Tom Moir

Feed-back

Springer

Feedback


Boston Dynamics

Tom Moir

Feedback

Springer

Tom Moir
Department of Electrical and Electronic
Engineering, School of Engineering,
Computer and Mathematical Sciences
Auckland University of Technology
Auckland, New Zealand

ISBN 978-3-030-34838-0 ISBN 978-3-030-34839-7 (eBook)
https://doi.org/10.1007/978-3-030-34839-7

Front Page picture credit. Atlas robot Image, courtesy of Boston Dynamics.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To James and Callum.

Preface

This book is an accumulation of nearly 40 years of theoretical and practical experience. It is intended mainly for Electrical and Electronic Engineers but would also be suitable for Mechatronics Engineers and some aspects would be of interest to Mechanical Engineers. A first-level university course in mathematics and circuit theory is assumed as prerequisites. This book cuts through the theory and gets to the important design aspects which every engineer should know. Many of the fundamental concepts in feedback theory are still in use, but with the advent of modern software packages many have become obsolete. The book therefore concentrates on the important and pays respect to what once was but is no longer necessary. A historic background of the theory is made in Chap. 1 and some basic mathematical concepts are introduced. Chapters 2 and 3 are like many other texts on the subject since they are required theory. Chapters 4 and 5 look at the theory and practical aspects of speed and position control using mainly frequency-domain techniques which have been tried and tested in industry in the past 50 years. The point is made that by far the best tool available for the control-system designer is the Bode-plot. Although it is one of the oldest methods, the asymptotic approach makes it relatively simple to create controller designs and physically realise hardware controllers. A point missing in many other textbooks in this area is structural-resonance and how it restricts the closing bandwidth of the loop. The concept of the "ideal" Bode-plot is introduced for the first time and how to shape the open-loop system frequency-response to that form. Chapters 7 and 8 begin what has become known as "Modern control-theory" using state-space analysis.

Chapter 9 moves on to the modern world and introduces digital control-systems and their advantages. Chapter 10 moves to the design realisation of digital controllers as used in real-time embedded systems. Chapter 11 examines the state-space approach using discrete-time representation of LTI systems. Chapter 12 studies the effect of noise in LTI systems and includes an introduction to the optimal Wiener filter. Chapter 13 includes the Kalman filter as both an optimal state-estimator and a parameter-estimator. Deterministic recursive-least-squares (RLS) is also discussed.

Chapter 14 includes much information missed in most textbooks. It is a practical guide to implementing digital servos with real results as captured on a LabView-based system with a student myRio as embedded processor. This includes the embedded programming of a Kalman filter and all design equations.

Chapters 15 and 16 are an introduction to nonlinear control-systems. Unlike most control books, this chapter covers the Phase-Locked Loop (PLL), which is usually covered in communication textbooks but omitted for some reason in the control literature. The dual of the PLL known as the Amplitude-Locked Loop (ALL) is also discussed. Chapter 16 includes some research topics that have not been seen before, and these include how feedback has been used in mathematical algorithms for hundreds of years without the mathematicians who developed the algorithms seeing them in this way.

Chapter 17 is a brief introduction to optimal control. A chapter cannot do justice to such a topic that has had many books written on it for the past 50 years or more. It is meant more of a beginners introduction and any reader should be in a good position to research with more complicated problems outwith the scope of this book.

Perhaps the most striking omission however is that the Nyquist plot is not covered, and neither is the Nichols plot. Both were "bread and butter" to nearly all undergraduate courses in control. However, although they are of some theoretical interest, they are of little use to the design-engineer. Very few if all people can, or wish to design using a Nyquist-diagram. It has always been the domain of academics and not the practicing engineer. The same goes for root-locus, but the beauty of the method and the insight it gains means that a section is reserved for it. No design is covered using it, but it can give great insight to a student learning linear-systems with feedback. The Routh method of stability analysis has been superseded by fast computer algorithms, but it is given a mention in the book. Much emphasis is put on the design aspect of the control-system. This is the key element for a practicing professional engineer. An engineer must have the ability to visualise and rationalise scientific principles and turn equations based on these principles into hardware and software products.

Many videos that demonstrate the design and test approach to digital-control are available on the authors YouTube channel: https://www.youtube.com/c/TJMoir

The book is organised in tutorial fashion. Example by example builds up from elementary topics to more advanced concepts nearer the end. This is the style that the author has used over the years to teach the subject. The analogue systems and feedback methods are taught well before the digital is attempted. This is intentional and designed to give the student a firm bedrock foundation in analogue before even attempting the digital concepts. This contrasts with many other books which teach analogue and digital simultaneously. The author believes there is little merit to this approach since the real world is analogue and to design in the digital world requires a foundation in analogue first. This is known as deep learning and can take longer than learning the two side by side, but knowledge retention and understanding is better.

Auckland, New Zealand/Natick, USA/Austin, USA/Champaign, USA — Tom Moir

Acknowledgements

I am indebted to many people over the span of my teaching and times in industry. Three people have influenced me greatly. First, Prof. Mike Grimble my Ph.D. supervisor who first taught me many of the basics of stochastic control and inspired me to follow a research career. Then an old colleague Dr. John Barret, whose elegance, knowledge and power over mathematics seems without limit. He taught me simplicity, correct notation and inspired me even more. Finally, with two academics being mentors, I was brought down to earth by a real control-systems engineer whom I worked with for around 30 years in his company, on and off. That man was design-engineer and inventor Archie Pettigrew and the company was Ampsys electronics Ltd. of Paisley, Scotland. Archie had to re-teach me many aspects that get confused in the textbooks, and his wonderful hands-on philosophy of an engineer being at the bench solving real problems stays with me.

I am indebted to the continual support of National Instruments, the makers of LabView software and hardware which I have used over the past 20 years since coming to New Zealand.

I would like to thank many of my past Ph.D. students who have also inspired me whilst passing through the education system and making their own contributions to knowledge:

Dr. T. G. Vishwanath, Dr. A. J. Anderson, Dr. H. S. Dabis, Dr. C. P. Hung, Dr. O. Dussarratt, Dr. G. Jong, Dr. J. Chen, Dr. H. Agaiby, Dr. J. Jeong, Dr. Z. Qi, Dr. V. Sanjee, Dr. J. Harris, Dr. R. Sharan, Dr. S. Chehrehsa.

Some others of notable mention are:

Joe Kinsler, David Taylor, David Nutt, Ross Twiname, John Sim, Alen Alexanderian, Stephen Pollock and Mark Bekerleg.

Auckland, New Zealand — Assoc. Prof. Tom Moir

Contents

Chapter 1
Introduction to Feedback Control

Before delving into the mathematics of control-systems, it is perhaps best to first look at some examples from history and progress to the digital age. The concept of feedback control is quite an easy one to grasp if viewed from the point of view of the human body and how we perform various actions. For example, catching a ball in flight is perhaps quiet a difficult problem for us to reproduce with a machine but easy for a human. A simpler example is driving a car as seen in Fig. 1.1.

The car itself can be modelled as having some kind of dynamic equations based on Newton's laws of motion. This of course is oblivious to the person steering since it is purely practice or learning which the steering algorithm is based on in the case of a human driver. The driver selects continuously an angle θ_{in}, which gets transferred to the angle of the steering wheels and the motion of the car and gives a certain angle θ_{out}. Sensors are required to measure this angle in the form of the human eyes or even sometimes hearing for predictive control of things which may happen a short time ahead. Adjustments to the steering wheel angle are then made based on observations by the driver. This is a closed-loop control system and is the foundation of everything we study in this book. We can think that the driver creates an error between the desired angle of the wheels and the actual angle and corrects. This is of course not done consciously. Without any form of sensor to measure the position of the car the car would drift off its path and crash. The control system itself or controller is performed by actions hidden in the human brain and based on learning. Continual adjustments are made as the car moves about the road until it reaches its destination. Of course there are other inputs to this control system such as brake and throttle and in some cases a manual gearbox selection.

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_1

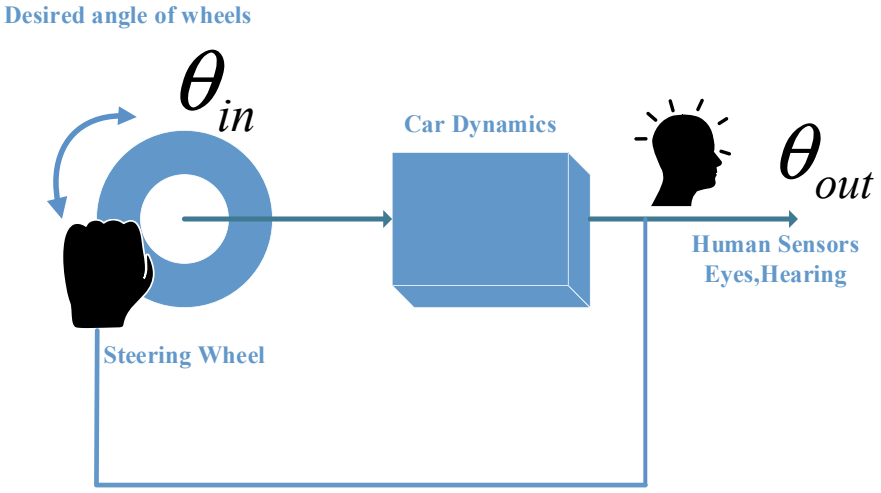


Fig. 1.1 Human driving a car

1.1 Historical Notes on Automatic-Control

Historically, it is recognised that one of the first applications of control-systems to machines was by James Watt, the Scottish inventor who greatly improved the steam-engine. It turns out that his famous governor for controlling the speed of the steam-engine (Fig. 1.2) was originally an invention used by Christian Huygens [1] and used to regulate the distance and pressure between millstones. James Watt in a later invention for the steam-engine used such a rotating centrifugal governor successfully. He is honoured recently by appearing on a 50 pounds sterling Bank of England banknote with his fellow business partner Matthew Boulton. It worked because the metal balls were thrown outwards as the speed increased and inwards when speed reduced. A linkage was attached to the sliding mechanism which opened or closed a throttle valve for steam intake. Damping was produced naturally because the slide of the sleeve on the shaft was not precise or by any means ideal. As machining became more precise with time it was found that instead of regulating the speed, the governor would give rise to oscillating speed (what was then called "hunting"). The exact mathematical reason was not understood until many years later when a mathematical theory of control-systems was developed. The development of the steam-engine is historically important as it led to the birth of the industrial revolution. Just as important however was James Watts fellow countryman James Clerk Maxwell who later in 1868 wrote a paper in an attempt to explain the operation of the governor [2].

Although Maxwell's paper is of great historic and scientific importance, it cannot be said that it in any way would be used by today's engineers, though we can see in this work that the beginnings of control-theory are developed including

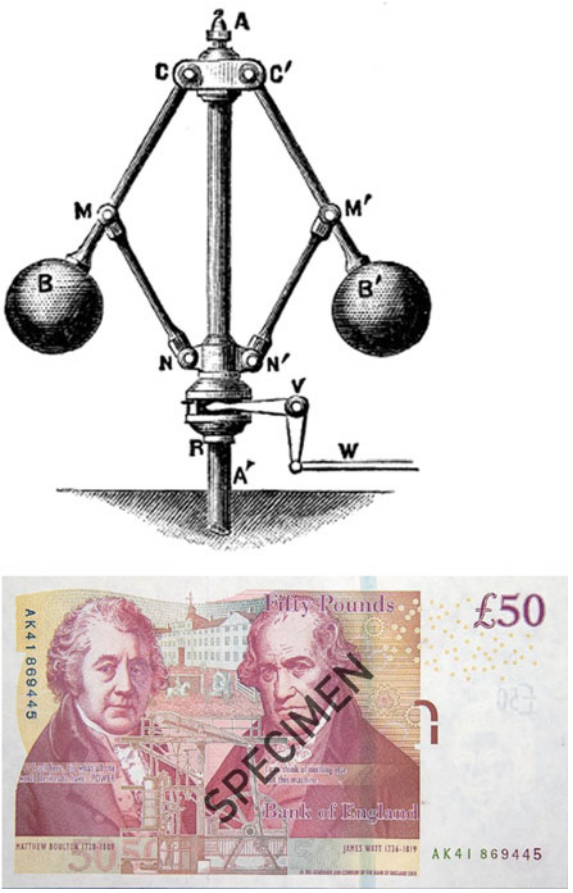


Fig. 1.2 Top: James Watt Steam-engine governor circa 1788 (Project Gutenberg [3]). Bottom: Bank of England 50 pound note features M. Boulton and J. Watt (rightmost)

what resembles today's closed-loop characteristic equation which Maxwell took to be third order. Maxwell's work was before the use of the transfer-function or Laplace transform which today's engineers see as standard, instead of working with classical differential equations. Of course the coefficients of this differential equation are of prime importance and Maxwell realised this but was unable to find conditions for stability. This was later left to the mathematician E Routh [4]. The so-called Routh table was a compulsory aspect of every course in control engineering until recently. The main reason being that the characteristic roots of a polynomial are nowadays easily found using numerical methods on software such as MATLAB™ and many other mathematical software packages.

In the early 20th century, most of the breakthroughs in control-engineering were made by electrical engineers or mathematicians. For example the Bell Telephone Laboratories had many fine engineering mathematicians who developed significant theoretical methods for explaining the stability or instability of amplifiers. Harold Black was an electrical engineer who was working on amplifiers for amplification

of voice. Until his work, amplifiers (then designed using Vacuum Tubes) were troubled by distortion and noise. This is because amplifiers, despite the best efforts of electronic engineers were inherently nonlinear. Of particular interest was the problem of transcontinental transmission using telephony. Black solved the problem in a flash of inspiration one day. He envisioned something which had never been attempted before in an electronic device, putting feedback from output to input around it. With some simple mathematics he could see that he may be able to make significant improvements [5]. The basic idea by Black is shown in Fig. 1.3 with a slight modification, in that a disturbance has been introduced. In this application the disturbance can be thought of as either distortion or noise. The input to the amplifier is V_{in} and the output V_o. The amplifier has a gain A and a proportion of the output signal is fed back to the input. This proportion is $\beta \leq 1$. This is a greatly simplified model in that it is not frequency dependent but will do as a simple introduction. To proceed further we note that there are two inputs and therefore we get around this by using the principle of superposition. We set each input to zero in turn and find the output in terms of each input. Then we find the overall output by adding the two outputs. This is only possible for linear systems which is an assumption we must at least make initially.

When d = 0

$$e = V_{in} - \beta V_o \tag{1.1}$$

$$V_o = Ae \tag{1.2}$$

From which we can write

$$V_o = \frac{A}{1 + A\beta} V_{in} \tag{1.3}$$

When $V_{in} = 0$ we can redraw Fig. 1.3. to give Fig. 1.4. Then we have

$$V_o = d - A\beta V_o \tag{1.4}$$

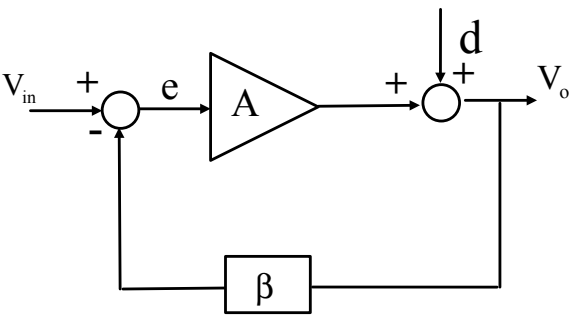


Fig. 1.3 Negative feedback around an amplifier with disturbance term d

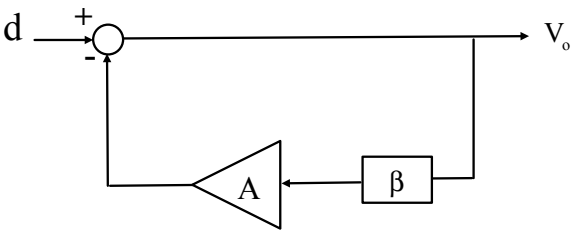


Fig. 1.4 Re-arrange block diagram with disturbance as input

From which

$$V_o = \frac{1}{1 + A\beta} d \tag{1.5}$$

Combining (1.5) with (1.3) we get that

$$V_o = \frac{A}{1 + A\beta} V_{in} + \frac{1}{1 + A\beta} d \tag{1.6}$$

The reasoning goes like this. We assume that the amplifier has a very large gain $A \gg 1$. Typically this could be thousands or even hundreds of thousands. This cannot be ideal and large at all frequencies as we shall see later, but as a first approximation we can make this assumption. Therefore in (1.6) we can safely assume that $A\beta \gg 1$. and (1.6) will reduce to

$$V_o = \frac{1}{\beta} V_{in} + 0 \tag{1.7}$$

In other words we have rejected the disturbance and the relationship from output to input is reliant on purely the feedback factor $\beta < 1$. By selecting this factor to be say 0.1, then the amplifier will in closed-loop have a gain of 10 and any noise or disturbances will be rejected. Blacks idea is fundamental to the understanding of modern operational amplifiers. Since the time of Black, early attempts were made to make general purpose amplifiers using feedback. Originally they were made out of vacuum tubes and obviously excessively large until the early 1970s when integrated circuits became commonplace. By re-writing the equations in terms of frequency-dependent transfer-functions we can also show that the closed-loop amplifier has a wider frequency-response than in open-loop. There are hence many desirable effects by this simple idea. A problem arises though is for any reason $1 + A\beta = 0$. Then we get a division by zero and the output goes to infinity. In practice the amplifier either goes unstable or we get sustained oscillations. A side-effect of this is that in those days early oscillators were designed by deliberately making amplifiers unstable in this manner. Accurate sustained oscillations are hard to get however without the amplifier saturating. The term in the denominator $1 + A\beta$ is known as the closed-loop characteristic equation and is the bit earlier recognised by Maxwell and later Routh. When written in terms of a

polynomial equation, the roots must have negative real parts for closed-loop stability. This will be shown later when we study Laplace transforms.

Of course, if all there was to control-theory was applying negative feedback then the subject would be quite basic. Unfortunately, we find that most of the time we apply negative feedback we often do not get the kind of ideal results we expect. Either the closed-loop system oscillates, or the transient behaviour is either too slow or has some form of ringing on it. The art of control engineering is to design compensators or controllers to make the closed-loop system have a specific desired characteristic. In the time of Black, he knew from some practical experience with amplifier design how to stop oscillations though a general theory did not appear until two of his colleagues solved the mystery. Their names were Harry Nyquist (later known for work on sampling theory) and Hendrik Bode. In 1932 Nyquist developed the now famously known Nyquist stability criterion [6]. Later came the work of Bode [7] which is perhaps more familiar to most engineers. The work of Nyquist is important as a method of understanding stability and uses a piece of classical mathematics known as the *principle of the argument*, a technique discovered in the 19th century by mathematician and engineer Augustin–Louis Cauchy [8]. As an analysis tool it has merit but as a development or design tool the work of Bode is more practical for engineers because asymptotic approximations to frequency response are easily sketched whereas a Nyquist plot is polar and more difficult to cope with when designing compensators.

Once the basic theory was understood in electrical engineering there were many applications in aircraft and rocketry. In 1914, Lawrence Sperry invented a closed-loop hydraulic autopilot system for aircraft based on gyroscopes which could follow a compass heading and altitude. Signals from the autopilot moved surfaces on the aircraft to manoeuvre its position.

Rocket pioneer Robert Goddard [9] determined that fins alone were not enough for rocket stability and paved the way for today's inertial navigation units and flight control by using a Gyro to determine its position and to steer. His image appears on a 1964 postal stamp for the US Government (Fig. 1.5). In WWII, the German V2

Fig. 1.5 Robert Godard, rocket pioneer and pioneer of inertial navigation

rockets had used some of Goddard's ideas and used a rudimentary analogue computer, gyroscopes and accelerometer to steer the V2 missiles.

The cold war led the US and Soviets to many forms of advancement in rocket research. Rocket trajectory control and navigation being the two major areas used in missile systems. The work of Rudolf Kalman was perhaps the most well known advancement in the field. Kalman offered a solution to both the problems of control regulation and state estimation (or filtering) [10]. It also signalled the moved by many researchers from transfer-function approaches in control-theory to state-variable methods. In the literature, the transfer-function methods have since been labelled *classical control theory* and anything involving state-space as *modern control theory*. Kalman had also showed that the problem of filtering or estimation of noisy states was the dual of the problem of optimal regulation in the noise-free case for control-systems. Kalman showed that optimal filtering and control were linked closely. The Kalman filter has had almost limitless applications apart from the problem of state-estimation however, in plain signal processing problems, image enhancement, deconvolution, adaptive filtering and system identification [11]. The Kalman regulator or state-feedback optimal control continues to see applications in the process industries [12] and many other industries [13].

In the digital age, many of the earlier problems that were solved have been re-visited using different technology. For example, gyros and accelerometers used for inertial guidance have been fabricated as tiny MEMs devices (Fig. 1.6).

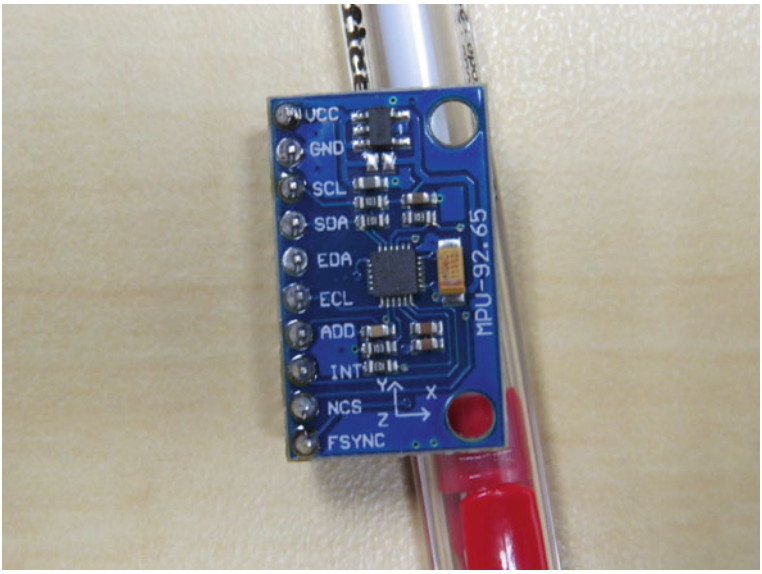


Fig. 1.6 Motion tracking sensor. The MPU-9250 (InvenSense) nine-axis gyro, accelerometer and compass mounted on a break-out board and resting on a pen

Such devices are small enough to be fitted into cell phones and hand-held devices to sense the angular position of a device. For example, when a cell phone changes its orientation, the screen revolves with it to portrait mode. This of course is not a control problem as such, but by taking the same technology and putting it into say a quadcopter, the heading and balance (pitch and roll) can be automatically maintained (Fig. 1.7).

Virtual reality (VR) headsets use similar technologies. A Kalman filter can be used to improve the measurements of the gyro and accelerometer (sensor fusion) and get accurate estimates of pitch and roll angle. With this data a control algorithm, either classical or modern can be applied. Motion-shake in digital cameras can be controlled using similar technology.

In contrast, the Jet Propulsion laboratory had quite a fearsome robot entry to the DARPA challenge in 2013 as shown in Fig. 1.8. You can see by the joints that are a great many degrees of freedom on each leg and the robot is intended for rough terrain where wheels cannot go. Hence the field of Robotics and control-systems are inexorably linked.

Fig. 1.7 Quadcopter hovering. An example of MEMs based embedded inertial control and navigation (pixabay creative commons)

Fig. 1.8 Jet Propulsion Laboratory entry for the DARPA Robotics Challenge in December 2013 (Courtesy JPL Laboratory USA)

1.2 Some Basic Mathematical Models and Templates for Signals and Systems

For control-systems there is a great deal of mathematics usually associated with the subject and for the most part many of it is unavoidable. However, the engineers approach is to categorise system models in such a way that they are easily recognisable by the type of output they give for a given input. First we need to have a set of commonly used signals as used as driving signals and a reason for using them. They are used mainly for test purposes.

We start with the unit step input, shown in Fig. 1.9.

The unit step signal is useful as it gives a sudden surge of energy to a system and the output (or response) to this step tells us about the transient behaviour of the system. It is defined mathematically as $u(t) = 1, t > 0$. In some texts we see signals defined as zero for negative-time i.e. for the step we can say $u(t) = 0, t < 0$. In this

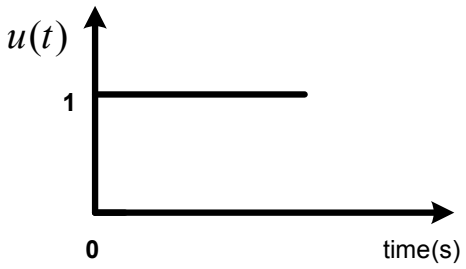


Fig. 1.9 Unit step signal as used for test purposes

book we try and ignore this style and take it that all time starts from zero. A signal that starts in negative time is known as *non-causal* and for our examples of realizable signals and systems this cannot be the case. The definition of non-causal does have its uses, particular in some areas such as optimal filtering. The step signal need not start at time zero and can start at some later time say $t = \tau$ (Fig. 1.10).

It is defined mathematically as $u(t) = 1, t > \tau$ and zero otherwise. A better notation is to use the method of Heaviside the famous English electrical pioneer and instead write $u(t - \tau)$. This is the so-called shift operator method and makes the definition more compact. Practically a step can be obtained from a square-wave oscillator or function generator by setting the repetition rate (period) to be long enough. It can also be very easily created in software as a constant of unit or scaled to any desired value.

The unit ramp signal (Fig. 1.11) is another commonly met signal and can be created as part of a triangular wave. Here it is termed unit because the gradient is unity.

This signal is also used as a more exacting input to a control-system to check how a system behaves for a rate-of change of input. For example, a motor could have a step applied and the speed rises up to some constant speed (we say when the time span is long enough that it has reached steady-state). With a ramp we test its response to a velocity input. An elevator system or maybe a crane could have a waveform as shown below in Fig. 1.12 as its input.

The waveform in Fig. 1.12 ramps up with slope unit to a value of 1 and then remains constant for 2 s and finally ramps negatively down to zero. This could be the desired input to a control-system and the motor output has to follow it as closely as possible. The area under the graph is of course distance travelled if $v(t)$ represents velocity.

Finally, the waveform we use for measuring frequency response, the sinewave (Fig. 1.13).

The sinewave is used in experiments to obtain the frequency-response of an unknown system. In Fig. 1.14. The sinewaves frequency is varied and careful measurements of the magnitude and phase of the output signal is made with respect to the input (Fig. 1.15).

Obtaining a frequency response is generally not an easy task for some electro-mechanical systems and on some, this direct method may not be suitable.

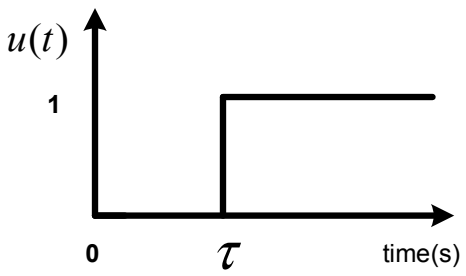


Fig. 1.10 Delayed unit step signal

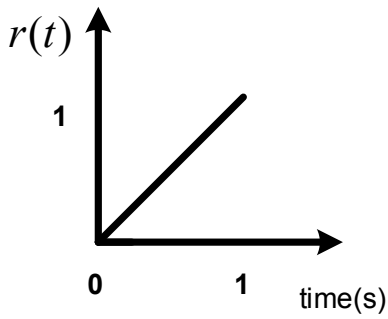


Fig. 1.11 Unit ramp signal $r(t) = t, t > 0$

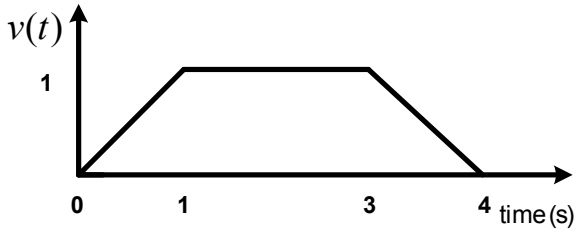


Fig. 1.12 Composite waveform

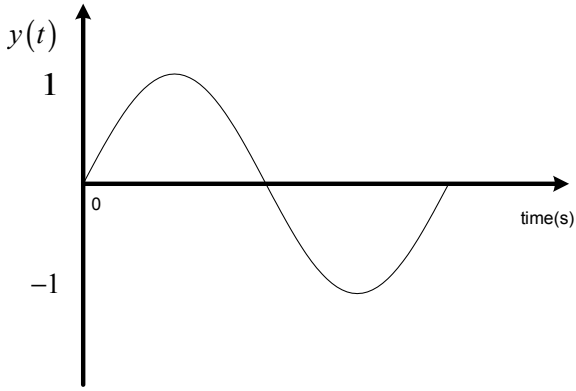


Fig. 1.13 Sine waveform $y(t) = \sin(\omega t)$

Usually the frequency response is taken from a very low frequency of a few Hz up to the structural (mechanical) resonance of the system or when it begins to vibrate. The theory behind this is dealt with in a later chapter of this book.

To examine the types of system we encounter is a wide ranging topic, but we can usually narrow it down to a few commonly met ones. For example, consider the simple network consisting of a capacitor and resistor as shown in Fig. 1.16. From Kirchoffs voltage law we can sum the voltage around the circuit.

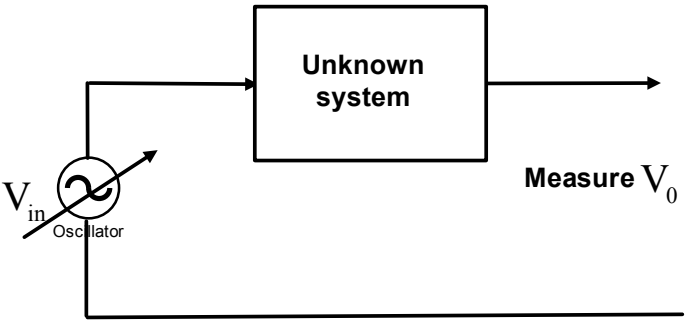


Fig. 1.14 Basic Frequency response experiment

Fig. 1.15 Magnitude ratio $\left|\frac{V_o}{V_{in}}(t)\right|$ and phase-shift $\phi(t)$ are measured and plotted on separate graphs as frequency is varied

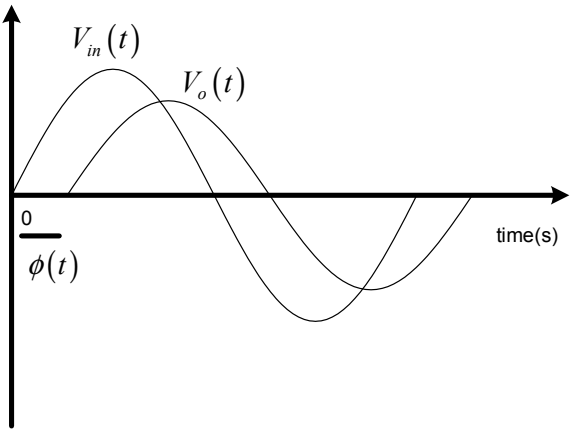
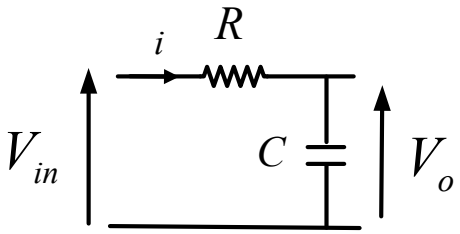


Fig. 1.16 RC Network circuit



Example 1.1 RC Circuit

The current through the capacitor is $i = C\frac{dV_o}{dt}$ and hence we must have that

$$\mathrm{V_{in}(t)} = \mathrm{V_o(t)} + \mathrm{CR}\frac{\mathrm{dV_o(t)}}{\mathrm{dt}} \tag{1.8}$$

This is an ordinary linear differential equation of the first-order (sometimes known as a first-order ODE). We can solve this ODE for particular inputs. The most interesting input to choose is the unit-step input from which we just put $V_{in} = 1$ in our solution. For this input we can then calculate the output using the classical ODE solution approach. We show the solution merely to illustrate the complexity of what is quite a relatively simple problem, at least one of the simplest in this field.

$$\begin{aligned}
&\frac{dV_o(t)}{dt} + \frac{1}{RC}V_o(t) = \frac{1}{RC}V_{in} \\
&\frac{dV_o(t)}{dt} + \frac{1}{RC}(V_o(t) - V_{in}) = 0 \\
&\frac{dV_o(t)}{(V_o(t) - V_{in})} = -\frac{dt}{RC} \\
&\int \frac{dV_o(t)}{(V_o(t) - V_{in})} = -\int \frac{dt}{RC}
\end{aligned} \tag{1.9a, b, c, d}$$

Take logs of both sides of (1.9d) above gives

$$\begin{aligned}
&\int \frac{dV_o(t)}{(V_o(t) - V_{in})} = -\int \frac{dt}{RC} \\
&\ln((V_o(t) - V_{in})) + c_1 = -\frac{t}{CR} + c_2 \\
&\ln((V_o(t) - V_{in})) = -\frac{t}{CR} + c
\end{aligned} \tag{1.10, a, b, c}$$

(where in (1.10c) we have combined the two integration constants c_1 and c_2 into one constant c). Now assume an initial condition, that when time t = 0, the capacitor voltage $V_o(t) = 0$.

$$\begin{aligned}
\ln((0 - V_{in})) &= -\frac{t}{CR} + c \\
c &= \ln((-V_{in}))
\end{aligned}$$

So we now have obtained the integration constant c. Substituting back into (1.10c) gives us

$$\begin{aligned}
&\ln((V_o(t) - V_{in})) = -\frac{t}{CR} + \ln((-V_{in})) \\
&\ln((V_o(t) - V_{in})) - \ln((-V_{in})) = -\frac{t}{CR} \\
&\ln\left(-\frac{((V_o(t) - V_{in}))}{V_{in}}\right) = -\frac{t}{CR} \\
&\ln\left(\frac{((V_{in} - V_o(t)))}{V_{in}}\right) = -\frac{t}{CR}
\end{aligned} \tag{1.11, a, b, c, d}$$

From (1.11,a,b,c,d) by the definition of the natural logarithm we have

$$e^{-\frac{t}{CR}} = \left(\frac{(V_{in} - V_o(t))}{V_{in}}\right) \tag{1.12}$$

Finally we have the result

$$V_o(t) = V_{in}\left(1 - e^{-\frac{t}{CR}}\right) \tag{1.13}$$

We can see immediately that as $t \to \infty$ we get the steady-state solution $V_o(t) \to V_{in}$. The way the exponential term dies out depends on the product CR which is usually named T = CR the *time-constant* of the circuit. Large T means it takes a long time to reach steady-state and a small time-constant reaches there faster. The point here is the complexity of getting the solution to this relatively simple problem which is only first order! The so-called first-order step-response is given by (1.13). Consider a case where the time-constant T = 1. The step response has the form shown below in Fig. 1.17.

We can work backwards from a first order step response and identify the time-constant since when time t = T in (1.13) we get $V_o(t) = V_{in}(1 - e^{-1}) = 0.632V_{in}$. So by looking at 63.2% of the final value and reading the value of time at that amplitude we obtain the time-constant. An alternative method is to measure the tangent to the response curve and measure its slope. Differentiating (1.13) with respect to time gives us $\frac{dV_o(t)}{dt} = -\frac{1}{CR}V_{in}$ when t = 0. So the slope is −1/T passing

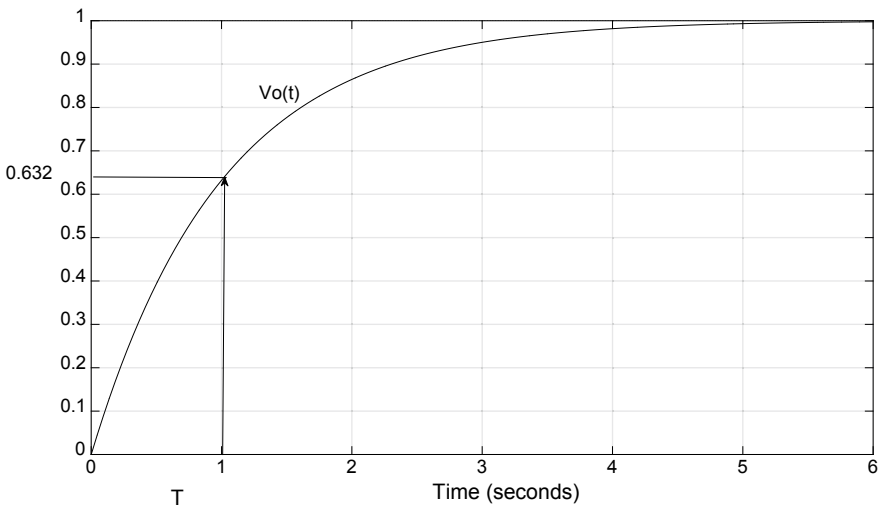


Fig. 1.17 Shows step response of a first-order system time constant T = 1

through t = 0 and tangential to the plot. We also name the value of the voltage when it becomes constant as the *steady-state* voltage and all values before it as *transient*. So the steady-state voltage is 1v, the same as the input. This is as expected since the capacitor charges to the voltage which is applied.

The important point taken from this is that there are a great many systems that are first-order or can be approximated to be first-order.

Example 1.2 RL Circuit

Once we have the template for first-order systems we can apply the rules to all of them. For example, consider the circuit in Fig. 1.18.

By Kirchoffs voltage law we can show

$$V_{in} = V_o + \frac{L}{R}\frac{dV_o}{dV_{in}} \tag{1.14}$$

and by comparison to (1.18) we can write down immediately the step-response to 1v input as

$$V_o(t) = V_{in}\left(1 - e^{-\frac{t}{L/R}}\right) \tag{1.15}$$

This is a similar result as before except the time-constant has now change to be T = L/R. We need not re-solve the ODE because it becomes a template for similar problems. This is not restricted to just electrical circuits.

Example 1.3 DC Motor

Consider the simplified dc-motor in Fig. 1.19. We have ignored an armature inductance.

A dc motor has a back-emf E_g which is proportional to the speed of the motor ω(rads/s) provided the field is fixed, which in the case of a permanent magnet motor is the case. The constant of proportionality is known as the motor constant k which in many cases is unknown, but does not effect our analysis. The armature current is labelled I_a and the armature driving voltage is V_{in}. We then write the equation of the motor as

$$V_{in} - E_g = I_a R_a \tag{1.16}$$

where R_a is armature resistance. Back emf is given by

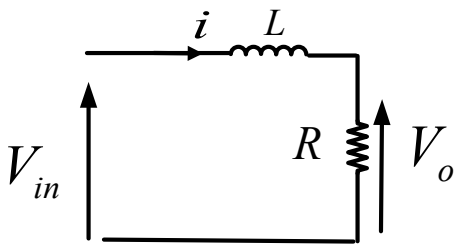


Fig. 1.18 RL network circuit

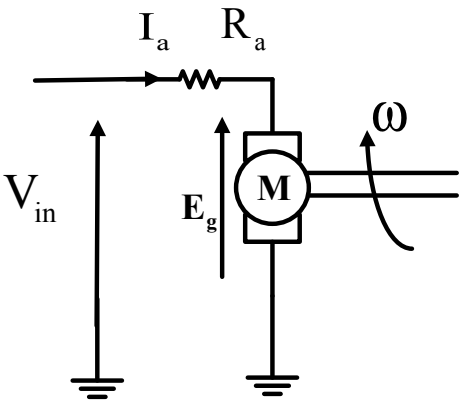


Fig. 1.19 Armature-controlled dc motor (permanent magnet)

$$E_g = k\omega \tag{1.17}$$

and developed torque of the motor

$$T = kI_a \tag{1.18}$$

Where k is the motor constant and turns out to be the same used for current as for speed when using these units. Now we do some formula substitution. Substitute current from (1.18) into (1.16) and use (1.17). This gives

$$V_{in} - k\omega = \frac{T}{k} R_a \tag{1.19}$$

In Fig. 1.19 we see that speed is the output but we have a torque term in (1.9a,b,c,d) which we must express in terms of speed. We can do this assuming that the motor has a moment of inertia J and using Newton's law as applied to rotational systems (ignoring friction)

$$T = J\frac{d\omega}{dt} \tag{1.20}$$

Combining we get

$$V_{in} = k\left(\omega + \frac{R_a J}{k^2}\frac{d\omega}{dt}\right) \tag{1.21}$$

So other than the scaling outside of the equation by k, the model of this system is also first-order with time-constant $\frac{R_a J}{k^2}$. It is perhaps slightly more difficult here to

recognise that the output speed rises up not to the same value as V_{in}, but instead to V_{in}/k. So for any dc motor we can approximate it to a first-order system and find its time constant by measurements on the step-response. It is clear that although the ODE approach does work, as it did in the time of Maxwell, it is far from intuitive and can get quickly complicated for more complex systems. For example, even for this case if we consider viscous friction of the motor (or for a load that is connected to the shaft) we get instead of (1.20)

$$T = J\frac{d\omega}{dt} + F\omega \tag{1.22}$$

where F is a coefficient of viscous friction. Then by a little algebra we get

$$V_{in} = \frac{k^2 + R_a F}{k}\left(\omega + \left(\frac{R_a J}{k^2 + R_a F}\right)\frac{d\omega}{dt}\right) \tag{1.23}$$

The ODE is still first order but we lose clarity as the problem becomes more complex. We can at least see that the time-constant is $T = \left(\frac{R_a J}{k^2 + R_a F}\right)$. We can see that provided the speed settles down to a constant value that ultimately e must have that $\frac{d\omega}{dt} \to 0$ as $t \to \infty$ leaving us

$$\omega_{ss} \to \frac{k}{k^2 + R_a F} V_{in} \tag{1.24}$$

where ω_{ss} is the steady-state speed of the motor.

To illustrate this further consider the following system in Fig. 1.20.

Example 1.4 RLC Circuit

Summing voltages around the loop we get

$$V_{in}(t) = L\frac{di(t)}{dt} + Ri(t) + V_0(t) \tag{1.25}$$

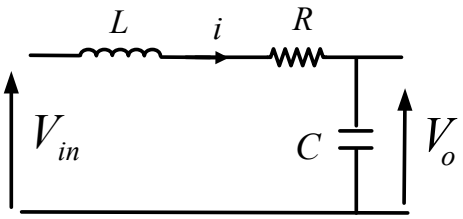


Fig. 1.20 RLC Circuit

but the current $i = C\frac{dV_o(t)}{dt}$. Substitute into (1.25) and we get

$$V_{in}(t) = LC\frac{d^2V_0(t)}{dt} + CR\frac{dV_0(t)}{dt} + V_0(t) \tag{1.26}$$

This is a second order ODE with constant coefficients (assuming neither of the three components values changes with time). It has a forced input. The literature is full of ways to solve such problems and the method of solution usually requires us first to solve the Homogeneous (or non-forced) problem

$$LC\frac{d^2V_0(t)}{dt} + CR\frac{dV_0(t)}{dt} + V_0(t) = 0 \tag{1.27}$$

It can then be shown that the solution to the Homogeneous problem involves a function of the form

$$V_{oh}(t) = c_1e^{r_1t} + c_2e^{r_2t} \tag{1.28}$$

where r_1, r_2 are the roots of x of the quadratic *characteristic equation*

$$LCx^2 + CRx + 1 = 0 \tag{1.29}$$

and c_1, c_2 are constants. Writing the characteristic equation more generally we have

$$ax^2 + bx + c = 0. \tag{1.30}$$

with $a = LC, b = CR, c = 1$.

There are three possible solutions which depend on the roots of the characteristic equation (or polynomial). By using the quadratic formula we get

1. *Overdamped.*

$$b^2 - 4ac > 0$$

There are two real roots for this case and the solution becomes

$$V_{oh}(t) = c_1e^{r_1t} + c_2e^{r_2t} \tag{1.31}$$

2. *Critically Damped*

$$b^2 - 4ac = 0$$

There is only one repeated root r and the solution is

$$V_{oh}(t) = c_1 e^{rt} + c_2 t e^{rt} \tag{1.32}$$

3. *Underdamped*

$$b^2 - 4ac < 0$$

There are two complex roots say $r_1 = \alpha + j\beta, r_2 = \alpha - j\beta$.

$$\begin{aligned} V_{oh}(t) &= c_1 e^{\alpha t}\cos(\beta t) + c_2 e^{\alpha t}\sin(\beta t) \\ &= C e^{\alpha t}\cos(\beta t - \phi) \end{aligned} \tag{1.33}$$

All of these solutions die out and go to zero as time progresses since in the Homogeneous case there is no forced input. To find the general solution we first need to find a particular solution, call this $V_{op}(t)$ and the general solution is then found by the addition of the two $V_o(t) = V_{oh}(t) + V_{op}(t)$. However, finding solutions in this way is rather cumbersome and although classical in terms of mathematics, is rather old fashion in the engineering world. Instead we rely on the *Laplace Transform* and for more complicated models we simulate using modern numerical software such as MATLAB. Differential equations still play their part when we encounter state-variable models. For a thorough treatment on engineering applications of ODEs the reader is referred to [14].

Chapter 2
The Laplace Transform and Linear Time-Invariant Systems

The previous chapter gave a brief introduction to the problem of feedback control and the modelling and mathematical methods needed. We showed that differential equations although ok for certain simple problems, rapidly become cumbersome for more complex engineering systems. Therefore the control-engineer tries where possible to avoid using ODEs and instead uses the mathematics pioneered by the French mathematician Pierre-Simon, marquis de Laplace (1749–1827). He is recognised as one of the all-time great scientists [15]. By using the Laplace transformation named in his honour, not only is the analysis of control-systems greatly simplified, but the design too benefits greatly due to block-diagram methods.

2.1 The Laplace Transform

The Laplace Transform enables us to change ODEs to algebraic form (the s-domain), and manipulate them in that domain before transforming back to the time-domain. Instead of ODEs, we end up with polynomials or ratios of polynomials known as transfer-functions. We can find Laplace Transforms of signals or systems and often a signal has a system with the same Laplace-Transform (LT). The definition of the unilateral LT is that for some signal $f(t), t > 0$ its corresponding LT is found from the integral

$$F(s) = \int_0^\infty f(t)e^{-st}dt \tag{2.1}$$

where $s = \sigma + j\omega$ is a complex variable (often termed complex frequency).

The shorthand way to denote a Laplace transform is by using script $\mathcal{L}$ thus

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_2

$$F(s) = \mathcal{L}\{f(t)\} \tag{2.2}$$

The *inverse* Laplace Transform is usually given the notation $f(t) = \mathcal{L}^{-1}\{F(s)\}$.

In some cases the lower limit of (2.1) is minus infinity indicating the so-called *Bilateral* LT. Usually it is convenient to use the Bilateral form in some proofs or in special optimal filters. For the time being we will restrict to the unilateral form. Now consider some examples.

2.1.1 Examples of Laplace Transform of Signals

We can divide the problem of finding the Laplace-Transform into one of signals and systems. We first consider signals.

