[\begin{smallmatrix} 0 & 1 \\ -2 & -3 \end{smallmatrix}\right]$, $B = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$, $C = [1\ 0]$.

Augmented system (from Chapter 11):

$$\bar{A} = \begin{bmatrix} 0 & 1 & 0 \\ -2 & -3 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad \bar{B} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

Controller poles: $-3 \pm j3$ and -5 (the extra pole for the integrator state).

$$\bar{K} = \texttt{place}(\bar{A}, \bar{B}, [-3+3j,\ -3-3j,\ -5]) = \begin{bmatrix} K_1 & K_2 & -K_I \end{bmatrix}.$$

Numerically, this gives $\bar{K} = \begin{bmatrix} 46 & 8 & -90 \end{bmatrix}$, so $K = \begin{bmatrix} 46 & 8 \end{bmatrix}$ and $K_I = 90$.

Observer poles: -12 and -13 (same as before — the observer sees the original 2-state system).

$$L = \texttt{place}(A', C', [-12,\ -13])' = \begin{bmatrix} 22 \\ 88 \end{bmatrix}.$$

MATLAB code: Listing 12.3 computes the state feedback, integral, and observer gains.

```
A = [0 1; -2 -3];  B = [0; 1];  C = [1 0];
n = size(A,1);

% Augmented system for integral action
Abar = [A, zeros(n,1); -C, 0];
Bbar = [B; 0];

% Controller poles (n+1 = 3 poles)
pc = [-3+3j, -3-3j, -5];
Kbar = place(Abar, Bbar, pc);
K  = Kbar(1:n);         % state feedback gains
KI = -Kbar(n+1);        % integral gain

% Observer poles (n = 2 poles)
po = [-12, -13];
L = place(A', C', po)';

fprintf('K  = [%.2f  %.2f]\n', K);
fprintf('KI = %.2f\n', KI);
fprintf('L  = [%.2f; %.2f]\n', L);
```

Listing 12.3: Computing gains for observer-based control with integral action

Example 12.4: MATLAB: Simulating the complete observer-based controller

We now simulate the full closed-loop system with observer, state feedback, and integral action for the mass–spring–damper. The closed-loop state vector is $\begin{bmatrix} x^\top & \hat{x}^\top & x_I \end{bmatrix}^\top$ (5 states in total). Listing 12.4 builds the composite system matrices and uses `step()` to verify the response.

```
%% Full simulation: observer + state feedback + integral action
A = [0 1; -2 -3];  B = [0; 1];  C = [1 0];  D = 0;
n = size(A,1);

% Augmented system for integral action
Abar = [A, zeros(n,1); -C, 0];
Bbar = [B; 0];

% State feedback + integral gain (3 controller poles)
pc = [-3+3j, -3-3j, -5];
Kbar = place(Abar, Bbar, pc);
K  = Kbar(1:n);          % K = [46  8]
KI = -Kbar(n+1);         % KI = 90

% Observer gain (2 observer poles, 4x faster)
po = [-12, -13];
L  = place(A', C', po)';  % L = [22; 88]

% Closed-loop in coordinates [x; xhat; xI]
%   dx    = A*x  + B*u,          u = -K*xhat + KI*xI
%   dxhat = (A-LC)*xhat + B*u + L*C*x
%   dxI   = r - C*x
Acl = [A,        -B*K,     B*KI;        % dx/dt
       L*C,  A-L*C-B*K,   B*KI;        % dxhat/dt
      -C,     zeros(1,n),     0];        % dxI/dt
Bcl = [zeros(n,1); zeros(n,1); 1];     % input = r
Ccl = [C, zeros(1,n), 0];              % output = y = C*x
Dcl = 0;

sys_cl = ss(Acl, Bcl, Ccl, Dcl);
disp('Closed-loop poles:'); disp(eig(Acl));

% Step response
figure;
[y,t,xall] = step(sys_cl, 5);
subplot(2,1,1);
plot(t, y, 'b', 'LineWidth', 1.5); hold on;
yline(1, 'k--'); ylabel('Output y(t)');
title('Observer-based control with integral action');

% True vs estimated states
subplot(2,1,2);
plot(t, xall(:,1), 'b', t, xall(:,3), 'r--', 'LineWidth', 1.5);
hold on;
plot(t, xall(:,2), 'b:', t, xall(:,4), 'r:', 'LineWidth', 1.5);
legend('x_1 (true)','xhat_1','x_2 (true)','xhat_2');
ylabel('States'); xlabel('Time (s)');
title('True vs estimated states');
```

Listing 12.4: Full simulation of observer-based control with integral action

Running this code produces a step response that settles with zero steady-state error (thanks to integral action) and shows the observer estimates converging rapidly to the true states. You should see that the five closed-loop poles are exactly $\{-3 \pm j3,\ -5,\ -12,\ -13\}$ — confirming the separation principle.

12.8 Simulink Implementation

Figure 12.3 shows the Simulink implementation. The key components are:

1. **Plant block**: a State-Space block with matrices A, B, C, D.
2. **Observer block**: a State-Space block implementing (12.3), with input $[u; y]$ and output $\hat{x}$.
3. **Integrator block**: integrates the error $r - y$ to produce x_I.
4. **Gain blocks**: $-K$ multiplied by $\hat{x}$, K_I multiplied by x_I, summed to form u.

