Cursus: Embedded Control Systems Lectures

Peter Slaets

September 6, 2014

Contents

Introduction

This course text is based on the following international reference books [1], [2], [3], [4]. This course aims at introducing the basic modelling and control concept using vaarious examples up to the software level.

Chapter 1

Dynamic systems: Modeling and Simulation

1.1 Mathematical modeling of dynamic systems

1.1.1 Introduction

Dynamic system modeling deals with the **mathematical modeling of dynamic systems and response analyses** of systems in order to understanding the dynamic nature of each system and improving the system's performance. **Response analyses** are frequently made through computer simulations of dynamic systems. Because many physical systems involve various types of components, a wide variety of different types of dynamic systems have been discussed in the previous courses: "signal and systems" and "control theory". The analysis and design methods presented in this course can be applied to mechanical, electrical, pneumatic, and hydraulic systems, as well as non-engineering systems, such as economic systems and biological systems. It is important that the mechanical engineering student be able to determine dynamic responses of such systems. First several terms need to be clarified in order to fully understand the discussion about dynamic systems.

What is a system ? A **system** is a combination of components acting together to perform a specific objective. A component is a single functioning unit of a system. By no means limited to the domain of the physical phenomena's, the concept of a system can be extended to abstract dynamic phenomena, such as those encountered in economics, transportation, population growth, and biology.

What is a dynamic system ? A system is called **dynamic** if its present output depends on past input; if its current output depends only on current input, the system is known as static. The output of a static system remains constant if the input does not change. The output changes only when the input changes. In a dynamic system, the output changes with time if the system is not in a state of equilibrium. In this course, we are concerned mostly with dynamic systems.

What is a mathematical model ? Any attempt to design a system must begin with a prediction of its performance before the system itself can be designed in detail or actually built. Such prediction is based on a mathematical description of the system's dynamic characteristics. This mathematical description is called a mathematical model. For many physical systems, useful mathematical models are described in terms of differential equations.

Linear and non-linear differential equations. Linear differential equations may be classified as linear, time-invariant differential equations and linear, time-varying differential equations. A linear, time-invariant differential equation is an equation in which a dependent variable and its derivatives appear as linear combinations. An example of such an equation is

$$\frac{d^2x}{dt^2} - 2\frac{dx}{dt} + 8x = 0 \tag{1.1}$$

Here all the coefficients of the terms are constant, a **linear, time-invariant differential equation** is also called a linear, constant-coefficient differential equation.

In the case of a linear, time-varying differential equation, the dependent variable and its derivatives appear as linear combinations, but a coefficient or coefficients of terms may involve the independent variable. An example of this type of differential equation is

$$\frac{d^2x}{dt^2} - 2(1 - \cos(5t))\frac{dx}{dt} + 8x = 0 \tag{1.2}$$

It is important to remember that, in order to be linear, the equation should not contain any powers or other functions of the dependent variables or its derivatives. A differential equation is called nonlinear if it is not linear. Two examples of **nonlinear differential equations** are

$$\frac{d^2x}{dt^2} - 2(1 - x^2)\frac{dx}{dt} + 8x = 0 \tag{1.3}$$

$$\frac{d^2x}{dt^2} - 2\frac{dx}{dt} + 8x^2 = 0 \tag{1.4}$$

Linear systems and non-linear systems. For linear systems, the equations that constitute the model are linear. In this course, we shall deal mostly with linear systems that can be represented by linear, time-invariant ordinary differential equations. The most important property of linear systems is that the principle of **superposition** is applicable. This principle states that the response produced by simultaneous applications of two different forcing functions or inputs is the sum of two individual responses. Consequently, for linear systems, the response to several inputs can be calculated by dealing with one input at a time and then adding the results. As a result of superposition, complicated solutions to linear differential equations can be derived as a sum of simple solutions. In an experimental investigation of a dynamic system, if cause and effect are proportional, thereby implying that the principle of superposition holds, the system can be considered linear. Although physical relationships are often represented by linear equations, in many instances the actual relationships may not be quite linear. In fact, a careful study of physical systems reveals that so-called linear systems are actually linear only within limited operating ranges. For instance, many hydraulic systems and pneumatic systems involve non-linear relationships among their variables, but they are frequently represented by linear equations within limited operating ranges. For **non-linear systems, the most important characteristic is that the principle of superposition is not applicable**. In general, procedures for finding the solutions of problems involving such systems are extremely complicated. Because of the mathematical difficulty involved, it is frequently necessary to **linearize a non-linear system near the operating condition**. Once a non-linear system is approximated by a linear mathematical model, a number of linear techniques may be used for analysis and design purposes.

Continuous-time systems and discrete-time systems. Continuous-time systems are systems in which the signals involved are continuous in time. These systems may be described by differential equations. Discrete-time systems are systems in which one or more variables can change only at discrete instants of time. (These instants may specify the times at which some physical measurement is performed or the times at which the memory of a digital computer is read out.) Discrete-time systems that involve digital signals and, possibly, continuous-time signals as well may be described by difference equations after the appropriate discretization of the continuous-time signals. This topic will be discussed in section 1.5.

1.1.2 Mathematical modeling

What is mathematical modelling.

Mathematical modeling involves descriptions of important system characteristics by sets of equations. By **applying physical laws** to a specific system, it may be possible to develop a mathematical model that describes the dynamics of the system. Such a model may include **unknown parameters**, which must then be evaluated through actual tests. Sometimes, however, the physical laws governing the behavior of a system are not completely defined, and formulating a mathematical model may be impossible. If so, an experimental modeling **identification process** can be used. In this process, the system is subjected to a set of known inputs, and its outputs are measured. Then a mathematical model is derived from the input-output relationships obtained.

Simplicity of mathematical model versus accuracy of results of analysis.

In attempting to build a mathematical model, a compromise must be made between the simplicity of the model and the accuracy of the results of the analysis. It is important to note that the results obtained from the analysis are valid only to the extent that the model approximates a given physical system. In determining a reasonably simplified model, we must decide which physical variables and relationships are negligible and which are crucial to the accuracy of the model. To obtain a model in the form of linear differential equations, any distributed parameters and non-linearities that may be present in the physical system must be ignored. If the effects that these ignored properties have on the response are small, then the results of the analysis of a mathematical model and the results of the experimental study of the physical system will be in good agreement. Whether any particular features are important may be obvious in some cases, but may, in other instances, require physical insight and intuition. Experience is an important factor in this connection. Usually, in solving a new problem, it is desirable first to build a simplified model to obtain a general idea about the solution. Afterward, a more detailed mathematical model can be built and used for a more complete analysis.

Remarks on mathematical models.

An engineer must always keep in mind that the model he or she is analyzing is an **approximate mathematical description** of the physical system; it is not the physical system itself. In reality, no mathematical model can represent any physical component or system precisely. Approximations and assumptions are always involved. Such approximations and assumptions restrict the range of validity of the mathematical model. (The degree of approximation can be determined only by experiments.) So, in making a prediction about a system's performance, any approximations and assumptions involved in the model must be kept in mind.

Mathematical modeling procedure.

The procedure for obtaining a mathematical model for a system can be summarized as follows:

1. Draw a **schematic diagram** of the system, and define the different variables.

2. Using physical laws, write equations for each component, combine them according to the system diagram, and obtain a **mathematical model**.

3. To verify the validity of the model, its **predicted performance**, obtained by solving the equations of the model, is **compared with experimental results**.

The question of the **validity** of any mathematical model can be answered only by experiments. If the experimental results deviate from the prediction to a great extent, the model must be modified. A new model is then derived and a new prediction compared with experimental results. The process is repeated until satisfactory agreement is obtained between the predictions and the experimental results.

1.1.3 Analysis and design of dynamic systems

This section briefly explains what is involved in the analysis and design of dynamic systems.

Design procedures

1. The engineer begins the design procedure knowing the **specifications** to be met and the dynamics of the components, the latter of which involve design parameters. The specification may be given in terms of both precise numerical values and vague qualitative descriptions. (Engineering specifications normally include statements on such factors as cost, reliability, space, weight, and ease of maintenance.) It is important to note that the specifications may be changed as the design progresses, for detailed analysis may reveal that certain requirements are impossible to meet.

2. Next, the engineer will apply any applicable **synthesis** techniques, as well as other methods, to build a **mathematical model** of the system. Once the design problem is formulated in terms of a model, the engineer carries out a mathematical design that yields a solution to the mathematical version of the design problem.

3. With the mathematical design completed, the engineer **simulates** the model on a computer to test the effects of various inputs and disturbances on the behavior of the resulting system. If the initial system configuration is not satisfactory, the system must be redesigned and the corresponding analysis completed. This process of design and analysis is repeated until a satisfactory system is found. To perform a **simulation and resulting analysis** of a mathematical model software packages like Matlab, Octave or Python are used. These software packages are optimized for certain mathematical model descriptions (transfer functions, Ordinary Differential Equation (ODE), block diagrams, state space models). All these representations are discussed in the next section.

4. Then a **prototype** physical system can be constructed. Note that the process of constructing a prototype is the reverse of mathematical modeling. The prototype is a physical system that represents the mathematical model with reasonable accuracy. Once the prototype has been built, the engineer tests it to see whether it is satisfactory. If it is, the design of the prototype is complete. If not, the prototype must be modified and retested. The process continues until a satisfactory prototype is obtained. The construction of a prototype is not included in this course but is a topic included in some master thesis's or post-graduate programs.

1.1.4 Mathematical model representation used in software

Two main groups of physical model representation can be identified: **non -linear** and **linear** physical model equations.
Both types are widely used in software and have different properties. In general the non-linear models are used to perform more accurate simulations whereas the linear models are more adequate to design a controller. Figure 1.1 shows the different model representations used in this course.

Non-linear representation

This is the most general mathematical representation of a system where the three different type are identified: Ordinary, Partial and Algebraic differential equations.

- An **Ordinary Differential Equation** is the most basic form where the solution is a function of **a single independent variable** and all equation are **explicitly** specified,

- The **Partial Differential Equation (PDE)** includes **partial derivatives** meaning that the solution is a **function of multiple variables**,

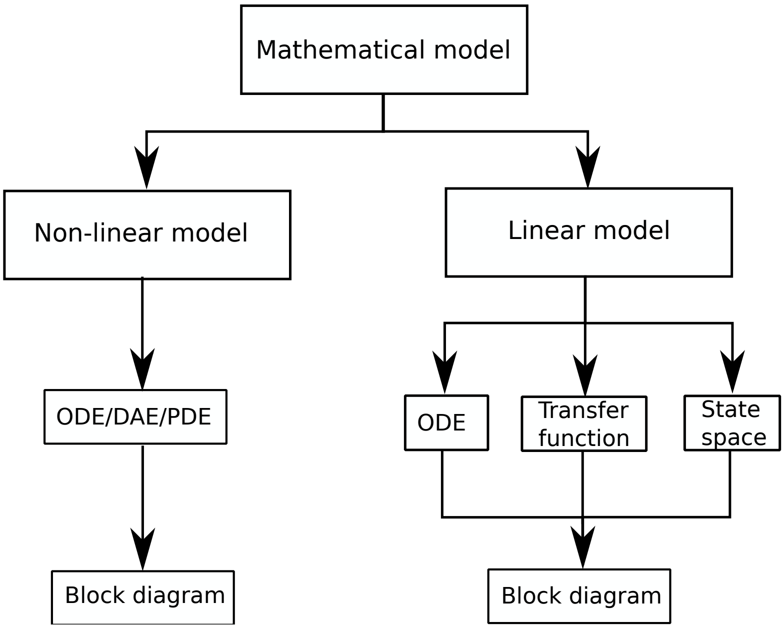

Figure 1.1: The different mathematical model representation discussed in this course

- A **Differential algebraic Equation (DAE)** explicitly defines differential equations making it impossible to isolate every derivative.

This course only considers ODE's because this is the most widely used method of non-linear simulation. More detailed information can be found in section 1.2

Linear model representation

Linear model representation have been used for a lot bigger because in the past most research was done on linear systems because of interesting features like **superposition** (makes it possible to specify a model of a subsystems and combining all subsystems to one big system by a simple operation like an addition or multiplication or the application of a **Laplace transform** to get a transfer function. More information regarding continues transfer functions, discrete transfer functions or state space models is given in sections 1.3, 1.4 and 1.5.

1.1.5 Examples of mechanical systems

Translational Motion

The cornerstone for obtaining a mathematical model, or the equations of motion, for any mechanical system is Newton's law,

$$\mathbf{F} = m\mathbf{a} \tag{1.5}$$

where $\mathbf{F}$[1] the vector sum of all forces applied to each body in a system, $\mathbf{a}$ is the vector acceleration of each body with respect to an inertial reference frame (that is, one that is neither accelerating nor rotating with respect to the stars); often called inertial acceleration, expressed in m/sec^2 and m is the body mass, expressed in kg.

Application of this law typically involves defining convenient coordinates to account for the bodys motion (position, velocity, and acceleration), determining the forces on the body using a freebody diagram, and then writing the equations of motion from Eq. 1.5. The procedure is simplest when the coordinates chosen express the position with respect to an inertial frame because in this case the accelerations needed for Newtons law are simply the second derivatives of the position coordinates.

Example 1.1.1. *A simplified Cruise Control Model*

***Question**: Write the equations of motion for the speed and forward motion of the car shown in Fig. 1.13 assuming that the engine imparts a force u as shown. For simplicity we assume that the rotational inertia of the wheels is negligible and that there is friction retarding the motion of the car that is proportional to the cars speed with a proportional constant, b.*

***Mathematical model**:*

The car model can be approximated for modeling purposes using the free-body diagram seen in Fig. 1.3, which defines coordinates, shows all forces acting on the body (heavy lines), and indicates the acceleration (dashed lines). The coordinate of the cars position, x, is the distance from the reference line shown and chosen so that positive is to the right. Note that in this case the inertial acceleration is simply the second derivative of x (that is, $a = \ddot{x}$) because the car position is measured with respect to an inertial reference. The equation of motion is found using

[1] Note that we use the convention of boldfacing the type to indicate that the quantity is a matrix or vector, possibly a vector function.

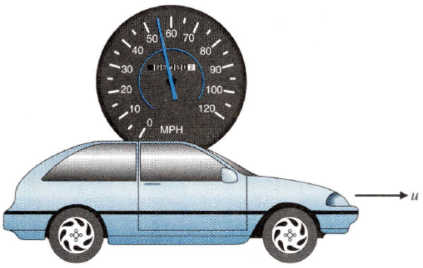

Figure 1.2: A car driven by a force u and counteracting a viscous friction force.

Eq. 1.5. The friction force acts opposite to the direction of motion; therefore it is drawn opposite the direction of positive motion and entered as a negative force in Eq. 1.5. The resulting equations are

$$u - b\dot{x} = m\ddot{x} \tag{1.6}$$

or

$$\ddot{x} + \frac{b}{m}\dot{x} = \frac{u}{m} \tag{1.7}$$

For the case of a cruise control where the variable of interest is the speed, $v(=\dot{x})$, the equation of motion becomes

$$\dot{v} + \frac{b}{m}v = \frac{u}{m} \tag{1.8}$$

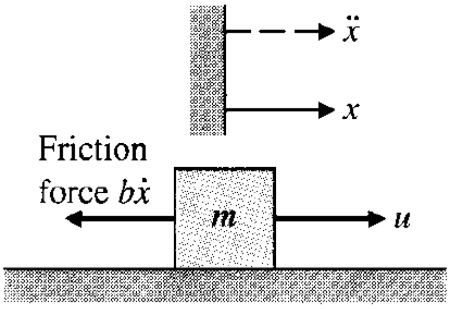

Figure 1.3: Free-body diagram for a force driven the car.

Example 1.1.2. *A quarter car suspension model*

Figure 1.4 shows a vehicle suspension system. Write the equations of motion for the vehicle and wheel motion assuming one-dimensional vertical motion of one quarter of the car mass above one wheel. A system consisting of one of the four wheel suspensions is usually refined to as a quarter-car model. Assume that the model is for a car with a mass of 1580 kg, including the four wheels, which have a mass of 20 kg each. By placing a known weight (an author) directly over a wheel and measuring the car's deflection, we find that $k_s = 130.000$ *N/m. Measuring the wheel's deflection for the same applied weight, we find that* $k_w \approx 1.000.000$ *N/m. By qualitatively observing the car's response as the author jumps off matches the damping factor* $\zeta = 0.7$ *curve, we conclude that* $b = \zeta * 2 * \sqrt{k_s * (1580/4 - 20)} \approx 9800$ *Nsec/m.*

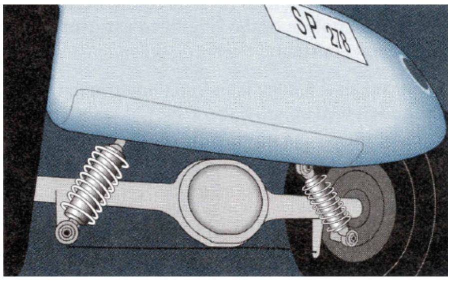

Figure 1.4: Car suspension system.

Mathematical model.

The system can be approximated by the simplified system shown in Fig. 1.5. The coordinates of the two masses, x and y, with the reference directions as shown, are the displacements of the masses from their equilibrium conditions. The equilibrium positions are offsets from the springs' unstretched positions because of the force of gravity. The shock absorber is represented in the schematic diagram by a dash-pot symbol with friction constant b. The magnitude of the force from the shock absorber is assumed to be proportional to the rate of change of the relative displacement of the two masses-that is, the force= $b(\dot{y} - \dot{x})$*. The force of gravity could be included in the free body diagram; however, its effect is to produce a constant offset of x and y. By defining x and y to be the distance from the equilibrium position, the need to include the gravity forces is eliminated.*

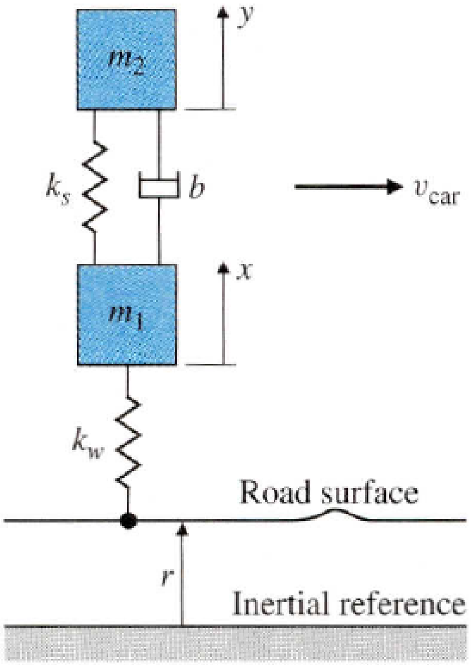

Figure 1.5: The quarter-car model

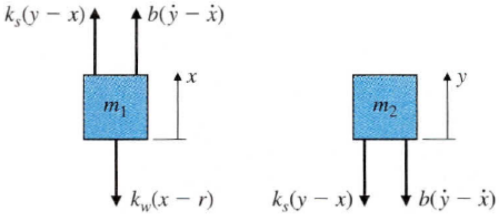

Figure 1.6: Free-body diagrams for a suspension system.

The force from the car suspension acts on both masses in proportion to their relative displacement with spring constant K_s. Figure 1.6 shows the free-body diagram of each mass.

The lower spring k_w represents the tire compressibility, for which there is insufficient damping (velocity-dependent force) to warrant including a dashpot in the model. The force from this spring is proportional to the distance the tire is compressed and the nominal equilibrium force would be that required to support m_1 and m_2 against gravity. By defining x to be the distance from equilibrium, a force will result if either the road surface has a bump ($r(t)$ changes from its equilibrium value of zero) or the wheel bounces (x changes). The motion of the simplified car over a bumpy road will result in a value of $r(t)$ that is not constant.

Applying Eq. 1.5 to each mass and noting that some forces on each mass are in the negative (down) direction yields the system of equations

$$b(\dot{y}-\dot{x})+k_s(y-x)-k_w(x-r) = m_1\ddot{x}, \tag{1.9}$$

$$-k_s(y-x)-b(\dot{y}-\dot{x}) = m_2\ddot{y}. \tag{1.10}$$

Some rearranging results in

$$\begin{aligned} \ddot{x} + \frac{b}{m_1}(\dot{x} - \dot{y}) + \frac{k_s}{m_1}(x - y) + \frac{k_w}{m_1}x &= \frac{k_w}{m_1}r && (1.11) \\ \ddot{y} + \frac{b}{m_2}(\dot{y} - \dot{x}) + \frac{k_s}{m_2}(y - x) &= 0 && (1.12) \end{aligned}$$

The most common source of error in writing equations for systems like these are sign errors. The method for keeping the signs straight in the preceding development entailed mentally picturing the displacement of the masses and drawing the resulting force in the direction that the displacement would produce. Once you have obtained the equations for a system, a check on the signs for systems that are obviously stable from physical reasoning can be quickly carried out. A stable system always has the same signs on similar variables. For this system, Eq. 1.11 shows that the signs on the x, $\dot{x}$-, and $\ddot{x}$-terms are all positive, as they must be for stability. Likewise, the signs on the y, $\dot{y}$, and $\ddot{y}$-terms are all positive in Eq. 1.12.

Example 1.1.3. *Dynamic model of a combustion engine cruise control*

Cruise control is the term used to describe a control system that regulates the speed of an automobile. Cruise control was commercially introduced in 1958 as an option on the Chrysler Imperial. The basic operation of a cruise controller is to sense the speed of the vehicle, compare this speed to a desired reference, and then accelerate or decelerate the car as required. The figure 1.7 shows a block diagram of this feedback system.

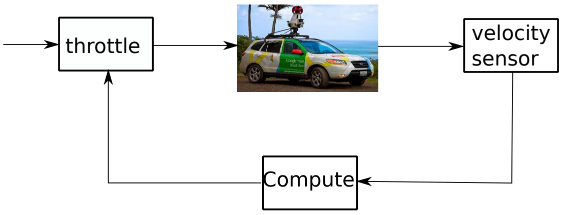

Figure 1.7: A block diagram of the cruise control systems based on velocity feedback.

To develop a mathematical model we start with a force balance for the car body. Let ν *be the speed of the car,* m *the total mass (including passengers),* F *the force generated by the contact of the wheels with the road, and* F_d *the disturbance force due to gravity, friction and aerodynamic drag. The equation of motion of the car is simply*

$$m\frac{d\nu}{dt} = F - F_d \tag{1.13}$$

The force F *is generated by the engine, whose torque is proportional to the rate of fuel injection, which is itself proportional to a control signal* $0 \leq u \leq 1$ *that controls the throttle position. The torque also depends on engine speed* ω*. A simple representation of the torque at full throttle is given by the torque curve*

$$T(\omega) = T_m\left(1 - \beta\left(\frac{\omega}{\omega_n} - 1\right)^2\right),$$

where the maximum torque T_m *is obtained at engine speed* ω_m*.*
Let n *be the gear ratio and* r *the wheel radius. The engine speed is related to the velocity through the expression*

$$\omega = \frac{n}{r}\nu = \alpha_n\nu,$$

and the driving force can be written as

$$F = \frac{nu}{r}T(\omega) = \alpha_n uT(\alpha_n, \nu) \tag{1.14}$$

Typical values of α_n for gears 1 through 5 are $\alpha_1 = 40$, $\alpha_2 = 25$, $\alpha_3 = 16$, $\alpha_4 = 12$ and $\alpha_5 = 10$. The inverse of α_n has a physical interpretation as the effective wheel radius. The figure to the right shows the torque as a function of vehicle speed. The figure 1.8 shows that the effect of the gear is to "flatten" the torque curve so that an almost full torque can be obtained almost over the whole speed range.

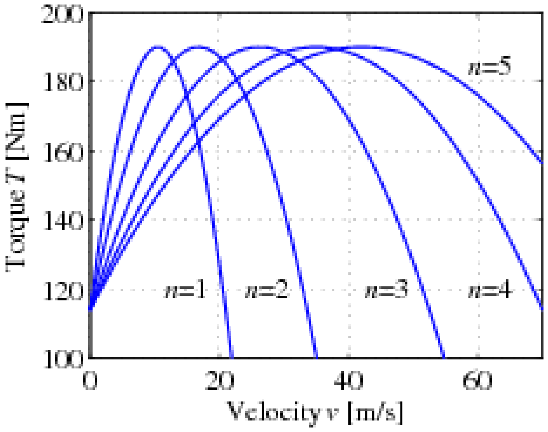

Figure 1.8: A torque-velocity curve of a motor as a function of the gear (n) ratio.

The disturbance force F_d has three major components: F_g, the forces due to gravity; F_r, the forces due to rolling friction; and F_a, the aerodynamic drag:

$$F_d = F_g + F_r + F_a. \tag{1.15}$$

Letting the slope of the road be θ, gravity gives the force $F_g = mg\sin\theta$, where $g = 9.8\ \frac{m}{s^2}$ is the gravitational constant. A simple model of rolling friction is

$$F_r = mgC_r sgn(\nu)$$

where C_r is the coefficient of rolling friction and $sgn(\nu)$ is the sign of ν or zero if $\nu = 0$. Finally, the aerodynamic drag is proportional to the square of the speed:

$$F_a = \frac{1}{2}\rho C_d A \nu^2, \tag{1.16}$$

where ρ is the density of air, C_d is the shape-dependent aerodynamic drag coefficient and A is the frontal area of the car.

1.1.6 Example of an electromechanical system

Example 1.1.4. *A DC motor model*

In this section, we obtain a mathematical model of a DC servomotor. To control the motion or speed of DC servomotors, we control the field current or armature current or we use a servodriver as a motor-driver combination. There are many different types of servodrivers. Most are designed to control the speed of DC servomotors, which improves the efficiency of operating servomotors. Here, however, we shall discuss only armature control of a DC servomotor and obtain its mathematical model in the form of a transfer function.

Armature control of DC servomotors. *Consider the armature-controlled DC servomotor shown in Figure 1.9, where the field current is held constant. In this system,*
R_a = *armature resistance,* [Ω]
L_a = *armature inductance, [H]*
i_a = *armature current, [A]*
i_f = *field current, [A]*
e_a = *applied armature voltage, [V]*
e_b = *back emf, [V]*
θ = *angular displacement of the motor shaft, [rad]*
T = *torque developed by the motor, [N-m]*
J = *moment of inertia of the motor and load referred to the motor shaft, [kgm^2]*
b = *viscous-friction coefficient of the motor and load referred to the motor shaft, [Nm/rad/s]*

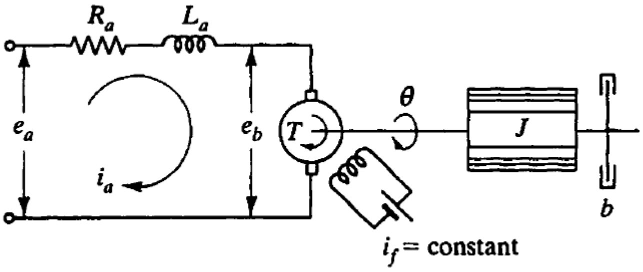

Figure 1.9: Armature-controlled DC servomotor.

The torque T developed by the motor is proportional to the product of the armature current i_a and the air gap flux ψ, which in turn is proportional to the field current i_f, or

$$T = K_f i_f$$

where K_f is a constant. The torque T can therefore be written as

$$T = K_f i_f K_1 i_a$$

where K_1 is the flux constant. For a constant field current, the flux becomes constant and the torque becomes directly proportional to the armature current, so

$$T = K_t i_a$$

where $K_t = K_f K_1$ is a motor-torque constant. Notice that if the sign of the current i_a is reversed, the sign of the torque T will be reversed, which will result in a reversal of the direction of rotor rotation.

When the armature is rotating, a voltage proportional to the product of the flux

and angular velocity is induced in the armature. For a constant flux, the induced voltage e_b is directly proportional to the angular velocity $\frac{d\theta}{dt}$, or

$$e_b = K_b \frac{d\theta}{dt} \tag{1.17}$$

where e_b is the back emf and K_b is a back-emf constant.

The speed of an armature-controlled DC servomotor is controlled by the armature voltage e_a. The differential equation for the armature circuit is

$$L_a \frac{di_a}{dt} + R_a i_a + e_b = e_a \tag{1.18}$$

The armature current produces the torque that is applied to the inertia and friction; hence,

$$J \frac{d^2\theta}{dt^2} + b \frac{d\theta}{dt} = T = K_t i_a \tag{1.19}$$

Assuming that all initial conditions are zero and taking the Laplace transforms of Equations 1.17, 1.18, and 1.19, we obtain the following equations:

$$\begin{aligned} E_b(s) &= K_b s \Theta(s) && (1.20) \\ E_a(s) &= (L_a s + R_a) I_a(s) + E_b(s) && (1.21) \\ T(s) &= K_t I_a(s) = (Js^2 + bs)\Theta(s) && (1.22) \end{aligned}$$

Considering $E_a(s)$ as the input and $\Theta(s)$ as the output and eliminating $I_a(s)$ and $E_b(s)$ from Equations 1.20, 1.21, and 1.22, we obtain the transfer function for the DC servomotor:

$$\frac{\Theta(s)}{E_a(s)} = \frac{K_t}{s[L_a J s^2 + (L_a b + R_a J)s + R_a b + K_t K_b]} \tag{1.23}$$

The inductance L_a in the armature circuit is usually small and may be neglected. If L_a is neglected, then the transfer function given by Equation 1.23 reduces to

$$\frac{\Theta(s)}{E_a(s)} = \frac{K_t}{s[R_aJs + R_ab + K_tK_b]} = \frac{\frac{K_t}{R_aJ}}{s(s + \frac{R_ab+K_tK_b}{R_aJ})} \tag{1.24}$$

Notice that the term $(R_ab + K_tK_b)/(R_aJ)$ in Equation corresponds to the damping term. Thus, the back emf increases the effective damping of the system. Equation may be rewritten as

$$\frac{\Theta(s)}{E_a(s)} = \frac{K_m}{s(T_ms + 1)} \tag{1.25}$$

where
$K_m = K_t/(R_ab + K_tK_b) =$ *motor gain constant*
$T_m = R_aJ/(R_ab + K_tK_b) =$ *motor time constant*
Equation 1.25 is the transfer function of the DC servomotor when the armature

voltage $e_a(t)$ is the input and the angular displacement $\theta(t)$ is the output. Since the transfer function involves the term 1/s, this system possesses an integrating property. (Notice that the time constant T_m of the motor becomes smaller as the resistance R_a is reduced and the moment of inertia J is made smaller.)

1.2 Continues-time systems: Time domain simulations

1.2.1 Introduction

This section explains how to perform a time domain simulation starting from a mathematical model consisting of Ordinary Differential Equations (ODE's) in section 1.2.2 or a block diagram in Matlab-Simulink in section 1.2.3 used in the examples in section 1.2.4 to perform the simulation .

1.2.2 ODE model

A large group of mathematical model described by a set of differential equation can be rewritten as a system of **Ordinary Differential Equations** (ODEs). These ODE's are well-known in literature and can be solved in software. To be able to apply an ODE the mathematical model should not contain partial derivatives and the model equations should be explicit meaning that you should be able to isolate the highest derivative in every equation. An example of a mathematical model equation that cannot be converted to an ODE is:

$$(\frac{d^2x(t)}{dt^2})^2 + 5\frac{dx(t)}{dt} - 4 = 0, \tag{1.26}$$

this type a equation can be solved by a Differential Algebraic Equation (DAE). This is however not a topic contained in this course.
An ordinary differential equation or ODE is an equation containing a function of one independent variable and its derivatives. The term "ordinary" is used in contrast with the term partial differential equation which may be with respect to more than one independent variable. In general an ODE is specified as:

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(t, \mathbf{x}) \tag{1.27}$$

where $\mathbf{x} = [x_1, \ldots, x_n]^T$ is a n dimensional vector and $\mathbf{F} = [F_1(t, \mathbf{x}), \ldots, F_m(t, \mathbf{x})]^T$ a vector of m functions. An example of a system of ODE's is:

- The Euler differential equation: $t^2\frac{d^2y(t)}{dt^2} + at\frac{dy(t)}{dt} + by(t) = 0$ or written in the ODE format of equation 1.26 this becomes

$$\begin{aligned} \frac{dy(t)}{dt} &= \frac{dy(t)}{dt} && (1.28) \\ \frac{d^2y(t)}{dt^2} &= \frac{1}{t^2}\left(-at\frac{dy(t)}{dt} - by(t)\right) && (1.29) \end{aligned}$$

  with $\mathbf{x} = [y(t), \frac{dy(t)}{dt}]$ and $F_1(t, \mathbf{x}) = \frac{dy(t)}{dt}$, $F_2(t, \mathbf{x}) = \frac{1}{t^2}\left(-at\frac{dy(t)}{dt} - by(t)\right)$.

1.2.3 Block diagram or Simulink model

Block diagrams of dynamic systems.

A block diagram of a dynamic system is a **pictorial representation** of the functions performed by each component of the system and of the flow of signals within the system. Such a diagram depicts the interrelationships that exist among the various components. Differing from a purely abstract mathematical representation, a block diagram has the advantage of indicating the **signal flows** of the actual system more realistically.

In a block diagram, all system variables are linked to each other through functional blocks. The **functional block**, or simply block, is a symbol for the mathematical operation on the input signal to the block that produces the output. The transfer functions of the components are usually entered in the corresponding blocks, which are connected by arrows to indicate the direction of the flow of signals. Note that a signal can pass only in the direction of the arrows. Thus, a block diagram of a dynamic system explicitly shows a **unilateral property**.

Figure 1.10a shows an element of a block diagram. The **arrowhead** pointing toward the block indicates the input to the block, and the arrowhead leading away from the block represents the output of the block. As mentioned, such arrows represent signals. Note that the dimension of the output signal from a block is the dimension of the input signal multiplied by the dimension of the transfer function in the block.

The **advantages** of the block diagram representation of a system lie in the fact that it is **easy to form** the overall block diagram for the entire system merely by connecting the blocks of the components according to the signal flow and that it is **possible to evaluate the contribution of each component** to the overall performance of the system.

In general, the **functional operation of a system can be visualized more readily** by examining a block diagram of the system than by examining the physical system itself. A block diagram contains information concerning dynamic behavior, but **it does not include any information about the physical construction** of the system. Consequently, many dissimilar and unrelated systems can be represented by the same block diagram. Note that in a block diagram the main source of energy is not explicitly shown and that the block diagram of a given system is not unique. A number of different block diagrams can be drawn for a system, depending on the point of view of the analysis. (See Example 1.10)

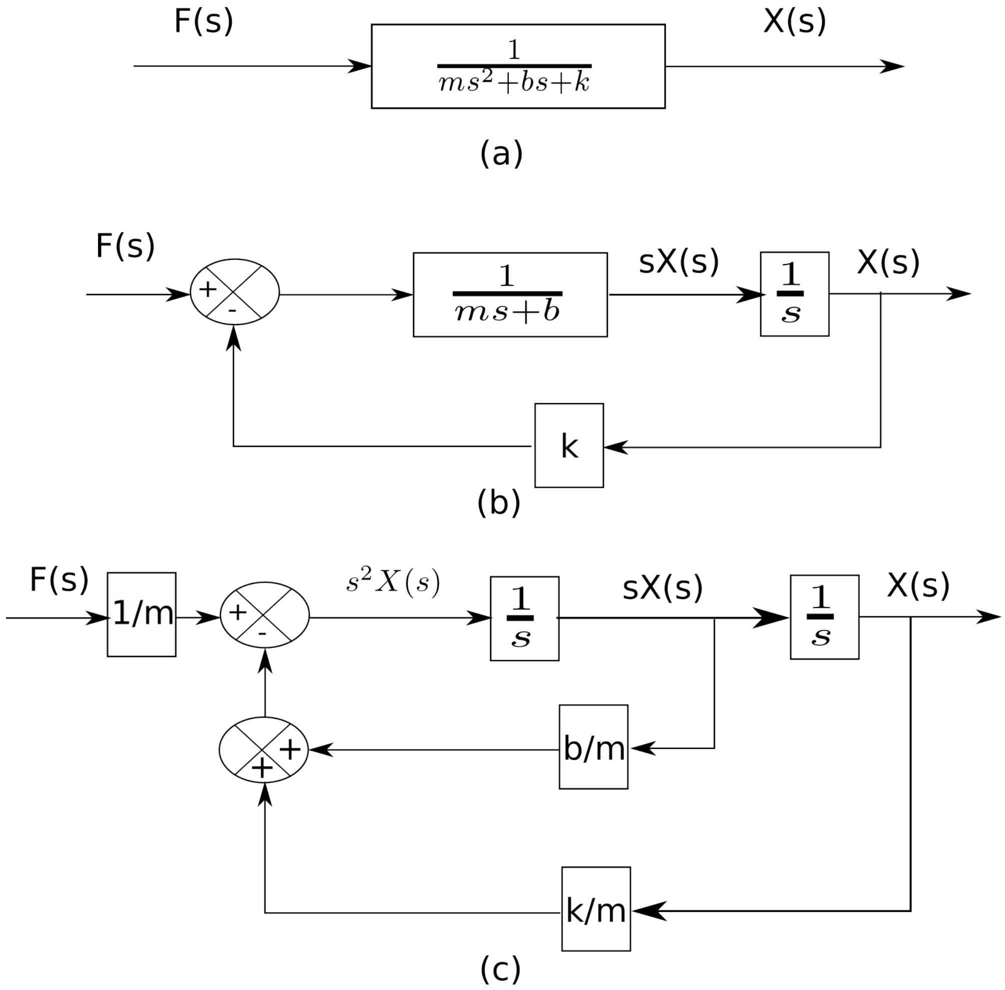

Figure 1.10: A block diagram for the system based on Equation 1.42 is shown in Figure 1.10(c). Figures 1.10(a), (b), and (c) are thus block diagrams for the same system that is shown in Figure 1.28. (Many different block diagrams are possible for any given system.)

Summation point. Figure 1.11 shows a circle with a cross, the symbol that stands for a summing operation. The plus or minus sign at each arrowhead indicates whether the associated signal is to be added or subtracted. It is important that the quantities being added or subtracted have the same dimensions and the same units.

Branch point. A branch point is a point from which the signal from a block goes concurrently to other blocks or summing points.

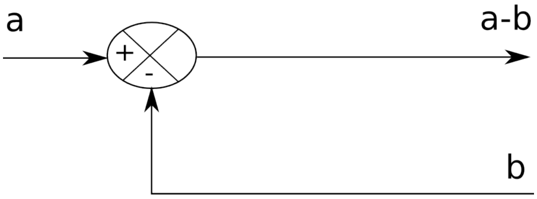

Figure 1.11: A summation point.

Block diagram of a closed-loop system. Figure 1.12 is a block diagram of a closed-loop system. The output C(s) is fed back to the summing point, where it is compared with the input R(s). The closed-loop nature of the system is indicated clearly by the figure. The output C(s) of the block is obtained by multiplying the transfer function G(s) by the input to the block, E(s).

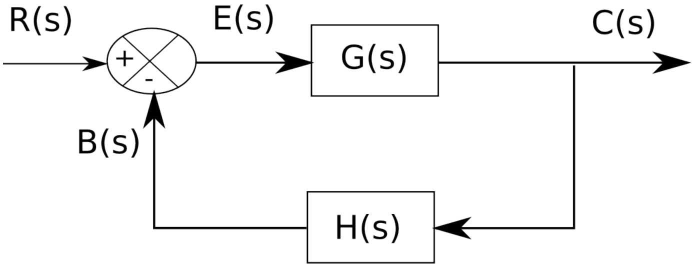

Figure 1.12: Block diagram of a closed loop system.

Most linear and non-linear systems can be represented by a block diagram consisting of blocks, summing points, and branch points. When the output is fed back to the summing point for comparison with the input, it is necessary to convert the form of the output signal to that of the input signal. This conversion is accomplished by the feedback element whose transfer function is $H(s)$, as shown in Figure 1.12.

Example 1.2.1. *Block diagram*
Consider a mass-spring-damper system with mass m, spring constant k and damping factor b. The transfer function of this system (see Example 1.3.1) is

$$\frac{X(s)}{F(s)} = \frac{1}{ms^2 + bs + k} \tag{1.30}$$

A block diagram representation of the system is shown in Figure 1.10(a). Notice that Equation 1.30 can be written as

$$(ms^2 + bs + k)X(s) = F(s) \tag{1.31}$$

Rewriting the latter equation as

$$(ms^2 + bs)X(s) = F(s) - kX(s) \tag{1.32}$$

we can obtain a different block diagram for the same system, as shown in Figure 1.10(b). Equation 1.30 can also be rewritten as

$$F(s) - [kX(s) + bsX(s)] = ms^2X(s) \tag{1.33}$$

or

$$\frac{1}{m}F(s) - \frac{k}{m}X(s) - \frac{b}{m}sX(s) = s^2X(s) \tag{1.34}$$

A block diagram for the system based on Equation 1.34 is shown in 1.10(c). Figures 1.10(a), (b), and (c) are thus block diagrams for the same system-that are shown in Figure 1.28. (Many different block diagrams are possible for any given system.)

1.2.4 Examples of ODE-models and Simulink

Example 1.2.2. *A simplified Cruise Control Model*

Question*: Simulate the step response of the cruise control model equations derived in example 1.1.1 using an ODE solver in Matlab, Python and Matlab-simulink.*

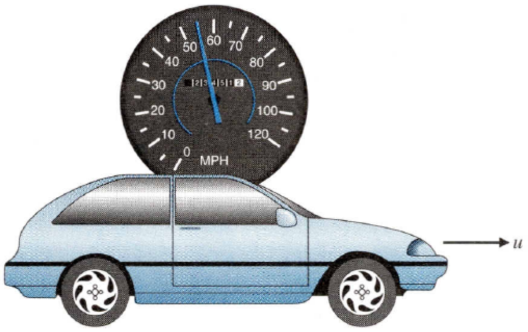

Figure 1.13: A car driven by a force u and counteracting a viscous friction force.

Solution*: Simulation velocity response to a force step input:*

Two type of continuous time simulations are discussed: a numerical ODE solver simulation in Matlab and a graphical simulation in Matlab-Simulink.

1. *Numerical ODE-solver simulation in Matlab:*

   Matlab is a ***numerical*** *computing environment. This implies that every differential equation is solved by performing a numerical approximation instead of specifying a symbolic or algebraic solution (like in Maple). This way Matlab is able to solve a set of non-linear differential equations that are impossible to solve with a symbolic solver.*
   A widely used ***numerical differential equation solver*** *is 'ODE45'. It solves non-stiff differential equations, using a medium order method. We will explain this later. The following command is used to run the solver in Matlab:*

   ```
   [TOUT,YOUT] = ode45('ODEFUN',TSPAN,Y0) with TSPAN = [T0 TFINAL].
   ```

   It integrates a system of differential equations $\dot{y} = f(t, y)$ from time $T0$ to $TFINAL$ with initial conditions $Y0$. ODEFUN is a function handle that specifies the system of differential equations $\dot{y} = f(t, y)$.

   For the cruise control example with a mass m=200 kg and friction coefficient b=5 Nm/s the function handle for a step input u=1 for $t > 0$ equals:

   ```
   function dy=ODEFUN(t,y)
     m=200;
     b=5;
     u=1;
     dy = -b/m*y + u/m;
   ```

   the ";" symbol indicates that the result of the value assignment will not be shown. You need to ***save this code in file called ODEFUN.m in your current working directory*** *to make it usable for your numerical solver in Matlab. We assume a simulation time of 500 seconds (TSTART=0 and TFINAL=500) and an initial velocity Y0=0 m/s, the ODE45 solver can be started by specifying:*

   ```
   TSTART=0
   TFINAL=500
   TSPAN=[TSTART,TFINAL]
   Y0=[0]
   [TOUT,YOUT] =ode45('ODEFUN',TSPAN,Y0);
   plot(TOUT,YOUT)
   xlabel('time')
   ylabel('Car velocity')
   ```

Fig. 1.14 is generated by the command plot$(TOUT, YOUT)$*, with* $TOUT$ *the x-values and* $YOUT$ *the y-values of all the data points simulated. It visualizes the time evolution of the car velocity when a unit step force is imposed.*

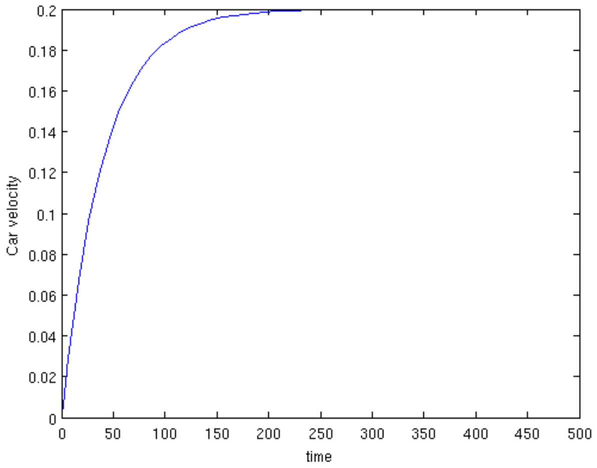

Figure 1.14: A plot of the solution for the ODE45 solver given the simplified cruise control model equation of example 1.1.1 and an initial velocity of 0 m/s.

2. *Graphical simulation using Matlab-Simulink:*

   Simulink, developed by MathWorks, is a data flow graphical programming language tool for modeling, simulating and analyzing multi-domain dynamic systems. Figure 1.15 shows a Simulink block diagram for our cruise control example. We identify a step input at the left and a summation that sums up u/m *and* $-(b/m)v$*. From Equation 1.8 we can derive that this sum equals to*

$$\dot{v} = \frac{u}{m} - \frac{b}{m}v.$$

   By integrating $\dot{v}$ *we can obtain the velocity* v *that we can use in our summator. This Simulink model is saved using the name Cruise_control1.mdl.*

   This Simulink block diagram uses parameters for the mass (m) and the friction coefficient (b). The corresponding numerical values need to be specified in Matlab in order to run the model. The Matlab script to set these parameters and run the simulation for Tsim= 500 *seconds is given here*

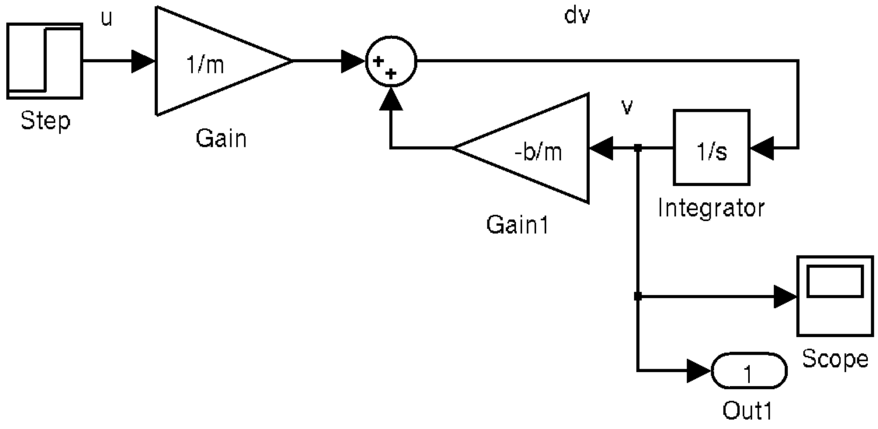

Figure 1.15: A Matlab-Simulink block diagram of the cruise control car model with a scope to monitor the car velocity and export block (Out1) for the car velocity to make the data available in the Matlab console.

```
m=200
b=5
Tsim=500
sim('Cruise_control1',Tsim)
plot(tout,yout)
xlabel('time')
ylabel('Car velocity')
```

at the end figure 1.14 is again generated showing the exported data (out1) that equals the velocity of the car as a function of time (tout). The result is identical to the numerical simulation using the ODE45 solver.