Example 2.1. Unit Step $f(t) = 1, t > 0$
The graph of the step signal has been illustrated in Chap. 1. By substituting into (2.1) we get

$$\begin{aligned} F(s) &= \int_0^\infty e^{-st} dt \\ &= \left[-\frac{1}{s} e^{-st} \right]_{t=0}^{t=\infty} \\ &= \frac{1}{s} \end{aligned} \tag{2.3a, b, c}$$

Example 2.2. Unit-Ramp $f(t) = t, t > 0$

$$F(s) = \int_0^\infty t e^{-st} dt \tag{2.4}$$

Integrating by parts, this is a standard integral, we find

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

Example 2.3. Unit Impulse $\begin{aligned} \delta(t) &= 1, t = 0 \\ &= 0, t \neq 0 \end{aligned}$
The unit impulse shows up more in signal processing than control-systems as it is not used as much as a test signal. However. It does play a significant role in sampling theory for digital control and is worth including here. Figure 2.1 shows a graph of the unit impulse (or delta-function) [16].

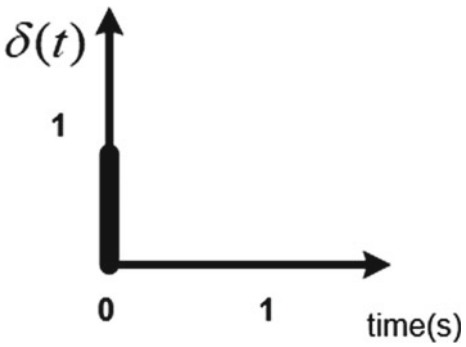


Fig. 2.1 Unit impulse

$$F(s) = \int_0^{\infty} \delta(t) e^{-st} dt = 1 \tag{2.5}$$

The above integral is found because the impulse only exists at time t = 0 and is zero everywhere else. Therefore by setting t = 0 in (2.4) removes the integral sign since the summation is at one point in time only. In control-systems, if such a signal is used as a test signal as the input to a system, the impulse-response of the system becomes the output of the system (Fig. 2.2).

Note that the output must always die down to zero as the impulse is a short burst of energy and the system must lose this energy after a period of time. The shape of the output will depend on what kind of impulse response g(t) we get. It is not commonly used to identify unknown systems, unlike the step, but in other disciplines (e.g. signal processing) it is very common. For example the acoustic impulse response of a large room or hall can be found by popping a balloon and recording the result digitally. This captures the impulse response of the room and for large acoustically colourful areas we can then use this data and use convolution to create music that was recorded in a dull or damped environment sound reverberant, as if it was recorded in a large hall.

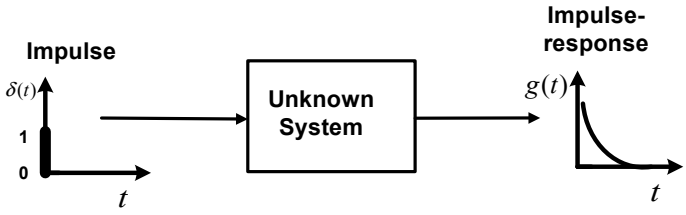


Fig. 2.2 Impulse response of unknown system

Example 2.4. Sine Wave $f(t) = \sin(\omega t)$

$$F(s) = \int_0^\infty \sin(\omega t) e^{-st} dt \tag{2.6}$$

Integrating by parts and after a lot of algebra we get

$$F(s) = \frac{\omega}{s^2 + \omega^2} \tag{2.7}$$

Another way is to substitute Euler's identity for sin and use $\sin(\omega t) = \frac{e^{j\omega t} - e^{-j\omega t}}{2j}$. We can show that

$$\mathcal{L}\{e^{j\omega t}\} = \frac{1}{s - j\omega}$$
$$\mathcal{L}\{e^{j\omega t}\} = \frac{1}{s + j\omega}$$

and by adding terms and multiplying by $1/2j$ we can obtain the same result.

Example 2.5. Decaying Exponential $f(t) = e^{-at}$

$$F(s) = \int_0^\infty e^{-at} e^{-st} dt = \int_0^\infty e^{-(s+a)t} dt \tag{2.8}$$

and this becomes

$$F(s) = \mathcal{L}\{e^{-at}\} = \frac{1}{s + a} \tag{2.9}$$

The most commonly met Laplace Transforms were calculated hundreds of years ago and therefore we need not derive them again. They appear in a table of transforms.

2.1.2 Laplace Transform of Systems

The last two entries in Table 2.1 are of particular importance here as we see that a derivative in an ODE can be replaced by multiplication by s provide there are no initial conditions. Likewise for a second derivative and no initial conditions we get

Table 2.1 Common laplace transforms

Time-domain	Laplace domain
$\delta(t)$ Unit impulse at t = 0	1
$u(t) = 1$, Unit step function	$\frac{1}{s}$
t Unit ramp	$\frac{1}{s^2}$
t^n	$\frac{n!}{s^{n+1}}$
e^{-at}	$\frac{1}{s+a}$
$\sin(\omega t)$ Sine wave	$\frac{\omega}{s^2+\omega^2}$
$\cos(\omega t)$	$\frac{s}{s^2+\omega^2}$
$e^{-at}\sin(\omega t)$	$\frac{\omega}{(s+a)^2+\omega^2}$
$\frac{dy(t)}{dt}$ 1st derivative	$sy(s) - y(0)$
$\frac{d^2y(t)}{dt^2}$ 2nd derivative	$s^2y(s) - sy(0) - \frac{dy(0)}{dt}$
$U(t-\tau) = 1$, unit delayed step	$\frac{e^{-s\tau}}{s}$
$y(t-\tau)$, delayed signal	$Y(s)e^{-s\tau}$

$\mathcal{L}\left\{\frac{d^2y(t)}{dt^2}\right\} = s^2y(s)$ and the same applies for higher order derivatives. So going back to our RC network described by (1.18).

$V_{in}(t) = V_o(t) + CR\frac{dV_o(t)}{dt}$, we can take Laplace Transforms of both sides of the equation as follows and obtain a simple algebraic expression.

$$\mathcal{L}\{V_{in}(t)\} = \mathcal{L}\left\{V_o(t) + CR\frac{dV_o(t)}{dt}\right\} \tag{2.10}$$

Giving for zero initial conditions (i.e. no initial charge on the capacitor).

$$V_{in}(s) = V_o(s) + CRsV_o(s) \tag{2.11}$$

$$V_{in}(s) = [1 + CRs]V_o(s) \tag{2.12}$$

Now we see the beauty of this method by defining the ratio $\frac{V_o}{V_{in}}(s)$ as the transfer-function of the system.

$$\frac{V_o}{V_{in}}(s) = \frac{1}{[1+CRs]} = \frac{1}{[1+sT]} \tag{2.13}$$

This now becomes the template for all first-order systems in the Laplace domain instead of an ODE. It is algebraic in nature (its denominator is a first-order polynomial $1 + sT$).

Likewise our second-order ODE (1.26) for the RLC circuit becomes

$$V_{in}(t) = LC\frac{d^2V_0(t)}{dt} + CR\frac{dV_0(t)}{dt} + V_0(t)$$

By taking Laplace Transforms the second-order derivatives become powers of s^2 and the first order powers of s. We quickly obtain (for zero initial conditions)

$$V_{in}(s) = \left[LCs^2 + CRs + 1\right]V_0(s) \tag{2.14}$$

and the ration between output and input becomes the transfer function

$$\frac{V_0}{V_{in}}(s) = \frac{1}{LCs^2 + CRs + 1} \tag{2.15}$$

and we label this as a second-order transfer-function. This leads us to a systems approach using transfer-functions instead of ODEs. They are far easier to remember and to manipulate than ODEs.

2.2 Linear Time-Invariant Systems

The usual approach to modelling and design of feedback systems is to make certain assumptions. Fortunately there are a great many applications where these assumptions hold or at least approximately hold in the real world. Often they do not and we as engineers find a way around the problem by approximating to our ideal theory or, as a last resort we have to extend the theory. The ideal systems we consider first are *linear* and *time-invariant*.

2.2.1 Linear Systems

There are a great many definitions of what a linear system is depending on whether you are a rigorous mathematician or a more practical engineer. The more rigorous approach is to consider what happens to our system when we have more than one input. A linear system must satisfy *superposition* (Fig. 2.3).

If a system has two inputs, then if we set each input to zero in turn and provide any input to the other, then the overall output will be the same as if both inputs were applied simultaneously.

Linear systems also satisfy *Homogeneity*.

This is if an input is amplified (or scaled) by some amount the output of the system is amplified by *the same* amount. For example, for a linear amplifier, if we double the input signal the output signal also doubles. Of course with any real

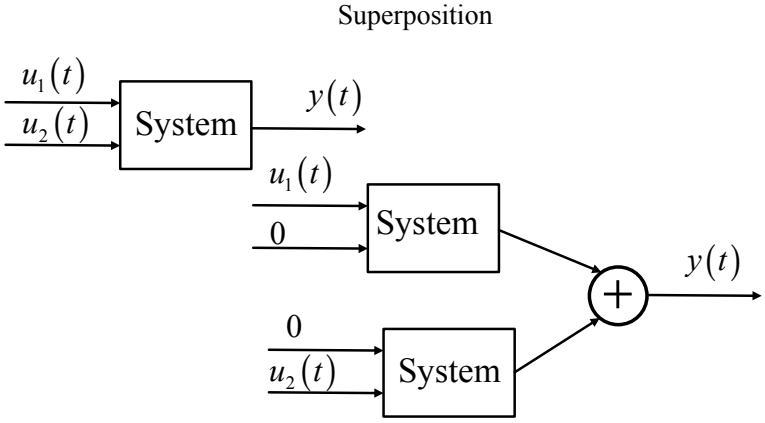


Fig. 2.3 Superposition principle

amplifier its output will eventually saturate and at this level of input it can no longer be considered linear. Mathematically, if the scaling factor is "a" then $aV_{in}(t) \rightarrow aV_o(t)$. Most practicing Engineers may not always apply two inputs to a system but may use a simpler approach such as applying a sine waveform and looking at the quality of the output (that is, does it still look like a pure sine wave or has it got harmonics in it?). In extreme circumstances if we apply a sine wave and a square wave comes out we conclude the system is nonlinear or is at least operating in the nonlinear region. We sweep the sine wave to check at all useable frequencies within the useable bandwidth.

2.2.2 Time-Invariant Systems

These are mostly encountered by practicing engineers. Usually our mathematical model or transfer function does not change with time in any way. An input applied at time zero gives the same output if we delay the input and apply it at a different time. This seems like an obvious thing, but there may well be system whose characteristics change over time due to temperature or perhaps a chemical process. This is a bit like having a time-constant in a first-order system that is a function of time and not constant. Mathematically we have that for an input $u(t)$ and output $y(t)$ that to satisfy time-invariance $u(t)$ gives rise to $y(t)$ and $u(t-\tau)$ must give rise to $y(t-\tau)$. So for a given delay at the input the output is delayed by the same amount (Fig. 2.4).

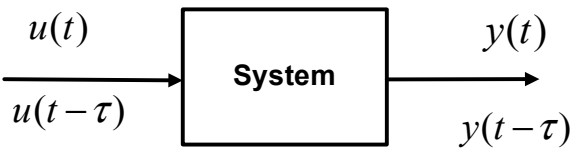


Fig. 2.4 Definition of a time-invariant system

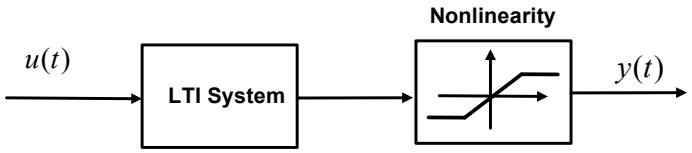


Fig. 2.5 LTI system followed by a nonlinearity

2.2.3 Linear Time-Invariant Systems (LTI)

Linear time-invariant systems are systems which satisfy both the linear and time-invariance properties above. For most of this book we consider LTI systems only unless otherwise stated. We can only use Laplace Transforms and the transfer function approach on LTI systems though often this rule is broken when static nonlinearities such as saturation are considered part of a control-system. We may draw the nonlinearity but separate the LTI part of the system as a separate block. For example (Fig. 2.5).

Provided we stick to the linear straight line part of the nonlinearity the linear rules apply. Otherwise nonlinear effects take place and we need to modify the theory.

2.3 Cascading Systems

When the output of one system is connected to the input of another we term this a *cascaded* system. In order for this to happen the output of the first system must not load (in electrical terminology) the input of the next. This is easy to see for two ideal amplifier. Suppose one has a gain of 10 and the next a gain of 2 (Fig. 2.6).

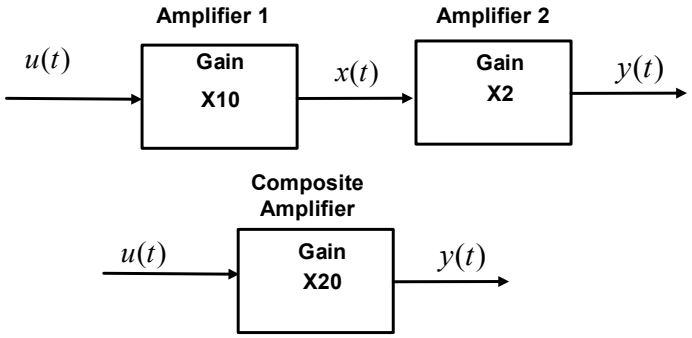


Fig. 2.6 Two ideal amplifiers in cascade

This would appear to be common sense since $x(t) = 10u(t), y(t) = 2x(t)$ then by combining these equations we have $y(t) = 20u(t)$. That is, we multiply the two gains together (or add if the gain is expressed logarithmically in dB). However, if the first amplifier has an output impedance which is low as compared to the input impedance of the second, then a loading condition occurs and when the gain is measured in cascade it may well be less than 20. This occurs even though the individual gains are correct. In fact voltage is lost in the connection of the two because current is drawn in the non-ideal case. For *operational amplifiers* the current drawn is negligible however and we are safe to use them in this way.

The question arises however as to how the mathematics works when two, more complex systems are connected if they are described by differential equations. The solution to this lies with the unit impulse and impulse response studied earlier (Fig. 2.2). The impulse-response is a time-domain way of working and its has been known for 100 s of years that for some dynamic $g(t)$ system driven by an impulse, that the output is found from the *convolution* integral.

$$g(t) = \int_0^t \delta(t-\tau)g(\tau)d\tau \tag{2.16}$$

The above integral is solved because the unit impulse is zero everywhere except $\tau = t$. This is extended for any type of input and we can show that for *two* cascaded systems with impulse responses $g(t), h(t)$ the composite impulse response $f(t)$ is given by the convolution integral

$$f(t) = \int_0^t g(t-\tau)h(\tau)d\tau \tag{2.17}$$

The word convoluted in the English language means mixed-up or twisted, and it is via the convolution integral that one system gets connected to another, in the mathematical sense. Simple multiplication as is the case with two amplifiers is no longer applicable. We write in shorthand notation that the symbol * represents convolution and therefore

$$f(t) = h(t) * g(t) \tag{2.18}$$

We can easily show with a change of variable in (2.17) that

$$f(t) = \int_0^t h(t-\tau)g(\tau)d\tau \tag{2.19}$$

and that therefore

$$f(t) = g(t) * h(t) \tag{2.20}$$

Convolution is therefore commutative. It doesn't matter (in theory at least) which way round we connect our systems, the same output occurs. Of course practically this does not hold, as with a feedback control system an amplifier always precedes a motor for instance or the motor would not drive at all due to lack of current. In this book, we avoid convolution by working with Laplace Transforms. An important result is to take the Laplace Transform of (2.17) thus

$$\mathcal{L}\{f(t)\} = \int_{t=0}^{t=\infty} \int_{\tau=0}^{\tau=t} g(t-\tau)h(\tau)d\tau e^{-st}dt \tag{2.21}$$

To proceed further it is perhaps easier to change the limits on our convolution integral. This is done without losing any generality. We can write

$$\mathcal{L}\{f(t)\} = \int_{t=0}^{t=\infty} \int_{\tau=0}^{\tau=\infty} g(t-\tau)h(\tau)d\tau e^{-st}dt \tag{2.22}$$

We can get away with this since the impulse response $g(t-\tau) = 0, \tau > t$ i.e. it does not exist in negative time. We then re-arrange the order of integration which in this case is straight-forward.

$$\mathcal{L}\{f(t)\} = \int_{\tau=0}^{\tau=\infty} \int_{\tau=0}^{\tau=\infty} g(t-\tau)h(\tau)e^{-st}dtd\tau \tag{2.23}$$

By substituting $v = t - \tau$, $dv = dt$ obtain

$$\begin{aligned}\mathcal{L}\{f(t)\} &= \int_{v=0}^{v=\infty} \int_{\tau=0}^{\tau=\infty} g(v)h(\tau)e^{-s(v+\tau)}dvd\tau \\ &= \int_{v=0}^{v=\infty} g(v)e^{-sv}dv \int_{\tau=0}^{\tau=\infty} h(\tau)e^{-s\tau}d\tau\end{aligned} \tag{2.24, 2.25}$$

Giving us the important result that the Laplace Transform of convolution is

$$F(s) = G(s)H(s) \tag{2.26}$$

In words this means that *Convolution in the time-domain is the same as Multiplication in the s-domain.*

The engineering importance of this result is that when cascading LTI systems we need only multiply their transfer-functions together instead of performing convolution. Hence we can work entirely in terms of polynomials or rather ratios of polynomials. For example, two first-order systems in cascade (Fig. 2.7).

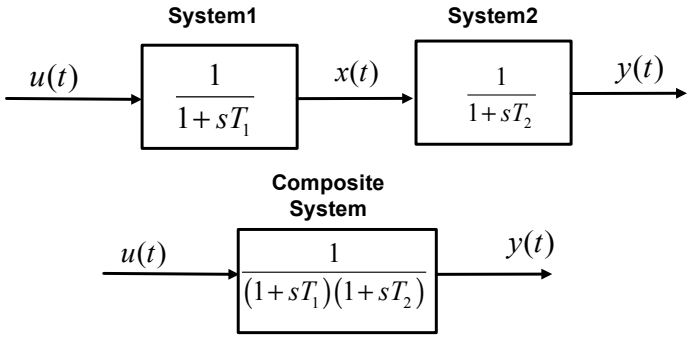


Fig. 2.7 Two cascaded first-order systems

Proceed with some caution however, since our first-order RC network described in (2.13) *when cascaded with itself gives us* Fig. 2.8.

If the individual TFs are

$$G(s) = \frac{1}{[1+sT]} \tag{2.27}$$

And from our convolution result we may expect that the cascaded TFs is

$$G_c(s) = \frac{1}{[1+sT]^2} \tag{2.28}$$

However, if we were to analyse Fig. 2.8 we would find it as having a second-order transfer function, but not (2.28). It may bear some resemblance be we know from Kirchhoff's law that current is drawn into the second RC from the first and clearly we can never just multiply the two together. If instead we have an isolating amplifier between them then we do get (2.28) as the TF (Fig. 2.9).

As far as possible when we design analogue control-systems or model them mathematically, it is best to ensure that adequate isolation is placed between various sub-systems. With operational amplifiers this is of course an easy task since their characteristic is nearly ideal with very low output impedance.

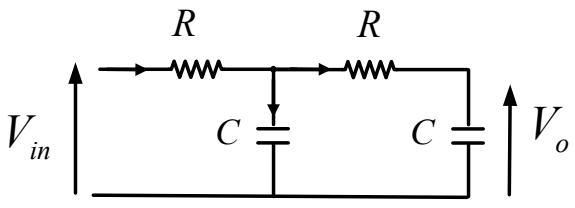


Fig. 2.8 Two first-order RC networks directly connected together

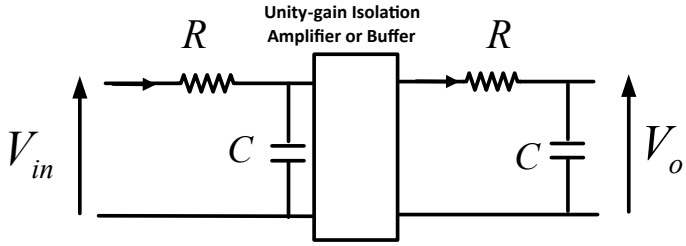


Fig. 2.9 Cascaded first-order systems where we can multiply the TFs

2.4 The Ubiquitous Integrator

One of the most common transfer-functions that is come across in control-systems theory and application is the integrator. The integrator is just as the word says on the packet, it integrates! Just like integration in mathematics it perform the exact same function (Fig. 2.10).

Its equation is found in nearly every textbook on electronics by summing the current at the non-inverting input to the amplifier. We get

$$V_o(t) = -\frac{1}{RC}\int_0^t V_o(t)dt \tag{2.29}$$

When we take Laplace Transforms, for transfer-functions we assume zero initial conditions. So the integrator TF is

$$V_o(s) = -\frac{1}{sCR}V_{in}(s) \tag{2.30}$$

The inverting sign is of no concern since in feedback systems the sign can easily be changed elsewhere. By defining $K = \frac{1}{CR}$ we can re-write (2.30) in the more familiar form

$$\left|\frac{V_o}{V_{in}}(s)\right| = \frac{K}{s} \tag{2.31}$$

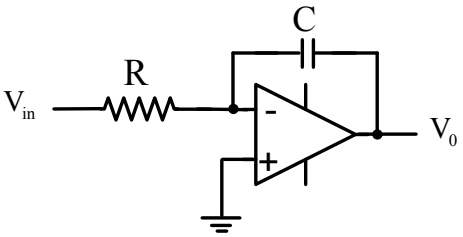


Fig. 2.10 Analogue integrator

We note that from (2.31) that this equation looks familiar. The step signal has also a similar TF. In fact many signals and systems have the same LTs. The Laplace method converts both signals and systems into ratios of polynomials. The integrator here in electronic format also appears in mechanical format as well. It is not so much that there is a mechanical integrator, but by adding certain transducers which measure say angular position, an integrator must be added to the block-diagram of the system model. For example, consider a rotational mass with moment of inertia J, coefficient of viscous damping F and a driving torque T. The output angular velocity is ω rads/s. we have the ODE of first order (Fig. 2.11).

$$\mathrm{T} = \mathrm{J}\frac{\mathrm{d}\omega}{\mathrm{dt}} + \mathrm{F}\omega \tag{2.32}$$

Taking LTs

$$\mathrm{T}(\mathrm{s}) = (\mathrm{Js} + \mathrm{F})\omega \tag{2.33}$$

we get a TF

$$\frac{\omega}{\mathrm{T}}(\mathrm{s}) = \frac{1}{\mathrm{Js} + \mathrm{F}} \tag{2.34}$$

However, if a sensor is attached to the end of the shaft which measures angular position θ_o, then the transfer function needs an integrator in the mathematical model since angular position is the integral of angular velocity. $\theta_o(t) = \int_0^t \omega(t)dt$. We get a modified TF

$$\begin{aligned}\frac{\theta_\mathrm{o}}{\mathrm{T}}(\mathrm{s}) &= \frac{\omega}{\mathrm{T}}(\mathrm{s})\frac{\theta_\mathrm{o}}{\omega}(\mathrm{s}) \\ &= \frac{1}{\mathrm{s}(\mathrm{Js} + \mathrm{F})}\end{aligned} \tag{2.35a, b}$$

We see an extra integrator appears in the TF just by nature of the problem itself. Rotational systems always have this integrator if we measure angular position with a sensor.

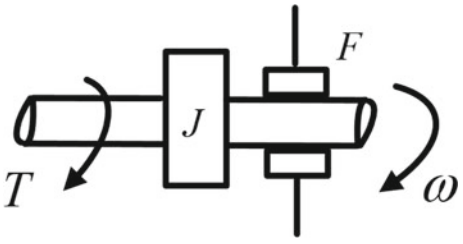


Fig. 2.11 Dynamics of rotating shaft with viscous friction

2.5 The Inverse Laplace Transform

The whole philosophy behind the LT is to convert to a different domain (the s-domain) where we can analyse the system algebraically. Once in the s-domain we can convolve by a signal input (and this becomes just multiplication in the s-domain as we have just seen). Then we return to the time-domain by taking the inverse LT. This is best illustrated with our first-order RC network. It has a Laplace TF which we can call G(s)

$$G(s) = \frac{1}{(1+sT)} \tag{2.36}$$

Let the input to this system be called r(t) and the output y(t). In Laplace form r(s) and y(s). If we apply a unit-step then we have a situation as shown in Fig. 2.12.

Multiplying we get that $y(s) = G(s)r(s)$ or

$$y(s) = \frac{1}{s(1+sT)} \tag{2.37}$$

The time-domain solution is the inverse LT which we write a shorthand as

$$\mathrm{y(t)} = \mathcal{L}^{-1}\{\mathrm{y(s)}\}$$

But if we examine the LT Table 2.1 we see there is no entry for this LT. This is quite common for most LTs and we get around the problem by using *partial-fractions*. This is an old method in algebra which is analogy to fractions in ordinary arithmetic. We write

$$\frac{1}{\mathrm{s(1+sT)}} = \frac{\mathrm{A}}{\mathrm{s}} + \frac{\mathrm{B}}{(1+\mathrm{sT})} \tag{2.38}$$

where A and B are unknown constants to be found. By multiplying both sides of (2.38) by

the denominator of the LHS of (2.38) we get

$$1 = \mathrm{A}(1+\mathrm{sT}) + \mathrm{Bs} \tag{2.39}$$

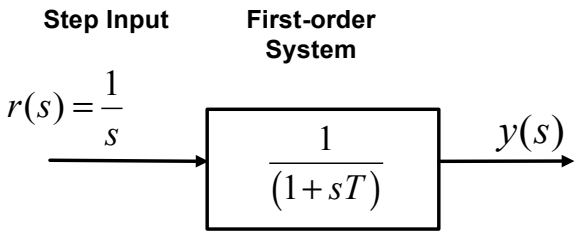


Fig. 2.12 Step input to a first-order system

Comparing powers of s^1 and s^0 gives us

$$0 = AT + B$$
$$1 = A$$

From which $B = -T$
Substitute back into (2.38) and we have

$$\frac{1}{\mathrm{s}(1+\mathrm{sT})} = \frac{1}{\mathrm{s}} - \frac{\mathrm{T}}{(1+\mathrm{sT})}$$
$$= \frac{1}{\mathrm{s}} - \frac{1}{\left(\mathrm{s}+\frac{1}{\mathrm{T}}\right)}$$

We use the templates in the LT table to find that $u(t) = 1 = \mathcal{L}^{-1}\left\{\frac{1}{s}\right\}, e^{-\frac{t}{T}} = \mathcal{L}^{-1}\left\{\frac{1}{s+\frac{1}{T}}\right\}$

and the solution becomes

$$\mathrm{y(t)} = 1 - \mathrm{e}^{-\frac{\mathrm{t}}{\mathrm{T}}} \tag{2.40}$$

Which is of course the same solution we obtained in Chap. 1 using the ODE approach but was much more complicated in finding the solution. We can summarise the use of partial fractions in the table below.

In Table 2.2 we have used K as the numerator though general polynomials also apply with the restriction that for any *physically-realisable system the order of the numerator polynomial must be no higher than the order of the denominator polynomial*. Hence a pure differentiator *s is not* physically realisable in an analogue (continuous-time) system. Neither is $\frac{(1+sT_1)^2}{(1+sT_2)}$ but when the orders are the same, for example $\frac{(1+sT_1)}{(1+sT_2)}$ or $\frac{(s^2+as+b)}{(s^2+cs+d)}$ they are. This is one reason why a pure differentiator can not exist. Any attempt to make one with an operational amplifier results in an

Table 2.2 Summary of commonly met partial fraction expansions

Combined LT	Partial fraction expansion
$\frac{K}{s(1+sT)}$	$\frac{A}{s} + \frac{B}{1+sT}$
$\frac{K}{(1+sT_1)(1+sT_2)}$	$\frac{A}{(1+sT_1)} + \frac{B}{(1+sT_1)}$
$\frac{K}{(1+sT_1)(1+sT_2)(1+sT_3)}$	$\frac{A}{(1+sT_1)} + \frac{B}{(1+sT_2)} + \frac{C}{(1+sT_3)}$
$\frac{K}{s^2(1+sT)}$	$\frac{A}{s} + \frac{B}{s^2} + \frac{C}{(1+sT)}$
$\frac{K}{s(1+sT)^2}$	$\frac{A}{s} + \frac{B}{(1+sT)} + \frac{C}{(1+sT)^2}$
$\frac{K}{s(s^2+as+b)}$	$\frac{A}{s} + \frac{Bs+C}{(s^2+as+b)}$

oscillating differentiator. This is due to the stability of the op-amp itself and can only be fixed by approximating differentiation over a limited frequency range. For example $\frac{s}{(1+sT)}$ is physically realisable and is an approximate differentiator over a limit frequency range. More details will be given in later chapters.

Chapter 3
Transfer Function Approach

The previous chapter introduced the Laplace Transform (LT) method as a substitute for using ordinary differential equations. It was shown that system can be cascaded together and their LT multiplied corresponding to time-domain convolution. The English Engineer Oliver Heaviside championed the LT or similar operator methods of his own. He concluded that he could substitute directly $p = \frac{d}{dt}$ (where nowadays we use the Laplace operator *s* directly) and convert easily from ODE to transfer function [17]. The mathematics was much debated at the time but is nowadays taken as being proven that Heavisides approach was mathematically correct. To this day Electrical Engineers use his approach routinely.

3.1 First and Second-Order Transfer-Functions

As an Electrical engineer or researcher, the first thing you notice about the LT method is that it can be applied directly to circuits and the ODE missed out entirely! For example, the complex Laplace operator is written as $s = \sigma + j\omega$ by convention where the imaginary part is frequency and the real part is decay-rate. So for capacitors, instead of their usual reactance written in complex form as $\frac{1}{j\omega C}$ we write $\frac{1}{sC}$ instead. Likewise for inductance we write sL instead of jωL. We then proceed in terms of s instead of jω and produce transfer functions directly from the circuit. This is well documented in almost any book on circuit theory. For example for the CR circuit (Fig. 3.1).

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_3

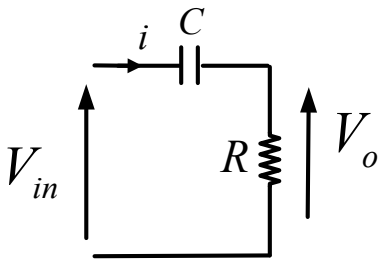


Fig. 3.1 CR circuit

Example 3.1 CR Circuit
The circuit is an impedance divider so we can say

$$V_o(s) = \frac{R}{R + 1/sC} V_{in}(s) \tag{3.1}$$

So that when simplified

$$V_o(s) = \frac{sCR}{1 + sCR} V_{in}(s) \tag{3.2}$$

By definition of the time-constant and transfer function $G(s) = \frac{V_o}{V_{in}}(s)$

$$G(s) = \frac{sT}{1 + sT} \tag{3.3}$$

We then draw it in block-diagram format.

Now we can use the method of Laplace to apply any kind of input (usually a step) and find the output. No differential equations are required at all. We note that this time there appears a numerator term unlike for the ordinary RC case. Applying a unit step input $V_{in}(s) = 1/s$

$$\begin{aligned} V_o(s) &= \frac{sT}{s(1 + sT)} \\ &= \frac{1}{s + \frac{1}{T}} \end{aligned} \tag{3.4a, b}$$

By using the table in Chap. 2 we find its inverse LT is

$$V_o(s) = \mathcal{L}^{-1}\left\{\frac{1}{(s + 1/T)}\right\} \tag{3.5}$$

$$V_o(t) = e^{-\frac{1}{T}} \tag{3.6}$$

a decaying exponential. The rate of decay depends on T, the times constant. By the same token we can let the real part of s be zero and write our transfer-function in terms of

$$G(j\omega) = \frac{j\omega T}{1 + j\omega T} \tag{3.7}$$

where angular frequency is $\omega = 2\pi f$ and f is frequency in Hz. As this is a ratio of complex numbers and nothing else (though dependent on frequency f), we can then find magnitude and phase and plot a frequency response. So the LT can lead us to both the transient and frequency domains depending on whether we take the full value of s or just its imaginary part.

The time-constant itself is related to what is termed the pole of the system. Poles are values of s where the magnitude is infinite. The reason why they are called poles is explained later.

Example 3.2 Sinusoidal Input Using LTs
Find the output of the system if the input is a sinewave of amplitude unity and frequency $\omega = 1$ rads/s.

Solution

$$u(t) = \sin(t)$$

From LT tables for a sinewave

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

Then from Fig. 3.2 the output of the system is

$$\begin{aligned} y(s) &= \frac{1}{(s^2 + 1)} u(s) \\ &= \frac{1}{(s^2 + 1)} \frac{1}{(s + 1)} \end{aligned}$$

Use partial fractions

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

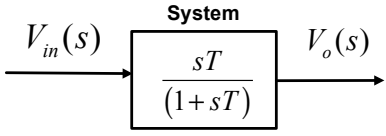


Fig. 3.2 Block-diagram representation of circuit

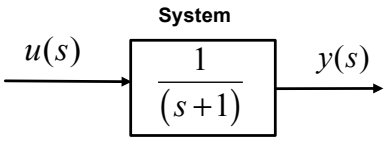


Fig. 3.3 First-order system with sinusoidal input

Multiply out

$$1 = \mathrm{A}\left(\mathrm{s}^2 + 1\right) + (\mathrm{Bs} + \mathrm{C})(\mathrm{s} + 1)$$

Compare coefficients and we get $A = 1/2$, $B = -1/2$ and $C = 1/2$. Substitute these values back and we get (Fig. 3.3)

$$\mathrm{y(s)} = \frac{1/2}{(\mathrm{s}+1)} + \frac{-0.5\mathrm{s} + 0.5}{(\mathrm{s}^2+1)}$$

$$\mathrm{y(s)} = \frac{1/2}{(\mathrm{s}+1)} + \frac{0.5}{(\mathrm{s}^2+1)} - \frac{0.5\mathrm{s}}{(\mathrm{s}^2+1)}$$

Find the inverse LT from tables gives us the time-response (Fig. 3.4)

$$\mathrm{y(t)} = 0.5(\mathrm{e}^{-\mathrm{t}} + \sin(\mathrm{t}) - \cos(\mathrm{t}))$$

We can see the effects of the exponential transient at the beginning and that in steady-state there is a sinusoidal output with a negative phase-shift and attenuated amplitude. This is what we expect from a LTI system. Usually the transient part has died out so quickly we cannot see it on an oscilloscope. We can also calculate the

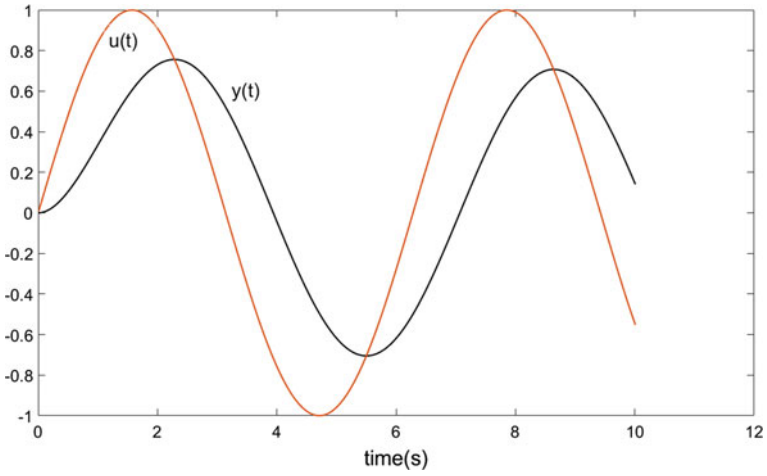


Fig. 3.4 Shows input and output of a LTI system

exact attenuation by substituting $\left|\frac{1}{(S+1)}\right|\Big|_{s=j\omega}$ where $|.|$ is magnitude of the complex expression. We get $\frac{1}{\sqrt{(1+\omega^2)}}\Big|_{\omega=1} = 0.707$. We can even calculate the amount of phase-shift by finding the phase at 1 rad/s. $\phi = -\tan^{-1} 1 = -45°$. The minus sign appears since the denominator gives negative phase-shift and the numerator has none. These quantities are not easily taken from an ODE and show the great advantage of using LTs.

3.2 Step-Response of Second-Order Systems

Like first-order systems, second-order systems are very common and used as a template for the mathematical modelling of many real systems. Our electrical RLC circuit of Fig. 1.20 is one of the most commonly used examples in the literature along with the mass-spring damper shown in Fig. 3.5 which is displaced by some distance $x(t)$

A free-body diagram is not really necessary here as the force balance is obvious from the diagram. We have the pulling force as $f(t)$ on the mass M whilst two other forces pull in the other direction. The two other forces are the force due to the spring $kx(t)$ and $B\frac{dx(t)}{dt}$ due to the dashpot. The spring force equation is due to Hooke's Law where k is the spring constant which varies from spring to spring. A dashpot is a mechanical device or at least models similar looking devices that provides less vibration, like the shock-absorber in a car. The ODE is second-order and becomes by force-balance and Newton's Law

$$f(t) = M\frac{d^2x(t)}{dt^2} + B\frac{dx(t)}{dt} + kx(t) \tag{3.8}$$

Taking LTs of the above

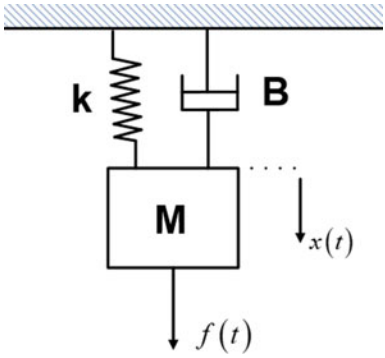


Fig. 3.5 Mass-spring damper 2nd order system

$$f(s) = Ms^2x(s) + Bx(s) + kx(s)$$

From which we obtain a transfer-function

$$\frac{x}{f}(s) = \frac{1}{Ms^2 + Bx + k} \tag{3.9}$$

The electrical RLC circuit and this mechanical one are perfect analogies of each other in different disciplines. It is important to understand the fundamental equations of this type of system. Quite how the response of such a system behaves depends on the individual mass-spring damper terms, just as with the individual resistive, capacitive and inductor terms. We use a general *generic* template for second-order systems, just as we use $G(s)\frac{1}{1+sT}$ for first-order. The template we use is that any second-order system can be written in the form

$$G(s) = \frac{y}{u}(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{3.10}$$

This is illustrated in Fig. 3.6.

Even though the numerator may not be exactly in the ideal form as shown in (3.10) the analysis that follows is still of great importance. The individual terms are usually named as ω_n, the un-damped natural frequency (sometimes just the natural frequency) in rads/s, ζ is dimensionless and is known as the damping factor (sometimes the *damping ratio*). There are quite a few cases to consider for the unit step-response when $u(s) = \frac{1}{s}$.

3.2.1 *Case A: No Damping* $\zeta = 0$

From Fig. 3.3 if the dashpot did not appear at all and there was no friction in the air, then the mass would oscillate up and down for a very long time. In Physics this is known as simple harmonic motion.

$$y(s) = \frac{\omega_n^2}{s(s^2 + \omega_n^2)} \tag{3.11}$$

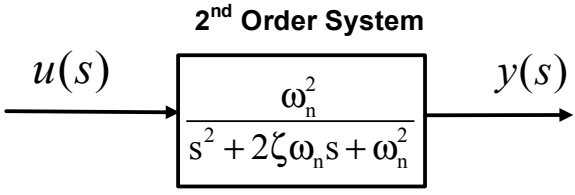


Fig. 3.6 Generic second-order system transfer-function

Expanding using partial fractions

$$\frac{\omega_n^2}{s\left(s^2+\omega_n^2\right)} = \frac{A}{s} + \frac{Bs+C}{\left(s^2+\omega_n^2\right)} \tag{3.12}$$

From which we find
A = 1, B = −1, C = 0

$$y(s) = \frac{1}{s} - \frac{s}{\left(s^2+\omega_n^2\right)}$$

$$y(t) = \mathcal{L}^{-1}\left\{\frac{1}{s} - \frac{s}{\left(s^2+\omega_n^2\right)}\right\}$$

$$y(t) = 1 - \cos(\omega_n t) \tag{3.13}$$

which is a pure oscillation.

3.2.2 *Case B: Critical Damping* $\zeta = 1$

We examine the roots of $s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$ when $\zeta = 1$ and find that there are two repeated roots at $s = -\omega_n$. Write (3.10) as

$$y(s) = \frac{1}{s(s+\omega_n)^2} \tag{3.14}$$

Use partial fractions

$$y(s) = \frac{A}{s} + \frac{B}{(s+\omega_n)} + \frac{C}{(s+\omega_n)^2}$$

This reduces to

$$y(s) = \frac{1}{s} - \frac{1}{(s+\omega_n)} - \frac{\omega_n}{(s+\omega_n)^2} \tag{3.15}$$

Take inverse Laplace Transforms

$$y(t) = 1 - e^{-\omega_n t}(1 + \omega_n t) \tag{3.16}$$

This response settles down to unity (the input) when t gets large enough.

3.2.3 Case C: Overdamped Case $\zeta > 1$

Factorising $s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$ we find two distinct real roots. In fact this case degenerates into two first-order systems in cascade. The roots are $r_1, r_2 = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}$. Equation (3.10) becomes

$$y(s) = \frac{\omega_n^2}{s\left(s + \zeta\omega_n + \omega_n\sqrt{\zeta^2 - 1}\right)\left(s + \zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1}\right)}$$

Partial fraction expansion gives

$$y(s) = \frac{A}{s} + \frac{B}{\left(s + \zeta\omega_n + \omega_n\sqrt{\zeta^2 - 1}\right)} + \frac{C}{\left(s + \zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1}\right)} \tag{3.17}$$

After a little algebra to find the coefficients A, B, C and inverse Laplace Transforming

$$y(t) = 1 + \frac{\omega_n}{2\sqrt{\varsigma^2 - 1}}\left[\frac{e^{-r_1 t}}{r_1} - \frac{e^{-r_2 t}}{r_2}\right] \tag{3.18}$$

When t gets large the steady-state solution converges to unity. Note that (3.18) is only true for $\zeta > 1$ and the square-root can never be complex. Thought of from another point of view we can merely factorise into two real roots and this gives us two first-order systems in cascade (Fig. 3.7).

3.2.4 Case D: Underdamped Case $\zeta < 1$

The final case is perhaps the most complex and interesting. We factorise the roots of the quadratic $s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$ when $\varsigma < 1$. This we can write as $r_1, r_2 = -\zeta\omega_n \pm j\omega_n\sqrt{1 - \zeta^2}$. For simplicity we define some new symbols. $\omega_d = \omega_n\sqrt{1 - \zeta^2}$ is the *damped* natural frequency. It is clearly smaller than the undamped case. The difference becomes more pronounced when the damping factor is closer to unity. Also define $\sigma = -\zeta\omega_n$ as the *decay-rate*. So the two complex

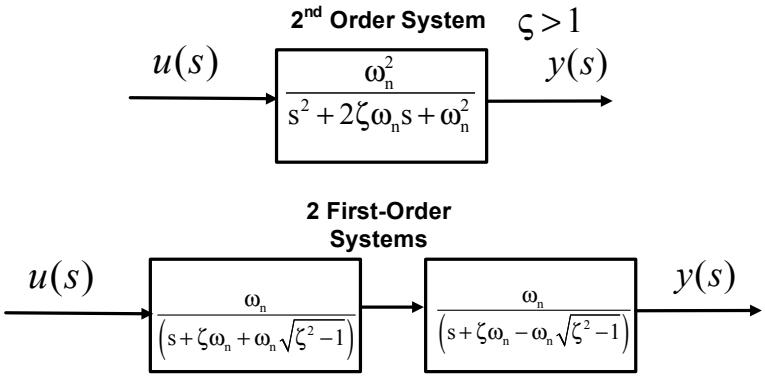


Fig. 3.7 Overdamped 2nd order system is equivalent to two first-orders in cascade

roots are $r_1, r_2 = \sigma \pm j\omega_d$. We then write by completing the square of the denominator

$$y(s) = \frac{\omega_n^2}{s\left[(s+\zeta\omega_n)^2 + \omega_d^2\right]} \tag{3.19}$$

Enable a partial fraction expansion

$$y(s) = \frac{A}{s} + \frac{Bs + C}{\left((s+\zeta\omega_n)^2 + \omega_d^2\right)} \tag{3.20}$$

where we can find $A = 1, B = 1, C = -2\zeta\omega_n$. Substituting into (3.20)

$$\begin{aligned} y(s) &= \frac{1}{s} - \frac{(s+\zeta\omega_n) + \zeta\omega_n}{\left((s+\zeta\omega_n)^2 + \omega_d^2\right)} \\ t &= \frac{1}{s} - \frac{(s+\zeta\omega_n)}{\left((s+\zeta\omega_n)^2 + \omega_d^2\right)} - \frac{\zeta\omega_n}{\left((s+\zeta\omega_n)^2 + \omega_d^2\right)} \\ &\frac{1}{s} - \frac{(s+\zeta\omega_n)}{\left((s+\zeta\omega_n)^2 + \omega_d^2\right)} - \frac{\zeta}{\sqrt{1-\zeta^2}} \frac{\omega_d}{\left((s+\zeta\omega_n)^2 + \omega_d^2\right)} \end{aligned} \tag{3.21a, b, c}$$

Inverse Laplace transforming using the table gives us

$$y(t) = 1 - e^{-\zeta\omega_n t}\cos(\omega_d t) - \frac{\zeta}{\sqrt{1-\zeta^2}} e^{-\zeta\omega_n t}\sin(\omega_d t) \tag{3.22}$$

Since decay-rate $\sigma = -\zeta\omega_n$

$$y(t) = 1 - e^{-\sigma t}\left[\cos(\omega_d t) + \frac{\zeta}{\sqrt{1-\zeta^2}}\sin(\omega_d t)\right] \tag{3.23}$$

These terms are classic damped sinusoid terms that decay at a rate determined by σ and oscillate at a frequency ω_d rad/s.

A useful result to use here to simplify (3.23) is that an equation of the form

$$\cos(\omega_d t) + \frac{\zeta}{\sqrt{1-\zeta^2}}\sin(\omega_d t)$$

Is more generally written as

$$A\cos(x) + B\sin(x) = \sqrt{A^2 + B^2}\sin(x + \phi)$$
$$\phi = \tan^{-1}\left(\frac{A}{B}\right) = \cos^{-1}\left(\frac{B}{\sqrt{A^2 + B^2}}\right)$$

Giving us the final solution

$$y(t) = 1 - \frac{1}{\sqrt{1-\zeta^2}}e^{-\sigma t}\sin\left(\omega_d t + \tan^{-1}\left(\frac{\sqrt{1-\zeta^2}}{\zeta}\right)\right)$$
$$1 - \frac{1}{\sqrt{1-\zeta^2}}e^{-\sigma t}\sin(\omega_d t + \cos^{-1}\zeta) \tag{3.24a, b}$$

We also see this equation in a slightly different form because

$$A\cos(x) + B\sin(x) = \sqrt{A^2 + B^2}\cos(x - \phi)$$
$$\phi = \tan^{-1}\left(\frac{B}{A}\right)$$

Resulting in

$$y(t) = 1 - \frac{1}{\sqrt{1-\zeta^2}}e^{-\sigma t}\cos\left(\omega_d t - \tan^{-1}\left(\frac{\zeta}{\sqrt{1-\zeta^2}}\right)\right) \tag{3.25}$$

A few useful formulae arise from this equation for the underdamped case [18]. Some are exact and some approximate.