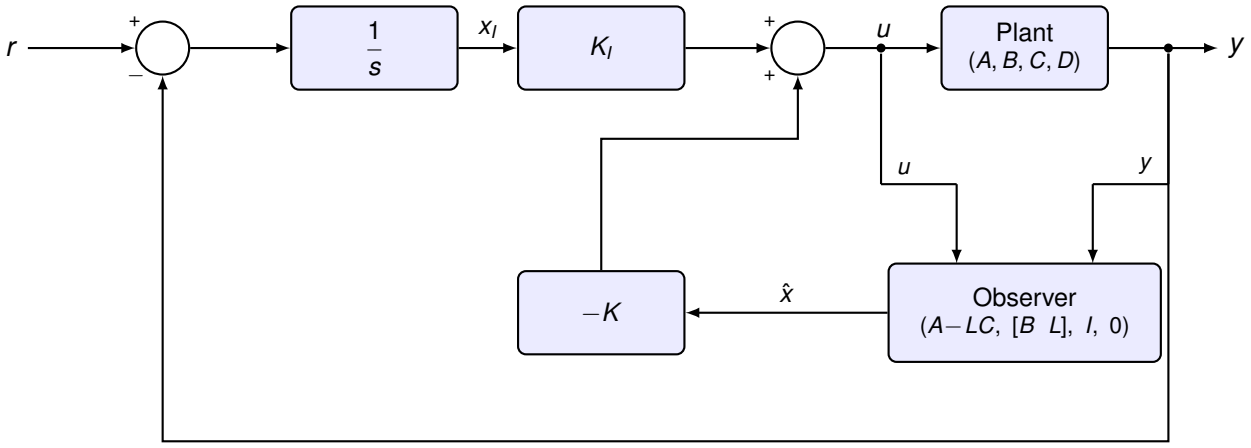


Figure 12.3: Observer-based output-feedback control with integral action. The outer loop subtracts y from r and integrates the error. The observer receives u and y from above; its output $\hat{x}$ passes through $-K$ back to the control summing junction.

The observer state-space block has the following matrices:

$$A_{\text{obs}} = A - LC, \quad B_{\text{obs}} = \begin{bmatrix} B & L \end{bmatrix}, \quad C_{\text{obs}} = I_n, \quad D_{\text{obs}} = 0,$$

with input vector $\begin{bmatrix} u \\ y \end{bmatrix}$ and output $\hat{x}$. Listing 12.5 sets up these matrices in MATLAB.

```
% Observer as a state-space system
A_obs = A - L*C;
B_obs = [B, L];        % inputs: [u; y]
C_obs = eye(n);         % output: full estimated state
D_obs = zeros(n, 2);

% Use these in a Simulink State-Space block
```

Listing 12.5: Setting up the observer block in MATLAB for Simulink

For the DC motor design from the previous section, the Simulink model uses

$$K = \begin{bmatrix} 46 & 8 \end{bmatrix}, \qquad K_I = 90, \qquad L = \begin{bmatrix} 22 \\ 88 \end{bmatrix}.$$

With a unit-step reference and a nonzero initial observer error, the simulation should produce the responses in Figures 12.4 and 12.5.