Example 1.2.3. *A quarter car model*
Question: *Simulate the step response of the quarter car model derived in example 1.1.2 using an ODE solver in Matlab, Python and Matlab-simulink.*

Solution: Simulation response to step input r(t):

1. *Numerical simulation in Matlab:*
   Because the numerical ODE solvers can only handle first order differential equations of the form $\dot{\mathbf{x}} = F(\mathbf{x}, t)$, however our current model in equations 1.11, 1.12 consists of a set of second order differential equations. In order to fit the ODE solver every higher order differential equation can be

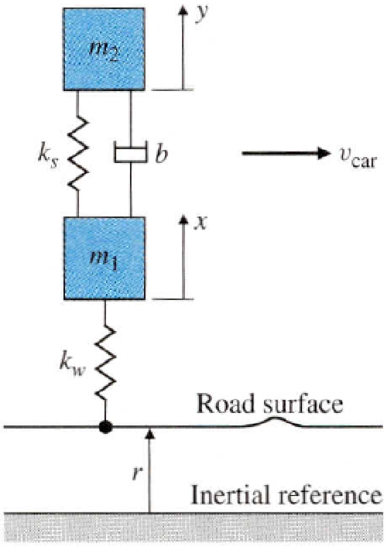

Figure 1.16: The quarter-car model

rewritten as a set of second order equations. This is accomplishes by introducing the new variables

$$\begin{aligned} v_x &= \dot{x} & (1.35)\\ v_y &= \dot{y} & (1.36) \end{aligned}$$

and replacing them in the derivatives of equations 1.11, 1.12:

$$\begin{aligned} \dot{v}_x &= \frac{k_w}{m_1}r - \frac{b}{m_1}(v_x - v_y) - \frac{k_s}{m_1}(x-y) - \frac{k_w}{m_1}x & (1.37)\\ \dot{v}_y &= -\frac{b}{m_2}(v_y - v_x) - \frac{k_s}{m_2}(y-x) & (1.38) \end{aligned}$$

The new vector with the variables then becomes:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \text{ and } \dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v}_x \\ \dot{v}_y \end{bmatrix}. \qquad (1.39)$$

For the quarter car suspension example with a mass $m_1 = 20kg$ and $m_2 = 1580/4 = 375$ kg the function handle for a step input u=1 for $t > 0$ becomes:

```
function dX=susp(t,X)
  m1=20;
  m2=375;
  b=9800;
  kw=1000000;
  ks=130000;
  u=1;
  ddx=-b/m1*[X(3)-X(4)]-ks/m1*[X(1)-X(2)]-kw/m1*X(1)+kw/m1*u;
  ddy= -b/m2*[X(4)-X(3)]-ks/m2*[X(2)-X(1)];
  dX=[X(3);X(4);ddx;ddy];
```

You need to ***save this code in file called susp.m in your current working directory*** *to make it usable for your numerical solver in Matlab. We assume a simulation time of 5 second (TSTART=0 and TFINAL=5) and an initial velocity Y0=0 m/s, the ODE45 solver can be started by specifying:*

```
TSPAN=[0,5]
Y0=[0;0;0;0]
[TOUT,YOUT] =ode45('susp',TSPAN,Y0);
plot(TOUT,YOUT(:,3))
xlabel('time')
ylabel('Velocity of mass 1')
```

at the end fig. 1.17 is generated, showing the time evolution of the car velocity when a unit step force is imposed.

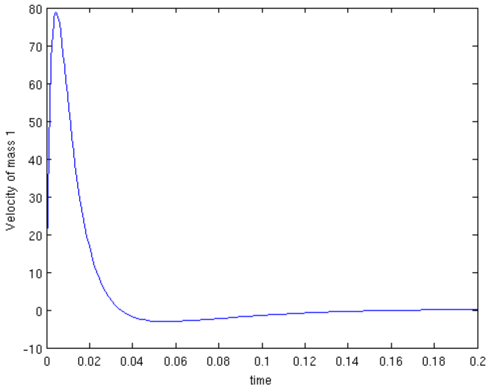

Figure 1.17: A plot of the step response velocity of mass 1 as a function of time obtained by the ode45 solver.

2. *Graphical simulation using Matlab-Simulink:*

Figure 1.19 shows a Simulink block diagram for our quarter car control example. We identify a step input at the left and a summator that sums up the right hand term of equation 1.39 and 1.38 . This model is saved using the name suspension.mdl. This Simulink block diagram uses parameters for the masses (m_1, m_2), the friction coefficient (b) and the spring constants (k_w, k_s). The corresponding numerical values need to be specified in Matlab in order to run the model. The Matlab script to set these parameters, set the initial velocities to zero ($Y0$) and run the simulation for 5 seconds is given here

```
m1=20;
m2=375;
b=9800;
kw=1000000;
ks=130000;
u=1;
sim('suspension',0.2)
plot(tout,yout(:,1))
xlabel('time')
ylabel('Velocity of mass 1')
```

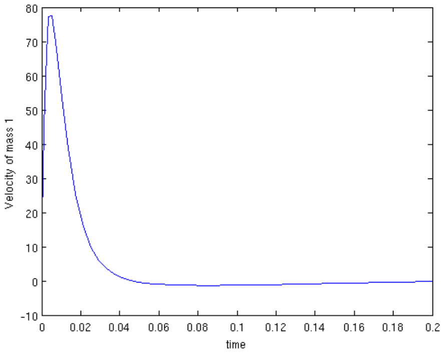

Figure 1.18: A plot of the numerical approximated velocity of mass 1 as a function of time obtained by the Matlab-Simulink model of fig. 1.19.

at the end a fig. 1.18 is generated showing the first column of the exported data (out1) that equals the velocity of mass 1 (v_x) as a function of time (tout).

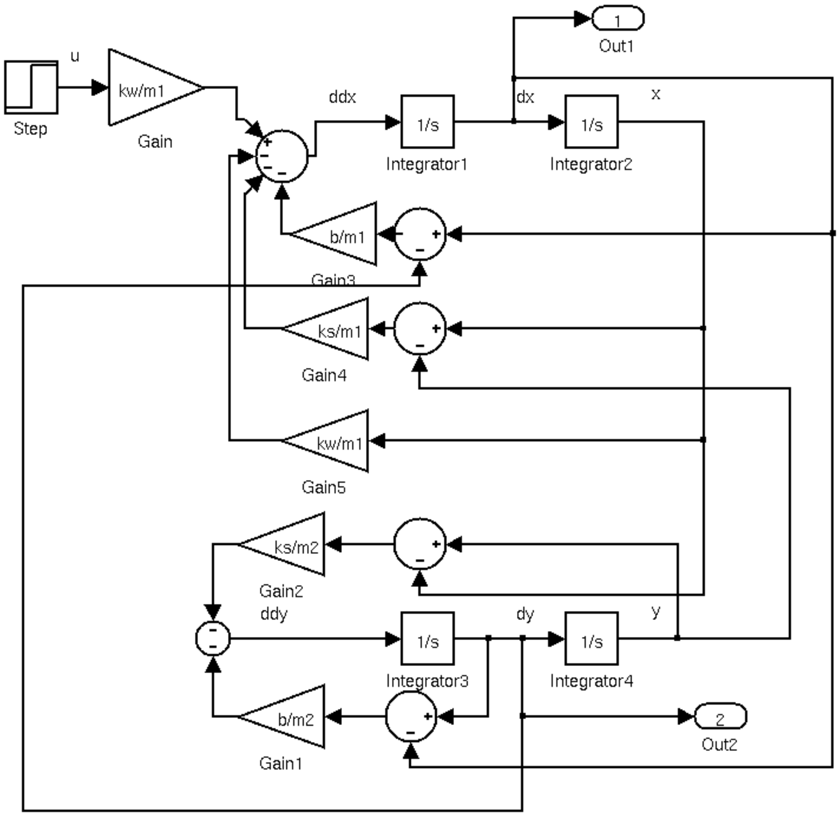

Figure 1.19: A Matlab-Simulink block diagram of the quarter car suspension modell with two block (Out1, Out2) the export of the velocity of mass 1 and mass 2 to make the data available in the Matlab console.

Example 1.2.4. *Combustion engine cruise control*

Question: *Simulate the step response of the combustion engine cruise control derived in example 1.13 using an ODE solver in Matlab, Python and Matlab-Simulink.*

Solution: Simulation response to step input on the throttle control signal u(t):

1. *Numerical simulation in Matlab:*

The mathematical model of equation 1.13 is already written as $\dot{\mathbf{x}} = F(\mathbf{x}, t)$.

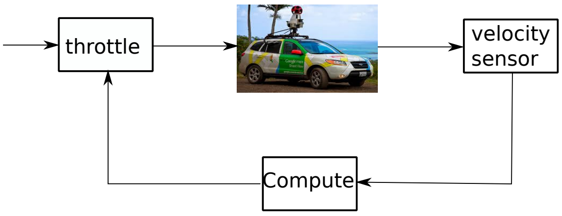

Figure 1.20: A block diagram of the cruise control systems based on velocity feedback.

If we take $\mathbf{x} = [x \quad v]$ *we get*

$$\begin{aligned}\frac{dx}{dt} &= v \\ \frac{dv}{dt} &= \frac{1}{m}\left[F - F_d\right]\end{aligned}$$

where the input force F *is specified by equation 1.14 and the disturbance force* F_d *by equation 1.15.*

A typical parameters values for the disturbance force are: rolling friction is C_r *= 0.01,* ρ *= 1.3* kg/m^3*,* C_d *= 0.32 and* A *= 2.4* m^2*. Typical parameters for the generated force by the combustion engine are* $T_m = 190Nm$*,* $\omega_m = 420$ *rad/s (about 4000 RPM) and* $\beta = 0.4$*. The function handle becomes:*

```
function [dy] = cruisedyn(t,y, u, gear, theta, m)
dy=[0;0];
v=y(2);

% Parameters for defining the system dynamics
alpha = [40, 25, 16, 12, 10]; % gear ratios
Tm = 190; % engine torque constant, Nm
wm = 420; % peak torque rate, rad/sec
beta = 0.4; % torque coefficient
Cr = 0.01; % coefficient of rolling friction
rho = 1.3; % density of air, kg/m^3
Cd = 0.32; % drag coefficient
A = 2.4; % car area, m^2
g = 9.8; % gravitational constant


% Saturate the input to the range of 0 to 1
```

```
u = min(u, 1);  u = max(0, u);

% Compute the torque produced by the engine, Tm
omega = alpha(gear) * v ;% engine speed

torque = u * Tm * ( 1 - beta * (omega/wm - 1)^2 );
F = alpha(gear) * torque;

dy(1) = y(2);

% Compute the external forces on the vehicle
Fr = m * g * Cr; % Rolling friction
Fa = 0.5 * rho * Cd * A * v^2; % Aerodynamic drag
Fg = m * g * sin(theta); % Road slope force
Fd = Fr + Fa + Fg; % total deceleration

% Now compute the acceleration
dy(2) = (F - Fd) / m
```

You need to ***save this code in file called cruisedyn.m in your current working directory*** *to make it usable for your numerical solver in Matlab. We assume a simulation time of 100 seconds (TSTART=0 and TFINAL=120) and a zero initial position and velocity* $Y0 = [0m \quad 0m/s]$*, the ODE45 solver can be started by execution the script ODE_cruise_control:*

```
close all %close all figures
clear all % clear all variables
global m u theta gear  % make these variables visible inside
%all Matlab functions

m=1000;
gear=4;

%%% The throttle input
u=1; %full throttle

%%% The steep angle
theta=4*pi/180;
TSPAN=[0:0.01:120]; %a vector of timesteps from 0 to 120 in steps of 0.01
Y0=[0;0];
[TOUT,YOUT] =ode45(@(t,y) cruisedyn(t,y, u, gear, theta, m),TSPAN,Y0);
```

```
plot(TSPAN,YOUT(:,2)) %plot the velocity YOUT(:,2) as a function of time
```

at the end fig. 1.21 is generated, showing the time evolution of the car velocity when a unit step force is imposed.

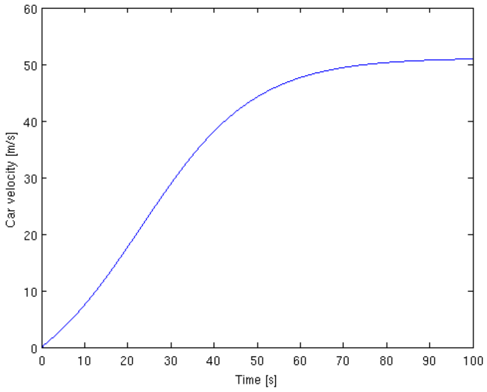

Figure 1.21: A plot of the step response velocity of the combustion cruise control as a function of time obtained by the ode45 solver.

2. *Graphical simulation using Matlab-Simulink:*

Figure 1.22 shows the Simulink block diagram (called “cruise_control_non_linear”) for the combustion engine cruise control example. The chosen input hill function generator only starts at 60 seconds by imposing a ramp function in Matlab-Simulink. This Simulink block diagram uses parameters for the disturbance force (A, Cd, rho, Cr) and the generated motor force ($Tm, wm, beta$). The corresponding numerical values need to be specified in Matlab in order to run the model. The Matlab script (called “Cruise_simulink”) to set these parameters, set the initial velocities to zero ($Y0$) and run the simulation for 100 seconds is chosen here

```
% Parameters for defining the system dynamics
alpha = [40, 25, 16, 12, 10]; % gear ratios
Tm = 190; % engine torque constant, Nm
wm = 420; % peak torque rate, rad/sec
beta = 0.4; % torque coefficient
Cr = 0.01; % coefficient of rolling friction
rho = 1.3; % density of air, kg/m^3
```

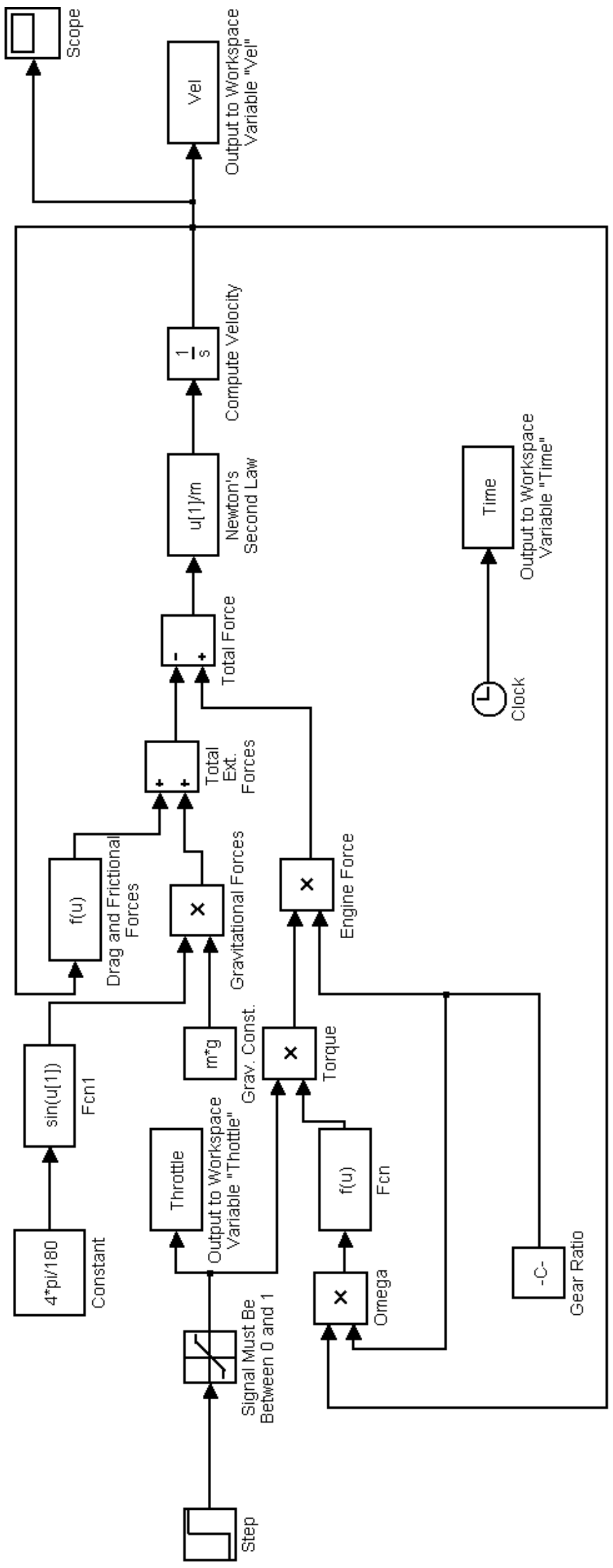

Figure 1.22: A Matlab-Simulink block diagram of the quarter car suspension modell with two block (Out1, Out2) the export of the velocity of mass 1 and mass 2 to make the data available in the Matlab console.

```
Cd = 0.32; % drag coefficient
A = 2.4; % car area, m^2
g = 9.8; % gravitational constant

m=1000; %% mass in kg
gear=4;
T=120; %simulation time

Simout = sim('cruise_control_non_linear',T);

plot(Time,Vel)
xlabel('Time [s]')
ylabel('Car velocity [m/s]')
```

at the end a fig. 1.23 is generated showing the exported velocity of the car (vel) and time vector (Time).

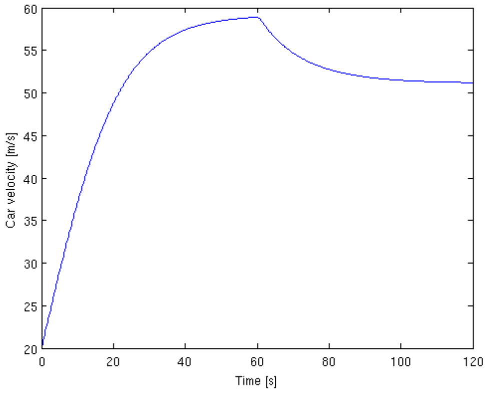

Figure 1.23: A plot of the numerical approximated velocity the combustion driven car as a function of time obtained by the Matlab-Simulink model of fig. 1.22.

Example 1.2.5. *DC motor*

Question*: Simulate the step response of the DC motor model derived in example 1.1.4 using an ODE solver in Matlab, Python and Matlab-Simulink.*

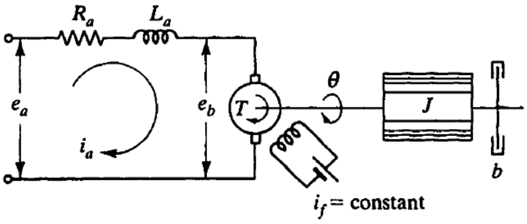

Figure 1.24: An armature(e_a)-controlled DC servomotor.

Solution: Simulation response to step input on the armature voltage *$e_a(t)$:*

1. *Numerical simulation in Matlab:*
   The mathematical model of equation 1.13 is already written as $\dot{\mathbf{x}} = F(\mathbf{x}, t)$. If we take $\mathbf{x} = [\theta \quad \omega \quad i_a]$ we get

$$\begin{aligned}\frac{d\theta}{dt} &= \omega \\ \frac{d\omega}{dt} &= \frac{1}{J}\left(K_t i_a - b\omega\right) \\ \frac{di_a}{dt} &= \frac{1}{L_a}\left(e_a - R_a i_a - K_b\omega\right)\end{aligned}$$

   where the derivative of the armature current ($\frac{di_a}{dt}$) is specified by 1.18 and the angular acceleration ($\frac{d\omega}{dt} = \frac{d^2\theta}{dt^2}$) is specified by equation 1.19.

   A typical parameters values for the DC motor are: motor torque constant K_t = 0.01 [Nm/A], back EMF constant K_b=0.1 [V/(rad/s), armature resistance R_a = 1 [Ω], armature inductance L_a = 0.01 [H] and the motor inertia J= 1 kgm^2. The function handle then becomes:

```
function [dy] = dcmotor(t,y, u)
% y=[theta omega I]
Kt=0.1;
Ra=1;
```

```
La=1e-3;
J=1;
Kb=0.1;

ea=u;
Td=0;

dy=zeros(3,1);
dy(1)=y(2);
dy(2)=Kt*y(3)-Td;
dy(3)= 1/La*(ea-y(3)*Ra-Kb*y(2));
```

You need to ***save this code in file called dcmotor.m in your current working directory*** *to make it usable for your numerical solver in Matlab. We assume a simulation time of 100 seconds (TSTART=0 and TFINAL=10) and a zero initial angular position, velocity, armature current* $Y0 = [0[m] \;\; 0[m/s] \;\; 0[A]]$*. The ODE45 solver can be started by execution the script ODE_dcmotor:*

```
close all %close all figures
clear all % clear all variables

TSPAN=[0:0.01:10];
Y0=[0;0;0]; %[theta, omega,Im]
u=24;

[TOUT,YOUT] =ode45(@(t,y) dcmotor(t,y, u),TSPAN,Y0);

s=1000;
plot(TSPAN(1:s),YOUT(1:s,1))
xlabel('time [s]')
ylabel('Position [rad]')
```

at the end fig. 1.25 is generated, showing the time evolution of the angular position of the DC motor when a unit step voltage is imposed on the armature voltage e_a*.*

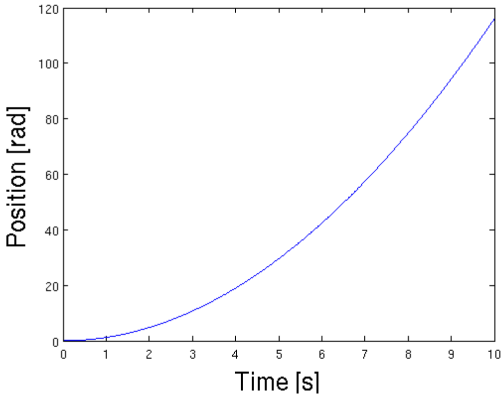

Figure 1.25: A plot of the step response of the angular position as a function of time obtained by the ode45 solver.

2. *Graphical simulation using Matlab-Simulink:*

Figure 1.26 shows the Simulink block diagram (called "DC_motorl_simulink_model") for the DC motor example. This Simulink block diagram uses following parameters K_t, K_b, R_a, L_a and J. The corresponding numerical values need to be specified in Matlab in order to run the model. The Matlab script (called "DC_motor_simulink") to set these parameters, set the initial position/velocities to zero in the integrators and run the simulation for 10 seconds with a disturbance torques equal to zero and an armature voltage of 24 [V] is given here

```
Kt=0.1;
Ra=1;
La=1e-3;
J=1;
Kb=0.1;

ea=24;
Td=0;

Simout = sim('DC_motor_simulink_model',10);

plot(Time,theta)
xlabel('Time [s]','FontSize', 20)
```

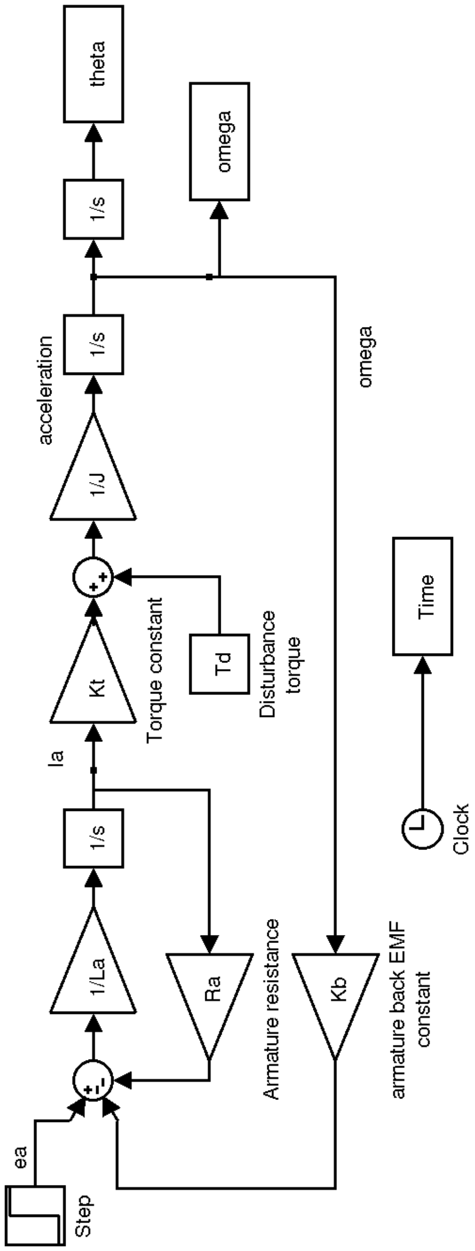

Figure 1.26: A Matlab-Simulink block diagram of a armature controlled DC motor with two output blocks the export of the angular position (theta) and the angular velocity (omega) to make them available as a vector in the Matlab console.

```
ylabel('Motor position [rad]','FontSize', 20)
```

at the end figure 1.27 is generated showing the exported position of the motor (theta) and time vector (Time).

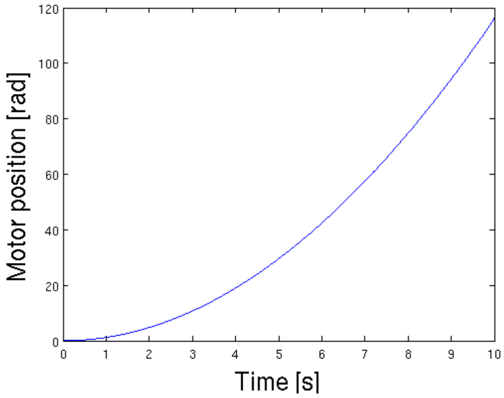

Figure 1.27: A plot of the numerical approximated motor position as a function of time with armature input of 24 Volt obtained by the Matlab-Simulink model of fig. 1.26.

1.3 Continuous-time systems: Transfer functions

1.3.1 Introduction

In this section, we present the transfer-function approach to modeling and analyzing dynamic systems. We first describe the concept of transfer functions (Section 1.3.2) and then we use linearization to approximate every possible non-linear model to a transfer function model (Section 1.3.3). Since software plays an important role in obtaining computational solutions of transient response problems, we present a detailed introduction to writing software programs to obtain response curves for time-domain inputs such as the step, impulse, ramp, and others (Section 1.3.4). Finally we illustrate all these concept using various examples in Section 1.3.5.

1.3.2 Transfer Function

The transfer function of a linear, time-invariant differential-equation system is defined as the ratio of the Laplace transform of the output (response function) to the Laplace transform of the input (driving function) **under the assumption that all initial conditions are zero**.

Consider the **linear time-invariant system** defined by the differential equation:

$$a_0\frac{d^n y}{dt^n} + a_1\frac{d^{n-1}y}{dt^{n-1}} + \ldots + a_{n-1}\frac{dy}{dt} + a_n y = b_0\frac{d^m x}{dt^m} + \ldots + b_{m-1}\frac{dx}{dt} + b_m x \qquad (m \le n) \tag{1.40}$$

where y is the system output, x is the input signal. The transfer function of this system is the ratio of the Laplace-transformed output to the Laplace-transformed input when all initial conditions are zero, or

$$\begin{aligned}\text{Transfer function} &= H(s) = \left|\frac{\mathcal{L}\{output\}}{\mathcal{L}\{input\}}\right|_{zero\ initial\ conditions} && (1.41)\\ &= \frac{Y(s)}{X(s)} = \frac{b_0 s^m + b_1 s^{m-1} + \ldots + b_{m-1}s + b_m}{a_0 s^n + a_1 s^{n-1} + \ldots + a_{n-1}s + a_n} && (1.42)\end{aligned}$$

By using the concept of a transfer function, it is possible to represent system dynamics by algebraic equations in s. If the highest power of s in the denominator the transfer function is equal to n, the system is called an **n th-order system**.

Important remarks regarding transfer functions. The applicability of the concept of the transfer function is **limited to linear, time-invariant differential equation systems**. Still, the transfer function approach is used extensively in the analysis and design of such systems. The following list gives some important comments concerning the transfer function of a system described by a linear, time-invariant differential equation:

1. The transfer function of a system is a mathematical model of that system, in that it is an operational method of expressing the **differential equation** that relates the output variable to the input variable.

2. The transfer function is a **property of a system** itself, unrelated to the magnitude and nature of the input or driving function.

3. The transfer function includes the units necessary to relate the input to the output; however, it does **not provide any information concerning the physical structure of the system**. (The transfer functions of many physically different systems can be identical.)

4. If the transfer function of a system is known, the output or **response can be studied** for various forms of inputs with a view toward understanding the nature of the system.

5. If the transfer function of a system is unknown, it may be **established experimentally** by introducing known inputs and studying the output of the system. Once established, a transfer function gives a full description of the dynamic characteristics of the system, as distinct from its physical description.

Example 1.3.1. *Consider the mechanical system shown in Figure 1.28. The displacement x(t) of the mass m is measured from the equilibrium position. In this system, the external force f(t) is the input and x(t) is the output. The equation of motion for the system is*

$$m\ddot{x}(t) + b\dot{x}(t) + kx(t) = f(t) \tag{1.43}$$

Taking the Laplace transform of both sides of this equation and assuming that all initial conditions are zero yields

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

where $X(s) = \mathcal{L}[x(t)]$ and $F(s) = \mathcal{L}[f(t)]$. Based on equation 1.42, the transfer function for the system becomes

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

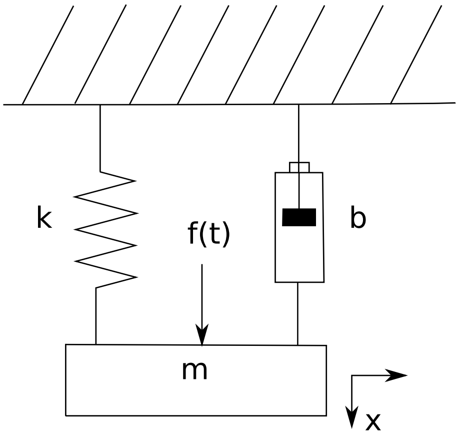

Figure 1.28: Mechanical mass-spring-damper system.

1.3.3 Linear approximation of physical systems

A large majority of physical systems are only linear within a certain range of his variables. Almost all physical systems become non-linear as the variables are increased without limitations. For example, the mass-spring-damper system of Figure 1.28 is linear as described by equation 1.43 as long as the mass is subjected to small deviations $x(t)$. If however $x(t)$ would increase continuously, eventually the spring would overextend and finally break. Therefore the question of **linearity and the range of applicability** must be considered for each system.

In general a necessary condition for a linear system can be determined in term of an excitation $x(t)$ and a response $y(t)$. When a system at rest is subjected to an excitation $x_1(t)$, it exhibits a response $y_1(t)$ and if it is subjected to an excitation $x_2(t)$, it provides a corresponding response $y_2(t)$. For a linear system, it is fundamental that the excitation $x_1(t) + x_2(t)$ corresponds with a response $y_1(t) + y_2(t)$. This important characteristic is called **superposition**.

Furthermore, it is necessary that a magnitude scale factor is preserved by a linear system. Meaning that an input of β times $x_1(t)$ results in an response of β times the $y_1(t)$. This is called the property of **homogeneity**.

The linearity of many mechanical and electrical systems can be assumed over a reasonable large range of the variables. One can linearize non-linear elements assuming small signal conditions. Consider a response $y(t)$ generated by excitation variables $x_1(t), \ldots, x_n(t)$ related by

$$y(t) = g(x_1(t), \ldots, x_n(t)), \tag{1.44}$$

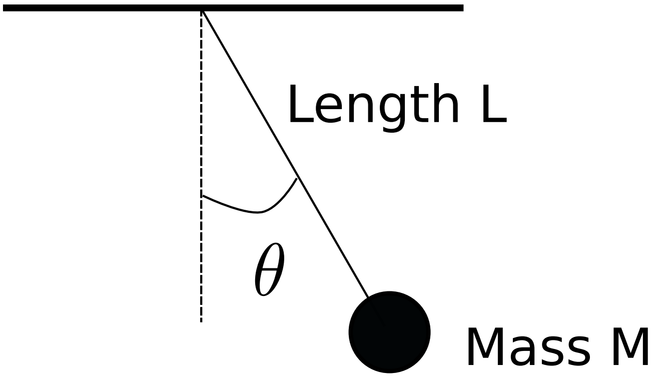

Figure 1.29: A pendulum with a load mass of M and a length of L. The mass of the wire is neglected together with all the friction.

where $g(x_1(t), \ldots, x_n(t))$ indicates $y(t)$ is a function of the variables $x_1(t), \ldots, x_n(t)$. To perform a linearization we first need to determine a normal operation point based on the required behaviour. For a controlled system this implies the desired control value and for a simulation this implies the assumed working point of the system. If we assume a operation point as $x_{1_0}, \ldots, x_{n_0}$ a **Taylor series** expansion around this operation point may be specified

$$\begin{aligned} y(t) &= g(x_1(t), \ldots, x_n(t)) \\ &\approx g(x_{1_0}, \ldots, x_{n_0}) + \left.\frac{\partial g}{\partial x_1}\right|_{x=x_{1_0},\ldots,x_{n_0}} (x_1(t) - x_{1_0}) + \ldots \\ &+ \left.\frac{\partial g}{\partial x_n}\right|_{x=x_{n_0},\ldots,x_{n_0}} (x_n(t) - x_{n_0}). \end{aligned} \tag{1.45}$$

The best way to illustrate this is using an example.

Example 1.3.2. *Pendulum model*

Consider the pendulum shown in figure!1.29. The torque on the mass is

$$T(t) = MgL \sin\theta(t), \tag{1.46}$$

where g is the gravity constant. If we assume the equilibrium condition for the mass is $\theta_0 = 0^0$. The Taylor expansion of equation 1.29 at the equilibrium is

$$\begin{aligned} T(t) &\approx MgL\sin(\theta_0) + MgL \left.\frac{\partial \sin\theta}{\partial\theta}\right|_{\theta=\theta_0} (\theta(t) - \theta_0), \\ &\approx MgL\sin(\theta_0) + MgL\cos\theta_0(\theta(t) - \theta_0) \end{aligned} \tag{1.47}$$

when $\theta_0 = 0^0$. *Then we have*

$$\begin{aligned} T(t) &= MGL\cos(0^0)(\theta(t) - 0^0) \\ T(t) &= MGL\theta(t) \end{aligned}$$

This approximation is reasonable accurate when $\pi/4 \leq \theta(t) \leq \pi/4$. *The transfer function between input torque and angular position then becomes:*

$$\frac{\Theta(s)}{T(s)} = \frac{1}{MGL}$$

If we assume an equilibrium point for the control circuit at $\theta_0 = 30^0$ *equation 1.47 becomes*

$$\begin{aligned} T(t) &\approx MgL\sin(30^0) + MgLcos(30^0)(\theta(t) - 30^0) \\ &\approx MgL\frac{1}{2} + MGL\frac{\sqrt{3}}{2}(\theta(t) - 30^0) \qquad (1.48) \end{aligned}$$

The transfer function between input torque ($T(t)$ *and angular position* $\theta(t)$ *cannot be determined because of the offset. To be able to specify a transfer function we need to change the variables* $T(t)$ *and* $\theta(t)$ *to new variables* $T'(t)$ *and* $\theta'(t)$ *that are equal to zero at the equilibrium point:*

$$\begin{aligned} T'(t) &= T(t) - MgL\frac{1}{2} \\ \theta'(t) &= \theta(t) - 30^0 \end{aligned}$$

Equation 1.48 can be rewritten in these two new variables $T'(t)$ *and* $\theta'(t)$ *leading to the transfer function*

$$\frac{\Theta'(s)}{T'(s)} = \frac{1}{MgL\frac{\sqrt{3}}{2}}. \qquad (1.49)$$

You can now perform your analysis using this transfer function but you need to keep in mind that both parameters have an offset in the original parameter space.

1.3.4 Transient-response analysis

This section presents the Matlab/Octave/Python approach to obtaining system responses when the inputs are time-domain inputs such as the step, impulse, and ramp functions.

Software representation of transfer-function systems Figure 1.30 shows a block with a transfer function. Such a block represents a system or an element of a system. To simplify our presentation, we shall call the block with a transfer function a system. MATLAB/Octave/Python given this system a name to represent it using the statement

```
sys = tf(num, den)
```

where name 'sys' is now used to represent the system (transfer function) i software.

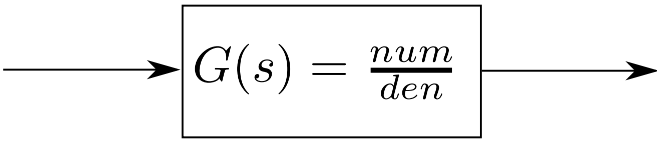

Figure 1.30: Block diagram of a transfer function system.

For example, consider the system

$$\frac{Y(s)}{X(s)} = \frac{s+12}{s^2+2s+22} \tag{1.50}$$

This system can be represented as two arrays, each containing the coefficients of the polynomials in decreasing powers of s as follows:

```
num = [1  12]
den = [1  2  22]
```

Step response. If num and den (the numerator and denominator of a transfer function) are known, we may define the system by

```
sys = tf(num,den)
```

Then, a command such as

```
step(sys) or step(num,den)
```

will generate a plot of a unit-step response (figure 1.31) and will display a response curve on the screen. The computation interval at and the time span of the response are determined by Matlab/Octave/Python. If we wish to compute the response every at seconds and plot the response curve for $0 \leq t \leq T$ (where T is an integer multiple of Δt), we enter the statement

$$t = 0 : \Delta t : T; \tag{1.51}$$

in the program and use the command

```
step(sys,t)  or step(num,den,t)
```

where t is the user-specified time. If step commands have left-hand arguments, such as

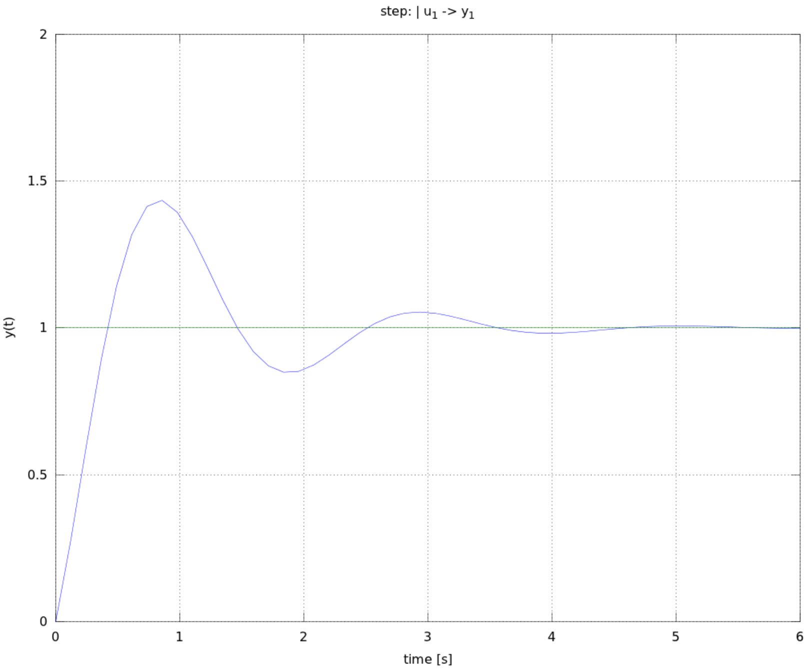

Figure 1.31: Unit-step response curve drawn by Octave.

```
y=step(sys,t)  or   y =step(num,den,t)
```

and

```
[y,t] = step(sys,t) or [y,t]=step(num,den,t)
```

Matlab produces the unit-step response of the system, but displays no plot to the screen but captures the data in vectors y,t. It is necessary to use a plot command to see response curves. The next example demonstrates the use of step commands.

Example 1.3.3. *Consider the spring-mass-dashpot system. The transfer function of the system is*

$$\frac{Y(s)}{U)s)} = \frac{bs+k}{ms^2+bs+k}$$

Assuming that m = 10 kg, b = 20 N-s/m, k = 100 N/m, and the input u(t) is a unit-step input (a step input of 1 m), obtain the response curve y(t). Substituting the given numerical values into the transfer function, we have

$$\frac{Y(s)}{U)s)} = \frac{20s+100}{10s^2+20s+100} = \frac{2s+10}{s^2+2s+10}$$

Matlab/Octave code	Python code

`num = [2 10];`
`den = [1 2 10];`
`sys = tf(num,den);`
`step(sys)`
`grid on`
	`from matplotlib.pyplot import *`
	`from control.matlab import *`
	
	`num = [1,12]`
	`den = [1,2,22]`
	`sys = tf(num, den)`
	`yout, T = step(sys)`
	`plot(T, yout)`
	`grid()`
	`show()`

Figure 1.32: Matlab/Octave/Python program 1.32.

Software Program 1.32 will produce the unit-step response y(t). The resulting unit-step response curve is shown in Figure 1.31.
In this plot, the duration of the response is automatically determined by the software. The title and axis labels are also automatically determined. If we wish to compute and plot the curve every 0.01 sec over the interval $0 \leq t \leq 8$*, we need to enter the following statement in the Matlab/Python program:*

```
(Matlab) t= 0:0.01:8     (Python) np.arange(0,8, 0.01);
```

Also, if we wish to change the title and axis labels, we enter the desired title and desired labels as shown in Program 1.33.
Note that if we did not enter the desired title and desired axis labels in the program, the title, x-axis label, and y-axis label on the plot are default values. When we enter the desired title and axis labels as shown in Program 1.33, the program erases the predetermined title and axis labels, except "(sec)" in the x-axis label, and replaces them with the ones we have specified. If the font sizes are too small, they can be made larger. For example, entering 'Fontsize ',20 in the title, xlabel, and ylabel variables as shown in Program 1.33. results in that size text appearing in those places. Figure 1.34 is a plot of the response curve obtained with Python Program 1.33.

Impulse response. The unit-impulse response of a dynamic system defined in the form of the transfer function may be obtained by use of one of the following commands:

Matlab/Octave code	Python code

t = 0:0.01 :8;
num = [2 10];
den=[1 2 10];
sys = tf(num,den);
step (sys, t)
grid
title('Unit-Step Response',
'Fontsize',20')
xlabe/{'t', 'Fontsize',20')
ylabel ('Output yl,
Fontsize ',20')
	from matplotlib.pyplot import *
	from control.matlab import *
	
	t = np.arange(0,8, 0.01)
	num = [1,12]
	den = [1,2,22]
	sys = tf(num, den)
	yout, T = step(sys,t)
	plot(T, yout)
	title('Unit step response',
	fontsize=20)
	xlabel('t',fontsize=20)
	ylabel('Output y',fontsize=20)
	grid()
	show()

Figure 1.33: Matlab/Octave/Python program 1.33.

```
    impulse(sys)   or   impulse(num,den)
    impulse(sys,t) or   impulse(num,den,t)
    y=impulse(sys)   or   y=impulse(num,den)
[y,t]=impulse(sys)   or   [y,t]=impulse(num,den)
```

The command impulse(sys) will generate a plot of the unit-impulse response and will display the impulse-response curve on the screen. If the command has a left-hand argument, such as y = impulse(sys), no plot is shown on the screen. It is then necessary to use a plot command to see the response curve on the screen. Before discussing computational solutions of problems involving impulse inputs, we present some necessary background material.

Obtaining response to arbitrary input. The command lsim produces the response of linear, time-invariant systems to arbitrary inputs. If the initial conditions of the system are zero, then

```
lsim(sys,u,t) or lsim(num,den,u,t)
```

produces the response of the system to the input u. Here, u is the input and t represents the times at which responses to u are to be computed. (The response

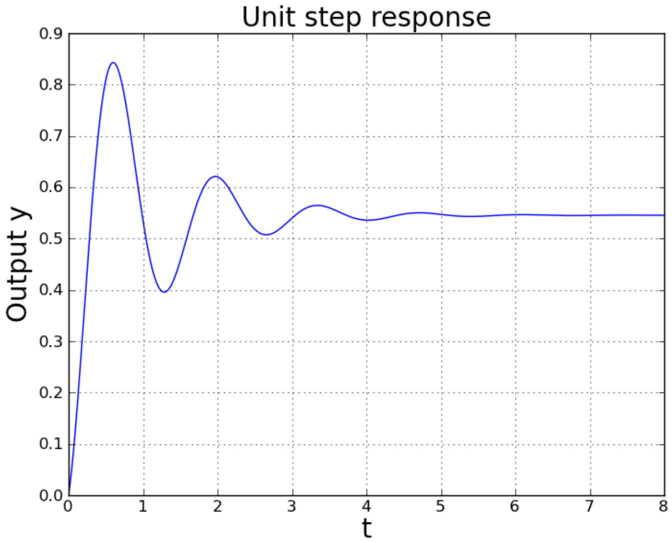

Figure 1.34: Unit-step response curve. Font sizes for title, xlabel, and ylabel are enlarged.

time span and the time increment are stated in t). If the initial conditions are nonzero, use the state-space approach presented in Chapter 1.4. If the initial conditions of the system are zero, then any of the commands

```
y= lsim(sys,u,t)    or   y= lsim(num,den,u,t)

[y,t] =lsim(sys,u,t)  or [y,t]= lsim(num,den,u,t)
```

returns the output response y. No plot is drawn. To plot the response curve, it is necessary to use the command plot(t,y). Note also that, by using lsim commands, we are able to obtain the response of the system to ramp inputs, acceleration inputs, and any other time functions that we can generate.

Example 1.3.4. *Consider once again the system shown in Figure1.35. Assume that m = 10 kg, b = 20 N-s/m, k =100 N/m, and u(t) is a unit-ramp input that is, the displacement u(t) increases linearly, or $u(t) = \alpha t$, where $\alpha = 1$. We shall obtain the unit-ramp response using the command*

```
lsim(sys,u,t)
```

The transfer function of the system is obtained by applying Newton's law:

$$m\frac{d^2y}{dt^2} = -b\left(\frac{dy}{dt} - \frac{du}{dt}\right) - k(y - u)$$

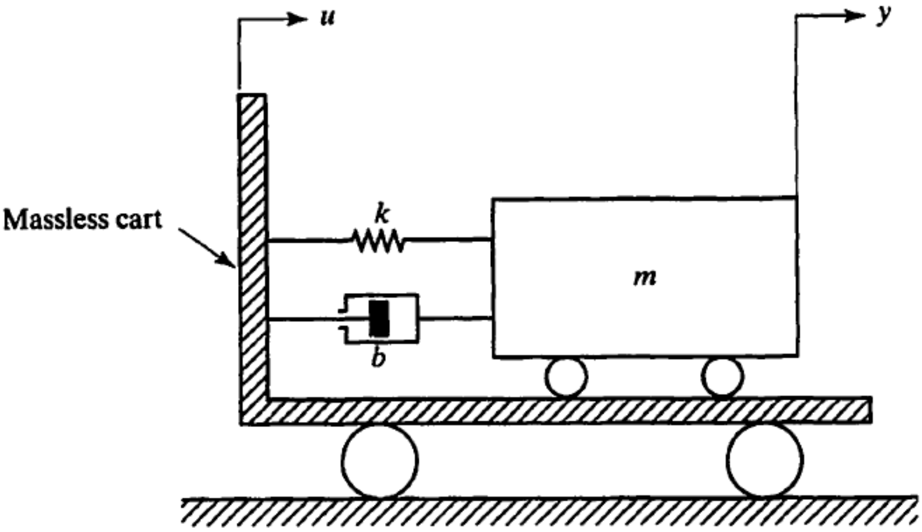

Figure 1.35: Spring-mass-dash-pot system mounted on a cart.