Settling time

$$T_s \approx \frac{4}{\zeta\omega_n} \tag{3.26}$$

Time to peak

$$T_p = \frac{\pi}{\omega_d} \tag{3.27}$$

Maximum overshoot

$$M_p = e^{-\frac{\pi\zeta}{\sqrt{1-\zeta^2}}} \tag{3.28}$$

And finally the *rise-time*

$$T_r \approx \frac{\pi}{2\omega_n} \tag{3.29}$$

We can now plot a family of step-response curves for various damping factors that we have just discussed in the text. For simplicity we normalise $\omega_n = 1$ in Fig. 3.8.

We can now return to our two examples in this book. First the RLC circuit from Fig. 1.20 (Fig. 3.9).

This is an impedance divider with transfer function

$$\frac{V_o}{V_{in}}(s) = \frac{1}{s^2LC + CRs + 1} \tag{3.30}$$

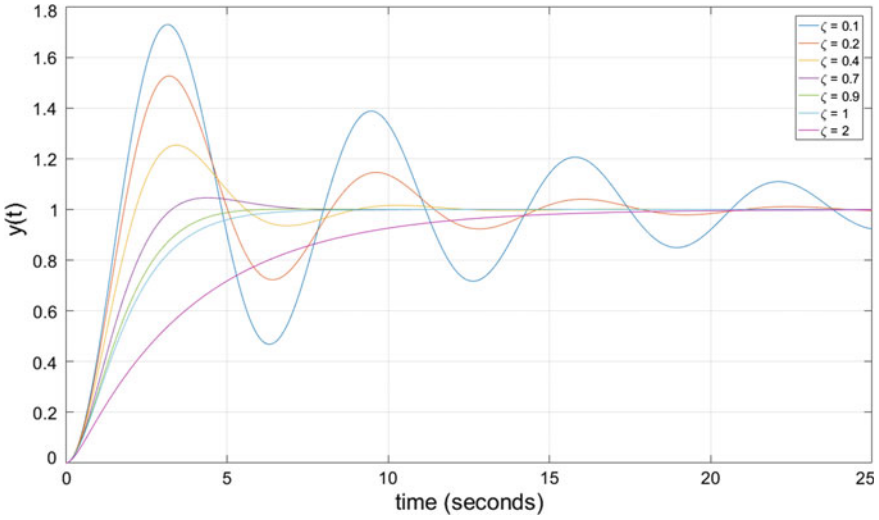


Fig. 3.8 Family of step-responses for different damping factors

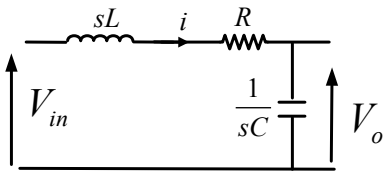


Fig. 3.9 RLC circuit

Which we write as

$$\frac{V_o}{V_{in}}(s) = \frac{1/LC}{s^2 + (R/L)s + (1/LC)} \tag{3.31}$$

Compare coefficients with the standard second-order template

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{3.32}$$

We get

$$\omega_n = \frac{1}{\sqrt{LC}}, 2\zeta\omega_n = \frac{R}{L}$$

Or

$$\zeta = \frac{R}{2}\sqrt{\frac{C}{L}}$$

Putting in some values. Suppose $R = 200\,\Omega, C = 1\,\text{nF}, L = 1\,\text{mH}$.

Then $\zeta = 0.1, \omega_n = 10^6\,\text{rad/s}$. Examining the standard template in Fig. 3.8 we would expect for this damping factor to get a 70% overshoot from a unit step. (value goes to 1.7 peak). We can calculate the transfer function for these values

$$G(s) = \frac{10^{12}}{s^2 + 2 \times 10^5 s + 10^{12}}$$

MATLAB can calculate the step-response for any such case quite easily by using the step() command.. A transfer-function must be defined as a ratio of polynomials in descending powers of *s*. So we can enter into MATLAB the following script for numerator and denominator polynomials.

MATLAB Code 3.1

```
num=1e12;

den=[1 2e5 1e12];

step(num,den)
```

We obtain the plot shown in Fig. 3.10.

This matches the family of step-responses for Fig. 3.8 when $\zeta = 0.1$. The undamped natural frequency $\omega_d = \sqrt{1-\zeta^2} = 0.99\,\text{rad/s}$ which is typical of systems with low damping factor i.e. the damped natural frequency is very close to the undamped one. Of course the damped natural frequency is different but the import thing is the indication of how much overshoot. Of course, while of interest theoretically, such a family of step-responses is less important nowadays then it once was due to the availability of powerful simulation software such as MATLAB. In earlier years these templates were used to get an indication without the use of simulation.

The spring-mass damper will also behave in some manner identical to the step-response family depending on the values. For it we have its transfer-function from (3.9)

$$G(s) = \left(\frac{1}{k}\right)\frac{k/M}{s^2 + (B/M)x + (k/M)} \tag{3.33}$$

In this case the overall transfer-function is scaled by $\left(\frac{1}{k}\right)$ to make it fit the standard model. This only changes the value of steady-state response and not the type of response or overshoot. By comparison we see $\omega_n = \sqrt{(k/M)}, \zeta = \frac{B}{2\sqrt{Mk}}$.

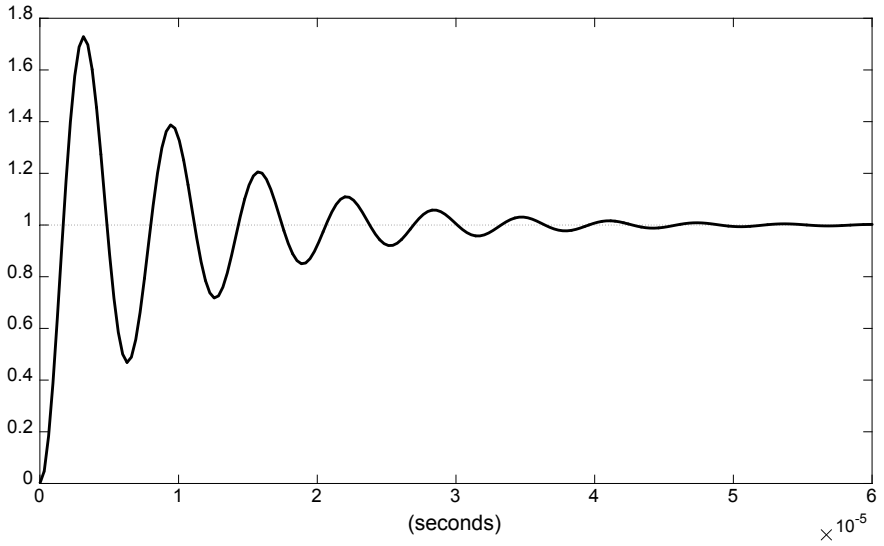


Fig. 3.10 Step-response of RLC circuit

3.3 Poles and Zeros

Having just looked at first and second-order systems we see that their order is defined by the largest power of s in the denominator. For LTI systems we can say that any system can be made up with a combination of cascading consecutive first or second-order transfer-functions. So a 3rd order system could be made up of three first-orders or one first-order and a system with complex roots for example. Likewise a 4th order could be 4 first-orders or perhaps two second-order systems with complex roots or two first and one second. As mentioned previously, we term the roots of the denominator of a transfer-function its *poles*. Likewise if there is a polynomial in the numerator of a transfer-function we call its roots *zeros*. To illustrate this consider the simple filter circuit in Fig. 3.11.

We find its transfer function by voltage division of impedances. The parallel RC network becomes

$$z_1(s) = \frac{R_1}{1 + sCR_1} \tag{3.34}$$

The voltage at the output becomes

$$\begin{aligned} V_o(s) &= \frac{R_2}{R_2 + \frac{R_1}{1+sCR_1}} V_{in}(s) \\ &= \frac{R_2(1+sCR_1)}{R_2(1+sCR_1) + R_1} V_{in}(s) \end{aligned} \tag{3.35a, b}$$

$$V_o(s) = \left(\frac{R_2}{R_1 + R_2}\right) \frac{(1+sCR_1)}{(1+sCR_1//R_2)} V_{in}(s) \tag{3.36}$$

Finally we have

$$\frac{V_o}{V_{in}}(s) = g\frac{(1+sT_1)}{(1+sT_2)} \tag{3.37}$$

$$g = \left(\frac{R_2}{R_1 + R_2}\right), T_1 = CR_1, T_2 = CR_1//R_2$$

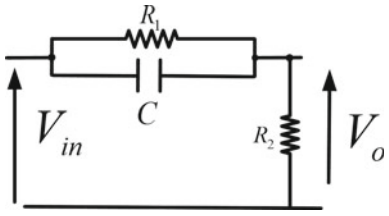


Fig. 3.11 Simple filter circuit

Examining (3.37) we see it has one pole and one zero. The zero is the solution of $1 + sT_1 = 0$ from which $s = -\frac{1}{T_1}$. The pole is the solution of $1 + sT_2 = 0$ from which $s = -\frac{1}{T_2}$. By the nature of the problem we see that $T_1 > T_2$. So a zero looks self-evident that it is the value of $s = \sigma + j\omega$ (complex frequency) that sets the output to zero. But why are poles so-called?

Consider a second-order system with complex poles at $p_1, p_2 = -0.5 \pm j0.5$

$$G(s) = \frac{1}{(s^2 + s + 0.5)} \tag{3.38}$$

Now we plot a 3D graph of $|G(\sigma \pm j\omega)|$ the magnitude of the complex expression (3.38) as the z-axis as $\sigma \pm j\omega$ varies and make the x and y axis the real and imaginary parts of $s = \sigma + j\omega$.

We see that the magnitude goes to infinity at the poles of the system. They look like poles sticking upwards. Likewise zeros are like holes in the surface. When you look in 3D the reason is obvious. However, we rarely use 3D plots and instead look vertically downwards towards the surface and term this the *s-plane*. Poles are marked with a cross X and zeros with an O. The s-Plane plot is shown in Fig. 3.13.

This is just an aerial view of Fig. 3.12.

3.4 Stability of Transfer-Functions

From Eq. (3.25) we have already seen that a second-order system rises up to a constant value for a step input. We also can see that the decay-rate is given by the exponential of the real part of the poles $p_1, p_2 = -\zeta\omega_n \pm j\omega_d$. In fact, for a complex pole say $\frac{1}{s + \sigma + j\omega}$ its inverse Laplace Transform is $\mathcal{L}^{-1}\left\{\frac{1}{s + \sigma + j\omega}\right\} = e^{-\sigma t}e^{-j\omega}$. This is a decaying sine-wave.

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

and provided the real part of the pole is negative the output must settle down and not rise up indefinitely. For example $e^{-2t}e^{-j\omega}$ dies out exponentially whereas $e^{2t}e^{-j\omega}$ grows exponentially. If the imaginary part is zero then we have a first-order system and it either dies out exponentially for a negative root or grows exponentially for a

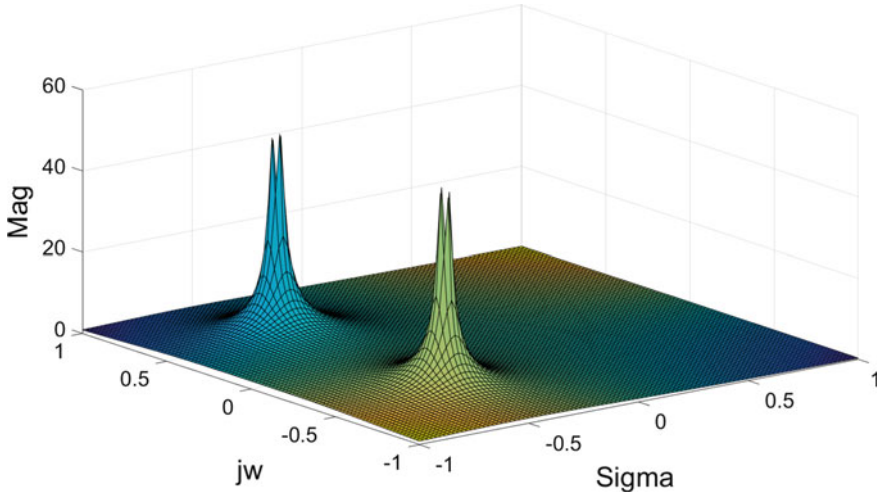


Fig. 3.12 3D plot showing the poles of the system

positive root. The condition for stability of a LTI system is that *its poles all must have negative real parts*. Often this is stated more easily by saying that the *poles must all lie in the left-hand the s-plane* (Figs. 3.13 and 3.14).

It is important to state that zeros have no effect on stability of a transfer-function. The do have a significant effect when negative-feedback is applied but just on open-loop the zeros are benign (Fig. 3.15).

Zeros do of course shape the type of response we get, but do not de-stabilise a system when in open-loop. Depending on the position of the poles will give rise to differing step-responses.

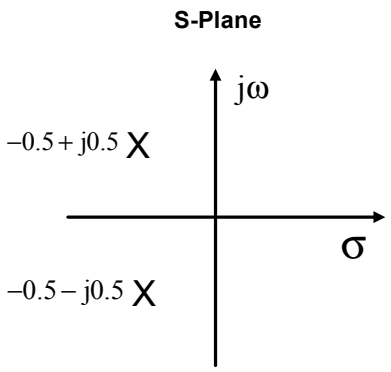


Fig. 3.13 s-plane plot of complex poles

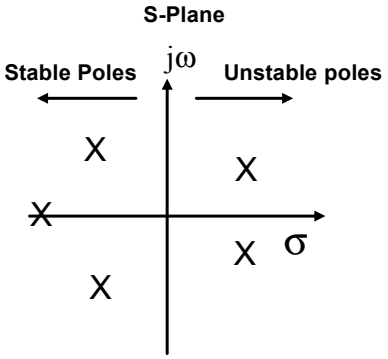


Fig. 3.14 Stable part of the s-plane

It can be seen that when conjugate pole-pairs are exactly on the imaginary axis that sustained oscillations occur. This is often termed critical stability or critically stable systems. They are just on the verge of bursting into instability. Designing systems that gave such outputs with little distortion was a lucrative business in the late 20th century before direct-digital synthesis of waveforms could be done reliably. In fact, in July 1939 a patent was filed by W. R. Hewlett for a valve oscillator that used feedback around it and an incandescent lamp to keep it from going unstable. As the poles move farther to the left of the plane the response speeds up for real poles and for complex ones the decay-rate increases so the oscillations damped quicker. We can make use of this by approximating high-order systems to second order in some cases at least. For example, the pole positions shown in Fig. 3.16 show a real pole which is let us say ten-times further away than the complex poles. We say the system has *dominan*t second-order poles. The real pole step-response will die out quickly and have little effect on us approximating the system to be second order (Fig. 3.17).

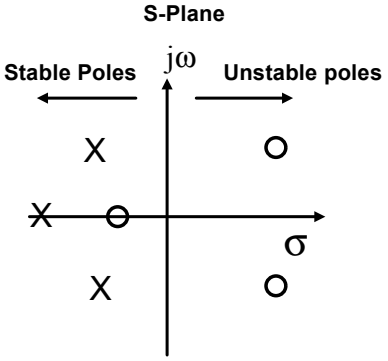


Fig. 3.15 Right-half plane zeros. System is still stable since poles lie in the left-half plane

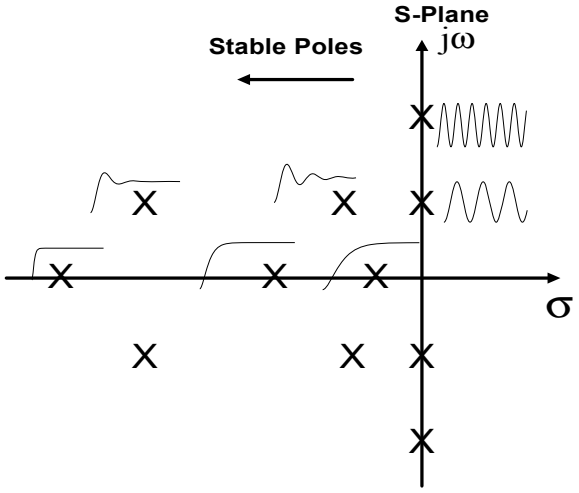


Fig. 3.16 Step-responses for different pole positions in left-hand plane

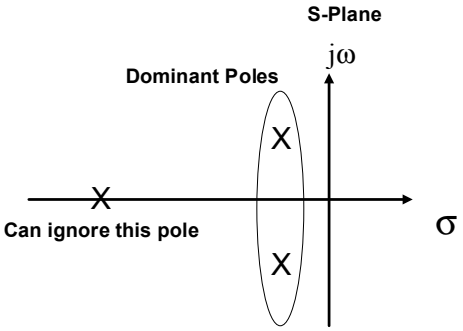


Fig. 3.17 Third-order poles but an approximate second-order system

3.5 The Final-Value Theorem

The final-value theorem (FVT) has its uses in control-system analysis. It can tell us what the steady-state value of the output of a system is for a given input without having to work out the tedious algebra or even simulate it. The theorem says that

$$\lim f(t)_{t\to\infty} = \lim sF(s)_{s\to 0} \tag{3.39}$$

In other words, the steady-state value of the output of a signal in the time-domain $f(\infty)$ can be found by taking its Laplace Transform F(s), multiplying by s and putting s to zero. A simple example will illustrate this. Take a first-order system

$$G(s) = \frac{100}{s+5}$$

Find the steady-state output of this system for a unit step input using the FVT. You will see that that systems are not always written for our convenience in the form $\frac{1}{1+sT}$ and some have extra added gain on the numerator. We could rearrange this into the more familiar form and often we do, but for the FVT there is no need. First we write

$$y(s) = \frac{100}{s+5}u(s)$$

For a unit step input $u(s) = \frac{1}{S}$. This gives us $y(s) = \frac{100}{S(S+5)}$. Now using the FVT (3.39), we multiply by s and this gives us $sy(s) = sX\frac{100}{S(S+5)}$. The single s term cancels and we then follow the FVT by putting *s = 0* in what is left. This gives a steady-state solution 100/5 = 20. The FVT is only valid for stable systems. For critically stable (oscillators!) or unstable systems we already know what sort of outputs we get so the results are invalid unless the system has left-hand plane poles.

Consider a third-order system. $G(s) = \frac{100}{(S+1)\left(S^2+S+0.5\right)}$. Let us use MATLAB to first find the step-response and then check the steady-state value with the FVT. There are a number of ways to enter the system into MATLAB. We actually have two systems in cascade, a first-order and a second-order with complex poles. Both systems are stable (Fig. 3.18)

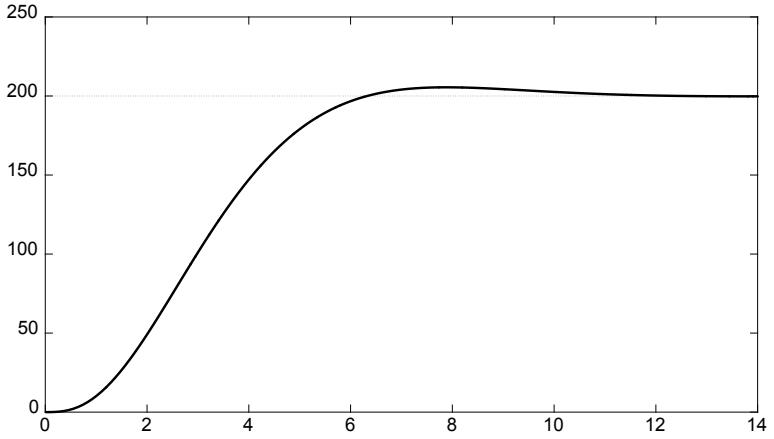


Fig. 3.18 Step-response of third-order system using MATLAB

MATLAB Code 3.2

```
num=100;
d1=[1  1];
d2=[1 1 0.5];
% Multiply the two denominator polynomials using convolution
%den=d1*d2 (convolution)
den=conv(d1,d2);
g=tf(num,den)
% creates the system g =num/den
g =

          100

  -------------------------

  s^3 + 2 s^2 + 1.5 s + 0.5

step(g)
```

MATLAB shows the steady-state value as 200. Using the FVT we have $y(s) = \frac{100}{S(S+1)\left(S^2+S+0.5\right)}$. Multiply by s and let s be zero gives us $sy(s) = \frac{100s}{S(S+1)\left(S^2+S+0.5\right)}\Big|_{s=0} = 100/(1 \times 0.5) = 200.$

3.6 A Note on the Routh Stability Criterion

As discussed in the introduction chapter, it has been known since the 19th century that the roots of a polynomial define the stability of a linear-dynamic system. Maxwell made this discovery in this paper on Governors for steam-engines and Routh found a general theory for any order of polynomial. There were no computers in those times and so finding the actual roots would have been a laborious exercise. Nowadays our technology has leaped us ahead of the Routh method since we can solve for the roots of almost any order polynomial we desire using software. The method is so elegant however that it is worth mentioning and has its uses for certain problems where a range of values for stability is needed in terms of a

parameter. Like many great discoveries the Routh method was discovered independently by another researcher, in this case Hurwitz. The method is nowadays known as the Routh Hurwitz method. We will not look at all the special cases but only the basic principle.

Consider a polynomial of order n.

$$a_n s^n + a_{n-1} s^{n-1} + a_{n-2} s^{n-2} \ldots a_1 s + a_0 = 0 \tag{3.40}$$

We build a special table known as the Routh table as follows

Row 1 s^n	a_n	a_{n-2}	a_{n-4} …
Row 2 s^{n-1}	a_{n-1}	a_{n-3}	a_{n-5} …

Essentially for the first row we are just writing down the coefficients of the polynomial starting at the highest order and missing the next value and so on until we have used up all the values. The second row consists of the values we missed in the first.

We then complete the rows as follows until the first column is completed.

Row 1 s^n	a_n	a_{n-2} …	a_0
Row 2 s^{n-1}	a_{n-1}	a_{n-3} …	0
Row 3 s^{n-2}	b_{n-1}	b_{n-3} …	0
Row 4 s^{n-3}	c_{n-1}	c_{n-3} …	0

and so on until all powers of *s* are used up as far as s^0. The entries in the first column are calculated according to

$$b_{n-1} = \frac{a_{n-1}a_{n-2} - a_n a_{n-3}}{a_{n-1}},\ b_{n-3} = \frac{a_{n-1}a_{n-4} - a_n a_{n-5}}{a_{n-1}},$$
$$c_{n-1} = \frac{a_{n-3}b_{n-1} - a_{n-1}b_{n-3}}{b_{n-1}}$$

This is illustrated with an example. Start with an easy one that we know has stable roots.

$$s^2 + 3s + 2 = 0$$

Row $1s^2$	1	2	0
Row $2s^1$	3	0	0
Row $3s^0$	$(3 \times 2 - 1 \times 0)/3 = 2$	0	0

The last row is calculated a bit like finding the determinant of a 2×2 matrix but in the wrong order! Multiply the 3 by 2 and subtract 3 times 0 and divide by 3. The

method is repeated until all powers of s are processed. So for a second-order system we only need one more row. Note we always divide by the number in the cell above the row we are evaluating. If that number is zero then we have a special way of dealing with this case. To find if the system is stable, we look only for sign changes in the *first column*. Since the numbers are 1, 3 and 2 there are no sign changes so we conclude the system has all poles in the left-hand plane. Note that we cannot say where those poles are, only that they are stable.

Now consider an unstable second-order system

$$s^2 - 3s + 2 = 0$$

Repeat the above procedure

Row $1s^2$	1	2	0
Row $2s^1$	−3	0	0
Row $3s^0$	$(-3 \times 2 - 1 \times 0)/3 = 2$	0	0

Examine the first column and see that the numbers go from 1, −3 to 2 so there are two sign changes and therefore two roots in the right half plane.

Now a third order example with two stable complex roots and one unstable real root.

$$s^3 - s^2 - 1.2s - 1.6 = 0$$

Row $1s^3$	1	−1.2	0
Row $2s^2$	−1	−1.6	0
Row $3s^1$	$(-1 \times -1.2 - 1 \times -1.6)/-13 = -2.8$	0	0
Row $4s^0$	$(-2.8 \times -1.6 - 0)/-2.8 = -1.6$	0	0

Examining the first column we have 1, −1, −2.8, −1.6 so there is only one sign change and therefore one root in the right half plane.

Occasionally a zero appear in the first column and this results in a division by zero in the next! This is avoided by replacing the zero with a small number say ε, a small positive constant and letting it tend to zero as shown in this example which has an unstable root at 1 and two stable complex roots.

$$s^3 + 2s^2 + s + 2 = 0$$

Row $1s^3$	1	1	0
Row $2s^2$	2	2	0
Row $3s^1$	$(2-2)/2 = 0$ make ε instead	0	0
Row $4s^0$	$(2\varepsilon - 0)/\varepsilon \to 2$	0	0

The first column elements are 1, 2, ε and 2. Hence no sign changes and this is considered stable or at least no roots in the right half plane. In actual fact such problems arise when there are roots on the imaginary axis and the roots are solutions of the row above the zero entry, namely

$$2s^2 + 2 = 0$$

$$s = \pm j$$

Where the Routh method can be useful over software is when we need an upper limit on say a gain term or parameter. For example, find the value of positive K which makes the roots of this polynomial unstable and at that limiting value what are the imaginary roots.

$$s^3 + 6s^2 + 11s + 6 + K = 0$$

Row s^3	1	11	0
Row $2s^2$	6	$6+K$	0
Row $3s^1$	$(60-K)/6$	0	0
Row $4s^0$	$6+K$	0	0

For there to be no sign change in the first column, we require that all entries are positive. The first two entries are 1 and 6 and the third must satisfy $(60 - K)/6 > 0$ so that $K < 60$. The last entry requires $6 + K > 0$ or $K > -6$. So the range of values for K which keep the roots in the left half plane are $-6 < K < 60$. If K is kept positive then we can reduce this to $0 < K < 60$. Of course, this could be calculated directly by trial and error by continually solving for the roots of the equation. This seems to be a more refined approach. We also see that at the limit when K = 60 the third row is zero indicating roots on the imaginary axis at (by reading row two)

$6s^2 + (6 + 60) = 0$. Hence the roots are at $s = \pm j\sqrt{11}$.

There are many other methods to determine stability, but these are for closed-loop systems rather than open loop. These we examine in the next chapter.

Chapter 4
Speed and Position-Control Systems

The previous chapters have laid down a firm foundation for linear time-invariant systems and categorised them into first-order, second-order or cascaded combinations. The introduction mentioned some desirable properties of negative feedback around amplifiers, but so far very little has been said about more complex systems. This chapter will examine the rationale behind using negative feedback and its many advantages by using the dc motor as the example for speed and position-control. We will also introduce the idea of block-diagram algebra, but as we need it instead of just a list of rules which traditionally appear in many books on the subject.

4.1 The Need for Feedback

The obvious question that it is often asked, is why is feedback necessary at all? For example, the small aluminium electric cart shown in Fig. 4.1 is intended for moving heavy objects of up to 24 kg with little effort. It uses two dc motors driven by pulse-width modulation (PWM) and has a microprocessor-controlled joystick so it can be steered in any direction.

The motors turn roughly at the same speed when moving forwards and the speed is differentially changed for steering. Both the motors are run in open-loop since there is no great need for precision speed control. Should one motor be slightly ahead of the other in speed then the operator can adjust the joystick to compensate. In actual fact, the only feedback is human based. However, were such a vehicle to be moved by a computer algorithm then there would be some need for a feedback system, at least for steering. A heading angle would need to be set and measurements made that the cart steers to the desired heading. There are many such examples of motors in particular that can be run without any form of feedback for the type of applications they are employed to do. A pump for fluids may not need any form of precision speed control and numerically-controlled machines usually use large stepper motors for position control.

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_4

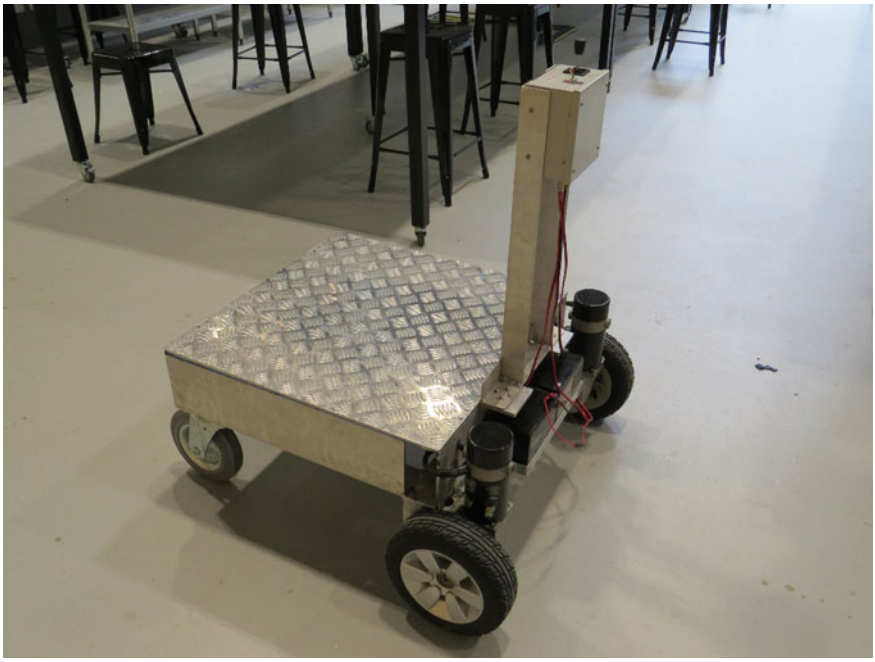


Fig. 4.1 Electric cart with joystick control

Figure 4.2 shows a two-wheeled cart with a 1 metre inverted pendulum on top. Such a system is inherently unstable. Without negative feedback the cart will fall over. Systems based on such designs are becoming more common for personal transport. There are many systems which open loop are unstable and much be stabilised dynamically. These include machines that walk or balance on two feet, rockets and fighter aircraft. Fighter aircraft, unlike their passenger equivalent are inherently unstable yet via feedback are stabilised. The advantage of doing so is that manoeuvrability of the aircraft is much better. Consider the generic approach to negative feedback with the following block-diagram (Fig. 4.3).

The signals and systems are all defined in terms of Laplace Transforms. The open-loop system is $G(s)$, sensor dynamics $H(s)$, output $y(s)$ and input $u(s)$. The error signal is defined as $e(s)$ and a disturbance is shown as $d(s)$. This block diagram covers a great many problems. A good understanding of it is necessary to proceed any further. The input $u(t)$ or in Laplace form $u(s)$ is the desired output or *setpoint*. The output $y(s)$ or $y(t)$ in the time-domain must follow the input as closely as follows. Simple algebraic analysis yields

$$e(s) = u(s) - H(s)y(s) \tag{4.1}$$

$$y(s) = d(s) + G(s)e(s) \tag{4.2}$$

Substitute the error (4.1) into (4.2)

Fig. 4.2 Inverted pendulum on wheels

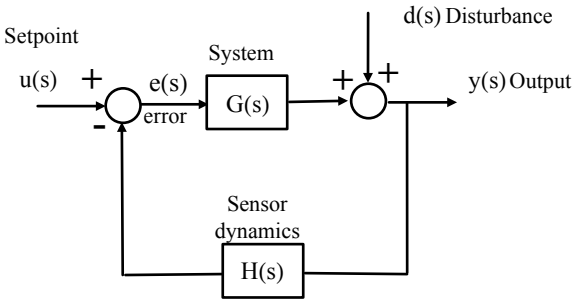


Fig. 4.3 Generic feedback problem

$$y(s) = d(s) + G(s)[u(s) - H(s)y(s)] \tag{4.3}$$

$$[1 + G(s)H(s)]y(s) = d(s) + G(s)u(s) \tag{4.4}$$

Solving for the output signal

$$y(s) = \frac{1}{[1 + G(s)H(s)]}d(s) + \frac{G(s)}{[1 + G(s)H(s)]}u(s) \tag{4.5}$$

First we consider the disturbance-free case $d(s) = 0$. We get the standard *closed-loop* system.

$$y(s) = \frac{G(s)}{[1 + G(s)H(s)]}u(s) \tag{4.6}$$

Now both transfer-functions are frequency-related and can be written as $G(j\omega)$, $H(j\omega)$. If we can ensure that the gain of the system is high enough then $|G(j\omega)| \gg 1$. Usually we cannot make this high enough at both low and high frequencies and the gain of the system gets smaller as frequency increases. Of course it may be that the system does not have this characteristic. Then we must introduce an extra system (the controller) to artificially ensure the gain is high. More of this later. For high gain we get $|G(s)H(s)| \gg 1$ and hence in (4.6) $|1 + G(s)H(s)| \approx G(s)H(s)$ and we get $y(s) = \frac{1}{H(s)}u(s)$. We see that one of the properties of feedback with high gain that we get the reciprocal of the feedback path as the closed-loop system. For many problems this is just a scaling factor. For example, when measuring angular velocity in rad/s or RPM of a motor, the speed signal is found from an analogue or digital tachometer and the actual speed reading is just a scaled version of the real speed. Similarly for angular shaft position of a motor the reading from the sensor is just a scaled version of the actual angular displacement in radians. If the feedback path has unity negative-feedback, $H(s) = 1$ and $y(s) = u(s)$. Often we can assume unity negative feedback by re-defining the output of the system to be after the sensor rather than before.

Whether or not the open-loop system is stable does not matter. What is important is that the *closed-loop* system is stable and we must have that the poles of the *closed-loop characteristic* equation

$$1 + G(s)H(s) = 0 \tag{4.7}$$

lie in the left-hand of the s-plane as shown in the previous chapter.

When the disturbance is not zero, by superposition (since this is an LTI system) we can set the input to zero and get

$$y(s) = \frac{1}{[1 + G(s)H(s)]} d(s) \tag{4.8}$$

For the same assumption of high enough gain, the denominator swamps the numerator and we get that $y(s)$ goes to zero hence rejecting the disturbance.

4.1.1 Analysis of the Error Signal

Substituting (4.2) into (4.1) gives us

$$e(s) = \frac{1}{[1 + G(s)H(s)]} u(s) - \frac{H(s)}{[1 + G(s)H(s)]} d(s) \tag{4.9}$$

Once again if we either assume, or by design assume $|G(s)H(s)| \gg 1$, then the denominator term swamps the numerator terms in (4.9) leaving the error going to zero. Of course if as we cannot hold the gain high enough at high frequencies a frequency must occur when the error is no longer small and increases with frequency. What this means is that outwith our useful range, the closed-loop system can no longer keep tracking the input. This is the usual case for all physical realisable systems as they must all have a finite bandwidth or operating frequency range. For example, operational amplifiers only work over a defined bandwidth as per the data-sheet. They stop amplifying eventually and start attenuating!

4.2 Speed or Velocity Control of dc Motors

For most motors, we can get by just on plain open-loop control if we want to control speed in a crude fashion. A hair-dryer for instance may have several speed settings but does not have any form of feedback sensor on the speed since the speed is not critical to its operation. Many fans have switchable speeds and whether the speed is precise is of little consequence. For our electric cart in Fig. 4.1, as more load is added the motors will slow down and if this is of some concern then either bigger batteries need to be used to increases the overall maximum power, or feedback has to be added to control the speed. Let us first therefore consider feedback around a dc motor to control its speed accurately. Consider the schematic-diagram below.

In Fig. 4.4 a few things needs explaining before proceeding. The symbol used for difference is shown in Fig. 4.5.

This symbol is used frequently to denote negative feedback. Its physical form is usually (though not always) in an analogue control-system the summing junction of an operational amplifier as shown in Fig. 4.6. Even though the amplifier adds

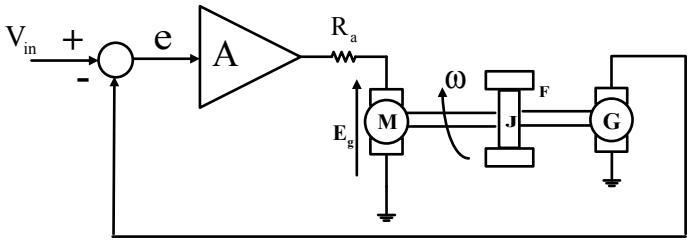
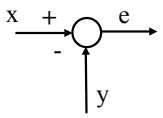


Fig. 4.4 Speed control of a dc motor

Fig. 4.5 Summing junction
e = x − y

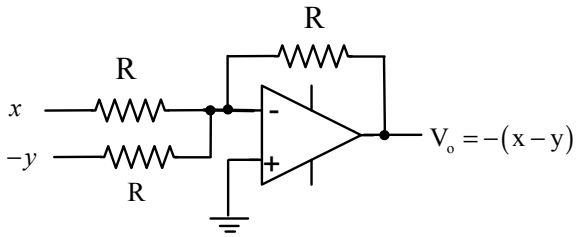


signals, this doesn't matter, since we arrange for the negative signal to have opposite polarity and the effect is the same. The overall invertion of the amplifier is likewise of no great concern as we can invert the sign back again later in the chain. Worst case is we may have to add an extra inverting amplifier if this is not possible, or use a subtracting amplifier instead (Fig. 4.7).

In Fig. 4.6 we have shown all resistances to be identical, but the feedback resistor can be replaced with a general impedance, as can the input resistors to the summing junction. There are certain technical issues with any op-amp circuit in a control-system which stems from any dc-offsets that occur at the output due to input offset currents. For example, in 4.7 when x = y we expect the output voltage to be zero. Electronic textbooks are full of methods to reduce these however, and biasing variable resistors can be used for some systems, though the approach is rather ad hoc in design methodology and is not a real fix. If this is not the case any dc offset will be amplified and act as an error which is not zero hence biasing the control-systems speed.

An op-amp is not powerful enough to drive a dc motor, so we need a dc power amplifier (an amplifier who's frequency response goes down to dc) after this in cascade. This then drives the motor of which here we only model the armature resistance and not its inductance. A better mathematical model would be to model

Fig. 4.6 A summing amplifier



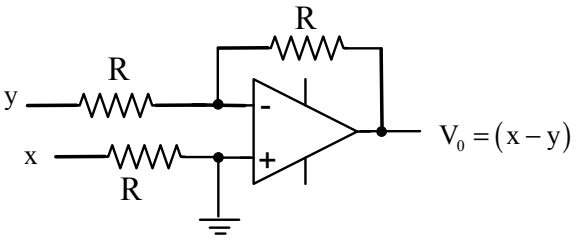


Fig. 4.7 A subtracting amplifier

inductance as well. The motor drives a load of moment of inertia *J* with coefficient of dynamic friction *F*. The friction term is the rotational equivalent of a dashpot in a translational damping system just as mass is the translational equivalent of inertia. There is no dashpot as such, this only models what friction there is present. Many motors have gear-boxes and a speed reduction comes next due to gears, but these are not always needed depending on the power of the motor. If a lot of torque is required then for a small motor we need a gear train. The sensor used is a small motor acting as a generator (or tacho). It gives a voltage directly proportional to angular velocity. This is quite old-fashioned, as today we use digital sensors with pulses to do the same thing, but this is covered in another chapter and the end result is the same in any case. It is far easier to learn digital-control after learning analogue than to go straight into it. To find the working of this closed-loop system we write down the equations and then draw a block-diagram.

Start with the dc motor and we have that its armature voltage is countered by its back emf.

$$v_a - E_g = I_a R_a \tag{4.10}$$

where I_a is armature current, E_g is generated of back emf, v_a is armature voltage and R_a is armature resistance. Since the field of the motor is fixed (we assume permanent magnetic field here),

$$E_g = k\omega \tag{4.11}$$

where k is the motor constant and ω is the speed of the motor in rads/s. Also the developed torque of the motor *T*

$$T = kI_a \tag{4.12}$$

Newton's laws give the dynamic relationship from Torque to speed of the inertia *J* and coefficient of rotational viscous friction *F*.

$$T = J\frac{d\omega}{dt} + F\omega \tag{4.13}$$

The armature voltage is driven by the dc amplifier with gain A and error

$$v_a = Ae = A(v_{in} - B\omega) \tag{4.14}$$

where v_{in} is the setpoint voltage or desired speed of the closed-loop system and B is the tacho constant. Here we have assumed that a voltage of value $v_T = B\omega$ comes from the tacho. It should be recognised that we know very few if any of these constants but this does not stop us modelling the system in open and closed-loop as this is important knowledge to proceed further with the design of the control-system. Simple proportional control as is shown here will not be enough, but is a good starting point. We term this proportional control when there is just a simple gain term. Usually this is not enough for a high-performance system. We can now draw from these equations the block-diagram of the motor and load. We start by neglecting the feedback, so this is for open-loop only (Fig. 4.8).

The first thing we notice that the dc motor itself looks like it has negative-feedback even without any external feedback! This is the effect of the back-emf of the motor. To reduce the diagram to something more recognisable we can multiply all the forward path blocks together as they are in cascade giving us the block-diagram of Fig. 4.9.

Now we use the simple block-diagram reduction method we already derived. For any system with forward-path transfer-function G and feedback-path H (we often drop the s notation for simplicity), then for negative-feedback the original and equivalent is shown in Fig. 4.10.

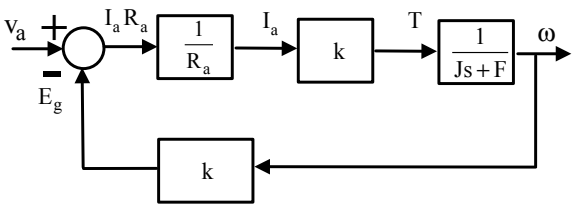


Fig. 4.8 Block diagram of motor plus load

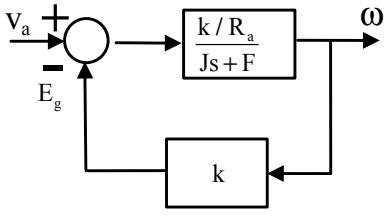


Fig. 4.9 Combine systems in forward path

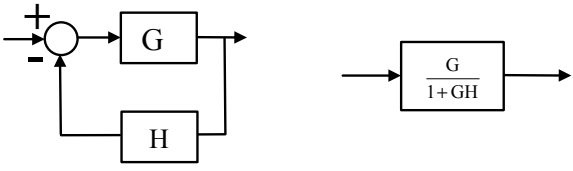


Fig. 4.10 Generic approach to basic feedback block-diagram reduction

Using this block-diagram method and applying it to Fig. 4.9 we obtain.

$$\frac{\omega}{v_a}(s) = \frac{\frac{k/R_a}{Js+F}}{1+\frac{k^2/R_a}{Js+F}} \tag{4.15}$$

The easiest way to deal with the simplification of an expression such as (4.15) is to multiply numerator and denominator by $Js + F$ giving

$$\frac{\omega}{v_a}(s) = \frac{k/R_a}{Js+F+k^2/R_a} \tag{4.16}$$

Now put this in a recognisable form. We use the first-order template

$$\frac{\omega}{v_a}(s) = \left(\frac{k}{R_aF+k^2}\right)\left(\frac{1}{1+s\frac{R_aJ}{R_aF+k^2}}\right) \tag{4.17}$$

with motor time-constant

$$T_m = \frac{R_aJ}{R_aF+k^2} \tag{4.18}$$

We can see that the larger is the inertia, the larger the time-constant and the longer a step-response will take to reach-steady state (from earlier theory). It is usual to model a dc-motor as a first-order system in this way though sometimes for higher bandwidth (faster) control-systems the armature inductance must also be included and even any mechanical resonances terms. For a simple model however this suffices as a good starting point. The fact that the numerator is scaled by $\left(\frac{k}{R_aF+k^2}\right)$ and is not unity doesn't matter at all, since this in turn will be scaled by the amplifier in the loop.

Applying the speed-loop (the outer feedback) we get a new block-diagram if we include our already calculated model for the dc-motor. First define a new constant $k_2 = \left(\frac{k}{R_aF+k^2}\right)$. The motor transfer-function is now $\frac{k_2}{1+sT_m}$ and when feedback from the tacho is applied we have the block-diagram as shown in Fig. 4.11 we have absorbed the amplifier gain into the motor since it is in cascade and we assume no loading provided the amplifier output impedance is low enough.

Then we use the generic approach to reducing a block-diagram (Fig. 4.10) and arrive at

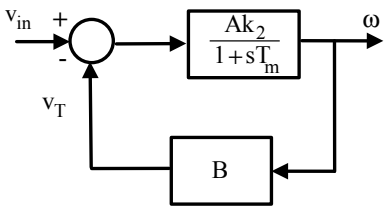


Fig. 4.11 Outer speed-loop is closed

$$\frac{\omega}{v_{in}}(s) = \left(\frac{\frac{Ak_2}{1+sT_m}}{1+\frac{BAk_2}{1+sT_m}}\right) \tag{4.19}$$

Further algebraic manipulation yields

$$\frac{\omega}{v_{in}}(s) = \left(\frac{\frac{Ak_2}{1+ABk_2}}{1+s\left(\frac{T_m}{1+ABk_2}\right)}\right) \tag{4.20}$$

This is yet another first-order system but the time-constant we can re-define as the closed-loop time-constant $T_c = \frac{T_m}{1+ABk_2}$ giving

$$\frac{\omega}{v_{in}}(s) = \left(\frac{k_3}{1+sT_c}\right) \tag{4.21}$$

where $k_3 = \frac{Ak_2}{1+ABk_2}$.

What interests us is that the closed-loop time-constant $T_c = \frac{T_m}{1+ABk_2}$ is the open-loop one divided by a large number A, the amplifier gain. This will make the closed-loop time-constant small and speed up the response-time significantly. This is one of the great advantages of negative-feedback, the increases in rise-time in closed-loop. We note that the pole of this system in (4.21) is at $s = -\left(\frac{1+ABk_2}{T_m}\right)$ and this is negative indicating closed-loop stability.

Example 4.1. Speed-Control

Determine the open and closed-loop unit step-responses of the system in Fig. 4.4. Use the following parameters. Armature resistance $R_a = 1\,\Omega$, Gain A = 100 (40 dB of gain), Moment of inertia $J = 1\,kgm^2$, dynamic friction coefficient F = 1 Nm per rad/s, k (motor constant) = 5 Nm/A, tacho constant B = 1 v/rad/s.

Solution Open-loop transfer-function of the motor + load.