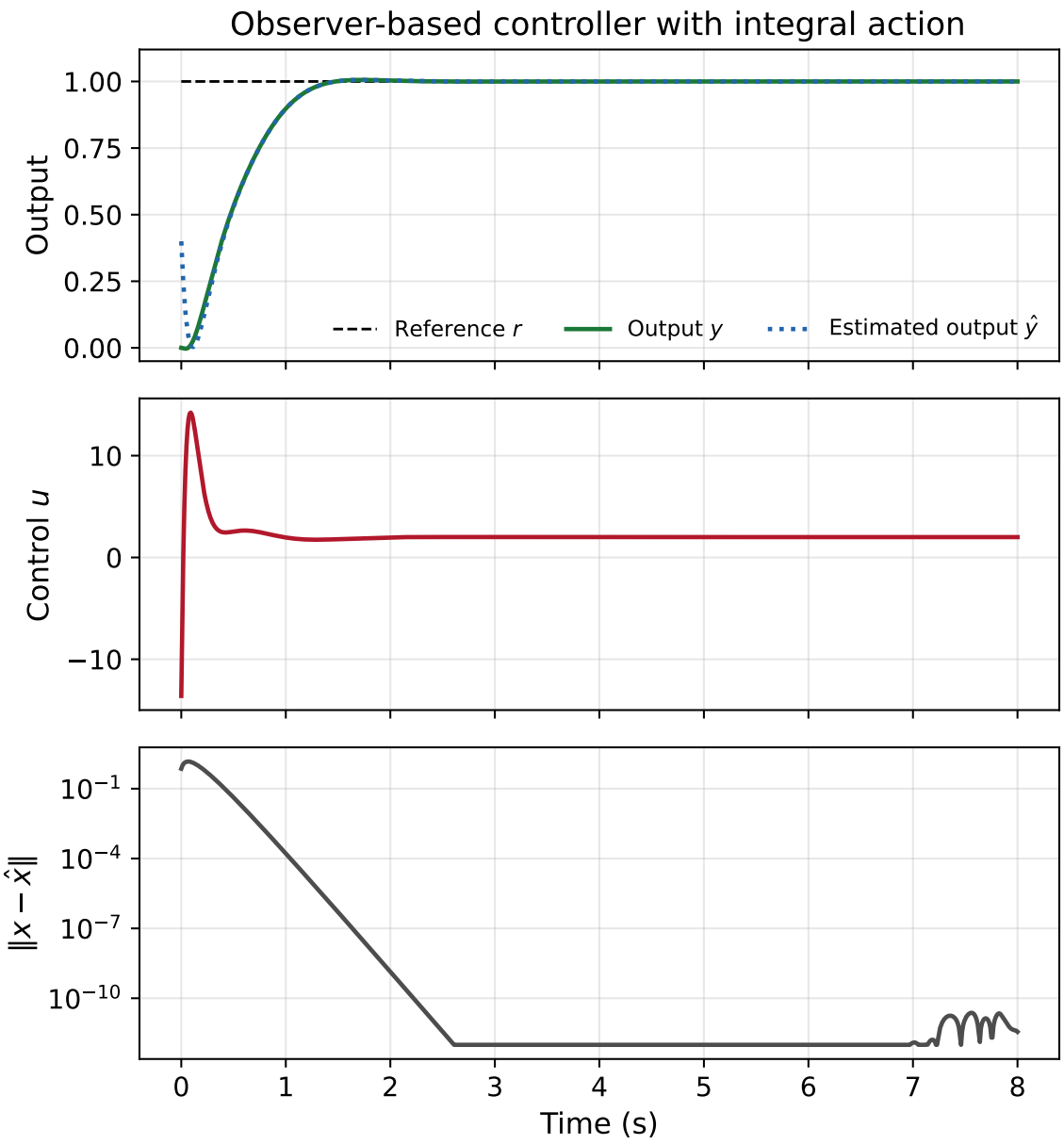


Figure 12.4: Closed-loop response for the observer-based controller with integral action. The top plot shows output tracking, the middle plot shows the control signal, and the bottom plot shows the decay of the estimation-error norm.

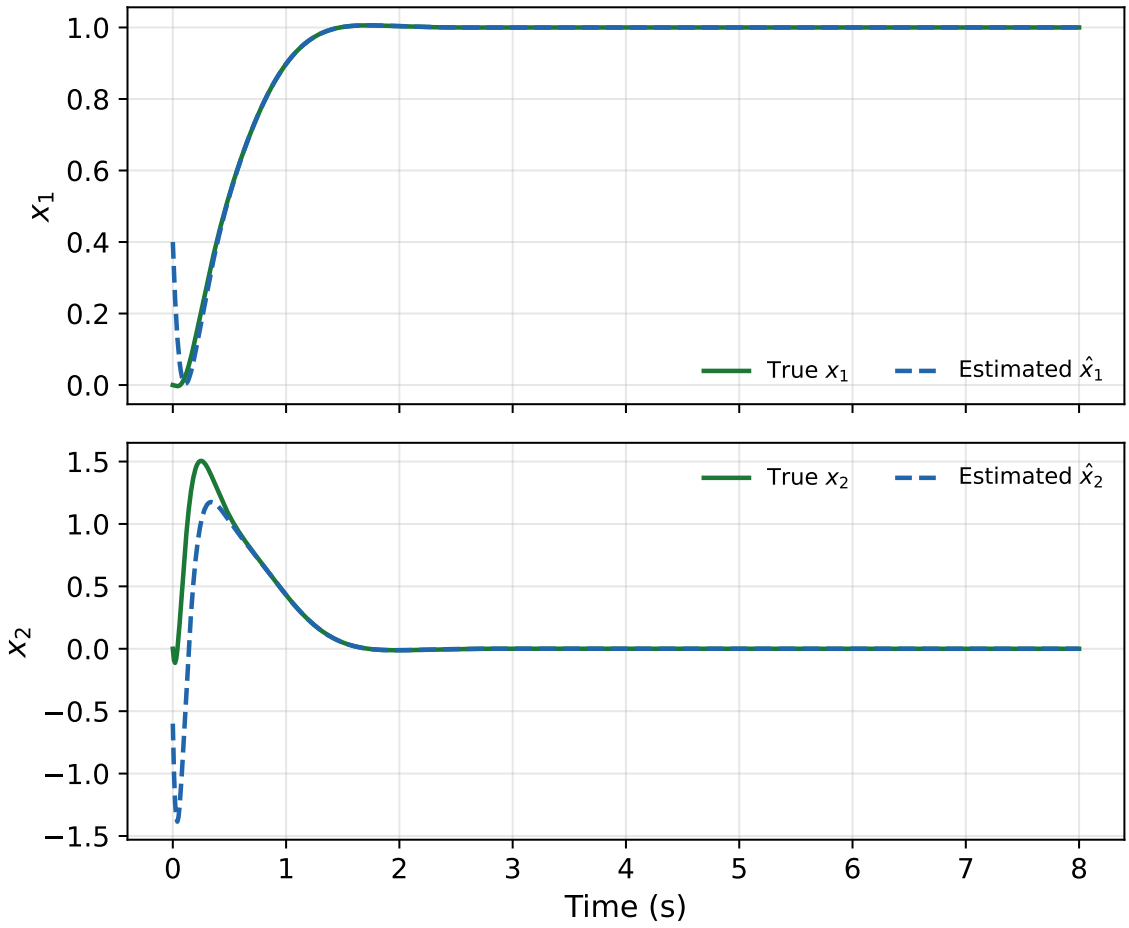


Figure 12.5: True and estimated states for the same simulation. The observer states converge rapidly to the plant states, which is what the Simulink observer block should reproduce.

12.9 Robustness: What Happens When the Model is Not Perfect?

What happens when the observer matrices differ from the true plant?

- The estimation error $e(t)$ no longer converges to exactly zero — a small residual error remains.
- If integral action is included, the steady-state tracking error is still zero for step references, because the integrator compensates for the model mismatch.
- The transient performance degrades, but the system remains stable provided the model errors are not too large.

Remark 12.5

This is another reason why integral action is essential in practice. Without it, any model error leads to a permanent steady-state offset. With it, the integrator absorbs the mismatch.