Taking the Laplace transform of the equation, assuming zero initial conditions, leads to the transfer function of the system

$$\frac{Y(s)}{U(s)} = \frac{bs+k}{ms^2+bs+k}$$

Filling in the numerical values leads to

$$\frac{Y(s)}{U(s)} = \frac{2s+10}{s^2+2s+10}$$

Program 1.36 produces the unit-ramp response. The resulting response curve y(t) versus t and the input ramp function u(t) versus t are shown in Figure 1.37.

Matlab/Octave code	Python code

`num = [2 10];`
`den=[1 2 10];`
`sys = tf(num, den);`
`t = 0:0.01 :4;`
`u = t;`
`y=lsim(sys,u',t');`
`plot(t,y,t,u);`
`grid`
`title('Unit-Ramp Response')`
`xlabel('t ' )`
`ylabel('Output y(t) and`
`input u(t) = t')`
`text(0.8,0.25,' y ')`
`text(0.15,0.8,'u')`
	`from matplotlib.pyplot import *`
	`from control.matlab import *`
	
	`num = [2,10]`
	`den=[1,2,10]`
	`sys = tf(num, den)`
	`t = np.arange(0,4, 0.01)`
	`u=t`
	`T, yout, xout = lsim(sys,u,t)`
	`plot(yout,T,T,T)`
	`grid()`
	`title('Unit-Ramp response')`
	`xlabel('t(sec)')`
	`ylabel('Output y(t) and input u(t) = t')`
	`text(0.8,0.25,'y')`
	`text(0.15,0.8,'u')`
	`show()`

Figure 1.36: Matlab/Octave/Python program 1.36.

1.3.5 Examples of transfer functions

Example 1.3.5. *A simplified Cruise Control Model*

Consider the simplified cruise control model of Examples 1.1.1. The differential equation of motion is defined by equation 1.8 as

$$\dot{v} + \frac{b}{m}v = \frac{u}{m}$$

The Laplace transform of this equation, assuming zero initial conditions, becomes

$$sV(s) + \frac{b}{m}V(s) \quad = \quad \frac{U(s)}{m} \tag{1.52}$$

from this equation we can easily obtain the transfer function

$$\frac{V(s)}{U(s)} = \frac{1}{ms + b}$$

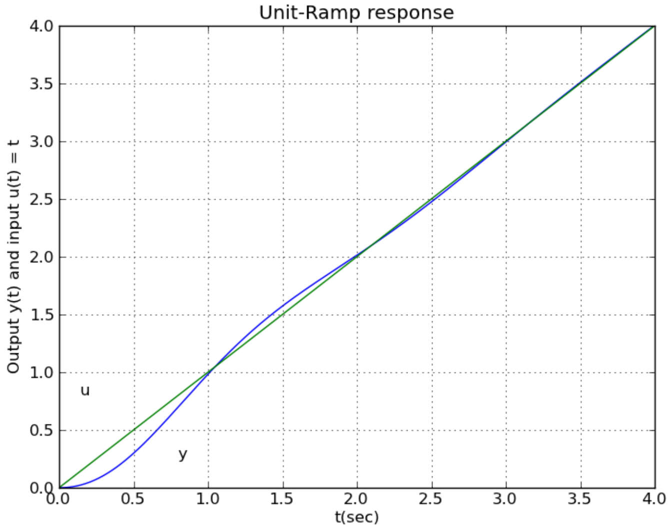

Figure 1.37: Plots of unit-ramp response curve y(t) and input ramp function u(t). (Plots are obtained with the use of the command plot(t,y,t,u).)

Since u(t) is a step force of magnitude 100 N, we may define u(t) = 100 step(t), where step(t) is a unit-step input of magnitude 1 N. Then Equations 1.52 can be written as

$$\frac{X(s)}{U(s)} = \frac{100}{ms + b} \tag{1.53}$$

Since step(t) is a unit-step input, can be obtained from Equation 1.53 with the use of a step command. (Step commands assume that the input is the unit-step input.) In this example, we shall demonstrate the use of the commands

```
y = step(sys,t)
```

and

```
plot(t,y)
```

with a mass m=200 kg and friction coefficient b=5 Nm/s. Matlab Program 1.38 produces the responses x(t) and y(t) of the system on one diagram. The response curve v(t) is shown in Figure 1.39.

Example 1.3.6. *A two-mass suspension model*

Consider the mechanical system shown in Figure 1.4 of example 1.1.2. The system is at rest initially. The displacements x(t) and y(t) are measured from their

Matlab/Octave code	Python code

t = 0:0.01 :5;
num = [100];
den=[200 5];
sys = tf(num,den);
y =step (sys, t)
plot(t,y)
grid
title(’Unit-Step Responses’,
’FontSize’, 20)
xlabel(’t [sec]’,’FontSize’, 20)
ylabel(’v(t) [m/s]’,
’FontSize’, 20)
	from matplotlib.pyplot import *
	from control.matlab import *
	
	t = np.arange(0,5, 0.01)
	num = [100]
	den=[100,5]
	sys = tf(num, den)
	y,t = step(sys,t)
	plot(t,y)
	grid()
	title(’Unit step response’)
	xlabel(’t [sec]’)
	ylabel(’v(t) [m/s] ’)
	show()

Figure 1.38: Matlab/Octave/Python program 1.38.

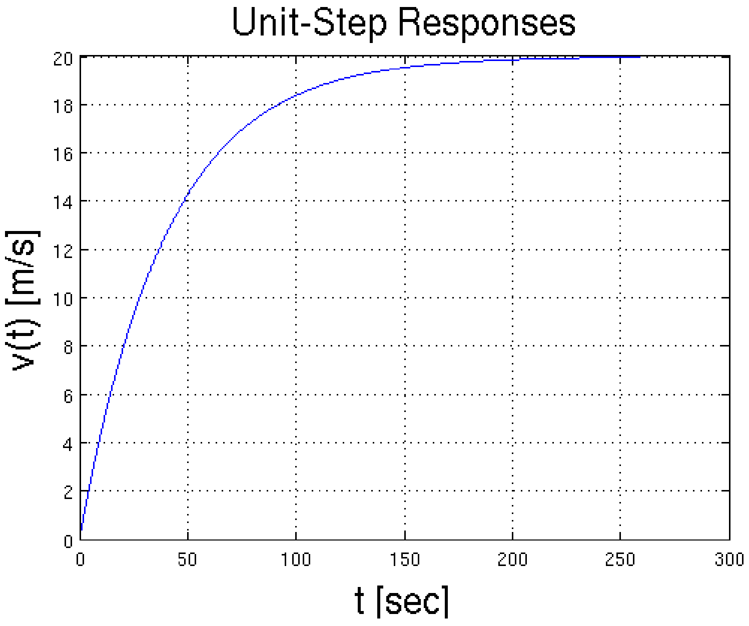

Figure 1.39: Step response curves v(t) to a scaled step of 100 [N] .

respective equilibrium positions. Assuming that r(t) is a step force input bump and the displacement x(t) is the output, obtain the transfer function of the system. Then, assuming that $m_1 = 20$, $m_2 = 375$ *kg, b = 9800 N-s/m,* k_w = *1e6 N/m,* k_s = *1.3e5 N/m, and r(t) is a step force of magnitude 0.3m, obtain an analytical solution for x(t).*

The equations of motion for the system are given by equations 1.12 and 1.11

$$\begin{aligned}\ddot{x} + \frac{b}{m_1}(\dot{x} - \dot{y}) + \frac{k_s}{m_1}(x - y) + \frac{k_w}{m_1}x &= \frac{k_w}{m_1}r \\ \ddot{y} + \frac{b}{m_2}(\dot{y} - \dot{x}) + \frac{k_s}{m_2}(y - x) &= 0\end{aligned}$$

Laplace transforming these two equations, assuming zero initial conditions, we obtain

$$s^2X(s) + \frac{b}{m_1}(sX(s) - sY(s)) + \frac{k_s}{m_1}(X(s) - Y(s)) + \frac{k_w}{m_1}X(s)$$

$$= \frac{k_w}{m_1}R(s) \tag{1.54}$$

$$s^2Y(s) + \frac{b}{m_2}(sY(s) - X(s)) + \frac{k_s}{m_2}(Y(s) - X(s)) = 0 \tag{1.55}$$

Solving Equation 1.55 for Y(s) and substituting the result into Equation 1.54, we get

$$Y(s) = \frac{\frac{b}{m_2}s + \frac{k_s}{m_2}}{s^2 + \frac{b}{m_2}s + \frac{k_s}{m_2}}X(s) \tag{1.56}$$

$$s^2X(s) + \frac{b}{m_1}s(1 - \frac{\frac{b}{m_2}s + \frac{k_s}{m_2}}{s^2 + \frac{b}{m_2}s + \frac{k_s}{m_2}})X(s) + \frac{k_s}{m_1}(1 - \frac{\frac{b}{m_2}s + \frac{k_s}{m_2}}{s^2 + \frac{b}{m_2}s + \frac{k_s}{m_2}})X(s) + \frac{k_w}{m_1}X(s)$$

$$= \frac{k_w}{m_1}R(s) \tag{1.57}$$

from which we obtain the transfer function

$$\begin{aligned}\frac{X(s)}{R(s)} &= \frac{\frac{k_w}{m_1}}{s^2 + \frac{b}{m_1}\frac{s^3}{s^2+\frac{b}{m_2}s+\frac{k_s}{m_2}} + \frac{k_s}{m_1}\frac{s^2}{s^2+\frac{b}{m_2}s+\frac{k_s}{m_2}} + \frac{k_w}{m_1}} \\ \frac{X(s)}{R(s)} &= \frac{k_w}{(m_1s^2 + k_w)(s^2 + \frac{b}{m_2}s + \frac{k_s}{m_2}) + bs^3 + k_ss^2}\end{aligned} \tag{1.58}$$

The transfer function X(s)/R(s) rewritten as polynomial in nominator and denominator is obtained from Equation 1.58:

$$\frac{X(s)}{R(s)} = \frac{k_w(s^2 + \frac{b}{m_2}s + \frac{k_s}{m_2})}{m_1s^4 + (b\frac{m_1}{m_2} + b)s^3 + (k_s\frac{m_1}{m_2} + k_s + k_w)s^2 + (k_w\frac{b}{m_2})s + k_w\frac{k_s}{m_2}} \tag{1.59}$$

Substituting the numerical values for m_1, m_2, k_s and k_w and b into the transfer functions given by Equations 1.59, we obtain

$$\begin{aligned}\frac{X(s)}{R(s)} &= \frac{1e6(s^2 + \frac{9800}{375}s + \frac{1.3e5}{375})}{20s^4 + (9800\frac{20}{375} + 9800)s^3 + (1.3e5\frac{20}{375} + 1.13e6)s^2 + (1e6\frac{9800}{375})s + 1e6\frac{1.3e5}{375}} \\ \frac{X(s)}{R(s)} &= \frac{1e6s^2 + 2.61e7s + 3.47e9}{20s^4 + 1.03e4s^3 + 1.14e6s^2 + 2.61e7s + 3.47e9} \end{aligned} \tag{1.60}$$

Since r(t) is a step force of magnitude 0.3 [m], we may define r(t) = 0.3 step(t), where step(t) is a unit-step input of magnitude 1 [m]. Then Equations 1.60 can be written as

$$\frac{X(s)}{U(s)} = 0.3\frac{1e6s^2 + 2.61e7s + 3.47e9}{20s^4 + 1.03e4s^3 + 1.14e6s^2 + 2.61e7s + 3.47e9} \tag{1.61}$$

To determine the response of the second output y(t) we can apply the transfer function from x(t) tot y(t) specified by equation 1.61

$$\frac{Y(s)}{X(s)} = \frac{\frac{b}{m_2}s + \frac{k_s}{m_2}}{s^2 + \frac{b}{m_2}s + \frac{k_s}{m_2}} \tag{1.62}$$

Since $r(t)$ is a unit-step input,x(t) and y(t) can be obtained based on Equations 1.61, 1.62 with the use of a step command. (Step commands assume that the input is the unit-step input.) Matlab Program 1.40 produces the responses x(t) and y(t) of the system on one diagram. The response curves x(t) and y(t) are shown in Figure 1.41.

Example 1.3.7. *Combustion engine cruise control*

Consider the combustion engine cruise control described by example 1.13. This is a non-linear model that can only be converted into a transfer function by linearizing the model around a working point. The non-linear differential equation is:

$$\begin{aligned} m\frac{dv}{dt} &= g_{CC}(\omega, u, \theta) \\ &= \frac{nu}{r}T_m(1 - \beta(\frac{\omega}{\omega_m} - 1)^2)) - mg\sin(\theta) - mgC_r sgn(v) - \frac{1}{2}\rho C_d A v^2. \end{aligned} \tag{1.63}$$

A first order Taylor approximation can be obtained by applying equation 1.45 to the non-linear model. Before we can apply a Taylor approximation we need to specify a working point. A cruise control tries to drive at a constant velocity, therefore

Matlab/Octave code

```
m1=20;
m2=375;
Kw=1e6;
Ks=1.3e5;
b=9800;

%Transfer function from r(t) to x(t):
TFX = tf([Kw Kw*b/m2 Kw*Ks/m2],
[m1  b*m1/m2+b  Ks*m1/m2+Ks+Kw Kw*b/m2 Kw*Ks/m2]);

%Transfer function from x(t) to y(t):
TFY =  tf([b/m2 Ks/m2],[1 b/m2 Ks/m2])

T=[0:0.001:0.5];
y1=step(TFX,T);
y2=step(TFY*TFX,T);

plot(T,y1,T,y2) % 2 on 1 plot
grid
title('Unit-Step Responses','FontSize', 20)
xlabel('t (sec)','FontSize', 20)
ylabel('x(t) and y(t)','FontSize', 20)
```

Python code

```
TODO
```

Figure 1.40: Matlab/Octave/Python program 1.40.

this required constant velocity V_e is chosen as linear velocity working point. This linear velocity corresponds with an angular velocity up to the gear ratio factor an including the radius of the wheel (r): $\omega_e = anV_e$. There are three variables in equation 1.63 (ω, u, θ). We have chosen a working point for ω and still need one for u and θ. If the steep angle θ is chosen equal to θ_e, a working point for u_e can be derived by imposing that in an equilibrium the velocity is constant or $\frac{dv}{dt} = 0$ in

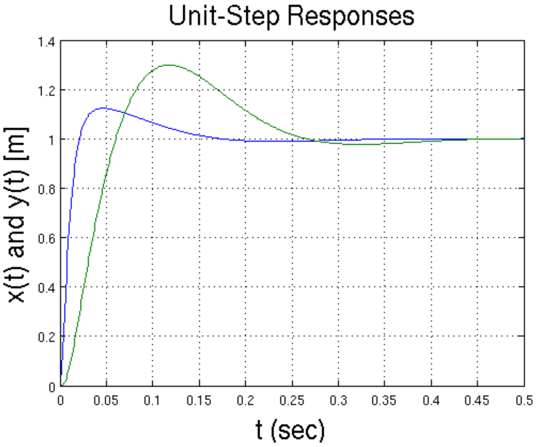

Figure 1.41: Step response of the first ($x(t)$) and the second mass ($y(t)$) in the quarter car suspension model of Example 1.1.2.

equation 1.63:

$$\begin{aligned} 0 &= g_{CC}(\omega_e, u_e, \theta_e) \\ &= an\ u_0 T_m(1-\beta(\frac{\omega_0}{\omega_m}-1)^2)) - mg\sin(\theta_0) - mgC_r sgn(v_0) - \frac{1}{2}\rho C_d A v_0^2. \end{aligned} \tag{1.64}$$

Now we can isolate u_0 in equation 1.64 to obtain

$$u_e = \frac{mgC_r sgn(v_e) + \frac{1}{2}\rho C_d A v_e^2}{anTm(1-\beta(\frac{\omega_e}{\omega_m}-1)^2))} \tag{1.65}$$

A Taylor approximation around ω_0, u_0, θ_0 then becomes:

$$\begin{aligned} m\frac{dv}{dt} &\approx g_{CC}(\omega_e, u_e, \theta_e) \\ &+ \left(anu_e T_m(-2)\beta(\frac{an\ v_e}{\omega_m}-1)\frac{an}{\omega_m} - \rho C_d A v_e\right)(v_- v_e) \\ &+ \left(an\ T_m(1-\beta(\frac{\omega_e}{\omega_m}-1)^2)\right)(u-u_e) \\ &- (m\ g\ \cos(\theta_e))(\theta-\theta_e) \end{aligned} \tag{1.66}$$

To be able to apply the Laplace transform we need to introduce new variables that are equal to zero at the working point: $v' = v + v_e$, $u' = u + u_e$ and $\theta' = \theta + \theta_e$. If *we introduce three new parameters* $a = \left(anu_eT_m(-2)\beta(\frac{an\ v_e}{\omega_m} - 1)\frac{an}{\omega_m} - \rho C_d A v_e\right)$, $b = \left(an\ T_m(1 - \beta(\frac{\omega_e}{\omega_m} - 1)^2)\right)$ *and* $bg = (m\ g\ \cos(\theta_e))$ *equation 1.66 is reduced to*

$$m\frac{dv'}{dt} \approx a\ v' + bu' + bg\ \theta' \tag{1.67}$$

Laplace transforming these two equations, assuming zero initial conditions, and a constant steep angle θ' *we obtain*

$$\frac{V'(s)}{U'(s)} = \frac{b}{ms - a} \tag{1.68}$$

Typical parameters values are: rolling friction is C_r *= 0.01,* ρ *= 1.3* k/m^3, C_d *= 0.32 and A = 2.4* m^2*. Typical parameters for the generated force by the combustion engine are* $T_m = 190Nm$, $\omega_m = 420$ *rad/s (about 4000 RPM) and* $\beta = 0.4$*. Matlab Program 1.43 produces the step response v(t) of the system.*
The response curves x(t) and y(t) are shown in Figure 1.42.

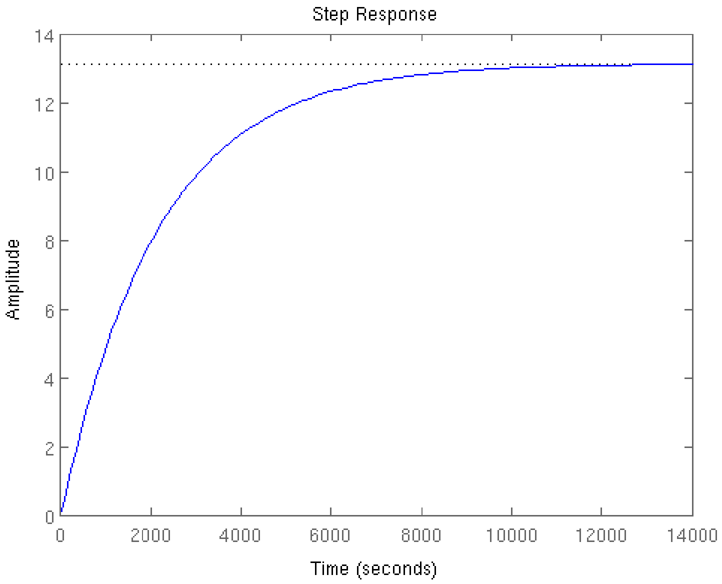

Figure 1.42: Step response of the car velocity v(t) in [m/s] for the combustion engine cruise control model of Example 1.13.

Matlab/Octave code

```
%%% The steep angle
theta=4*pi/180;

gears = [40, 25, 16, 12, 10]; % gear ratios
m=1600;        % mass of car kg
Tm = 190;      % engine torque constant, Nm
wm = 420;      % peak torque rate, rad/sec
bbeta = 0.4;   % torque coefficient
Cr = 0.01;     % coefficient of rolling friction
rho = 1.3;     % density of air, kg/m^3
Cd = 0.32;     % drag coefficient
A = 2.4;       % car area, m^2
g = 9.8;       % gravitational constant
gear=1;

%%% Linearisation points:
v_e=25.8;              %equilibrium velocity m/s
theta_e=4*pi/180;      %equilibrium slope deg


%Compute the trottle u_e required to keep velocity
 ve at slope thetae, velocity ve, and gear
an=gears(gear);
A0=Tm*an*(1-bbeta*(an*v_e/wm-1)^2);
A1=m*g*Cr+rho*Cd*A*v_e^2/2+m*g*sin(theta);
u_e=A1/A0;

%Compute linearisation parameters:
w=an*v_e;
T=Tm*(1-bbeta*(w/wm-1)^2);
pT=-2*bbeta*Tm*(w/wm-1)/wm;
a=(an^2*u_e*pT-rho*Cd*A*v_e)/m;
b=an*T/m;
bg=g*cos(theta_e);

%dv=1/m*((A0*u_e-A1) +a*(y(2)-v_e)+b*(ul-u_e)
    -bg*(theta-theta_e))

TF= tf(bg,[m -a])

step(TF);
```

Python code

```
TODO
```

Figure 1.43: Matlab/Octave/Python program 1.43.

1.4 Continuous-time systems: State Space models

1.4.1 Introduction

In this section we shall present introductory material on state-space analysis of control systems.

Modern Control Theory. The modern trend in engineering systems is toward greater complexity, due mainly to the requirements of complex tasks and good accuracy. Complex systems may have multiple inputs and multiple outputs and may be time varying. Because of the necessity of meeting increasingly stringent requirements on the performance of control systems, the increase in system complexity, and easy access to large scale computers, modern control theory, which is a new approach to the analysis and design of complex control systems, has been developed since around 1960. This new approach is based on the concept of state. The concept of state by itself is not new since it has been in existence for a long time in the field of classical dynamics and other fields.

Modern Control Theory Versus Conventional Control Theory. Modern control theory is contrasted with conventional control theory in that the former is applicable to multiple-input-multiple-output systems, which may be linear or nonlinear, time invariant or time varying, while the latter is applicable only to linear time-invariant single-input-single-output systems. Also, modern control theory is essentially a time-domain approach, while conventional control theory is a complex frequency-domain approach. Before we proceed further, we must define state, state variables, state vector, and state space.

State. The state of a dynamic system is the smallest set of variables (called state variables) such that the knowledge of these variables at $t = t_0$, together with the knowledge of the input for $t \geq t_0$, completely determines the behavior of the system for any time $t \geq t_0$.
Note that the concept of state is by no means limited to physical systems. It is applicable to biological systems, economic systems, social systems, and others.

State Variables. The state variables of a dynamic system are the variables making up the smallest set of variables that determine the state of the dynamic system. If at least n variables $x_1, x_2, \ldots, x_n$, are needed to completely describe the behavior of a dynamic system (so that once the input is given for $t \geq t_0$ and the initial state at $t = t_0$ is specified, the future state of the system is completely determined), then such n variables are a set of state variables.

Note that state variables need not be physically measurable or observable quantities. Variables that do not represent physical quantities and those that are neither measurable nor observable can be chosen as state variables. Such freedom in choosing state variables is an advantage of the state-space methods. Practically, however, it is convenient to choose easily measurable quantities for the state variables, if this is possible at all, because optimal control laws will require the feedback of all state variables with suitable weighting.

State Vector. If n state variables are needed to completely describe the behavior of a given system, then these n state variables can be considered the n components of a vector $\mathbf{X}$. Such a vector is called a state vector. A state vector is thus a vector that determines uniquely the system state $\mathbf{X}(t)$ for any time $t \geq t_0$, once the state at $t = t_0$ is given and the input u(t) for $t \geq t_0$ is specified.

State Space. The n-dimensional space whose coordinate axes consist of the x_1 axis, x_2 axis, ... , x_n axis, where $x_1, x_2, ..., x_n$ are state variables; is called a state space. Any state can be represented by a point in the state space.

State-Space Equations. In state-space analysis we are concerned with three types of variables that are involved in the modeling of dynamic systems: input variables, output variables, and state variables. The state-space representation for a given system is not unique, except that the number of state variables is the same for any of the different state-space representations of the same system.

The dynamic system must involve elements that memorize the values of the input for $t \geq t_1$. Since integrators in a continuous-time control system serve as memory devices, the outputs of such integrators can be considered as the variables that define the internal state of the dynamic system. Thus the outputs of integrators serve as state variables. The number of state variables to completely define the dynamics of the system is equal to the number of integrators involved in the system.

Assume that a multiple-input-multiple-output system involves n integrators. Assume also that there are r inputs $u_1(t), u_2(t), \ldots, u_r(t)$ and m outputs $y_1(t), y_2(t), \ldots, y_m(t)$. Define n outputs of the integrators as state variables: $x_1(t), x_2(t), \ldots, x_n(t)$. Then the system may be described by

$$\begin{aligned}
\dot{x}_1(t) &= f_1(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t) \\
\dot{x}_2(t) &= f_2(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t) \\
&\vdots \\
\dot{x}_n(t) &= f_n(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t)
\end{aligned} \tag{1.69}$$

The outputs $y_1(t), y_2(t), \ldots, y_n, (t)$ of the system may be given by

$$\begin{aligned} y_1(t) &= g_1(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t) \\ y_2(t) &= g_2(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t) \\ &\vdots \\ y_m(t) &= g_m(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t) \end{aligned} \tag{1.70}$$

If we define

$$\begin{aligned} \mathbf{x}(t) &= \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix}, \quad \mathbf{f}(\mathbf{x}, \mathbf{u}, t) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t) \end{bmatrix}, \\ \mathbf{y}(t) &= \begin{bmatrix} y_1(t) \\ \vdots \\ y_m(t) \end{bmatrix}, \quad \mathbf{g}(\mathbf{x}, \mathbf{u}, t) = \begin{bmatrix} g_1(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t) \\ \vdots \\ g_m(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_r; t) \end{bmatrix}, \\ \mathbf{u}(t) &= \begin{bmatrix} u_1(t) \\ \vdots \\ u_r(t) \end{bmatrix} \end{aligned}$$

then Equations 1.69 and 1.70 become

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{1.71}$$

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}, \mathbf{u}, t) \tag{1.72}$$

where Equation 1.71 is the state equation and Equation 1.72 is the output equation. If vector functions **f** and/or **g** involve time t explicitly, then the system is called a time-varying system.

If Equations 1.71 and 1.72 are linearized about the operating state, then we have the following linearized state equation and output equation:

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) \tag{1.73}$$

$$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t) \tag{1.74}$$

where $\mathbf{A}(t)$ is called the state matrix, $\mathbf{B}(t)$ the input matrix, $\mathbf{C}(t)$ the output matrix, and $\mathbf{D}(t)$ the direct transmission matrix. These matrices can be obtained for nonlinear systems by a linearization as described in Section 1.3.3. A block diagram representation of Equations 1.73 and 1.74 is shown in Figure 1.44.

If vector functions **f** and **g** do not involve time t explicitly then the system is called a time-invariant system. In this case, Equations 1.73 and 1.74 can be simplified to

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{1.75}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{1.76}$$

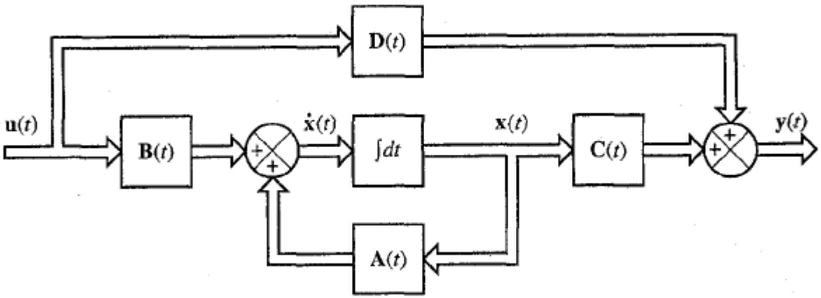

Figure 1.44: Block diagram of the linear, continuous-time control system represented in state space.

Equation 1.75 is the state equation of the linear, time-invariant system. Equation 1.76 is the output equation for the same system. In this book we shall be concerned mostly with systems described by Equations 1.75 and 1.76.
In what follows we shall present an examples for deriving a state equation from a transfer function or an ODE model.

Example 1.4.1. *Consider the mechanical system shown in Figure 1.45. We assume that the system is linear. The external force u(t) is the input to the system, and the displacement y(t) of the mass is the output. The displacement y(t) is measured from the equilibrium position in the absence of the external force.This system is a single-input-single-output system. From the diagram, the system equation is*

$$m\ddot{y} + b\dot{y} + ky = u \tag{1.77}$$

This system is of second order. This means that the system involves two integrators. Let us define state variables $x_1(t)$ and $x_2(t)$ as

$$\begin{aligned} x_1(t) &= y(t) \\ x_2(t) &= \dot{y}(t) \end{aligned}$$

or

$$\dot{x}_1 = x_2 \tag{1.78}$$

$$\dot{x}_2 = -\frac{k}{m}x_1 - \frac{b}{m}x_2 + \frac{1}{m}u. \tag{1.79}$$

The output equation is

$$y = x_1 \tag{1.80}$$

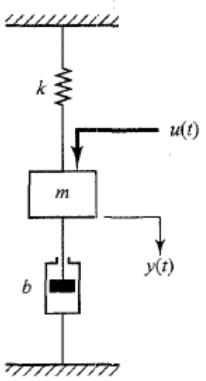

Figure 1.45: Mechanical system.

In a vector-matrix form, Equations 1.78 and 1.79 can be written as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u \tag{1.81}$$

and

$$y = [1 \quad 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{1.82}$$

Equation 1.81 is a state equation and Equation 1.82 is an output equation for the system. Equations 1.81 and 1.82 are in the standard form:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \end{aligned}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ -\frac{1}{m} \end{bmatrix}, \quad \mathbf{C} = [1 \quad 0], \quad \mathbf{D} = [0] \tag{1.83}$$

1.4.2 From State-Space model to Transfer Function

In what follows we shall show how to derive the transfer function of a single-input-single-output system from the state-space equations. Let us consider the system whose transfer function is given by

$$\frac{Y(s)}{U(s)} = G(s) \tag{1.84}$$

This system may be represented in state space by the following equations:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{1.85}$$

$$y(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{1.86}$$

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ is the input, and $\mathbf{y}$ is the output.The Laplace transforms of Equations 1.85 and 1.86 are given by

$$\begin{aligned} s\mathbf{X}(s) - \mathbf{x}(0) &= \mathbf{AX}(s) + \mathbf{B}U(s) \qquad (1.87) \\ Y(s) &= \mathbf{CX}(s) + \mathbf{D}U(s) \qquad (1.88) \end{aligned}$$

Since the transfer function was previously defined as the ratio of the Laplace transform of the output to the Laplace transform of the input when the initial conditions were zero, we set $\mathbf{x}(0)$ in Equation 1.87 to be zero. Then we have

$$s\mathbf{X}(s) - \mathbf{AX}(s) = \mathbf{B}U(s)$$

or

$$(s\mathbf{I} - \mathbf{A})\mathbf{X}(s) = \mathbf{B}U(s)$$

By premultiplying $(s\mathbf{I} - \mathbf{A})^{-1}$ to both sides of this last equation, we obtain

$$\mathbf{X}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}U(s) \qquad (1.89)$$

By substituting Equation 1.89 into Equation 1.87, we get

$$H(s) = \frac{Y(s)}{U(s)} = \left[\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}\right] \qquad (1.90)$$

This is the transfer-function expression of the system in terms of $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{D}$. Note that the right-hand side of Equation 1.89 involves $(s\mathbf{I} - \mathbf{A})^{-1}$. Hence H(s) can be written as

$$H(s) = \frac{Q(s)}{|s\mathbf{I} - \mathbf{A}|}$$

where Q(s) is a polynomial in s. Therefore, $|s\mathbf{I} - \mathbf{A}|$ is equal to the characteristic polynomial of H(s). In other words, the eigenvalues of $\mathbf{A}$ are identical to the poles of H(s). If we consider a multiple-input-multiple-output system. Assume that there are r inputs $u_1, u_2, \ldots, u_r$, and m outputs $y_1, y_2, \ldots, y_m$. Define

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_r \end{bmatrix}$$

The transfer matrix $\mathbf{G}(s)$ relates the output $\mathbf{Y}(s)$ to the input $\mathbf{u}(s)$, or

$$\mathbf{G}(s) = \frac{\mathbf{Y}(s)}{\mathbf{U}(s)} = \left[\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}\right] \qquad (1.91)$$

1.4.3 State-space representation of dynamic systems

A dynamic system consisting of a finite number of lumped elements may be described by ordinary differential equations in which time is the independent variable. By use of vector-matrix notation, an n th-order differential equation may be expressed by a first-order vector-matrix differential equation. If n elements of the vector are a set of state variables, then the vector-matrix differential equation is a state equation. In this section we shall present methods for obtaining state-space representations of continuous-time systems.

State-Space Representation of n th-Order Systems of Linear Differential Equations in which the Forcing Function Does Not Involve Derivative Terms.

Consider the following nth-order system:

$$y^n + a_1 y^{n-1} + \cdots + a_{n-1}\dot{y} + a_n y = u \qquad (1.92)$$

Noting that the knowledge of $y(0), \dot{y}(0), \ldots, y^{n-1}(0)$, together with the input u(t) for $t \geq 0$, determines completely the future behavior of the system, we may take $y(t), \dot{y}(t), \ldots, y^{n-1}(t)$ as a set of n state variables. (Mathematically, such a choice of state variables is quite convenient. Practically, however, because higher-order derivative terms are inaccurate, due to the noise effects inherent in any practical situations, such a choice of the state variables may not be desirable.) Let us define

$$\begin{aligned} x_1 &= y \\ x_2 &= \dot{y} \\ &\vdots \\ x_n &= y^{n-1} \end{aligned}$$

Then Equation 1.75 can be written as

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}_{n-1} &= x_n \\ \dot{x}_n &= a_n x_1 - \ldots - a_1 x_n + u \qquad (1.93) \end{aligned}$$

or

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \qquad (1.94)$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \dots & -a_1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \tag{1.95}$$

The output can be given by

$$y = [1 \;\; 0 \quad \dots \quad 0] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{1.96}$$

or

$$Y(s) \;\; = \;\; \mathbf{CX} \tag{1.97}$$

Note that the state-space representation for the transfer function system

$$\frac{Y(s)}{U(s)} = \frac{1}{s^n + a_1 s^{n-1} + \ldots + a_{n-1}s + a_n} \tag{1.98}$$

is given also by Equation 1.94 and 1.97.

State-Space Representation of n th Order Systems of Linear Differential Equations in which the Forcing Function Involves Derivative Terms.

Consider the differential equation system that involves derivatives of the forcing function, such as

$$y^n + a_1 y^{n-1} + \cdots + a_{n-1}\dot{y} + a_n y = b_0 u^{n-1} + \ldots + b_{n-1}\dot{u} + b_n u \tag{1.99}$$

The main problem in defining the state variables for this case lies in the derivative terms. The state variables must be such that they will eliminate the derivatives of u in the state equation. One way to obtain a state equation and output equation is to define the following n variables as a set of n state variables:

$$\begin{aligned} x_1 &= y - \beta_0 u \\ x_2 &= \dot{y} - \beta_0 \dot{u} - \beta_1 u \\ &\vdots \\ x_n &= y^{n-1} - \beta^0 u^{n-1} - \beta_1 u^{n-2} - \ldots - \beta_{n-2}\dot{u} - \beta_{n-1}u \end{aligned} \tag{1.100}$$

where $\beta_0, \beta_1, \ldots, \beta_n$, are determined from

$$\begin{aligned}
\beta_0 &= b_0 \\
\beta_1 &= b_1 - \beta_0 a_1 \\
&\vdots \\
\beta_n &= b_n - a_1\beta_{n-1} - \ldots - a_{n-1}\beta_1 - a_n\beta_0
\end{aligned} \tag{1.101}$$

With this choice of state variables the existence and uniqueness of the solution of the state equation is guaranteed. (Note that this is not the only choice of a set of state variables.) With the present choice of state variables, we obtain

$$\begin{aligned}
\dot{x}_1 &= x_2 + \beta_1 u \\
\dot{x}_2 &= x_3 + \beta_2 u \\
&\vdots \\
\dot{x} &= -a_n x_1 - a_{n-1}x_2 - \ldots - a_1 x_n + \beta_n u
\end{aligned} \tag{1.102}$$

Equation 1.102 and the output equation can be written as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \ldots & -a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix} u$$

$$y = [1 \;\; 0 \;\; \ldots \;\; 0] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \beta_0 u \tag{1.103}$$

or

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \tag{1.104}$$

$$\dot{\mathbf{y}} = \mathbf{C}\mathbf{x} + \mathbf{D}u \tag{1.105}$$

In this state-space representation, matrices **A** and **C** are exactly the same as those for the system of Equation 1.92. The derivatives on the right-hand side of Equation 1.99 affect only the elements of the B matrix. Note that the state-space representation for the transfer function system

$$\frac{Y(s)}{U(s)} = \frac{b_0 s^{n-1} + \ldots + b_{n-1}s + b_n}{s^n + a_1 s^{n-1} + \ldots + a_{n-1}s + a_n} \tag{1.106}$$

is given also by Equation 1.104 and 1.105.

Example 1.4.2. *Consider the spring-mass-dashpotsystem mounted on a massless cart as shown in Figure 1.46. A dashpot is a device that provides viscous friction, or damping. It consists of a piston and oil-filled cylinder. Any relative motion between the piston rod and the cylinder is resisted by the oil because the oil must flow around the piston (or through orifices provided in the piston) from one side of the piston to the other. The dashpot essentially absorbs energy. This absorbed energy is dissipated as heat, and the dashpot does not store any kinetic or potential energy. The dashpot is also called a damper.*

Let us obtain mathematical models of this system by assuming that the cart is standing still for $t < 0$ and the spring-mass-dashpotsystem on the cart is also standing still for $t < 0$. In this system, u(t)is the displacement of the cart and is the input to the system. At t = 0, the cart is moved at a constant speed, or u = constant. The displacement y(t) of the mass is the output. (The displacement is relative to the ground.) In this system, m denotes the mass, b denotes the viscous friction coefficient, and k denotes the spring constant.We assume that the friction force of the dashpot is proportional to $\dot{y} - \dot{u}$ and that the spring is a linear spring; that is, the spring force is proportional to $y - u$.

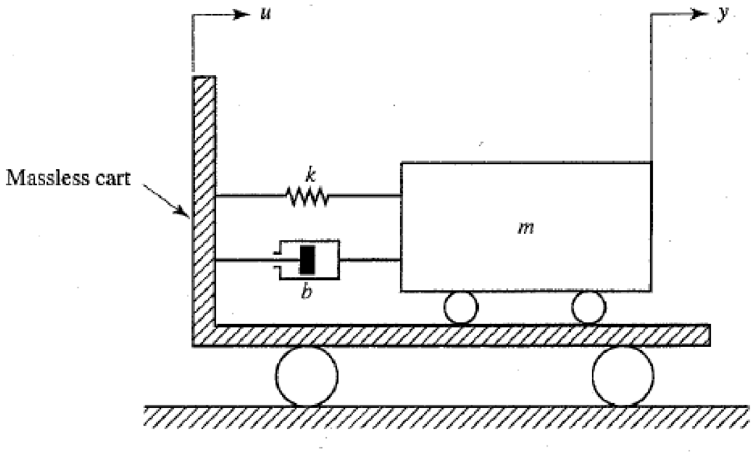

Figure 1.46: Spring-mass-dashpot system mounted on a cart.

For translational systems, Newton's second law states that

$$m\frac{d^2y}{dt^2} = -b\left(\frac{dy}{dt} - \frac{du}{dt}\right) - k(y - u)$$

Taking the ratio of $Y(s)$ *to* $U(s)$*, we find the transfer function of the system to be*

$$G(s) = \frac{Y(s)}{U(s)} = \frac{bs+k}{ms^2+bs+k}$$

Such a transfer function representation of a mathematical model is used very frequently in control engineering.
Next we shall obtain a state-space model of this system. We shall first compare the differential equation for this system

$$\ddot{y} + \frac{b}{m}\dot{y} + \frac{k}{m}y = \frac{b}{m}\dot{u} + \frac{k}{m}u$$

we rewrite this equation to the standard form and identify a_1, a_2, b_0, b_1 *as follows:*

$$a_1 = \frac{b}{m}, \quad a_2 = \frac{k}{m}, \quad b_0 = 0, \quad b_1\frac{b}{m}, \quad b_2 = \frac{k}{m}$$

Referring to Equation 1.101, we have

$$\begin{aligned} \beta_0 &= 0 \\ \beta_1 &= \frac{k}{m} \\ \beta_2 &= \frac{k}{m} - \left(\frac{b}{m}\right)^2 \end{aligned}$$

The state space equation therefore becomes (1.108)

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{b}{m} \\ \frac{k}{m} - \left(\frac{b}{m}\right)^2 \end{bmatrix} u \tag{1.107}$$

$$y = [1 \quad 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{1.108}$$

1.4.4 Transformation of mathematical models with software

Software is quite useful to transform the system model from transfer function to state space, and vice versa. We shall begin our discussion with transformation from transfer function to state space.
Let us write the closed-loop transfer function as

$$\frac{Y(s)}{U(s)} = \frac{\text{numerator polynomial in s}}{\text{denominator polynomial in s}} = \frac{\text{num}}{\text{den}}$$

Once we have this transfer-function expression, the software command

```
[A, B, C, D] = tf2ss(num,den)
```

will give a state-space representation. It is important to note that the state-space representation for any system is not unique. There are many (infinitely many) state-space representations for the same system. The software command gives one possible state-space representation.

Transformation From Transfer Function To State Space

Consider the transfer function system

$$\frac{Y(s)}{U(s)} = \frac{s}{s^3 + 14s^2 + 56s + 160} \tag{1.109}$$

Software transforms the transfer function given by Equation 1.109 into the state-space representation. For the example system considered here, Matlab/Octave/Python Program 1.47 will produce matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{D}$.

Matlab/Octave code	Python code

`>> num = [0 0 1 0];`
`>> den=[1 4 56 160];`
`>> [A,B,C,D] = tf2ss(num,den)`
`A =`
 	
`    0     1     0`
`    0     0     1`
`  -160   -56    -4`
 	
`B =`
 	
`   0`
`   0`
`   1`
 	
`C =`
 	
`   0   1   0`
 	
`D = 0`
	`from matplotlib.pyplot import *`
	`from control.matlab import *`
	`num = [0, 0, 1, 0]`
	`den=[1, 4, 56, 160]`
	`sys = tf2ss(num,den)`
	`print(sys)`
	
	`  A = [[  -4.   -56. -160.]`
	` [   1.     0.     0.]`
	` [   0.     1.     0.]]`
	
	`B = [[ 1.]`
	` [ 0.]`
	` [ 0.]]`
	
	`C = [[ 0.  1.  0.]]`
	
	`D = [[ 0.]]`

Figure 1.47: Matlab/Octave/Python program 1.47.

Transformation From State Space To Transfer Function

To obtain the transfer function from state-space equations, use the following command:

```
[num,den]=ss2tf(A,B,C,D,iu)
```

iu must be specified for systems with more than one input. For example, if the system has three inputs (u_1, u_2, u_3) , then iu must be either 1,2, or 3, where 1 implies u_1, 2 implies u_2, and 3 implies u_3.

Matlab/Octave code	Python code

A = [0 1 0; 0 0 1 ; -5 -25 -5];
B = [0; 25; -120]
C = [1 0 0];
D = [0];
[num,den] = ss2tf(A,B,C,D)
num =
 	
25.0000 5.0000
 	
den =
 	
1.0000 5.0000 25.0000 5.0000
	from matplotlib.pyplot import *
	from control.matlab import *
	A = [[0,1,0],[0,0,1],[-5,-25,-5]]
	B = [[0], [25], [-120]]
	C = [1,0,0]
	D = [0]
	sys=ss2tf(A,B,C,D)
	 [0 0 24 4]
	[1. 5. 25. 5.]

Figure 1.48: Matlab/Octave/Python program 2.37.

1.4.5 Examples of state space models

Example 1.4.3. *Simplified Cruise Control*

The simplified cruise control equation 1.8 can be rewritten as a state space model by introducing a state vector $\mathbf{X} = [v]$ *and an input vector* $\mathbf{u} = [u]$ *leading to following matrices:*

$$\begin{aligned} \mathbf{A} &= \left[-\frac{b}{m} \right] \\ \mathbf{B} &= \left[\frac{1}{m} \right] \end{aligned}$$

The Matlab program 1.49 creates the state space model and determines the step response.

Example 1.4.4. *A quarter car suspension model*

The quarter car model equations 1.11, 1.12 can be rewritten as a state space model by introducing a state vector $\mathbf{X} = [x, \quad \dot{x}, \quad y, \quad \dot{y}]^T$ *and an input vector* $\mathbf{u} = [r]$

Matlab/Octave code	Python code

`m=200 ; % kg`
`b=5 ; %N/(m/s)`
 	
`A=[-b/m];`
`B=[1/m];`
 	
`C=[1]; % we take v(t) as output`
`D=[0];`
 	
`sys=ss(A,B,C,D); %create the state space model`
 	
`step(sys);`
	`TODO`

Figure 1.49: Matlab/Octave/Python program 1.49.

leading to following state space matrices:

$$
\begin{aligned}
\mathbf{A} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ (-K_s - K_w)/m_1 & -b/m_1 & K_s/m_1 & b/m_1 \\ 0 & 0 & 0 & 1 \\ K_s/m_2 & b/m_2 & -K_s/m_2 & -b/m_2 \end{bmatrix} \\
\mathbf{B} &= \begin{bmatrix} 0 \\ K_w/m_1 \\ 0 \\ 0 \end{bmatrix}
\end{aligned}
$$

The Matlab program 1.50 creates the state space model and determines the step response. Additional the Bode diagram is shown, the damping factor is calculated and it is shown that the poles of the **A** *matrix are equal to the poles of the system.*

Example 1.4.5. *A combustion engine cruise control*

The combustion engine cruise control model equation 1.63 is non-linear, making it impossible to create a linear state space model. In section 1.3 a linearization around a chosen working point was performed leading to equation 1.67:

$$m\frac{dv'}{dt} \approx a\ v' + bu' + bg\ \theta', \tag{1.110}$$

Matlab/Octave code	Python code

m1=20; %first mass
m2=375;% second mass
Kw=1e6; %wheel spring constant
Ks=1.3e5;%suspension spring constant
b=9800; %damping constant of the suspension
 	
A= [0 1,0,0 ; (-Ks-Kw)/m1,-b/m1, Ks/m1,b/m1 ;
    0,0,0,1; Ks/m2,b/m2,-Ks/m2,-b/m2];
B= [0;Kw/m1;0;0];
C=[1,0,0,0; 0,0,1,0];
D=[0;0];
 	
sys = ss(A,B,C,D);
 	
step(sys);
figure;
bode(sys);
damp(sys); determine damping of th system
 	
%Convert to transfer function:
[num,den]=ss2tf(A,B,C,D,1);
TF1=tf(num(1,:),den)
 	
eig(A); %Caluclate eigenvalues of A
pole(TF1) %determine the poles of TF1
	TODO

Figure 1.50: Matlab/Octave/Python program 1.50.

with a, b and bg specified in example 1.3.7. We can be rewritten this equation as a linear state space model by introducing a state vector $\mathbf{X} = [x' \;\; \frac{dx'}{dt} = v']^T$ *and an* *input vector* $\mathbf{u} = [u', \theta']$ *leading to following state space matrices:*

$$\begin{aligned}
\mathbf{A} &= \begin{bmatrix} 0 & 1 \\ 0 & \frac{a}{m} \end{bmatrix} \\
\mathbf{B} &= \begin{bmatrix} 0 & 0 \\ \frac{b}{m} & \frac{bg}{m} \end{bmatrix}
\end{aligned}$$

The Matlab program 1.51 creates the state space model and determines the step response. Additional the Bode diagram is shown, the damping factor is calculated and it is shown that the poles of the **A** *matrix are equal to the poles of the system.*

Example 1.4.6. *DC motor model*

TODO

Matlab/Octave code

```
gears = [40, 25, 16, 12, 10]; % gear ratios
m=1600;          % mass of car kg
Tm = 190;        % engine torque constant, Nm
wm = 420;        % peak torque rate, rad/sec
bbeta = 0.4;     % torque coefficient
Cr = 0.01;       % coefficient of rolling friction
rho = 1.3;       % density of air, kg/m^3
Cd = 0.32;       % drag coefficient
A = 2.4;         % car area, m^2
g = 9.8;         % gravitational constant

%%% Linearisation points:
vref=20;            %reference value for velocity m/s
v_e=20;             %equilibrium velocity m/s
theta_e=0;          %equilibrium slope deg
u_e=0.1616;         %equilibrium throttle
gear=4;             %gear

%Compute the trottle u_e required to keep velocity v_e
%at slope theta_e, velocity v_e, and gear an
an=gears(gear);
A0=Tm*an*(1-bbeta*(an*v_e/wm-1)^2);
A1=m*g*Cr+rho*Cd*A*v_e^2/2+m*g*sin(theta_e);
u_e=A1/A0;

%Compute linearisation parameters:
w=an*v_e;
T=Tm*(1-bbeta*(w/wm-1)^2);
pT=-2*bbeta*Tm*(w/wm-1)/wm;
a=(an^2*u_e*pT-rho*Cd*A*v_e)/m;
b=an*T/m;
bg=g*cos(theta_e);

%%% two state X=[x v] and two input U=[u, theta]
A = [ 0,1;0,a];
B =[ 0 0 ;b -bg ];
C =[1 0];  %position output
D=[0,0];

sys = ss(A,B,C,D);

step(sys,5); % simulate 5 seconds
```

Python code

```
TODO
```

Figure 1.51: Matlab/Octave/Python program 1.51.