From (4.17)

$$\frac{\omega}{v_a}(s) = \left(\frac{k}{R_aF + k^2}\right)\left(\frac{1}{1 + s\dfrac{R_aJ}{R_aF + k^2}}\right) = \frac{k_2}{1 + sT_m}.$$

Find $k_2 = \left(\dfrac{k}{R_aF + k^2}\right) = \left(\dfrac{5}{1\times 1 + 5^2}\right) = 0.192$. $T_m = \dfrac{R_aJ}{R_aF + k^2} = \dfrac{1\times 1}{1 + 5^2} = 38.5$ ms.

$$\frac{\omega}{v_a}(s) = \frac{0.192}{1 + 0.0385s}.$$

Now use MATLAB to plot the step-response for a unit-step input.

Solution. Closed-loop transfer-function of the motor + load.
From (4.20)

$$\frac{\omega}{v_{in}}(s) = \left(\frac{\dfrac{Ak_2}{1 + ABk_2}}{1 + s\left(\dfrac{T_m}{1 + ABk_2}\right)}\right)$$

$$1 + ABk_2 = 193$$

$$\frac{\omega}{v_{in}}(s) = \left(\frac{\dfrac{Ak_2}{1 + ABk_2}}{1 + s\left(\dfrac{T_m}{1 + ABk_2}\right)}\right)$$

$$\frac{\omega}{v_{in}}(s) = \left(\frac{\dfrac{0.192\times 100}{1 + 0.192\times 100}}{1 + s\left(\dfrac{T_m}{0.192\times 100}\right)}\right)$$

$= \left(\dfrac{0.95}{1 + S(2\times 10^{-3})}\right)$. So that the closed-loop time-constant is $\dfrac{T_m}{1 + ABk_2}$ or 1.9 ms. This is 38.5/1.9 = 20 times faster in closed-loop. Now use MATLAB to plot the step-responses and get the result of Fig. 4.12.

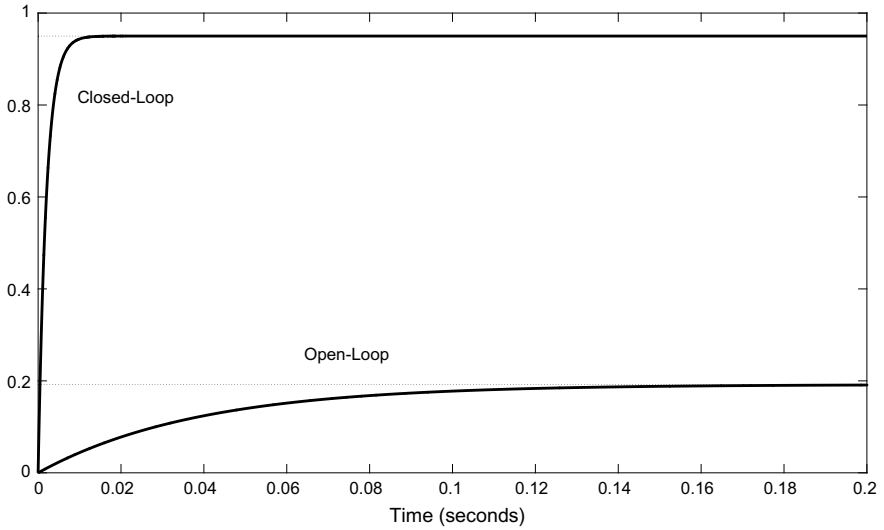


Fig. 4.12 Step response to speed of motor plus load (open-loop) and closed-loop

MATLAB Code 4.1

```
num=0.192;
den=[0.0385 1];
step(num,den)
hold on
num=1;
den=[0.2e-3 1];
step(num,den)
```

We see that not only is the closed-loop system 20 times faster, but the setpoint reaches closer to unity (the setpoint or desired value) in closed-loop whereas there is a larger error in steady-state in open-loop.

Example 4.2. Load-Torque

Find an expression for the error introduced by a load-torque of magnitude 1 Nm for the same parameters as Example 4.1.

The problem suddenly becomes much harder since the load-torque (a disturbance) is embedded in the inner loop of the closed-loop system. This is shown in Fig. 4.13.

In the previous example we could reduce each loop in a nested fashion with the inner-loop first, but now the load-torque gets in the way. We need another few rules of block-diagram algebra. Here they are below (Figs. 4.14, 4.15).

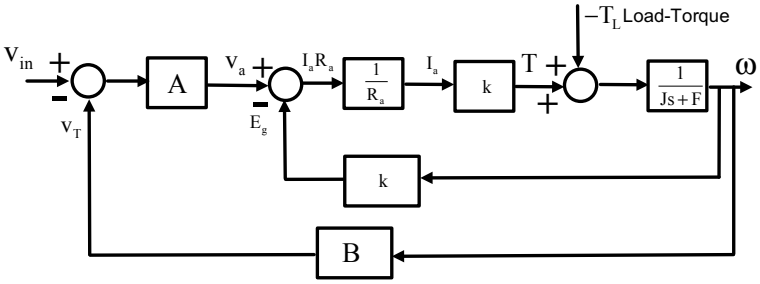


Fig. 4.13 Block-diagram showing load-torque

Fig. 4.14 Moving a summing-junction before a block

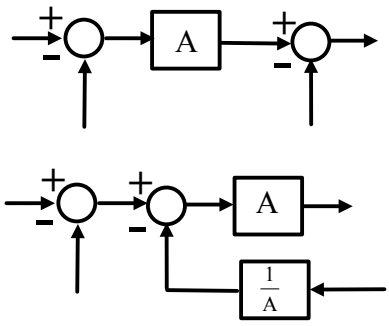
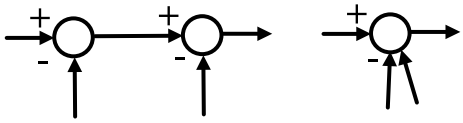


Fig. 4.15 Combining two summing-junctions



The first of these block-diagram manipulations is termed moving a summing-junction before a block and requires a second block with the reciprocal of the block you are moving it past to the left to become added as shown in Fig. 4.14. In Fig. 4.16, two summing-junctions are combined to one summing-junction with two negative inputs. So in Fig. 4.13 we must move the second summing-junction to the left before the *A* block and introduce a reciprocal *A* where k is at present. This makes k now *k/A* Then we combine the two summing-junctions into one with two negative inputs. Figure 4.16 shows the result of these moves and leaves us with two loops with identical sources and end-points.

Since the two outer loops now come from the same source to the same destination we can now add them giving the result of Fig. 4.17.

Now we use superposition and set the setpoint to zero and a negative sign now appears in from of the gain *A* (Fig. 4.18).

The minus sign is more conveniently put in place of the plus sign at the load-torque summing-junction. Once we have done this we can re-arrange the

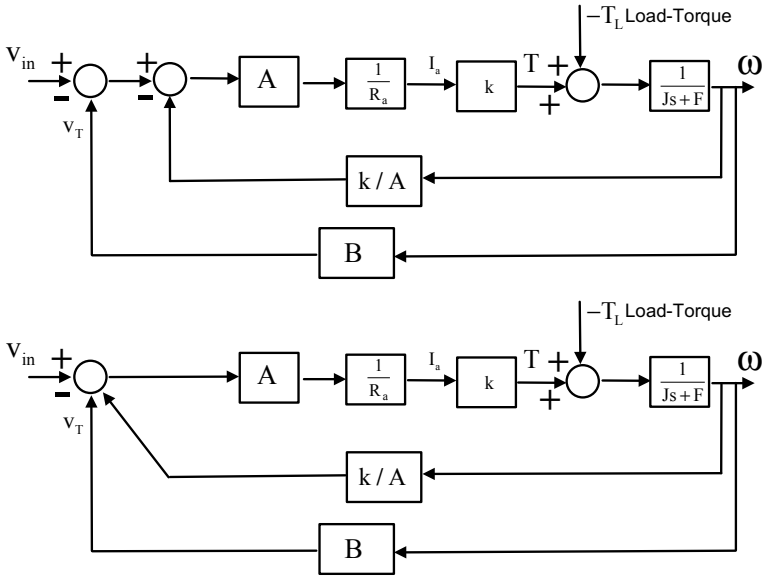


Fig. 4.16 Block-diagram reduction. We try and isolate the load-torque from the inner loop

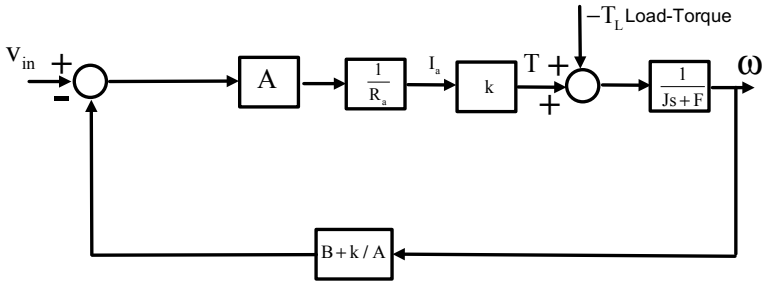


Fig. 4.17 Combining two loops

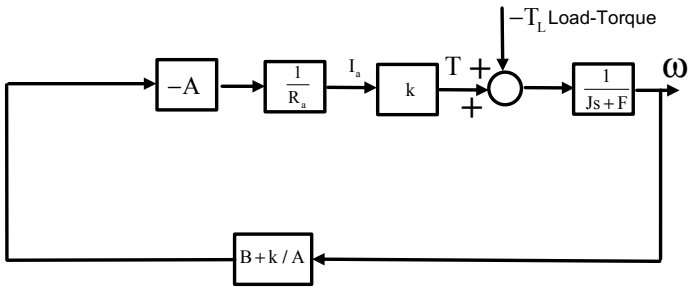


Fig. 4.18 Put setpoint to zero

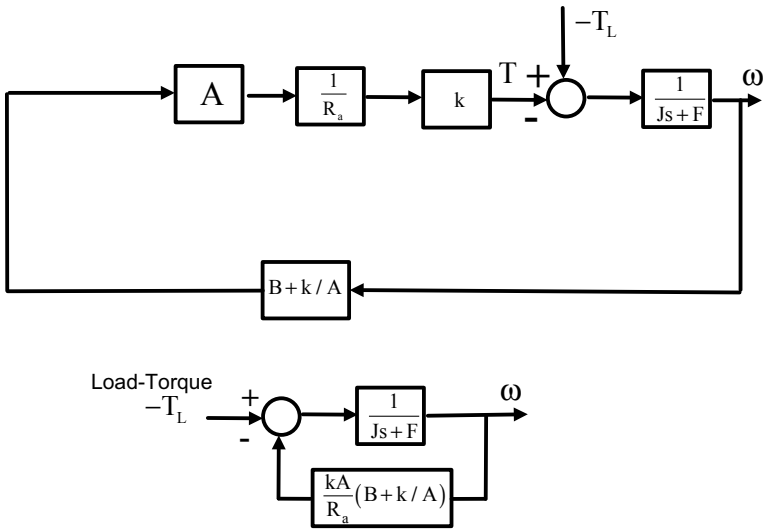


Fig. 4.19 Change the sign at the summing junction and re-arrange

blocks so that the torque is in a more familiar place, the left hand side of the diagram (Fig. 4.19).

Now we have a relatively simple arrangement to simplify. Note that the load-torque is negative. This is because the torque slows the motor down and does not speed it up in our favour!

$$\frac{\omega}{-T_L}(s) = \frac{\frac{1}{Js+F}}{1+\left(\frac{1}{Js+F}\right)\frac{kA(B+k/A)}{R_a}} \tag{4.22}$$

$$\frac{\omega}{-T_L}(s) = \frac{R_a}{R_aJs+R_aF+kAB+k^2} \tag{4.23}$$

$$\frac{\omega}{-T_L}(s) = \frac{R_a/\left(R_aF+kAB+k^2\right)}{1+s\frac{R_aJ}{R_aF+kAB+k^2}} \tag{4.24}$$

We note that the closed-loop motor time-constant $T_c = \frac{R_aJ}{R_aF+kAB+k^2}$ and this can be found to be 1.9 ms as was calculated previously. Negative feedback has the same poles no matter where the input is put in a loop. We also find $R_a/\left(R_aF+kAB+k^2\right) = 1.9\times10^{-3}$.

$$\frac{\omega}{-T_L}(s) = \frac{1.9\times10^{-3}}{1+1.9\times10^{-3}s} \tag{4.25}$$

This is just a first-order system. If the input is a step of magnitude 1 Nm load-torque, the final value of the speed can be found from the *final-value theorem*. $T_L = \frac{1}{s}$ then the speed is found for a gain of $A = 100$ from

$$\begin{aligned}\omega(t)|_{t\to\infty} &= -s \times \frac{1.9 \times 10^{-3}}{s(1 + 1.9 \times 10^{-3}s)}\Big|_{s=0} \\ &= -1.9 \times 10^{-3}\text{rad/s}\end{aligned} \tag{4.26}$$

So the speed drops by a small amount due to the load-torque. This speed gets superimposed on top of whatever the stead-state speed is. The speed drop is not of any significance in steady-state because the gain of the loops fights this disturbance. We introduce *Simulink* as a method of simulating the effect of load disturbance. Simulink is usually bundled with MATLAB in most academic institutions.

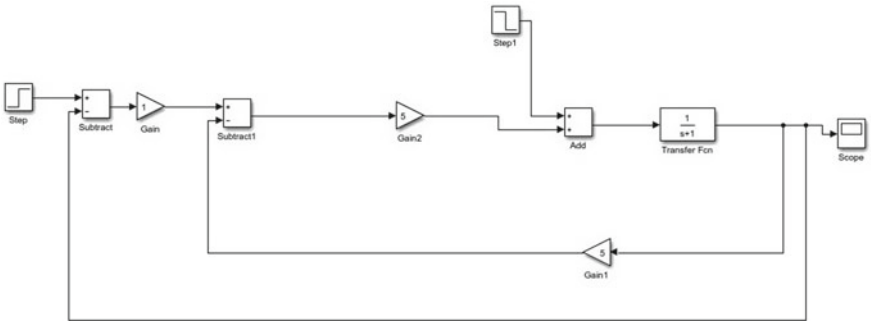


Fig. 4.20 Simulink simulation of speed-control plus load

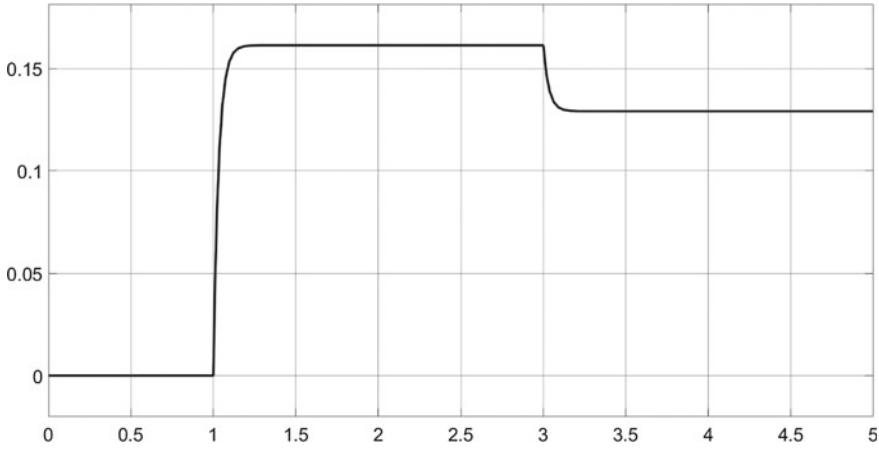


Fig. 4.21 A = 1. Unit step of load-torque applied at time 3 s

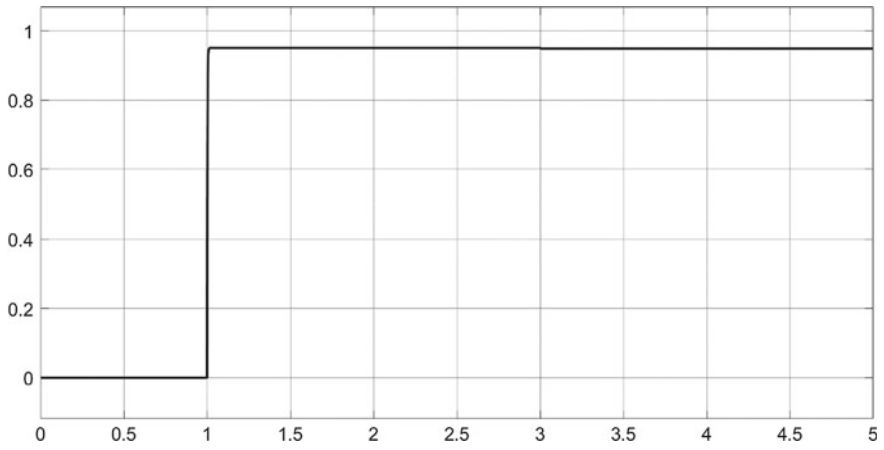


Fig. 4.22 Gain A = 100. Unit step of load applied at time 3 s

Figure 4.20 shows the Simulink model and Fig. 4.21 shows the effect of a load step of magnitude unity when the gain $A = 1$. A unit step input was applied first at time 1 s followed by the load torque at time 3 s.

The drop in speed is clearly visible and this drop has a transient which is also first-order in nature with similar dynamics to the step-response but in the negative direction.

In contrast, reducing the forward-path gain A to 100 gives us the result of Fig. 4.22 (see also the result from Eq. (4.26)) We can hardly see the effect of the disturbance at all. It is only just discernible at time 3 s. Also note that the output speed is closer to unity, the desired speed and hence the error has clearly been reduced.

The higher is the loop-gain, the more effort can be put into rejecting a disturbance and the nearer the output gets to the desired or setpoint value.

We can improve on this more once we have learned controller design.

4.3 Position Control of dc Motors

Position-control or position servos as they are often known have been around for a very long time as applied to dc motors. Small varieties can be bought off-the-shelf (Fig. 4.23) and are used to control model aircraft and robots. They come in a variety of sizes with nylon or metallic gear-boxes. The gear-boxes are fixed ratio and used to get more torque out of the servo. The alternative is to use a large motor which is not desirable for many of their applications and is more expensive. The purpose of a position servo is to control the angle of a motor and not its speed. To turn the motor by a desired amount of degrees and come to a halt as quickly as possible is the main aim.

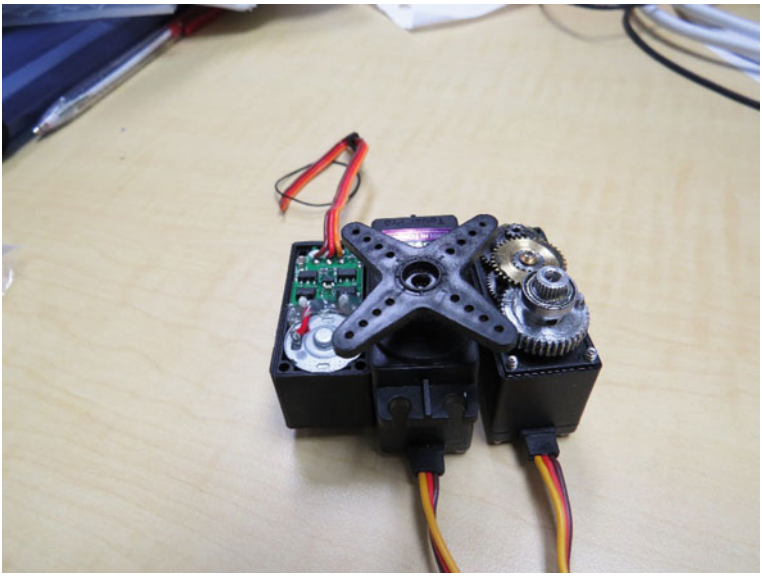
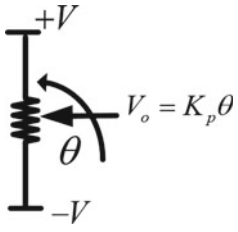


Fig. 4.23 Shows the photograph of a model remote-control (RC) servo as used by hobbyists

As an off-the-shelf solution to small problems the hobby servo is a great invention. It combines many of the concepts studied here. They have a standard for the setpoint which is based on pulse-width modulation (PWM). This is historic since that was a convenient method of modulating a reference signal (since dc cannot be modulated easily) and transmitting it to model aircraft. Within the electronics of the servo a low-pass filter converts this back to dc. In common with nearly all servos of the era it was invented in, it uses a potentiometer attached to the driving shaft as a sensor to determine a voltage directly proportional to angular displacement (Fig. 4.24).

Hence the voltage at the wiper of the potentiometer is directly proportional to the angular movement. The constant of proportionality is called K_pv/rad. The only problem with analogue potentiometers is that their travel is restricted to less than one turn of the shaft, usually about 180° or less. This is usually enough for many applications however, but if multiple turns are needed then we usually have to move to digital control and use quadrature-encoders instead.

Fig. 4.24 A potentiometer



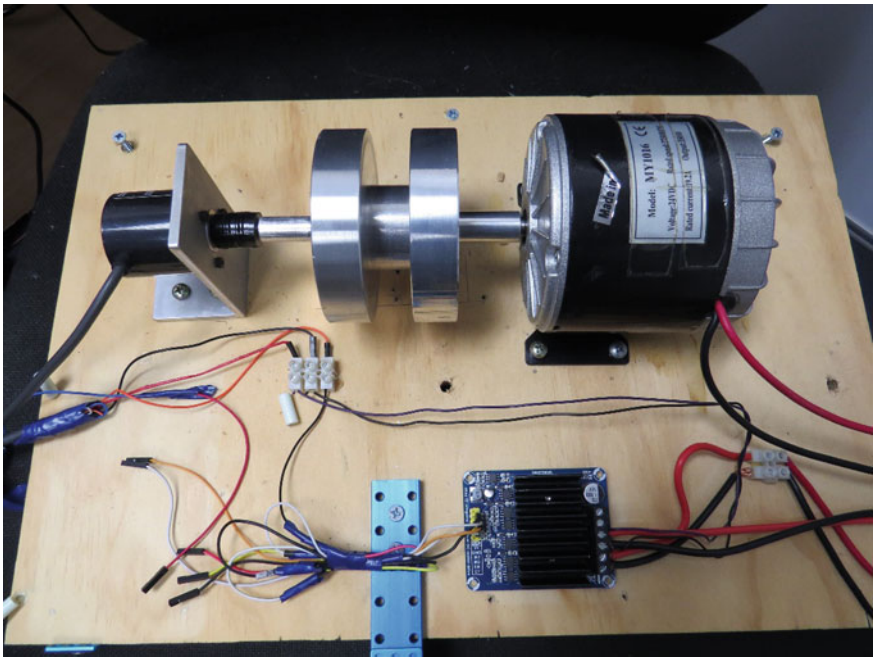


Fig. 4.25 Home-built rig for demonstrating digital-control of a 350 W motor

In Fig. 4.25 the dc motor is connected to a winding shaft made of aluminium whose angular position is to be controlled. To detect the angular position a digital quadrature encoder is attached directly to the end of the connecting shaft. A quadrature encoder (also known as rotary-encoders) uses two channels (usually named A and B) to sense position optically. The pattern of visibility and blanking of a light is arranged as in Fig. 4.26.

In Fig. 4.26, B leads A for say clockwise and vice-versa for anti-clockwise rotation. Pulses are counted in a given time to determine velocity if it is required.

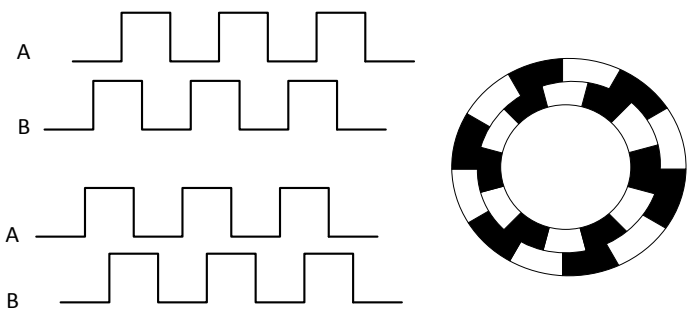


Fig. 4.26 Quadrature or incremental encoder

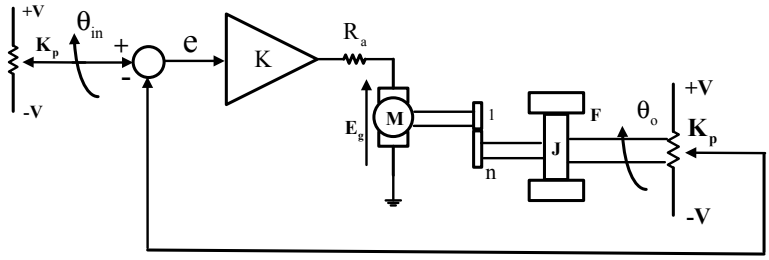


Fig. 4.27 Traditional analogue position servo

The Quad encoder is not an absolute position sensor unlike a potentiometer. When the system is powered-up some form of reference or *homing* needs to be done to tell the controller where it is. The advantage is much less friction than a potentiometer and a servo can have a setpoint of 100 s of turns if need be (in fact the number of turns is only limited by the word length of an integer for counting). To do this we need a second encoder acting as the setpoint usually though a computer generated setpoint will also suffice. Traditionally a position servo looked like Fig. 4.27.

Usually two potentiometers are used, one for the setpoint and one for the sensor to measure the angle of the shaft. The first thing we notice is that such a system cannot be used in open-loop. In open-loop the shaft turns continuously and we require it to turn only a desired number of degrees. This is really an important concept and the position servo has found applications since the days of vacuum tubes in all manner of positioning systems from large radio telescopes through missile launching systems to the hobbyist market. Often the hardware differs as we have already mentioned. For instance, dc amplifiers run a constant quiescent current to hold the motor in place and are not an efficient way of doing the job. Nowadays they have been replaced almost entirely with the H-Bridge. The H-Bridge is a glorified switch (see Fig. 4.28). It is driven from pulse-width modulation (PWM) instead of dc. It has done for the servo what the switching regulator has done for the dc regulator or appliance charger.

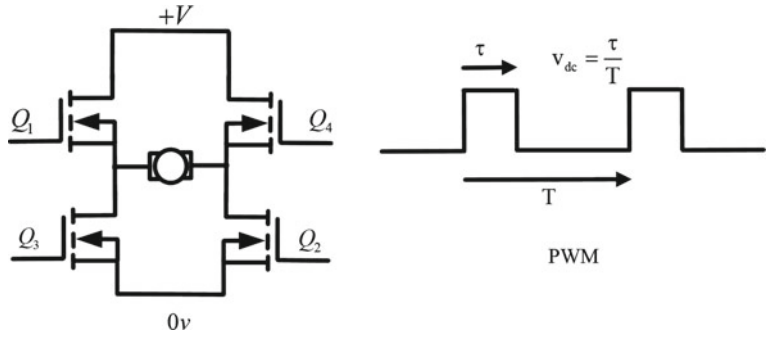


Fig. 4.28 H-Bridge concept

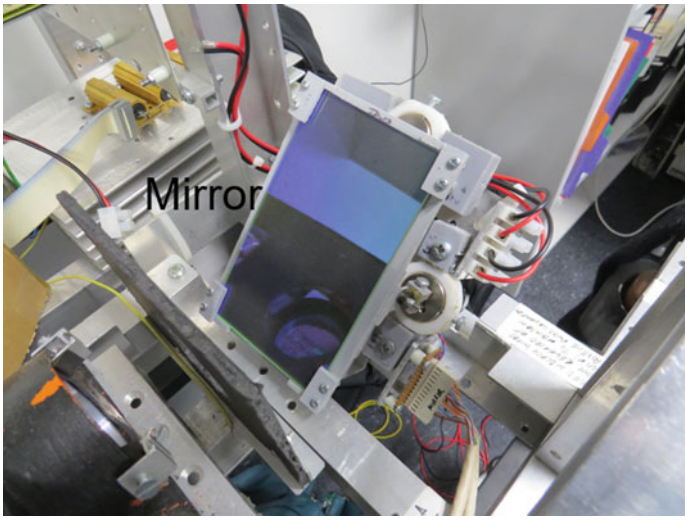


Fig. 4.29 Mirror in a Scheiner principle optometer (Courtesy David Taylor AUT)

In Fig. 4.28 the four MOSFET switches are turned on such that for one direction Q1 and Q2 are on and Q3, Q4 are off and for the reverse Q3 and Q4 are turned on with Q1, Q2 off. Although the MOSFET switches could just be fully on or off in this way for forward and reverse, there is no control of current through the motor and so on-off switching is enabled, which is PWM. The average value of the current is determined by the PWM applied voltage as shown in Fig. 4.28. Although Bipolar transistors can be used for H bridges, MOSFETs are preferred due to their low *on* resistances and less heat is generated. Not all servos have rotational shafts. Servos can be driven in a translational mode by using a Voice-Coil Motor (VCM). In Fig. 4.29 a mirror needs to be controlled precisely and at speed by computer-control. Two VCM motors are used to position the pitch and roll of the mirror as part of a special Optometer.

One of the VCMs is visible from the rear of the mirror (Fig. 4.30). A Linear-Variable Differential Transformer (LVDT) is used for both axis to measure the displacement. This works on a different principle from a potentiometer, namely inductance rather than resistance.

So there are a number of differing ways of solving similar problems. For simplicity we stick initially to the diagram in Fig. 4.27. Whether there is an H-Bridge or linear amplifier, potentiometer or LVDT, potentiometer or quadrature-detector makes little difference to the mathematical model we use. The laws of physics are just the same whether the control-system is analogue, digital or somewhere in-between. We note that in the case of the servo in Fig. 4.27 we have included a gear-ratio to model a gear-train. The purpose of the gears are to increase torque to the load. A similar reduction in speed is experienced at the load since a gear-train behaves as follows.

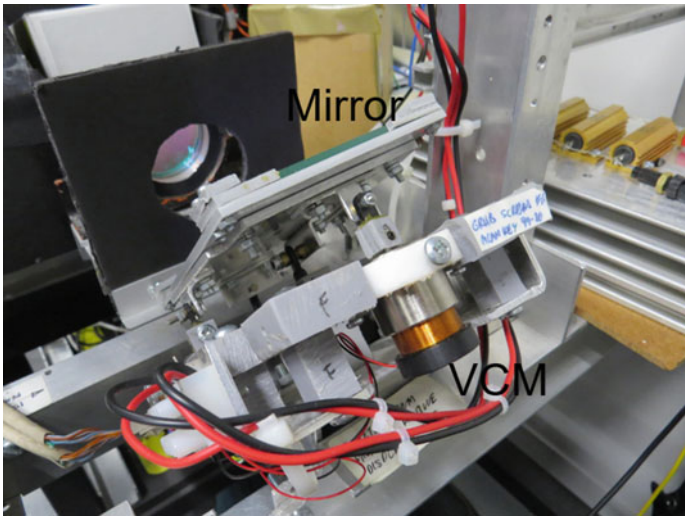


Fig. 4.30 Shows the VCM driving the mirror in the optometer

Ideal gear-train

Assume in Fig. 4.31 that the number of teeth on the n1 gear is less than the number on n2. Then for an ideal gear-train the power-out is the same as the power input.

$$\omega_{in} T_{in} = \omega_o T_o \tag{4.27}$$

Or in more familiar form like a transformer analogy

$$\frac{\omega_{in}}{\omega_o} = \frac{T_o}{T_{in}} \tag{4.28}$$

But we know that $\omega_{in} = \frac{n2}{n1}\omega_o$. Substitute into (4.28) and we get

$$\frac{n2}{n1}\omega_o T_{in} = \omega_o T_o \tag{4.29}$$

So that

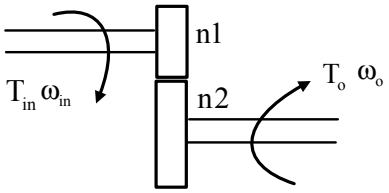


Fig. 4.31 Mathematics of gears

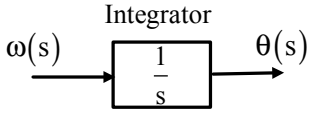


Fig. 4.32 Block diagram showing angular velocity to angular position

$$T_0 = \frac{n2}{n1} T_{in} \tag{4.30}$$

It is easier to just replace n2/n1 with n as in Fig. 4.27. The gear behave like a transformer. If speed goes down the torque increases by the same amount. One more thing we need before drawing the block-diagram dynamics of Fig. 4.27. Unlike the speed-control we now measure angular displacement. $\frac{d\theta(t)}{dt} = \omega(t)$. In Laplace format we get the block-diagram of Fig. 4.32.

We see that there is an integrator, but this is not an electronic integrator. Nevertheless it has the same properties as an integrator. Using this information we draw the entire block-diagram of a position-control system as shown in Fig. 4.33. Note that we have distinguished between motor torque and shaft torque, motor speed and shaft speed. The first thing we can simplify is at the first summing junction. We see the potentiometers are the same with the same constants. A block-diagram simplification can be used here as follows (Fig. 4.34).

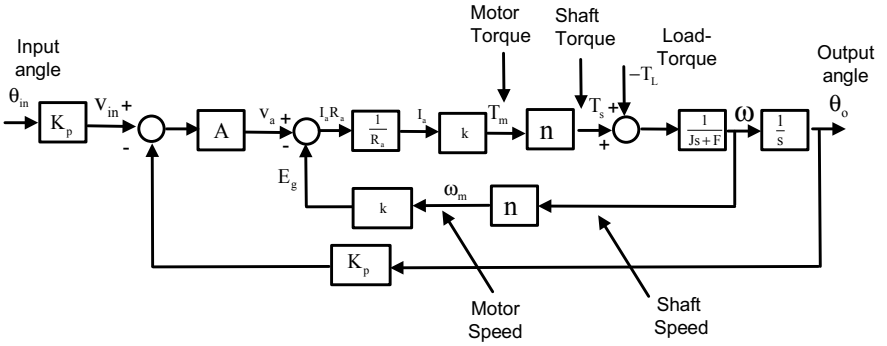


Fig. 4.33 Block-diagram of position-control-system

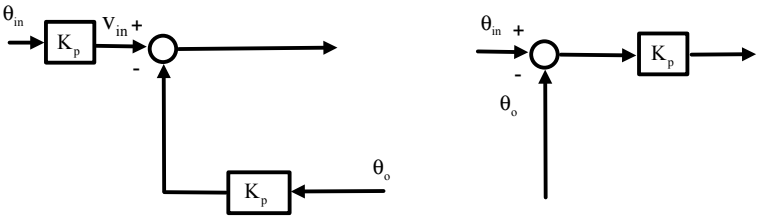


Fig. 4.34 Block-diagram algebra rule

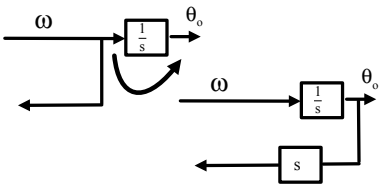


Fig. 4.35 Move a pick-off point in front of a block

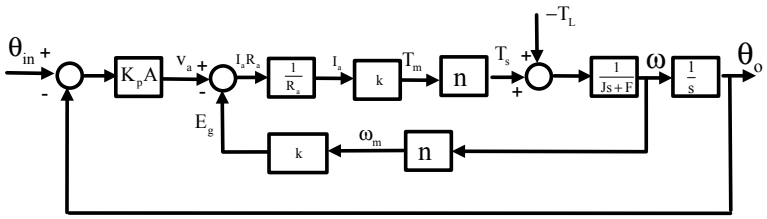


Fig. 4.36 Merge the two potentiometers together

One final block-diagram reduction is termed moving a pick-off point in front of a block. This is handy if we want to merge outer and inner loops together (Fig. 4.35).

We don't use that last step straight away, but from the first we can re-draw the block-diagram as shown in Fig. 4.36.

4.3.1 The No-Load-Torque Case

To find the block-diagram when the load-torque is zero we can either use clever block-diagram manipulation methods or just reduce the inner-loop then outer. When there is no load torque it is easier to use the latter method and we proceed as follows from Fig. 4.36 with $-T_L = 0$. The inner-loop gives us:

$$\frac{\omega}{v_a}(s) = \frac{\frac{(kn/R_a)}{(Js+F)}}{1+\frac{k^2n^2/R_a}{(Js+F)}} \tag{4.31}$$

$$= \frac{kn}{R_aJs + R_aF + k^2n^2} \tag{4.32}$$

$$= \frac{kn/\left(R_aF + k^2n^2\right)}{1 + s\frac{R_aJ}{R_aF+k^2n^2}} \tag{4.33}$$

This is again a first-order system which we will write in the form

$$\frac{\omega}{v_a}(s) = \frac{k_2}{1 + sT_m} \tag{4.34}$$

where $k_2 = kn/\left(R_aF + k^2n^2\right)$ and the time-constant $T_m = \frac{R_aJ}{R_aF + k^2n^2}$.

Now we can add the integrator, and motor plus inertia gives us

$$\frac{\theta_o}{v_a}(s) = \frac{k_2}{s(1 + sT_m)} \tag{4.35}$$

This then has negative feedback around it with forward path gain K_pA and unity negative feedback. We obtain

$$\frac{\theta_o}{\theta_{in}}(s) = \frac{\frac{AK_pk_2}{s(1+sT_m)}}{1 + \frac{AK_pk_2}{s(1+sT_m)}} \tag{4.36}$$

$$= \frac{AK_pk_2}{s(1 + sT_m) + AK_pk_2}$$

$$= \frac{AK_pk_2}{s^2T_m + s + AK_pk_2}$$

Finally

$$\frac{\theta_o}{\theta_{in}}(s) = \frac{AK_pk_2/T_m}{s^2 + s/T_m + AK_pk_2/T_m} \tag{4.37}$$

This is a classic second-order system. Compare the two characteristic polynomials.

$s^2 + s/T_m + AK_pk_2/T_m$ with $s^2 + 2\zeta\omega_n s + \omega_n^2$ and we find the undamped natural frequency to be $\omega_n = \sqrt{AK_pk_2/T_m}, \zeta = \frac{1}{2T_m\sqrt{AK_pk_2/T_m}}$.

Putting some numbers in this problem, similar to the speed-control example. Determine closed-loop unit step-response of the system in Fig. 4.36. Use the following parameters. Armature resistance $R_a = 1\,\Omega$, Gain A = 100 (40 dB of gain), Moment of inertia $J = 1\,kgm^2$, dynamic friction coefficient F = 1 Nm per rad/s, k (motor constant) = 5 Nm/A, potentiometer constant $K_p = 2v/rad$, gear ratio n = 10, where the gears can be a train of gears but we represent them by a single ratio. We find after substitution of the given values that $T_m = \frac{R_aJ}{R_aF + k^2n^2} = 1/(1 + 25 \times 100) = 0.399\,ms$. Also we have $k_2 = kn/\left(R_aF + k^2n^2\right) = 50/(1 + 2500) = 0.02$. Then

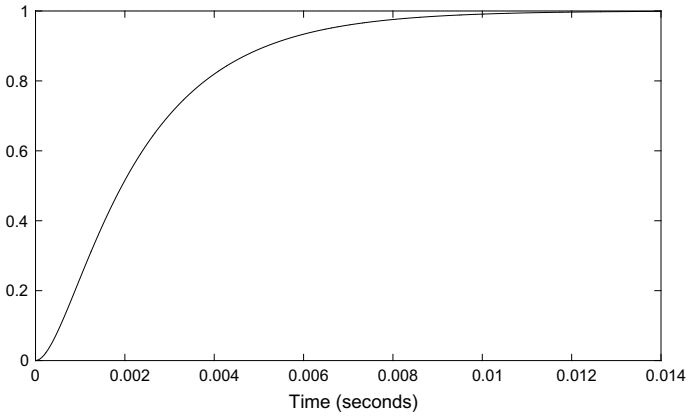


Fig. 4.37 Step-response of position-servo, gain A = 100,000

$$\frac{\theta_o}{\theta_{in}}(s) = \frac{AK_p k_2/T_m}{s^2 + s/T_m + AK_p k_2/T_m} = \frac{10^4}{s^2 + 2501s + 10^4}$$

This has an undamped natural frequency of 100 rad/s and a damping factor of $\zeta = 12.5$. This would give a step-response which is far too slow and heavily damped for any practical purposes. Therefore we repeat the procedure but increase the gain to A = 10,000. We get

$$\frac{\theta_o}{\theta_{in}}(s) = \frac{AK_p k_2/T_m}{s^2 + s/T_m + AK_p k_2/T_m} = \frac{1 \times 10^6}{s^2 + 2501s + 1 \times 10^6}$$

Using these values gives us an undamped natural frequency of 1000 rad/s and a damping factor of $\zeta = 1.25$. Using the results of Chap. 3, this would give us roughly critical damping (Fig. 4.37).

Although this looks like a promising result, it is unlikely that the gain would reach that far before other dynamics which we have not modelled come into play. For example the inductance of the motor plays a role meaning the actual system is at least of order 3 and as we shall see later mechanical resonances also play a part.

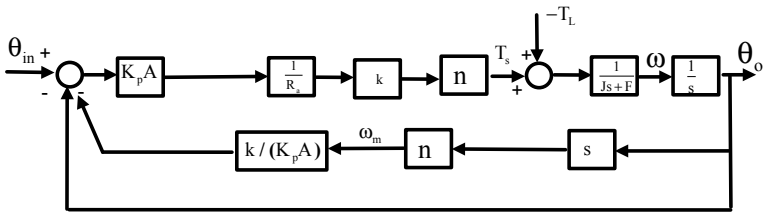


Fig. 4.38 Merging two loops into one using: (i) Moved inner summing junction to the left beyond the K_pA block, (ii) moved the pickoff point to the right beyond the integrator block

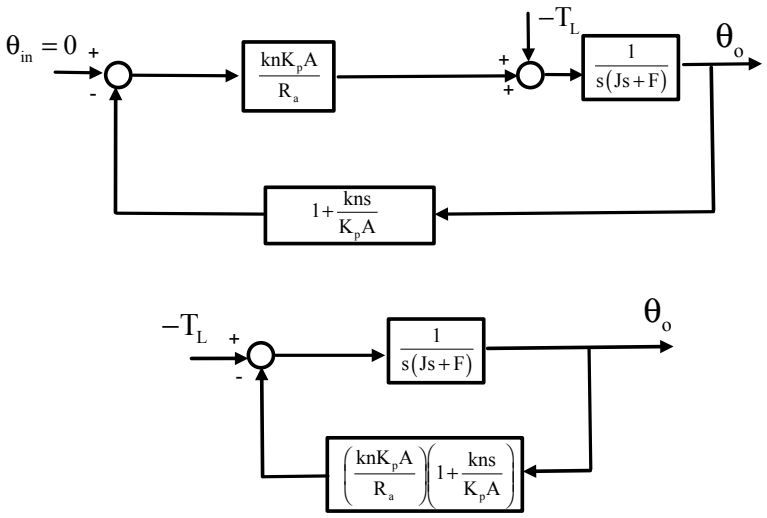


Fig. 4.39 Final two steps in obtaining transfer-function relating output angular position to disturbance torque

4.3.2 Effect of Load-Torque

Before deciding on a better stradegy other than just a single gain, consider first what happens to the position of a load torque is applied of magnitude unity at time t = 3 s (Fig. 4.38).

The final steps are shown in Fig. 4.39 by using superposition and setting the setpoint to zero. We then put the disturbance torque to the left and we have a familiar looking generic feedback block-diagram.

Reducing the block diagram in Fig. 4.39

$$\frac{\theta_o}{-T_L}(s) = \frac{\frac{1}{s(Js+F)}}{1+\frac{(knK_pA+k^2n^2s)}{R_as(Js+F)}} \tag{4.38}$$

$$= \frac{R_a}{R_as(Js+F)+knK_pA+k^2n^2s}$$

$$= \frac{R_a}{R_aJs^2+s(k^2n^2+R_aF)+knK_pA} \tag{4.39}$$

$$= \frac{R_a/(R_aJ)}{s^2+s\frac{k^2n^2+R_aF}{R_aJ}+\frac{knK_pA}{R_aJ}}$$

For a unit-step of torque $T_L = 1/s$

$$\theta_o(s) = -\frac{1/J}{s\left(s^2+s\frac{k^2n^2+R_aF}{R_aJ}+\frac{knK_pA}{R_aJ}\right)} \tag{4.40}$$

Multiply by s and let s go to zero gives us the final value theorem solution for steady-state position deviation created by load-torque:

$$\theta_o = -\frac{R_a}{knK_pA} \tag{4.41}$$

This will be small provided the gain A is large. For A = 100,000 and the same values as for the step input with no load-torque this value is negligible.

We conclude by reminding ourselves that we have shown the effect of negative feedback is to speed up the system by an amount in proportion to the forward-path gain and to reduce disturbance by an amount governed by the forward-path gain. These are both excellent properties provided we can keep the closed-loop system stable and herein lies the essence of control-engineering.

Chapter 5
Frequency Response Methods

In the previous chapters, the basic theory has been laid down of dynamic systems and a few applications and benefits of using negative feedback have been shown. Although for simple academic examples we can always get nicely formed solutions, many questions remain unanswered. For example, how do we arrive at these mathematical models in the first place? Electrical networks are fairly straightforward to analyse and likewise circuitry with operational amplifiers, but when mixed with mechanical additions the complexities quickly magnify themselves. This can be done, but can take a great deal of time and therefore methods of identifying the unknown system have been developed for this very purpose. Some methods involve sophisticated least-squares and recursive-least squares approaches for example [19] and I have no doubt that as machines and software become more manageable, cheaper and portable, that these methods may well win the day. However, engineers still do a lot of bench work using frequency analysis, and it is this approach that we concentrate on in the chapter.

5.1 Frequency-Response of Linear Systems

Consider Fig. 5.1, where a sinewave is injected into an unknown system and the input and output amplitudes measured along with the phase-difference.

This method is very popular still with electronic engineers but perhaps less so in areas where it may not be practical to inject a sinewave. For example in chemical systems, thermal systems and fluid systems the method whilst still theoretically possible, may be hard or impossible to carry out in practice. Therefore we can use other methods where this is the case. If at all practical though, the frequency response measurement is still the best way to proceed. Most of the equipment is readily available and no computer interfacing is needed or data capture. In order to interpret the data obtained however we need to know templates of various classes of linear systems, just as we did in the case of the step-response. Consider the systems

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_5

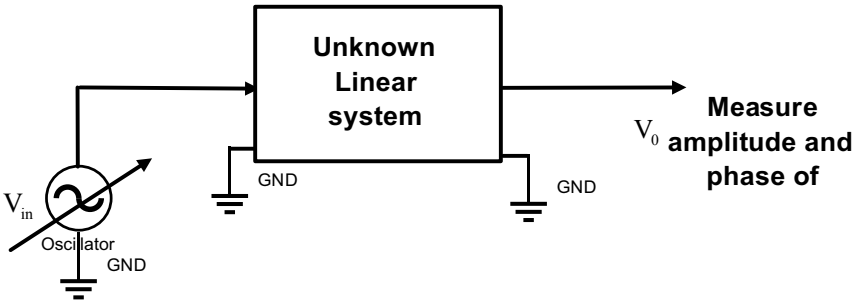


Fig. 5.1 Method of measuring frequency-response

shown in Fig. 5.2. If we know the frequency-response of these, then any combination of these can be cascaded for any order linear-system. The plot in dB magnitude versus frequency (rad/s) and phase versus frequency (rad/s) is known as the *Bode-Plot*. We substitute $s = j\omega$ to find the theoretical frequency-response. Since s is a complex-number, we can then calculate the dB magnitude and degrees of phase as a function of angular frequency ω rad/s. We could use frequency in Hz but in control textbooks rad/s is used more frequently.