12.10 Summary

Chapter Summary

- An **observer** (Luenberger observer) estimates the unmeasured states from the known input u and measured output y.
- The observer equation is $\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x})$.
- The estimation error decays according to $\dot{e} = (A - LC)e$. The gain L is chosen to place the eigenvalues of $A - LC$ at desired locations.
- Observer design requires **observability** of (A, C).
- Observer poles are typically placed 3–5$\times$ faster than the controller poles.
- The **separation principle** guarantees that the controller (K) and observer (L) can be designed independently. The closed-loop poles are the union of both sets.
- For zero steady-state error, add **integral action**: $u = -K\hat{x} + K_I x_I$, where $\dot{x}_I = r - y$.
- MATLAB: `L = place(A', C', po)'` for the observer gain.

12.11 Beyond This Module: A Glimpse of Advanced Control

The tools from this module — transfer functions, state-space models, stability analysis, PID tuning, pole placement, observers, and integral action — form the foundation of control engineering. This section briefly introduces directions you may encounter in advanced modules or industry. None are examinable.

12.11.1 Discrete-time control

In practice, controllers run on digital processors that sample at discrete intervals. Discrete-time control replaces the Laplace transform with the **z-transform** and differential equations with difference equations. The design concepts (pole placement, observers, integral action) carry over directly. MATLAB's `c2d()` converts continuous-time designs to discrete-time equivalents.

12.11.2 Linear Quadratic Regulator (LQR)

LQR finds the optimal state feedback gain K by minimising a cost function:

$$J = \int_0^\infty \left[x^\top Q\, x + u^\top R\, u\right] dt,$$

where Q penalises state deviations and R penalises control effort. LQR produces guaranteed stability margins and avoids trial-and-error pole selection. In MATLAB: `K = lqr(A, B, Q, R)`.

12.11.3 Kalman filter

The Luenberger observer assumes deterministic signals. The **Kalman filter** is the optimal observer for systems with Gaussian noise — it minimises the mean-squared estimation error. It is the dual of LQR: LQR optimises K, the Kalman filter optimises L. Applications include GPS, inertial navigation, robotics, and financial modelling.

12.11.4 Linear Quadratic Gaussian (LQG) control

Combining LQR with the Kalman filter gives the **LQG controller**. By the separation principle, the two designs remain independent. LQG is widely used in aerospace, process control, and robotics.

12.11.5 Robust control (H_∞)

Robust control explicitly accounts for model uncertainty. The H_∞ framework guarantees stability and performance for all plants within a defined uncertainty set — essential in aerospace and automotive applications requiring worst-case guarantees.

12.11.6 Model Predictive Control (MPC)

At each time step, MPC solves an optimisation problem over a finite prediction horizon, explicitly handling:

- constraints on inputs (actuator limits),
- constraints on states and outputs (safety limits),
- a cost function balancing tracking performance and control effort.

Only the first control action is applied; the optimisation repeats at the next step. MPC is standard in chemical process control, building energy management, autonomous vehicles, and robotics.

12.11.7 Nonlinear and adaptive control

All methods in this module assume linearity. Advanced nonlinear techniques include feedback linearisation, sliding mode control, backstepping, and Lyapunov-based design. **Adaptive control** adjusts controller parameters online as the plant changes.

12.11.8 Machine learning and data-driven control

Reinforcement learning can discover control policies from interaction with the environment, and neural networks can approximate complex nonlinear dynamics. Safety-critical applications still require model-based guarantees — ML methods are most powerful when combined with the classical foundations from this module.

Remark 12.6

Every method above builds on the concepts from this module. State-space models, controllability, observability, stability, pole placement, and observers are the language of modern control theory.

12.12 End-of-Chapter Problems

Problem 1. Open-loop vs closed-loop observer. Consider $A = \begin{bmatrix} 0 & 1 \\ -5 & -6 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$.

(a) Write the error dynamics for the open-loop estimator (no correction term). What are the eigenvalues?

(b) Now design a Luenberger observer with poles at -20 and -25. Find L.

(c) Compare the convergence speed of both approaches.

Problem 2. Observability check. For each system, determine whether an observer can be designed:

(a) $A = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 1 \end{bmatrix}$.

(b) $A = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$.

(c) $A = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix}$, $C = \begin{bmatrix} 0 & 1 \end{bmatrix}$.

Problem 3. Observer gain computation. Given $A = \begin{bmatrix} 0 & 1 \\ -10 & -7 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$:

(a) Verify observability.

(b) Compute L so that the observer poles are at -30 and -35.

(c) Verify using MATLAB: `L = place(A', C', [-30, -35])'`.

Problem 4. Separation principle verification. Using the system from Problem 3:

(a) Design K to place the controller poles at $-5 \pm j5$.

(b) Design L to place the observer poles at -25 and -30.

(c) Write the 4×4 closed-loop system matrix in coordinates (x, e).

(d) Verify that the eigenvalues are exactly $\{-5 + 5j,\ -5 - 5j,\ -25,\ -30\}$.

Problem 5. Observer-based controller with integral action. For the system $A = \begin{bmatrix} 0 & 1 \\ -4 & -5 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$:

(a) Form the augmented system for integral action.

(b) Compute $\bar{K} = [K \quad -K_I]$ with controller poles at $-4 \pm j4$ and -6.

(c) Compute L with observer poles at -20 and -25.

(d) Simulate the step response in MATLAB. Verify zero steady-state error.

Problem 6. Effect of observer speed. Using the system and controller from Problem 5:

(a) Simulate with observer poles at -8 and -10 (slow observer).

(b) Simulate with observer poles at -40 and -50 (fast observer).

(c) Compare the transient responses. What do you observe about the initial transient?

(d) Discuss the trade-off between convergence speed and noise sensitivity.

Problem 7. Robustness test. Using the design from Problem 5, suppose the true plant has $A_{true} = \begin{bmatrix} 0 & 1 \\ -5 & -6 \end{bmatrix}$ (a 25% parameter error).

(a) Simulate the step response with the controller designed for the nominal plant.