1.5 Discrete-time systems: discrete domain analysis

1.5.1 Introduction

In recent years there has been a rapid increase in the use of digital controllers in control systems. Digital controls are used for achieving optimal performance, like for example maximum productivity, profit or minimum cost, or energy use. **Decision-making capability** and **flexibility** in the control program are major advantages of digital control systems. The current trend toward **digital rather than analog control** for dynamic systems is mainly due to the availability of **low-cost digital computers** and the advantages found in working with digital signals rather than continuous-time signals,

Types of Signals: continuous-time, analog signal, discrete-time, sampled-data, digital

A **continuous-time signal** is a signal defined over a continuous range of time. The amplitude may assume a continuous range of values or may assume only a finite number of distinct values. The process of representing a variable by a set of distinct values is called **quantization**, and the resulting distinct values are called quantized values. The quantized variable changes only by a set of distinct steps.
An **analog signal** is a signal defined over a continuous range of time whose amplitude can assume a continuous range of values. Figure 1.52(a) shows a continuous-time analog signal, and Figure 1.52(b) shows a continuous-time quantized signal (quantized in amplitude only).
Be aware that the analog signal is a special case of the continuous-time signal. In practice however the terms continuous-time and analog are frequently interchanged although they are strictly not the same.

A **discrete-time signal** is a signal defined only at discrete instants of time (in which the independent variable t is quantized). In a discrete-time signal, if the amplitude can assume a continuous range of values, then the signal is called **a sampled-data signal**. A sampled-data signal can be generated by sampling an analog signal at discrete instants of time. Figure 1.52(c) shows a sampled-data signal.

A **digital signal** is a discrete-time signal with quantized amplitude. Such a signal can be represented by a sequence of numbers, for example, in the form of binary numbers. Figure 1.52(d) depicts a digital signal. Clearly it is a signal quantized both in amplitude and in time. The use of the digital controller requires quanti-

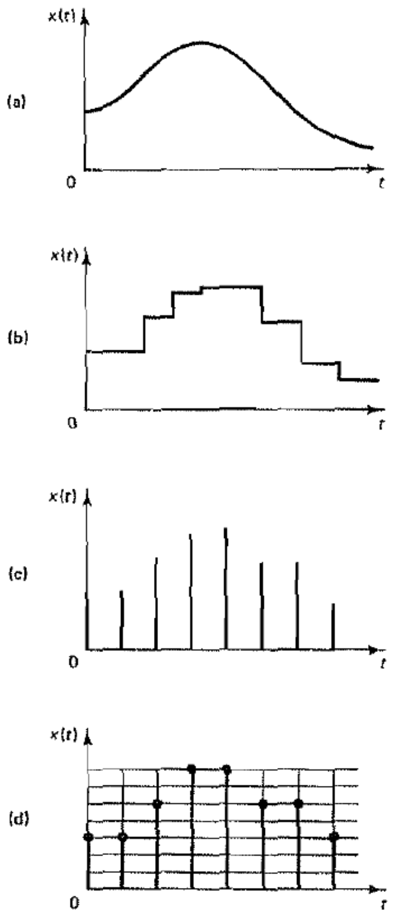

Figure 1.52: (a) Continuous-time analog signal; (b) continuous-time quantized signal; (c) sampled-data signal; (d) digital signal.

zation of signals both in amplitude and in time.

In control engineering, the controlled object is a plant or process. It may be a physical plant or process or a nonphysical process such as an economic process. Most plants and processes involve continuous-time signals; therefore, if digital controllers are involved in the control systems, **signal conversions** (analog to digital and digital to analog) become necessary. Standard techniques are available for such signal conversions; we shall discuss them in Section 1.6.7.

Many industrial control systems include continuous-time signals, sampled-data signals, and digital signals. Therefore, in this course we use the term **'discrete-**

time control systems' to describe the control systems that include some forms of sampled-data signals (amplitude-modulated pulse signals) and/or digital signals (signals in numerically coded form).

Only linear, time-invariant systems are considered

The discrete-time control systems considered in this course are mostly **linear and time invariant**, although nonlinear and/or time-varying systems are occasionally included in discussions.A **linear system** may be described by linear differential or linear difference equations. A **time-invariant** linear system is one in which the coefficients in the differential equation or difference equation do not vary with time, that is, one in which the properties of the system do not change with time.

Discrete-time control systems and continuous-time control systems.

Discrete-time control systems are control systems in which one or more variables can change only at discrete instants of time, These instants, which we shall denote by kT or $t_k(k = 0, 1, 2, \ldots)$, may specify the times at which some physical measurement is performed or the times at which the memory or a digital computer is read out. The time interval between two discrete instants is taken to be sufficiently short that the data for the time between them can be approximated by simple interpolation.

Discrete-time control systems differ from continuous-time control systems in that signals for a discrete-time control system are in sampled-data form or in digital form. If a digital computer is involved in a control system as a digital controller, any sampled data must be converted into digital data.

Continuous-time systems, whose signals are continuous in time, may be described by differential equations. Discrete-time systems, which involve sampled- data signals or digital signals and possibly continuous-time signals as well, may be described by **difference equations** after the appropriate discretization of continuous-time signals.

1.5.2 Discrete-time signals

A discrete-time signal is a sequence or a series of signal values defined in discrete points of time, see Figure 1.53. These discrete points of time can be denoted t_k where k is an integer time index. The distance in time between each point of time is the time-step, which can be denoted h. Thus,

$$h = t_k - t_{k-1} \tag{1.111}$$

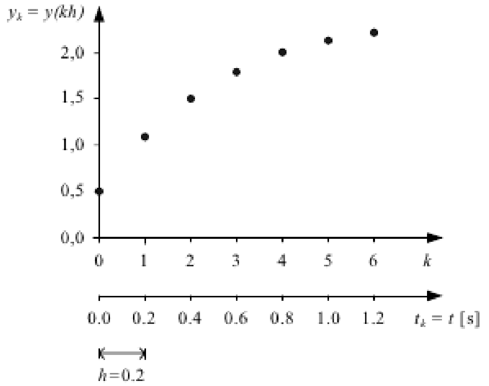

Figure 1.53: Discrete-time signal

The time series can be written in various ways:

$$\{x(t_k)\} = \{x(kh)\} = \{x[k]\} = x[0], x[1], x[2], \ldots \tag{1.112}$$

To make the notation simple, we can write the signal as x(t_k) or x(k). Examples of discrete-time signals are logged measurements, the input signal to and the output signal from a signal filter, the control signal to a physical process controlled by a computer, and the simulated response for a dynamic system.

1.5.3 Digital control systems

Figure 1.54 depicts a block diagram of a digital control system showing a configuration of the basic control scheme. The system includes feedback and feedforward control.

Signals in a digital control system.

Figure 1.55 shows a block diagram of a digital control system. The basic elements of the system are shown by the blocks. The controller operation is controlled by the clock. In such a digital control system, some points of the system pass signals of varying amplitude in either continuous time or discrete time, while other points pass signals in numerical code, as depicted in the figure.
The **output** of the plant is a continuous-time signal. The **error signal** is converted into digital form by the sample-and-hold circuit and the analog-to-digital converter. The conversion is done at the sampling time.

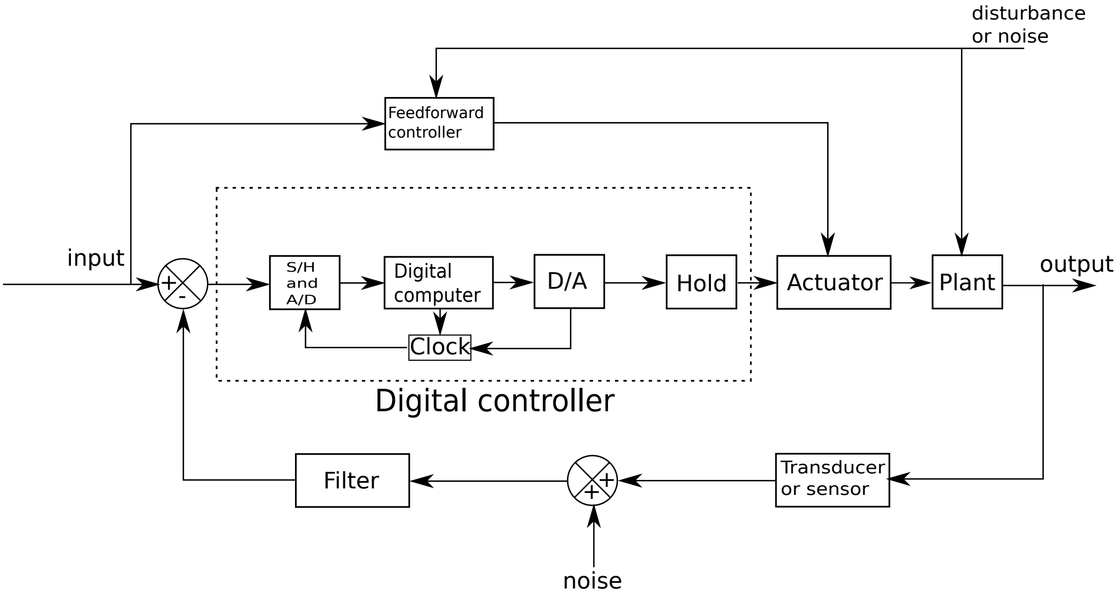

Figure 1.54: Block diagram of a digital control system.

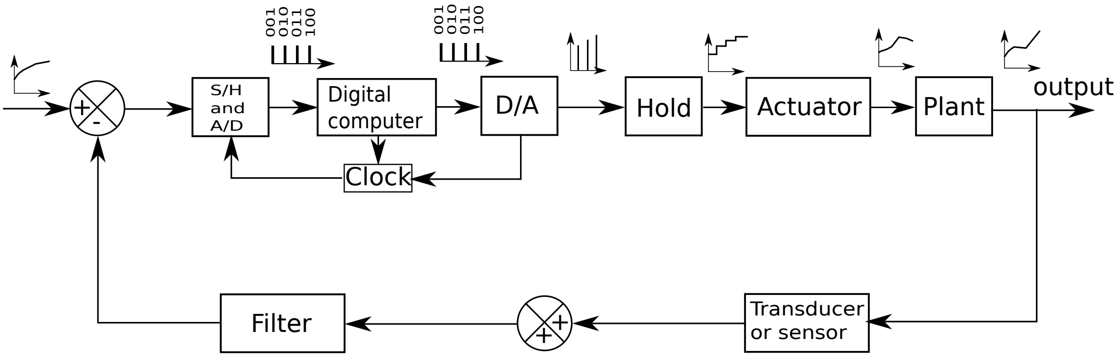

Figure 1.55: Block diagram of a digital control system showing signals in binary or graphic form.

The **digital computer** processes the sequences of numbers by means of an algorithm and produces new sequences of numbers. At every sampling instant a coded number (usually a binary number consisting of eight or more binary digits) must be converted to a physical control signal, which is usually a continuous-time or analog signal. The **digital-to-analog** converter and the **hold circuit** convert the sequence of numbers in numerical code. The output of the hold circuit, a continuous-time signal, is fed to the plant, either directly or through the actuator, to control its dynamics.

Definitions of Terms.

Before we discuss digital control systems in detail, we need to define some of the terms that appear in the block diagram of Figure 1.55.

Sample-and-Hold (S/H)
"Sample-and-hold" is a general term used for a sample-and-hold amplifier. It describes a circuit that receives an analog input signal and holds this signal at a constant value for a specified period of time. Usually the signal is electrical, but other forms are possible, such as optical and mechanical.

Analog-to-Digital Converter
An analog-to-digital converter, also called an encoder, is a device that converts an analog signal into a digital signal, usually a numerically coded signal. Such a converter is needed as an interface between an analog component and a digital component. A sample-and-hold circuit is often an integral part of a commercial available A/D converter. The conversion of an analog signal into the corresponding digital signal (binary number) is an approximation, because the analog signal can take on an infinite number of values,whereas the variety of different numbers that can be formed by a finite set of digits is limited. This approximation process is called **quantization**.

Digital-to-Analog Converter
A digital-to-analog converter, also called decoder, is a device that converts a digital signal (numerically coded data) into an analog signal. Such a converter is needed as an interface between a digital component and an analog component.

Transducer.
A transducer is a device that converts an input signal into an output signal of another form, such as a device that converts at pressure signal into a voltage output.

Types of Sampling Operations.
A sampling operation is basic in transforming a continuous-time signal into a discrete-time signal. There are several different types of sampling operations of practical importance:

1. **Periodic sampling.** In this case, the sampling instants are equally spaced, or $t_k =$ kT $(k = 0, 1, 2, \ldots)$. Periodic sampling is the most conventional type of sampling operation.

2. **Multiple-order sampling.** The pattern of the t_k's is repeated periodically; that is, $t_{k+r} - t_k$ is constant for all k.

3. **Multiple-rate sampling.** In a control system having multiple loops, the largest time constant involved in one loop may be quite different from that in other loops. Hence, it may be advisable to sample slowly in a loop involving a large time constant, while in a loop involving only small time constants the sampling rate must be fast. Thus, a digital control system may have different sampling periods in different feedback paths or may have multiple sampling rates.

4. **Random sampling.** In this case, the sampling instants are random, or t_k is a random variable.

In this course we shall treat only the case where the sampling is periodic.

1.5.4 Data acquisition, conversion and distribution systems

With the rapid growth in the use of digital computers to perform digital control actions, both the data-acquisition system and the distribution system have become an important part of the entire control system. The signal conversion that takes place in the digital control system involves the following operations:

1. **Multiplexing and demultiplexing**

2. **Sample and hold**

3. **Analog-to-digital conversion(quantizing and encoding)**

4. **Digital-to-analog conversion (decoding)**

Figure 1.56(a) shows a block diagram of a data-acquisition system, and Figure 1.56(b) shows a block diagram of a data-distribution system. In the data-acquisition system the input to the system is a physical variable such as position, velocity, acceleration, temperature, or pressure. Such a **physical variable** is first converted into an electrical signal (a voltage or current signal) by a suitable **transducer**. Once the physical variable is converted into a voltage or current signal, the rest of the data-acquisition process is done by electronic means.

In Figure 1.56(a) the **amplifier** (frequently an operational amplifier) that follows the transducer performs one or more of the following functions: It amplifies the voltage output of the transducer; it converts a current signal into a volt or it buffers the signal. The **low-pass filter** that follows the amplifier attenuates the

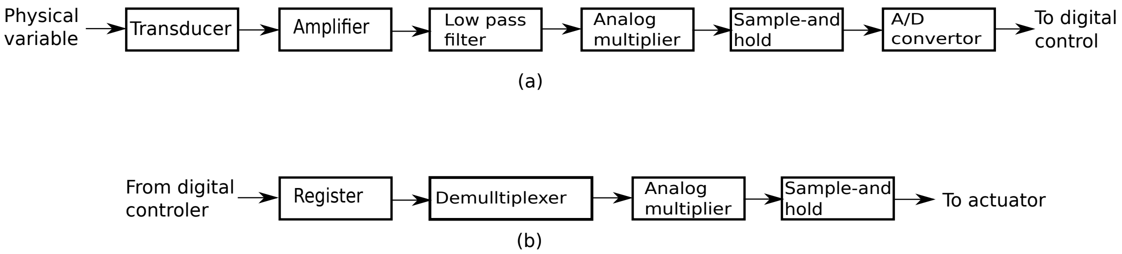

Figure 1.56: (a) Block diagram of a data acquisition system; (b) block diagram of a data-distribution system

high-frequency signal components, such as noise signals. The output of the lowpass filter is an analog signal. This signal is fed to the analog **multiplexer**. The output of the multiplexer is fed to the **sample-and-hold circuit**, whose output is, in turn, fed to the **analog-to-digital converter**. The output of the converter is the signal in digital form; it is fed to the digital controller.
The reverse of the dataacquisition process is the data-distribution process. As shown in Figure 1.56(b), a data-distribution system consists of **registers**, a **demultiplexer**, **digitalto-analog converters**, and **hold circuits**. It converts the signal in digital form (binary numbers) into analog form. The output of the D/A converter is fed to the hold circuit, The output of the hold circuit is fed to the analog actuator, which, in turn, directly controls the plant under consideration.

1.5.5 Difference equation model

The basic model type of continuous-time dynamic systems is the differential equation. Analogously, the basis model type of discrete-time dynamic systems is the difference equation.
A linear second order difference equation with u as input variable and y as output variable is defined as:

$$y[k] = -a_1 y[k-1] - a_0 y[k-2] + b_1 u[k-1] + b_0 u[k-2] \tag{1.113}$$

where a_i and b_j are coefficients of the difference equation, or model parameters. We can say that this difference equation is normalized since the coefficient of y(k) is 1 (the other coefficients then have unique values).

The difference equation 1.113 may be written in other equivalent forms. One equivalent form is

$$y[k+2] + a_1 y[k+1] + a_0 y[k] = b_1 u[k+1] + b_0 u[k]$$

where there are no time delayed terms, only time advanced terms (or terms without any advance or delay). This form can be achieved by increasing each time index by 2.

Example 1.5.1. *A low-pass filter as a difference equation*
The following difference equation implements a discrete-time low-pass filter.

$$y[k] = ay[k-1] + (1-a)u[k]$$

with 'a' a filter parameter.

Example 1.5.2. *The following difference equation implements a discrete-time PI (proportional+integral) controller. K_p and T_i are controller parameters.*

$$u[k] = u[k-1] + k_p(1 + \frac{h}{T_i}e[k] - K_p e[k-1])$$

where u is the control signal generated by the controller, and e is the control error (which is the difference between the set-point and the process measurement). K_p and T_i are controller parameters, and h is the sampling interval.

Example 1.5.3. *Given the following model of a first order system in the form of a differential equation:*

$$\dot{y}(t) = \frac{1}{T}y(t) + \frac{K}{T}u(t)$$

u is the input, y the output, K the gain and T the time constant. Applying the Euler forward method for numerical solution (see Section 1.6.7) of this differential equation yields the following difference equation:

$$y[k] = \left(1 - \frac{h}{T}\right)y[k-1] + \frac{Kh}{T}u[k-1]$$

where h is the discrete time-step.

1.5.6 Calculating responses

Calculating dynamic responses from difference equations

A difference equation is actually itself representation or formula for describing responses in the form of time functions.

Example 1.5.4. *Calculating the dynamic responses for a difference equation*
See Example 1.5.3 Assume the following parameter values:

$$h = 0.1,\ T = 1,\ K = 2$$

The difference equation becomes

$$\begin{aligned} y[k] &= (1-\frac{0.1}{1})y[k-1]+\frac{2\times 0.1}{1}u[k-1] \\ &= 0.9y[k-1]+0.2u[k-1] \end{aligned} \tag{1.114}$$

Assume that u is a step of amplitude U at discrete time $k = 0$, and that the initial value of y is y_0 . From 1.114 we can calculate the first two response in y as follows:

$$\begin{aligned} y[1] &= 0.9y(0)+0.2u(0) \\ &= 0.9y_0+0.2U \\ y[2] &= 0.9y(1)+0.2u(1) \\ &= 0.9[0.9y_0+0.2U]+0.2\times 0 \\ &= 0.81y_0+0.18U \\ y[3] &= \ldots \end{aligned}$$

1.5.7 Block diagram of difference equation models

A block diagram gives a graphical representation of a mathematical model. The block diagram shows the structure of the model, e.g. how subsystems are connected. Furthermore, block diagram models can be represented directly in graphical simulation tools such as Matlab-Simulink or LabVIEW.
Figure 1.57 shows the most frequently used blocks - or the elementary blocks - used in block diagrams of difference equation models.
A comment about the time delay block: the output y(k-1) is equal to the time delayed input, y(k):

$$y[k-1] = z^{-1}y[k] \tag{1.115}$$

The operator z^{-1} is a time-step delay operator, and it can be regarded as an operator of the time-step delay. More information about this operator will be given in the next section.

Example 1.5.5. *Example 1.5.1 is a low-pass filtering algorithm. It is repeated here:*

$$y[k] = ay[k-1]+(1-a)u[k] \tag{1.116}$$

Using the elementary blocks shown in Figure 1.57 a block diagram of this difference equation can be drawn as shown in Figure 1.58

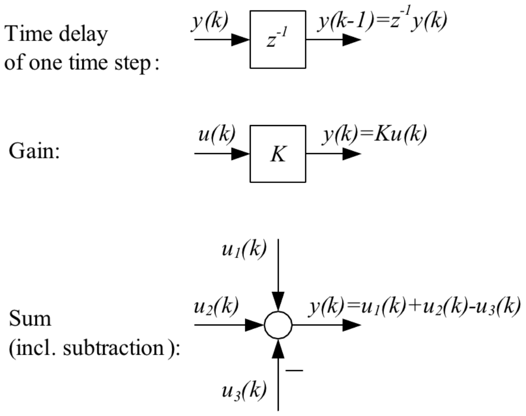

Figure 1.57: Elementary blocks for drawing block diagrams of difference equation models.

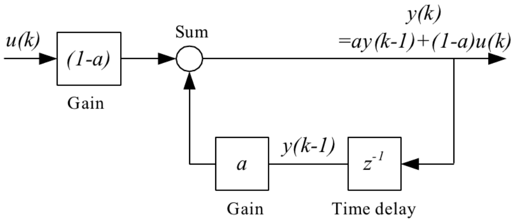

Figure 1.58: The block diagram corresponding with equation 1.116.

1.6 Discrete-time domain: the Z-transform

1.6.1 Introduction

A mathematical tool commonly used for the analysis and synthesis of discrete-time control systems is the Z-transform. The role of the Z transform in discrete-time systems is similar to that of the Laplace transform in continuous-time systems.

In a linear discrete-time control system, a linear difference equation characterizes the dynamics of the system. To determine the system response to a given input, such a difference equation must be solved. With the Z-transform method, the solutions to linear difference equations become algebraic.

The main objective of this chapter is to present definitions of the Z-transform, basic theorems associated with the Z-transform, and methods for finding the inverse Z-transform. Solving difference equations by the Z-transform method is also discussed.

1.6.2 The Z-transform

The Z-transform method is an operational method that is very powerful when working with discrete-time systems. In what follows we shall define the Z-transform of a time function or a number sequences.

In considering the Z-transform of a time function $x(t)$, we consider only the sampled values of x(t), that is, $x(O), x(T), x(2T), \ldots$, where T is the sampling period. The Z-transform of a time function x(t), where t is nonnegative, or of a sequence of values x(kT), where k takes zero or positive integers and T is the sampling period, is defined by the following:

$$X(z) = \mathcal{Z}[x(t)] = \mathcal{Z}[x(kT)] = \sum_{k=0}^{\infty} x(kT)z^{-k} \tag{1.117}$$

For a sequence of numbers x(k), the Z-transform is defined by

$$X(z) = \mathcal{Z}[x[k]] = \sum_{k=0}^{\infty} x[k]z^{-k} \tag{1.118}$$

The Z-transform defined by Equation 1.117 or 1.118 is referred to as the one-sided Z-transform.
The symbol $\mathcal{Z}$ denotes "the Z-transform of". In the one-sided Z-transform, we assume x(t)=0 for $t < 0$ or x(k) =0 for $k < 0$.

The Z-transform of x(t), where $-\infty < t < \infty$, or of x(k), where k takes integer values ($k = 0, \pm1, \pm2, \ldots$), is defined by

$$X(z) = \mathcal{Z}[x(t)] = \mathcal{Z}[x(kT)] = \sum_{k=-\infty}^{\infty} x(kT)z^{-k} \tag{1.119}$$

or

$$X(z) = \mathcal{Z}[x[k]] = \sum_{k=-\infty}^{\infty} x[k]z^{-k} \tag{1.120}$$

The Z-transform defined by Equation 1.119 or 1.120 is referred to as the twosided Z-transform, the time function x(t) is assumed to be nonzero for $t < 0$ and the sequence x(k) is considered to have non-zero values for $k < O$. Both the one-sided and two sided Z-transforms are series in powers of z". (The latter involves both positive and negative powers of z".) In this course, only the one-sided Z-transform is considered in detail.

For engineering applications the one-sided Z-transform will have a convenient closed-form solution in its **region of convergence**. [Note that whenever X (z), an infinite series in z^{-1}, converges outside the circle $|z| = R$, where R is called the **radius of absolute convergence**, in using the Z-transform method for solving discrete time problems it is not necessary each time to specify the values of Z-over which X(z) is convergent.

Notice that expansion of the right-hand side of Equation 1.117 gives

$$X(z) = x(O) + x(T)z^{-1} + x(2T)z^{-2} + \ldots + x(kT)z^{k} + \ldots \tag{1.121}$$

Equation 1.121 implies that the Z-transform of any continuoustime function x(t) may be written in the series form by inspection. The z^{-k} in this series indicates the position in time at which the amplitude $x(kT)$ occurs. Conversely, if X(z) is given in the series form as above, the inverse Z-transform can be obtained by inspection as a sequence of the function x(kY) that corresponds to the values of x(t) at the respective instants of time.

If the Z-transform is given as a ratio of two polynomials in z, then the inverse Z-transform may be obtained by several different methods, such as the **direct division method**, the computational method, the partial-fraction-expansion method (see Section 1.6.5 for details).

1.6.3 The Z transforms of elementary functions

In this section we shall present Z-transforms of several elementary functions. It is noted that in one-sided Z-transform theory, in sampling a discontinuous function x(t), we assume that the function is continuous from the right; that is, if discontinuity occurs at $t = O$, then we assume that $x(O)$ is equal to $x(0+)$ rather than to the average at the discontinuity, $[x(0) + x(0+)]/2$.

Unit-Step Function

Let us find the Z-transform of the unit-step function

$$step(t) = \begin{cases} 1, & 0 \leq t \\ 0, & t < 0 \end{cases} \tag{1.122}$$

As just noted, in sampling a unit step function we assume that this function is continuous from the right; that is, $1(0) = 1$. Then, referring to Equation 1.117, we have

$$\begin{aligned} X(z) &= \mathcal{Z}[1(t)] = \sum_{k=0}^{\infty} 1z^{-k} = \sum_{k=0}^{\infty} z^{-k} \\ &= 1 + z^{-1} + z^{-2} + \dots \\ &= \frac{1}{1-z^{-1}} \\ &= \frac{z}{z-1}. \end{aligned}$$

To obtain this result we applied the formula of the geometric series (when $\|r\| < 1$) : $\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$. Notice that the series converge if $\|z\| > 1$. In finding the Z-transform, the variable Z-acts as a dummy operator. It is necessary to specify the region of Z over which X(z) is convergent. It suffices to know that such a region exists. The Z-transform X(z) of a time function x(t) obtained in this way is valid throughout the Z-plane except at poles of X(z).

Unit-Ramp Function

Consider the unit-ramp function

$$ramp(t) = \begin{cases} t, & 0 \le t \\ 0, & t < 0 \end{cases} \tag{1.123}$$

Notice that $x(kT) = kT$ for $k = 0, 1, 2, \dots$. The Z-transform of the unit-ramp function can be written as

$$\begin{aligned} X(z) &= \mathcal{Z}[x(kT)] = \sum_{k=0}^{\infty} kTz^{-k} = T\sum_{k=0}^{\infty} kz^{-k} \\ &= T(z^{-1} + 2z^{-2} + 3z^{-3} + \dots) \\ &= Tz^{-1}(1 + z^{-1} + z^{-2} + \dots)(1 + z^{-1} + z^{-2} + \dots) \\ &= T\frac{z^{-1}}{(1-z^{-1})^2} \\ &= \frac{Tz}{(z-1)^2} \end{aligned}$$

Exponential Function

Let us find the Z-transform of

$$exp(t) = \begin{cases} e^{-at}, & 0 \le t \\ 0, & t < 0 \end{cases} \tag{1.124}$$

Since

$$x(kT) = e^{-at}, \qquad k = 0, 1, 2, \ldots$$

we have

$$\begin{aligned} X(z) &= \mathcal{Z}[e^{at}] = \sum_{k=0}^{\infty} e^{-akT} z^{-k} \\ &= 1 + e^{-aT} z^{-1} + e^{-2aT} z^{-2} + e^{-3aT} z^{-3} + \ldots \\ &= \frac{1}{1 - e^{-aT} z^{-1}} \\ &= \frac{z}{z - e^{-aT}} \end{aligned}$$

Example 1.6.1. *Obtain the Z-transform of*

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

Whenever a function in s is given, one approach for finding the corresponding Z-transform is to convert X(s) into x(t) and then find the Z-transform of x(t). Another approach is to expand X(s) into partial fractions and use a Z-transform table to find the Z-transforms of the expanded terms. Other approaches will be discussed in Section 1.6.5.

The inverse Laplace transform of $X(s)$ is

$$x(t) = 1 - e^{-t}, \quad 0 \leq t \tag{1.125}$$

Hence

$$\begin{aligned} X(z) &= \mathcal{Z}[1 - e^{-t}] = \frac{1}{1 - z^{-1}} - \frac{1}{1 - e^{-T} z^{-1}} \\ &= \frac{(1 - e^{-T}) z^{-1}}{(1 - z^{-1})(1 - e^{-T} z^{-1})} \\ &= \frac{(1 - e^{-T}) z}{(z - 1)(z - e^{-T})} \end{aligned}$$

f[k]	$\mathcal{Z}$ { f(t) }
$\delta(kT)$	1
$\delta(n-kT)$	z^{-k}
$1(kT)$	$\frac{z}{z-1}$
kT	$\frac{Tz}{(z-1)^2}$
$(kT)^2$	$\frac{T^2z(z+1)}{(z-1)^3}$
$(kT)^3$	$\frac{T^3z^2(z^2+4z+1)}{(z-1)^4}$
e^{-akT}	$\frac{z}{z-e^{-aT}}$
$1-e^{-akT}$	$\frac{(1-e^{-aT})z}{(z-1)(z-e^{-aT})}$
$\sin(\omega kT)$	$\frac{Z-\sin\omega T}{z^2-2z\cos\omega T+1}$
$\cos(\omega kT)$	$\frac{z(z-\cos\omega T)}{z^2-2z\cos\omega T+1}$

Table 1.1: A list of the most frequently used Z-transformed discrete-time functions.

1.6.4 Properties of the Z-transform

The use of the Z-transform method in the analysis of discrete-time control systems may be facilitated if theorems of the Z-transform are referred to. In this section we present important properties and useful theorems of the Z-transform. We assume that the time function x(t) is Z-transformable and that x(t) is zero for $t < 0$.

Multiplication by a constant

If X(z) is the Z-transform of x(t), then

$$\mathcal{Z}[ax(t)] = a\mathcal{Z}[x(t)] = aX(z)$$

where a is a constant. To prove this, note that by definition

$$\mathcal{Z}[ax(t)] = \sum_{k=0}^{\infty} ax(kT)z^{-k} = a\sum_{k=0}^{\infty} x(kT)z^{-k} = aX(z)$$

Linearity of the Z-transform.

The Z-transform possesses an important property: linearity. This means that, if f(k) and g(k) are Z-transformable and α and β are scalars, then x(k) formed by a linear combination

$$x(k) = \alpha f(k) + \beta g(k)$$

has the Z-transform

$$X(z) = \alpha F(z) + \beta G(z)$$

where F(z) and G(z) are the Z-transforms of f(k) and g(k), respectively.

The linearity property can be proved by referring to Equation 1.118 as follows:

$$\begin{aligned} X(z) &= \mathcal{Z}[x(k)] = \mathcal{Z}[\alpha f(k) + \beta g(k)] \\ &= \sum_{k=0}^{\infty}[\alpha f(k) + \beta g(k)]z^{-k} \\ &= \alpha\sum_{k=0}^{\infty} f(k)z^{-k} + \beta\sum_{k=0}^{\infty} g(k)z^{-k} \\ &= \alpha\mathcal{Z}[f(k)] + \beta\mathcal{Z}[g(k)] \\ &= \alpha F(z) + \beta G(z) \end{aligned}$$

Shifting theorem

The shifting theorem presented here is also referred to as the real translation theorem. If x(t) = 0 for $t < 0$ and x(t) has the Z-transform X(z),then

$$\mathcal{Z}[x(t-nT)] = z^{-n}X(z) \tag{1.126}$$

and

$$\mathcal{Z}[x(t+nT)] = z^{n}\left[X(z) - \sum_{k=0}^{n-1} x(kT)z^{-k}\right] \tag{1.127}$$

where n is a zero or positive integer. To prove equation 1.126,

$$\begin{aligned} \mathcal{Z}[x(t-nT)] &= \sum_{k=0}^{\infty} x(kT-nT)z^{-k} \\ &= z^{-n}\sum_{k=0}^{\infty} x(kT-nT)z^{-(k-n)} \qquad (1.128) \end{aligned}$$

by defining $m = k - n$, Equation 1.128 can be rewritten as:

$$\mathcal{Z}[x(t-nT)] = z^{-n}\sum_{m=-n}^{\infty} x(mT)z^{-m}$$

Since x(mT)=0 for $m < 0$, we may change the lower limit of the summation from $m = -n$ to $m = 0$. Hence,

$$\mathcal{Z}[x(t-nT)] = z^{-n}\sum_{m=0}^{\infty} x(mT)z^{-m} = z^{-n}X(z)$$

The number sequence x(k), Equation 1.127, we obtain

$$\begin{aligned} \mathcal{Z}[x(k+1)] &= zX(z) - Z - x(0) \qquad (1.129) \\ \mathcal{Z}[x(k+2)] &= Z - \mathcal{Z}[x(k+1)] - Z - x(1) = z^2X(z) - z^2x(0) - zx(1) \end{aligned}$$

Similarly

$$\mathcal{Z}[x(k+n)] = z^nX(z) - z^nx(0) - z^{n-1}x(1) - z^{n-2}x(2) - \ldots - zx(n-1) \tag{1.130}$$

where n is a positive integer.

Example 1.6.2. *Find the Z-transforms of unit-step functions that are delayed by 1 sampling period and 4 sampling periods, respectively. Using the shifting theorem given by Equation 1.126, we get*

$$\mathcal{Z}[step(t-T)] = z^{-1}\mathcal{Z}[step(t)] = \frac{z^{-1}}{1-z^{-1}}$$

And for a delay of 4 periods we gather

$$\mathcal{Z}[step(t-4T)] = z^{-4}\mathcal{Z}[step(t)] = \frac{z^{-4}}{1-z^{-1}}$$

Note that z^{-1} represents a delay of 1 sampling period T, regardless of the value T !!!

Initial Value Theorem

If x(t) has a Z-transform equal to X(z) and if $\lim_{Z-\rightarrow\infty} X(z)$ exists, then the initial value x(0) of x(t) or x[k] is given by :

$$x(0) = \lim_{Z-\rightarrow\infty} X(z) \tag{1.131}$$

The initial value theorem is convenient for checking Z-transform calculations for possible errors. Since x(O) is usually known, a check of the initial value by $\lim_{Z-\rightarrow\infty} X(z)$ can easily spot errors in X (z), if any exist.

Final Value Theorem

Assume that x[k], where x[k] = 0 for $k < O$, has the Z-transform X(z) and that all the poles of X(z) lie inside the unit circle, with the possible exception of a simple pole at $z- = 1$. This is the condition for the stability of X(z), or the condition for x[k] (k = 0, 1, 2, ...) to remain finite. Then the final value of x[k], that is, the value of x(k) as k approaches infinity, can be given by

$$\lim_{k\rightarrow\infty} x[k] = \lim_{z-\rightarrow 1}\left[(1-z^{-1})X(z)\right] \tag{1.132}$$

the final value theorem is very useful in determining the behavior of x[k] as $k \rightarrow \infty$ from its Z-transform X(z).

Example 1.6.3. *Determine the final value of $x(\infty)$ of*

$$X(z) = \frac{1}{1-z^{-1}} - \frac{1}{1-e^{-aT}z^{-1}}, \quad a > 0$$

by using the final value theorem. We obtain using Equation 1.132,

$$\begin{aligned}
x(\infty) &= \lim_{z-\rightarrow 1}\left[(1-z^{-1})X(z)\right] \\
&= \lim_{z-\rightarrow 1}\left[(1-z^{-1})\left(\frac{1}{1-z^{-1}} - \frac{1}{1-e^{-aT}z^{-1}}\right)\right] \\
&= \lim_{z-\rightarrow 1}\left[1 - \frac{1-z^{-1}}{1-e^{aT}z^{-1}}\right]
\end{aligned}$$

Name	Time domain	Z-domain
Shift	$x(k-n)step(k)$	$z^{-n}X(z)+\frac{1}{z^n}\sum_{k=1}^{n}x[-k]z^k$
Shift	$x(k+n)step(k)$	$z^{n}X(z)-z^n\sum_{k=1}^{n}x[k]z^{-k}$
Initial value	$\lim\limits_{k\to 0^+} x(k)$	$\lim\limits_{Z-\to 0} X(z)$
Final value	$\lim\limits_{k\to\infty} x(k)$	$\lim\limits_{Z-\to 1} \frac{z-1}{z}X(z)$

Table 1.2: List of important properties of the Z-transform.

1.6.5 The inverse Z-transform

The Z-transformation serves the same role for discrete-time control systems that the Laplace transformation serves for continuous-time control systems. For the Z-transform to be useful, we must be familiar with methods for finding the inverse Z-transforms.

The notation for the inverse Z-transform is $\mathcal{Z}^{-1}$. The inverse Z-transform of X(z) yields the corresponding time sequence x(k).

It should be noted that only the time sequence at the sampling instants is obtained from the inverse Z-transform. Thus, the inverse Z-transform of X(z) yields a unique x[k], but does not yield a unique x(t), This means that the inverse Z-transform yields a time sequence that specifies the values of x(t) only at discrete instants of time, $t \approx 0, T, 2T, \ldots$, and says nothing about the values of x(t) at all other times, That is, many different time functions x(t) can have the same x(kT).

When X(z), the Z-transform of x(kT) or x[k], is given, the operation that determines the corresponding x(kT) or x[k] is called **the inverse Z-transformation**. An obvious method for finding the inverse Z-transform is to refer to a Z-transform table. However, unless we refer to an extensive Z-transform table, we may not be able to find the inverse Z-transform of a complicated function of z, (If we use a less extensive table of Z-transforms, it is necessary to express a complex Z-transform as a sum of simpler Z-transforms. Refer to the partial-fraction-expansion method presented in this section.)

Other than referring to 2 transform tables, four methods for obtaining the inverse Z-transform are commonly available:

1. Direct division method

2. Computational method
3. Partial-fraction-expansion method
4. Inversion integral method (not discussed in this course)

Direct division method

In the direct division method we obtain the inverse Z-transform by expanding $X(z)$ into an infinite power series in z^{-1}. This method is useful when it is difficult to obtain the closed-form expression for the inverse Z-transform or it is desired to find only the first terms of x[k].

The direct division method originates from the fact that if X(z) is expanded into a power series in z^{-1}, that is, if

$$\begin{aligned} X(z) &= \sum_{k=0}^{\infty} x(k) z^{-k} \\ &= x(0) + x(1)z^{-1} + x(2)z^{-2} + \ldots + x(k)z^{-k} + \ldots \end{aligned}$$

then x(kT) or x[k] is the coefficient of the z^{-k} term. Hence, the values of x(kT) or x[k] for $k = 0, 1, 2, \ldots$ can be determined by inspection.

if X(z) is given in the form of a rational function, the expansion into an infinite power series in increasing powers of z^{-1} can be accomplished by simply dividing the numerator by the denominator, where both the numerator and denominator of $X(z)$ are written in increasing powers of z^{-1}. If the resulting series is convergent, the coefficients of the z^{-k} term in the series are the values x(kT) of the time sequence or the values of x[k] of the number sequence.

Although the presented method gives the values of $x(0), x(T), x(2T), \ldots$ or the values of $x(0), x(1), x(2), \ldots.$ in a sequential manner, it is usually difficult to obtain an expression for the general term from a set of values of x(kT) or x[k].

Example 1.6.4. *Find x[k] for $k = 0, 1, 2, 3, 4$ when X(z) is given by*

$$X(z) = \frac{10z + 5}{(z-1)(z-0.2)}$$

First, rewrite X(z) as a ratio of polynomials in z^{-1}, as follows:

$$X(z) = \frac{10z^{-1} + 5z^{-2}}{1 - 1.2z^{-1} + 0.2z^{-2}}$$

$$
\begin{array}{r l}
 & \qquad\qquad\quad \underline{10z^{-1} + 17z^{-2} + 18.4z^{-3} + 18.68z^{-4} + \dots} \\
1 - 1.2z^{-1} + 0.2z^{-2}) & 10z^{-1} + 5z^{-2} \\
 & \underline{10z^{-1} - 12z^{-2} + 2z^{-3}} \\
 & \qquad\quad 17z^{-2} - 2z^{-3} \\
 & \qquad\quad \underline{17z^{-2} - 20.4z^{-3} + 3.4z^{-4}} \\
 & \qquad\qquad\qquad 18.4z^{-3} - 3.4z^{-4} \\
 & \qquad\qquad\qquad \underline{18.4z^{-3} - 22.08z^{-4} + 3.68z^{-5}} \\
 & \qquad\qquad\qquad\qquad\quad 18.68z^{-4} - 3.68z^{-5} \\
 & \qquad\qquad\qquad\qquad\quad \underline{18.68z^{-4} - 22.416z^{-5} + 3.736z^{-6}}
\end{array}
$$

Computational method

In what follows, we present two computational approaches to obtain the inverse Z-transform:

1. **Software approach**
2. **Difference equation approach**

Example 1.6.5. *Consider a system $G(z)$ defined by*

$$G(z) = \frac{0.4673z^{-1} - 0.3393z^{-2}}{1 - 1.5327z^{-1} + 0.6607z^{-2}} \tag{1.133}$$

If we assume the input to this system is a Kronecker delta input $\delta(kT)$ that is 1 if k=0 and otherwise 0. The Z-transform of the Kronecker delta input is X(z)=1. This way the transfer function in Equation 1.133 represents an input-output relation with a Kronecker delta input:

$$G(z) = \frac{Y(z)}{X(z)} = \frac{0.4673z^{-1} - 0.3393z^{-2}}{1 - 1.5327z^{-1} + 0.6607z^{-2}} = \frac{0.4673z - 0.3393}{z^2 - 1.5327z + 0.6607} \tag{1.134}$$

Software Approach.

Software can be used to find the inverse Z-transform. Referring to Equation 1.134, the input X(z) is the Z-transform of the Kronecker delta input. In software the Kronecker delta input is given by

```
x = [1,zeros(1,N)]
```

where N corresponds to the number of discretetime steps of the process considered. Since the Z-transform of the Kronecker delta input X(z) is equal to unity, the response of the system to this input is

$$Y(z) = \frac{0.4673z - 0.3393}{z^2 - 1.5327z + 0.6607}$$

Hence the inverse Z-transform of G(z) is given by $y(0), y(1), y(2), \ldots$. Let us obtain y(k) up to k = 40.

To obtain the inverse Z-transform of G(z) with software, we proceed as follows: Enter the numerator and denominator as follows:

```
num = [0.4673 -0.3393]
den = [1 -1.5327 0.6607]
```

Then specify the discrete transfer function in the Z-domain

```
G = tf(num,den,1)
```

with 1 a chosen sampling time that is currently not relevant but necessary for the software to know we are defining a discrete transfer function ! The response is obtained by applying the 'impulse' function

```
y = impulse(G,inp,Tstop)
```

here G represent the discrete system, inp is the index of the input that is being excited (this is only relevant when the system G has multiple outputs like with a state space model) and Tstop is the stop time for the simulation. The complete program together with the Matlab outpur is shown in Figure 1.59.

Difference equation approach.

Note that Equation 1.134 can be written as

$$(z^2 - 1.5327z + 0.6607)Y(z) = (0.4673z - 0.3393)X(z)$$

we can convert this equation into the difference equation as follows:

$$y[k+2] - 1.5327y[k+1] + 0.6607y[k] = 0.4673x[k+1] - 0.3393x[k]$$

where x[0]=1 and x[k]=0 for $k < 0$, and y[k]=0 for $k < 0$.

The initial data y[0] and y[] can be determined as follows: By substituting $k = -2$ into Equation 1.134, we find]

$$y[0] - 1.5327y[-1] + 0.6607y[-2] = 0.4673x[-1] - 0.3393x[-2]$$

from which we get

$$y[0] = 0$$

Next we substitute $k = -1$ into Equation 1.135, we obtain

$$y[1] - 1.5327y[0] + 0.6607y[-1] = 0.4673x[0] - 0.3393x[-1]$$

from which we gather

$$y(1) = 0.4673.$$

You can continue this procedure to find all the solution for y[k] wit $0 \leq k$.

<table>
<tr><th>Matlab/Octave code</th><th>Python code</th></tr>
<tr><td><pre>
>>num = [0 ,0.4673 ,-0.3393];
>>den = [1 ,-1.5327, 0.6607] ;
>>sys=tf(num,den,1);
>>y=impulse(sys,1,40)
y =
 Columns 1 through 5:
   0.00000   0.46730   0.37693   0.26898   0.16322
 Columns 6 through 10:
   0.07246   0.00322 -0.04294 -0.06795 -0.07577
 Columns 11 through 15:
  -0.07124 -0.05912 -0.04355 -0.02769 -0.01367
 Columns 16 through 20:
  -0.00265   0.00497   0.00936   0.01107   0.01078
 Columns 21 through 25:
   0.00921   0.00699   0.00463   0.00248   0.00074
 Columns 26 through 30:
  -0.00050 -0.00126 -0.00160 -0.00162 -0.00142
 Columns 31 through 35:
  -0.00111 -0.00077 -0.00044 -0.00017   0.00004
 Columns 36 through 40:
   0.00016   0.00023   0.00024   0.00022   0.00018
 Column 41:
   0.00013
</pre></td><td><pre>
TODO
</pre></td></tr>
</table>

Figure 1.59: Matlab/Octave/Python program to obtain the inverse z-transform

Partial-fraction-expansion method

The partial-fraction-expansion method presented here, which is parallel to the partial-fraction-expansion method used in Laplace transformation, is widely used in routine problems involving Z-transforms.