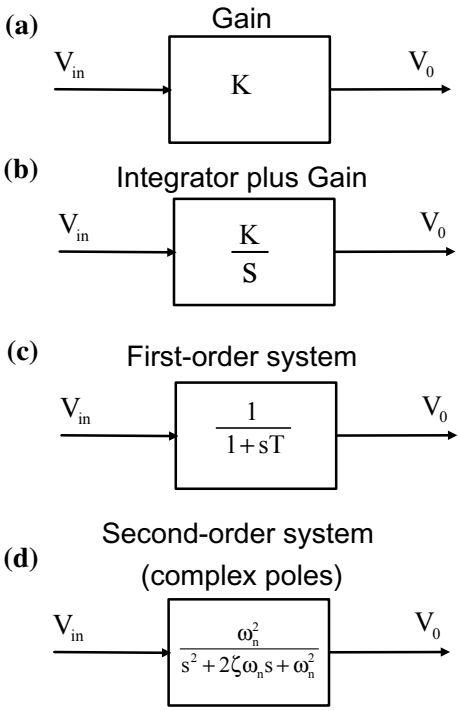


Fig. 5.2 Types of linear system encountered. **a** Ordinary gain, **b** integrator and gain, **c** first-order system, **d** second-order system with complex poles

5.1.1 Ordinary Gain

See Fig. 5.3.

In theory it is not possible to design an amplifier with just gain and no other dynamics. Since we will be considering these other cases however we can treat the ideal case first. Consider a system with just a gain *K*.

Its gain in dB is by definition called the log-magnitude (LM)

LM = 20log10(K) dB at all frequencies and the phase is zero degrees. For example, for a gain of K = 100, the frequency-response is a horizontal straight line at 20log10(100) = 40 dB, phase angle zero for all frequencies.

5.1.2 Integrator Plus Gain

We come across this quite a lot in control-systems

$$G(s) = \frac{K}{s}, G(j\omega) = \frac{K}{j\omega}.$$

$LM = 20\,Log10\left|\frac{K}{\omega}\right|$, Phase is $-90°$ due to the imaginary term on the denominator.

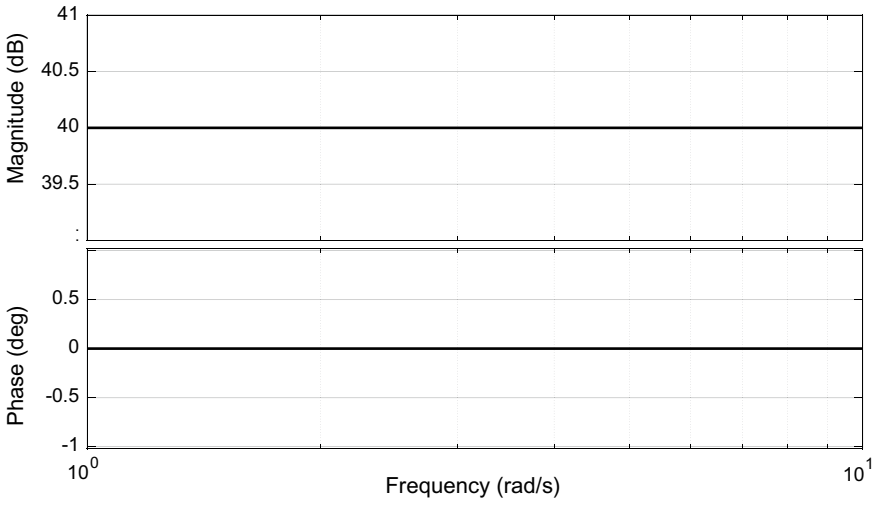


Fig. 5.3 Pure gain Bode-Plot

In Fig. 5.4 we see that the frequency axis is logarithmic and scaled in powers of 10. A power of 10 we call a *decade*. Hence moving from say 1–10 rad/s we say this is a decade, or 100–1000 etc. We cannot go down to dc since the gain of an integrator is infinite at dc. We usually start at some low convenient frequency. Of special interest is when we have $\mathrm{LM} = 20\mathrm{Log}10\left|\frac{\mathrm{K}}{\omega}\right| = 0$. That is to say, at what frequency does the magnitude plot hit 0 dB? We must have that at 0 dB (unity gain) $\mathrm{K} = \omega$. Also we have

$$\begin{aligned} \mathrm{LM} &= 20\mathrm{Log}10\left|\frac{\mathrm{K}}{\omega}\right| \\ &= 20\mathrm{Log}10\mathrm{K} - 20\mathrm{Log}10\omega \end{aligned} \tag{5.1}$$

Examining $-20\mathrm{Log}10\omega$, we can see that for any decade of frequency the gain goes down -20 dB. For example suppose $-20\mathrm{Log}10(10) = -20$ dB, $-20\mathrm{Log}10(100) = -40$ dB. The magnitude plot is therefore a straight line with slope -20 dB/decade and the phase is a constant phase of $-90°$ *at all frequencies*. The slope of the magnitude plot can also be expressed differently. In music theory, an *octave* is the interval between a pitch at one frequency and at double its frequency. So going from 1 to 2 Hz or 1 kHz to 2kHz are both octaves. Likewise in rad/s, $-20\mathrm{Log}10\omega$ if we go from 1 to 2 rad/s the magnitude drops by 0 $-20\log10(2)$ or -6 dB. Hence the slope is -6 dB per octave or -20 dB/decade. Take your pick though we usually use decades.

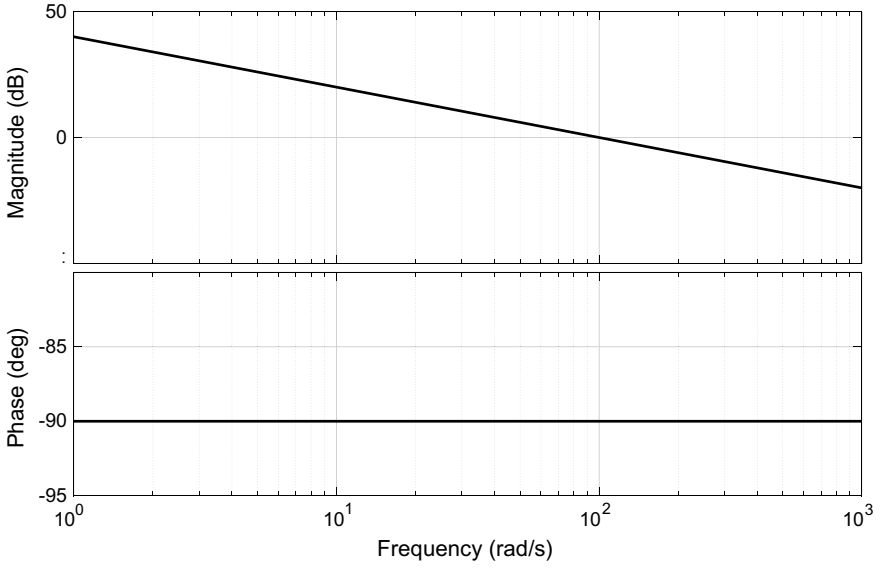


Fig. 5.4 Bode plot of integrator plus gain

5.1.3 First-Order System

It is with the first-order system that we see the beauty of the method. For magnitude at least we can make a few approximations.

$$\begin{aligned} G(s) &= \frac{1}{1+sT}, G(j\omega) = \frac{1}{1+j\omega T} \\ |G(j\omega)| &= \frac{1}{\sqrt{1+(\omega T)^2}} \end{aligned} \tag{5.2}$$

Take log-magnitude.

$$LM = 20Log10|G(j\omega)| = -20Log10\sqrt{1+(\omega T)^2} \tag{5.3}$$

At low frequencies we have $(\omega T)^2 \ll 1$ and at high-frequencies we have $(\omega T)^2 \gg 1$. Therefore we can simplify (5.3). At low frequencies LM = 0 dB and at high frequencies we have $LM = -20Log10(\omega T)$. These are two straight lines which intersect when $\omega = 1/T$. The intersection frequency is often known as the corner frequency though it has other names too such as the −3 dB frequency, break frequency or cutoff frequency. It is called the −3 dB frequency because if we substitute $\omega = 1/T$ into (5.3) we get $LM = -20Log10\sqrt{1+1} = -3.01$ dB. The gain therefore stays at 0 dB (unity) until a frequency is reached whereby it drops by −3 dB and keeps dropping thereafter at a slope of −20 dB/decade (or −6 dB/octave). The phase is calculated directly as

$$\phi(\omega) = -\tan^{-1}\omega T \tag{5.4}$$

So we can summarise that the *maximum* phase-shift that a first-order system gives us is −90°. The output lags behind the input signal in phase and is always lagging behind. This is why sometimes the system is known as a simple lag. The single pole (first-order system) has a magnitude that attenuates with frequency beginning at the corner frequency. It has an attenuation of −3 dB at the corner frequency and goes on attenuating at a rate −20 dB/decade. So for example if the corner frequency is 1 rad/s as in Fig. 5.5, the gain will be −20 dB down a decade later at 10 rad/s. Another important landmark is on the phase-plot at the corner frequency which we name $\omega_c = 1/T$. If we substitute this value of frequency into (5.4) we get

$$\phi(\omega_c) = -\tan^{-1} 1 = -45^o \tag{5.5}$$

Put this together, and for the magnitude we can approximate it as two straight lines on the frequency plot. One line is termed the low-frequency asymptote and

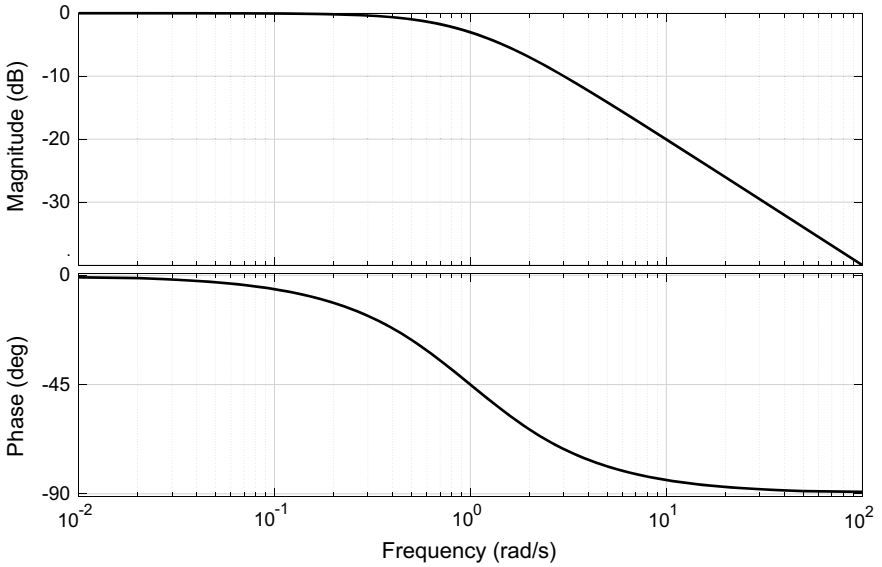


Fig. 5.5 Bode-plot for first-order system, T = 1

goes through unity gain (0 dB) and the second has a slope of −20 dB/decade intersecting the frequency $\omega = \omega_c$. This is illustrated in Fig. 5.6.

We leave the phase as it is and can deduce the phase from the magnitude for most cases, though there are exceptions when there are time-delays or right-half plane zeros. This is not that common though can happen in some applications.

For the integrator plus gain we only have one asymptote and the phase is −90° at all frequencies and so this is easy to sketch (Fig. 5.7).

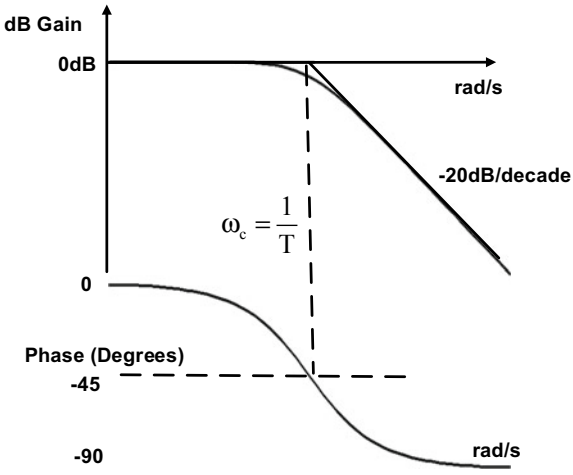


Fig. 5.6 Asymptotic first-order system Bode-plot

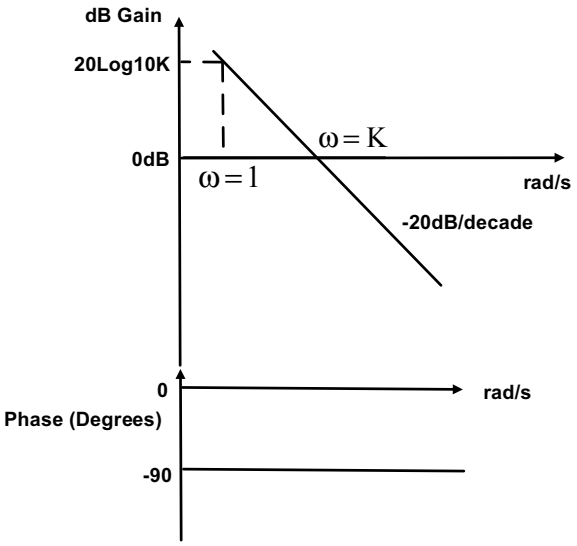


Fig. 5.7 Bode-plot of combined Integrator and gain K

5.2 Composite Bode-Plots

Before looking at the case of second-order systems, we can approach it by cascading two first-order systems. Take the integrator and first-order system. This is like our motor plus load transfer function where angular position is measured as the output.

$$G(s) = \frac{K}{s(1+sT)} \tag{5.6}$$

The beauty of the Bode method is that because we are dealing with dB (logarithms), anything multiplied in the s- or frequency-domain (convolution), becomes additive in terms of its log-magnitude.

$$\begin{aligned} &20\text{Log}|G_1(j\omega)||G_2(j\omega)| \\ &= 20\text{Log}|G_1(j\omega)| + 20\text{Log}|G_2(j\omega)| \end{aligned} \tag{5.7}$$

Consequently we need only add asymptotes to get higher-order plots. The phase adds too since multiplication of complex-numbers results in addition of phase. Of course much of the phase may be negative since poles are on the denominator of an expression and not the numerator. Clarifying this we know that the phase of $a + jb$ = minus phase of $\frac{1}{a+jb}$ or $-\tan^{-1}\frac{b}{a}$.

For (5.6) we can draw the magnitude directly by drawing the integrator-gain combination first. It will have a slope of −20 dB/decade. However, if we then add the first-order system then at low frequencies its asymptote has a value 0 dB so

makes no change to the integrator since we add 0 dB. However, when we reach the corner frequency $\omega_c = 1/T$ we will now have two slopes of −20 dB/decade to contend with. The composite slope must now be −40 dB/decade at frequencies beyond $\omega_c = 1/T$ (Fig. 5.8).

Note that the phase is identical to that of the first-order system but has been shifted by −90° downwards. Usually we do not bother drawing in the 3 dB frequency. We conclude that the maximum phase-shift for two poles is −180°, the maximum roll-off is −40 dB/decade. When plotting any order of Bode-plot, it is best to have the transfer-functions in the form $G(s) = \frac{K}{s(1+sT_1)(1+sT_2)(1+sT_3)}$ etc. *and not* in the form $G(s) = \frac{K}{s(s+a)(s+b)(s+c)}$ or mistakes are easily made. When zeros appear, the Bode-plot just gets flipped. Of course a system such as $G(s) = (1+sT)$ is not physically realisable unless it comes attached to a pole! However, to illustrate the effect of a zero we plot the Bode-plot of a single zero (Fig. 5.9).

We can see it is just the same as the Bode-plot of $G(s) = \frac{1}{(1+sT)}$ but flipped through the vertical axis. The corner frequency is still in the same place but the second asymptote now goes up at 20 dB/decade instead of down. If such a slope appears with a pole which has a negative similar slope, then at high-frequencies the two slopes will cancel each other out and the slope will go flat. So a +20 dB/decade cancels with a −20 dB/decade. For say one zero and two poles we would get +20 and −40 dB/decade at high frequencies and this would give us a net -20 dB/decade

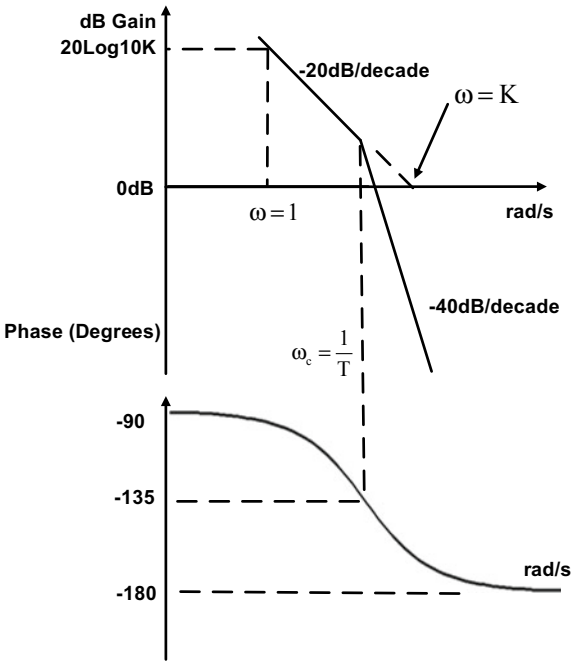


Fig. 5.8 Asymptotic composite Bode-plot in combined integrator-gain and first-order

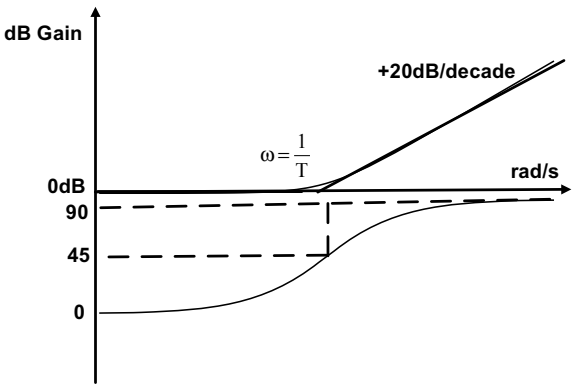


Fig. 5.9 Bode-plot of a single zero

and so on. The phase is just positive so goes to +90° through +45° at the corner frequency. Now consider a more difficult example with a zero on the numerator.

Example 5.1 Frequency-response with poles and zero.

Plot the bode-plot of $G(s) = \frac{10^5(S+1000)}{(S+10)(S+100)(S+10{,}000)}$.

The first thing we notice about this example is that it is in the wrong mathematical form to draw. So we re-arrange the transfer-function so that it always has leading "1" terms as the poles and zeros as follows $G(s) = \frac{10^8(1+S/1000)}{10^7(1+S/10)(1+S/100)(1+S/10{,}000)} = \frac{10(1+S/1000)}{(1+S/10)(1+S/100)(1+S/10{,}000)}$

Now we see three pole corner frequencies at 10,100 and 10,000 rad/s and one zero at 1000 rad/s. So we have three −20 dB/decade roll-offs and one +20 dB/decade rise. We first note that as there is no integrator we can put s = 0 and show that the dc gain is 10 or 20 dB. At high frequencies the roll-off must be 2nd order or −40 dB/decade since 3 × −20 dB/decade + 1 × 20 dB/decade = −40 dB/decade. We can make an attempt at plotting this but the phase will never be precise. What we can say about the phase is that it must start at zero as there are no integrators, and end at −180° which corresponds to the −40 dB/decade at high frequencies. So knowing the key frequencies and characteristics is what is most important here as we can now use MATLAB to get us the exact plot. The code we use is as follows. We create three systems and convolve them together. We take the zero with the first pole though this is optional, we could take the zero with any of the poles (Fig. 5.10).

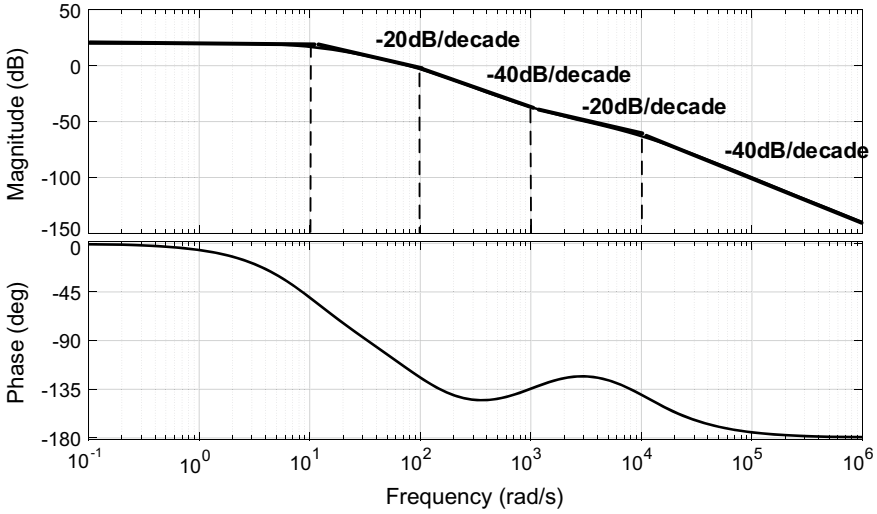


Fig. 5.10 Bode-plot of third-order system with zero

MATLAB Code 5.1

```
% create linear system g1
num=1e8*[1e-3  1];
den=1e7*[0.1 1];

% create linear system g2
g1=tf(num,den)
num=[1];
den=[0.01 1];
g2=tf(num,den)

% create linear system g3
num=[1];
den=[1e-4 1];
g3=tf(num,den)
% convolve them into a system g
g=g1*g2*g3
%Plot the Bode-plot
bode(g)
grid
```

Looking at the phase we see that it is on its way going negative when it then goes positive when the frequency is 1000 rad/s. This is the effect of a zero, it causes positive-going phase or sometimes we call this *phase-advance*. The last pole however wins in the end and pulls the phase back down again towards −180°. Clearly, although we have a third-order system it behaves like a second-order at high frequencies.

5.3 Bode-Plots with Complex Poles

We now analyse a second order system of the form

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{5.8}$$

Substitute $s = j\omega$ into (5.8) and get

$$G(j\omega) = \frac{\omega_n^2}{2\zeta\omega_n j\omega + \omega_n^2 - \omega^2} \tag{5.9}$$

$$= \frac{1}{\left(1 - \left(\frac{\omega}{\omega_n}\right)^2\right) + 2\zeta j\left(\frac{\omega}{\omega_n}\right)} \tag{5.10}$$

Take log-magnitude

$$20\log 10|G(j\omega)| = -20\log 10\left|\sqrt{\left(1 - \left(\frac{\omega}{\omega_n}\right)^2\right)^2 + 4\zeta^2\left(\frac{\omega}{\omega_n}\right)^2}\right| \tag{5.11}$$

$$= -10\log 10\left|1 - 2\left(1 - 2\zeta^2\right)\left(\frac{\omega}{\omega_n}\right)^2 + \left(\frac{\omega}{\omega_n}\right)^4\right| \tag{5.12}$$

The phase is found from (5.8)

$$\phi(\omega) = -\tan^{-1}\left(\frac{2\zeta\left(\frac{\omega}{\omega_n}\right)}{\left(1 - \left(\frac{\omega}{\omega_n}\right)^2\right)}\right) \tag{5.13}$$

At low frequencies we can approximate the log magnitude since $\omega \ll \omega_n$ and $\left(\frac{\omega}{\omega_n}\right)^2 \ll 1$ giving a low-frequency asymptote $= -10\log 10\left|1 - 2\left(1 - 2\zeta^2\right)\left(\frac{\omega}{\omega_n}\right)^2 + \left(\frac{\omega}{\omega_n}\right)^4\right| \to -10\log 10|1| = 0\,\text{dB}$. This is the same as for a first-order system. When $\left(\frac{\omega}{\omega_n}\right)^2 \gg 1$ and log magnitude becomes $20\log 10|G(j\omega)| = -10\log 10\left|\left(\frac{\omega}{\omega_n}\right)^4 + 4\zeta^2\left(\frac{\omega}{\omega_n}\right)^2\right|$. Now

$$-10\log 10\left|\left(\frac{\omega}{\omega_n}\right)^4+4\zeta^2\left(\frac{\omega}{\omega_n}\right)^2\right|\approx -10\log 10\left(\frac{\omega}{\omega_n}\right)^4=-40\log 10\left(\frac{\omega}{\omega_n}\right) \quad (5.14)$$

This is an asymptote at −40 dB/decade and probably what we would expect since two first-order systems in cascade will give a net slope at high frequencies also of −40 dB/decade.

We plot a family of second-order systems for varying damping-factor. A key frequency however is when $\omega=\omega_n$, whereby (5.14) is 0 dB and meets with the 0 dB asymptote at low frequencies. This is therefore the corner frequency for second-order systems (Fig. 5.11).

From (5.13) when $\left(\frac{\omega}{\omega_n}\right)=1$ the phase is −90°. At high frequencies the phase −180°. Since a second-order is two firsts, it is not surprising that the phase is two lots of −90° at high frequency.

It is easy to be fooled into thinking that when $\left(\frac{\omega}{\omega_n}\right)=1$, (i.e. $\omega=\omega_n$) maximum overshoot in the frequency-domain occurs. However, by differentiating the expression for magnitude $\frac{1}{\sqrt{\left(1-\left(\frac{\omega}{\omega_n}\right)^2\right)^2+4\zeta^2\left(\frac{\omega}{\omega_n}\right)^2}}$ we find that the peak in the magnitude is given by

$$M_r=\frac{1}{2\zeta\sqrt{1-\zeta^2}} \quad (5.15)$$

which only holds for $\zeta<1/\sqrt{2}$. This occurs at a frequency commonly known as the *resonant frequency*

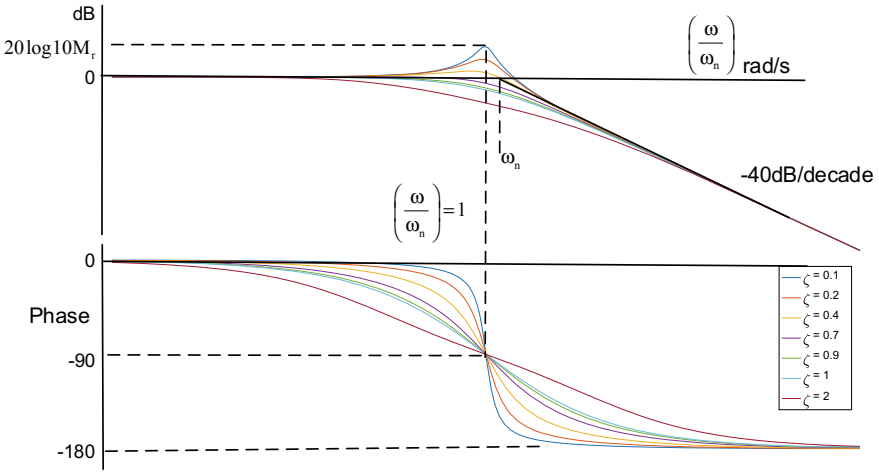


Fig. 5.11 Family of Bode-plots for second-order systems with varying damping-factors

$$\omega_r = \omega_n \sqrt{1 - 2\zeta^2} \tag{5.16}$$

For $\zeta > 1/\sqrt{2}$ the maximum of the curve is unity or 0 dB. Of course for small damping factor the resonant frequency is very close to the undamped natural frequency in any case.

Example 5.2

Complex Poles

Let us look closer at the complex-poles case and consider a system with transfer-function $G(s) = \frac{2}{S^2+S+2}$, which has damping factor $\zeta = 0.35$ If we draw a 3D plot but examine through where the real part of s is zero (like slicing a cake) we get the result of Fig. 5.12.

The frequency-response (or Bode-plot) looks at the face when sigma, the real part of s is zero. The pole is further up the hill if viewed in 3D in the direction of the viewer. So the pole-zero diagram is the 3d plot looking from above as shown in a previous example (Fig. 3.11) and the frequency-response is the slice through the middle looking at that face along the imaginary axis.

The complex-poles example usually finds application in control-theory when a system has some form of structural-resonance which happens with electro-mechanical systems. Usually a structural resonance does not come alone and has many other resonant terms associated with it. It represents the upper limit of frequency we can consider when designing control-systems.

Example 5.3

Bode-Plot of Fourth-Order System

Let us put this all together and plot the Bode-plot of a fourth-order system. Consider the system $G(s) = \frac{394\times10^7}{S(1+1.59\times10^{-3}S)(S^2+6.28\times10^3S+3.94\times10^7)}$, plot the Bode-plot of the system.

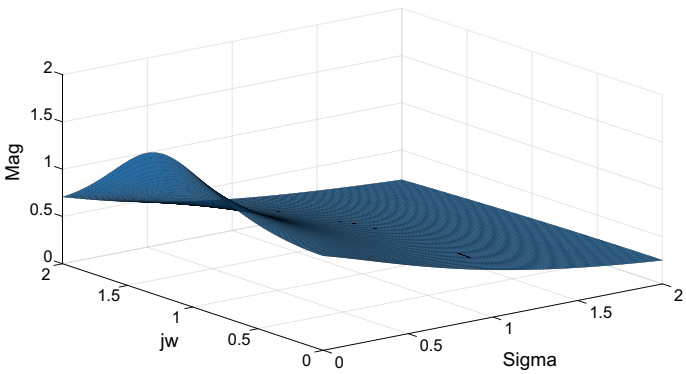


Fig. 5.12 Frequency-response as seen from 3d slice through real(s) = 0

For such a system we usually use MATLAB to plot the Bode-plot. It is however useful to know what the result will look like before opening MATLAB. Like with arithmetic, we all use calculators nowadays but it is at least useful to know what range of numbers the answer should lie within in case an error is made. This system has an integrator-gain (the gain is usually lumped together with any integrator as it is easier to sketch the plot), a first-order pole and a second-order pole. Because it has an integrator, we cannot calculate its dc gain when s=0 since it is infinite. We first make sure that it is in the correct format—which it is. All terms are in the standard form and we need not simplify further. The first time-constant is 1.59 ms which is easily calculated to be $\omega_c = 1/T = 628.9$ rad/s (or 100 Hz). The undamped natural frequency is $\omega_n = \sqrt{3.94 \times 10^7} = 6276.9$ rad/s or 1 kHz. The damping factor is found because $2\zeta\omega_n = 6.28 \times 10^3$ giving $\zeta = 0.5$. We know from Fig. 5.11 and from (5.15) that $M_r = \frac{1}{2\zeta\sqrt{1-\zeta^2}} = 1.154$, indicating only a small resonance peak in the frequency-domain. Finally, there are no zeros but 4 poles. Four poles means a final slope at high frequencies of −20 dB/decade × 4 or −80 dB/decade and a phase-shift that must end up at −90° × 4 or −360°. The phase—shift at low frequencies must be −90° due to the integrator.

Summarising for this example

(a) dB must start at −20 dB/decade due to the integrator and finish at –80dB/decade
(b) Phase must start at −90° and finish at −360°
(c) Resonant frequency is 1 kHz with damping factor 0.5. Peak in the frequency overshoot is 1.154 or 1.244 dB.

Using MATLAB we draw the plot as given in Fig. 5.13. It has all the characteristics we predicted. Note the sharp drop in phase caused by the second-order underdamped system. This kind of phase-shift is not desirable when applying negative-feedback as will be discussed later.

5.4 Some Commonly Met Bode-Plots

We use Bode-plots most of the time in classical control-system design. The main reason is because we can sketch the magnitude very easily using asymptotic approximations. The phase is not so easy to plot but as shown in the previous example we can make good estimates and by using tools such as MATLAB we can easily obtain an accurate plot. There are some very commonly met electronic systems we use in control-systems design and it is good to familiarise yourself with the circuit, transfer function and Bode-plot.

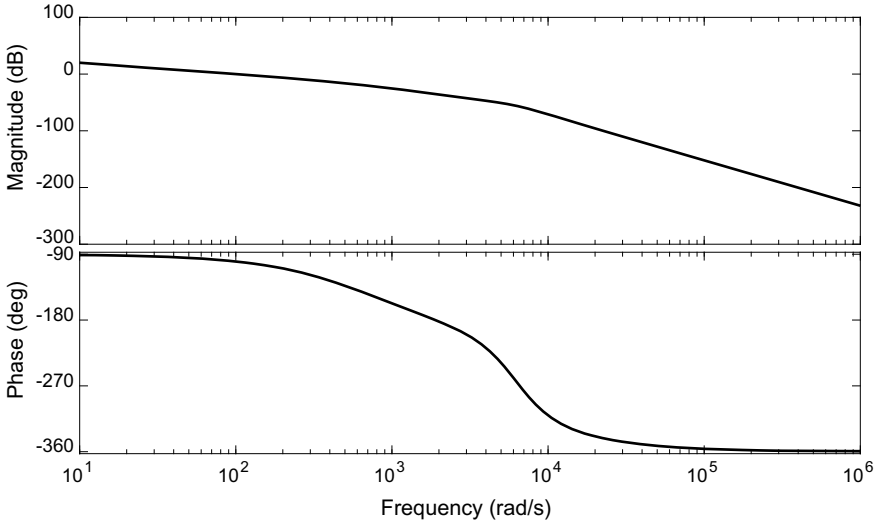


Fig. 5.13 Bode-plot of fourth-order system

5.4.1 Phase-Lead Compensator (Passive)

This is called a phase-lead or phase-advance circuit. We find the transfer-function by impedance division from Fig. 5.14.

$$G(s) = \frac{R_2}{R_2 + \frac{R_1}{1 + sCR_1}} \tag{5.17}$$

After simplification we can write this in a more familiar way

$$G(s) = \alpha\left(\frac{1 + sT_1}{1 + sT_2}\right) \tag{5.18}$$

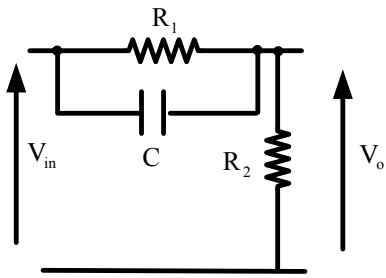


Fig. 5.14 Phase-lead (passive version) circuit

where $\alpha = \frac{R_2}{R_1+R_2}$, $T_1 = CR_1, T_2 = CR_1//R_2$. Here $R_1//R_2 = \frac{R_1R_2}{R_1+R_2}$.

This circuit (from (5.17)) has a dc gain when s = 0 which is $G(0) = \alpha = \frac{R_2}{R_1+R_2}$ and a high-frequency gain when s is infinite given by $G(\infty) = \alpha\frac{T_1}{T_2} = 1$. Note, a trick in finding the high-frequency gain is to first divide numerator and denominator by s as follows.

$G(s) = \alpha\left(\frac{1+sT_1}{1+sT_2}\right) = \alpha\left(\frac{1/s+T_1}{1/s+T_2}\right)$ and then let s be large giving us our result (Fig. 5.15).

The reason why this circuit is important is due to its phase characteristic. The phase goes to a maximum. Positive going phase is an important thing to have in a control-system and if you don't have enough you can add your own! This is what this circuit does. It also changes the gain characteristic but we have to live with this and allow for it. There are a few calculations of relevance. First define the upper-corner frequency as $\omega_h = \frac{1}{T_2}$ which is due to a pole, and the lower corner frequency as $\omega_l = \frac{1}{T_1}$ which is due to a zero. Then the ratio between them defines the span of max frequency to minimum and we term this the *span-ratio p*.

$$p = \frac{\omega_h}{\omega_\ell} = \frac{T_1}{T_2} \tag{5.19}$$

The span-ratio is not a word heard much in academic circles but is used a lot in industry. We will use this quite a lot in later design work. The maximum amount of phase-lead can be calculate as follows. First find an expression for phase from (5.18)

$$\phi(\omega) = \tan^{-1}(\omega T_1) - \tan^{-1}(\omega T_2) \tag{5.20}$$

Differentiating we find the value of frequency for maximum phase $\frac{d\phi(\omega)}{d\omega} = 0$ as

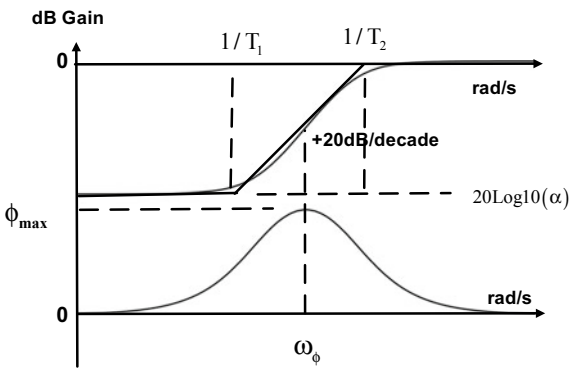


Fig. 5.15 Bode = plot of passive phase-lead

$$\omega_\phi = \sqrt{\omega_h \omega_\ell} \tag{5.21}$$

Equation (5.21) represents the *geometric root* of the upper and lower corner frequencies. It is the frequency at which maximum phase-shift occurs. Substituting this value back into the phase expression (5.21) we can show that the maximum phase is given by

$$\phi_{max} = \sin^{-1}\left(\frac{p-1}{p+1}\right) \tag{5.22}$$

The interesting thing about (5.20) is that we can derive values of maximum phase for various common span ratios. Some are easy to remember (Table 5.1).

Usually in control-systems design a span of p = 10 is used which gives us 55° phase-advance. The niceties of this is covered in a later chapter.

5.4.2 Phase-Lead Compensator (Active)

The active version of the phase-lead (or phase-advance) has similar properties to the passive version except it has a low output impedance and an extra gain term can be set for convenience (Fig. 5.16).

The transfer function is found by first finding the Laplace impedance of the input impedance term. It can be found from

$$z_2(s) = R_2 // \left(R_1 + \frac{1}{sC}\right) \tag{5.23}$$

$$\begin{aligned} &= \frac{R_2\left(R_1 + \frac{1}{sC}\right)}{R_1 + R_2 + \frac{1}{sC}} \\ &= \frac{R_2(1 + sCR_1)}{sC(R_1 + R_2) + 1} \end{aligned} \tag{5.24}$$

Table 5.1 Maximum phase-lead and span ratio p

Span-ratio p	$\phi_{max} = \sin^{-1}\left(\frac{p-1}{p+1}\right)$ degrees (approx.)
2	20
3	30
4	37
5	42
8	51
9	53
10	55

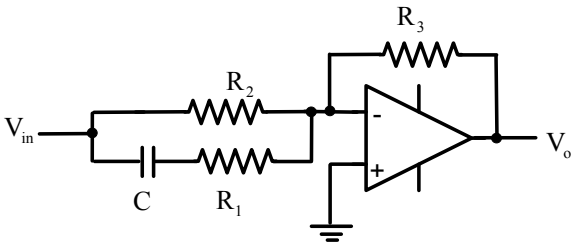


Fig. 5.16 Active phase-lead circuit

The feedback impedance is just $z_1(s) = R_3$ and so the transfer function becomes the ratio of these with an inversion. $\frac{V_o}{V_{in}}(s) = -\frac{z_1}{z_2}(s)$ results in

$$\frac{V_o}{V_{in}}(s) = -\left(\frac{R_3}{R_2}\right)\frac{1+sC(R_1+R_2)}{1+sCR_1}$$

Define $K = \left(\frac{R_3}{R_2}\right)$, $T_1 = C(R_1+R_2)$, $T_2 = CR_1$, $T_1 > T_2$ and ignore the change of sign since we can compensate easily for this in a control-system.

$$G(s) = K\frac{1+sT_1}{1+sT_2} \tag{5.25}$$

This has exactly the same properties as the passive version except the dc gain K can be set to be anything we wish. The time-constants are of course defined differently and the span-ratio is

$p = \frac{T_1}{T_2} = \frac{(R_1+R_2)}{R_1}$ which is more or less just a ratio of resistor values and easy to change. Although less common nowadays due to the advent of digital-control systems, phase-advances like this were common in many analogue systems and design is achieved quickly by a swift change in resistor values to change span-ratio. For example we can say that roughly for a span of p = 10 that $R_2 = 9R_1$ or just make it ten-times bigger and have a span of p = 11. Then change R_3 to adjust overall gain.

5.4.3 Three Classes of Integrator

We consider three different integration arrangements. The circuits are shown in Fig. 5.17. In all cases for the analysis we ignore the inversion as it does not effect the dynamic properties.

(a) **Pure Integrator with gain**

This has already been considered in Sect. 2.4 and has a transfer-function

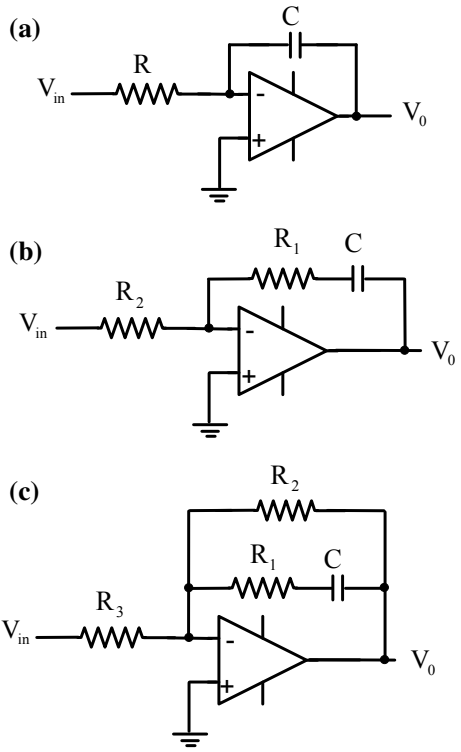


Fig. 5.17 Three different integrator related circuits. **a** Pure Integrator, **b** bandwidth-limited integrator, **c** phase-lag circuit

$$G(s) = \frac{K}{s} \tag{5.26}$$

where $K = \frac{1}{CR}$. This is an integrator with gain K and has already been considered in Fig. 5.7.

(b) **Bandwidth-limited integrator**

The transfer-function is found from

$$G(s) = \frac{R_1 + \frac{1}{sC}}{R_2} \tag{5.27}$$

$$G(s) = \frac{K(1 + sT)}{s} \tag{5.28}$$

where $K = \frac{1}{CR_2}$ and $T = CR_1$.

The Bode-plot is an integrator plus gain which has a zero at a frequency $1/T$. The zero provides +20 dB/decade of gain which cancels the integrator slope and

makes it horizontal at this frequency. The phase must start at low frequencies at phase −90° and end up at zero.

The Bode-plot of the circuit in Fig. 5.17b is shown in Fig. 5.18.

This circuit therefore provides an integrator but only up to a frequency $\omega_c = 1/T\,\text{rad/s}$.

(c) **Phase-lag circuit**

The last of the trio of integrators is essential the opposite of the phase-lead. It could be called a poor-mans integrator. An analysis yields.

$$G(s) = K\frac{1+sT_1}{1+sT_2} \tag{5.29}$$

where $K = \frac{R_2}{R_3}$, $T_1 = CR_1$, $T_2 = C(R_1 + R_2)$, $T_1 < T_2$. The dc gain is $G(0) = K$ and the high-frequency gain is $G(\infty) = K\frac{T_1}{T_2}$. The phase which characterises this circuit bears a close resemblance to the phase-lead except the phase is negative and not positive. This is illustrated in Fig. 5.19 for K = 1.

Unlike positive-going phase, negative phase is not much benefit to us in control-systems. However, we can see from Fig. 5.19 that the gain behaves like an integrator over a limited frequency range. This we can make use of provided the phase is away from sensitive frequency ranges where the overall gain crosses unity (see next section). Compare with Fig. 5.18 which is also a limited form of integration and see that in the case of $G(s) = \frac{K(1+sT)}{s}$ that the phase is −90° at low frequencies whereas it is zero and just a negative blip and goes back to zero for the phase-lag case. There are subtle differences that a designer can exploit. There is a passive form of the phase-lag circuit which we show below without analysis (Fig. 5.20).

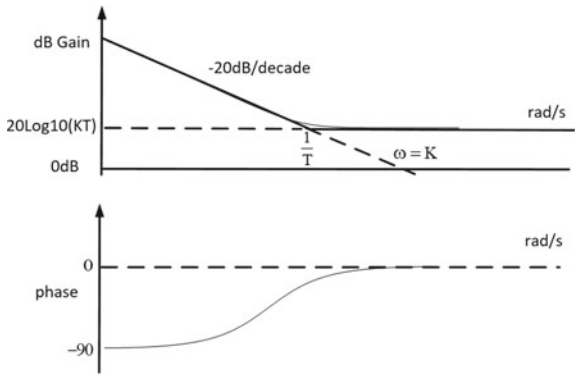


Fig. 5.18 Bode-plot of bandwidth-limited integrator

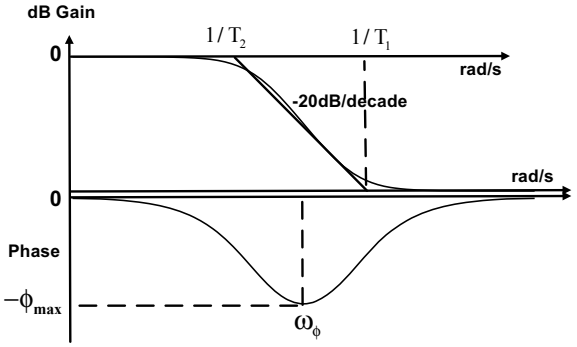


Fig. 5.19 Active phase-lag circuit

5.4.4 *Lag-Lead Circuit*

Finally we present the circuit which combines the properties of both the phase-lead and phase-lag. It is not surprising it is known as a lag-lead circuit (Fig. 5.21).

It's transfer-function has the form

$$G(s) = K\left(\frac{1+sT_1}{1+sT_2}\right)\left(\frac{1+sT_3}{1+sT_4}\right) \tag{5.30}$$

where $K = \frac{R_2}{R_3}$, $T_1 = C_2(R_3 + R_4)$, $T_2 = C_2R_4$, $T_3 = C_1R_1$, $T_4 = C_1(R_1 + R_2)$. To maintain the lag-lead properties we must ensure $T_4 > T_3 > T_2 > T_1$. Its Bode-plot is shown in Fig. 5.22.