(b) Is the system still stable?

(c) What happens to the steady-state error? Explain why integral action helps.

Problem 8. MATLAB comprehensive exercise. Consider the third-order system:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}.$$

(a) Verify controllability and observability.

(b) Design state feedback with poles at -4, -5, -6.

(c) Design an observer with poles at -20, -25, -30.

(d) Add integral action with an additional pole at -8.

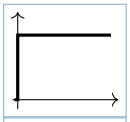
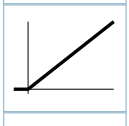
(e) Simulate the complete observer-based controller with integral action for a unit step reference.

(f) Plot the true state $x(t)$ and estimated state $\hat{x}(t)$ on the same graph. Verify that the estimation error converges quickly.

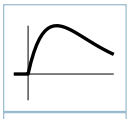
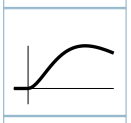
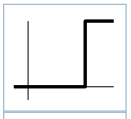
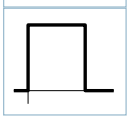
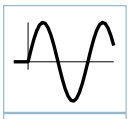
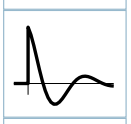
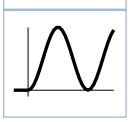
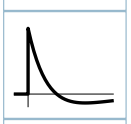
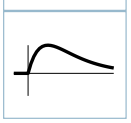
Laplace Transform Table

Laplace transform conventions

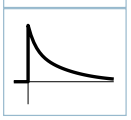
This appendix uses unilateral Laplace transforms and causal time signals. The sketches show representative waveform shapes; the algebraic pairs are the reference definitions.

Description	Time domain	Sketch	Laplace domain
Definition	$f(t) = \mathcal{L}^{-1}\{F(s)\}$		$F(s) = \mathcal{L}\{f(t)\} = \int_0^{\infty} f(t)e^{-st}\,\mathrm{d}t$
Linearity	$g(t) = af(t)$		$G(s) = aF(s)$
	$g(t) = f(t) + h(t)$		$G(s) = F(s) + H(s)$
Convolution	$g(t) = (f * h)(t) = \int_0^t f(\tau)h(t-\tau)\,\mathrm{d}\tau$		$G(s) = F(s)H(s)$
Differentiation	$g(t) = \frac{\mathrm{d}f(t)}{\mathrm{d}t}$		$G(s) = sF(s) - f(0)$
	$g(t) = \frac{\mathrm{d}^2 f(t)}{\mathrm{d}t^2}$		$G(s) = s^2F(s) - sf(0) - f'(0)$
	$g(t) = \frac{\mathrm{d}^n f(t)}{\mathrm{d}t^n}$		$G(s) = s^nF(s) - \sum_{k=0}^{n-1} s^{n-1-k} f^{(k)}(0)$
Integration	$g(t) = \int_0^t f(\tau)\,\mathrm{d}\tau$		$G(s) = \frac{F(s)}{s}$
	$g(t) = \int_0^t \int_0^{\lambda} f(\tau)\,\mathrm{d}\tau\,\mathrm{d}\lambda$		$G(s) = \frac{F(s)}{s^2}$
Unit step	$f(t) = u(t)$		$F(s) = \frac{1}{s}$
Unit ramp	$f(t) = t\,u(t)$		$F(s) = \frac{1}{s^2}$
Unit parabola	$f(t) = \frac{1}{2}t^2u(t)$		$F(s) = \frac{1}{s^3}$
Powers of t	$f(t) = t^n u(t)$		$F(s) = \frac{n!}{s^{n+1}}$
Exponential decay	$f(t) = e^{-at}u(t)$		$F(s) = \frac{1}{s+a}$

Continued on next page

Description	Time domain	Sketch	Laplace domain
	$f(t) = te^{-at}u(t)$		$F(s) = \frac{1}{(s+a)^2}$
	$f(t) = t^2e^{-at}u(t)$		$F(s) = \frac{2}{(s+a)^3}$
	$f(t) = t^ne^{-at}u(t)$		$F(s) = \frac{n!}{(s+a)^{n+1}}$
Pure delay T	$f(t) = g(t-T)u(t-T)$		$F(s) = e^{-sT}G(s)$
Delayed unit step	$f(t) = u(t-T)$		$F(s) = \frac{e^{-sT}}{s}$
Unit impulse	$f(t) = \delta(t)$		$F(s) = 1$
Rectangular pulse	$f(t) = u(t) - u(t-T)$		$F(s) = \frac{1-e^{-sT}}{s}$
Sinusoid	$f(t) = \sin(\omega t)u(t)$		$F(s) = \frac{\omega}{s^2+\omega^2}$
	$f(t) = \cos(\omega t)u(t)$		$F(s) = \frac{s}{s^2+\omega^2}$
	$f(t) = e^{-at}\sin(\omega t)u(t)$		$F(s) = \frac{\omega}{(s+a)^2+\omega^2}$
	$f(t) = e^{-at}\cos(\omega t)u(t)$		$F(s) = \frac{s+a}{(s+a)^2+\omega^2}$
	$f(t) = 1 - \cos(\omega t)$		$F(s) = \frac{\omega^2}{s(s^2+\omega^2)}$
Compound signals	$f(t) = 1 - e^{-at}$		$F(s) = \frac{1}{s} - \frac{1}{s+a} = \frac{a}{s(s+a)}$
	$f(t) = \frac{1}{a}\left(t - \frac{1-e^{-at}}{a}\right)$		$F(s) = \frac{1}{s^2(s+a)}$
	$f(t) = \frac{1}{a^2}\left(1 - e^{-at} - ate^{-at}\right)$		$F(s) = \frac{1}{s(s+a)^2}$
	$f(t) = (1-at)\,e^{-at}$		$F(s) = \frac{s}{(s+a)^2}$
	$f(t) = \frac{e^{-at} - e^{-bt}}{b-a}$		$F(s) = \frac{1}{(s+a)(s+b)}$