The method requires that all terms in the partial fraction expansion be easily recognizable in the table of Z-transform pairs.

To find the inverse Z-transform, if X(z) has one or more zeros at the origin (Z=

0), then X(z)/Z or X (z) is expanded into a sum of simple first- or second-order terms by partial fraction expansion, and a Z-transform table is used to find the corresponding time function of each expanded term. Be aware that the only reason that we expand X(z)/Z-into partial fractions is that each expanded term has a form that may easily be found from commonly available Z-transform tables.

To expand X(z) into partial fractions, we first factor the denominator polynomial of X(z) and find the poles of X(z):

$$X(z) = \frac{b_0 z^m + b_1 z^{m-1} + \ldots + b_{m-1} z + b_m}{(z-p_1)(z-p_2)\ldots(z-p_n)}.$$

We then expand X(z)/z into partial fractions so that each term is easily recognizable in a table of Z-transforms.

A commonly used procedure for the case where all the poles are of simple order and there is at least one zero at the origin (that is, $b_m = 0$) is to divide both sides of X(z) by z and then expand X(z)/z into partial fractions, Once X(z)/z is expanded, it will be of the form

$$\frac{X(z)}{z} = \frac{a_1}{z-p_1} + \frac{a_2}{z-p_2} + \ldots + \frac{a_n}{z-p_n}.$$

The coefficient a_i can be determined by multiplying both sides of this last equation by $z - p_i$ and setting $z = p_i$. This will result in zero for all the terms on the right-hand side except the a_i term, in which the multiplicative factor $z - p_i$, has been canceled by the denominator. Hence, we have

$$a_i = \lim_{z=p_i} \left[(z-p_i)\frac{X(z)}{z} \right]. \tag{1.135}$$

Note that such determination of a_i is only valid for simple poles.

If $\frac{X(z)}{z}$ involves a multiple pole, for example, a double pole at $Z = p_1$ and no other poles, then $\frac{X(z)}{z}$ will have the form

$$\frac{X(z)}{z} = \frac{c_1}{(z-p_1)^2} + \frac{c_2}{z-p_1}$$

The coefficients c_1 and c_2 are determined from

$$\begin{aligned} c_1 &= \lim_{z=p_1} \left[(z-p_1)^2 \frac{X(z)}{z} \right] \\ c_2 &= \lim_{z=p_1} \left\{ \frac{d}{dz} \left[(z-p_1)^2 \frac{X(z)}{z} \right] \right\} \end{aligned}$$

Example 1.6.6. *Let us obtain the inverse Z-transform of*

$$X(z) = \frac{z^2 + z + 2}{(z-1)(z^2 - z + 1)}$$

by using the partial-fraction-expansion method. We can expand X(z) into partial fractions as follows:

$$X(z) = \frac{4}{z-1} + \frac{-3z+2}{z^2-z+1} = \frac{4z^{-1}}{1-z^{-1}} + \frac{-3z^{-1}+2z^{-2}}{1-z^{-1}+z^{-2}}$$

Noting that the two poles involved in the quadratic term of this last equation are complex conjugates, we rewrite X(z) as follows:

$$\begin{aligned} X(z) &= \frac{4z^{-1}}{1-z^{-1}} + \frac{-3z^{-1}+2z^{-2}}{1-z^{-1}+z^{-2}} \\ &= 4z^{-1}\frac{1}{1-z^{-1}} + az^{-1}\frac{1-0.5z^{-1}}{1-z^{-1}+z^{-2}} + bz^{-1}\frac{0.5z^{-1}}{1-z^{-1}+z^{-2}} \end{aligned}$$

We find a an b by stating that $a(1-0.5z^{-1}) + b0.5z^{-1} = -3 + 2z^{-1}$*: a=-3 and b=1. Since*

$$\begin{aligned} \mathcal{Z}[e^{-akT}\cos(\omega kT)] &= \frac{1 - e^{aT}z^{-1}\cos(\omega T)}{1 - 2e^{-aT}z^{-1}\cos(\omega T) + e^{-2aT}z^{-2}} \\ \mathcal{Z}[e^{-akT}\sin(\omega kT)] &= \frac{e^{aT}z^{-1}\sin(\omega T)}{1 - 2e^{-aT}z^{-1}\cos(\omega T) + e^{-2aT}z^{-2}} \end{aligned}$$

by identifying $e^{-2aT} = 1$ *and* $\cos\omega T = 0.5$ *in this case, we have* $\omega T = \pi/3$ *and* $\sin\omega T = \sqrt{3}/2$*. Hence we obtain*

$$\begin{aligned} \mathcal{Z}^{-1}\left[\frac{1-0.5z^{-1}}{1-z^{-1}+z^{-2}}\right] &= 1^k \cos\frac{k\pi}{3} \\ \mathcal{Z}^{-1}\left[\frac{0.5z^{-1}}{1-z^{-1}+z^{-2}}\right] &= \mathcal{Z}^{-1}\left[\frac{1}{\sqrt{3}}\frac{(\sqrt{3}/2)z^{-1}}{1-z^{-1}+z^{-2}}\right] = \frac{1}{\sqrt{3}}1^k \sin\frac{k\pi}{3} \end{aligned}$$

Thus as final solution we get

$$x(k) = 4(1^{k-1}) - 3(1^{k-1})\cos\frac{k\pi}{3} + \frac{1}{\sqrt{3}}(1^{k-1})\sin\frac{k\pi}{3} \qquad (1.136)$$

1.6.6 Z-transform method to solve difference equations

Difference equations can be solved easily by use of a digital computer, provided the numerical values of all coefficients and parameters are given. However, closed-form expressions for x(k) cannot be obtained from the computer solution, except for very special cases. The usefulness of the Z-transform method is that it enables us to obtain the closed-form expression for x[k].

Consider the linear time-invariant discretetime system characterized by the following linear difference equation:

$$x[k]+a_1x[k-1]+\ldots+a_nx[k-n]=b_0u[k]+b_1u[k-1]+\ldots+b_nu[k-n] \quad (1.137)$$

where u[k] and x[k] are the systems input and output, respectively, at the k-th iteration. In describing such a difference equation in the Z-plane, we take the Z-transform of each term in the equation.

Let us define

$$\mathcal{Z}[x(k)] = X(z)$$

Then $x(k+1), x(k+2), x(k+3), \ldots.$ and $x(k-1), x(k-2), x(k-3), \ldots.$. May be expressed in terms of X(z) and the initial conditions. Their exact Z-transforms are given in Table 1.1 and are their properties in Table 1.2 for convenient reference.

Example 1.6.7. *Solve the following difference equation by use of the Z-transform method:*

$$x(k+2)+3x(k+1)+2x(k)=0, \quad x(0)=0, \quad x(1)=-1$$

Taking the Z-transform of x(k+2),x(k+1), and x(k), we obtain

$$z^2X(z)-z^2x(0)-zx(1)+3zX(z)-3zx(0)+2X(z)=0 \quad (1.138)$$

After substitution of the initial data and simplification we get

$$\begin{aligned} X(z) &= \frac{z}{z^2+3z+2} = \frac{z}{(z+1)} - \frac{z}{(z+2)} \\ &= \frac{1}{1+z^{-1}} - \frac{1}{1+2z^{-1}} \end{aligned}$$

Applying the table of inverse Z-transforms gives us

$$x(k)=(-1)^k-(-2)^k, \qquad k=0,1,2,\ldots$$

1.6.7 Z-plane analysis of discrete-time control systems

The Z-transform method is particularly useful for analyzing and designing single-input-single-output linear time-invariant discrete-time control systems. This chapter presents background material necessary for the analysis and design of discrete-time control systems in the Z-plane. The main advantage of the Z-transform method is that it enables the engineering to apply conventional continuous-time design methods to discrete-time systems that may be partly discrete time and partly continuous time.

1.6.8 Impulse sampling and data hold

Discrete-time control systems may operate partly in discrete time and partly in continuous time. Thus, in such control systems some signals appear as discrete-time functions (often in the form of a sequence of numbers or a numerical code) and other signals as continuous-time functions. In analyzing discrete-time control systems, the Z-transform theory plays an important role. To see why the Z-transform method is useful in the analysis of discretetime control systems, we first introduce the concept of impulse sampling and then discuss data hold.

Impulse sampling

Consider an analog signal $x_a(t)$. The sequence x[k] with values $x[k] = x_a(kT)$ is said to be derived from $x_a(t)$ by periodic sampling and T is called the sampling period. In a typical digital control scheme shown in Fig. 1.60, the operation of deriving a sequence from a continuous-time signal is performed by an A/D converter. A simple symbolic representation of the sampling operation is shown in figure 1.61.

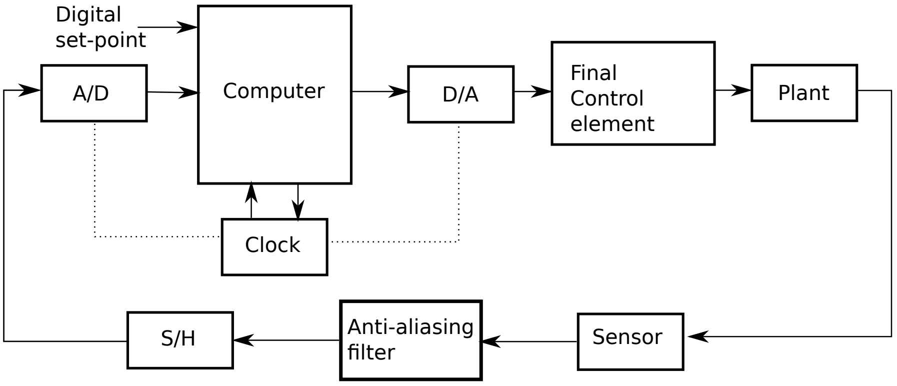

Figure 1.60: Configuration of the basic digital control scheme.

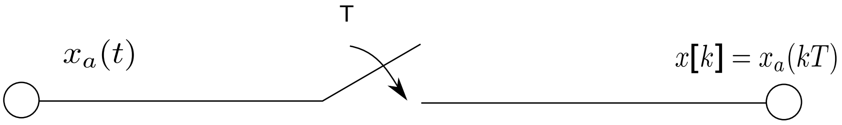

Figure 1.61: Symbolic representation of the sampling operation.

To establish a relationship of the sequence x[k] to the continuous-time function $x_a(t)$ from which this sequence is derived, we take the following approach.

We treat each sample of the sequence x[k] as an impulse function of strength equal to the value of the sample (Impulse function $A\delta(t - t_0)$ is an impulse of strength A occurring at $t = t_0$). The idea is to give a mathematical description to periodic samples of a continuous-time function in such a way that we can analyze the samples end the function simultaneously using the same tool (Laplace transform). The sequence x[k] can be viewed as a train of impulses represented by continuous-time function $x^*(t)$:

$$\begin{aligned} x^*(t) &= x(0)\delta(t) + x(1)\delta(t-T) + x(2)\delta(t-2T) + \dots \\ &= \sum_{k=0}^{\infty} x(k)\delta(t-kT) \end{aligned} \tag{1.139}$$

Typical signals $x_a(t)$, x(k) and $x^*(t)$ are shown in Fig. 1.62.

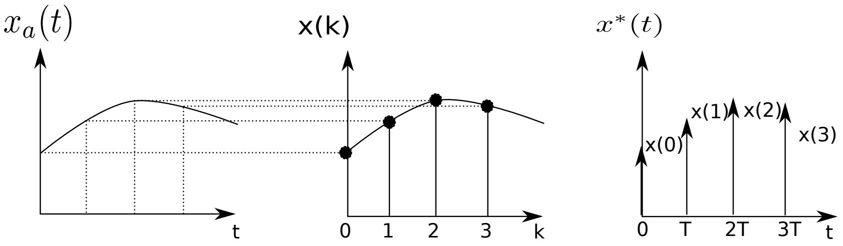

Figure 1.62: Conversion of a continuous-time signal to a sequence.

The sampler of Fig. 1.61 can thus be viewed as an impulse modulator with the carrier signal

$$\delta_T(t) = \sum_{k=0}^{\infty} \delta(t-kT) \tag{1.140}$$

and modulating signal $x_a(t)$. The Laplace transform of x(t) is

$$
\begin{aligned}
\mathcal{L}[x^*(t)] &= X^*(s) = \int_0^\infty x^*(t)e^{-st}dt \\
&= \int_0^\infty \sum_{k=0}^{\infty} x(k)\delta(t-kT)e^{-st}dt = \sum_{k=0}^{\infty} x(k)e^{-skT} \qquad (1.141)
\end{aligned}
$$

The notation $X^*(s)$ is used to symbolize the (Laplace) transform of $x^*(t)$ - the impulse modulated $x_a(t)$.

It is important to emphasize here that the impulse modulation model is a mathematical representation of sampling; not a representation of any physical system designed to implement the sampling operation. We have introduced this representation of the sampling operation because it leads to a simple derivation of a key result on sampling (given in the next section) and because this approach allows us to obtain a transfer function model of the hold operation.

The hold Operation

It is the inverse of the sampling operation-conversion of a sequence to a continuous-time function. In computer-controlled systems, it is necessary to convert the control actions calculated by the computer as a sequence of numbers, to a continuous-time signal that can be applied to the process.

The problem of hold operation may be posed as follows: Given a sequence y[0], y[1], ..., y[k], ..., we have to construct $y_a(t)$, $t \geq 0$. A commonly used solution to the problem of hold operation is polynomial extrapolation. Using a Taylor's series expansion about t= kT, we can express $y_a(t)$ as

$$
\begin{aligned}
y_a(t) &= y_a(kT) + \dot{y}_a(kT)(t-kT) + \frac{\ddot{y}_a(kT)}{2!}(t-kT)^2 + \ldots; \\
&\quad kT \leq t \leq (k+1)T \qquad (1.142)
\end{aligned}
$$

where

$$
\begin{aligned}
\dot{y}_a &= \left.\frac{dy_a(t)}{dt}\right|_{t=kT} = \frac{1}{T}\left[y_a(kT) - y_a((k-1)T)\right] \\
\ddot{y}_a &= \left.\frac{d^2y_a(t)}{dt^2}\right|_{t=kT} = \frac{1}{T}\left[\dot{y}_a(kT) - \dot{y}_a((k-1)T)\right] \\
&= \frac{1}{T^2}\left[(y_a(kT) - 2y_a((k-1)T + y_a((k-2)T)\right] \qquad (1.143)
\end{aligned}
$$

If only the first term in expansion 1.142 is used, the data hold is called a **zero-order hold** (ZOH). Here we assume that the function $y_a(t)$ is approximately constant within the sampling interval at a value equal to that of the function at the preceding sampling instant. Therefore, for a given input sequence y[k], the output of ZOH is given by

$$y_a(t) = y[k]; \quad kT \leq t < (k+1)T \tag{1.144}$$

The first two terms in Eqn. 1.142 are used to realize the first-order hold. For a given input sequence y[k], the output of the **first-order hold** is given by

$$y_a(t) = y(k) + \frac{t - KT}{T}\left[y(k) - y(k-1)\right] \tag{1.145}$$

It is obvious from Eqn. 1.142 that the higher the order of the derivative to be approximated, the larger will be the number of delay pulses required. The time-delay adversely affects the stability of feedback control systems. Furthermore, a high-order extrapolation requires complex circuitry and results in high costs of construction. The ZOH is the simplest and most commonly used data hold device. The standard D/A converters are often designed in such a way that the old value is held constant until a new conversion is ordered.

A Model of the Sample-and-Hold operation

In the digital control structure of Fig. 1.60, discrete-time processing of continuous-time signals is accomplished by the system depicted in Fig.1.63. The system is a cascade of an **A/D converter** followed by a discrete-time system (computer program) followed by a **D/A converter**. Note that the overall system is equivalent to a continuous-time system since it transforms the continuous time input signal $x_a(t)$ into the continuous-time signal $y_a(t)$. However,the properties of the system are dependent on the choice of the discrete-time system and the sampling rate.
In the special case of discrete-time signal processing with a unit-gain algorithm and negligible time delay (i.e., y[k] = x[k]), the combined action of the A/D converter, the computer, end the D/A converter can be described as a system that samples the analog signal and produces another analog signal that is constant over the sampling periods. Such a system is called a **sample-and-hold (S/H)**. Input-output behavior of a S/H system is described diagrammatically in Fig. 1.64. In the following, we develop an idealized model for S/H systems.

A S/H operations require modeling of two processes:

1. **Extracting the samples**, and

2. **holding the result fixed for one period.**

The impulse modulator effectively extracts the samples in the form of $x(k)\delta(t-kT)$. The remaining problem is to construct a linear time-invariant system which will convert this impulse into a pulse of height x[k] and width T. The S/H may therefore be modeled by Fig. 1.65, wherein the ZOH is a system whose response to a unit-impulse $\delta(t)$ is a unit-pulse $g_{h0}(t)$ of width T. The Laplace transform of the impulse response $g_{h0}(t)$ is the transfer function of the hold operation, namely

$$G_{h0}(s) = \mathcal{L}[g_{h0}(t)] = \int_0^\infty g_{h0}(t)e^{-st}dt = \int_0^T e^{-st}dt = \frac{1-e^{-sT}}{s} \tag{1.146}$$

Figure 1.65 is a block diagram representation of the transfer function model of the S/H operation.

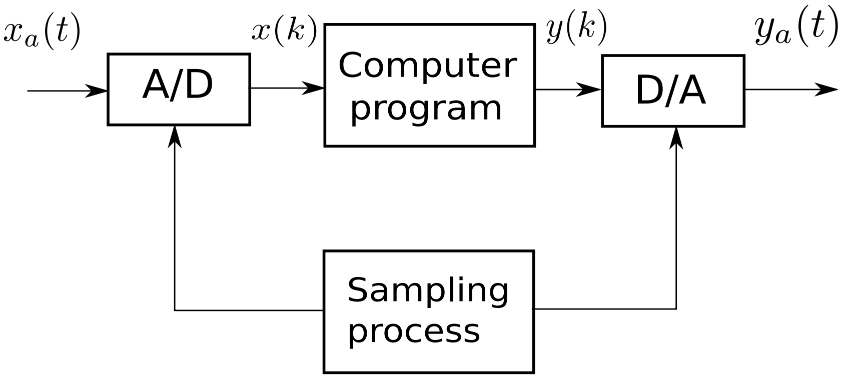

Figure 1.63: Discrete-time processing of continuous-time signals.

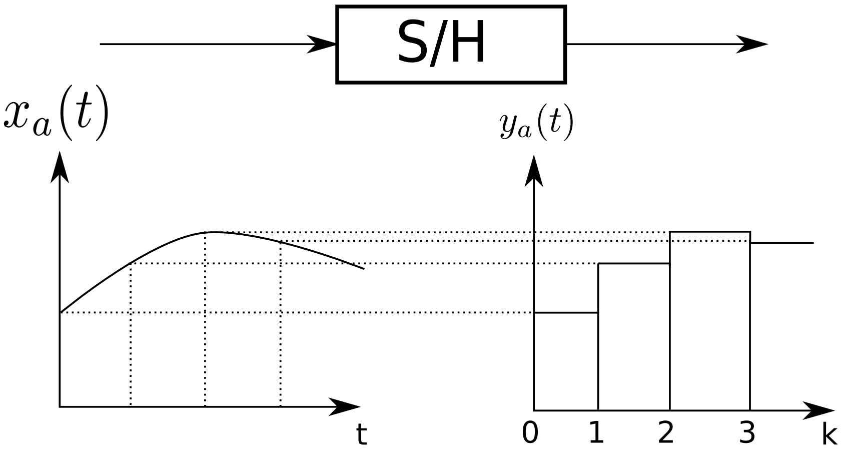

Figure 1.64: Discrete-time processing of continuous-time signals.

1.6.9 Principles of discretization

Most of the industrial processes that we are called upon to control are continuous-time processes. Mathematical models of continuous-time processes are usually based around differential equations or, equivalently, around transfer functions in the operator s. A very extensive range of well-tried methods for control system analysis and design are in the continuous-time form.

To move from the continuous-time form to the discrete-time form requires some mechanism for time discretization (we shall refer to this mechanism simply as discretization). In this section, principles and various methods of discretization will be presented. An understanding of various possible approaches helps the formation of a good theoretical foundation for the analysis and design of digital control systems.

The main point is to be aware of the significant features of discretization and to have a rough quantitative understanding of the errors that are likely to be introduced by various methods. We will shortly see that none of the discretization methods preserves the characteristics of the continuous-time system exactly.

The specific problem of this section is: given a transfer function $G(s)$, what discrete-time transfer function $G(z)$ will have **approximately the same characteristics** ?

We present four methods to solve this problem.

1. **Impulse-invariant discretization**
2. **Step-invariant discretization**
3. **Discretization based on finite-difference approximation of derivatives**
4. **Discretization based on bilinear transformation**

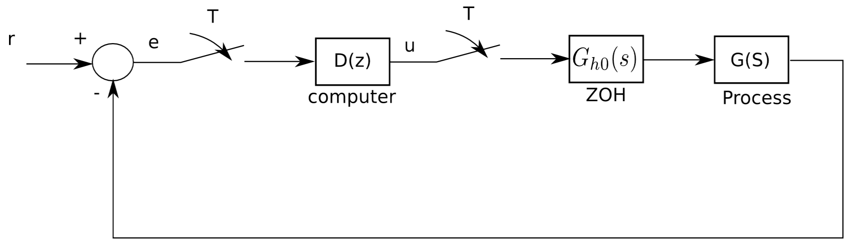

Figure 1.65: A model of a sample-and-hold operation

Impulse Invariance

The basis for impulse invariance is to choose an impulse response for the discrete-time system that is similar to the impulse response of the analog system. The use of this procedure is often motivated not so much by a desire to preserve the impulse-response shape, as by the knowledge that if the analog system is band-limited, then the discrete-time frequency response will closely approximate the continuous-time frequency response.

Step Invariance

In some design problems, a primary objective may be to control some aspect of the time response, such as the step response. In such cases, a natural approach might be to discretize the continuous-time system by waveform-invariance criteria. In this subsection, we consider the step-invariant discretization.

The step-invariant discrete-time system is obtained by placing a unit step on the input to the analog system $G_a(s)$ and a sampled unit step on the input to the discrete-time system. The transfer function G(z) of the discrete-time system is adjusted until the output of the discrete-time system represents samples of the output of the analog system. The input to the analog system $G_a(s)$ is u(t)=a unit step function. Since $\mathcal{L}\{u(t)\} = 1/s$, the output y(t) of the analog system is given by

$$y(t) = \mathcal{L}^{-1}\left\{\frac{G_a(s)}{s}\right\}$$

Output samples of the discrete-time system are defined to be

$$y(kT) = \mathcal{L}^{-1}\left\{\frac{G_a(s)}{s}\right\}\Big|_{t=kT}$$

The Z-transform of this quantity yields the Z-domain output of the discrete-time system. This gives

$$Y(z) = \mathcal{Z}\left\{\mathcal{L}^{-1}\left\{\frac{G_a(s)}{s}\right\}\Big|_{t=kT}\right\} \tag{1.147}$$

Since $\mathcal{Z}\left\{step(t)\right\} = \frac{z}{z-1}$, where $step(k)$ is the unit-step sequence, the output y(k) of the discrete-time system G(z) is given by

$$Y(z) = G(z)\left\{\frac{z}{z-1}\right\} \tag{1.148}$$

Comparing Eqn. 1.147 with Eqn. 1.148 we obtain

$$\begin{aligned} G(z) &= (1-z^{-1})\left[\mathcal{Z}\left\{\mathcal{L}^{-1}\left\{\frac{G_a(s)}{s}\right\}\Big|_{t=kT}\right\}\right] \\ &= (1-z^{-1})\left[\mathcal{Z}\left\{\frac{G_a(s)}{s}\right\}\right] \end{aligned}$$

This last equation can be written as follows:

$$G(z) = \mathcal{Z}\left\{\frac{1-e^{-st}}{s}G_a(s)\right\} \tag{1.149}$$

The right hand side of Eqn. 1.149 can be viewed as the Z-transform of the analog system $G_a(s)$ preceded by zero-order hold (ZOH). Introducing a fictitious sampler and ZOH for analytical purposes, we can use the model of Fig. 1.66 to derive a step-invariant equivalent of analog systems. For obvious reasons, step-invariant equivalence is also referred to as ZOH equivalence.

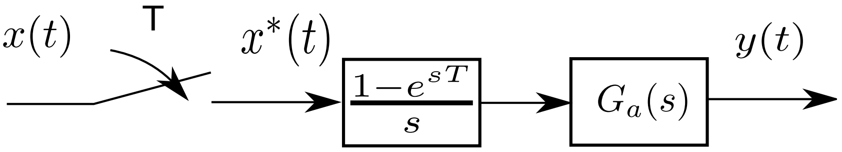

Figure 1.66: $G_a(s)$ preceded by a fictitious sample-and-hold device.

In the next chapter we will use the ZOH equivalence to obtain discrete-time equivalents of a feedback control systems.

Equivalent discrete-time systems obtained by the step-invariance method may exhibit the frequency folding phenomena and may therefore present the same kind of aliasing errors as found in impulse-invariance method. Notice, however, that the presence of 1/s term in $G_a(s)/s$ causes high-frequency attenuation. Consequently, the equivalent discrete-time system obtained by the step-invariance method will exhibit smaller aliasing errors than that obtained by the impulse-invariance method.

As for stability, the equivalent discrete-time system obtained by the step-invariance method is stable if the original continuous-time system is a stable one.

Example 1.6.8. *Figure 1.67 shows the model of a plant driven by a D/A converter. In the following, we derive the transfer function model relating y(kT) to r(kT).*
The standard D/A converters are designed in such a way that the old value of the input sample is held constant until a new sample arrives. The system of Fig. 1.67

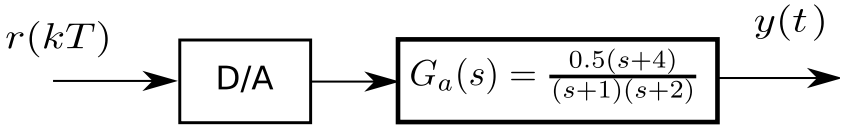

Figure 1.67: A plant driven by a D/A converter.

can therefore be viewed as an analog system $G_a(s)$ preceded by zero-order hold, and we can use ZOH equivalence to obtain the transfer function model relating y(kT) to r(kT).

ZOH equivalent (step-invariant equivalent) of $G_a(s)$ can be determined as follows. Since

$$\frac{1}{s}G_a(s) = \frac{0.5(s+4)}{s(s+1)(s+2)} = \frac{1}{s} - \frac{1.5}{s+1} + \frac{0.5}{s+2}$$

we then get

$$\mathcal{Z}\left[\frac{1}{s}G_a(s)\right] = \frac{z}{z-1} - \frac{1.5z}{z-e^{-t}} + \frac{0.5z}{z-e^{-2T}}$$

From Eqn. 1.147,

$$\begin{aligned} G(z) &= \frac{z-1}{z}\left[\frac{z}{z-1} - \frac{1.5z}{z-e^{-T}} + \frac{0.5z}{z-e^{-2T}}\right] \\ &= 1 - \frac{1.5(z-1)}{z-e^{-T}} + \frac{0.5(z-1)}{z-e^{-2T}} \end{aligned}$$

Let the sampling frequency be 20 rad/sec, so that

$$T = \frac{2\pi}{20} = 0.31416 \ sec$$
$$e^{-T} = 0.7304; \quad e^{-2T} = 0.5335$$

With these values, we get the following step-invariant equivalent of the given analog system:

$$G(z) = \frac{0.17115z - 0.04535}{z^2 - 1.2639z + 0.3897} \tag{1.150}$$

Finite-difference approximation of derivatives

Another approach to transforming a continuous-time system into a discrete-time one is to approximate derivatives in a differential equation representation of the continuous-time system by finite differences. This is a common procedure in digital simulations of analog system and is motivated by the intuitive notion that the derivative of a continuous-time function can be approximated by the difference between consecutive samples of the signal to be differentiated. To illustrate the procedure, consider the first-order differential equation

$$\frac{dy(t)}{dt} + ay(t) = r(t) \tag{1.151}$$

The backward-difference method consists of replacing r(t) by r[k], y(t) by y[k]; and the first derivative dy(t)/dt by the **first backward difference**:

$$\left.\frac{dy(t)}{dt}\right|_{t=kT} \approx \frac{y[k] - y[k-1]}{T} \tag{1.152}$$

This yields the difference equations

$$\frac{y[k] - y[k-1]}{T} + ay[k] = r[k] \tag{1.153}$$

If T is sufficiently small, we would expect the solution y[k] to yield a good approximation to the samples of y(t).

To interpret the procedure in terms of a mapping of continuous-time function $G_a(s)$ to a discrete-time function G(z), we apply the Laplace transform to Eqn.1.151 and Z-transform to Eqn. 1.153. to obtain

$$G_a(s) = \frac{Y(s)}{R(s)} = \frac{1}{s+a}$$

and

$$\left(\frac{1 - z^{-1}}{T}\right) Y(z) + aY(z) = R(z)$$

so that

$$G(z) = \frac{Y(z)}{R(z)} = \frac{1}{\left(\frac{1-z^{-1}}{T}\right) + a}$$

Computing $G_a(s)$ with G(z), we see that

$$G(z) = G_a(z)|_{s=(1-z^{-1})/T} \tag{1.154}$$

Therefore, $s = \frac{1-z^{-1}}{T}; z = \frac{1}{1-sT}$ in a mapping from the s-plane to the z-plane when the backward-difference method is used to discretize Eqn. 1.151.

The stability region in the s-plane can be mapped by Eqn. 1.154 into the Z-plane as follows. Noting that the stable region in the s-plane is given by $Re(s) < 0$, the stability region in the Z-plane under the mapping is shown in figure 1.68.

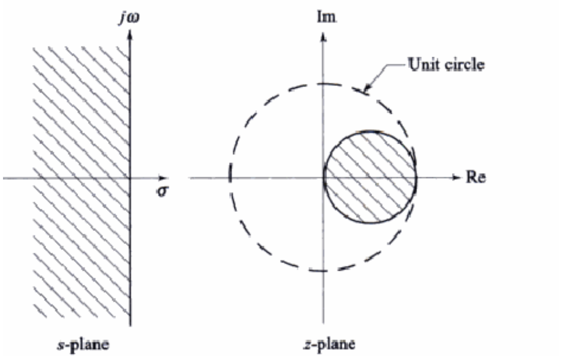

Figure 1.68: Mapping of the s-plane to the s-plane: using backward difference approximation of the derivative .

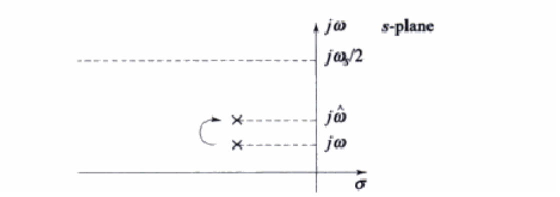

Figure 1.69: The warping effect.

The backward-difference method is simple and will produce a stable discrete-time system for a stable continuous-time system. Also, the entire s-plane imaginary axis is mapped only once onto the small Z-plane circle; the folding or aliasing problems do not occur. The penalty is a warping of the equivalent s-plane poles as shown in Fig. 1.69. This situation is reflected in the relationship between the exact z-transformation and the backward-difference approximation.

Let us now investigate the behavior of the equivalent discrete-time system when the derivative dy(t)/dt in Eqn. 1.151 is replaced by **forward difference**:

$$\left.\frac{dy(t)}{dt}\right|_{t=kT} \approx \frac{y[k+1]-y[k]}{T} \tag{1.155}$$

This yields the following difference equation approximation for Eqn. 1.151:

$$\frac{y[k+1]-y[k]}{T} + ay[k] = r[k] \tag{1.156}$$

Applying Laplace transform to Eqn. 1.151 and Z-transform to Eqn. 1.156, we obtain

$$G_a(s) = \frac{Y(s)}{R(s)} = \frac{1}{s+a} \tag{1.157}$$

and

$$G(z) = \frac{Y(z)}{R(z)} = \frac{1}{\left(\frac{z-1}{T}\right) + a} \tag{1.158}$$

The right-hand sides of Eqns. 1.157 and 1.158 become identical if we let

$$s = \frac{z-1}{T} \tag{1.159}$$

We may consider Eqn. 1.159 for the mapping from the s-plane to the Z-plane when the forward-difference method is used to discretize Eqn. 1.151.

One serious problem with the forward-difference approximation method is regarding stability. The left hand side of the s-plane is mapped into the region $Re\left(\frac{z-1}{T}\right) < 0$ or $Re(z) < 1$. This means that the poles of the left half of the s-plane may be mapped outside the unit circle in Z-plane. Hence the discrete-time system obtained by this method my become unstable.

Rectangular rules for integration

Consider the continuous-time system of Eqn. 1.151

$$\frac{dy(t)}{dt} = -ay(t) + r(t) \tag{1.160}$$

or

$$y(t) = y(0) - a\int_0^t y(\tau)d\tau + \int_0^t r(\tau)d\tau \tag{1.161}$$

In numerical analysis, the procedure known as the rectangular rule for integration proceeds by approximating the continuous-time function by continuous rectangles, as illustrated in Fig. 1.70, and then adding their areas to compute the total integral. We thus approximate the area as given below:
(i) *Forward rectangular rule for integration*

$$\int_{(k-1)T}^{kT} y(t)dt \approx [y(k-1)]T \tag{1.162}$$

(ii) *Backward rectangular rule for integration*

$$\int_{(k-1)T}^{kT} y(t)dt \approx [y(k)]T \tag{1.163}$$

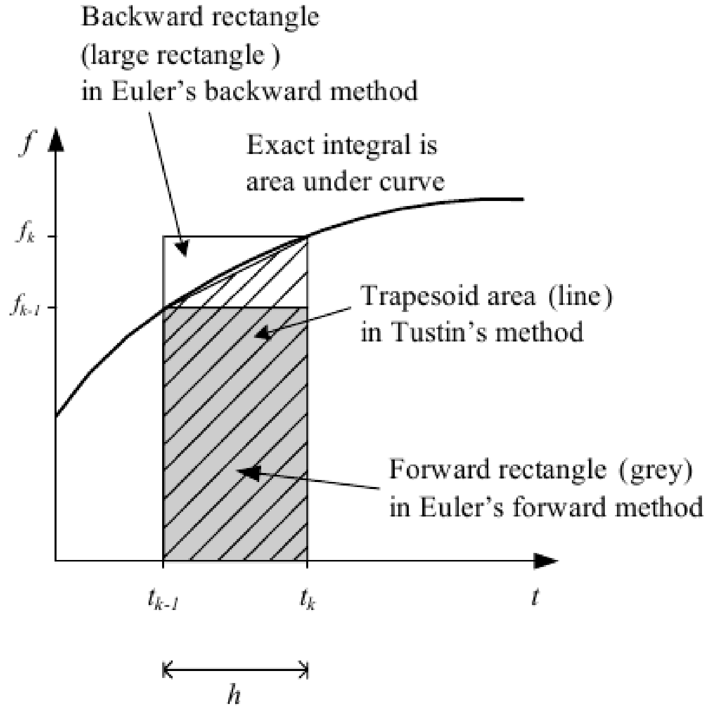

Figure 1.70: Numerical approximations to the integral.

With the forward rule for integration, the continuous-time system 1.160 is converted to the following recursive algorithm:

$$y(k) = y(k-1) - aTy(k-1) + Tr(k-1)$$

The z-transformation of this equation gives

$$\frac{Y(z)}{R(z)} = \frac{1}{\frac{z-1}{T} + a}$$

The forward rectangular rule for integration thus results in the s-plane to Z-plane mapping:

$$s = \frac{z-1}{T}$$

which is same as the one obtained by forward-difference approximation of derivatives (Eqn. 1.159). Similarly, it can easily be established that the backward rectangular rule for integration results in s-plane to Z-plane mapping which is same as the one obtained by backward-difference approximation of derivatives (Eqn. (1.154)).

Bilinear Transformation

The technique based on finite-difference approximation to differential equations for deriving a discrete-time system from an analog system, has the advantage that Z-transform of the discrete-time system is trivially derived from the Laplace transform of the analog system by an algebraic substitution. The disadvantages of these mappings are that $j\omega$-axis in the s-plane generally does not map into the unit circle in the Z-plane and (for the case of forward-difference method) stable analog systems may not always map into stable discrete-time systems.
A nonlinear one-to-one mapping from the s-plane to the Z-plane which eliminates the disadvantages mentioned above and which preserves the desired algebraic form is the **bilinear transformation** defined by

$$s = \frac{2}{T}\frac{z-1}{z+1} \tag{1.164}$$

This transformation is invertible with the inverse mapping given by

$$Z- = \frac{1+(T/2)s}{1-(T/2)s} \tag{1.165}$$

The bilinear transformation also arise from a particular approximation methodthe **trapezoidal rule** for numerically integrating differential equations. Let us consider a continuous-time system for which the describing equation is (Eqn. 1.151)

$$\dot{y}(t) = -ay(t) + r(t) \tag{1.166}$$

or

$$y(t) = y(0) - a\int_0^T y(\tau)d\tau + \int_0^T r(\tau)d\tau \tag{1.167}$$

Laplace transformation of Eqn. (1.166) gives the transfer function of the continuous-time system.

$$\frac{Y(s)}{R(s)} = G_a(s) = \frac{1}{s+a}$$

Applying bilinear transformation (Eqn. 1.164) to this transfer function, we obtain

$$G(z) = \frac{1}{\frac{2}{T}\left(\frac{z-1}{z+1}\right)+a}$$

In numerical analysis, the procedure known as the trapezoidal rule for integration proceeds by approximating the continuous-time function by continuous trapezoids, as illustrated in Fig. 1.71, and then adding their areas to compute the total integral. We thus approximate the are:

$$\int_{(k-1)T}^{kT} y(t)dt \approx \frac{1}{2}[y(k)+y(k-1)]T$$

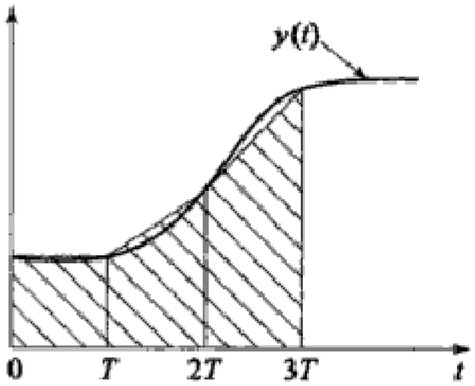

Figure 1.71: Trapezoidal rule for integral approximation.

With this approximation. Eqn. 1.167 can be converted to the following recursive algorithm:

$$y(k) = y(k-1) - \frac{aT}{2}[y(k)+y(k-1)] + \frac{T}{2}[r(k)+r(k-1)]$$

The Z-transformation of this equation gives

$$Y(z) = z^{-1}Y(z) - \frac{aT}{2}[Y(z)+z^{-1}Y(z)] + \frac{T}{2}[R(z)+z^{-1}R(z)]$$

or

$$\frac{Y(z)}{R(z)} = \frac{1}{\frac{2}{T}\left(\frac{z-1}{z+1}\right)+a}$$

This result is identical to the one obtained from the transfer function of the continuous-time system by bilinear transformation.

The nature of bilinear transformation is best understood from Fig. 1.72, which shows how the s-plane is mapped onto the Z-plane. As seen in the figure, the entire $j\omega$-axis is in the s-plane is mapped onto the unit circle in the z-plane. The left half of s plane is mapped inside the unit circle in the z-plane, and the right half of s-plane is mapped outside the z-plane unit circle. These properties can easily be established.

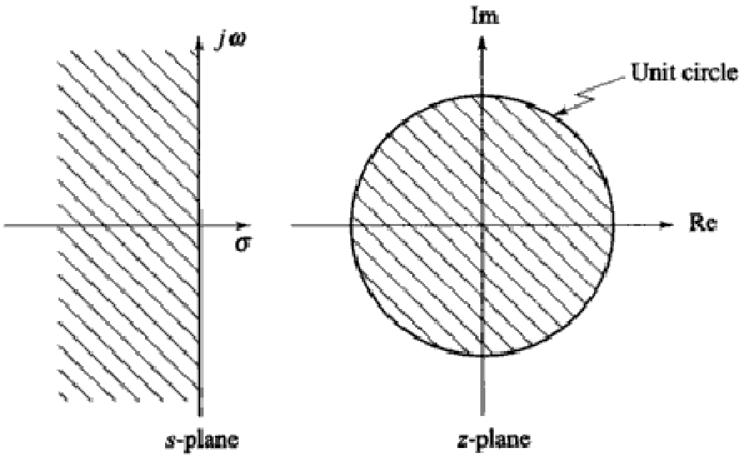

Figure 1.72: Mapping of the s-plane to the Z-plane using bilinear transformation.

1.6.10 Discretization with software

In software, the command 'c2d' computes discrete equivalent of the continuous-time system. Two parameters need to be defined: the sampling time *dt* and the discretization method (ZOH, bilinear, forward or backward Euler)

```
c2d(TF,dt,'method')
```

Depending on the software program this command generates a transfer function (Matlab), state space model (Octave) or both depending on the input (Python). Programs 1.73, 1.74 show an example for the three software platforms executing a ZOH, bilinear discretization.

Matlab (top)/Octave (bottom) code

```
>> num = [3, 2, 1];
>> den=[-1, 4, 1];
>> dt= 0.5;
>> TF = tf(num,den);
>> DTF = c2d(TF,dt,'zoh')
 -3 z^2 - 3.24 Z-- 1.055
  ------------------
   z^2-9.203 z+7.389
------------------------
>> num = [3, 2, 1];
>> den=[-1, 4, 1];
>> dt= 0.5;
>> TF = tf(num,den);
>> dss = c2d(TF,dt);
>> [n,d]=ss2tf(dss.a,dss.b
,dss.c,dss.d)
n =
 -3.0000  3.2403 -1.0547
d =
  1.0000 -9.2034  7.3891
```

Python

```
import numpy as np
from scipy.signal import cont2discrete as c2d

numc = np.array([3, 2, 1])
denc = np.array([-1, 4, 1])
dt = 0.5

num, den, dt=c2d((numc, denc),dt,method='zoh')
print(num)
print(den)
[[-3.          3.24027011 -1.05465432]]
[ 1.         -9.20344032  7.3890561 ]
```

Figure 1.73: Matlab/Octave/Python program 1.73.

<table>
<tr><th>Matlab (top)/Octave (bottom) code</th><th>Python code</th></tr>
<tr><td>

```
>> num = [3, 2, 1];
>> den=[-1, 4, 1];
>> dt= 0.5;
>> TF = tf(num,den);
>> DTF = c2d(TF,dt,’tustin’)
   57 z^2 - 94 Z-+ 41
  -----------------
     z^2+34 z-31

------------------------

>> num = [3, 2, 1];
>> den=[-1, 4, 1];
>> dt= 0.5;
>> TF = tf(num,den);
>> dss = c2d(TF,’bil’,dt);
>> [n,d]=ss2tf(dss.a,dss.b
,dss.c,dss.d)
n =
 57.000 -94.000 41.000
d =
 1.0000 34.0000 -31.0000
```

</td><td>

```
import numpy as np
from scipy.signal import cont2discrete as c2d

n = np.array([3, 2, 1])
d = np.array([-1, 4, 1])
dt = 0.5

num, den, dt=c2d((n, d),dt,method=’bilinear’)
print(num)
print(den)
[[ 57. -94.  41.]]
[  1.  34. -31.]
```

</td></tr>
</table>

Figure 1.74: Matlab/Octave/Python program 1.74.

Chapter 2

Dynamic systems: Control

2.1 Analysis of continuous control systems

2.1.1 Introduction

This chapter presents basic information on control systems. Our discussions are limited to **time-domain analysis** and design based on the **transient-response analysis** and **root-locus analysis**. We begin the chapter by defining some terms that are essential in describing control systems; we then follow with a description of closed-loop and open-loop control systems. Finally, the advantages and disadvantages of closed-loop and open-loop control systems are compared.

Plants

A plant is a piece of equipment-perhaps a set of machine parts functioning together-the purpose of which is to perform a particular operation. In this book, we shall call any physical object that is to be controlled a plant.

Disturbances.

A disturbance is a signal that tends to affect the value of the output of a system adversely. If the disturbance is generated within the system, it is called internal; an external disturbance is generated outside the system and is an input.

Feedback control.

Feedback control refers to an operation that, in the presence of disturbances, tends to reduce the difference between the output of a system and the reference input and that does so on the basis of this difference. Here, only unpredictable disturbances are so specified, since predictable or known disturbances can always be compensated for within the system.

Feedback control systems.

A system that maintains a prescribed relationship between the output and the reference input by comparing them and using the difference as a means of control is called a feedback control system or simply a control system. An example is a room temperature control system. By measuring the actual room temperature and comparing it with the reference temperature (the desired temperature), the thermostat turns the heating or cooling equipment on or off in such a way as to ensure that the temperature of the room remains at a comfortable level, regardless of outside conditions. Feedback control systems, of course, are not limited to engineering, but can be found in various nonengineering fields as well. An example, consider the control of automobile speed by a human operator. For a given situation, the driver decides on an appropriate speed, which may be the posted speed limit on the road or highway involved. This speed acts as the reference speed. The driver observes the actual speed by looking at the speedometer. If he or she is traveling too slowly, the driver depresses the accelerator and the car speeds up. If the actual speed is too high, the driver releases the pressure on the accelerator and the car slows down. This is is a feedback control system with a human operator. The human operator here can easily be replaced by a mechanical, an electrical, or some similar device. Instead of the driver observing the speedometer, an electric generator can be used to produce a voltage that is proportional to the speed. This voltage can be compared with a reference voltage that corresponds to the desired speed. The difference in the voltages can then be used as the error signal to position the throttle to increase or decrease the speed as needed.

Closed-loop control systems.

Feedback control systems are often referred to as closed-loop control systems. In practice, the terms feedback control and closed-loop control are used interchangeably. In a closed-loop control system, the actuating error signal, which is the difference between the input signal and the feedback signal (which may be the output signal itself or a function of the output signal and its derivatives), is fed to the controller so as to reduce the error and bring the output of the system to a desired value. The term closed-loop control always implies the use of feedback control action in order to reduce system error.

Open-loop control systems.

Those control systems in which the output has no effect on the control action are called open-loop control systems. In other words, in an open-loop control system, the output is neither measured nor fed back for comparison with the input. One practical example is a washing machine. Soaking, washing, and rinsing in the washer operate on a time basis. The machine does not measure the output signal, that is, the cleanliness of the clothes. In an open-loop control system, the output is not compared with the reference input. Thus, to each reference input, there corresponds a fixed operating condition, and as a result, the accuracy of the system depends on calibration. In the presence of disturbances, an open-loop control system will not perform the desired task. Open-loop control can be used, in practice, only if the relationship between the input and output is known and if there are neither internal nor external disturbances. Clearly, such systems are not feedback control systems.

Closed-loop versus open-loop control systems.