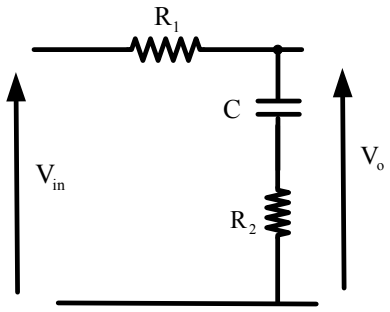


Fig. 5.20 Passive form of phase-lag circuit

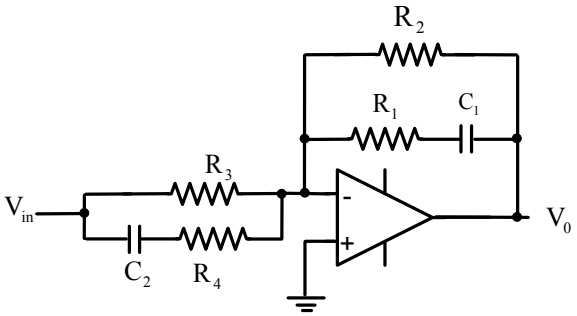


Fig. 5.21 Lag-lead circuit

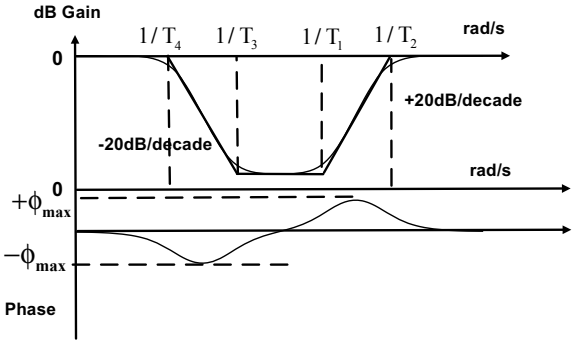


Fig. 5.22 Lag-lead Bode-plot

5.4.5 *Leaky Integrator*

The leaky integrator is just a first-order low-pass filter with corner frequency placed at a very low value so that it is still an approximate integrator. It is used to damped an integrator when designing controllers. It is less severe than a pure integrator. A pure integrator responds to any dc at its input, a leaky one just filters. Usually a high-value resistance is used (Fig. 5.23).

Finally a look at differentiators. Despite differentiators appearing in many books, such a circuit, although it can be made is not practical for a few reasons. Firstly it will oscillate and secondly a differentiator would in theory have a slope of = 20 dB/decade that rises up until the bandwidth of the whole system is met, amplifying noise and disturbances. Differentiators are not good on their own. Instead we prefer to roll off the transfer function so it goes up at 20 dB/decade and then falls flat or even starts going down again just to be extra safe (Fig. 5.24).

The non-ideal differentiator has a transfer-function given by

$$G(s) = \frac{sCR}{1 + sCR_1} \tag{5.31}$$

$$= K\frac{s}{1 + sT} \tag{5.32}$$

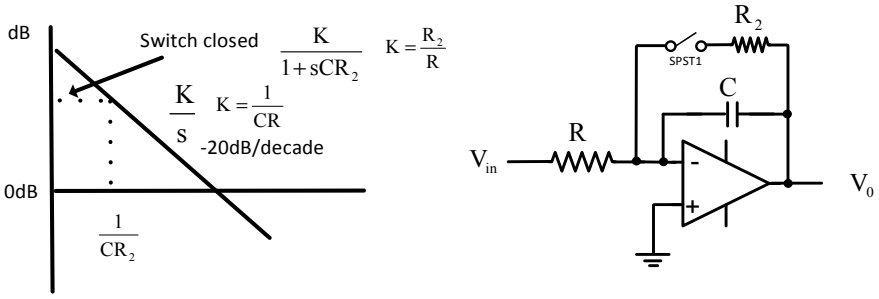


Fig. 5.23 Leaky integrator and pure integrator

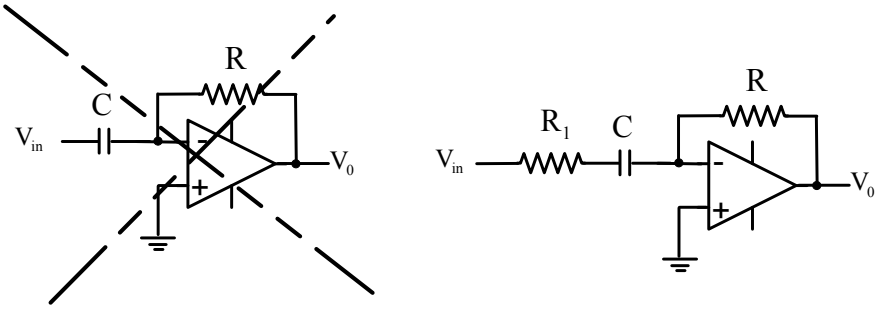


Fig. 5.24 Ideal versus non-ideal differentiator

At dc it has a gain of zero and at high-frequencies the gain becomes $K/T = \frac{R}{R_1}$ since the capacitor becomes a short-circuit. The phase must start at +90° and go towards zero at high-frequencies. We can either calculate this directly or realise that a pure differentiator has +90° at all frequencies, but the pole drags the phase down when added, because it has −90° phase-shift at high-frequencies.

5.5 Non Minimum-Phase Systems

A system is termed *minimum-phase* if all of its zeros lie in the left-half of the s-plane. If we make this assumption then just knowing the magnitude of a Bode-plot is enough for us to estimate the phase. For example, consider the following transfer-function.

$$G(s) = \frac{(s+1000)}{s(s+100)(s+10{,}000)} \tag{5.33}$$

We know that the log-magnitude of this is composed of the following sections or sub-systems

(a) A pure integrator slope −20 dB/decade
(b) A pole at 100 rad/s slope −20 dB/decade
(c) A pole at 10,000 rad/s slope −20 dB/decade
(d) A zero at 1000 rad/s

We know the order that the poles and zero come into play i.e. the frequency. So we have the integrator first gives a slope of −20 dB/decade, followed by the 100 rad/s pole gives a slope of −40 dB/decade. Then a zero comes into play at 1000 rad/s so the slope must return to –20 dB/decade. Finally at 10,000 rad/s the slope must fall down again at −40 dB/decade because of the pole at that frequency. If the system is minimum phase then we can say that the phase must begin at −90° and move down towards −180° and then back up due to the zero and finally back down again to −180°. A rule of thumb is to take the number of poles n_p from the number of zeros n_z and multiply by −90° to find the phase at high frequencies. Write this as $-90^o\left(n_p - n_z\right)$ Likewise the magnitude slope is $-20\left(n_p - n_z\right)$ db/decade at high frequencies. These are uniquely determined. Given the magnitude, the phase follows. However, for non minimum phase systems or systems with a time-delay, the magnitude does not result in a unique phase. The circuit shown in Fig. 5.25 has transfer-function

$$G(s) = \frac{sT - 1}{sT + 1} \tag{5.34}$$

with $T = CR$.

We can see that this system as a zero at $s = 1/T$ and a pole at $s = -1/T$. The pole is in the left-half plane and so the system is definitely stable. The zero in the right-half plane makes the system non minimum phase. The name non minimum comes from the fact that by calculating the phase (we need to be careful of the

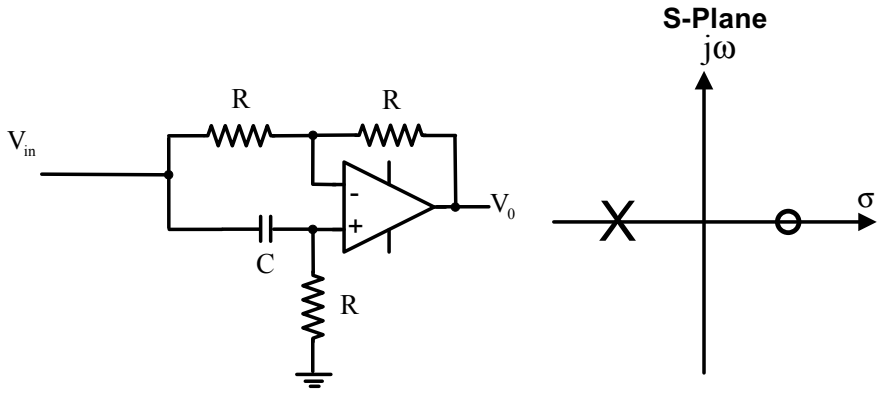


Fig. 5.25 All-pass circuit and its pole-zero plot

quadrant when calculating the arctan in this case as $-1+j\omega T$ is in the second quadrant) we get

$$\phi(\omega) = -\tan^{-1}\omega T - \tan^{-1}\omega T + \pi \tag{5.35}$$

or

$$\phi(s) = -2\tan\omega T + \pi \tag{5.36}$$

So the phase is zero degrees at high frequencies and not what might be expected (zero degrees for a single minimum phase zero and a pole). The phase is 180° at dc. Hence the phase is not at its expected minimum value. If we calculate the magnitude we find

$$|G(j\omega)| = \left|\frac{j\omega T - 1}{j\omega T + 1}\right| \tag{5.37}$$

which is one or 0 dB (Fig. 5.26).

Note, a transfer function of a similar all-pass variety is $G(s) = \frac{1-sT}{1+sT}$ and its phase starts at zero and goes to −180°. This is essentially the same effect but with a minus sign. Often it is easier to use this form and assume there is another inversion. The circuit shown is used to phase-shift a signal without effecting its amplitude, but sometimes in process-control systems non minimum-phase problems arise naturally. The important thing to recognise is the miss-match between magnitude and phase. We can also use the all-pass as a way of deducing the Bode-plot of a non minimum-phase transfer function. For example consider the following non minimum-phase transfer function.

$$G(s) = \frac{100(s-100)}{(s+10)(s+1000)} \tag{5.38}$$

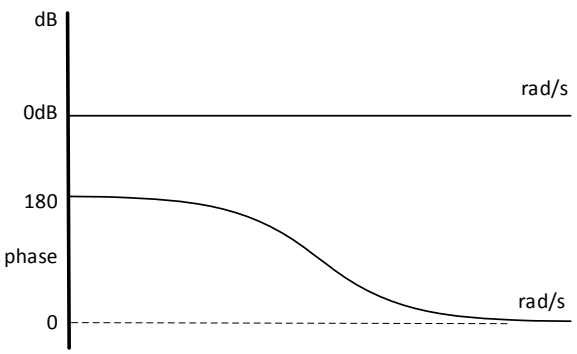


Fig. 5.26 Bode-plot of all-pass circuit

We usually work in standard form for transfer-functions but for this case it is at first easier to leave it as it is. We introduce an all-pass transfer-function

$$G_a(s) = \frac{(s - 100)}{(s + 100)} \tag{5.39}$$

We then re-write (5.38) incorporating (5.39)

$$G(s) = G_m(s)G_a(s) \tag{5.40}$$

where the minimum-phase part is

$$G_m(s) = \frac{100(s + 100)}{(s + 10)(s + 1000)} \tag{5.41}$$

Now we will consider the minimum-phase part on its own and add it the all-pass later. The all-pass will only effect the phase and not the magnitude. Writing Eq. (5.41) in standard form

$$G_m(s) = \frac{(1 + s/100)}{(1 + s/10)(1 + s/1000)} \tag{5.42}$$

So (5.42) is easily plotted. It has unity gain at dc, two poles at 10 and 100 rad/s and a zero at 100 rad/s. The Bode-plot should have a roll-off first at 10 rad/s or

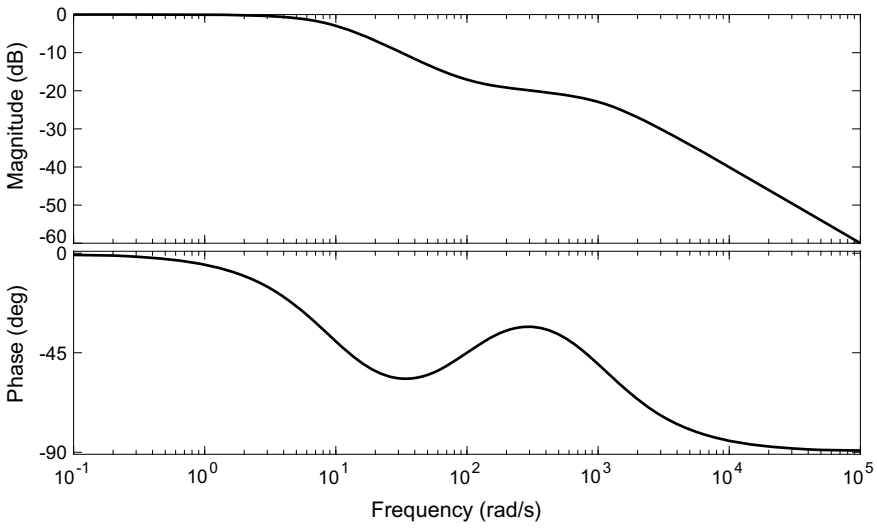


Fig. 5.27 Bode-plot of minimum-phase system

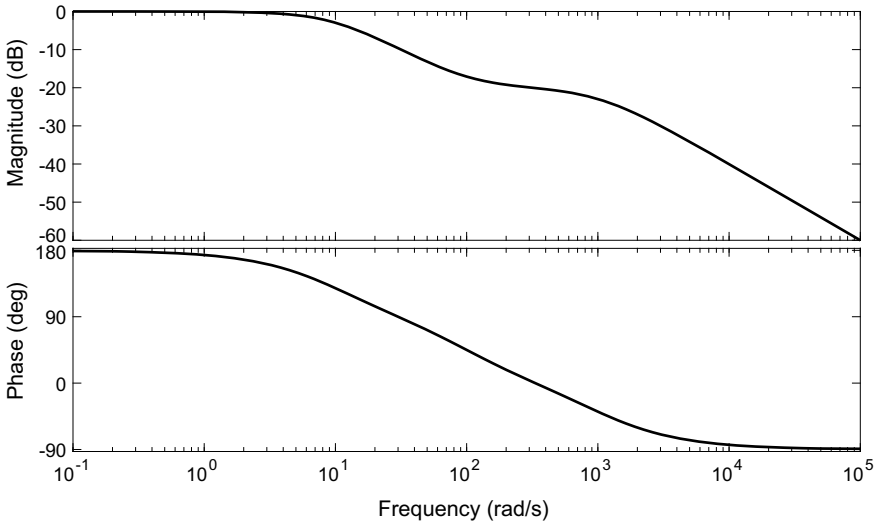


Fig. 5.28 Bode-plot of non-minimum phase system example

−20 dB/decade, should then go flat due to the zero at 100 rad/s up to 1000 rad/s where it will again roll-off at −20 dB/decade (Fig. 5.27).

We now deduce that the Bode-plot of (5.38) must be the same magnitude but the phase only must differ. Depending on how we represent the all-pass transfer-function it will either start at 180° rather than zero and move to 90° at high-frequencies, or start at zero as shown and go down to −270° instead of −90°. Of course the phase at mid-frequencies will also change since the all-pass has its own phase-characteristic which is non-linear (each frequency gets phase-shifted by a differing amount that is not proportional) (Fig. 5.28).

5.6 Time-Delays in Linear-Systems

Another problem that can arise with Bode-plots is when we have a time-delay in the system. A time-delay is characterised by the following transfer-function:

$$G_d(s) = e^{-s\tau} \tag{5.43}$$

where τ is time-delay measured in seconds.

For a signal say $u(t)$ that passes through a pure time-delay of magnitude τ seconds, we define its output as

$$y(t) = u(t - \tau) \tag{5.44}$$

Mostly such problems arise in process-control where so-called transport-delay of materials in a process causes this time-delay. It is less common in electronic systems except perhaps in digital-control where an inherent one-step time-delay equal to the sampling-rate is always present within the system. The problem with an expression such as (5.43) is that it is irrational. That is we cannot see what its poles and zeros are and it is not easy to see how to deal with this. However, G_d is a complex-number represented in Euler exponential form $re^{-j\theta}$. Its magnitude is unity and phase $-\theta$ radians.

Hence from (5.43) we get

$$|G_d(j\omega)| = \left|e^{-j\omega\tau}\right| = 1 \tag{5.45}$$

with phase

$$\phi\left(e^{-j\omega\tau}\right) = -\omega\tau \tag{5.46}$$

Figure 5.29 shows the Bode-plot as obtained from MATLAB for a pure time-delay with $\tau = 1$ s.

So the delay only contributes to phase-shift and not to magnitude. This is very similar to the non minimum-phase case and the resemblance can be seen further by expressing

$$G_d(s) = e^{-s\tau}$$

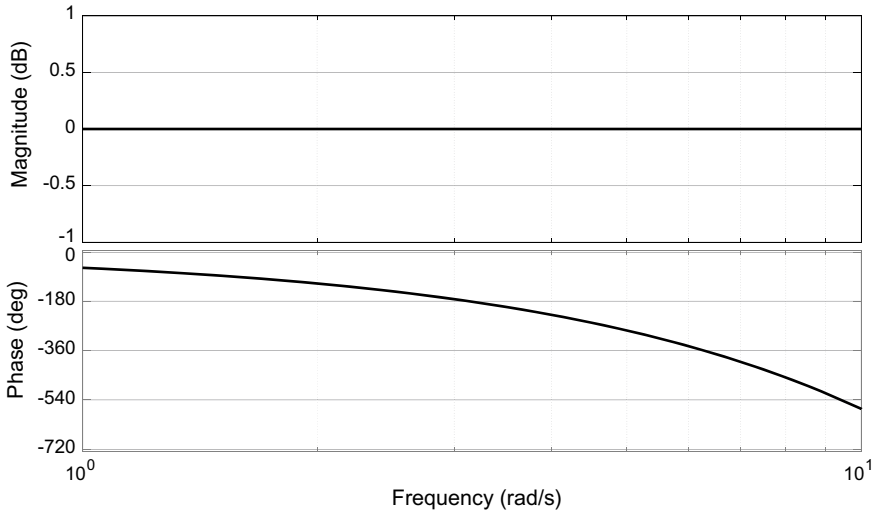


Fig. 5.29 Bode-plot of pure time-delay

But we can write

$$G_d(s) = \frac{e^{-s\tau/2}}{e^{s\tau/2}} \tag{5.47}$$

We know that for an exponential expression e^{-ax}, it has a Maclaurin power-series

$$e^{-ax} = 1 - ax + \frac{(ax)^2}{2!} - \frac{(ax)^3}{3!} + \cdots \tag{5.48}$$

From which we can expand numerator and denominator of (5.47) to a first-order approximation

$$\frac{e^{-s\tau/2}}{e^{s\tau/2}} \approx \frac{1 - s\tau/2}{1 + s\tau/2} \tag{5.49}$$

This is now a rational first-order transfer-function, but we note it is non minimum-phase. This approximation is usually called a Padé approximation after the French mathematician Henri Padé. The pure time-delay has *linear-phase* but on a log frequency scale looks nonlinear. We can obtain higher-order Padé approximations as required. For instance for order 2 [18].

$$e^{-s\tau} \approx \frac{1 - s\tau/2 + s^2\tau^2/8}{1 + s\tau/2 + s^2\tau^2/8} \tag{5.50}$$

Chapter 6
Stability and Design of Closed-Loop Systems

We have already looked at the stability of ordinary LTI systems when expressed as transfer-functions. The poles must lie in the left half of the s-plane. However, when we apply feedback to a system the poles change their values depending on the gain of the loop. Perhaps the easiest place to start is by examining the following system which is quite generic and has a gain K which can be varied (Fig. 6.1).

The open-loop system is then $KG(s)H(s)$ and in closed-loop we get the result we have used in previous chapters, namely

$$\frac{y}{u}(s) = \frac{KG(s)}{1 + KG(s)H(s)} \tag{6.1}$$

From (6.1) we can see that the poles of the closed-loop system are now given by the roots of

$$1 + KG(s)H(s) = 0 \tag{6.2}$$

and for closed-loop stability they must all lie in the left half of the s-plane. Equation (6.2) is usually known as the closed-loop characteristic equation (CLCE). This leads us nicely to the method of Root-Locus.

6.1 Root-Locus Method

The root-locus method is an ingenious method devised by Walter Evans [20] which we must remind ourselves was done at a time with no electronic calculators or easily available computers. Earlier in the 19th century Routh [4] and later independently Hurwitz [21] had devised a method of determining which roots were stable and which were not. The method did not calculate the roots however. The Evans root-locus method promised a whole design methodology based on roots

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_6

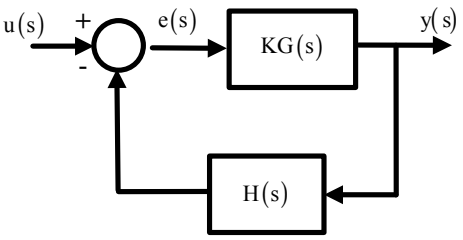


Fig. 6.1 Feedback around an LTI system with gain K

alone. He termed the path that the roots followed when the gain K was varied as the *root-locus*. To a certain extent the root-locus method has now been superseded by modern root-finding algorithms on fast computers, but the various patterns that the roots follow are worth knowing and adds to our overall understanding of feedback. From (6.2) we can write two equations

$$|\mathrm{KG(s)H(s)}| = 1 \tag{6.3}$$

and

$$\arg(\mathrm{KG(s)H(s)}) = -180^\circ + 360\ell \tag{6.4}$$

where $\arg(\mathrm{KG(s)H(s)})$ denotes the complex angle or phase and $\ell = 0, 1, 2$. We can also write (6.3) as

$$|\mathrm{G(s)H(s)}| = \frac{1}{\mathrm{K}} \tag{6.5}$$

We conclude that all the closed-loop roots of the CLCE in (6.2) must have the previously defined phase of -180° and magnitude satisfying (6.3). Evans invented a special graphical method and a special protractor like tool which he called the *spirul*e. Students would carefully plot to scale open-loop pole positions and use his tool to determine if roots were on the genuine root-locus or not. They could also determine the gain anywhere on the locus. The simplest way to understand is with examples.

6.1.1 First-Order System

Let $H(s) = 1, G(s) = \frac{1}{1+\mathrm{sT}}$ and form the CLCE (6.2). We get

$1 + \mathrm{KG(s)H(s)} = 1 + \frac{\mathrm{K}}{1+\mathrm{sT}} = 0$, and after re-arranging we find

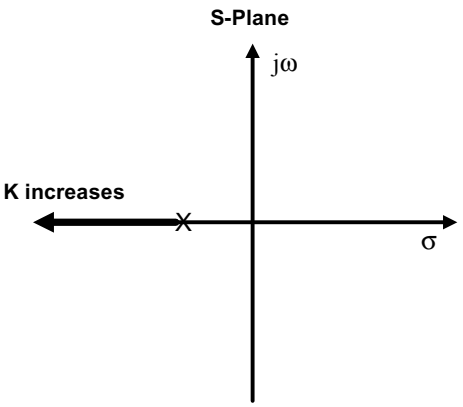


Fig. 6.2 Root-locus of first-order system with negative feedback

$$1 + sT + K = 0 \tag{6.6}$$

From which the pole can be written as a function of gain K

$$s = -(K + 1)/T \tag{6.7}$$

Now plot on the s-plane the open-loop pole $s = -1/T$ and the locus formed as K is varied upwards in value (Fig. 6.2).

So the pole just becomes more and more stable. Actually the closed-loop pole moves further to the left which speeds the closed-loop system up since fast poles are far to the left and slow ones near the imaginary axis. We therefore conclude that with negative feedback and a simple gain that the closed-loop system can never be unstable (practical real-life considerations are of course a different matter because the first-order system will be in reality much higher in order).

6.1.2 Second-Order System

Consider a second-order system with real-roots (the complex-roots case is somewhat similar).

$H(s) = 1, G(s) = \frac{1}{(1+sT_1)(1+sT_2)}$. We have two open-loop poles at $s = -1/T_1, -1/T_2$. After a little experimentation we can show that the closed-loop poles follow a locus as shown in Fig. 6.3.

The loci emerge from the two real poles and touch half way between the poles and then shoot upwards and become complex giving us an underdamped system. The arrows denote gain K increasing. We can conclude that a second-order system also cannot be unstable as gain increases. Instead, all that happens is that the damped natural frequency increases since second-order poles are defined by $s =$

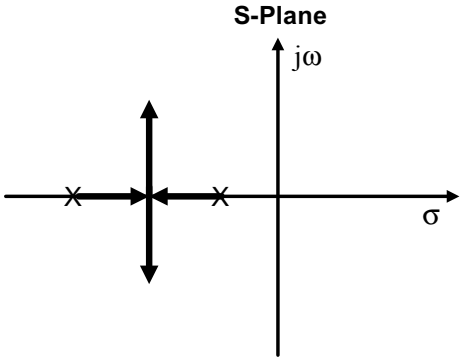


Fig. 6.3 Root-locus of second-order system

$-\zeta\omega_n \pm j\omega_d$ and any complex pole on the loci must be of that form. The angle which subtends the real axis with a complex pole must be $\vartheta = \tan^{-1}\left(\frac{\omega_d}{\zeta\omega_n}\right) = \tan^{-1}\left(\frac{\omega_n\sqrt{1-\zeta^2}}{\zeta\omega_n}\right)$. Which can be written as $\vartheta = \tan^{-1}\left(\frac{\sqrt{1-\zeta^2}}{\zeta}\right) = \cos^{-1}(\zeta)$. This means that the damping factor $\zeta = \cos(\vartheta)$ and it must reduce as the gain increases (Fig. 6.4).

The discovery made was that the roots eventually follow patterns of asymptotes depending on the number of poles minus number of zeros $(n_p - n_z)$. We can summarise and extend this by looking at different asymptotic behaviour in Fig. 6.5.

We see that any system that has a net gain in poles more than two has the potential for instability as the gain is increased in closed-loop. This is because the asymptotes point towards the right and must ultimately move the loci in that direction. The centre of the asymptotes is given as σ_a where

$$\sigma_a = \frac{\sum P - \sum Z}{(n_p - n_z)} \tag{6.8}$$

The summation symbol represents the sum of poles or zeros respectively. Consider the following third-order system.

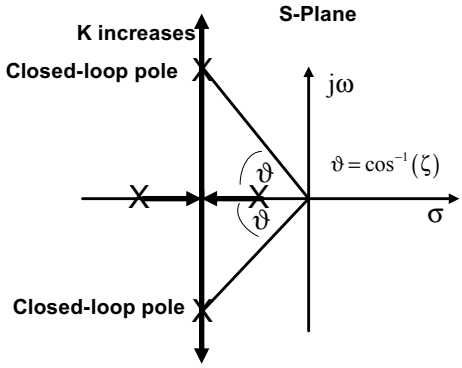


Fig. 6.4 Damping factor gets smaller with gain

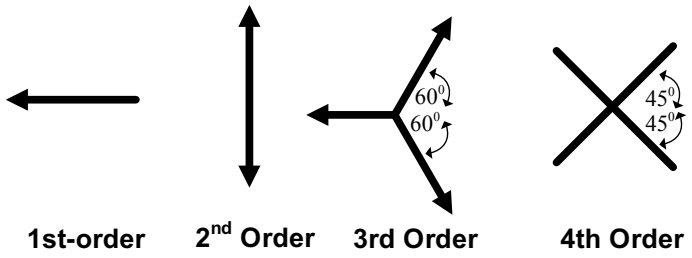


Fig. 6.5 Asymptotic behaviour for different orders of $(n_p - n_z)$

6.1.3 Third Order System with Feedback

$$G(s)H(s) = \frac{1}{(s+1)(s+2)(s+3)} \tag{6.9}$$

The centre of the asymptotes is $\sigma_a = \frac{\sum P - \sum Z}{(n_p - n_z)} = \frac{-1-2-3}{3} = -2$. We must have third-order asymptotic behaviour. The root-locus can be plotted on MATLAB as follows

MATLAB Code 6.1

```
% Root-Locus of three poles
num=1;
den1=[1 1];
den2=[1 2];
den3=[1 3];
den=conv(conv(den1,den2),den3)
g=tf(num,den)
rlocus(g)
```

Note that this time we have used the conv() command (convolution) to multiply the three denominator terms together. Convolution is the same as polynomial multiplication (Fig. 6.6).

By clicking on the locus just where it is on the imaginary axis, MATLAB returns the gain value at the point of instability as approximately K = 60. An interesting thing now happens if we introduce a zero at −3.5. Our open-loop system is $G(s)H(s) = \frac{(S+3.5)}{(S+1)(S+2)(S+3)}$ and $(n_p - n_z)$ now changes down to 2 giving second-order asymptotic behaviour (Fig. 6.7).

The pole at −3 locus moves towards the zero leaving two other poles which form a dominant second-order system which cannot be unstable. Zeros therefore play an important part in the stability of closed-loop systems (but not in open-loop). They act like attractors for the locus. It is a good idea to have a net number of poles-zeros equal to two so as to get this type of behaviour. Even for much higher-order systems

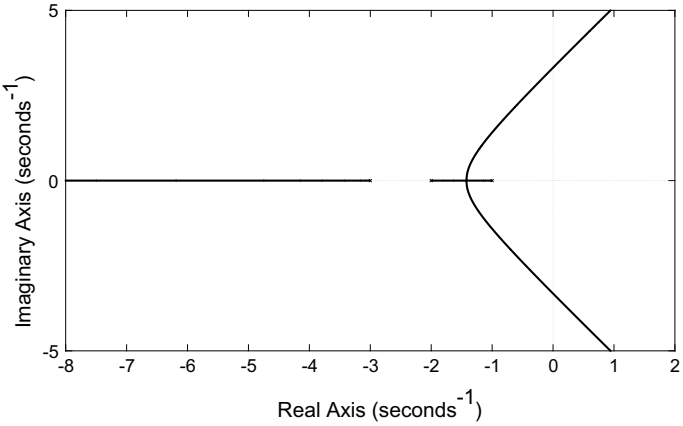


Fig. 6.6 Root-locus of 3rd-order system

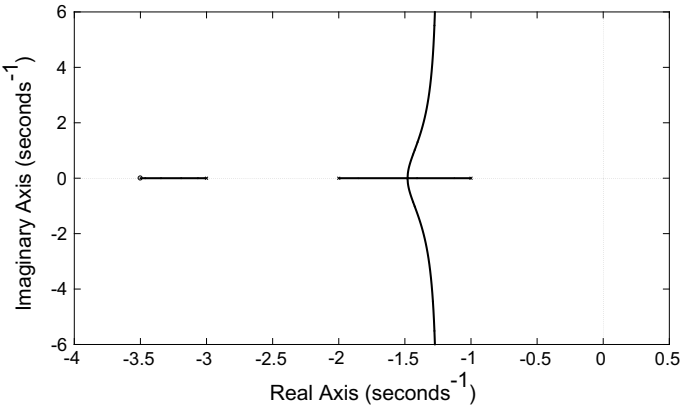


Fig. 6.7 Root-locus of three poles and one zero

the same applies. Zeros save the day. In analogue systems at least we cannot introduce zeros on their own as they are not physically realisable, so we usually have to introduce pole-zero pairs to change the characteristic or *compensate* the system in closed-loop. To illustrate this further consider the following system with resonance.

6.1.4 Fourth Order System with Feedback

The open loop system is $G(s)H(s) = \frac{\omega_n^2}{S(S+2)\left(S^2+2\zeta\omega_n S+\omega_n^2\right)}$ where $\zeta = 0.5$, $\omega_n =$ 6.283 rad/s (Fig. 6.8).

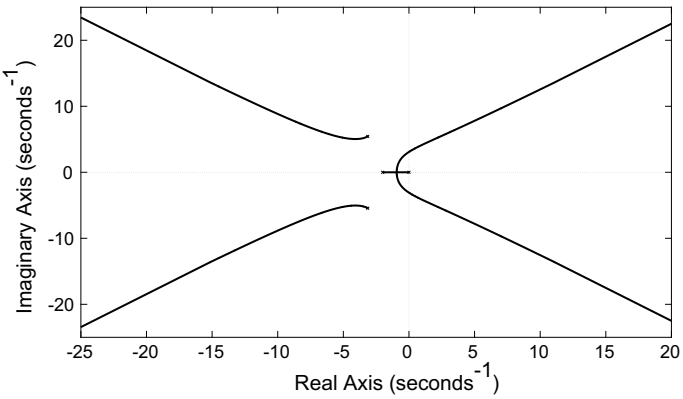


Fig. 6.8 Root-locus of fourth-order system

By clicking on the locus as it crosses the imaginary axis we find from MATLAB that the limiting gain is approximately $K = 10.4$. Now if we want to increase the gain so as to provided a smaller error and better tracking of the system in closed-loop we may want to add a zero. However, we cannot add a solo zero and must add a zero with a pole. We add the following transfer-function in cascade and zoom in on the important part of the locus near the imaginary axis. We obtain the plot of Fig. 6.9.

$$C(s) = \frac{(s+4)}{(s+40)}$$

Other than the interesting looped locus formed by the zero, we see that the zero bends the locus towards it. This is of some significance because if we re-measure

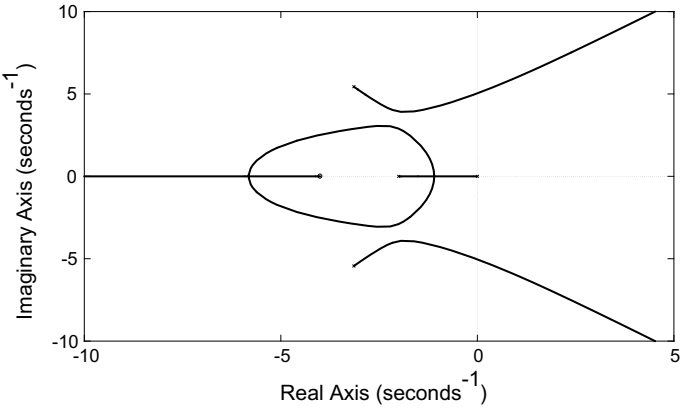


Fig. 6.9 Added compensator to fourth-order system C(s)G(s)H(s). Pole at −40 is not shown

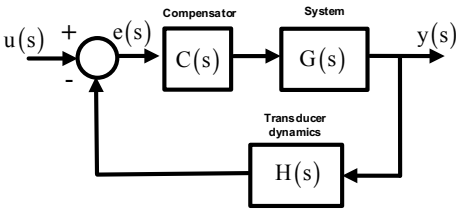


Fig. 6.10 Adding a cascade compensator

the gain as it crosses the imaginary axis we find that the maximum gain is now $K = 145$. We have now enabled the system to take more gain than before by including an extra zero and pole pair. We illustrate the concept in Fig. 6.10. This method of improvement or *compensation* is the de facto method in control-system design. Not always just by a lead-compensator of course, but often by combinations of different compensators as we shall see later in the book.

There are a great many rules associated with the root-locus method. These include the angles of the asymptotes, the fact that asymptotes never cross, departure and arrival angles and breakaway points. However, with a modern tool such as MATLAB the hand drawing of such plots is becoming obsolete. The method is a useful learning tool but is rarely if ever used among professional design-engineers. They tend to prefer the method we discuss next, frequency-response design.

6.2 Recognising Closed-Loop Instability Using Bode-Plots

Chapter 5 looked at how to plot the frequency-response of any linear time-invariant transfer-function. We know that given a transfer function, it is easy to plot the dB magnitude by using asymptotic straight-line approximations, and to deduce the phase from this too. Of course, as with root-locus, 21st century computing methods are far more powerful than they once were and the need even to know how to plot accurately by hand is becoming less of a necessity. This does not stop the Bode method from still being the best way by far for designing control-systems. By design here I mean by scientific principles and not by so-called "tuning" methods which are more of an ad hoc nature. We first look again at the closed-loop characteristic Eq. (6.2), $1 + KG(s)H(s) = 0$. This forms the denominator of the closed-loop transfer-function if $G(s)$ represents all dynamics in the forward path and $H(s)$ all dynamics in the feedback path. When this equation is zero we effectively get a situation where the closed-loop system "blows up" or goes unstable. This happens when

$$|KG(j\omega)H(j\omega)| = 1 \tag{6.10}$$

and the phase is

$$\arg(KG(j\omega)H(j\omega)) = -180^\circ \tag{6.11}$$

Put into words: *When the forward-path gain is unity (0 dB) at some frequency say* ω_ϕ *and the phase at that same frequency is* -180° *then the closed-loop system will be unstable*. It is important to note that the gain we are talking about that is unity, is the *open-loop* gain and so is the phase. The closed-loop performance can then be *predicted* from these open-loop results as in "what would happen if I now close the loop"? So when the open-loop frequency-response gain hits 0 dB (we call this the unity-gain crossover frequency ω_ϕ), we measure the phase. If it is -180° *or more* then we say that this system when the loop is closed will be unstable. If the phase is less than -180° then we work on the amount it is less, by defining the metric known as *phase-margin* of the system. The phase-margin is the difference that the actual phase is at unity-gain from -180°. It is defined as being positive for stable systems and negative for unstable. When the phase-margin is zero we have an oscillation at frequency ω_ϕ. In fact, in the early days of oscillators before direct-digital synthesis, all oscillators were designed to have zero degrees phase-margin. The difficulty was getting them to stay at -180° and maintain unity gain to get an oscillator whose frequency changes linearly. In our case we need the exact opposite, we need good margins of phase for a good damped response.

We also use a second metric. At a frequency where the phase just intersects -180° we measure the gain in dB and expect it to be less than 0 dB. This is the amount of gain left in the system before instability occurs and we term this the *gain-margin*.

To illustrate all this consider a third-order system $G(s)H(s) = \frac{10^7}{(S+10)(S+100)(S+1000)}$ similar to that we have already considered in Sect. 6.1. This can be written as $G(s)H(s) = \frac{10}{(1+S/10)(1+S/100)(1+S/1000)}$. Figure 6.11 shows the Bode-plot of the open-loop system. If g is the transfer-function (system) in MATLAB then margin (g) will show and compute the phase and gain margins.

For a phase-margin of around 55° we can safely predict a well-damped step-response were we to close the loop. Remember the prediction is for *closed-loop* given open-loop frequency data. We can also say that if the open-loop gain was increased by around 22 dB that the closed-loop system would be unstable. This information comes from the *gain-margin*. How do we know though what sort of phase-margin is good and what is bad? Well we know zero degrees is bad and increasingly higher gives more damping, but we can make a good estimate using the following analysis. Let the phase-margin be ϕ_m. Then for any order system $G(j\omega)$ with unity negative feedback, its closed-loop gain is $\frac{|G(j\omega)|}{|1+G(j\omega)|}$. The phase-margin occurs at open-loop unity-gain $|G(j\omega)| = 1$ and phase given by $180 - \phi_m$. The closed-loop magnitude can then be written as $M = \frac{|G(j\omega)|}{|1+G(j\omega)|} = \frac{|e^{-j(180-\phi_m)}|}{|1+e^{-j(180-\phi_m)}|} = \frac{1}{|1+e^{-j(180-\phi_m)}|}$. We can then plot M versus different ϕ_m. This we show in Fig. 6.12.

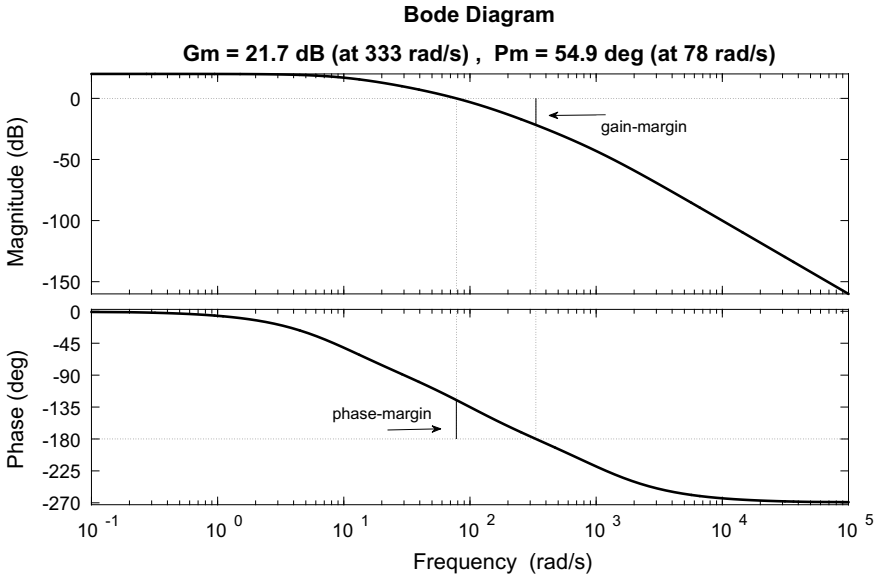


Fig. 6.11 Bode-plot of open-loop system showing phase and gain-margins

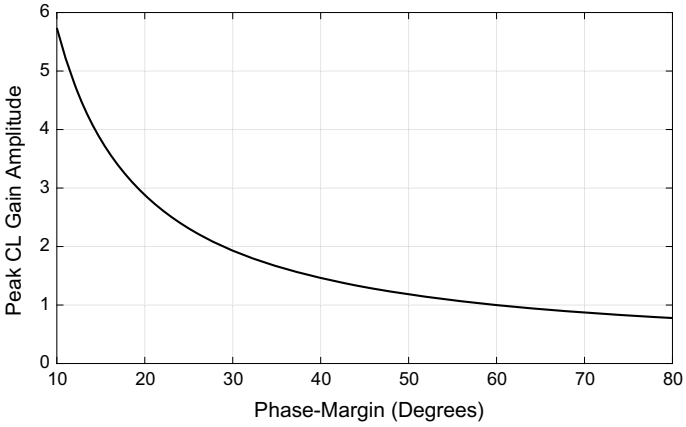


Fig. 6.12 Closed-loop gain versus phase-margin

We see that for a phase-margin of 60° we have no peak in the closed-loop gain other than unity. For a 30° phase-margin we have a closed-loop gain of 2 which is significant. This is of course for unity negative feedback only but gives us a fair representation that 60° of phase-margin or greater is considered at least acceptable and anything less than 40 is getting bad. It is the phase-margin that determines performance. The gain-margin only tells us how much gain there is left-over. Note also that we should have one only unity-gain crossover frequency. Often with

a system with resonance we can end up having more than one phase-margin. For example, one phase-margin could be positive and the second zero or negative resulting in a good-looking step-response but with a high-frequency oscillation superimposed on top. The oscillation is often known as a *parasitic*. To understand further it is best first to consider operational-amplifier problems before tackling electro-mechanical systems.

Consider a typical operational amplifier (op-amp). Although an op-amp has many poles, we can usually approximate it to a first-order system with a large forward-path gain. For-example the forward-path transfer-function of a typical op-amp can be written as $G(j\omega) = \frac{K}{1+sT}$ where $T = 15.9\,\text{ms}$, $K = 10^5$. (a 3 dB frequency of 10 Hz). The Bode-plot is shown in Fig. 6.13.

MATLAB defaults to a frequency axis which is not high enough for this example so we use the command `bode(g,{1,10000000})` to extend the axis. We immediately see that at the unity-gain crossover frequency the phase is 90° indicating an excellent phase-margin. The gain-margin is infinite too, so their appears to be little problem at all. In fact if we apply a ratio of resistors as feedback as in Fig. 6.14 we will get near ideal operation.

The feedback-term is not frequency dependant and therefore when we multiply $G(j\omega)H(j\omega) = \frac{K}{1+sT}\beta$ this in no way effects the phase characteristic of the op-amp. Therefore the closed-loop stability stays the same. The closed-loop gain of the op-amp of course change to the reciprocal of the feedback network β. This was of course the original intention by the inventor Black in 1934, who was first to realise the importance of feedback around an amplifier. In this way we can define easily a stable closed-loop gain that does not drift with temperature. For the inverting amplifier this is harder to see, but we can show an equivalent circuit which is perhaps not obvious at first, to see how we can relate it to control-theory [22] (Fig. 6.15).

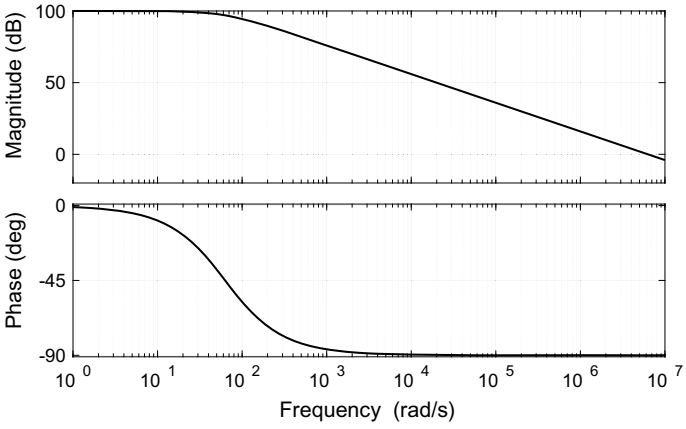


Fig. 6.13 Bode-plot of op-amp

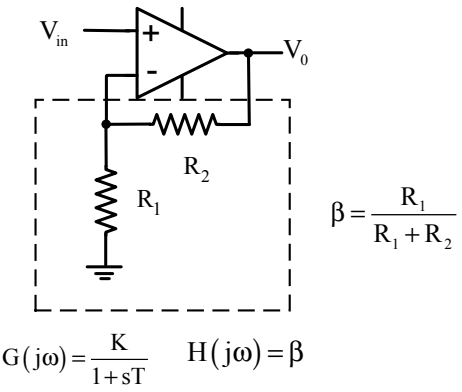


Fig. 6.14 Non-inverting op-amp configuration

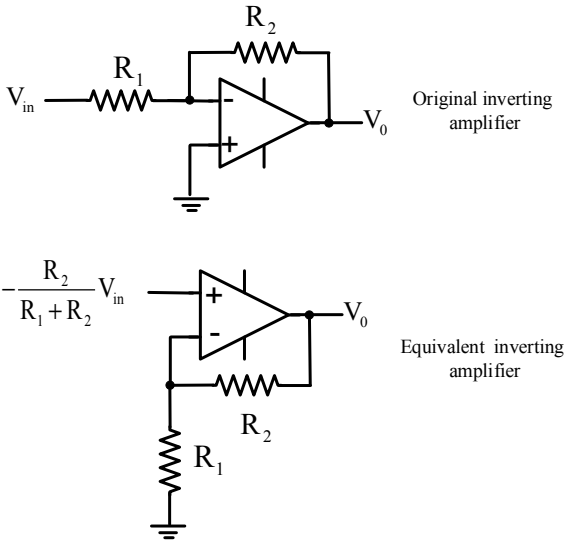


Fig. 6.15 Inverting op-amp configuration

This also applies to general impedances where we replace resistances *R* with impedance *Z*. So apart from the scaling outside of the loop by $-\frac{R_2}{R_1+R_2}V_{in}$, which does not effect the stability of the loop in any way, the two problems are the same. The inverting amplifier also poses no stability threat in closed-loop since it too has no frequency-dependent components.

6.2.1 Ideal and Practical Differentiator Circuit

Now let us change some of the components to impedances. Consider an ideal differentiator (the one we are always told not to use but never given a real explanation as to why!) and its equivalent circuit (Fig. 6.16).