Continued on next page

Description	Time domain	Sketch	Laplace domain
	$f(t) = 1 - \frac{b}{b-a}e^{-at} + \frac{a}{b-a}e^{-bt}$		$F(s) = \frac{ab}{s(s+a)(s+b)}$
	$f(t) = \frac{e^{-at}}{(b-a)(c-a)} + \frac{e^{-bt}}{(a-b)(c-b)} + \frac{e^{-ct}}{(a-c)(b-c)}$		$F(s) = \frac{1}{(s+a)(s+b)(s+c)}$
2nd order impulse	$f(t) = \frac{\omega_n}{\sqrt{1-\zeta^2}} e^{-\zeta\omega_n t} \sin\left(\omega_n\sqrt{1-\zeta^2}\, t\right)$		$F(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$
2nd order step	$f(t) = 1 + A_1 e^{P_1 t} + A_2 e^{P_2 t}$		$F(s) = \frac{\omega_n^2}{s\left(s^2 + 2\zeta\omega_n s + \omega_n^2\right)}$
Alternative underdamped form	$f(t) = 1 - \frac{e^{-\zeta\omega_n t}}{\sqrt{1-\zeta^2}} \sin\left(\omega_n\sqrt{1-\zeta^2}\, t + \cos^{-1}\zeta\right)$		$F(s) = \frac{\omega_n^2}{s\left(s^2 + 2\zeta\omega_n s + \omega_n^2\right)}$

Here $a, T, \omega, \omega_n \in \mathbb{R}$, $T > 0$, $t, \tau, \lambda \geq 0$, and n is a non-negative integer. The second-order sine forms assume $0 \leq \zeta < 1$. The pole form of the second-order step response assumes distinct poles, with

$$A_1 = -\frac{1}{2} - \frac{\zeta}{2\sqrt{\zeta^2-1}}, \qquad P_1 = -\zeta\omega_n + \omega_n\sqrt{\zeta^2-1},$$

$$A_2 = -\frac{1}{2} + \frac{\zeta}{2\sqrt{\zeta^2-1}}, \qquad P_2 = -\zeta\omega_n - \omega_n\sqrt{\zeta^2-1}.$$

Glossary of Terms

Ackermann's formula	A closed-form expression that computes the state feedback gain K directly from the controllability matrix and the desired characteristic polynomial	348
Actuator	The physical device that converts the controller's command into a physical action (force, torque, voltage) applied to the plant	4
Anti-windup	A strategy to prevent integral windup by limiting or resetting the integrator when the actuator saturates	257
Asymptotic stability	A system property where every natural response decays to zero as $t \to \infty$; equivalently, all poles lie in the open left-half s-plane	202
Augmented system	A state-space model extended with an integral-of-error state to enable integral action within state feedback	353
Bandwidth	The frequency range over which the closed-loop system responds effectively, approximately the -3 dB frequency of the closed-loop magnitude	281
BIBO stability	Bounded-input bounded-output stability: every bounded input produces a bounded output	201
Block diagram	A graphical representation of a control system showing subsystems as blocks with signals flowing through summing junctions and pick-off points	26, 133
Block diagram reduction	Simplifying a block diagram by combining series, parallel, and feedback connections into a single equivalent transfer function	135
Bode plot	Magnitude (dB) and phase (degrees) of $G(j\omega)$ plotted against logarithmic frequency	266
Break frequency	The frequency at which the asymptotic magnitude slope changes on a Bode plot; for $1/(s/a+1)$, the break is at $\omega = a$	267
Causality	The requirement that a system cannot respond to future inputs; the transfer function must be proper ($m \leq n$)	62
Characteristic equation	The equation $\det(sI - A) = 0$ whose roots are the poles that determine stability	199
Closed-loop control	A system in which the output is measured and fed back for comparison with the reference, allowing automatic error correction	4
Closed-loop transfer function	The transfer function from reference to output; for unity feedback with loop gain $L(s)$: $T(s) = L(s)/(1 + L(s))$	16

Controllability	The property that any initial state can be driven to any final state in finite time by a suitable input; tested by $\operatorname{rank}(\mathcal{C}) = n$	325
Controllability matrix	$\mathcal{C} = [B \quad AB \quad A^2B \quad \cdots \quad A^{n-1}B]$; its rank determines whether (A, B) is controllable	327
Controllable canonical form	A state-space realisation where the transfer function denominator coefficients appear in the last row of A and the system is guaranteed controllable	334
Controller	The component that processes the error signal and determines the control effort applied to the plant	4
Convolution	$y(t) = \int_0^t g(t-\tau)x(\tau)\,d\tau$; computes the output of an LTI system from its impulse response and input; corresponds to multiplication in the Laplace domain	30
Damped natural frequency	$\omega_d = \omega_n\sqrt{1-\zeta^2}$; the frequency at which an underdamped second-order system oscillates	166
Damping ratio	The parameter ζ determining whether a second-order response is underdamped ($0 < \zeta < 1$), critically damped ($\zeta = 1$), or overdamped ($\zeta > 1$)	166
Decibel (dB)	Logarithmic unit: $20\log_{10}\lvert G(j\omega)\rvert$; converts multiplicative gains into additive quantities on Bode plots	268
Derivative action	The D term of a PID controller; responds to the rate of change of the error, adding damping and improving transient response	234
Derivative kick	A transient spike in the control signal caused by the D term acting on a step reference change; mitigated by differentiating only the output	243
Disturbance	An unwanted external signal acting on the plant that pushes the output away from its desired value	7
Duality	(A, B) is controllable if and only if $(A^\top, B^\top)$ is observable; links the two structural properties through transposition	333
Error signal	$e(t) = r(t) - y(t)$; the difference between the reference and the measured output	7
Feedthrough matrix	The matrix D in $y = Cx + Du$; when $D \neq 0$ the output depends instantaneously on the input	108
Final value theorem	$\lim_{t\to\infty} f(t) = \lim_{s\to 0} sF(s)$, valid when all poles of $sF(s)$ lie in the open left-half plane	35
First-order system	$G(s) = K/(\tau s + 1)$; characterised by DC gain K and time constant τ; step response is a rising exponential	153
Frequency response	$G(j\omega)$: describes how the system modifies the amplitude and phase of sinusoids at each frequency	265
Gain crossover frequency	ω_{gc}: the frequency where $\lvert L(j\omega)\rvert = 1$ (0 dB); used to compute phase margin	281
Gain margin	The factor by which open-loop gain can increase before instability, measured at the phase crossover frequency	284
Hysteresis	A dead band in on–off control to prevent rapid switching near the set point	8