An advantage of the closed-loop control system is the fact that the use of feedback makes the system response relatively insensitive to external disturbances and internal variations in system parameters. It is thus possible to use relatively inaccurate and inexpensive components to obtain accurate control of a given plant, whereas doing so is impossible in the open-loop case. From the point of view of stability, the open-loop control system is easier to build, because system stability is not a major problem. By contrast, stability is a major problem in the closed-loop control system, which may tend to overcorrect errors that can cause oscillations of constant or changing amplitude.

General requirements of control systems.

A primary requirement of any control system is that it must' be **stable**. In addition to absolute stability, a control system must have a reasonable **relative stability**; that is, the response must show reasonable damping. Moreover, the speed of response must be reasonably fast. A control system must also be capable of reducing errors to zero or to some small tolerable value. Because the needs for reasonable relative stability and for steady-state accuracy tend to be incompatible, in designing control systems it is necessary to make the most effective compromise between the two.

2.1.2 Block diagrams

A system may consist of a number of components. To show the functions performed by each component, block diagrams are frequently used in the analysis and design of control systems. This section first defines the open-loop transfer function, feedforward transfer function, and closed-loop transfer function. Finally, a progamming approach to obtaining transfer functions or state-space representations of series-connected systems, parallel-connected systems, and feedback-connected systems is presented.

Open-loop transfer function and feedforward transfer function.

Figure 2.1 shows the block diagram of a closed-loop system with a feedback element. The ratio of the feedback signal B(s) to the actuating error signal E(s) is called the **open-loop transfer function**. That is,

$$\mathrm{open-loop\ transfer\ function} = \frac{\mathrm{B(s)}}{\mathrm{E(s)}} = \mathrm{G(s)H(s)}$$

The ratio of the output C(s) to the actuating error signal E(s) is called the **feedforward transfer function**

$$\text{feedforward transfer function} = \frac{\mathrm{C(s)}}{\mathrm{E(s)}} = \mathrm{G(s)}$$

If the feedback transfer function is unity, then the open-loop transfer function and the feedforward transfer function are the same.

Closed-loop transfer function

For the system shown in Figure 2.1 , the output C(s) and input R(s) are related as follows:

$$\begin{aligned} C(s) &= G(s)E(s) \\ E(s) &= R(s) - B(s) = R(s) - H(s)C(s) \end{aligned}$$

Eliminating E(s) from these equations gives

$$C(s) = G(s)[R(s) - H(s)C(s)]$$

or

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s)H(s)}$$

The transfer function relating C(s) to R(s) is called the **closed-loop transfer function**. This transfer function relates the closed-loop system dynamics to the dynamics of the feed forward elements and feedback elements.

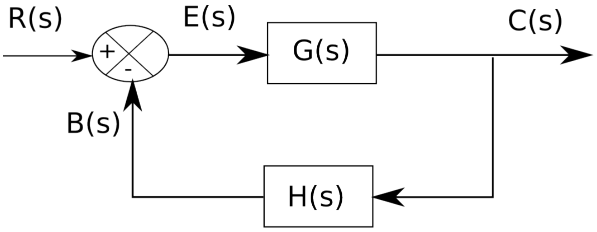

Figure 2.1: Block diagram of a closed-loop system with a feedback element.

Using Matlab/Octave/Python to obtain transfer functions of series-connected blocks, parallel-connected blocks, and feedback-connected blocks

A physical system may involve many interconnected blocks. In what follows, we shall consider series-connecled blocks, parallel-connected blocks, and feedback-connected blocks. Any linear, time-invariant system may be represented by combinations of series-connected blocks, parallel-connected blocks, and feedback-connected blocks.

Series-connected blocks.

In the system shown in Figure 2.2, G_1, and G_2, are series connected. System G_1, and system G_2, are respectively defined by

```
sys1=tf(num1,den1)
sys2=tf(num2,den2)
```

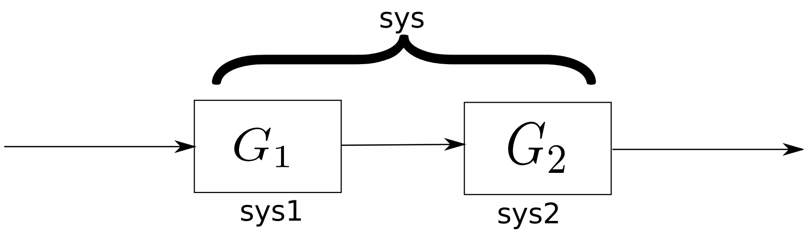

Figure 2.2: Series-connected blocks.

provided that these two systems are themselves defined in terms o f transfer func tions. Then the series-connected system G_1,G_2, can be given by

```
sys = series(sys1 ,sys2)
```

The system's numerator and denominator can be given by

```
[num,den] = series(num1,den1,num2,den2)
```

Example 2.1.1. *Consider the case where*

$$G_1 = \frac{10}{s^2+2s+10}, \qquad G_2 = \frac{5}{s+5}$$

Programs 2.3 produces the transfer function of the series-connected system.

Parallel-connected blocks. Figures 2.4(a) and (b) show parallel-connected systems. In Figure 2.4(a) two systems G_1, and G_2, are added, while in Figure 2.4(b) system G_1, is subtracted from system G_2.The parallel-connected system $G_1 + G_2$ is given by

```
sys = parallel(sys1,sys2)
```

or

```
[num,den] = parallel(num1,den1,num2,den2)
```

Matlab/Octave code

```
>>num1 = [10];
>>den1=[1 2 10];
>>sys1 = tf(num1, den1);
>>num2 = [5];
>>den2=[1 5];
>>sys2 = tf(num2, den2);
>>sys = series(sys1,sys2)
Transfer function:
           50
-------------------------
s^3 + 7 s^2 + 20 s + 50
```

Python code

```
from matplotlib.pyplot import *
from control.matlab import *

num1 = [10]
den1=[1,2,10]
sys1 = tf(num1, den1)
num2 = [5]
den2=[1,5]
sys2 = tf(num2, den2)
sys = series(sys1,sys2)
print sys
          50
-----------------------
s^3 + 7 s^2 + 20 s + 50
```

Figure 2.3: Matlab/Octave/Python program 2.3.

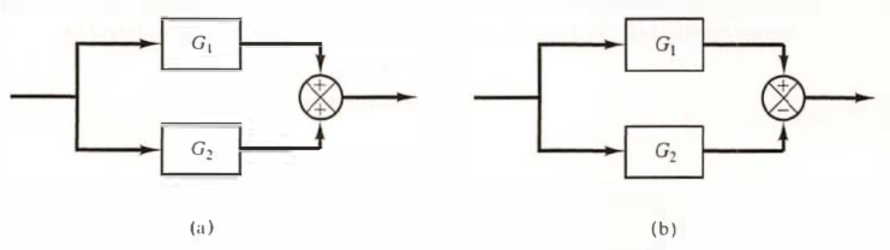

Figure 2.4: Parallel-connected blocks.

Example 2.1.2. *Consider the case where*

$$G_1 = \frac{10}{s^2 + 2s + 10}, \qquad G_2 = \frac{5}{s+5}$$

Programs 2.5 produces the transfer function of the series-connected system.

Feedback-connected blocks: Figure 2.6(a) shows a negative feedback system, and Figure 2.6(b) shows a positive feedback system.
If G and H are defined in terms of transfer functions, then

```
sysg = tf[numg,deng]
sysh = tf[numh,denh]
```

Matlab/Octave code

```
>>num1 = [10];
>>den1=[1 2 10];
>>sys1 = tf(num1, den1);
>>num2 = [5];
>>den2=[1 5];
>>sys2 = tf(num2, den2);
>>sys = parallel(sys1,sys2)
Transfer function:
  5 s^2 + 20 s + 100
-----------------------
s^3 + 7 s^2 + 20 s + 50
```

Python code

```
from matplotlib.pyplot import *
from control.matlab import *

num1 = [10]
den1=[1,2,10]
sys1 = tf(num1, den1)
num2 = [5]
den2=[1,5]
sys2 = tf(num2, den2)
sys = parallel(sys1,sys2)
print sys
   5 s^2 + 20 s + 100
-----------------------
s^3 + 7 s^2 + 20 s + 50
```

Figure 2.5: Matlab/Octave/Python program 2.5.

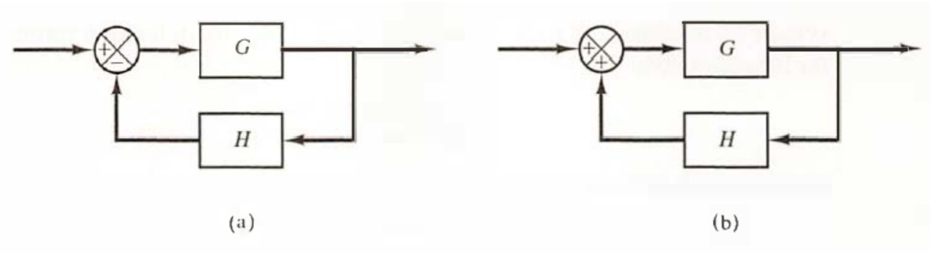

Figure 2.6: Parallel-connected blocks.

and the entire feedback system is given by

```
sys = feedback(sysg,sysh)
```

or

```
[num,den] = feedback(sysg,sysh)
```

If the system has a unity feedback function, then H=[1] and sys can be given by

```
sys = feedback(sysg, 1)
```

Note that, in treating the feedback system, the program assumes that the feedback is negative. If the system involves a positive feedback, we need to add " + 1 " in the argument of feedback as follows:

```
sys = feedback(sysg,sysh,+1)
```

Example 2.1.3. *Consider the case where*

$$G_1 = \frac{5}{s^2+2s}, \qquad G_2 = 0.1s+1$$

Programs 2.7 produces the transfer function of the series-connected system.

Matlab/Octave code

```
>>numg = [5];
>>deng=[1 2 0];
>>sysg = tf(numg, deng);
>>numh = [0.1 1];
>>denh=[1];
>>sysh = tf(numh, denh);
>>sys = feedback(sysg,sysh)
Transfer function:
    5
---------------
s^2 + 2.5 s + 5
```

Python code

```
from matplotlib.pyplot import *
from control.matlab import *

numg = [5]
deng=[1,2,0]
sysg = tf(numg, deng)
numh = [0.1 1]
denh=[1]
sysh = tf(numh, denh)
sys = feedback(sysg,sysh)
print sys
       5
---------------
s^2 + 2.5 s + 5
```

Figure 2.7: Matlab/Octave/Python program 2.7.

2.1.3 Automatic controllers

An automatic controller compares the actual value of the plant output with the desired value, determines the deviation. and produces a control signal that will reduce the deviation to zero or a small value. The way in which the automatic controller produces the control signal is called **control action**. Here, we describe the fundamental control actions commonly used in industrial automatic controllers. We then briefly discuss an electronic controller.

Control actions.

The control actions normally found in industrial automat ic controllers consist of the following: two-position, or on-off; proportional; integral; derivative; and combinations of proportional, integral, and derivative. A good understanding of the basic properties of various control actions is necessary for the engineer to select the one best suited to his or her particular application.

Classifications of industrial automatic controllers.

Industrial automatic controllers can be classified according to their control action as follows:

1. Two-position, or on-off, controllers
2. Proportional controllers
3. Integral controllers
4. Proportional-plus-integral controllers
5. Proportional-plus-derivative controllers
6. Proportional-plus-integral-plus-derivative controllers

Automatic controller, actuator, and sensor (measuring element).

Figure 2.8 is a block diagram of an industrial control system consisting of an **automatic controller**, an **actuator**, a **plant** and a **sensor** or measuring element. The controller detects the actuating error signal, which is usually at a very low power level, and amplifies it to a sufficiently high level. (Thus the automatic controller comprises an error detector and an amplifier.) Quite often, a suitable feedback circuit, to gether with an amplifier, is used to alter the actuating error signal to produce a better control signal. The **actuator** is an element that produces the input to the plant according to the control signal, so that the feedback signal will correspond to the reference input signal.

The **sensor** or measuring element is a device that converts the output variable into another suitable variable, such as a displacement. pressure, or voltage, which can be used to compare the output with the reference input signal. This element is in the feedback path of the closed-loop system. The set point of the controller must be converted to a reference input of the same units as the feedback signal from the sensor or measuring element.

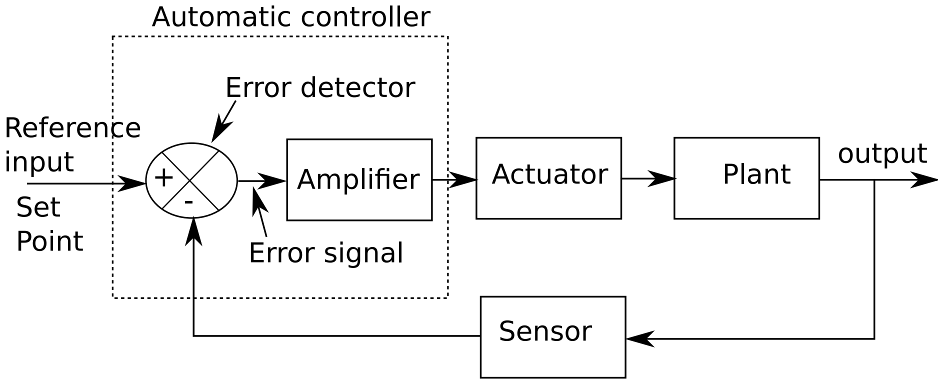

Figure 2.8: Block diagram of an industrial control system consisting of an automatic controller, an actuator, the plant and a sensor (measuring element).

Proportional, integral, and derivative control actions.

Proportional, integral, and derivative control actions are basic control actions found in industrial automatic controllers. For each control action, the output of the controller, M(s), and the actuating error signal E(s) are related by a transfer function of a specific form. In what follows, we illustrate transfer functions M(s)/E(s) for proportional control action, proportional-plus-integral control action, proportional-plus-derivative control action, and proportional-plus-integral-plus-derivative control action.
For **proportional control** action, M(s) and E(s) are related by

$$\frac{M(s)}{E(s)} = G_c(s) = K_p$$

where K_p is termed the **proportional gain**.
For **integral control** action, the relationship betwcen M(s) and E(s) is

$$\frac{M(s)}{E(s)} = G_c(s) = \frac{K_i}{s}$$

where K_i is called the **integral gain**.
For **proportional-plus-integral control** action, M(s) and E(s) are related by

$$\frac{M(s)}{E(s)} = G_c(s) = K_p(1 + \frac{1}{T_i s})$$

where K_p is the proportional gain and T_i is a constant called the integral time constant.

For **proportional-plus-derivative control** action, M(s) and E(s) are related by

$$\frac{M(s)}{E(s)} = G_c(s) = K_p(1 + T_d s)$$

where K_p is the proportional gain and T_d is a constant called the derivative time constant.

Finally, for **proportional-plus-integral-plus-derivative control** action, M(s) and E(s) are related by

$$\frac{M(s)}{E(s)} = G_c(s) = K_p(1 + \frac{1}{T_i s} + T_d s)$$

where K_p is the proportional gain, T_i is the integral time constant, and T_d is the derivative time constant.

2.1.4 Transient response specifications

Because systems that store energy cannot respond instantaneously, they exhibit a transient response when they are subjected to inputs or disturbances. Consequently, the transient-response characteristics constitute one of the most important factors in system designed.

In many practical cases, the desired performance characteristics of control systems can be given in terms of transient-response specifications. Frequently, such performance characteristics are specified in terms of the transient response to a unit-step input, since such an input is easy to generate and is sufficiently drastic. (If the response of a linear system to a step input is known, it is mathematically possible to compute the system's response to any input.)

The transient response of a system to a unit-step input depends on initial conditions. For convenience in comparing the transient responses of various systems, it is common practice to use a standard initial condition: The system is at rest initially, with its output and all time derivatives set to zero. Then the response characteristics can be easily compared.

Transient-response specifications.

The transient response of a practical control system often exhibits damped oscillations before reaching a steady state. In specifying the transient-response characteristics of a control system to a unit-step input, it is common to name the following:

1. Delay time, t_d

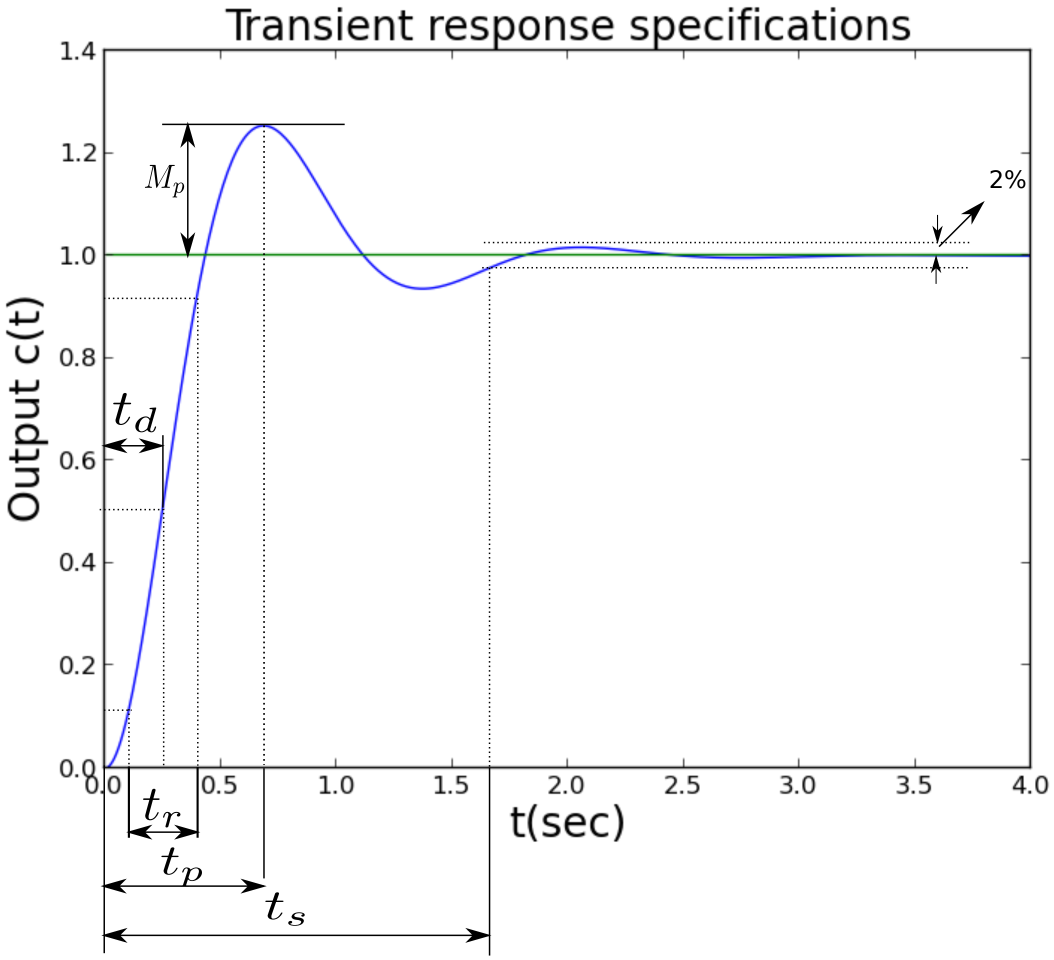

Figure 2.9: Transient-response specifications.

2. Rise time, t_r

3. Peak time, t_p

4. Maximum overshoot, M_p

5. Settling time, t_s

These specifications are defined next and are shown graphically in Figure 2.9

Delay time.

The delay time t_d is the time needed for the response to reach half the final value the very first time.

Rise time.

The rise time t_r, is the time required for the response to rise from 10% to 90%, 5% to 95%, or 0% to 100% of its final value. For underdamped second-order systems, the 0% to 100% rise time is normally used. For overdamped systems, the 10% to 90% rise time is common.

Peak time.

The peak time T_p is the time required for the response to reach the first peak of the overshoot.

Maximum (percent) overshoot.

The maximum overshoot M_p is the maximum peak value of the response curve [the curve of c(t) versus t], measured from c(∞). If c(∞) = 1 , the maximum percent overshoot is M_p X 100%. If the final steady-state value c (∞) of the response differs from unity, then it is common practice to use the following definition of the maximum percent overshoot:

$$\text{maximum percent overshoot} = \frac{\mathrm{c(T_p) - c(\infty)}}{\mathrm{c(\infty)}} \times 100\% \tag{2.1}$$

The amount of the maximum (percent) overshoot directly indicates the relative stability of the system.

Settling time.

The settling time t_s, is the time required for the response curve to reach and stay within 2% of the final value. In some cases, 5%, instead of 2%, is used as the percentage of the final value. Throughout this book, however, we use the 2% criterion. The settling time is related to the largest time constant of the system.

Comments.

If we specify the values of t_d, t_r t_p, t_s and M_p the shape of the response curve is virtually fixed.

Note that not all these specifications necessarily apply to any given case. For instance, for an overdamped system, the terms peak t_p and maximum overshoot M_p do not apply.

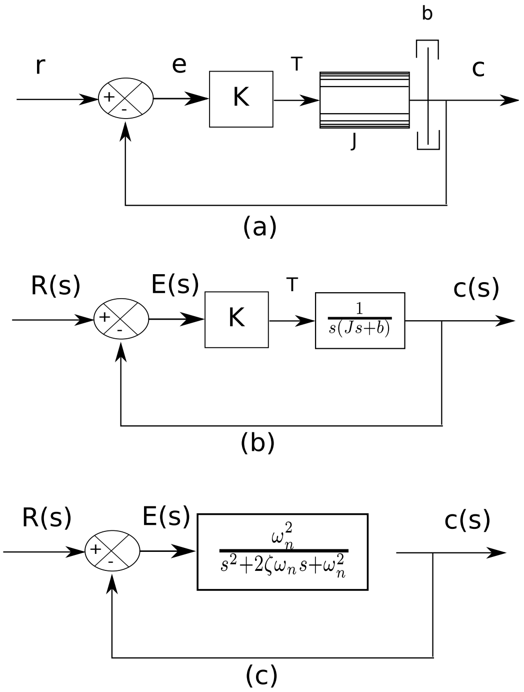

Figure 2.10: (a) Position control system;(b) block diagram: (c) block diagram of a second-order system in standard form.

Position control system.

The position control system (servo system) shown in Figure 2.10 consists of a proportional controller and load elements (inertia and viscous-friction elements). Suppose that we wish to control the output position C in accordance with the input position r.
The equation for the load elements is

$$J\ddot{c} + b\dot{c} = T$$

where T is the torque produced by the proportional controller, whose gain constant is K. Taking Laplace transforms of both sides of this last equation, assuming zero initial conditions, we find that

$$Js^2C(s) + bsC(s) = T(s)$$

So the transfer function between C(s) and T(s) is

$$\frac{C(s)}{T(s)} = \frac{1}{s(Js+b)}$$

With the use of this transfer function, Figure 2.10(a) can be redrawn as shown in 2.10(b). The closed-loop transfer function is then

$$\frac{C(s)}{R(s)} = \frac{K}{Js^s + bs + K} = \frac{K/J}{s^2 + (b/J)s + K/J}$$

or

$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2.2}$$

where

$$\begin{aligned} \omega_n &= \sqrt{K/J} = \text{undamped natural frequency} \\ \zeta &= \frac{b}{2\sqrt{KJ}} \end{aligned}$$

In terms of ζ and ω_n the block diagram of 2.10(b) can be redrawn as shown in 2.10(c).

Next, let us consider the unit-step response of this system when $0 < \zeta < 1$. For a unit-step input, we have R(s) = 1/s. Then

$$\begin{aligned} C(s) &= \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}\frac{1}{s} \\ &= \frac{1}{s} - \frac{s + 2\zeta\omega_n}{s^2 + 2\zeta\omega_n s + \omega_n^2} \\ &= \frac{1}{s} - \frac{\zeta\omega_n}{(s+\zeta\omega_n)^2 + \omega_d^2} - \frac{s + \zeta\omega_n}{(s+\zeta\omega_n)^2 + \omega_d^2} \end{aligned} \tag{2.3}$$

where $\omega_d = \omega_n\sqrt{1-\zeta^2}$. The inverse Laplace transform of Equation 2.3 gives

$$\begin{aligned} c(t) &= 1 - \frac{\zeta}{\sqrt{1-\zeta^2}} e^{-\zeta\omega_n t}\sin(\omega_d t) - e^{-\zeta\omega_d t}\cos(\omega_d t) \\ &= 1 - e^{-\zeta\omega_n t}\left(\frac{\zeta}{\sqrt{1-\zeta^2}}\sin(\omega_d t) + cos(\omega_d t)\right) \end{aligned} \tag{2.4}$$

or

$$c(t) = 1 - \frac{e^{-\zeta\omega_n t}}{\sqrt{1-\zeta^2}}\sin\left(\omega_d t + \tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}\right) \tag{2.5}$$

A family of curves C(t) plotted against t with various values of ζ is shown in Figure 2.11, where functions only of ζ.

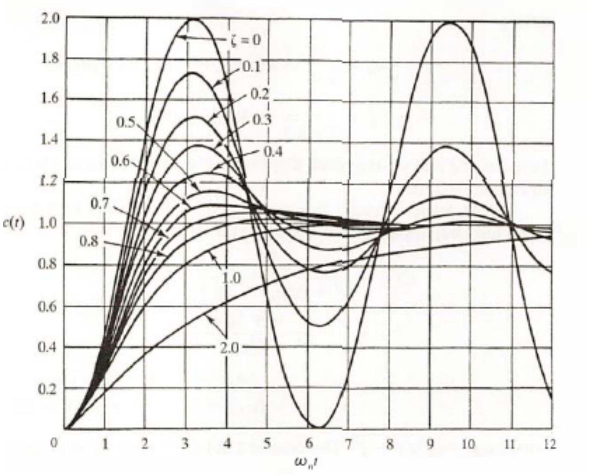

Figure 2.11: Unit-step response curves for the system shown in Figure 2.10(c).

Second-order systems and transient-response specifications.

Let us now obtain the rise time, peak time, maximum overshoot, and settling time of the second-order system given by Equation 2.2. These values will be derived in terms of ζ and ω_n. The system is assumed to be underdamped !

Rise time t_e. We find the rise time t_r, by setting $c(t) = 1$ in Equation 2.4. Since $e^{-\zeta\omega_n t_r} \neq 0$, Equation c(t,) = 1 yields

$$\frac{\zeta}{\sqrt{1-\zeta^2}} \sin(\omega_d t_r) + cos(\omega_d t_r) = 0 \tag{2.6}$$

Thus the rise time t_r is

$$t_r = \frac{1}{\omega_d} \tan^{-1}\left(-\frac{\sqrt{1-\zeta^2}}{\zeta}\right) \tag{2.7}$$

Peak time We obtain the peak time by differentiating c(t) in Equation 2.4 with respect to time t and letting this derivative equal zero. That is,

$$\frac{dc(t)}{dt} = -\omega_n \zeta e^{-\zeta\omega_n t}\left(\frac{\zeta}{\sqrt{1-\zeta^2}} \sin(\omega_d t) + \cos(\omega_d t)\right) - \omega_d e^{-\zeta\omega_n t}\left(-\frac{\zeta}{\sqrt{1-\zeta^2}} \cos(\omega_d t) + \sin(\omega_d t)\right)$$

It follows that

$$\sin(\omega_d t) = 0$$

Since the peak time corresponds to the first peak overshoot, we have $\omega_d t_p = \pi$. Then

$$t_p = \frac{\pi}{\omega_d} \tag{2.8}$$

The peak time t_p corresponds to one half-cycle of the frequency of damped oscillation.

Maximum overshoot M_p The maximum overshoot occurs at the peak time $t_p = \pi/\omega_d$. Thus from Equation 2.4

$$\begin{aligned} M_p &= c(t_p) - 1 \\ M_p &= e^{-\zeta\omega_n(\pi/\omega_d)}\left(\frac{\zeta}{\sqrt{1-\zeta^2}}\sin(\pi) + cos(\pi)\right) \\ &= e^{-\zeta\pi/\sqrt{1-\zeta^2}} \end{aligned} \tag{2.9}$$

Since $c(\infty) = 1$, the maximum percent overshoot is $e^{-\zeta\pi/\sqrt{1-\zeta^2}}$ x 100%

Settling time t_s. For an underdamped second-order system, the transient response for a unit-step input is given by Equation 2.4. Notice that the response curve c(t) always remains within a pair of the envelope curves, as shown in Figure 2.12. The curves $1 \pm e^{-\zeta\omega_n t}/\sqrt{1-\zeta^2}$ are the envelope curves of the transient response to a unit-step input. The time constant of these envelope curves is $1/\zeta\omega_n$, ... The settling time t_s, is four times this time constant, or

$$t_s = \frac{4}{\zeta\omega_n}$$

Note that the settling time is inversely proportional to the undamped natural frequency of the system. Since the value of ζ is usually determined from the requirement of permissible maximum overshoot, the settling time is determined primarily by the undamped natural frequency ω_n. In other words, the duration of the transient period can be varied, without changing the maximum overshoot, by adjusting the undamped natural frequency ω_n. From the preceding analysis, it is clear that ω_n must be large if we are to have a rapid response. To limit the maximum overshoot M_p and make the settling time small, the damping ratio ζ should not be too small. Note that if the damping ratio is between 0.4 and 0.8, then the maximum percent overshoot for a step response is between 25% and 2.5%.

2.1.5 Root locus controller design

The basic characteristic of the transient response of a closed-loop system is closely related to the location of the closed-loop poles. If the system has a variable loop gain, then the location of the closed-loop poles depends on the value of the loop gain chosen. It is important, therefore, that the designer know how the closed-loop poles move in the s-plane as the loop gain is varied. From the design viewpoint, in some systems simple gain adjustment may move the closed-loop poles to desired locations. Then the design problem may become merely the selection of an

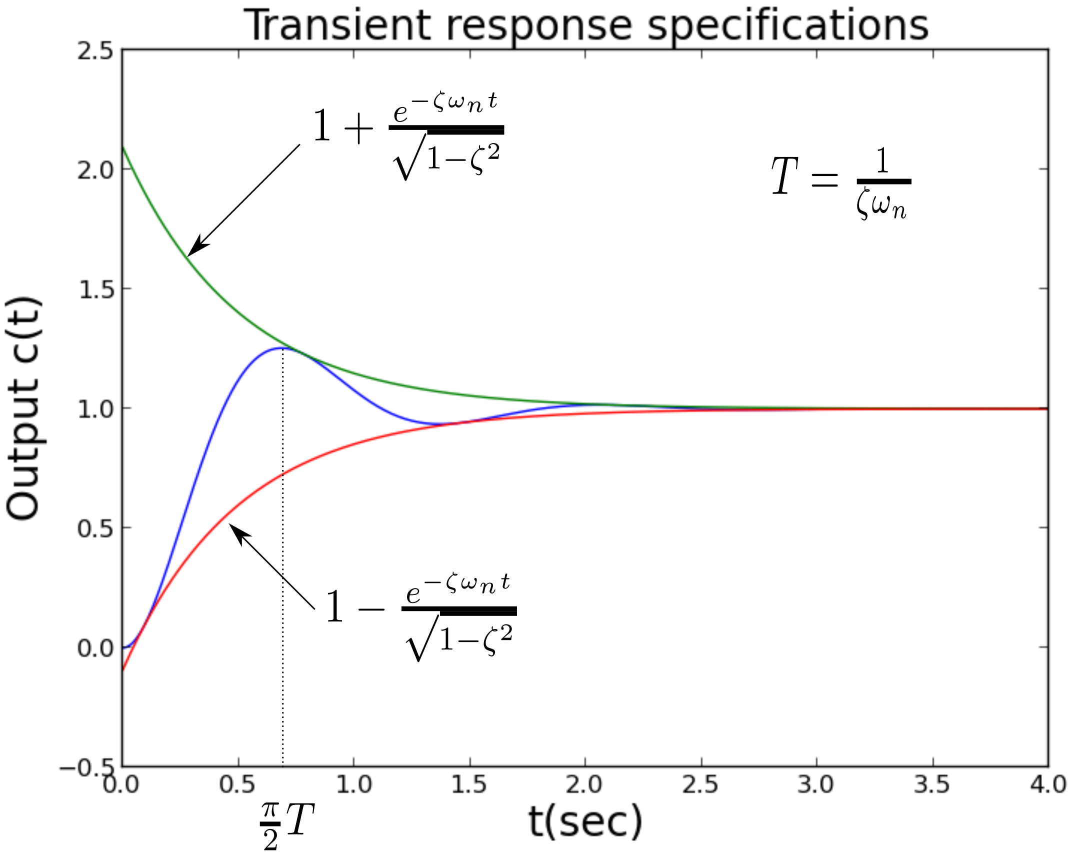

Figure 2.12: Unit-step response curve and its envelope curves.

appropriate gain value. If, however, the gain adjustment alone does not yield a desired result, the addition of a compensator to the system will become necessary. A simple method for finding the roots of the characteristic equation has been developed by W. R. Evans and is used extensively in control engineering. In this method, called the root-locus method, the roots of the characteristic equation are plotted for all values of a system parameter. The roots corresponding to a particular value of this parameter can then be located on the resulting graph. Note that the parameter is usually the gain, but any other variable of the open-loop transfer function may be used. Unless otherwise stated, we shall assume that the gain of the open-loop transfer function is the parameter to be varied through all values, from zero to infinity. By using the root-locus method. the designer can predict the effects on the location of the closed-loop poles of varying the gain value or adding open-loop poles and/or open-loop zeros. Therefore, it is desired that the designer have a good understanding of the method for generating the root loci of the closed-loop system, both by hand and with the use of computer software.

Root-locus method.

The basic idea behind the root-locus method is that the values of s that make the transfer function around the loop equal to - 1 must satisfy the characteristic equation of the system.

The locus of roots of the characteristic equation of the closed-loop system as the gain is varied from zero to infinity gives the method its name. Such a plot clearly shows the contributions of each open-loop pole or zero to the locations of the closed loop poles.

In designing a linear control system, we find that the root-locus method proves quite useful, since it indicates the manner in which the open-loop poles and zeros should be modified so that the response meets system performance specifications. The method is particularly suited to obtaining approximate results very quickly.

Angle and magnitude conditions.

Consider the system shown in Figure 2.13 The closed-loop transfer function is

$$\frac{C(s)}{R(s)} = \frac{G(s}{1 + G(s)H(s)} \tag{2.10}$$

The characteristic equation for this closed-loop system is obtained by setting the denominator of the right-hand side of Equation 2.10 equal to zero. That is,

$$1 + G(s)H(s) = 0$$

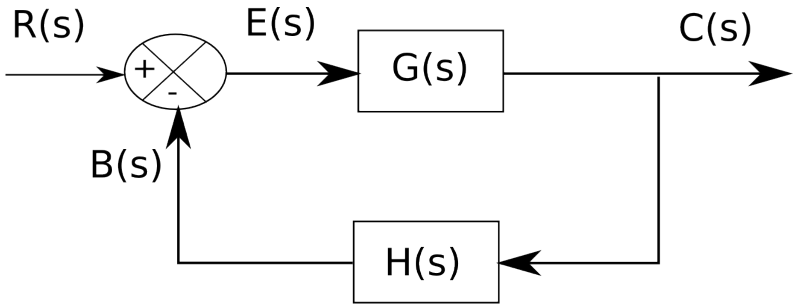

Figure 2.13: Closed-loop control system.

or

$$G(s)H(s) = -1 \tag{2.11}$$

Here, we assume that G(s)H(s) is a ratio of polynomials in s. Since G(s)H(s) is a complex quantity, Equation 2.11 can be split into two equations by equating the angles and magnitudes of both sides, respectively, to obtain the following:
Angle condition:

$$\angle G(s)H(s) = \pm 180^0(2k+1) \quad k = 0, 1, 2, \ldots \tag{2.12}$$

Magnitude condition:

$$|G(s)H(s)| = 1 \tag{2.13}$$

The values of s that fulfill both the angle and magnitude conditions are the roots of the characteristic equation, or the closed-loop poles. A plot of the points in the complex plane satisfying the angle condition alone is the root locus. The roots of the characteristic equation (the closed-loop poles) corresponding to a given value of the gain can be determined from the magnitude condition. The details of applying the angle and magnitude conditions to obtain the closed-loop poles are presented later in this section.
In many cases, G(s)H(s) involves a gain parameter K, and the characteristic equation may be written as

$$1 + \frac{K(s+z_1)(s+z_2)\ldots(s+Z_m)}{(s+p_1)(s+p_2)\ldots(s+p_n)} = 0 \tag{2.14}$$

Then the root loci for the system are the loci of the closed-loop poles as the gain K is varied from zero to infinity.

Root-locus plots with software

In this section, we present the software programs approach to generating root-locus plots.

In plotting root loci deal with the system equation given in the form of Equation 2.14, which may be written as

$$1 + K\frac{\text{num}}{\text{den}} = 0 \tag{2.15}$$

where num is the numerator polynomial and den is the denominator polynomial. Note that both vectors num and den must be written in descending powers of s. A software command commonly used for plotting root loci is

```
rlocus(num,den)
```

With this command, the root-locus plot is drawn on the screen. The gain vector K is automatically determined. Note that command

```
rlocus(num,den, K)
```

utilizes the user-supplied gain vector K. (The vector K contains all the gain values for which the closed-loop poles are to be computed.)
If the preceding two commands are invoked with left-hand arguments, that is

```
[r,K]=rlocus(num,den)
[r,K]= rlocus(num,den,K)
```

the matrix r containing the locations of complex roots and gain vector K. The plot command

```
plot(r,'-')
```

plots the root loci. Note that, the gain vector is determined automatically.

Example 2.1.4. *Consider a system whose open-loop transfer function G(s)H(s) is*

$$G(s)H(s) = \frac{K}{s(s+0.5(*s^2+0.6s+10)}$$

To set the plot region on the screen to be square, enter the command axis ('equal'). With this command, a line with unity slope is at a true 45 angle and is not skewed by the irregular shape of the screen. Entering MATLAB Program 2.14 into the computer, we obtain the root-locus plot shown in Figure 2.15.

Matlab/Octave code	Python code

>> num =[1];
>> den = [1 1.1 10.3 5 0];
>> [r, K]= r1ocus(num,den);
>> plot(r, '-');
>> axis('equal');
>> v=[-4 4 -4 4];
>> axis(v)
>> grid
>> title(' Root-Locus Plot of
G(s) K/[s(s+0.5)(s^2+0.6s+1 0)]')
>> xlabel ('Real Axis');
>> ylabel ('Imaginary Axis');
	from matplotlib.pyplot import *
	from control.matlab import *
	
	num = [1]
	den=[1,1.1,10.3,5,0]
	r,k = rlocus(num, den)
	axis('equal')
	v=[-4,4,-4,4]
	axis(v)
	grid()
	title(' Root-Locus Plot of
	G(s) K/[s(s+0.5)(s^2+0.6s+1 0)]')
	show()

Figure 2.14: Matlab/Octave/Python program 2.14.

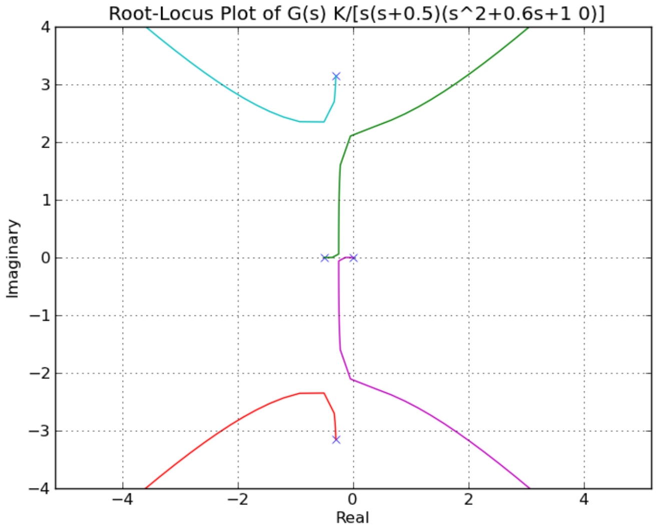

Figure 2.15: Root-locus plot

2.1.6 Practical design of PID controllers

It is interesting to note that more than half of the industrial controllers in use today utilize PID or modified PID control schemes. Analog PID controllers are mostly hydraulic, pneumatic, electric, and electronic types or their combinations. Currently, many of these are transformed into digital forms through the use of microprocessors.

Because most PID controllers are adjusted on-site, many different types of tuning rules have been proposed in the literature. With these tuning rules, PID controllers can be delicately and finely tuned. Also, automatic tuning methods have been developed, and some PID controllers may possess on-line automatic tuning capabilities. Many practical methods for bumpless switching (from manual opera tion to automatic operation) and gain scheduling are commercially available.

The usefulness of PID controls lies in their general applicability to most control systems. In the field of process control systems, it is a well-known fact that both basic and modified PID control schemes have provcd their usefulness in providing satisfactory control, although they may not provide optimal control in many given situations.

PID control of plants.

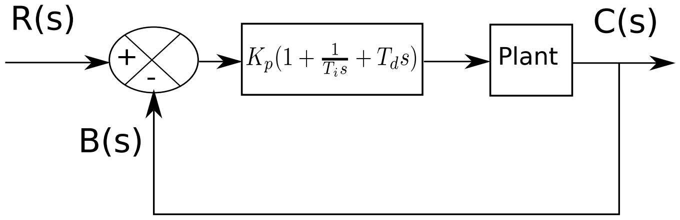

Figure 2.16: PID control of a plant.

Figure 2.16 shows the PID control of a plant. If a mathematical model of the plant can be derived, then it is possible to apply various design techniques for determining the parameters of the controller that will meet the transient and steady-state specifications of the closed-loop system. However, if the plant is so complicated that its mathematical model cannot be easily obtained, then an analytical approach to the design of a PID controller is not possible. In that case, we must resort to **experimental approaches** to the tuning of PID controllers.

The process of selecting the controller parameters to meet given performance specifications is known as **controller tuning**. Ziegler and Nichols suggested rules for tuning PID controllers (to set values K_p, T_i, and T_d) based on experimental step

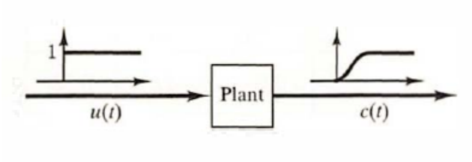

Figure 2.17: Unit-step response of a plant.

responses or based on the value of K_p that results in marginal stability when only the proportional control action is used. Ziegler-Nichols rules, presented next, are convenient when mathematical models of plants are not known. (These rules can, of course, be applied to the design of systems with known mathematical models.)

Ziegler-Nichols rules for tuning PID controllers.

Ziegler and Nichols proposed rules for determining values of the proportional gain K_p, integral time T_i, and derivative time T_d based on the transient-response characteristics of a given plant. Such determination of the parameters of PID controllers or tuning of PID controllers can be made by engineers on-site by experimenting on the plant. (Numerous tuning rules for PID controllers have been proposed since Ziegler and Nichols offered their rules. Here, however, we introduce only the Ziegler-Nichols tuning rules.) The two methods called Ziegler-Nichols tuning rules are aimed at obtaining 25% maximum overshoot in the step response.

First method.
In the first Ziegler-Nichols method, we obtain the response of the plant to a unit-step input experimentally, as shown in Figure 2.17. If the plant involves neither integrators nor dominant complex-conjugate poles, then such a unit-step response curve may look like an S-shaped curve, as shown in Figure 2.18. (If the response does not exhibit an S-shaped curve, this method does not apply.) Step-response curves of this nature may be generated experimentally or from a dynamic simulation of the plant. The S-shaped curve may be characterized by two constants-the delay lime L and a time constant T determined by drawing a line tangent to the S-shaped curve at the inflection point. These constants are determined by the intersections of the **tangent line** with the time axis and the line C(t) = K, as shown in Figure 2.18. The transfer function C(s)/U(s) may then be approximated by a first-order system with a transport lag as follows:

$$\frac{C(s)}{U(s)} = \frac{ke^{-Ls}}{Ts+1}$$

Ziegler and Nichols suggested setting the values of K_p, T_i, and T_d, according to the

Type of controller	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

Table 2.1: Ziegler-Nichols Tuning Rule Based on Step Response of Plant (First Method).

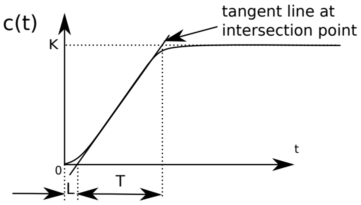

Figure 2.18: S-shaped response curve.

formula shown in Table **??**.

Second method.

In the second Ziegler-Nichols method, we first set $T_i = \infty$ and $T_d = O$. Using the proportional control action only (see Figure 2.19), we increase K_p from 0 to a critical value K_{cr} at which the output first exhibits sustained oscillations. (If the output does not exhibit sustained oscillations for whatever value K_p may take, then this method does not apply.) Thus the critical gain K_{cr} and the corresponding period P_{cr} are determined experimentally. (See Figure 2.20.) Ziegler and Nichols suggested that we set the values of the parameters K_p, T_i and T_d, according to the formula shown in Table 2.2.

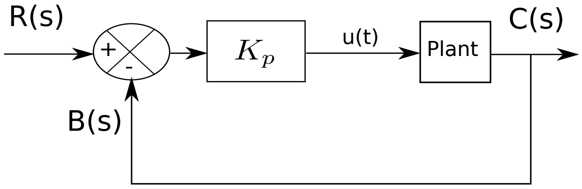

Figure 2.19: Closed-loop system with a proportional controller.

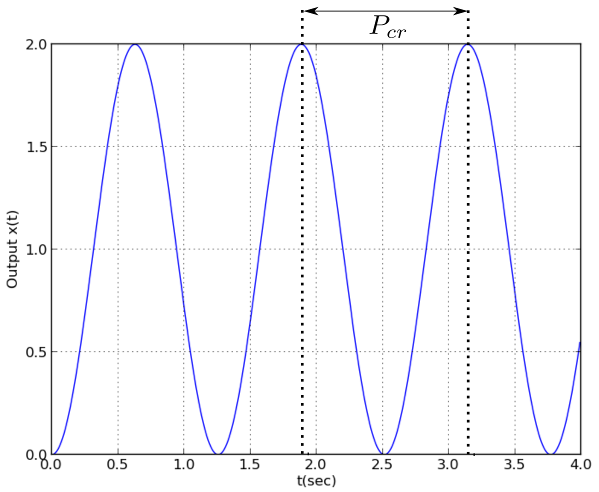

Figure 2.20: Sustained oscillation with per Period P_{cr}.

Type of controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Table 2.2: Ziegler-Nichols Tuning Rule Based on Critical Gain K_{cr} and Critical Period P_{cr} (Second Method).

Comments.

Ziegler-Nichols tuning rules (and other lUning rules presented in the literature) have been widely used to tune PID controllers in process control systems for which the plant dynamics are not precisely known. Over many years, such tuning rules have proved to be highly useful. Ziegler-Nichols luning rules can, of course, be applied to plants whose dynamics are known. (If plant dynamics are known, many analytical and graphical approaches to the design of PID controllers are available in addition to Ziegler-Nichols tuning rules.)

If the transfer function of the plant is known, a unit-step response may be calculated or Ihe critical gain K_{cr} and critical period P_{cr} may be calculated. Then, with those calculated values, it is possible to determine the parameters K_p, T_i, and T_d from Tables 2.1,2.2. However, the real usefulness of Ziegler-Nichols tuning rules (and other tuning rules) becomes apparent when the plant dynamics are **not**

known, so that no analytical or graphical approaches to the design of controllers are available.