For transfer-functions we have $H(j\omega) = \frac{1}{1+j\omega CR}$ in the feedback path and $G(j\omega) = \frac{K}{1+j\omega T}$ in the amplifier forward-path. Define $H(j\omega) = \frac{1}{1+j\omega T_2}$ where $T_2 = CRs$. Now we pick some values. Let C = 100 nF, R = 100 kΩ. This gives a time-constant of T_2 = 0.01 s or 100 rad/s. Multiplying the amplifier open-loop transfer-function with the feedback network gives us $G(j\omega)H(j\omega) = \frac{K}{(1+j\omega T)(1+j\omega T_2)}$. With the given values we have a Bode-plot as shown in Fig. 6.17.

The unity-gain crossover frequency is $\omega_\phi = 25100$ rad/s or approx 4 kHz. The phase at this frequency is nearly −180° and MATLAB tells us there is 0.372° of phase-margin. Therefore the system will oscillate at 4 kHz in closed-loop. This is why pure differentiators are not used in practice. However, with a little trick we can add a series resistance R_s to the capacitor as shown in Fig. 6.18 in the equivalent circuit.

We now have the same forward-path op-amp transfer-function but with a change of feedback transfer-function to $H(j\omega) = \frac{1+j\omega CR}{1+j\omega C(R_s+R)}$. Let us design the series resistor as $R_s = 1\,k\Omega$. Our feedback transfer-function is now $H(j\omega) = \frac{1+j\omega T_1}{1+j\omega T_2}$ with $T_1 = CR,\ T_2 = C(R_s + R) > T_1$. Now $H(j\omega)$ we have met before Eq. (5.29) but a different circuit. It is known as a lag compensator. To see its effect here we plot the composite Bode-plot which is shown in Fig. 6.19. Note that the external scaling of the

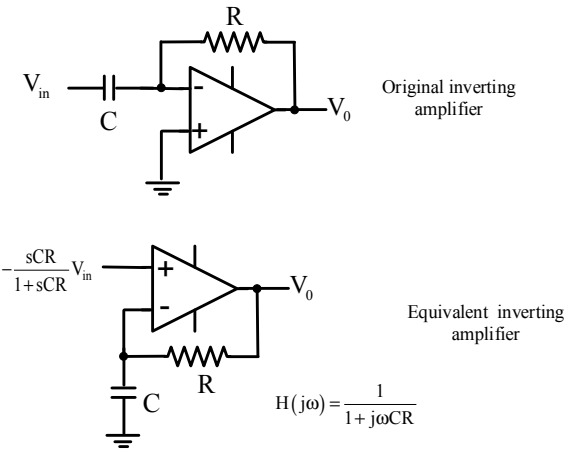


Fig. 6.16 Ideal differentiator and equivalent circuit

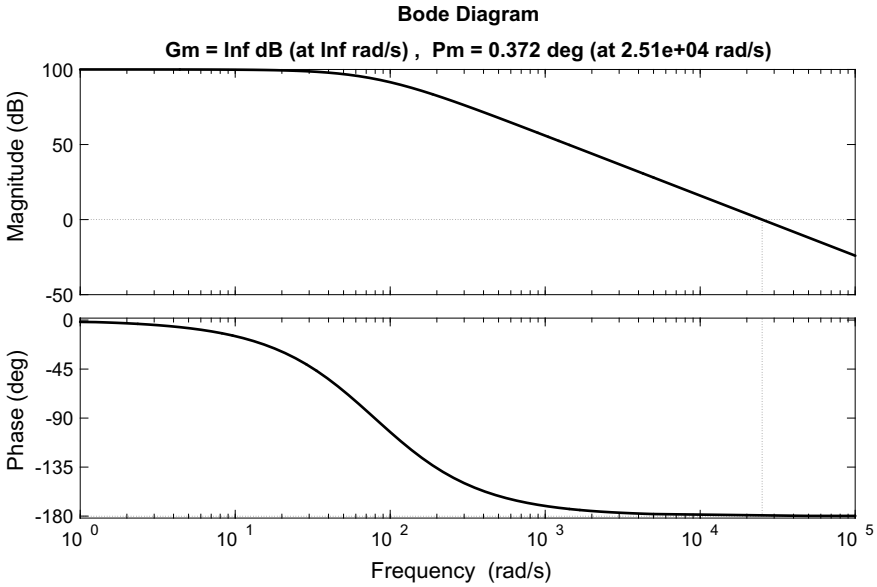


Fig. 6.17 Open-loop Bode-plot of ideal differentiator

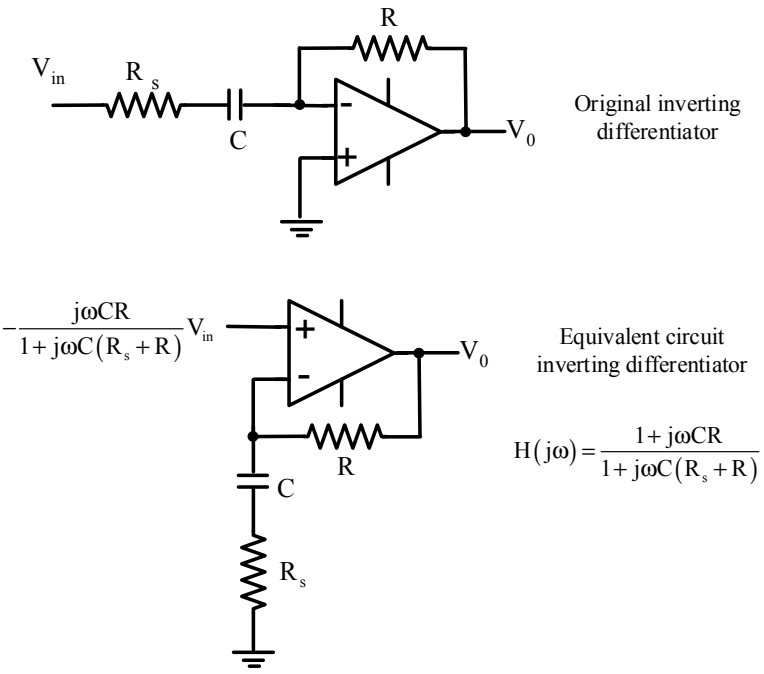


Fig. 6.18 Non-ideal (practical) differentiator

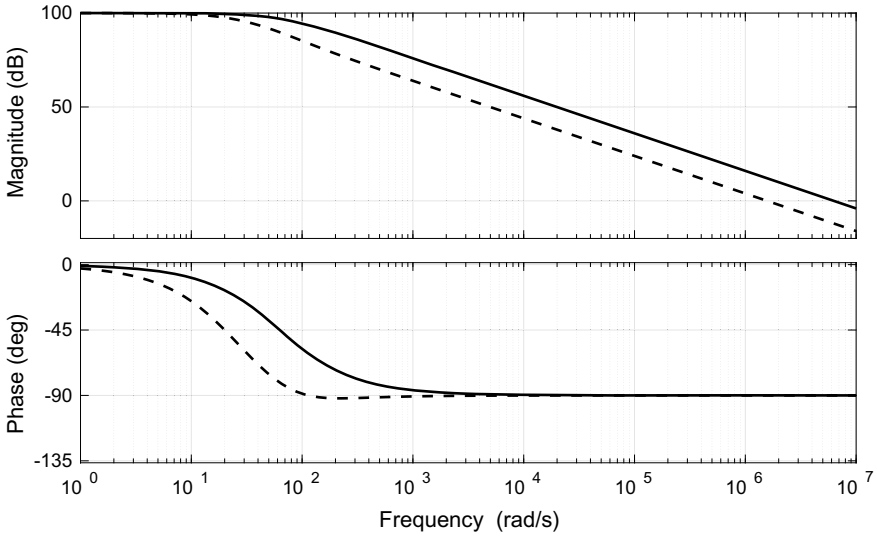


Fig. 6.19 Bode-plot of compensated system (practical differentiator)

input signal in the equivalent circuit of Fig. 6.18 is outside the loop and we need not consider its effect on stability. The MATLAB code that produces this plot is shown below.

MATLAB Code 6.2

```
% Compensation for differentiator
%Compare Bode-plots
R=100000;
Rs=300000;
C=100e-9;
T1=C*R;
T2=C*(R+Rs);
K=1e5;
T=15.9e-3;
num=[T1 1];
den=[T2 1];
gc=tf(num,den)
num2=K;
den2=[T 1];
g=tf(num2,den2)
gt=gc*g
% Compensated Bode-plot
bode(gt,{1,10000000})
grid
hold on
% Bode plot of op-amp - ordinary
bode(g,{1,10000000})
```

Now we see that the broken line represents the compensated system. The phase margin remains at 90° for the ordinary op-amp and for the practical differentiator. We do see however that the unity-gain crossover frequency has also dropped as a consequence. For this kind of example it will make little difference to the performance of the op-amp. Furthermore, any extra phase-shift that is created due to this compensator is at low frequencies and well away from the crossover frequency where it can do some harm to phase-margin.

6.2.2 *Capacitance Loading in an Op-Amp*

This is a good example discussed in [22] and is often faced in practice. Suppose we have an inverting amplifier connected to a long cable. The cable acts as a capacitor and we can show a circuit such as Fig. 6.20 where C_L represents the capacitance loading effect.

This becomes a little clearer in Fig. 6.21 when we look into the internal workings of the op-amp and model its output resistance R_o.

The best way to proceed is to take the Thevenin-equivalent circuit of the op-amp inner workings. Removing the resistor-divider R_1, R_2 and looking back into the voltage source KV_e we get the circuit of Fig. 6.22.

Analysing Fig. 6.22

$$V^- = \frac{R_1}{R_1 + R_2 + Z_{TH}} V_{TH} \tag{6.12}$$

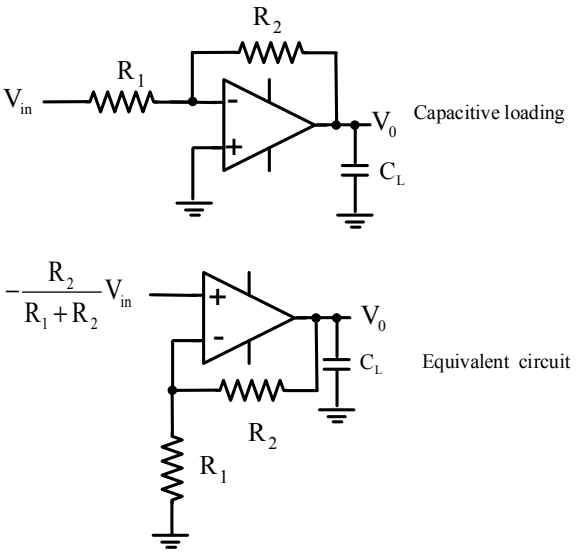


Fig. 6.20 Capacitive loading of an op-amp

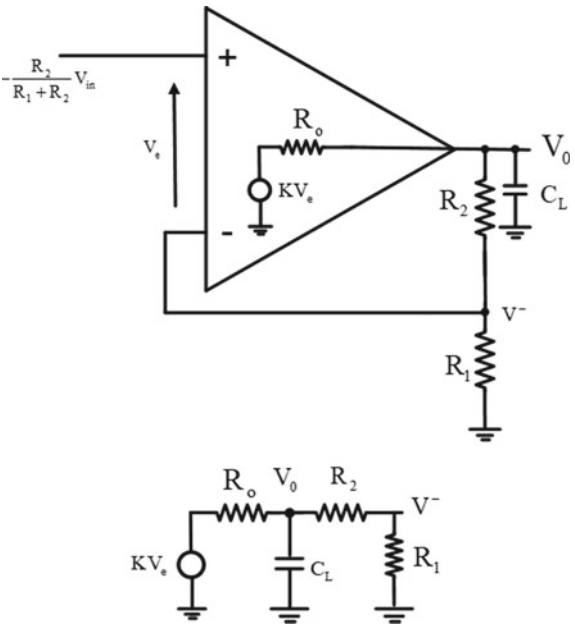


Fig. 6.21 More detail for capacitive-loading

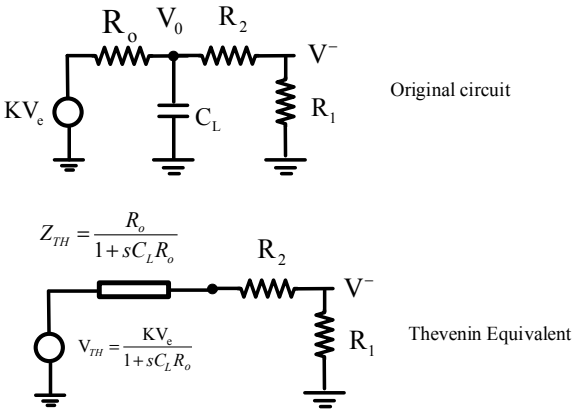


Fig. 6.22 Thevenin equivalent circuit

$$= \frac{R_1}{R_1 + R_2 + \frac{R_o}{1 + sC_L R_o}} \frac{1}{(1 + sC_L R_o)} KV_e \tag{6.13}$$

$$= \frac{R_1 KV_e}{(1 + sC_L R_o)(R_1 + R_2) + R_o} \tag{6.14}$$

$$= \frac{\left(\frac{R_1}{R_1 + R_2 + R_o}\right) KV_e}{\left(1 + sC_L \frac{R_o(R_1 + R_2)}{R_1 + R_2 + R_o}\right)} \tag{6.15}$$

But since R_o is small we have

$$V^- \approx \frac{\left(\frac{R_1}{R_1+R_2}\right)}{1+sC_LR_0} KV_e \tag{6.16}$$

Now we put some values in. Let the load capacitance be $C_L = 2\,\text{nF}$, output impedance $R_o = 1000\,\Omega$, $R_1 = R_2 = 10\,\text{k}\Omega$ and the other values as per the previous example i.e. $G(j\omega) = \frac{K}{1+sT}$ where $T = 15.9\,\text{ms}$, $K = 10^5$.

Define $T_L = C_L R_0$. Removing K from (6.16) and replacing it with $G(j\omega)$ instead (since they are in cascade anyway), we have

$$V^- \approx \frac{0.5}{1+j\omega T_L} V_e = H(j\omega)V_e \tag{6.17}$$

Now if we plot the Bode-plot for $G(j\omega)H(j\omega) = \frac{K}{(1+j\omega T)} \frac{0.5}{(1+j\omega T_L)}$ we see that all that has happened is that an extra pole has been added to the open-loop system making the op-amp second-order instead of first (Fig. 6.23).

The phase-margin is significantly reduced to 22°, which will give a significant amount of overshoot and ringing.

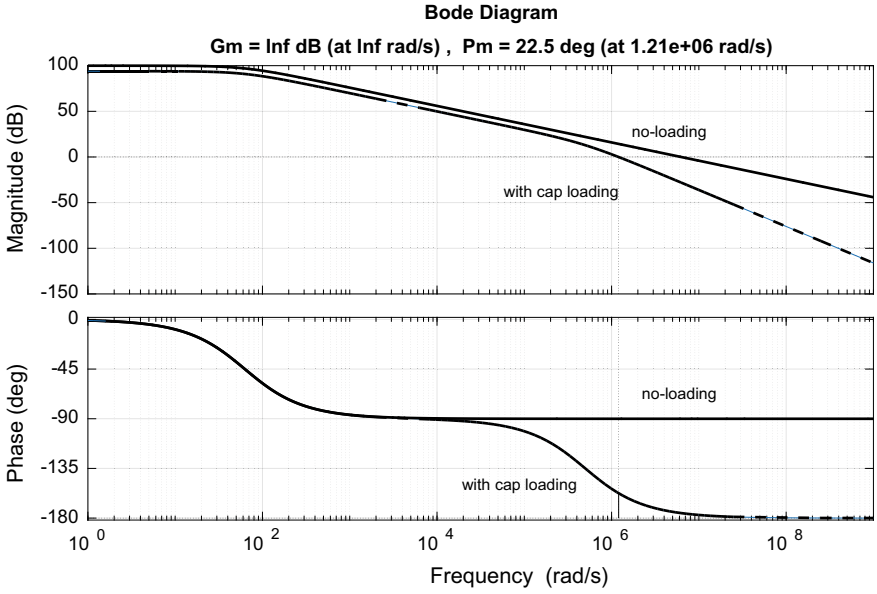


Fig. 6.23 Bode-plot for capacitance loading

6.3 The Ideal Bode-Plot

For a given open-loop frequency-response we can always make it stable, but we must examine at what cost? Not financially, but we find in control-systems design that if we improve one thing we inevitably alter something else. We already know that increasing the gain in most systems improves the accuracy but also reduces phase-margin and gives overshoot. We must find compromises and try and get as near the ideal as possible. The concept of an ideal Bode-plot has not been discussed in hardly any textbook before as they tend to dwell on specific types of compensation. For example design by lag, lead or lag-lead, PID etc. We must always remember however that just because we can stabilise a closed-loop system does not mean we cannot make further improvements, for example to the speed of response without compromising the former properties. An ideal Bode-plot would have an infinite gain. However, the next best thing would be to have as high a gain as possible given certain restrictions. This is really an optimisation problem but without the usual mathematics.

Consider the Bode-plot of Fig. 6.24.

If we consider low-frequencies to be frequencies below unity-gain crossover and high-frequencies above this, then we must have high-gain at low frequencies and low gain at high-frequencies. High-gain at low-frequencies reduces errors in steady-state and improves accuracy (the ability for the output to track the input). We cannot have high gain at all frequencies as such a system would be unstable. Therefore we must roll-off down at some slope. Usually we can make this no more than −40 dB/decade (this constitutes two integrators at this range of frequencies). If we continued through unity-gain with two integrators there would be no

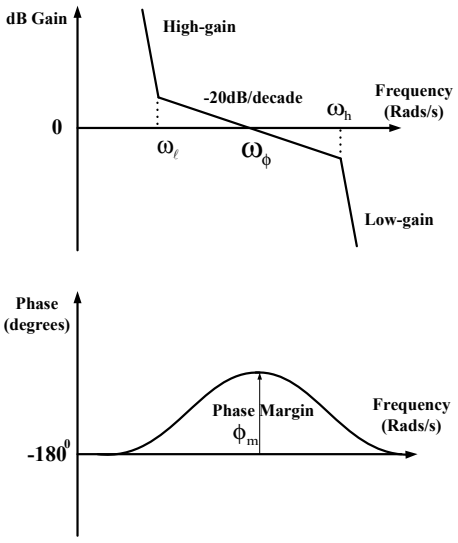


Fig. 6.24 Near-ideal Bode-plot

phase-margin (since two integrators = −180° of phase shift). Therefore we must roll off at respectable slope through unity-gain and usually this is −20 dB/decade. We can also go through flat since the actual non-asymptotic plot will just squeeze through and not be flat. Usually however we cut through unity gain at −20 dB/decade. Finally we must roll-off at high-frequencies as steep as possible to attenuate noise and any resonances which usually lurk at high-frequencies. We have termed unity-gain crossover as frequency ω_ϕ, also known as the *bandwidth*. This frequency must be as high as possible since the higher the bandwidth, the faster is the closed-loop system. We cannot compromise usually on bandwidth unless we have no choice. What keeps us from increasing the bandwidth indefinitely is that there will be higher-order poles which come into play at high frequencies and these cause phase-shift in the negative direction. We see that the phase-margin must be good—say 60° minimum and more if possible if no overshoot is required. This is good design. We must not compromise too much on phase-margin either.

Gain-margin is not shown, but must be at least 6 dB or more.

Another phenomena that restricts bandwidth is structural resonance in electro-mechanical systems. These are modelled with 2nd order underdamped systems and therefore have a great deal of negative phase-shift. We need to keep about an order of magnitude down in frequency from any resonances. Any attempt to filter them out with zeros cancelling poles is usually met with an even higher frequency resonance. So resonances are best designed out of the system by using rigid-body dynamics or special materials that have high resonant frequencies.

We define the ratio of the upper roll-off frequency to the lower as the span-ratio p. We already have met this when studying some commonly met Bode-plots from circuits.

$$p = \frac{\omega_h}{\omega_\ell} \tag{6.18}$$

In many cases we can relate span-ratio to phase-margin. For example from Table 5.1 a span of p = 10:1 gives us around 55° of phase margin. Reducing the span-ratio for a given *fixed-bandwidth* will reduce the phase-margin but will increase the gain at lower-frequencies. So there will be better disturbance-rejection but less stability. Note that we do not compromise much on bandwidth. We cannot compare compensation (controller) methods that give differing bandwidth. A controller that gives more stability simply won't do, if the speed of response has slowed down as a consequence. The design steps we follow usually depend on the type of system, but for the usual position-servo, the amplifier-motor-load model is well known to be a transfer-function of the form $G(s) = \frac{K}{s(1+sT_m)}$ as a first-order approximation (ignoring the inductance in the armature). So let us first have a look at how to design the best controller for this. (in the classical sense).

6.4 Example. Bode Based Compensation of a Motor + Load (Position Feedback)

Suppose we have an amplifier + dc-motor and load with transfer-function

$$G(s) = \frac{K}{s(1 + sT_m)} \tag{6.20}$$

Let us put some values in. K = 100, $T_m = 3.2\,\text{ms}$. Use MATLAB to plot the initial Bode-plot (Fig. 6.25).

Examining the Bode-plot, we must not be fooled into thinking that as is good, even though the phase-margin is 73° with infinite gain-margin. The bandwidth is 100 rad/s. We can do much better. We can increase the bandwidth and we can improve the low-frequency-performance. At present it has a mechanical integrator at low-frequencies. We can put a second electrical integrator in place by using a cascaded compensator as shown in Fig. 6.26. The compensator is placed

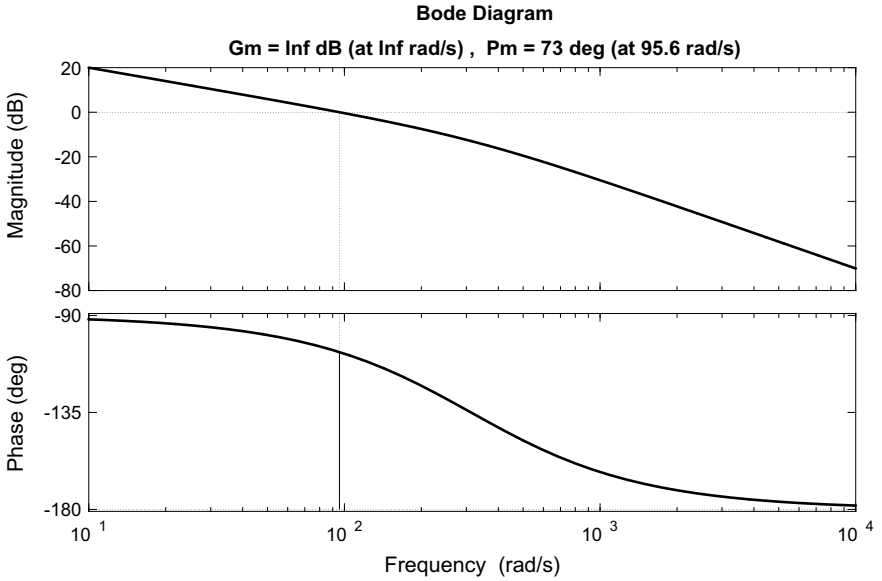
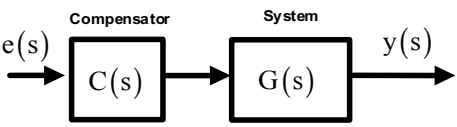


Fig. 6.25 Bode-plot of amplifier, motor + load

Fig. 6.26 Adding a compensator (or controller)



usually as part of or before the power amplifier. Here the error in the system is driving the compensator. We assume the amplifier and system are combined here since everything is in cascade they all multiply together anyway. *A compensator is just another name for controller*. One definition comes from amplifier design and the other from control-theory. The two terms are interchangeable.

We will perform the design in steps.

Step 1

Increase the gain and cancel the motor pole with a zero. Another way of thinking is that we are adding an integrator at low frequencies and the taking it out at a frequency $1/T_m$.

$$C_1(s) = \frac{1000(1 + sT_m)}{s} \tag{6.21}$$

The other name for such a compensator is proportional plus integral. This is because if we divide by s we get $C_1(s) = 1000T_m + \frac{1000}{s}$ which is the sum of an integrator and gain term (which is termed proportional control). Combining the compensator with the system we get

$$\begin{aligned} C_1(s)G(s) &= \frac{1000(1 + sT_m)}{s} \frac{K}{s(1 + sT_m)} \\ &= \frac{1000K}{s^2} \end{aligned}$$

A revised Bode-plot is shown in Fig. 6.27. The bandwidth has been increased at least three-fold to 316 rad/s. But we now have zero phase-margin or a sustained oscillation at that same frequency of 316 rad/s if the loop were to be closed. Remember at all time this is open-loop design which predicts closed-loop performance.

Step 2

We need a decent phase-margin so we use a phase-lead compensator. We have already looked at phase-lead (advance) compensators in electronic form. They are a transfer-function of the form.

$$C_2(s) = \frac{(1 + sT_1)}{(1 + sT_2)}, T_1 > T_2 \tag{6.22}$$

The phase advance is just what we need since it has a bulging phase which we can adjust by selection of the span-ratio $p = \frac{T_1}{T_2}, T_1 > T_2$. To design with it we first need to know where to put the maximum phase. This is easy since the oscillation is at 316 rad/s (Fig. 6.28). We place maximum phase at the frequency of oscillation to stop the oscillation.

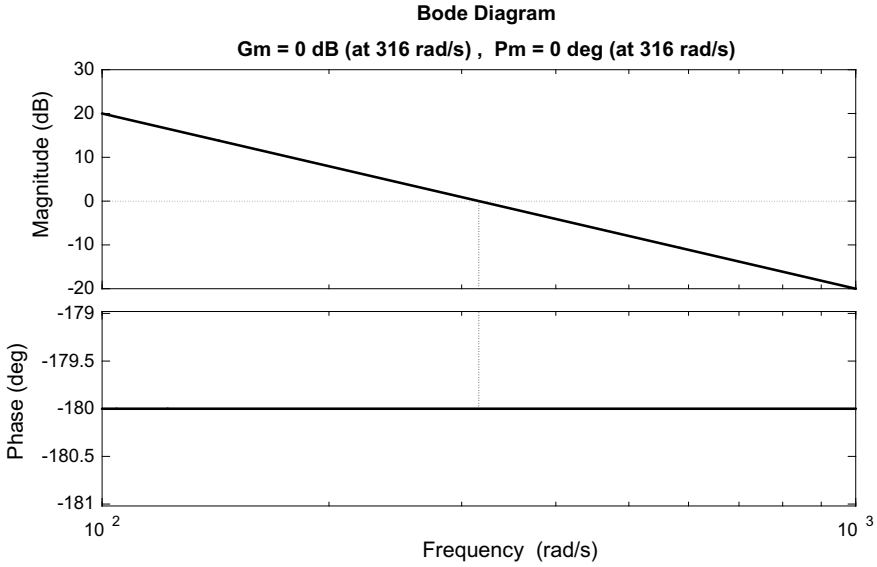


Fig. 6.27 Adding a second integrator at low-frequencies

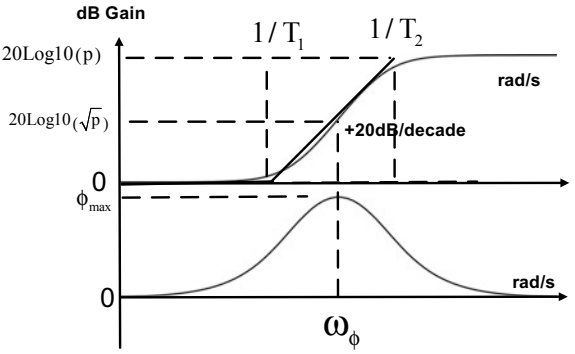


Fig. 6.28 Phase-lead (advance) compensator

The equations of the phase-lead are $\omega_h = \frac{1}{T_2}, \omega_\ell = \frac{1}{T_1}, p = \frac{\omega_h}{\omega_\ell}$. The maximum gain is the span-ratio p and the gain at maximum phase is $\sqrt{p}$. Since the gain rises up at high-frequencies, it is better to scale the compensator so that it has unity-gain at maximum frequency (so we do not have to re-adjust the overall gain) making it

$$C_2(s) = \frac{(1 + sT_1)}{\sqrt{p}(1 + sT_2)}, T_1 > T_2 \tag{6.23}$$

Maximum phase occurs at the geometric root of the upper and lower corner-frequencies, namely $\omega_\phi = \sqrt{\omega_h \omega_\ell}$.

Since $\omega_h = p\omega_\ell$ and $\omega_\phi^2 = \omega_h \omega_\ell$ we have $\omega_\phi^2 = p\omega_\ell^2$ or $\omega_\phi = \sqrt{p}\omega_\ell$. We can also show that $\omega_\phi = \frac{\omega_h}{\sqrt{p}}$ so that the two important equations are

$$\omega_h = \sqrt{p}\omega_\phi \tag{6.24}$$

$$\omega_\ell = \frac{\omega_\phi}{\sqrt{p}} \tag{6.25}$$

If we make the span-ratio $p = 10$, and $\omega_\phi = 316$ rad/s then $\omega_h = \sqrt{10}\omega_\phi = 999.27$ rad/s. Similarly $\omega_\ell = \frac{\omega_\phi}{\sqrt{p}} = 99.9$ rad/s. A span-ratio of 10:1 will give no frequency-domain peak in closed-loop but will still have some overshoot in the time-domain. The time-constants associated with the lead compensator for our values are then $T_2 = \frac{1}{\omega_h} = 1$ ms and $T_1 = \frac{1}{\omega_\ell} = 10$ ms. The compensator becomes from (6.23)

$$C_2(s) = \frac{(1 + sT_1)}{3.16(1 + sT_2)}, T_1 > T_2$$

for the given time-constants. We add this to the overall compensator and get

$$C(s) = C_1(s)C_2(s) = \frac{1000(1 + sT_m)}{s} \frac{(1 + sT_1)}{\sqrt{p}(1 + sT_2)} \tag{6.26}$$

Unity-gain bandwidth is still 316 rad/s, so we have not compromised the bandwidth in any way and the phase-margin is 55° due to the 10:1 span ratio. The neat way of including the reciprocal of the square-root of the span-ratio in the lead compensator enables the gain calculation to be precise. Otherwise we would have to drop the gain afterwards so that the phase maximum corresponds with unity-gain so as to maintain stability. The MATLAB Bode-plot is shown below (Fig. 6.29) and was obtained with the code given below.

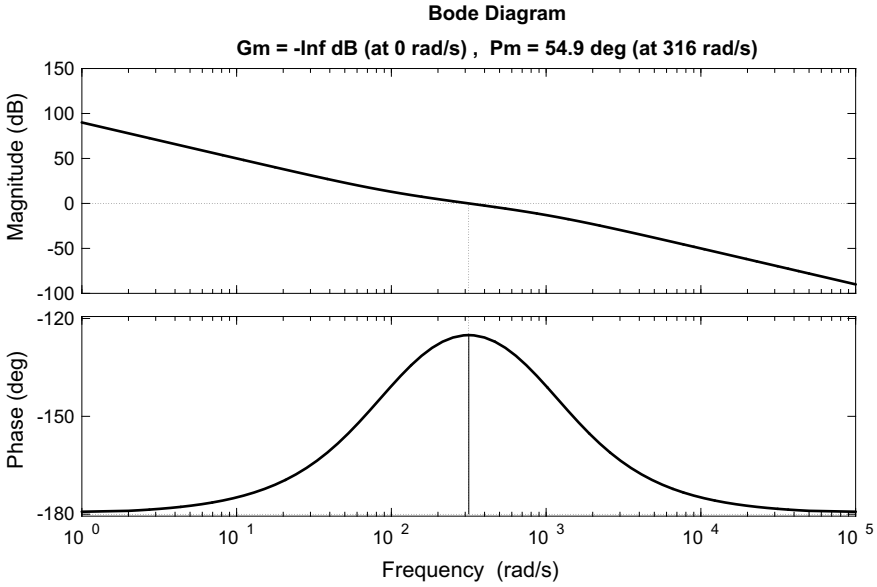


Fig. 6.29 Compensated system Bode-plot

MATLAB Code 6.3

```
% Compensation for motor
 % Motor + load  transfer-function first
K=100;
Tm=3.2e-3;
num=K;
den1=[Tm 1];
g=tf(num,den1)
num2=1;
den2=[1 0];
gi=tf(num2,den2);
gm=g*gi
%Compensator 1
num3=1000*[Tm 1];
den3=[1 0];
gc1=tf(num3,den3)
%Compensator 2
num4=[10e-3 1];
den4=3.16*[1e-3 1];
gc2=tf(num4,den4)
% Overall compensator – put two together
gt=gc1*gc2*gm
bode(gt,{1,1000})
grid
margin(gt)
```

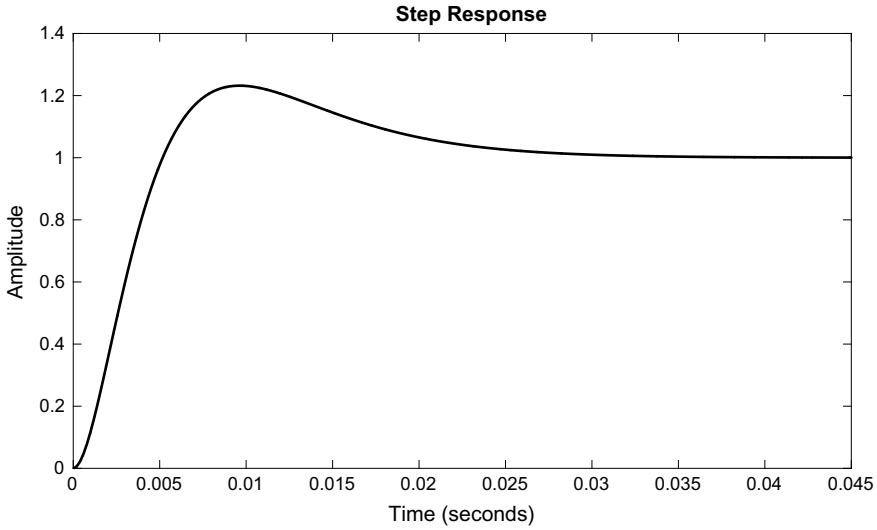


Fig. 6.30 Step-response of closed-loop system

Once we have the open-loop compensated system named gt in MATLAB we can easily calculate the closed-loop step-response from (Fig. 6.30).

```
gc = gt/(1 + gt)
step(gt)
```

The closed-loop system still has a 20% overshoot and this can be improved in a number of ways. We could increase the span-ratio for instance or even try introducing a second phase-advance. Increasing the span-ratio is not the best of ideas since it reduces the gain at low frequencies. So we can try and add a *second* phase-advance in cascade. Make the second one identical to the first.

$$C_3(s) = \frac{(1+sT_1)}{3.16(1+sT_2)}, T_1 > T_2$$

So our total compensator is now

$$C(s) = C_1(s)C_2(s)^2 = \frac{1000(1+sT_m)}{s}\frac{(1+sT_1)^2}{p(1+sT_2)^2} \tag{6.27}$$

and the Bode-plot becomes (Fig. 6.31).

If we examine carefully we see that the magnitude must go through flat from an asymptotic point of view but in reality it just squeezes through unity gain at the right frequency. The phase-margin has now shot up to be 110° and the closed-loop step-response is shown in Fig. 6.32.

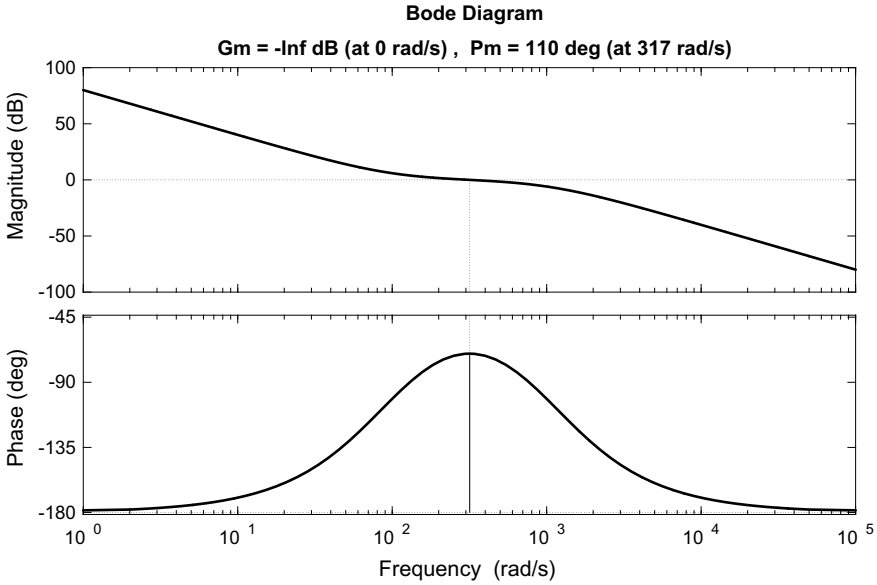


Fig. 6.31 Bode-plot of compensator with two phase-advances

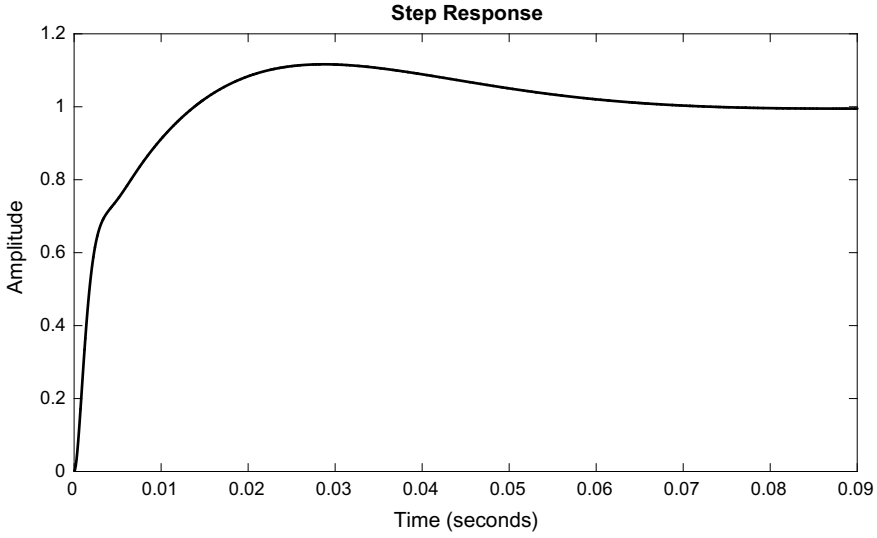


Fig. 6.32 Step-response with a double phase-advance

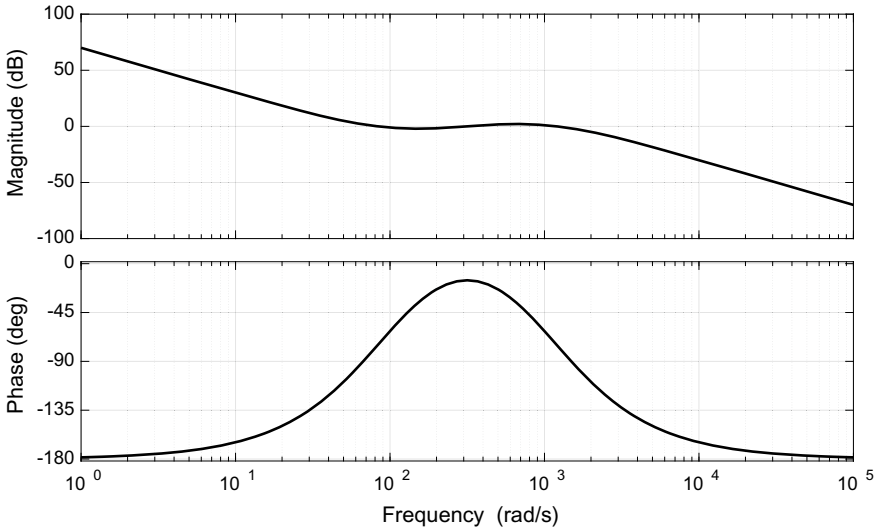


Fig. 6.33 Bode-plot with three phase-advances

The overshoot is now down to 10% and the step-response has an interesting look to it in the transient period. In all of this design we have not changed the bandwidth so the speed of response (the transient or rise-time) has not changed, only the damping due to phase-advance.

The obvious question now is, how many phase-advances can we add? Let us try one more identical phase-advance. We get the Bode-plot of Fig. 6.33.

Close examination of the region around 0 dB magnitude will reveal that we have no less than three phase-margins because the magnitude has crossed the line three times and each phase-margin is stable! The step-response is shown in Fig. 6.34.

We see that the output reverses in the transient period. This is unsatisfactory and clearly we require only one phase-margin, one crossing of the 0 dB line by the magnitude. What is happening is that the closed-loop system is behaving like more than one system at the same time. The fast response starts with larger bandwidth followed by the others all adding to give this bizarre result which we should avoid. There is another reason however for avoiding the use of too many phase-advances and that is when we have structural resonances in the system, and it is this example we look at next.

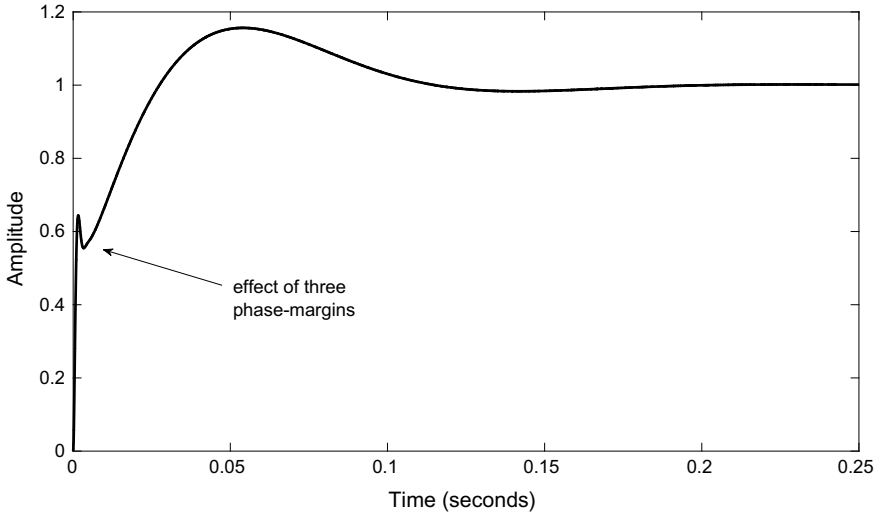


Fig. 6.34 Step response of a system with three phase-advances and three phase-margins!

6.5 Compensation with Structural Resonance

Many electro-mechanical systems have a structure that vibrates at a natural frequency. Even many servos will shake violently at certain frequencies unless the mechanical dynamics are of a rigid-body structure. Identifying the frequency at which the resonance occurs can be done either by excitation with a sine-wave, or by applying servo-theory to stabilise the loop unsuccessfully, and measuring the resonant frequency in closed-loop which will appear as a parasitic, added to the normal transient response. Consider our motor plus load again but let us add a structural resonance around 1 kHz with damping-factor 0.1.

$$G(s) = \frac{K}{s(1+sT_m)} \frac{\omega_n^2}{\left(s^2 + 2\zeta\omega_n s + \omega_n^2\right)} \tag{6.28}$$

For values K = 100, $T_m = 3.2\,\text{ms}$, $\omega_n = 6.28 \times 10^3$ rad/s and $\zeta = 0.1$ we get the Bode-plot of Fig. 6.35.

We see the distinct feature of the structural resonance and the sharp negative turn in phase caused by it. It is almost a negative step in phase. The previous example illustrated how a phase-advance is used, but in that example there was nothing to restrict the bandwidth of the system. Here we have a resonance which gives a barrier to higher-frequencies because the phase-margin will suffer if we make it too high. To begin with let us first try and just raise the gain and the bandwidth will rise with it from the present value of around 100 rad/s. Let the cascade compensator be $C(s) = 20$ (Fig. 6.36).

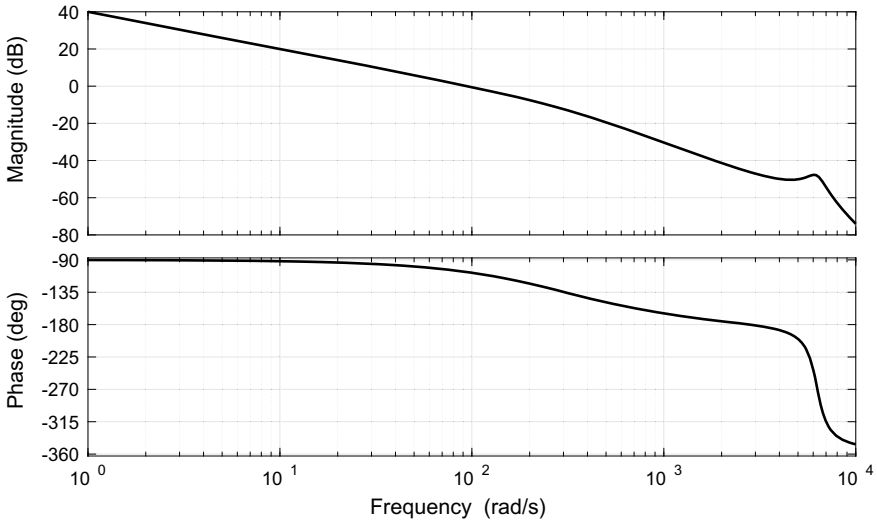


Fig. 6.35 Bode-plot of system with structural resonance

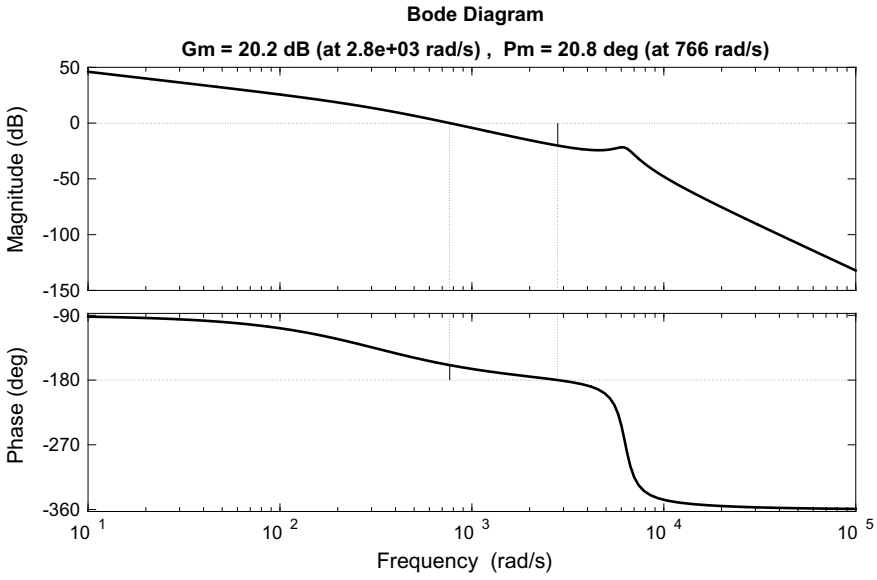


Fig. 6.36 Control, gain only (proportional control)

We now have a phase-margin of 21° and a gain-margin of 20 dB. So an increase in gain of more than 10 will cause instability. The new bandwidth is 766 rad/s. The closed-loop step-response after unity negative feedback is applied is shown in Fig. 6.37.