Impedance	$Z(s) = V(s)/I(s)$ for a circuit element; replaces components with algebraic expressions (R, sL, $1/sC$) for Laplace-domain analysis	69
Impulse response	$g(t) = \mathcal{L}^{-1}\{G(s)\}$; the output when the input is $\delta(t)$; fully characterises an LTI system	30
Initial value theorem	$f(0^+) = \lim_{s\to\infty} sF(s)$; determines the initial value without computing the full inverse transform	35
Integral action	The I term of a PID controller; accumulates past error, eliminating steady-state error for step references	234
Integral windup	Excessive integrator accumulation during actuator saturation, causing sluggish recovery	257
Inverse Laplace transform	$f(t) = \mathcal{L}^{-1}\{F(s)\}$; recovers the time-domain signal, typically via partial fraction expansion	30
Lag compensator	$C(s) = K_c(Ts + 1)/(\beta Ts + 1)$ with $\beta > 1$; boosts low-frequency gain for better steady-state accuracy	295
Laplace transform	$F(s) = \int_0^\infty f(t)e^{-st}\,dt$; converts time-domain signals to functions of s, turning differential equations into algebraic ones	30
Lead compensator	$C(s) = K_c(Ts + 1)/(\alpha Ts + 1)$ with $\alpha < 1$; adds positive phase near crossover to improve phase margin and speed	295
Linear time-invariant (LTI) system	A system with constant-coefficient linear dynamics where superposition holds and the transfer function is independent of time	30
Loop transfer function	$L(s) = C(s)G(s)H(s)$; the product of all transfer functions around the feedback loop	284
Luenberger observer	$\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x})$; estimates unmeasured states from known inputs and measured outputs	370
Marginal stability	Bounded but non-decaying response; occurs with simple imaginary-axis poles and all other poles in the left-half plane	205
Mason's gain formula	Computes a transfer function from a signal-flow graph using forward paths, loop gains, and non-touching loop products	138
Matrix exponential	$e^{At} = I + At + (At)^2/2! + \cdots$; the state transition matrix mapping $x(0)$ to $x(t)$	119
Natural frequency	ω_n in $\omega_n^2/(s^2+2\zeta\omega_n s+\omega_n^2)$; the oscillation frequency with zero damping	166
Observability	The property that the initial state can be uniquely determined from the input and output over a finite interval; tested by $\operatorname{rank}(\mathcal{O}) = n$	325
Observability matrix	$\mathcal{O} = [C;\ CA;\ CA^2;\ \cdots;\ CA^{n-1}]$; its rank determines whether (A, C) is observable	325
Observable canonical form	The transpose of CCF; guaranteed observable, with denominator coefficients in the first column of A	334
Observer-based controller	Combines state feedback with a Luenberger observer, using $\hat{x}$ when not all states are measured	369
On–off control	The actuator is either fully on or fully off; also called bang–bang control	8

Open-loop control	Control without output measurement; performance depends entirely on calibration	4
Partial fraction expansion	Decomposing a rational $F(s)$ into simpler fractions matching known inverse-transform pairs	45
Peak time	$t_p = \pi/\omega_d$; time for an underdamped second-order step response to reach its first maximum	171
Percentage overshoot	$M_p = 100\, e^{-\pi\zeta/\sqrt{1-\zeta^2}}\%$; the amount the step response exceeds its final value	171
Phase crossover frequency	ω_{pc}: frequency where the open-loop phase reaches -180°; used to compute gain margin	281
Phase margin	$PM = 180^\circ + \angle L(j\omega_{gc})$; the phase lag that would bring the system to the stability boundary	284
PID controller	$C(s) = K_p + K_I/s + K_D s$; the sum of proportional, integral, and derivative actions; the most widely used industrial controller	233
Plant	The physical system being controlled (motor, heat exchanger, vehicle, etc.)	4
Pole placement	Choosing K so that the eigenvalues of $A - BK$ match desired closed-loop pole locations	344
Poles	Values of s making the transfer function denominator zero; they determine the natural modes, speed, and stability of the response	45
Proper transfer function	Numerator degree $\leq$ denominator degree ($m \leq n$); strictly proper when $m < n$	63
Proportional action	The P term of a PID controller; output proportional to the current error; provides immediate response but usually leaves steady-state error	234
Reference signal	The desired value or set point $r(t)$ that the controller aims to track	5
Resonant peak	A rise in $\lvert G(j\omega)\rvert$ near a lightly damped pole, corresponding to overshoot in the time domain	274
Rise time	Time for the step response to go from 10% to 90% of its final value	157
Routh array	The table of coefficients in the Routh–Hurwitz test; sign changes in the first column equal the number of unstable roots	211
Routh–Hurwitz criterion	An algebraic test determining the number of right-half-plane roots of a polynomial from its coefficients, without computing the roots	209
Second-order system	$G(s) = K\omega_n^2/(s^2 + 2\zeta\omega_n s + \omega_n^2)$; the standard model for transient-response analysis	165
Sensor	The measurement device that closes the feedback loop by converting the physical output into a signal for comparison with the reference	7
Separation principle	State feedback gain K and observer gain L can be designed independently; the closed-loop poles are the union of the controller and observer poles	375
Settling time	Time for the step response to stay within a specified band (commonly 2%) of the final value; approximately $4/(\zeta\omega_n)$	158