Generally, for plants with complicated dynamics, but **no integrators**, Ziegler-Nichols tuning rules can be applied. If, however, the plant has an integrator, the rules may not be applicable in some cases, either because the plant does not exhibit the S-shaped response or because the plant does not exhibit sustained oscillations no matter what gain K is chosen. If the plant is such that Ziegler-Nichols rules can be applied, then the plant with a PID controller tuned by such rules will exhibit approximately 10% to 60% maximum overshoot in step response. On the average (obtained by experimenting on many different plants), the maximum overshoot is approximately 25%. (This is quite understandable, because the values suggested in Tables 2.1 and 2.2 are based on the average.) In a given case, if the maximum overshoot is excessive, it is always possible (experimentally or otherwise) to fine-tune the closed-loop system so that it will exhibit satisfactory transient responses. In fact, Ziegler-Nichols tuning rules give an educated guess for the parameter values and provide a starting point for fine-tuning.

2.1.7 Frequency domain analysis and control of dynamic systems

This chapter deals with the frequency-response approach to the analysis and design of control systems. By the term frequency response, we mean the steady-state response of a system to a sinusoidal input.

An advantage of the frequency-response approach is that frequency-response tests are, in general, simple and can be made accurately by use of readily available sinusoidal signal generators and precise measurement equipment. Often. the transfer functions of complicated components can be determined experimentally by frequency-response tests.

In this chapter, we first present Bode diagrams (logarithmic plots). We then discuss the concept of the phase margin and gain margin is introduced. Finally, the frequency-response approach to the design of control systems is treated. Software approaches to obtain Bode diagrams and margins are included in this chapter. It is noted that although the frequency response of a control system presents a qualitative picture of the transient response, the correlation between the frequency and transient responses is indirect, except in the case of second-order systems. In designing a closed-loop system, we adjust the frequency-response characteristic of the open-loop transfer function by using several design criteria in order to obtain acceptable transient-response characteristics for the system.

2.1.8 Bode diagram of the frequency response

A useful way to represent frequency-response characteristics of dynamic systems is the Bode diagram. (Bode diagrams are also called logarithmic plots of frequency response.) In this section we treat basic materials associated with Bode diagrams, using first- and second-order systems as examples. We then discuss the problem of identifying the transfer function of a system from the Bode diagram.

Bode diagrams.

A sinusoidal transfer function may be represented by two separate plots, one giving the magnitude versus frequency of the function, the other the phase angle (in degrees) versus frequency. A Bode diagram consists of two graphs: a curve of the logarithm of the magnitude of a sinusoidal transfer function and a curve of the phase angle; both curves are plotted against the frequency on a logarithmic scale. The standard representation of the logarithmic magnitude of $G(j\omega)$ is $20log|G(j\omega)|$, where the base of the logarithm is 10. The unit used in this representation is the decibel, usually abbreviated dB. The main advantage of using a Bode diagram is that multiplication of magnitudes can be converted into addition.

Note that the experimental determination of a transfer function can be made simple if frequency-response data are presented in the form of a Bode diagram.

Cutoff frequency and bandwidth.

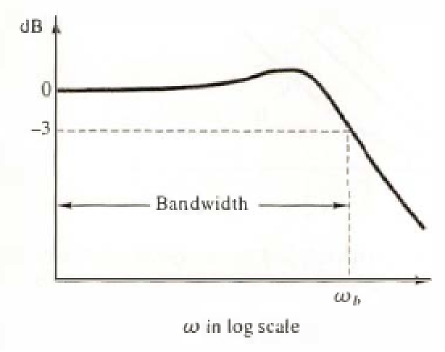

Figure 2.21: Logarithmic plot showing cutoff frequency ω_b and bandwidth.

Referring to Figure 2.21, the frequency at which the magnitude of the closed-loop frequency response is 3 dB below its zero-frequency value is called the cutof frequency. Thus,

$$\left|\frac{C(j\omega)}{R(j\omega)}\right| < \left|\frac{C(j0)}{R(j0)}\right| - 3dB \quad \text{for } \omega > \omega_{\mathrm{b}}$$

For systems in which

$$\left|\frac{C(j\omega)}{R(j\omega)}\right| < -3dB \quad \text{for } \omega > \omega_{\mathrm{b}}$$

The closed-loop system filters out the signal components whose frequencies are greater than the cutoff frequency and transmits those signal components with frequencies lower than the cutoff frequency.
The frequency range $0 \leq \omega \leq \omega_b$ which the magnitude of the closed loop does not drop - 3 dB is called the **bandwidth** of the system. The bandwidth indicates the frequency where the gain starts to fall off from its low-frequency value. Thus, the bandwidth indicates how well the system will track an input sinusoid. Note that, for a given ω_b the rise time increases with increasing damping ratio ζ. On the other hand, the bandwidth decreases with increasing ζ. Therefore, the rise time and the bandwidth are inversely proportional to each other.
The specification of the bandwidth may be determined by the following factors:

1. The **ability to reproduce the input signal**. A large bandwidth corresponds to a small rise time, or a fast response. Roughly speaking, we can say that the bandwidth is proportional to the speed of the response.

2. The necessary **filtering** characteristics for high-frequency noise.

For the system to follow arbitrary inputs accurately, it is necessary that it has a large bandwidth. From the viewpoint of noise, however, the bandwidth should not be too large. Thus, there are conflicting requirements on the bandwidth, and a compromise is usually necessary for good design. Note that a system with a large bandwidth requires high-performance components, so the cost of components usually increases with the bandwidth.

Cutoff rate.

The cutoff rate is the slope of the log-magnitude curve near the cutoff frequency. The cutoff rate indicates the ability of a system to distinguish a signal from noise. Note that a closed-loop frequency response curve with a steep cutoff characteristic may have a large resonant peak magnitude, which implies that the system has a relatively small stability margin.

Plotting Bode diagrams

In software, the command 'bode' computes magnitudes and phase angles of the frequency response of continuous-time, linear, time-invariant systems.
When the command 'bode' (without left-hand arguments) is entered in the computer, it produces a Bode plot on the screen. When invoked with lefthand arguments, as in

```
[mag,phase,w] = bode(num,den,w)
```

'bode' returns the frequency response of the system in matrices mag, phase, and w. No plot is drawn on the screen. The matrices mag and phase contain the magnitudes and phase angles respectively, of the frequency response of the system, evaluated at user-specified frequency points. The phase angle is returned in degrees. The magnitude can be converted to decibels with tbe statement

```
magdB =20*log10(mag)
```

To specify the frequency range, use the command logspace(d1,d2) or logspace(d1,d2,n). logspace(d1,d2) generates a vector of 50 points logarithmically equally spaced between decades 10^{d1} and 10^{d2}. That is, to generate 50 points between 0.1 rad/s and 100 rad/s, enter the command

```
w = logspace(-1,2)
```

logspace(d1,d2,n) generates n points logarithmically equally spaced between decades 10^{d1} and 10^{d2}. For example, to generate 100 points between 1 rad/s and 1000 rad/s, enter the following command:

```
w = logspace(0,3,100)
```

To incorporate these frequency points when plotting Bode diagrams, use the command bode(num,den,w), each of which employs the user-specified frequency vector w.

Example 2.1.5. *Plot a Bode diagram of the transfer function*

$$G(s) = \frac{25}{s^2 + 4s + 25}$$

When the system is defined in the form

$$G(s) = \frac{num(s)}{den(s)}$$

use the command bode(num,den) to draw the Bode diagram. [When the numerator and denominator contain the polynomial coefficients in descending powers of s, bode(num,den) draws the Bode diagram.) Program 2.22 plots the Bode diagram for this system. The resulting Bode diagram is shown in Figure 2.23.

Matlab/Octave code	Python code

`>> num = [25];`
`>> den=[1 4 25];`
`>> bode(num,den)`
`>> grid`
`>> title(’ Bode Diagram`
`of G(s) = 25/(51\2+45+25)’)`
	`from matplotlib.pyplot import *`
	`from control.matlab import *`
	
	`num = [25]`
	`den=[1,4,25]`
	`sys=tf(num,den)`
	`bode(sys)`
	`grid()`
	`title(’Bode Diagram of G(s)=25/(sˆ2+4s+25)’)`
	`show()`

Figure 2.22: Matlab/Octave/Python program 2.22.

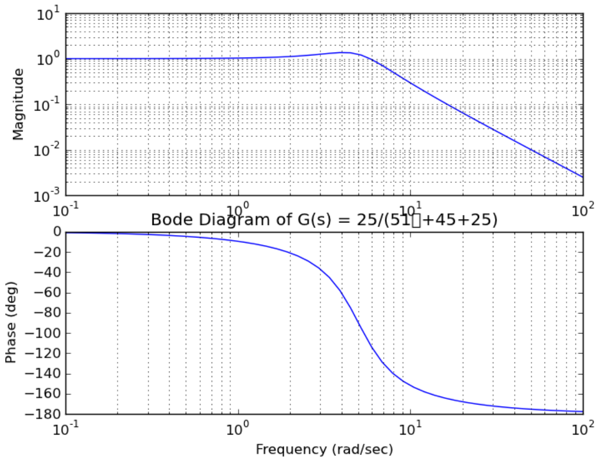

Figure 2.23: Bode diagram of $G(s) = \frac{25}{s^2+4s+25}$

Phase and gain margins.

As the gain is decreased to a certain value, the $G(j\omega)$ locus passes through the $-1 + jO$ point. This means that, with this gain, the system is on the edge of instability and will exhibit sustained oscillations. For a small value of the gain K, the system is stable.
In general, the closer the $G(j\omega)$ locus comes to encircling the $-1 + jO$ point, the more oscillatory is the system response. The closeness of the $G(j\omega)$ locus to the $-1 + jO$ point can be used as a measure of the margin of stability. (This does not apply, however, to conditionally stable systems.) It is common practice to represent the closeness in terms of phase margin and gain margin.

Phase margin.

The phase margin is that amount of additional phase lag at the gain crossover frequency required to bring the system to the verge of instability. The gain crossover frequency is the frequency at which $|G(j\omega)|$, the magnitude of the open-loop transfer function, is unity. The phase margin γ is 180^0 plus the phase angle ϕ of the open-loop transfer function at the gain crossover frequency, or

$$\gamma = 180^0 + \phi$$

Figures 2.24 illustrate the phase margins of a stable system and an unstable system in Bode diagrams. In the Bode diagrams, the critical point in the complex plane corresponds to the O-dB line and -180^0 line.

Gain margin.

The gain margin is the reciprocal of the magnitud e $|G(j\omega)|$ at the frequency at whIch the phase angle is -180^0. Defining the phase crossover frequency ω_l to be the frequency at which the phase angle of the open-loop transfer function equals -180^0 gives the gain margin K_g:

$$K_g = \frac{1}{G(j\omega_l)}$$

In terms of decibels,

$$K_g dB = 20 log Kg = -20 log|G(j\omega_l)|$$

Expressed in decibels, the gain margin is positive if K_g is greater than unity and negative if K_g is smaller than unity. Thus, a positive gain margin (in decibels) means that the system is stable, and a negative gain margin (in decibels) means that the system is unstable. The gain margin is shown in Figures 2.24.
For a stable minimum-phase system, the gain margin indicates how much the gain can be increased before the system becomes unstable. For an unstable system, the gain margin is indicative of how much the gain must be decreased to make the system stable.

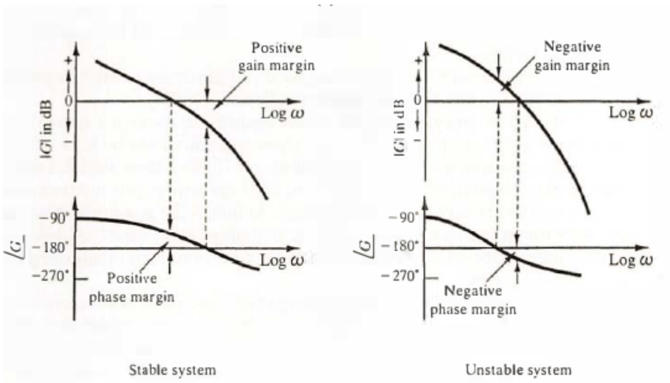

Figure 2.24: Phase and gain margins or stable and unstable systems: Bode diagrams.

A few comments on phase and gain margins.

The phase and gain margins of a control system are a measure of the closeness of loci the $-1 + jO$ point. Therefore, these margins may be used as design criteria.

Note that either the gain margin alone or the phase margin alone does not give a sufficient indication of relative stability. Both should be given in the determination of relative stability.

For a minimum-phase system, the phase and gain margins must be positive for the system to be stable. Negative margins indicate instability. Proper phase and gain margins ensure against variations in a system's components. For satisfactory performance, the phase margin should be between 30^0 and 60^0, and the gain margin should be greater than 6 dB. With these values, a minimum-phase system has guaranteed stability, even if the open-loop gain and time constants of the components vary to a certain extent. Although the phase and gain margins give only rough estimates of the effective damping ratio of a closed-loop system, they do offer a convenient means for designing control systems or adjusting the gain constants of systems.

For minimum-phase systems, the magnitude and phase characteristics of the open-loop transfer function are definitely related. The requirement that the phase margin be between 30 and 60 means thal, in a Bode diagram, the slope of the log-magnitude curve at the gain crossover frequency is more gradual than -40dB/decade. In most practical cases, a slope of -20 dB/decade is desirable at the gain crossover frequency for stability. If the slope is -40 dB/decade, the system could be either stable or unstable. (Even if the system is stable, however, the phase margin is smaiL) I f Ihe slope at Ihe gain crossover frequency is -60 dB/decade or steeper, the system will be unstable.

For a nonminimum-phase system with unstable open loop, the stability condition will not be satisfied unless the $G(j\omega)$ plot encircles the - 1 + jO point. Hence, such a stable nonminimum-phase system will have negative phase and gain margins. It is also important to point out that conditionally stable systems will have two or more phase crossover frequencies, and some higher order systems with complicated numerator dynamics may also have two or more gain crossover frequencies, as shown in Figure 2.24. For stable systems having two or more gain crossover frequencies, the phase margin is measured at the highest gain crossover frequency.

Determining margins with software

In software, the command 'margin' computes the amplitude and phase margin value and frequency of the frequency response of continuous-time, linear, time-invariant systems.

```
[gm,pm,wg,wp] = margin(sys)
```

'margin' returns the margins of the system in the vector value gm (gain) and pm (phase, together with the correspond frequency wg, wp. No plot is drawn on the screen.

Program 2.25 determines the margins for the system (sys).

Matlab/Octave code	Python code

`>> num = [25];`
`>> den=[1 4 25 2];`
`>> sys =tf(num,den);`
`>> margin(sys)`
`(3.516, 86.580, 4.715, 0.910)`
	`from matplotlib.pyplot import *`
	`from control.matlab import *`
	
	`num = [25]`
	`den=[1,4,25,2]`
	`sys=tf(num,den)`
	`margin(sys)`
	`(3.516, 86.580, 4.715, 0.910)`

Figure 2.25: Matlab/Octave/Python program 2.25.

2.2 Analysis of discrete control systems

2.2.1 Design of a digital controller

During recent decades, the design procedures for analog control systems have been well formulated and a large body of knowledge accumulated. The analog-design methodology, based on conventional techniques of root locus and Bode plots or the timing method of Ziegler and Nichols, may be applied to designing digital control systems. The procedure would be to first design the analog form of the controller or compensator to meet a particular set of performance specifications. Having done this, the analog form can be transformed to a discrete-time formulation. This approach is based on the fact that a digital system with a high sampling rate approximates to an analog system. The justification for using digital control under these circumstances must be that the practical limitations of the analog controller are overcome, the implementation **cheaper**, or that the **supervisory control** and **communications** more easily implemented.

However, the use of high sampling rates wastes computer power, and can lead to problems of arithmetic precision, etc. One is therefore driven to find methods of design which take account of the sampling process.

The alternative approach is to design controllers directly in the discrete-time domain based on the time-domain specifications of closed-loop system response. The controlled plant is represented by a discrete-time model which is a continuous-time system observed, analyzed, and controlled at discrete intervals of time. Since the time response is the ultimate objective of the design. This approach, which we shall now consider, provides a direct path to the design of digital controllers. The features of direct digital design are that sample rates are generally lower than for discretized analog design, and the design is directly performance based.

Figure 2.26 shows the basic structure of a digital control system. The design problem generally evolves around the choice of the control function D(z) in order to import a satisfactory form to the closed-loop transfer function. The choice is constrained by the form of the function $G_{h0}G(z)$ representing the fixed process elements.

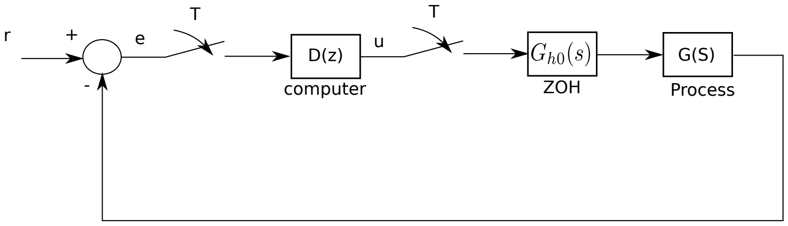

Figure 2.26: Basic structure of a digital controller.

A wide variety of digital-design procedures is available; these fall into two categories:

1. **direct synthesis procedures**, and
2. **iterative design procedures**.

The direct synthesis procedures assume that the control function D(z) is not restricted in any way by hardware or software limitations and can be allowed to take any form demanded by the nature of the fixed process elements and the specifications of the required system performance. This design approach has found wider applications in digital control systems than has the equivalent technique used with the analog systems. In a digital control system, realization of the required D(z) may involve no more than programming a special purpose software procedure. With analog systems, the limitation was in terms of the complications involved in designing special purpose analog controllers.

The design obtained by a direct synthesis procedure will give perfect nominal performance. However, the performance may be inadequate in the field because of the sensitivity of the design to plant disturbances and modeling errors. It is important that a control system is robust in its behavior with respect to the discrepancies between the model and the real process and uncertainties in disturbances acting on the process. Robustness property of some of the standard control structures, such as a three-term (PID) control algorithm, has been very well established. The design of such algorithms calls for an iterative design procedure where the choice of control function D(z) is restricted to using a standard algorithm with variable parameters; the designer must then examine the effect of the choice of controller parameters on the system performance and make an appropriate final choice.

The iterative design procedures for digital control systems are similar to the techniques evolved for analog system design using root locus and frequency response plots.

This chapter is devoted to the design of digital control algorithms, both by iterative design and direct synthesis procedures.

2.2.2 Z-plane specifications of control system design

Different properties that describe the performance of a feedback control system (Fig.) are listed below.

- Stability
- Steady-state accuracy
- Transient accuracy

Stability

Stability is a very important property of closed-loop systems. Almost every working system is designed to be stable. We seek to improve the system performance within the constraints imposed by stability considerations.

Steady-State Accuracy

Steady-state accuracy refers to the requirement that after all transients become negligible. The error between the reference input r and the controlled output y must be acceptably small. The specification on steady-state accuracy is often based on polynomial inputs of degree k: $r(t) = \frac{t^k}{k!}u(t)$. If $k = 0$, the input is a step of unit amplitude; if $k = 1$, the input is a ramp with unit slope; and if $k = 2$, the input is a parabola with unit second derivative. From the common problems of mechanical motion, these inputs are called, respectively, position, velocity, and acceleration inputs.
For quantitative analysis, we consider the unity-feedback discrete-time system shown in Fig.2.26. The steady-state error is the difference between the reference input r(k) and the controlled output y(k) when steady-state is reached, i.e., steady-state error

$$e^*_{ss} = \lim_{k\to\infty}[r(k) - y(k)] \tag{2.16}$$

Using the final value theorem,

$$e^*_{ss} = \lim_{Z-\to 1}[(z-1)E(z)] \tag{2.17}$$

provided that $(Z--1)E(z)$ has no poles on the boundary and outside of the unit circle in the z-plane.
For the system shown in Fig 2.26, define

$$G_{h0}G(z) = (1 - z^{-1})\mathcal{Z}\left[\frac{G(s)}{s}\right] \tag{2.18}$$

Then we have $\frac{Y(s)}{R(z)} = \frac{G_{h0}G(z)}{1+G_{h0}G(z)}$ and

$$E(z) = R(z) - Y(z) = \frac{R(z)}{1 + G_{h0}G(z)}. \tag{2.19}$$

By substituting Eqn. 2.19 into Eqn. 2.17 we obtain

$$\begin{aligned} e^*_{ss} &= \lim_{Z-\to 1}[(z-1)E(z)] && (2.20)\\ &= \lim_{Z-\to 1}\left[(z-1)\frac{R(z)}{1 + G_{h0}G(z)}\right] && (2.21)\end{aligned}$$

Type of input	Steady-state error		
	Type-0 system	Type-1 system	Type-2 system
Unit-step	$\frac{1}{1+K_p}$	0	0
Unit-ramp	∞	$\frac{1}{K_v}$	0
Unit-parabolic	∞	∞	$\frac{1}{Ka}$

Figure 2.27: Steady-state errors for various inputs end systems. Where K_p represents the position error, K_v the velocity error and K_a the acceleration error.

Thus, the steady-state error of a discrete-time system with unity feedback depends on the reference input signal R(z), and the forward-path transfer function $G_{h0}G(z)$. By the nature of the limit in Eqn 2.21, we see that the result of the limit can be zero or can be a constant different from zero. Also the limit may not exist, in which case the final-value theorem does not apply. However, it is easy to see from basic definition (2.16) that $e^*_{ss} = \infty$ in this case anyway because E(z) will have a pole at Z= 1 that is of order higher than one. Discrete-time systems having a finite nonzero steady-state error when the reference input is a zero-order polynomial input (a constant) are labeled Type-0. Similarly, a system that has finite nonzero steady-state error to a first-order polynomial input (a ramp) is called a Type-1 system, and a system with finite nonzero steady-state error to a second-order polynomial input (a parabola) is called a Type-2 system.

Transient Accuracy

Specifications of transient performance may be made in the time domain in terms of rise time, peak time, peak overshoot, settling time. etc. The use of root locus plots for the design of digital control systems necessitates the translation of time-domain performance specifications into desired locations of closed-loop poles in the Z-plane. Whereas the use of frequency response plots necessitates the translation of time-domain specifications in terms of frequency response features such as bandwidth, phase margin, gain margin, resonance peak. resonance frequency, etc.

Specifications in terms of root locations in Z-plane

Our approach is to first obtain the transient response specifications in terms of characteristic roots in the s-plane, and then use the relation

$$z = e^{sT} \tag{2.22}$$

to map the s-plane characteristic roots to the Z-plane.

The unit-step response of an underdamped second-order system

$$\frac{Y(s)}{R(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2.23}$$

where
ζ = damping ratio, and ω_n = undamped natural frequency. The transient response specifications in terms of rise time t_r, peak time t_p, peak overshoot M_p, and settling time t_s, can be approximated to the parameters ζ and ω_n, of the second-order system defined by Eqn. (2.23). These relationships are already explained in Section **??**.
The specification on speed of response in terms of t_r, t_p, and/or t_s, should be consistent as all these depend on ζ and ω_n. The greater the magnitude of ω_n when ζ is constant, the more rapid the response approaches the desired steady-state value. The value of ω_n, is limited by measurement noise considerations - a system with large ω_n has large bandwidth and will therefore allow the high frequency noise signals to affect its performance.
We need to now convert the specifications on ζ and ω_n into guidelines on the placement of poles and zeros in the Z-plane in order to guide the design of digital controls. We do so through the mapping (2.22).

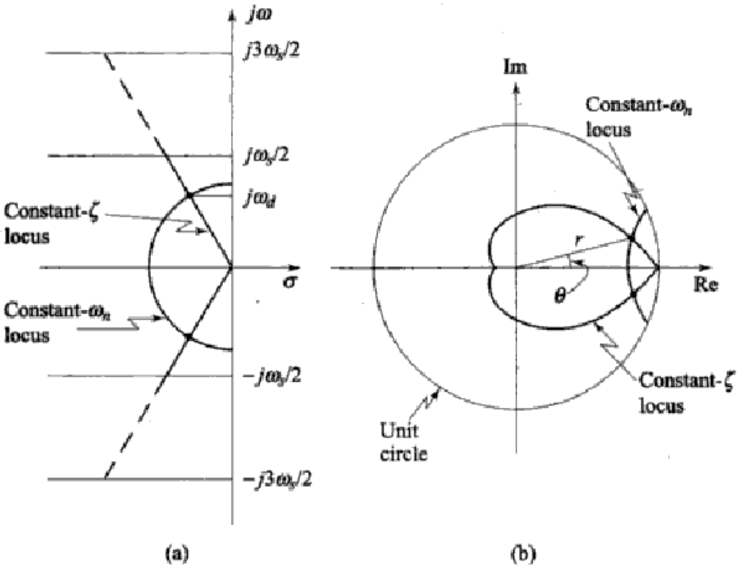

Figure 2.28: Mapping of s-plane patterns on the Z-plane.

Figure 2.28 illustrates the translation of specifications on ζ and ω_n, to the characteristic root locations in the Z-plane. The s-plane poles

$$s_{1,2} = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2} = -\zeta\omega_n \pm j\omega_d \tag{2.24}$$

for constant ζ, lie along a line in the s-plane (2.28a). In the Z-plane,

$$z_{1,2} = e^{-\zeta\omega_n T} e^{\pm j\omega_n\sqrt{1-\zeta^2}} = re^{j\theta} \tag{2.25}$$

The magnitude of Z-(i.e., the distance to the origin) is $r = e^{-\zeta\omega_n T}$ and the angles with the positive real axis of the Z-plane, measured positive in the counterclockwise direction, are $\theta = \omega_n T\sqrt{1-\zeta^2}$. It should be observed that the Z-plane pole locations depend on the s-plane positions as well as the sampling interval T.
As ω_n increases for a constant ζ. the magnitude of Z-decreases and the phase angle increases; constant-ζ locus is a logarithmic spiral in the Z-plane (Fig. 2.28b). Increasing ω_n negatively, gives the mirror image.

Dominant poles

Most control systems found in practice are of high order. The preferred locations of closed-loop poles given by Fig. 2.28b realize the specified transient performance only if the other closed-loop poles and zeros of the system have negligible effect on the dynamics of the system, i.e., only if the closed-loop poles corresponding to specified ζ and ω_m, are dominant.
In the following, we examine the relationship between the pole-zero patterns and the corresponding step-responses of discrete-time systems. Our attention will be restricted to the step responses of the discrete-time system with transfer function

$$\frac{Y(z)}{R(z)} = \frac{K(z-z_1)(z-z_2)}{(z-p)(z-re^{j\theta})(z-re^{-j\theta})} = \frac{K(z-z_1)(z-z_2)}{(z-p)(z^2-2r\cos\theta z+r^2)} \tag{2.26}$$

for a selected set of values of the parameters K, z_1, z_2, p, r and θ. We assume that the roots of the equation

$$z^2 - 2r\cos\theta z + r^2 = 0 \tag{2.27}$$

are the preferred closed-loop poles corresponding to the specified values of ζ and ω_n. Complex-conjugate pole pairs corresponding to $\zeta = 0.5$ with $\theta = 18^0$, 45^0 and 72^0 will be considered in our study. The pole pair with $\theta = 18^0$ is shown in Fig. 2.29.
To study the effect of zero location, we let $z_2 = p$ and explore the effect of the (remaining) zero location z_1, on the transient performance. We take the gain K to be such that the steady-state output value equals the step size. For a unit-step input,

$$Y(z) = \left[\frac{K(z-z_1)}{z^2-2rcos\theta z+r^2}\right]\left(\frac{z}{z-1}\right) \tag{2.28}$$

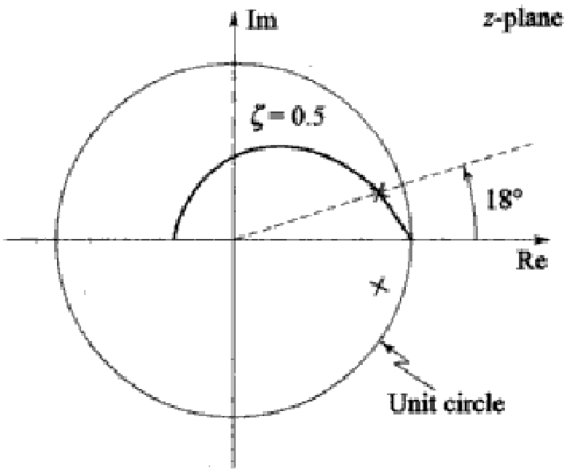

Figure 2.29: A complex-conjugate pole pair of the system .

with

$$K = \frac{1 - 2r\cos\theta z + r^2}{(1 - z_1)} \qquad (2.29)$$

The major effect of the zero z_1 on the step response y(k) is to change the peak overshoot, as may be seen from the step responses plotted in Fig. 2.30a.

Figure 2.30b shows plots of peak overshoot versus zero location for three different cases of complex-conjugate pole pairs. The major feature of these plots is that the zero has very little influence when on the negative real axis, but its influence is dramatic when it comes near +1.

To study the influence of a third pole on a basically second-order response, we again consider the system (2.29), but this time we fix $z_1 = z_2, = -1$ and let p vary from near -1 to near +1. In this case, the major influence of the moving singularity is on the rise time of the step response. Figure 2.30c shows plots of rise time versus extra pole location for three different cases of complex-conjugate pole pairs. We see here that the extra pole causes the rise time to get very much longer as the location of p moves toward z = +l and comes to dominate the response.

Our conclusions from these plots are that the addition of a pole or a zero to a given system has only a small effect if the added singularities are in the range 0 to -1. However, a zero moving toward z= +1 greatly increases the system overshoot. A pole placed toward z= +1 causes the response to slow down and thus primarily affects the rise time which is being progressively increased. The pole pair corresponding to specified ζ and ω_n, is a dominant pole pair of the closed-loop system only if the influence of additional poles and zeros is negligibly small on the dynamic response of the system.

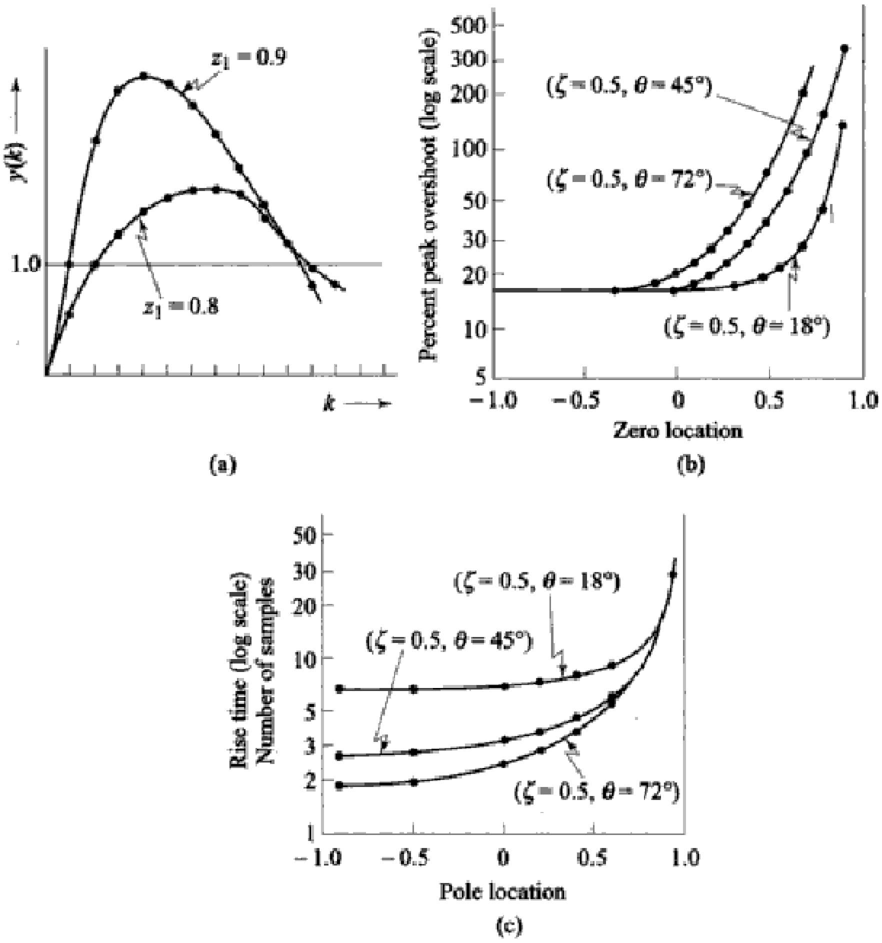

Figure 2.30: Effect of am extra zero on a discrete-time second order system, $\zeta = 0.5, \theta = 18^0$; (b) effect of an extra zero on a discrete-time second-order system; (c) effect of an extra pole on a discrete-time second-order system.

2.2.3 Digital compensator design using frequency response plots

All the frequency response methods of continuous-time systems are directly applicable for the analysis and design of digital control systems. For a system with closed-loop transfer function

$$\frac{Y(z)}{R(z)} = \frac{G_{h0}(z)G(z)}{1 + G_{h0}(z)G(z)} \tag{2.30}$$

the absolute and relative stability conditions can be investigated by making the frequency response plots of $G_{h0}G(z)$. The frequency response plots of $G_{h0}G(z)$ can be obtained by setting

$$z = e^{j\omega T}; \quad \text{T = sampling interval} \tag{2.31}$$

and then letting the frequency ω vary from $-\omega_s/2$ to $\omega_s/2$, $\omega_s = 2\pi/T$. Computer assistance is normally required to make the frequency response plots.
Since the frequency appears in the form $Z- = e^{j\omega T}$, the discrete-time transfer functions are typically not rational functions and the simplicity of Bodes design technique is all together lost in the Z-plane. The simplicity can be regained by transforming the discrete-time transfer function in the Z-plane to a different plane (called w) by the bilinear transformation

$$Z- = \frac{1 + wT/2}{1 - wT/2} \tag{2.32}$$

By solving Eqn. 2.32 for w, we obtain the inverse relationship

$$w = \frac{2}{T}\frac{z-1}{z+1} \tag{2.33}$$

Through the Z-transformation and the w-transformation, the primary strip of the left half of the s-plane is first mapped into inside of the unit circle in the Z-plane, and then mapped into the entire left half of the w-plane. The two mapping processes are depicted in Fig. 2.31. Notice that as s varies from 0 to $j\omega_s/2$ along the jωaxis in the s-plane, Z-varies from 1 to ∞ along the unit circle in the Z-plane, and w varies from 0 to on along the imaginary axis in the w-plane. The bilinear transformation (2.32) does not have any physical significance in itself and therefore all w-plane quantities are fictitious quantities that correspond to the physical quantifies of either the s-plane or the Z-plane. The correspondence between the real frequency ω and the fictitious w-plane frequency, denoted as ν, is obtained as followed:

$$j\nu = \frac{2}{T}\frac{e^{j\omega T} - 1}{e^{j\omega T} + 1} = \frac{2}{T}\frac{e^{j\omega T/2} - e^{-j\omega T/2}}{e^{j\omega T/2} + e^{-j\omega T/2}} = \frac{2}{T}j\tan\frac{\omega T}{2} \tag{2.34}$$

or

$$\nu = \frac{2}{T}\tan\frac{\omega T}{2} \tag{2.35}$$

Thus a nonlinear relationship or warping exists between the two frequencies w and ν. As w moves from 0 to $\omega_s/2$, ν moves from 0 to on ∞ (2.32).
Note that for relatively small $\frac{\omega T}{2} < 17^0$ or about 0.3 rad),

$$\nu = \frac{2}{T}\tan\frac{\omega T}{2} \approx \frac{2}{T}\left(\frac{\omega T}{2}\right) \approx \omega \tag{2.36}$$

and the warping effect on the frequency response is negligible.

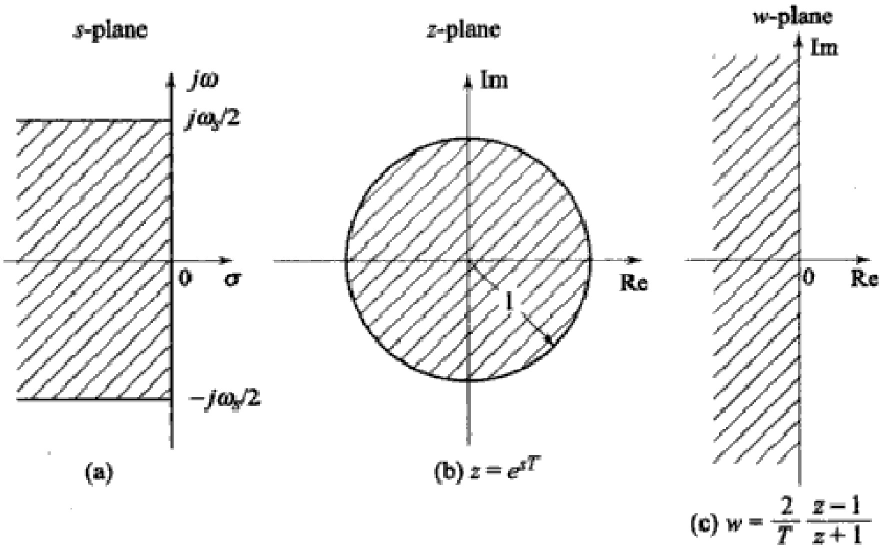

Figure 2.31: Diagrams showing mapping from s-plane to Z-plane and from Z-plane to w-plane.

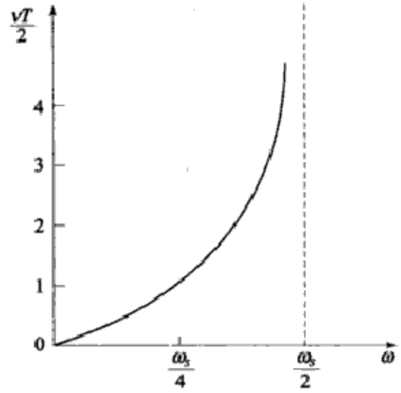

Figure 2.32: Relationship between the fictitious frequency ν and actual freqcuency ω.

Example 2.2.1. *Consider a process with transfer function*

$$G(s) = \frac{10}{s(\frac{1}{5}s + 1)} \tag{2.37}$$

which, when preceded by a ZOH (T = 0.1 sec) has the discrete-time transfer func-

tion

$$G_{h0}G(z) = (1 - z^{-1})\mathcal{Z}\left[\frac{50}{s^2(s+5)}\right] = \frac{0.215(z+0.85)}{(z-1)(z-0.61)} \tag{2.38}$$

Using the bilinear transformation

$$Z- = \frac{1 + \frac{wT}{2}}{1 - \frac{wT}{2}} = \frac{1 + 0.05w}{1 - 0.05w} \tag{2.39}$$

$G_{h0}G(z)$ can be transformed into $G_{h0}G(w)$ given below:

$$G_{h0}G(w) = \frac{10\left(1 - \frac{w}{20}\right)\left(1 + \frac{w}{246.67}\right)}{w\left(1 + \frac{w}{4.84}\right)} \tag{2.40}$$

Notice that the gain of $G_{h0}G(z)$ can be transformed into $G_{h0}G(w)$ is precisely the same as that of G(s) it is 10 in both the cases. This will always be true for a $G_{h0}G(w)$ computed using the bilinear transformation given by Eqn. 2.32. The gain of 10 in Eqn. 2.40 is the K_v of the uncompensated system 2.38 . We also note that in Eqn. 2.40 the denominator looks very much similar to that of G(s) and that the denominator: will be the some as T approaches zero. This would also have been true for any zeros of $G_{h0}G(w)$ that corresponded to zeros of G(s) but our example does not have any.

2.2.4 Digital compensator design using root locus plots

Design of compensation networks using the root locus plots is a well established procedure in analog control systems. This is essentially a trial-and-error method whereby varying the controller parameters, the roots of the characteristic equation are relocated to favorable locations.
The characteristic equation of a discrete-time system can always be written in the form

$$1 + F(z) = 0 \tag{2.41}$$

where F(z) is a rational function of z. From Eqn. 2.41, it is seen that the roots of the characteristic equation (i.e.. the closed-loop poles of the discrete-time system) occur only for those values of Z-where

$$F(z) = -1 \tag{2.42}$$

Sinze Z-is a complex variable, Eqn 2.42 is converted into two conditions given below.

1. **Magnitude condition**: $|F(z)| = 1$
2. **Angle condition**: $\angle F(z) = \pm 180^0(2q+1); \quad q = 0, 1, 2, \ldots$

Root-locus plots with software

In plotting root loci deal with the system equation given in the form of Equation 2.41, which may be written as

$$1 + K\frac{\text{num}}{\text{den}} = 0 \tag{2.43}$$

where num is the numerator polynomial and den is the denominator polynomial. Note that both vectors num and den must be written in descending powers of z. A software command used for plotting a discrete root loci is

```
TF=tf(num,den,T)
rlocus(TF)
```

where T represents the sampling time. With this command, the root-locus plot is drawn on the screen. The gain vector K is automatically determined. Note that command

```
rlocus(TF, K)
```

utilizes the user-supplied gain vector K. (The vector K contains all the gain values for which the closed-loop poles are to be computed.)
If the preceding two commands are invoked with left-hand arguments, that is

```
[r,K]= rlocus(TF)
[r,K]= rlocus(TF,K)
```

the matrix r has length K rows and length den-1 columns containing the locations of complex roots and gain vector K. The plot command

```
plot(r,'-')
```

plots the root loci. Note that, the gain vector is determined automatically.

Example 2.2.2. *Consider a system whose open-loop transfer function G(s)H(s) is*

$$G(s)H(s) = \frac{K}{s(s+2))}$$

which when preceded by a zero-order hold (T=0.2 sec) has the discrete function

$$\begin{aligned} G_{h0}G(z) &= (1-z^{-1})\mathcal{Z}\left[\frac{K}{s^2(s+2)}\right] && (2.44)\\ &= \frac{0.01758Z - +0.01539}{z^2 - 1.67Z - +0.6703} && (2.45)\end{aligned}$$

Entering MATLAB Program 2.33 into the computer. we obtain the rool-locus plot shown in Figure 2.34.

Matlab/Octave code	Python code

num =[1];
den = [1 2 0];
CTF=tf(num,den)
T=0.2;
DTF=c2d(CTF,T,'zoh');
r1ocus(DTF);
	Not possible yet !

Figure 2.33: Matlab/Octave/Python program 2.33.

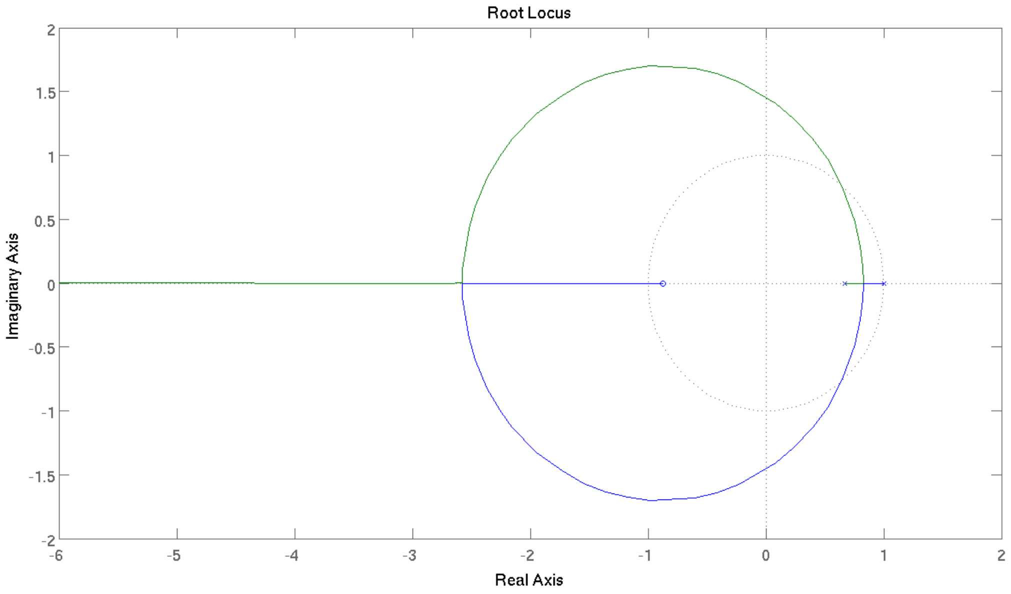

Figure 2.34: Discrete root-locus plot.

2.2.5 Z-plane synthesis

Much of the style of the transform domain techniques we have been discussing in this chapter grew out of the limitations of technology which was available for realization of the compensators with pneumatic components or electric networks and amplifiers. In the digital computer, such limitations on realization are of course, not relevant, and one can ignore these particular constraints.

One design method which eliminates these constraints begins from the very direct point of view that we are given a process (plus hold) transfer function $G_{h0}G(z)$. that we want to construct a desired transfer function, M(z), between input r and output y and that we have the computer transfer function, D(z), to do the job as per the feedback control structure of Fig. 2.35.

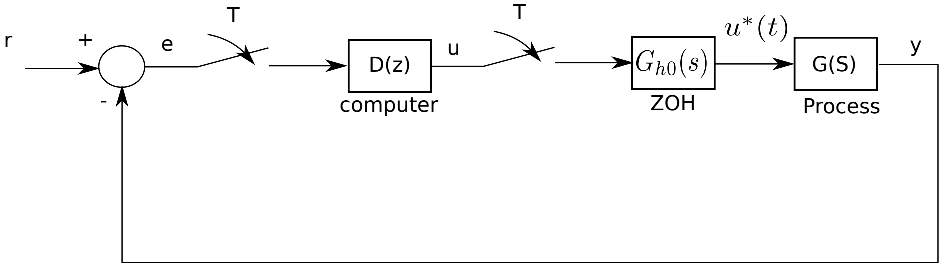

Figure 2.35: A feedback system with digital compensation.

The closed-loop transfer function is given by the formula

$$M(z) = \frac{D(z)G_{h0}G(z)}{1 + D(z)G_{h0}G(z)} \tag{2.46}$$

from which we get the design formula

$$D(z) = \frac{1}{G_{h0}G(z)}\left[\frac{M(z)}{1 - M(z)}\right] \tag{2.47}$$

As is seen from Eqn. 2.47, the controller transfer function consists of the inverse of the plant transfer function and the additional term which depends on the system closed-loop transfer function. Thus the design procedure outlined above looks for a D(z) which will cancel the process effects and add whatever is necessary to give the desired performance.

For prescribing the required closed-loop transfer function M(z), the following restrictions have to be noted.

Realizability of digital Controller

Assume that a digital controller

$$D(z) = \frac{Q_v(z)}{P_\mu(z)} = \frac{q_0 z^\nu + q_1 z^{\nu-1} + \ldots + q_\nu}{z^\mu + p_1 z^{\mu-1} + \ldots + p_\mu} \tag{2.48}$$

is cascaded with the process

$$G_{h0}G(z) = \frac{B_m(z)}{A_n(z)} = \frac{b_0 z^m + b_1 z^{m-1} + \ldots + b_m}{z^n + a_1 z^{n-1} + \ldots + a_n}; m \leq n \tag{2.49}$$

in the control loop given by Fig. 2.35. For D(z) to be physically realizable, $\nu \leq \mu$. The closed loop transfer function

$$M(z) = \frac{D(z)G_{h0}G(z)}{1 + D(z)G_{h0}G(z)} = \frac{Q_v(z)B_m(z)}{P_\mu(z)A_n(z) + Q_v(z)B_m(z)} = \frac{N_{\nu+m}(z)}{D_{\mu+n}(z)} \qquad (2.50)$$

The order of the numerator polynomial of M(z) is $\nu + m$, and the order of the denominator polynomial of M(z) is $\mu + n$. The pole excess of M(z) is therefore $\{(\mu - \nu) + (n - m)\}$

This means that because of the condition of realizability of digital controller, the pole excess of the closed-loop transfer function $M(z)$ has to be greater than or equal to the pole excess of the process transfer function $G_{h0}G(z)$.

If the digital controller D(z) given by Eqn. 2.47 and the process $G_{h0}G(z)$ are in a closed loop, the poles and zeros of the process are canceled by the zeros and poles of the controller. The cancellation is perfect if the process model $G_{h0}G(z)$ matches the process exactly. Since the process models used for design practically never describe the process behavior exactly, the corresponding poles and zeros will not be canceled exactly; the cancellation will be approximately.

For poles and zeros of $G_{h0}G(z)$ which are sufficiently spread in the inner of the unit disc in the Z-plane, the approximation in cancellation leads to only small deviations of the assumed behavior M(z) in general. However, one has to be careful if $G_{h0}G(z)$ has poles or zeros on or outside the unit circle. Imperfect cancellation may lead to weakly damped or unstable behavior. Therefore the design of digital controllers according to Eqn. 2.47 has to be restricted to cancellation of poles and zeros of $G_{h0}G(z)$ located inside the unit circle. This imposes certain restrictions on the desired transfer function M(z) as is seen below.

Assume that $G_{h0}G(z)$ involves an unstable (or critically stable) pole at $z = \alpha$. Let us define

$$G_{h0}G(z) = \frac{G_1(z)}{z - \alpha}$$

were $G_1(z)$ does not include a term that cancels with $(Z - -\alpha)$. Then the closed-loop transfer function becomes

$$M(z) = \frac{D(z)\frac{G_1(z)}{z-\alpha}}{1 + D(z)\frac{G_1(z)}{z-\alpha}} \qquad (2.51)$$

Since we require that no new zero of D(z) to cancel the pole of $G_{h0}G(z)$ at $Z- = \alpha$, we must have

$$1 - M(z) = \frac{1}{1 + D(z)\frac{G_1(z)}{z-\alpha}} = \frac{z - \alpha}{z - \alpha + D(z)G_1(z)} \qquad (2.52)$$

that is, $1 - M(z)$ must have $Z- = \alpha$ as a zero. This argument applies equally if $G_{h0}G(z)$ involves two or more unstable (or critically stable) poles.
Also note from Eqn. 2.51 that if poles of D(z) do not cancel zeros of $G_{h0}G(z)$, then the zeros of $G_{h0}G(z)$ become zeros of M(z).
Let us summarize what we have stated concerning cancellation of poles and zeros of $G_{h0}G(z)$.

1. Since the digital controller D(z) should not cancel unstable (or critically stable) poles of $G_{h0}G(z)$, all such poles of $G_{h0}G(z)$ must be included in $1 - M(z)$ as zeros.

2. Zeros of $G_{h0}G(z)$ that lie on or outside the unit circle should not be canceled with poles of D(z); all such zeros of $G_{h0}G(z)$ must be included in M(z) as zeros.

The design procedure thus essentially involves the following three steps:

1. The closed-loop transfer function M(z) of the final system is determined from the performance specifications, and the fixed parts of the system,i.e. $G_{h0}G(z)$

2. The transfer function D(z) of the digital controller is found using the design formula 2.47.

3. The digital controller D(z) is synthesized.

Step 1 is certainly the most difficult one to satisfy. In order to pass step 1, a designer must fulfill the following requirements.
(i) The digital controller D(z) must be physically realizable.
(ii) The poles and zeros of on $G_{h0}G(z)$ or outside the unit circle should not be canceled by D(z).
(iii) The system specifications on transient and steady-state accuracy be satisfied.

For common types of specifications (error constants, stability margins, speed of response. etc.), these requirements are usually easily satisfied. However, we can ask for more than what has been achieved earlier by compensation techniques based on root locus and frequency response plots. Design examples will best illustrate this point.

Example 2.2.3. *The plant of sampled-data system of Fig. 2.35 is described by the transfer function*

$$G(s) = \frac{1}{s(10s + 1)} \tag{2.53}$$

The sampling period is 1 sec.
The problem is to design a digital controller D(z) to realize the following specifications:
(i) $K_v \geq 1$
(ii) $\zeta = 0.5$, *and*
(iii) t_s*(2% tolerance band)* $\leq$ *8 sec.*
The selection of a suitable M(z) is described by the following steps.
(i) The z-transfer function of the plant is given by

$$\begin{aligned} G_{h0}G(z) &= (1-z^{-1})\mathcal{Z}\left[\frac{1}{s^2(10s+1)}\right] && (2.54) \\ &= 0.04837\frac{(z+0.9672)}{(z-1)(z-0.9048)} && (2.55) \end{aligned}$$

Since $G_{h0}G(z)$ *has one more pole than zero, M(z) must have a pole excess of at least one. (ii)* $G_{h0}G(z)$ *has a pole at* $z=1$*. This must be included in* $1-M(z)$ *as zero. i.e..*

$$1 - M(z) = (z-1)F(z) \qquad (2.56)$$

where F(z) is a ratio of polynomials of appropriate dimensions.
(iii) The transient accuracy requirements are specified as $\zeta = 0.5$, $\omega_n = 1$ *(*$t_s = 4/\zeta\omega_n$*). With a sampling period T=1 sec, this maps to a pair of dominant closed-loop poles in the z-plane with*

$$\begin{aligned} |z_{1,2}| &= e^{-\zeta\omega_n T} = 0.6065 && (2.57) \\ \angle z_{1,2} &= \pm\omega_n T\sqrt{1-\zeta^2} = \pm\frac{0.866\times 180}{3.14} = \pm 49.64^0 && (2.58) \end{aligned}$$

This corresponds to

$$z_{1,2} = 0.3928 \pm j0.4621 \qquad (2.59)$$

The closed-loop tranfer function M(z) should have dominant poles at the root of the equation

$$\Delta(z) = z^2 - 0.7856Z - +0.3678 \qquad (2.60)$$

The steady-state accuracy requirements demand that steady-state error to unit-step input is zero, and steady-state error to unit-rump input is less than $1/K_v$

$$\begin{aligned} E(z) &= R(z) - Y(z) = R(z)[1-M(z)] = R(z)(z-1)F(z) \\ e^*_{ss}|_{unit\ step} &= \lim_{Z-\to 1}(z-1)\frac{z}{z-1}(z-1)F(z) = 0 \end{aligned}$$

Thus, with the choice of M(z) given by Eqn. 2.56. the steady-state error to unit step input is always zero.

$$\begin{aligned} e^*_{ss}|_{ramp} &= \lim_{Z \longrightarrow 1} (z-1)\frac{Tz}{(z-1)^2}(z-1)F(z) = TF(1) = 1/K_v \\ T &= 1 \textit{ and } K_v = 1 \\ F(1) = 1 & \end{aligned} \tag{2.61}$$

From Eqn 2.56 and 2.60, we observe that

$$F(z) = \frac{z-\alpha}{z^2 - 0.7856z + 0.3678}$$

meets the requirements on reliability of D(z), cancellation of poles and zeros of $G_{h0}G(z)$ *, and transient accuracy. The requirement on steady-state accuracy is also met if we choose a such that (refer Eqn. 2.61)*

$$\frac{1-\alpha}{1 - 0.7856 + 0.3678} = 1 \tag{2.62}$$

This gives $\alpha = 0.4178$.
Therefore

$$\begin{aligned} F(z) &= \frac{z - 0.4178}{z^2 - 0.7856z + 0.3678} \\ 1 - M(z) &= \frac{(z-1)(z-0.4178)}{z^2 - 0.7856z + 0.3678} \\ M(z) &= \frac{0.6322z - 0.05}{z^2 - 0.7856z + 0.3678} \end{aligned} \tag{2.63}$$

Now turning to the basic design formula 2.47, we compute

$$\begin{aligned} D(z) &= \frac{1}{G_{h0}G(z)}\left[\frac{M(z)}{1-M(z)}\right] \\ &= \frac{(z-1)(z-0.9048)}{(0.04837)(z+0.9672)}\left[\frac{0.6322z - 0.05}{(z-1)(z-0.4178)}\right] \\ &= 13.07\frac{(z-0.9048)(z-0.079)}{(z+0.9672)(z-0.4178)} \end{aligned} \tag{2.64}$$

A plot of the step response of the resulting design is provided in Fig. 2.36. which shows the control effort.

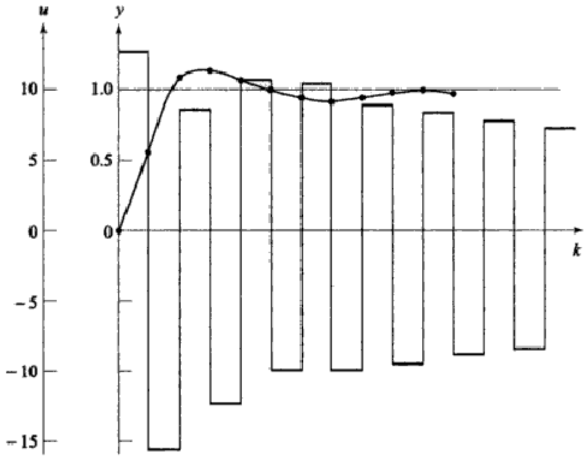

Figure 2.36: Step response to controlled system

2.3 Analysis of control systems in State Space

2.3.1 Usefull vector-matrix methods

Eigenvalues of an n $\times$ n matrix A.

The eigenvalues of an n roots of the characteristic equation n×n matrix **A** are the zeros of

$$|\lambda\mathbf{I} - \mathbf{A}| = 0.$$

The eigenvalues are also called the characteristic roots.
Consider, for example, the following matrix **A**:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix} \tag{2.65}$$

The characteristic equation is

$$\begin{aligned} |\lambda\mathbf{I} - \mathbf{A}| &= \begin{vmatrix} \lambda & -1 & 0 \\ 0 & \lambda & -1 \\ 6 & 11 & \lambda+6 \end{vmatrix} \\ &= \lambda^3 + 6\lambda^2 + 11\lambda + 6 \\ &= (\lambda+1)(\lambda+2)(\lambda+3) \end{aligned}$$

The eigenvalues of **A** are the roots of the characteristic equation, or -1, -2, and -3.

Invariance of Eigenvalues.

To prove the invariance of the eigenvalues under a linear transformation, we must show that the characteristic polynomials $|\lambda\mathbf{I} - A|$ and $|\lambda\mathbf{I} - \mathbf{P}^{-1}A\mathbf{P}|$ are identical. Since the determinant of a product is the product of the determinants, we obtain

$$\begin{aligned} |\lambda\mathbf{I} - \mathbf{P}^{-1}A\mathbf{P}| &= |\lambda\mathbf{I}\mathbf{P}^{-1}\mathbf{P} - \mathbf{P}^{-1}A\mathbf{P}| \\ &= |\mathbf{P}^{-1}(\lambda\mathbf{I} - \mathbf{A})\mathbf{P}| \\ &= |\mathbf{P}^{-1}||(\lambda\mathbf{I} - \mathbf{A})||\mathbf{P}| \\ &= |\mathbf{P}^{-1}||\mathbf{P}||(\lambda\mathbf{I} - \mathbf{A})| \end{aligned}$$

Noting that the product of the determinants $|\mathbf{P}^{-1}|$ and $|\mathbf{P}|$ is the determinant of the product $|\mathbf{P}^{-1}\mathbf{P}| = 1$, we obtain

$$|\lambda\mathbf{I} - \mathbf{P}^{-1}A\mathbf{P}| = |\lambda\mathbf{I} - A|$$

Thus, we have proved that the eigenvalues of **A** are invariant under a linear transformation.

Nonuniqueness of a Set of State Variables.

It has been stated that a set of state variables is not unique for a given system. Suppose that $x_1, x_2, \ldots, x_n$ are a set of state variables. Then we may take as another set of state variables any set of functions

$$\begin{aligned}
\hat{x}_1 &= X_1(x_1, x_2, \ldots, x_n)\\
\hat{x}_2 &= X_2(x_1, x_2, \ldots, x_n)\\
&\vdots\\
\hat{x}_n &= X_n(x_1, x_2, \ldots, x_n)
\end{aligned}$$

provided that, for every set of values $\hat{x}_1, \ldots, \hat{x}_n$ there corresponds a unique set of values $x_1, \ldots, x_n$ and vice versa. Thus, if $\mathbf{x}$ is a state vector, then $\hat{\mathbf{x}}$, where

$$\hat{\mathbf{x}} = \mathbf{P}\mathbf{x}$$

is also a state vector, provided the matrix $\mathbf{P}$ is nonsingular. Different state vectors convey the same information about the system behavior.

2.3.2 Solving the time-invariant State Equation

In this section,we shall obtain the general solution of the linear time-invariant state equation. We shall first consider the homogeneous case and then the nonhomogeneous case.

Solution of Homogeneous State Equations.

Before we solve vector-matrix differential equations, let us review the solution of the scalar differential equation

$$\dot{x} = ax \tag{2.66}$$

In solving this equation, we may assume a solution x(t) of the form

$$x(t) = b_0 + b_1 t + b_2 t^2 + \ldots + b_k t^k + \ldots \tag{2.67}$$

By substituting this assumed solution into Equation 2.66, we obtain

$$b_1 + 2b_2 t + 2b_2 t^2 + \ldots + kb_k t^{k-1} + \ldots = a(b_0 + b_1 t + b_2 t^2 + \ldots + b_k t^k + \ldots) \tag{2.68}$$

If the assumed solution is to be the true solution, Equation 2.68 must hold for any t. Hence, equating the coefficients of the equal powers of t , we obtain

$$\begin{aligned}
b_1 &= ab_0 \\
b_2 &= \frac{1}{2}ab_1 = \frac{1}{2}a^2b_0 \\
b_3 &= \frac{1}{3}ab_2 = \frac{1}{3\times 2}a^3b_0 \\
&\vdots \\
b_k &= \frac{1}{k!}a^kb_0
\end{aligned}$$

The value of b_0 is determined by substituting $t = 0$ into Equation 2.67, or

$$x(0) = b_0$$

Hence, the solution x(t) can be written as

$$\begin{aligned}
x(t) &= \left(1 + at + \frac{1}{2!}a^2t^2 + \ldots + \frac{1}{k!}a^kt^k + \ldots\right)x(0) \\
&= e^{at}x(0)
\end{aligned}$$

We shall now solve the vector-matrix differential equation

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} \tag{2.69}$$

where **x**=n-vector and $\mathbf{A} = n \times n$ matrix. By analogy with the scalar case, we assume that the solution is in the form of a vector power series in t , or

$$\mathbf{x}(t) = \mathbf{b}_0 + \mathbf{b}_1t + \mathbf{b}_2t^2 + \ldots + \mathbf{b}_kt^k + \ldots \tag{2.70}$$

By substituting this assumed solution into Equation 2.69, we obtain

$$\mathbf{b}_1+2\mathbf{b}_2t+3\mathbf{b}_2t^2+\ldots+k\mathbf{b}_kt^{k-1}+\ldots = \mathbf{A}(\mathbf{b}_0+\mathbf{b}_1t+\mathbf{b}_2t^2+\ldots+\mathbf{b}_kt^k+\ldots) \tag{2.71}$$

If the assumed solution is to be the true solution, Equation 2.71 must hold for all t. Thus, by equating the coefficients of like powers on both sides of Equation 2.71, we obtain

$$\begin{aligned}
\mathbf{b}_1 &= \mathbf{A}\mathbf{b}_0 \\
\mathbf{b}_2 &= \frac{1}{2}\mathbf{A}\mathbf{b}_1 = \frac{1}{2}\mathbf{A}^2\mathbf{b}_0 \\
\mathbf{b}_3 &= \frac{1}{3}\mathbf{A}\mathbf{b}_2 = \frac{1}{3\times 2}\mathbf{A}^3\mathbf{b}_0 \\
&\vdots \\
\mathbf{b}_k &= \frac{1}{k!}\mathbf{A}^k\mathbf{b}_0
\end{aligned}$$

The value of $\mathbf{b}_0$ is determined by substituting $t = 0$ into Equation 2.70, or

$$\mathbf{x}(0) = \mathbf{b}_0$$

Hence, the solution $\mathbf{x}(t)$ can be written as

$$\begin{aligned} \mathbf{x}(t) &= \left(\mathbf{I} + \mathbf{A}t + \frac{1}{2!}\mathbf{A}^2 t^2 + \ldots + \frac{1}{k!}\mathbf{A}^k t^k + \ldots \right) \mathbf{x}(0) \\ &= e^{\mathbf{A}t}\mathbf{x}(0) \end{aligned}$$

Since the matrix exponential is very important in the state-space analysis of linear systems, we shall next examine its properties.

Matrix Exponential.

It can be proved that the matrix exponential of an n× n matrix $\mathbf{A}$,

$$e^{\mathbf{A}t} = \sum_{k=0}^{\infty} \frac{\mathbf{A}^k t^k}{k!}$$

converges absolutely for all finite t. (Hence, computer calculations for evaluating the elements of $e^{\mathbf{A}t}$ by using the series expansion can be easily carried out.) Because of the convergence of the infinite series $\sum_{k=0}^{\infty} \frac{\mathbf{A}^k t^k}{k!}$, the series can be differentiated term by term to give

$$\frac{d}{dt} e^{\mathbf{A}t} = \mathbf{A} e^{\mathbf{A}t}$$

The matrix exponential has the property that

$$e^{\mathbf{A}(t+s)} = e^{\mathbf{A}t} e^{\mathbf{A}s}$$

Laplace Transform Approach to the Solution of Homogeneous State Equations.

Let us first consider the scalar case:

$$\dot{x} = ax \tag{2.72}$$

Taking the Laplace transform of Equation 2.71, we obtain

$$sX(s) - x(0) = aX(s) \tag{2.73}$$

where $X(s) = \mathcal{L}[x]$. Solving Equation 2.72 for X(s) gives

$$X(s) = (s - a)^{-1} x(0)$$

The inverse Laplace transform of this last equation gives the solution

$$x(t) = e^{at}x(0) \tag{2.74}$$

The foregoing approach to the solution of the homogeneous scalar differential equation can be extended to the homogeneous state equation:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) \tag{2.75}$$

Taking the Laplace transform of both sides of Equation 2.75, we obtain

$$s\mathbf{X}(s) - \mathbf{x}(0) = \mathbf{A}\mathbf{X}(s) \tag{2.76}$$

where $X(s) = \mathcal{L}[x]$. Hence,

$$(s\mathbf{I} - \mathbf{A})\mathbf{X}(s) = \mathbf{x}(0)$$

Premultiplying both sides of this last equation by $(s\mathbf{I} - \mathbf{A})^{-1}$, we obtain

$$\mathbf{X}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{x}(0)$$

The inverse Laplace transform of this last equation gives the solution x(t). Thus,

$$\mathbf{x}(t) = \mathcal{L}^{-1}\left[(s\mathbf{I} - \mathbf{A})^{-1}\right]x(0) \tag{2.77}$$

Note that

$$(s\mathbf{I} - \mathbf{A})^{-1} = \frac{\mathbf{I}}{s} + \frac{\mathbf{A}}{s^2} + \frac{\mathbf{A}^2}{s^3} + \ldots$$

Hence, the inverse Laplace transform of $(s\mathbf{I} - \mathbf{A})^{-1}$,gives

$$\mathcal{L}^{-1}\left[(s\mathbf{I} - \mathbf{A})^{-1}\right] = \mathbf{I} + \mathbf{A}t + \frac{\mathbf{A}^2t^2}{2!} + \frac{\mathbf{A}^3t^3}{3!} + \ldots = e^{\mathbf{A}t} \tag{2.78}$$

(The inverse Laplace transform of a matrix is the matrix consisting of the inverse Laplace transforms of all elements.) From Equations 2.77 and 2.78, the solution of Equation 2.75 is obtained as

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0)$$

The importance of Equation 2.78 lies in the fact that it provides a convenient means for finding the closed solution for the matrix exponential.

State-Transition Matrix.

We can write the solution of the homogeneous state equation

$$\dot{\mathbf{x}}(t = \mathbf{A}\mathbf{x}(t) \tag{2.79}$$

as

$$\mathbf{x}(t) = \mathbf{\Phi}(\mathbf{t})\mathbf{x}(0) \tag{2.80}$$

where $\mathbf{\Phi}(\mathbf{t})$ is an n X n matrix and is the unique solution of

$$\dot{\mathbf{\Phi}}(\mathbf{t}) = \mathbf{A}\mathbf{\Phi}(\mathbf{t})$$

Solution of Nonhomogeneous State Equations.

We shall begin by considering the scalar case

$$\dot{x} = ax + bu \tag{2.81}$$

Let us rewrite Equation 2.81 as

$$\dot{x} - ax = bu$$

Multiplying both sides of this equation by e^{-at}, we obtain

$$e^{-at}\left[\dot{x}(t) - ax(t)\right] = \frac{d}{dt}\left[e^{-at}x(t)\right] = e^{-at}bu(t)$$

Integrating this equation between 0 and t gives

$$e^{-at}x(t) - x(0) = \int_0^t e^{-a\tau}bu(\tau)d\tau$$

or

$$x(t) = e^{at}x(0) + e^{at}\int_0^t e^{-a\tau}bu(\tau)d\tau$$

The first term on the right-hand side is the response to the initial condition and the second term is the response to the input u(t).
Let us now consider the nonhomogeneous state equation described by

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{2.82}$$

where $\mathbf{x}$ =n-vector, $\mathbf{u}$ = r-vector and $\mathbf{A}$ = n X n constant matrix and $\mathbf{B}$ = n Xr constant matrix.
By writing Equation 2.82 as

$$\dot{\mathbf{x}}(t) - \mathbf{A}\mathbf{x}(t) = \mathbf{B}\mathbf{u}(t) \tag{2.83}$$

and premultiplying both sides of this equation by $e^{\mathbf{A}t}$, we obtain

$$e^{-\mathbf{A}t}\left[\dot{\mathbf{x}}(t)-\mathbf{A}\mathbf{x}(t)\right]=\frac{d}{dt}\left[e^{-\mathbf{A}t}\mathbf{x}(t)\right]=e^{-\mathbf{A}t}\mathbf{B}\mathbf{u}(t)$$

Integrating the preceding equation between 0 and t gives

$$e^{-\mathbf{A}t}\mathbf{x}(t)-\mathbf{x}(0)=\int_0^t e^{-\mathbf{A}\tau}\mathbf{B}\mathbf{u}(\tau)d\tau$$

or

$$\mathbf{x}(t)=e^{\mathbf{A}t}\mathbf{x}(0)+\int_0^t e^{-\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)d\tau \tag{2.84}$$

Equation 2.84 can also be written as

$$\mathbf{x}(t)=\mathbf{\Phi}(t)\mathbf{x}(0)+e^{\mathbf{A}t}\int_0^t \mathbf{\Phi}(t-\tau)\mathbf{B}\mathbf{u}(\tau)d\tau \tag{2.85}$$

where $\mathbf{\Phi}(t)=e^{\mathbf{A}t}$. Equation 2.84 or 2.85 is the solution of Equation 2.83.The solution $\mathbf{x}$(t) is clearly the sum of a term consisting of the transition of the initial state and a term arising from the input vector.

Laplace Transform Approach to the Solution of Nonhomogeneous State Equations.

The solution of the nonhomogeneous state equation

$$\dot{\mathbf{x}}=\mathbf{A}\mathbf{x}+\mathbf{B}\mathbf{u} \tag{2.86}$$

can also be obtained by the Laplace transform approach. The Laplace transform of this last equation yields

$$s\mathbf{X}(s)-\mathbf{x}(0)=\mathbf{A}\mathbf{X}(s)+\mathbf{B}\mathbf{U}(s)$$

Premultiplying both sides of this last equation by $(s\mathbf{I}-\mathbf{A})^{-1}$, we obtain

$$\mathbf{X}(s)=(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{x}(0)+(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s)$$

The inverse Laplace transform of this last equation can be obtained by use of the convolution integral as follows:

$$\mathbf{x}(t)=e^{\mathbf{A}t}\mathbf{x}(0)+\int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{U}(\tau)d\tau$$

Solution in Terms of $x(t_0)$. Thus far we have assumed the initial time to be zero. If, however, the initial time is given by t_0 instead of 0, then the solution to Equation 2.86 must be modified to

$$\mathbf{x}(t)=e^{\mathbf{A}(t-t_0)}\mathbf{x}(t_0)+\int_{t_0}^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{U}(\tau)d\tau$$

Example 2.3.1. *Obtain the time response of the following system:*

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{2.87}$$

where u(t)=1(t) is the unit-step function occurring at $t = 0$. *The state-transition matrix* $\mathbf{\Phi}(t) = e^{\mathbf{A}t}$ *is*

$$\mathbf{\Phi}(t) = \mathcal{L}^{-1}[(s\mathbf{I} - \mathbf{A})^{-1}]$$

Since

$$s\mathbf{I} - \mathbf{A} = \begin{bmatrix} s & -1 \\ 2 & s+3 \end{bmatrix}$$

the inverse of $(s\mathbf{I} - \mathbf{A})$ *is given by*

$$\begin{aligned} (s\mathbf{I} - \mathbf{A})^{-1} &= \frac{adj(s\mathbf{I} - \mathbf{A})}{|s\mathbf{I} - \mathbf{A})|} = \frac{1}{(s+1)(s+2)} \begin{bmatrix} s+3 & 1 \\ -2 & s \end{bmatrix} \\ &= \begin{bmatrix} \frac{s+3}{(s+1)(s+2)} & \frac{1}{(s+1)(s+2)} \\ \frac{-2}{(s+1)(s+2)} & \frac{1}{(s+1)(s+2)} \end{bmatrix} \end{aligned}$$

Hence

$$\mathbf{\Phi}(t) = e^{\mathbf{A}t} = \mathcal{L}^{-1}[(s\mathbf{I} - \mathbf{A})^{-1}] = \begin{bmatrix} 2e^{-t} - e^{2t} & e^{-t} - e^{-2t} \\ -2e^{-t} + 2e^{-2t} & -e^{-t} + 2e^{-2t} \end{bmatrix} \tag{2.88}$$

The response to the unit-step input is then obtained as

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t \begin{bmatrix} 2e^{-(t-\tau)} - e^{2(t-\tau)} & e^{-(t-\tau)} - e^{-2(t-\tau)} \\ -2e^{-(t-\tau)} + 2e^{-2(t-\tau)} & -e^{-(t-\tau)} + 2e^{-2(t-\tau)} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} 1(t) d\tau$$

or

$$\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} 2e^{-t} - e^{2t} & e^{-t} - e^{-2t} \\ -2e^{-t} + 2e^{-2t} & -e^{-t} + 2e^{-2t} \end{bmatrix} \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} + \begin{bmatrix} \frac{1}{2} - e^{-t} + \frac{1}{2}e^{-2t} \\ e^{-t} - e^{-2t} \end{bmatrix} \tag{2.89}$$

2.3.3 Controllability.

A system is said to be controllable at time t_0 if it is possible by means of an unconstrained control vector to transfer the system from any initial state $x(t_0)$ to any other state in a finite interval of time.

A system is said to be observable at time t_0 if, with the system in state $x(t_0)$, is possible it to determine this state from the observation of the output over a finite time interval. The concepts of controllability and observability were introduced by Kalman. They play an important role in the design of control systems in state space. In fact, the conditions of controllability and observability may govern the existence of a complete solution to the control system design problem. The solution to this problem may not exist if the system considered is not controllable. Although most physical systems are controllable and observable, corresponding mathematical models may not possess the property of controllability and observability. Then it is necessary to know the conditions under which a system is controllable and observable. This section deals with controllability and the next section discusses observability. In what follows, we shall first derive the condition for complete state controllability.

Complete State Controllability of Continuous-Time Systems. Consider the continuous-time system:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \tag{2.90}$$

where $\mathbf{x}$ = state vector (n-vector), u = control signal (scalar), $\mathbf{A}$ = n X n matrix, $\mathbf{B}$ = n X 1 matrix. The system is said to be state controllable at $t = t_0$ if it is possible to construct an unconstrained control signal that will transfer an initial state to any final state in a finite time interval $t_0 \leq t \leq t_1$ If every state is controllable, then the system is said to be completely state controllable. It can be proved that the condition for complete state controllability is that the n X n matrix

$$\left[\mathbf{B} \ \vdots \ \mathbf{AB} \ \vdots \ \ldots \ \vdots \ \mathbf{A}^{n-1}\mathbf{B}\right] \tag{2.91}$$

be of rank n, or contain n linearly independent column vectors.

2.3.4 Observability.

In this section we discuss the observability of linear systems. Consider the unforced system described by the following equations:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} \qquad (2.92) \\ \mathbf{y} &= \mathbf{C}\mathbf{x} \qquad (2.93) \end{aligned}$$

The system is said to be completely observable if every state $\mathbf{x}(t_0)$ can be determined from the observation of $\mathbf{y}(t)$ over a finite time interval, $t_0 \leq t \leq t_1$ .The system is, therefore, completely observable if every transition of the state eventually affects every element of the output vector.The concept of observability is useful in solving the problem of reconstructing unmeasurable state variables from measurable variables in the minimum possible length of time. In this section we treat only linear, time-invariant systems. Therefore, without loss of generality, we can assume that $t_0 = 0$. The concept of observability is very important because,

Matlab/Octave code	Python code

A = [0 1 0; 0 0 1 ; -5 -25 -5];
B = [0; 25; -120]
C = [1 0 0];
D = [0];
sys =ss(A,B,C,D);
co= ctrb(sys);
if rank(co)==size(A,1)
Controllability=1
else Controllability =0
endif
	from control.matlab import *
	from numpy import size
	from numpy.linalg import matrix_rank
	
	A = [[0,1,0],[0,0,1],[-5,-25,-5]]
	B = [[0], [25], [-120]]
	C = [1,0,0]
	D = [0]
	co=ctrb(A,B)
	Controllability=0
	if matrix_rank(co)==size(A,axis=1):
	  Controllability=1
	print('Controllability=',
	Controllability)

Figure 2.37: Matlab/Octave/Python program 2.37.

in practice, the difficulty encountered with state feedback control is that some of the state variables are not accessible for direct measurement, with the result that it becomes necessary to estimate the unmeasurable state variables in order to construct the control signals.

The system described by Equations 2.92 and 2.93 is completely observable if and only if the n X nxm matrix

$$\left[\mathbf{C} \ \vdots \ \mathbf{CA} \ \vdots \ \ldots \ \vdots \ \mathbf{CA}^{n-1}\right] \tag{2.94}$$

is of rank n or has n linearly independent column vectors. This matrix is called the observability matrix.

2.3.5 Pole Placement

In this section we shall present a design method commonly called the pole placement or pole-assignment technique. We assume that all state variables are measurable and are available for feedback. It will be shown that if the system considered is completely state controllable, then poles of the closed-loop system may be placed at any desired locations by means of state feedback through an appropriate state feedback gain matrix.

The present design technique begins with a determination of the desired closed-loop poles based on the transient-response and/or frequency-response requirements, such as speed, damping ratio, or bandwidth, as well as steady-state requirements. Let us assume that we decide that the desired closed-loop poles are to be at $s = \mu_1, s = \mu_2, ..., s = \mu_n$. By choosing an appropriate gain matrix for state feedback, it is possible to force the system to have closed-loop poles at the desired locations, provided that the original system is completely state controllable.

In this chapter we limit our discussions to single-input-single-output systems. That is, we assume the control signal u (t) and output signal y(t) to be scalars.

In what follows we shall prove that a necessary and sufficient condition that the closed-loop poles can be placed at any arbitrary locations in the s plane is that the system be completely state controllable. Then we shall discuss methods for determining the required state feedback gain matrix.

Design by Pole Placement.

In the conventional approach to the design of a single-input-single-output control system, we design a controller (compensator) such that the dominant closed-loop poles have a desired damping ratio ζ and an undamped natural frequency ω_n. In this approach, the order of the system may be raised by 1 or 2 unless pole-zero cancellation takes place. Note that in this approach we assume the effects on the responses of non-dominant closed-loop poles to be negligible.

Different from specifying only dominant closed-loop poles (the conventional design approach), the present pole-placement approach specifies all closed-loop poles. (There is a cost associated with placing all closed-loop poles, however, because placing all closed-loop poles requires successful measurements of all state variables or else requires the inclusion of a state observer in the system.) There is also a requirement on the part of the system for the closed-loop poles to be placed at arbitrarily chosen locations. The requirement is that the system be completely state controllable.

Consider a control system

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}u \\ y &= \mathbf{C}\mathbf{x} + Du \end{aligned} \tag{2.95}$$

where $\mathbf{x}$ = state vector (n-vector), y= output signal (scalar), u = control signal (scalar), $\mathbf{A}$ = n X n constant matrix, $\mathbf{B}$ =n X 1 constant matrix, $\mathbf{C}$ = 1 X n constant matrix and D = constant (scalar).

We shall choose the control signal to be

$$u = -\mathbf{K}\mathbf{x} \tag{2.96}$$

This means that the control signal u is determined by an instantaneous state. Such a scheme is called state feedback. The 1 X n matrix **K** is called the state feedback gain matrix. We assume that all state variables are available for feedback. In the following analysis we assume that u is unconstrained. A block diagram for this system is shown in 2.38.

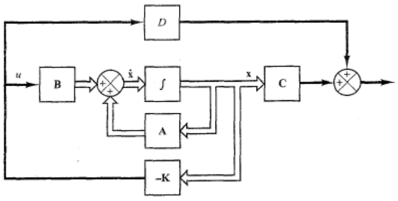

Figure 2.38: Design of Control Systems in State Space.

This closed-loop system has no input. Its objective is to maintain the zero output. Because of the disturbances that may be present, the output will deviate from zero. The nonzero output will be returned to the zero reference input because of the state feedback scheme of the system. Such a system where the reference input is always zero is called a regulator system. (Note that if the reference input to the system is always a nonzero constant, the system is also called a regulator system.) Substituting Equation 2.95 into Equation 2.96 gives

$$\dot{\mathbf{x}}(t) = (\mathbf{A} - \mathbf{BK})\mathbf{x}(t)$$

The solution of this equation is given by

$$\mathbf{x}(t) = e^{(\mathbf{A}-\mathbf{BK})t}\mathbf{x}(0) \tag{2.97}$$

where x(0) is the initial state caused by external disturbances.The stability and transient-response characteristics are determined by the eigenvalues of matrix $\mathbf{A} - \mathbf{BK}$. If matrix K is chosen properly, the matrix $\mathbf{A} - \mathbf{BK}$ can be made an asymptotically stable matrix, and for all $\mathbf{x}(0) \neq 0$, it is possible to make $\mathbf{x}(t)$ approach 0 as t approaches infinity. The eigenvalues of matrix $\mathbf{A} - \mathbf{BK}$ are called the regulator poles. If these regulator poles are placed in the left-half s plane, then $\mathbf{x}(t)$ approaches 0 as t approaches infinity. The problem of placing the regulator poles (closed-loop poles) at the desired location is called a pole-placement problem.

Determination of Matrix K Using Direct Substitution Method.

If the system is of low order ($n \leq 3$), direct substitution of matrix **K** into the desired characteristic polynomial may be simpler. For example, if $n = 3$, then write the state feedback gain matrix **K** as

$$\mathbf{K} = [k_1 \ \ k_2 \ \ k_3]$$

Substitute this $\mathbf{K}$ matrix into the desired characteristic polynomial $|s\mathbf{I}-\mathbf{A}+\mathbf{BK}|$ and equate it to $(s-\mu_1,)(s-\mu_2)(s-\mu_3)$,or

$$|s\mathbf{I}-\mathbf{A}+\mathbf{BK}| = (s-\mu_1,)(s-\mu_2)(s-\mu_3) \tag{2.98}$$

Since both sides of this characteristic equation are polynomials in s, by equating the coefficients of the like powers of s on both sides, it is possible to determine the values of k_1, k_2, k_3. This approach is convenient if n = 2 or 3. (For n = 4, 5 , 6, ... , this approach may become very tedious.)

Note that if the system is not completely controllable, matrix $\mathbf{K}$ cannot be determined. (No solution exists.)

Choosing the Locations of Desired Closed-Loop Poles.

The first step in the pole-placement design approach is to choose the locations of the desired closed-loop poles. The most frequently used approach is to choose such poles based on experience in the root-locus design, placing a dominant pair of closed-loop poles and choosing other poles so that they are far to the left of the dominant closed-loop poles.

Note that if we place the dominant closed-loop poles far from the $j\omega$ axis, so that the system response becomes very fast, the signals in the system become very large, with the result that the system may become nonlinear. This should be avoided.

Another approach is based on the quadratic optimal control approach. This approach will determine the desired closed-loop poles such that it balances between the acceptable response and the amount of control energy required. Note that requiring a high-speed response implies requiring large amounts of control energy. Also, in general, increasing the speed of response requires a larger, heavier actuator, which will cost more.

Example 2.3.2. *Consider the regulator system given by*

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{B}u \tag{2.99}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -5 & -6 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The system uses the state feedback control $u = -\mathbf{Kx}$. Let us choose the desired closed-loop poles at

$$s = -2 + j4, \quad s = -2 - j4, \quad s = -10$$

Determine the state feedback gain matrix $\mathbf{K}$*. First, we need to check the controllability matrix of the system. Since the controllability matrix* $\mathbf{M}$ *is given by*

$$\mathbf{M} = \left[\mathbf{B} \ \vdots \ \mathbf{AB} \ \vdots \ \mathbf{A}^2\mathbf{B}\right] = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & -6 \\ 1 & -6 & 31 \end{bmatrix} \tag{2.100}$$

we find that $|\mathbf{M}| = -1$*, and therefore, rank* $M = 3$*. Thus, the system is completely state controllable and arbitrary pole placement is possible. By defining the desired state feedback gain matrix* $\mathbf{K}$ *as*

$$\mathbf{K} = [k_1 \quad k_2 \quad k_3]$$

and equating $|s\mathbf{I} - \mathbf{A} + \mathbf{BK}|$ *with the desired characteristic equation, we obtain*

$$\begin{aligned} |s\mathbf{I} - \mathbf{A} + \mathbf{BK}| &= \begin{vmatrix} s & -1 & 0 \\ 0 & s & -1 \\ 1+k_1 & 5+k_2 & s+6+k_3 \end{vmatrix} \\ &= s(s(6+s+k_3)+(5+k_2))+1+k_1 \\ &= s^3+(6+k_3)s^2+(5+k_2)s+1+k_1 \\ &= (s+2-j4)(s+2+j4)(s+10) = s^3+15s^2+60s+200 \end{aligned}$$

from which we obtain

$$k_1 = 199, \quad k_2 = 55, \quad k_3 = 8$$

2.3.6 Pole Placement with software

Pole-placement problems can be solved easily with software. The software has two commands (acker and place) for the computation of feedback-gain matrix $\mathbf{K}$. The command acker is based on Ackermann's formula. This command applies to single-input systems only. The desired closed-loop poles can include multiple poles (poles located at the same place).

If the system involves multiple inputs, for a specified set of closed-loop poles the state-feedback gain matrix $\mathbf{K}$ is not unique and we have an additional freedom (or freedoms) to choose $\mathbf{K}$. There are many approaches to constructively utilize this additional freedom (or freedoms) to determine $\mathbf{K}$. One common use is to maximize the stability margin. The pole placement based on this approach is called the robust pole placement. The software command for the robust pole placement is place. Although the command place can be used for both single-input and multiple-input systems, this command requires that the multiplicity of poles in the desired closed-loop poles be no greater than the rank of B. That is, if matrix B is an n X 1 matrix, the command place requires that there be no multiple poles in the set of desired closed-loop poles.

For single-input systems, the commands acker and place yield the same **K**. (But for multiple-input systems, one must use the command place instead of acker.)

It is noted that when the single-input system is barely controllable, some computational problem may occur if the command acker is used. In such a case the use of the place command is preferred, provided that no multiple poles are involved in the desired set of closed-loop poles.

To use the command acker or place, we first enter values for the matrices: $\mathbf{A}, \mathbf{B}, \mathbf{J}$. Where **J** matrix is the matrix consisting of the desired closed-loop poles such that

$$\mathbf{J} = [\mu_1 \;\; \mu_2 \;\; \ldots \;\; \mu_2]$$

Then we enter

```
K = acker(A,B,J)
```

or

```
K = place(A,B,J)
```

It is noted that the command $eig(\mathbf{A} - \mathbf{B} * \mathbf{K})$ may be used to verify that K obtained gives the desired eigenvalues.

Example 2.3.3. *Consider the same system as treated in Example 2.3.2. The system equation is*

$$\dot{\mathbf{x}} \;=\; \mathbf{A}\mathbf{x} + \mathbf{B}u \tag{2.101}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -5 & -6 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

By using the state feedback control $u = -\mathbf{K}\mathbf{x}$. It is desired to have closed-loop poles at

$$s = -2 + j4, \quad s = -2 - j4, \quad s = -10$$

*Determine the state feedback-gain matrix **K** with software.*

*Software programs that generate matrix **K** are shown in Programs 2.39 and 2.40. Software Program 2.39 uses command acker and software Program 2.40 uses command place.*

Matlab/Octave code	Python code

A = [0 1 0;0 0 1;-1 -5 -6];
B = [0;0;1];
J = [-2+j*4 -2-j*4 -10];
K = acker(A,B,J)
K =
 	
199   55   8
	No acker command in python

Figure 2.39: Matlab/Octave/Python program 2.39.

Matlab/Octave code	Python code

A = [0 1 0;0 0 1;-1 -5 -6];
B = [0;0;1];
J = [-2+j*4 -2-j*4 -10];
K = place(A,B,J)
K =
 	
199   55   8
	from matplotlib.pyplot import *
	from control.matlab import *
	
	A = [[0,1,0],[0,0,1],[-1,-5,-6]];
	B = [[0],[0],[1]];
	J = [-2+4j,-2-4j,-10];
	K = place(A,B,J)
	print(K)

Figure 2.40: Matlab/Octave/Python program 2.40.

Example 2.3.4. *Consider the same system as discussed in Example 2.3.3. It is desired that this regulator system have closed-loop poles at*

$$s = -2 + j4, \quad s = -2 - 4j, \quad s = -10$$

The necessary state feedback gain matrix **K** *was obtained in Example 2.3.3 as follows:*

$$\mathbf{K} = [199 \quad 55 \quad 8]$$

Using software, obtain the response of the system to the following initial condition:

$$\mathbf{x}(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.102}$$

***Response to Initial Condition**: To obtain the response to the given initial condition* $\mathbf{x}(0)$*, we substitute* $u = -\mathbf{Kx}$ *into the plant equation to get*

$$\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{BK})\mathbf{x}, \quad \mathbf{x}(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

To plot the response curves (x_1 *versus t ,* x_2 *versus t , and* x_3 *versus t), we may use the command initial. We first define the state-space equations for the system as follows:*

$$\begin{aligned} \dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{BK})\,\mathbf{x} - \mathbf{IU} \\ \mathbf{y} &= \mathbf{Ix} + \mathbf{I}u \end{aligned} \tag{2.103}$$

where we included **U** *(a three-dimensional input vector). This* **U** *vector is considered 0 in the computation of the response to the initial condition. Then we define*

```
sys= ss(A - BK, eye(3), eye(3), eye(3))
```

and use the initial command as follows:

```
x=initial(sys,[1;0;0],t)
```

where t is the time duration we want to use, such as

```
t=0;0.01:4;
```

Then obtain x l , x2, and x3 as follows:

```
x1=[ 1 0 0]*x';
x2=[ 0 1 0]*x';
x3=[ 0 0 1]*x';
```

and use the plot command. This program is shown in software Program 2.42. The resulting response curves are shown in Figure 2.41.

2.3.7 Design of servo systems

Work in progess ...

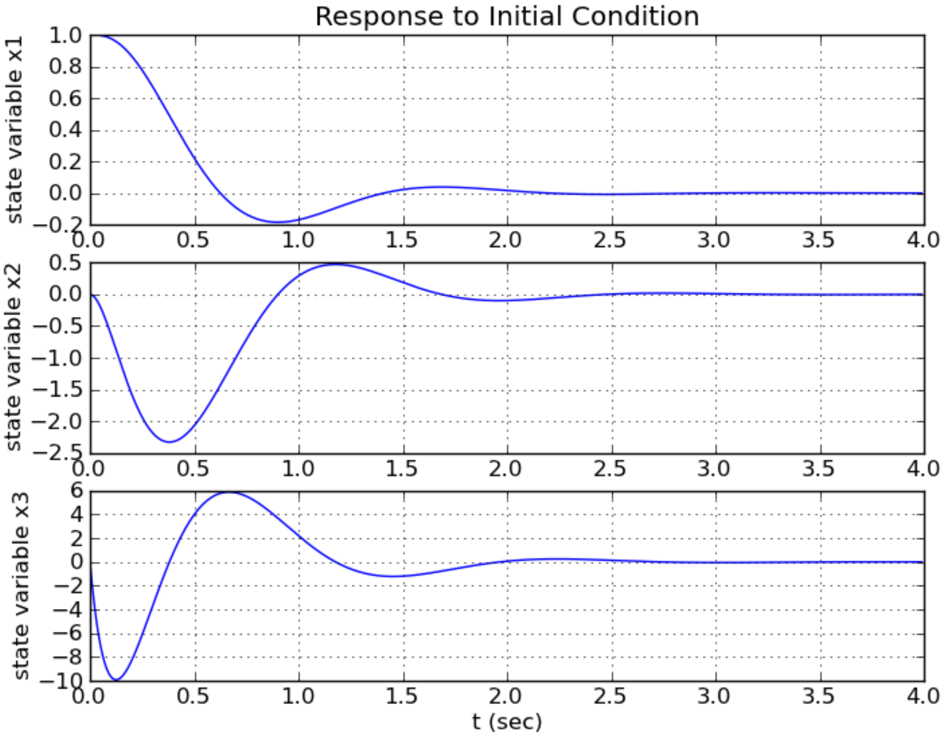

Figure 2.41: Response to initial condition.

2.3.8 State observer

Work in progess ...

2.3.9 Design of control systems with observer

Work in progess ...

Matlab/Octave code	Python code

% Response to initial condition:
A = [0 1 0;0 0 1;-1 -5 -6];
B = [0;0;1];
K = [1 99 55 8];
sys = ss(A-B*K, eye(31, eye(31, eye(3));
t = 0:0.01:4;
x = initial(sys,[1 ;0;0],t);
x1 = [1 0 0]*x’;
x2 = [0 1 0]*x’;
x3 = [0 0 1]*x’;
subplot(3,1,1); plot(t,x1), grid
title(’Response to Initial Condition’)
ylabel(’state variable x1’)
subplot(3,1,2); plot(t,x2),grid
ylabel(’state variable x2’)
subplot(3,1,3); plot(t,x3),grid
xlabel(’t (sec)’)
ylabel(’state variable x3’)	

Figure 2.42: Matlab/Octave/Python program 2.42.

Bibliography

[1] *Control Systems: Principles and Design.* Tata McGraw-Hill Publishing Company, 2002.

[2] *Digital Control And State Variable Method.* McGraw-Hill Education (India) Pvt Limited, 2008.

[3] K. J. Astrom and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers.* Princeton University Press, illustrated edition edition, Apr. 2008.

[4] K. Ogata. *System Dynamics.* Prentice Hall, 1998.