State equation	$\dot{x}(t) = Ax(t) + Bu(t)$; describes how the internal state of an LTI system evolves	104
State feedback	$u = -Kx + r$; every state variable is fed back through a gain vector, enabling placement of all n closed-loop poles when controllable	343
State matrix	The matrix A in $\dot{x} = Ax + Bu$; its eigenvalues determine the system's natural modes and stability	107
State transition matrix	$\Phi(t) = e^{At}$; maps $x(0)$ to $x(t)$ for the unforced system	119
State variable	A variable in the minimum set needed to describe a system's internal condition; together with future inputs, determines the future response	103
State vector	$x(t) = [x_1(t), \dots, x_n(t)]^\top$; its dimension n is the system order	103
State-space representation	$\dot{x} = Ax + Bu$, $y = Cx + Du$; a model expressed through matrices A, B, C, D	103
Static error constants	K_p, K_v, K_a: position, velocity, and acceleration error constants relating system type to steady-state error	195
Steady-state error	$\lim_{t\to\infty} e(t)$; measures how accurately the output tracks the reference after transients decay	186
Summing junction	The point in a block diagram where signals are added or subtracted; forms the error $e = r - y$	7
System type	The number of free integrators (poles at $s = 0$) in the open-loop transfer function; determines which reference inputs can be tracked with zero steady-state error	195
Time constant	τ of a first-order system; the step response reaches 63.2% at $t = \tau$ and is practically complete by 4τ–5τ	154
Transfer function	$G(s) = Y(s)/U(s)$ under zero initial conditions; characterises the input–output dynamics of an LTI system	30
Unity feedback	Feedback with $H(s) = 1$; the output is compared directly with the reference	25
Zeros	Values of s making the transfer function numerator zero; they shape the transient response but do not determine stability alone	64
Ziegler–Nichols tuning	Two empirical PID tuning methods: the reaction curve method (open-loop step response) and the sustained oscillation method (ultimate gain K_u and period P_u)	246

Bibliography

[1] Ibn al-Razzaz al Jazari. *The Book of Knowledge of Ingenious Mechanical Devices*. D. Reidel Publishing Company, Dordrecht, 1974. Translated by Donald R. Hill; original work completed in 1206.

[2] Hendrik Wade Bode. *Network Analysis and Feedback Amplifier Design*. D. Van Nostrand Company, New York, 1945.

[3] CENELEC. *EN 50128: Railway Applications — Communication, Signalling and Processing Systems — Software for Railway Control and Protection Systems*, 2011.

[4] CENELEC. *EN 50129: Railway Applications — Communication, Signalling and Processing Systems — Safety Related Electronic Systems for Signalling*, 2018.

[5] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2nd edition, 2013.

[6] International Electrotechnical Commission. *IEC 61513: Nuclear Power Plants — Instrumentation and Control Important to Safety — General Requirements for Systems*, 2011. Edition 2.0.

[7] International Electrotechnical Commission. *IEC 62304: Medical Device Software — Software Life Cycle Processes*, 2015. Consolidated version IEC 62304:2006+AMD1:2015 CSV, edition 1.1.

[8] International Organization for Standardization. *ISO 26262: Road Vehicles — Functional Safety*, 2018. Second edition.

[9] Rudolf Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. Transactions of the ASME, Series D.

[10] Hassan K. Khalil. *Control Systems: An Introduction*. Michigan Publishing Services, Ann Arbor, MI, 2023.

[11] David G. Luenberger. Observing the state of a linear system. *IEEE Transactions on Military Electronics*, 8(2):74–80, 1964.

[12] David G. Luenberger. An introduction to observers. *IEEE Transactions on Automatic Control*, 16(6):596–602, 1971.

[13] James Clerk Maxwell. On governors. *Proceedings of the Royal Society of London*, 16:270–283, 1868.

[14] Manfred Morari and Jay H. Lee. Model predictive control: Past, present and future. *Computers & Chemical Engineering*, 23(4–5):667–682, 1999.

[15] Harry Nyquist. Regeneration theory. *Bell System Technical Journal*, 11(1):126–147, 1932.

[16] Quanser Inc. *Lab Manual: Control Systems Design and Analysis Using the Quanser Controls Board for NI ELVIS III*. Markham, Ontario, Canada, 2018. Lab manual.

[17] Edward John Routh. *A Treatise on the Stability of a Given State of Motion: Particularly Steady Motion*. Macmillan and Co., London, 1877.

[18] RTCA, Inc. *DO-254: Design Assurance Guidance for Airborne Electronic Hardware*, 2000.

[19] RTCA, Inc. *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*, 2011.

[20] SAE International. *ARP 4754A: Guidelines for Development of Civil Aircraft and Systems*, 2010. Revision A, revised December 2010.

[21] Kemin Zhou, John C. Doyle, and Keith Glover. *Robust and Optimal Control*. Prentice Hall, Upper Saddle River, NJ, 1996.