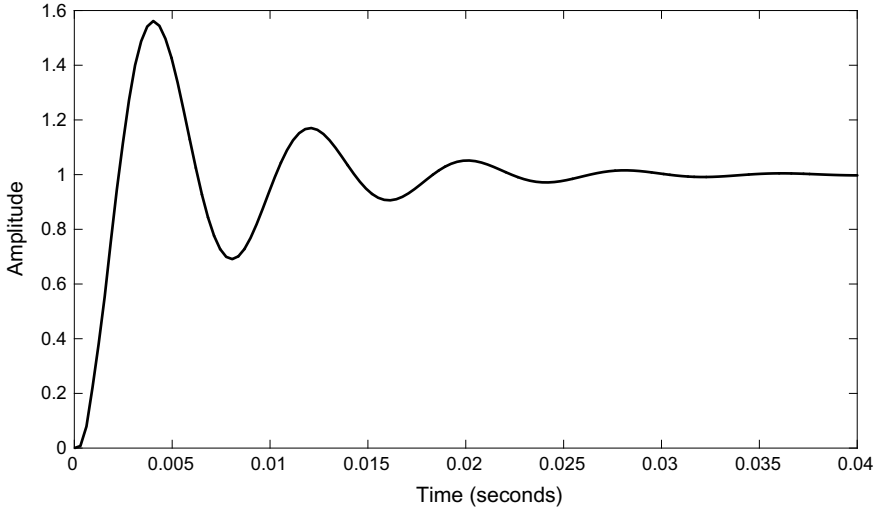


Fig. 6.37 Closed-loop step-response, gain only as controller

The closed-loop system has a 60% overshoot so we attempt to put a phase-advance in place with a span-ratio of $p = \frac{\omega_h}{\omega_\ell} = 10$ as before. We will place the phase-advance at the same unity-gain crossover after the gain was applied, $\omega_\phi = 766$ rad/s. The phase-advance is found from the previously derived formulae for the upper and lower corner-frequencies $\omega_h = \sqrt{p}\omega_\phi$, $\omega_\ell = \frac{\omega_\phi}{\sqrt{p}}$, and then we solve for the time-constants by using $T_2 = \frac{1}{\omega_h}, T_1 = \frac{1}{\omega_\ell}$. The compensator becomes

$$C(s) = \frac{20(1+sT_1)}{\sqrt{p}(1+sT_2)}, T_1 > T_2 \tag{6.29}$$

where $T_1 = 4.1\,\text{ms}, T_2 = 0.41\,\text{ms}$. The original gain of 20 is included in the compensator. This phase-advance is designed for a span of 10:1 and will have a maximum phase of around 55° at 766 rad/s. The compensated Bode-plot is shown in Fig. 6.38.

The phase-margin is now a respectable 76° with a gain-margin of nearly 15 dB. The closed-loop step-response is found in Fig. 6.39. There is still a slight parasitic oscillation present which is shown deliberately to illustrate the problem of structural resonance. A small reduction in gain (at the expense of bandwidth and hence speed of response) will remove this.

Although MATLAB claims the gain-margin is nearly 15 dB and this is technically correct, one look at the Bode-plot and you will see that long before the gain can rise by that amount at the phase-shift of −180°, the peak of the resonance will cross 0 dB creating a second phase-margin which is unstable. So although we can

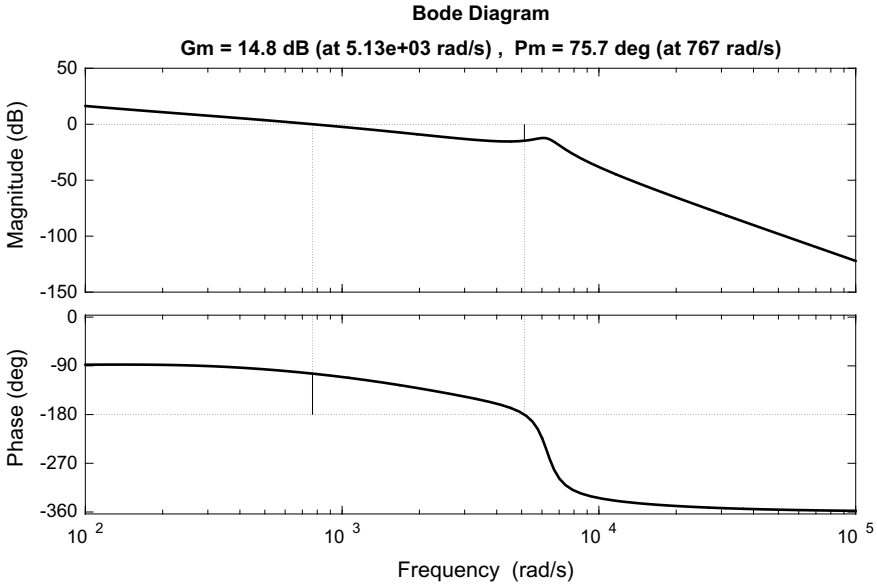


Fig. 6.38 Original system with phase-advance added at 766 rad/s maximum phase

claim the gain-margin is 15 dB, if we increase the gain by far less there will be a second phase-margin and oscillations will occur at that frequency first. In fact the early stages are in fact shown superimposed on the step-response of Fig. 6.39. (slight ripple).

A slightly zoomed Bode-plot is shown in Fig. 6.40 explaining this point.

To prove this further, let us increase the gain by a factor of 2 (6 dB). Remember we should have 15 dB of gain margin. The new step response is shown in Fig. 6.41 showing more oscillations, this is a parasitic oscillation cased by the structural resonance and not by the amount of phase-margin—since phase margin is ok, we have 76° of phase-margin left.

We have increased the speed of response due to the gain change but have run up against a barrier. We can go no further in bandwidth so we must back off to get rid of the oscillation.

Finally we can increase the gain of the loop at low-frequencies so as to cope with a greater level of disturbance. For this we use an integrator as before and take it out (the Bode-plot goes flat) at a higher frequency. Probably a good frequency to stop integrating is the motor-time-constant frequency. We therefore initially design the compensator at low-frequencies to be $C_1(s) = \frac{(1+sT_m)}{s}$ and overall the compensator is then (Figs. 6.42 and 6.43).

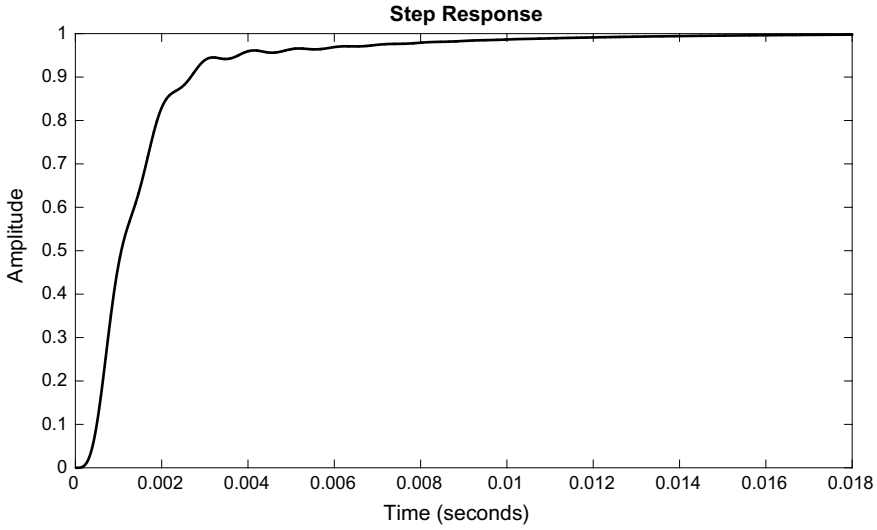


Fig. 6.39 Step-response of compensated system with resonance

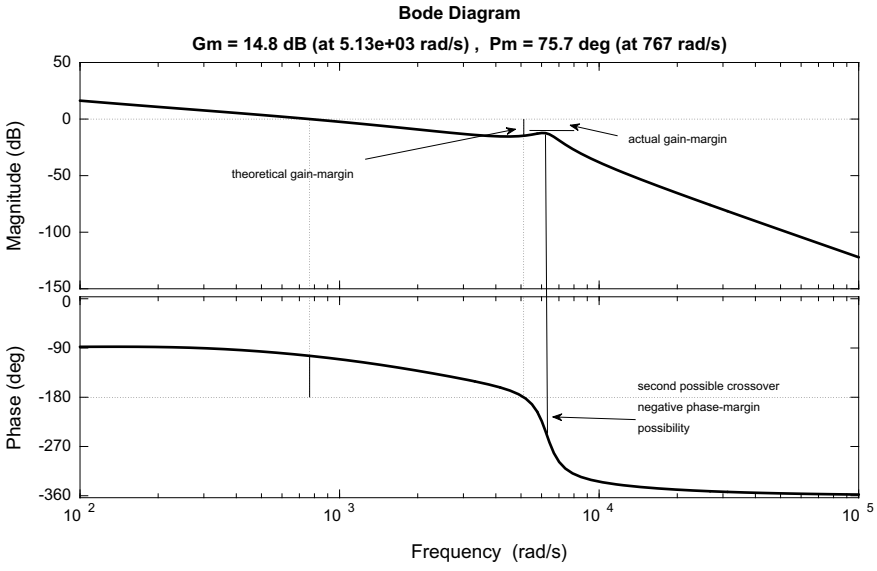


Fig. 6.40 Shows theoretical and actual gain-margin

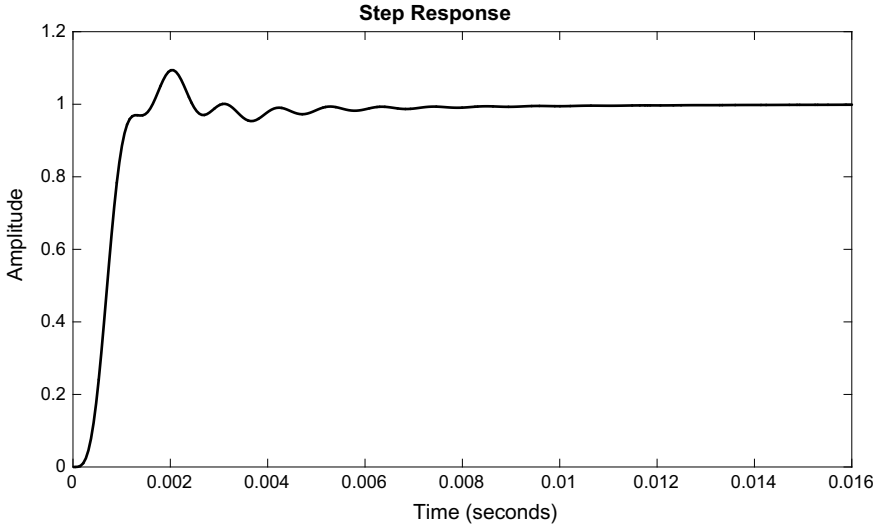


Fig. 6.41 Increase of gain by 6 dB shows parasitic oscillation due to peak in resonance

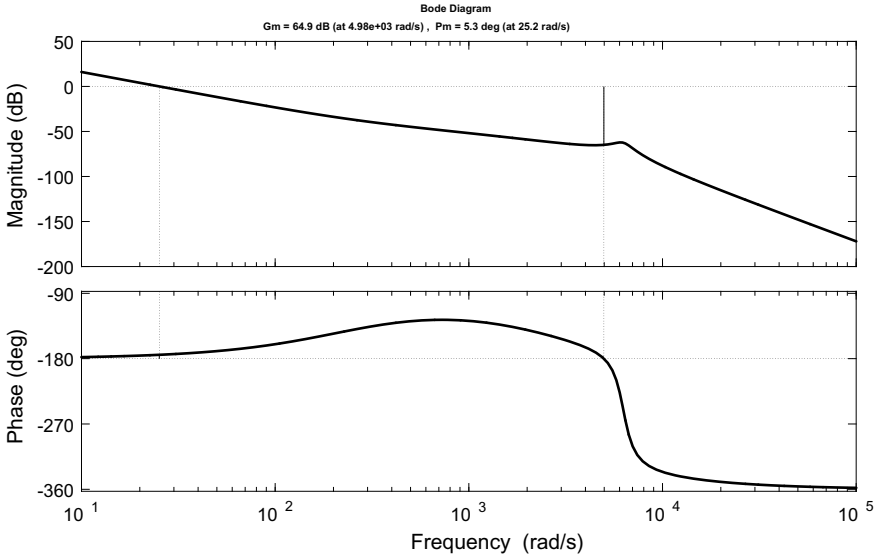


Fig. 6.42 The effect of adding an integrator at low frequencies—Bode-plot

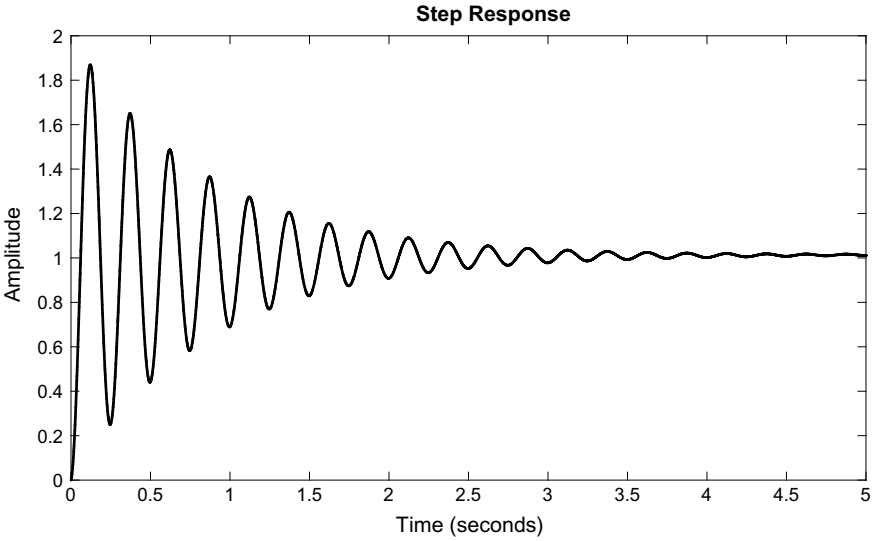


Fig. 6.43 The effect of adding an integrator at low-frequencies—step-response

$$C(s) = \frac{(1 + sT_m)}{s} \frac{20(1 + sT_1)}{\sqrt{p}(1 + sT_2)}, T_1 > T_2 \tag{6.30}$$

What has happened is that our phase-advance is still working but working at the wrong frequency. The Bode-plot crossed 0 dB at 25 rad/s—much too low. The effect are oscillations at 25 rad/s with 5° phase-margin. We must increase the gain to make use of the available phase-margin and also increase the bandwidth at the same time. A quick glance at the Bode-plot tells us that we need around 50 dB of extra gain to get us there. Increasing the gain by 50 dB (about $\times 316$) gives us the following plots (Figs. 6.44 and 6.45).

We have also succeeded in a slight increase in bandwidth to 828 rad/s giving a faster response at the expense of overshoot. The final compensator is

$$C(s) = \frac{316(1 + sT_m)}{s} \frac{20(1 + sT_1)}{\sqrt{p}(1 + sT_2)}, T_1 > T_2 \tag{6.31}$$

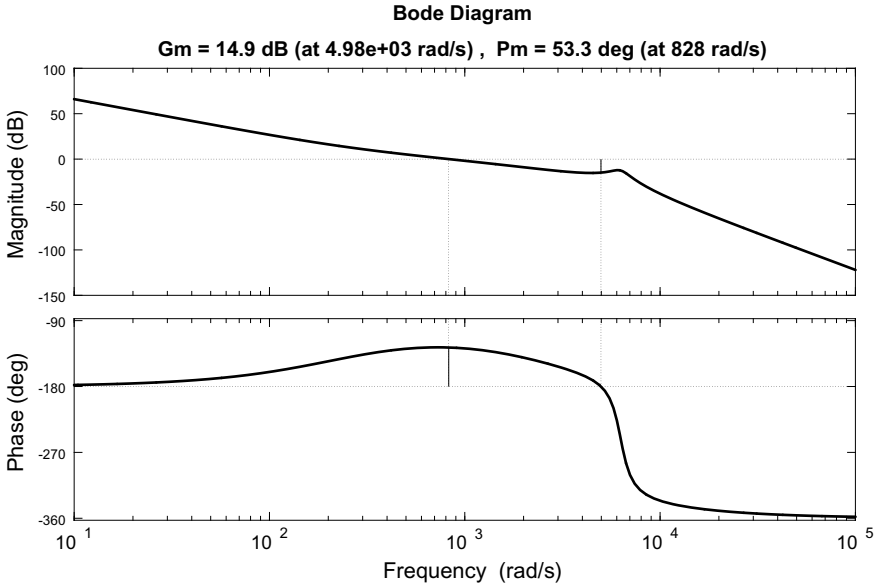


Fig. 6.44 Bode-plot. Increased gain by 50 dB

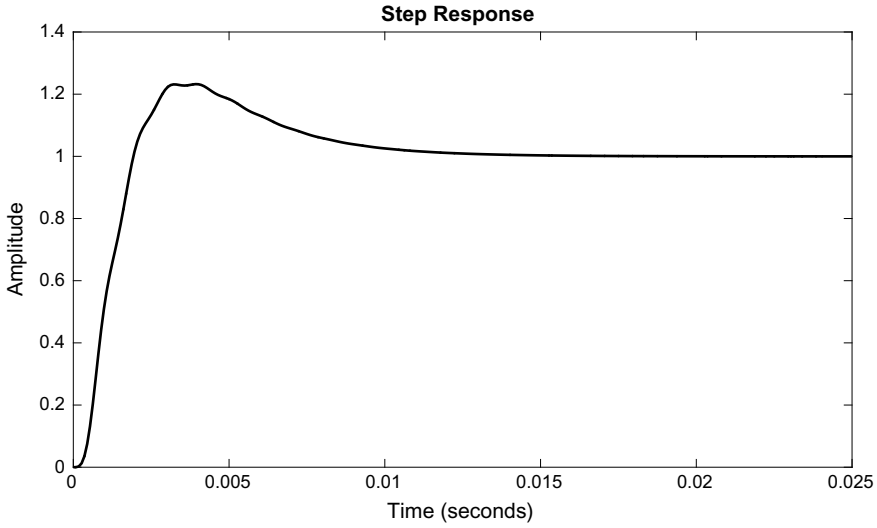


Fig. 6.45 Step-response of compensated system with 50 dB extra gain

The MATLAB code which produced this is shown below.

MATLAB Code 6.4

```
% Compensation for motor with resonance
 K=100;
Tm=3.2e-3;
num=K;
den1=[Tm 1];
g=tf(num,den1)
num2=1;
den2=[1 0];
gi=tf(num2,den2);
gm1=g*gi
%resonance
wn=6.28e3;
zeta=0.1;
num3=[wn*wn];
den3=[1 2*zeta*wn  wn^2];
gr=tf(num3,den3)
gm=gm1*gr
% compensator - phase-advance
w0=766;
span=10;
x=sqrt(span)

wh=x*w0;
wl=wh/span;
T2=1/wh
T1=1/wl

num4=20*[T1 1];
den4=x*[T2  1];
gc=tf(num4,den4);

% add PI at low frequencies
num5=[Tm 1];
den5=[1 0];
gpi=tf(num5,den5)

% composite compensator
gt=316*gc*gm*gpi;
bode(gt,{100,100000})
grid
margin(gt)
```

```
figure
%Close the loop
gcl=gt/(1+gt);
step(gcl)
```

Of course this isn't necessarily the end of the design process, but we can see that the process is constant re-evaluation and changing of parameters to shape the Bode-plot into something resembling the ideal shape. Many points have been raised in these examples that have rarely if ever appeared in other text-books on the subject. This is the difference between industrial design of controllers and the more academic approach. The only way to learn is to sit down and experiment either with simulations and preferably with the real system on the bench. The important points to remember is that there is always an upper limit for bandwidth caused by structural resonance (because the phase of a resonance is so severe we cannot approach too close to it). Therefore, a trade-off has been made between maximum speed of response and phase-margin (that is, the amount of overshoot or damping). The design method is a constantly improving bandwidth and by adding phase-advance (lead) where necessary. At the same time we require a high gain at low-frequencies to get good tracking for all types of input and good disturbance rejection.

6.6 PID Controllers and Auto-tuning

The previous section is the definitive approach to design of servo-systems using Bode-design. To achieve the goals however it is first necessary to have an accurate model of the system. With motors, amplifiers and many electrical systems this may be difficult but not impossible to achieve. However, there are many other problems where the model is not so well known and a more generic approach is required to feedback-control. One that is far simpler yet with results that can be similar. For a

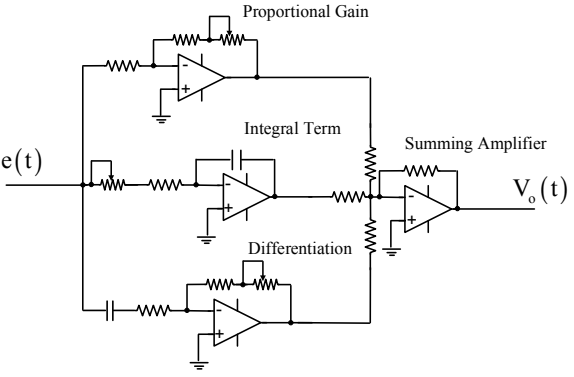


Fig. 6.46 PID controller

great many applications, the so-called proportional plus integral plus derivative controller (PID) is the way to design controllers. Consider the circuit of such a controller (Fig. 6.46).

The circuit is composed of three parallel terms (a three-term controller). The first is just proportional gain as we have met before, the second is a pure integrator and the third is a differentiator. The three terms all have adjustment potentiometers to adjust the values of gain, integration gain and differential gain respectively. The differentiator as we have already shown cannot be ideal, so it only differentiates to a certain frequency and then becomes a constant gain. However, when doing the mathematics on this circuit it is easier to simply the problem and assume an ideal differentiator. The circuit usually acts on the error of the control-system as follows:

$$V_o(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \tag{6.32}$$

where the three gain terms K_P, K_I, K_D are known respectively as the proportional, integral and derivative gains. Taking Laplace transforms of (6.32) we find

$$V_o(s) = \left(K_P + \frac{K_I}{s} + K_D s\right) e(s) \tag{6.33}$$

In the circuit and equations for this topology of PID the proportional gain is independent of other gain terms. Sometimes a PID is written in a slightly alternative form for the ideal differentiator case as $V_o(s) = K\left(1 + \frac{1}{sT_I} + sT_D\right)e(s)$. The three-terms are best explained as follows.

6.6.1 Proportional Control K_P Only

Using a proportional gain of $C(s) = K_P$ and no integral or derivative terms we find that the closed-loop step-response will often have an error. That is to say the steady-state error is often finite rather than zero. Of course this depends on the type of system. For some systems (those with say a mechanical integrator) there will be no error. The very least a proportional term can do therefore is to speed up the closed-loop system. There is a limit to how far the gain can be pushed, and beyond some point the inevitable happens and the system starts oscillating. Therefore, if we want a faster response we are stuck because the system just oscillates more and more as the proportional gain increases.

6.6.2 Proportional Plus Derivative Control. (PD) $K_P + K_D s$

Derivative control (or derivative action as it is sometimes called) $C(s) = K_P + K_D s$ when added acts the same as phase-advance, it provides damping to the system. Adding a derivative term therefore will damped out any oscillations introduced by the proportional term.

6.6.3 Proportional Plus Integral Control. (PI) $K_P + \frac{K_I}{s}$

Proportional plus integral control $C(s) = K_P + \frac{K_I}{s}$ will reduce steady-state error to zero and provide good tracking. In fact, we have already used PI control in a different form. In Sects. 6.5 and 6.4 we used a compensator of the form $C(s) = K\frac{(1+sT)}{s}$. If we divide by s we get $C(s) = KT + \frac{K}{s}$ which is the same thing as PI control. A PI controller adds an integrator at low frequencies. It has a Bode-plot which is a pure integrator that then goes flat into a constant gain at frequency $\omega = \frac{1}{T}$.

6.6.4 Proportional Plus Integral Plus Derivative Control (PID) $K_P + \frac{K_I}{s} + K_D s$

A PID controller gives us the benefits of both PI and PD. It can reduce errors (high-gain at low frequencies) by using an integrator, and it can reduce oscillations and instability due to the derivative term.

6.6.5 Comparison with Lag-Lead Type Control

In the previous sections we use a PI type controller and phase-lead in cascade. If we closely examine the transfer function of such a controller

$$C(s) = \frac{K(1+sT_1)}{s}\frac{(1+sT_2)}{(1+sT_3)}, T_1 > T_2 > T_3 \tag{6.34}$$

Note: A PI controller is just a kind of phase-lag controller with an ideal integrator. A phase-lag only differs in that it has a transfer-function of the form $\frac{(1+sT_a)}{(1+sT_b)} T_a < T_b$ rather than $\frac{(1+sT_a)}{s}$. That is, the denominator is a pure integrator.

Multiply out (6.34) and re-arrange and we get

$$C(s) = K\left[\frac{1}{s} + (T_1 + T_2) + sT_1T_2\right]\frac{1}{(1+sT_3)} \tag{6.35}$$

Compare (6.35) to (6.33) and we see it has the same form except a low-pass filter consisting of $\frac{1}{(1+sT_3)}$ appears in cascade. In fact our PID controller does benefit from an extra pole at high-frequencies and we can say that the two approaches are identical. In general of course we can always add more than one phase-advance or lag-compensator to the lag-lead approach and it is not so restricted in form.

6.6.6 PID Example

Consider the same example as one which used lag-lead control. Here we use a shortcut method for design, the MATLAB auto-tuning facility. The system is given as

$$G(s) = \frac{K}{s(1+sT_m)}\frac{\omega_n^2}{\left(s^2 + 2\zeta\omega_n s + \omega_n^2\right)} \tag{6.36}$$

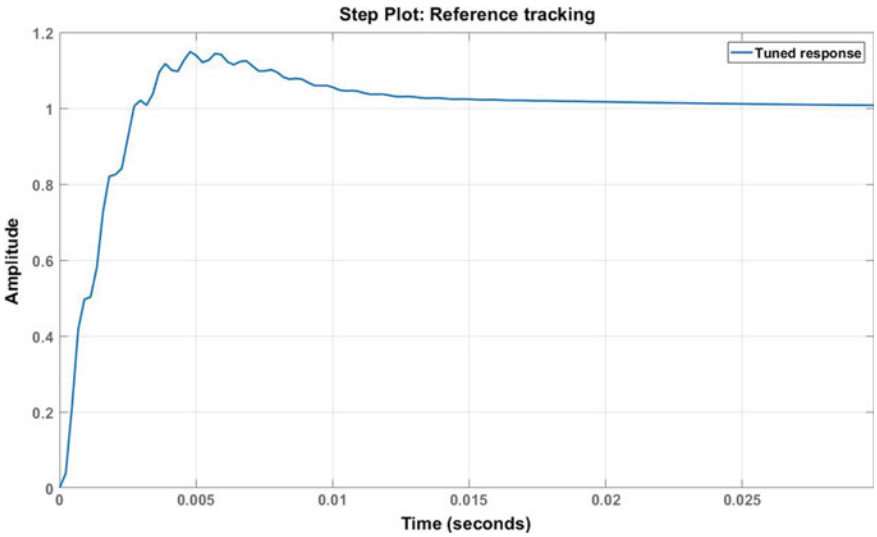


Fig. 6.47 Auto-tuning using MATLAB *pidtool ()*. Step-response in closed-loop for PID

For values K = 2000, $T_m = 3.2$ ms, $\omega_n = 6.28 \times 10^3$ rad/s and $\zeta = 0.1$. We use MATLAB *auto-tuning* method *pidtool ()* for fastest response and it gives us the step-response of Fig. 6.47. The auto-tuner found the optimal values as $K_P = 0.548$, $K_I = 35.52$, $K_D = 0.000916$.

The results are very similar to our manual design using lag-lead. The MATLAB code is shown below.

MATLAB Code 6.5

```
% PID auto MATLAB compensation for motor with structural resonance

%%%%%%%%%%%%%%
K=2000;
Tm=3.2e-3;
num=K;
% Motor time-constant
den1=[Tm 1];
g=tf(num,den1)
%%%%%%%%%%%%%%
%Integrator
num2=1;
den2=[1 0];
gi=tf(num2,den2);
%%%%%%%%%%%%%%
gm1=g*gi

%resonance
wn=6.28e3;
zeta=0.1;
num3=[wn*wn];
den3=[1 2*zeta*wn  wn^2];
gr=tf(num3,den3)
%%%%%%%%%%%%%%
%Total open-loop system
gm2=gm1*gr
%%%%%%%%%%%%%%
% Auto-tune
pidtool(gm2)
```

We should never belittle the power of the simple PID controller. They are used extensively in industry and in the very least a great place to start when closing a loop if there is little known about the system dynamics. Tuning is only difficult in problems that are open-loop unstable. For example, two-wheel self-balancing personal transport use PID control successfully, but tuning them can sometimes be a difficult experience when oscillations occur in such large mechanical systems.

6.7 The Type of a System

Many text-books talk of the type of a system as being the number of pure integrators present in the loop (or the system itself before compensation). This book avoids it by assuming there is such a thing as the ideal Bode-plot. The ideal-Bode-plot is essentially saying that we need as many integrators as possible to squeeze into get the gain high at low-frequencies. Usually one is not enough. However, a steady-state analysis for different types of system is a useful learning tool and we present it herein.

Suppose we have an open-loop system

$$G(s) = \frac{(b_0 + b_1 s + b_2 s^2 + \cdots + b_n s^n)}{(1 + a_1 s + a_2 s^2 + \cdots + a_n s^n)} \tag{6.37}$$

If the denominator polynomial is s has a term which is not unity, then we can divide numerator and denominator by that term and always end up with (6.37), provided the transfer-function has no pure integrators within it. The dc gain $G(0) = b_0$ is therefore finite. We term this type of system a *Type-0* system since there are zero integrators within it. Now we consider the effect of unity negative feedback around this transfer-function on the error. The error is written in terms of the system input $r(s)$. We assume the closed-loop system *is stable* otherwise the error will grow without limit.

$$e(s) = \frac{1}{1 + G(s)} r(s) \tag{6.38}$$

Now we consider two types of input, a unit-step and unit-ramp. These are two very common types of input to a control-system.

6.7.1 *Type-0 System: Step-Input* $r(s) = 1/s$

$$e(s) = \frac{1}{1 + G(s)} r(s)$$

$$e(s) = \frac{1}{s}\left(\frac{1}{1 + G(s)}\right)$$

Now we can use the final-value theorem to find the steady-state value of the error. Multiply by s and let s tend to zero.

$$e(t)|_{t\to\infty} = \frac{s}{s}\left(\frac{1}{1 + G(s)}\right)\Bigg|_{s\to 0}$$

So the steady-state error is

$$e_{ss} = \left(\frac{1}{1+G(0)}\right) \tag{6.39}$$

This is finite, $e_{ss} = \left(\frac{1}{1+b_0}\right)$ and never zero. Now consider a ramp input.

6.7.2 Type-0 System: Ramp Input $r(s) = 1/s^2$

Use the same analysis

$$e(s) = \frac{1}{s^2}\left(\frac{1}{1+G(s)}\right)$$

$$e(t)|_{t\to\infty} = \frac{s}{s^2}\left(\frac{1}{1+G(s)}\right)\Bigg|_{s\to 0}$$

The steady-state error is now

$$e_{ss} = \left(\frac{1}{s+sG(0)}\right)_{s=0} \tag{6.40}$$

which is infinite. Therefore we conclude that a type-0 system with no integrators cannot follow a ramp input but can follow a step with give a finite-error in steady-state.

6.7.3 Type-1 System: Step Input $r(s) = 1/s$

Now we assume the system has a pure integrator in cascade (hence the name type-1 for one integrator). Repeat the above analysis.

$$e(s) = \frac{1}{1+\frac{1}{s}G(s)}r(s)$$

The steady-state error is found using the final-value-theorem

$$e(t)|_{t\to\infty} = \frac{s}{s}\left(\frac{1}{1+\frac{1}{s}G(s)}\right)\Bigg|_{s\to 0}$$

$$e_{ss} = \left(\frac{1}{1+\infty}\right) = 0 \tag{6.41}$$

So a type-1 system gives zero-steady-state error to a step input. This is why integrators are so desirable in a closed-loop system.

6.7.4 Type-1 System: Ramp Input $r(s) = 1/s^2$

$$e(s) = \frac{1}{s^2}\left(\frac{1}{1+\frac{1}{s}G(s)}\right)$$

$$e(t)|_{t\to\infty} = \frac{s}{s^2}\left(\frac{1}{1+\frac{1}{s}G(s)}\right)\Bigg|_{s\to 0}$$

$$\begin{aligned} e_{ss} &= \left(\frac{1}{s+\frac{s}{s}G(s)}\right)\Bigg|_{s\to 0} \\ &= \frac{1}{G(0)} \end{aligned} \tag{6.42}$$

We conclude that a type-1 system has a finite error in steady-state for a ramp-input.

Now consider a type-2 system that has two integrators.

6.7.5 Type-2 System: Step Input $r(s) = 1/s$

$$e(s) = \frac{1}{1+\frac{1}{s^2}G(s)}r(s)$$

The steady-state error is found using the final-value-theorem

$$e(t)|_{t\to\infty} = \frac{s}{s}\left(\frac{1}{1+\frac{1}{s^2}G(s)}\right)\Bigg|_{s\to 0}$$

$$e_{ss} = \left(\frac{1}{1+\infty}\right) = 0 \tag{6.43}$$

Once again we have zero steady-state error to a step-input.

6.7.6 Type-2 System: Ramp Input $r(s) = 1/s^2$

$$\begin{aligned} e(s) &= \frac{1}{s^2}\left(\frac{1}{1+\frac{1}{s^2}G(s)}\right) \\ e(t)|_{t\to\infty} &= \frac{s}{s^2}\left(\frac{1}{1+\frac{1}{s^2}G(s)}\right)\Bigg|_{s\to 0} \\ e_{ss} &= \left(\frac{1}{s+\frac{s}{s^2}G(s)}\right)\Bigg|_{s\to 0} \\ &= \frac{1}{0+\infty} \\ &= 0 \end{aligned} \tag{6.44}$$

We conclude that a type-2 system has zero steady-state error to a ramp input. Clearly a type-2 system has the best properties of all the others provided we can keep it stable. It is possible to consider type-3 systems but there becomes stability problems when we close the loop. Such systems are hard to keep stable at all. We therefore stop at type-2 and use two integrators where possible. One integrator may well be already there in mechanical form (for a dc-motor or a system that measures angular position, since position is the integral of velocity). We can add a second electrical integrator and usually do. When we add a second integrator however, we usually do not add a pure integrator as this also leads to instability. Instead we add an integrator at low-frequencies and take it out at some higher-frequency with a zero. That is to say, the form $C(s) = \frac{1+sT}{s}$ is an integrator that goes flat at a frequency $1/T$ rad/s. This is once again the familiar proportional plus integral (PI) controller. As a final note we can also consider a more severe type of input to a system, a parabolic input $r(t) = \frac{1}{2}t^2$ has Laplace-transform $r(s) = 1/s^3$. Using the same analysis we can show that a type-2 system has finite steady-state error to a parabolic input. So it cannot follow it without error, but this is still a good result for such an input.

Chapter 7
State-Space System Descriptions

The 1960s and 1970s saw a return from the transfer-function approach to differential-equations. Usually engineers would avoid such approaches but this was not just ODEs in the classical sense, that is to say, solving countless such equations of differing types using a variety of mathematical methods. This was a special class of ODE, a vector-based approach that had nicely formed solutions that made sense from the point of view of engineering. The motivating factors were as follows. Engineers needed to be able to cope with multiple-input, multiple-output systems (Multivariable). Such systems arose in aircraft and missile-systems and even the space-race. State-variables (or state-space as it is more commonly called) offered a method of describing systems that made it far easier to cope with multivariable systems. The Laplace-transform operator method is ingenious but also fails when systems that vary with time are considered or even non-linear systems. This gave us the birth of state-variable descriptions and control-laws based on the theory. Transfer-functions didn't die however, there was a fight back mainly by the British researchers who considered transfer-function matrices (a matrix of transfer-functions) to describe multivariable systems and later polynomial-based systems emerged as a more elegant way of describing and solving problems. This chapter considers the state-variable methods hereafter referred to as state-space.

7.1 Introduction to State-Space

It is best to begin with a very simple problem, that of a first-order system. Systems described in such a way in Laplace format would have a transfer-function such as

$$G(s) = \frac{1}{1 + sT} \tag{7.1}$$

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_7

where T is the time-constant of the system. If we recall, these transfer-functions arise from ordinary-differential equations (ODEs). We can work backwards to the ODE

$$y(s) = \frac{1}{1+sT}u(s)$$

Which can be written as

$$y(s)(1+sT) = u(s) \tag{7.2}$$

Use inverse-Laplace and return to the time-domain

$$T\dot{y}(t) + y(t) = u(t) \tag{7.3}$$

Now divide by the time-constant T and re-arrange (using the dot notation for derivative)

$$\dot{y}(t) = -\frac{1}{T}y(t) + \frac{1}{T}u(t) \tag{7.4}$$

Re-define some variables. Let

$x(t) = y(t), a = -\frac{1}{T}, b = \frac{1}{T}$ (note that a will be always negative for a stable system)

Then we can re-write (7.4) as a first-order differential equation in state-space format

$$\dot{x}(t) = ax(t) + bu(t) \tag{7.5}$$

This we call a scalar state-space system. It is an ODE of the first-order and we can solve it to find $x(t)$.

Take (7.5) and write

$$\dot{x}(t) - ax(t) = bu(t); x(0) = x_0 \tag{7.6}$$

Introduce an integrating factor to make the left-hand side of (7.6) a differential of the form.

$$e^{-at}(\dot{x}(t) - ax(t)) = \frac{d}{dt}(e^{-at}x(t)) \tag{7.7}$$

(since if we use the product-rule on the right-hand side we get the left-hand side of (7.7)).

Multiply (7.6) by e^{-at} and we must get

$$\frac{d}{dt}(e^{-at}x(t)) = e^{-at}bu(t) \tag{7.8}$$

Integrate with limits 0 to time t both sides of (7.8). Note the change of variable is needed

$$e^{-at}x(t) - x(0) = \int_0^t e^{-a\tau}bu(\tau)d\tau \tag{7.9}$$

and we get

$$e^{-at}x(t) - x(0) = \int_0^t e^{-a\tau}bu(\tau)d\tau \tag{7.10}$$

Then

$$x(t) = e^{at}x(0) + e^{at}\int_0^t e^{-a\tau}bu(\tau)d\tau \tag{7.11}$$

and finally

$$x(t) = e^{at}x_0 + \int_0^t e^{a(t-\tau)}bu(\tau)d\tau \tag{7.12}$$

Equation (7.12) is composed of two parts, the response of the system to initial conditions (the unforced-solution) and the addition of the forced part which is a convolution integral. Just to show that this actually works, consider the first-order system with zero-initial conditions and let us solve (7.12) when the input is constant u = 1, a step input. From (7.12)

$$x(t) = \int_0^t e^{a(t-\tau)}bd\tau \tag{7.13}$$

$$x(t) = e^{at}b\left[-\frac{1}{a}e^{-a\tau}\right]_{\tau=0}^{\tau=t}$$

$$= -e^{at}\frac{b}{a}[e^{-at} - 1]$$
$$= -\frac{b}{a} + \frac{b}{a}e^{at}$$

But $\frac{b}{a} = -1$ resulting in

$$x(t) = y(t) = 1 - e^{-t/T} \tag{7.14}$$

Which is the standard first-order step-response we derived earlier in the book.

Although important for later work, this result only applies to first-order system and we need a more general approach. What we do is write a vector form of Eq. (7.5).

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \tag{7.15}$$

We use bold here to define vectors and **A** and **B** are also bold as they are a matrix and vector respectively. For a single-input single-output (SISO) system the input u is of course scalar though we have room to increase dimension by making it a vector for the multivariable case. For the present, we consider SISO systems only and then the **A** matrix is n × n square with **B** a vector of size n. The state-vector is defined as

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ . \\ . \\ . \\ x_n(t) \end{bmatrix} \tag{7.16}$$

We can also write (7.16) in row form by using the transpose notation

$$\mathbf{x}(t) = [\, x_1(t) \quad x_2(t) \quad . \quad . \quad . \quad x_n(t) \,]^T.$$

The state vector and individual states can have physical values or they can be purely mathematical in nature. First consider the more mathematical approach where we define our states in a special form. Formats that form specific patterns of matrices are termed *canonical* forms.

7.1.1 Example of State-Space Realisation

Consider the second-order transfer-function

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{7.17}$$

Using the same technique as we did above we can work back to the ODE that generated this

$$y(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} u(s)$$

$$y(s)\left[s^2 + 2\zeta\omega_n s + \omega_n^2\right] = \omega_n^2 u(s)$$

Take inverse Laplace Transforms

$$\ddot{y}(t) + 2\zeta\omega_n \dot{y}(t) + \omega_n^2 y(t) = \omega_n^2 u(t) \tag{7.18}$$

This is second-order and therefore we must use 2 states. How we choose these states is not unique but if we chose a convenient form

$$\begin{aligned} x_1(t) &= y(t) \\ x_2(t) &= \dot{y}(t) \end{aligned} \tag{7.19}$$

(and we can continue this idea of course for any order system).

We require the derivatives of the states as in (7.15). From (7.19)

$$\begin{aligned} \dot{x}_1(t) &= \dot{y}(t) \\ &= x_2(t) \end{aligned}$$

and from (7.18)

$$\begin{aligned} \dot{x}_2(t) &= \ddot{y}(t) \\ &= -2\zeta\omega_n \dot{y}(t) - \omega_n^2 y(t) + \omega_n^2 u(t) \\ &= -2\zeta\omega_n x_2(t) - \omega_n^2 x_1(t) + \omega_n^2 u(t) \end{aligned} \tag{7.20}$$

Writing all this in vector form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega_n^2 & -2\zeta\omega_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \omega_n^2 \end{bmatrix} u(t) \tag{7.21}$$

We are not finished yet however, as although this describes the states we defined there is no equation for the output $y(t) = x_1(t)$. We must also write this in terms of the state-vector thus

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.22}$$

This last equation we write in general as

$$y(t) = \mathbf{C}\mathbf{x}(t) \tag{7.23}$$

The output and inputs are scalar quantities and not written in bold. **C** is a row vector of dimension *n*. We use a graphical representation of the states in a form known as the *signal-flow graph*. This is quite useful to see the structure of the system. In fact, it is a little bit like a block-diagram but without any summing junctions. Instead, we use a dot or node, which does the same thing. It *always sums* however and there are no subtractions. Any negative terms are signed in the feedback path. We use the definition of the states and integrators with feedback. To illustrate this consider our second-order system. First, we write the block-diagram (Fig. 7.1).

In the block-diagram we can reduce the inner loop and then the outer loop to arrive at the transfer-function provided we replace the integrators with their Laplace equivalent 1/s. The inner-loop becomes $\frac{1}{(s+2\zeta\omega_n)}$. Reducing the outer-loop gets us back to the transfer-function $\frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$ (Fig. 7.2).

Integral signs and Laplace integration symbols are interchangeable depending if the graph is in the s or time-domains. The block-diagram is similar to the old techniques used in *Analogue Computing*. Any system can be broken down in this way.

However, we could have picked the states entirely differently. For example, suppose we had picked instead

$$\begin{aligned} x_1(t) &= \dot{y}(t) \\ x_2(t) &= y(t) \end{aligned} \tag{7.24}$$

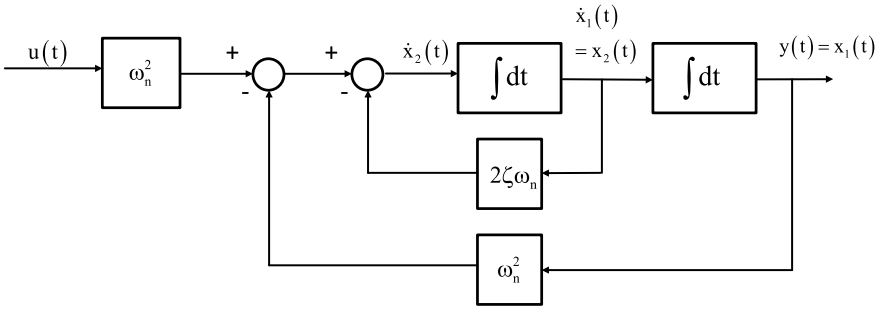


Fig. 7.1 Block-diagram of second-order system

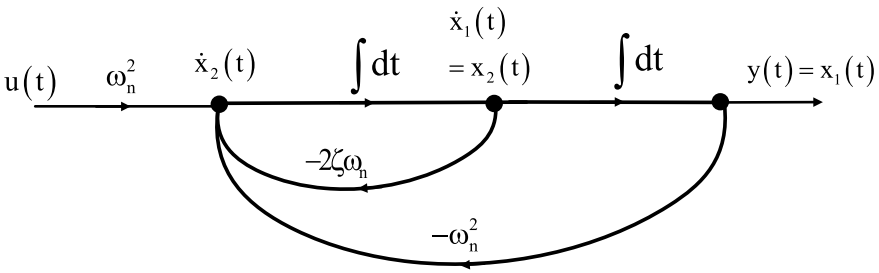


Fig. 7.2 Signal-flow graph of second-order system. First realisation (7.21), (7.22)

Then

$$\begin{aligned}\dot{x}_2(t) &= x_1(t)\\ \dot{x}_1(t) &= \ddot{y}(t)\\ &= -2\zeta\omega_n x_1(t) - \omega_n^2 x_2(t) + \omega_n^2 u(t)\end{aligned}$$

Or in matrix format

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-2\zeta\omega_n & -\omega_n^2\\ 1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}\omega_n^2\\ 0\end{bmatrix}u(t) \tag{7.25}$$

$$y(t) = \begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} \tag{7.26}$$

Figure 7.3 Shows the signal-flow graph for this realisation.

The only difference is that the outputs of each of the integrators has been labelled the other-way around with the other state.

Another state-space realisation is as follows:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-2\zeta\omega_n & 1\\ -\omega_n^2 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ \omega_n^2\end{bmatrix}u(t) \tag{7.27}$$

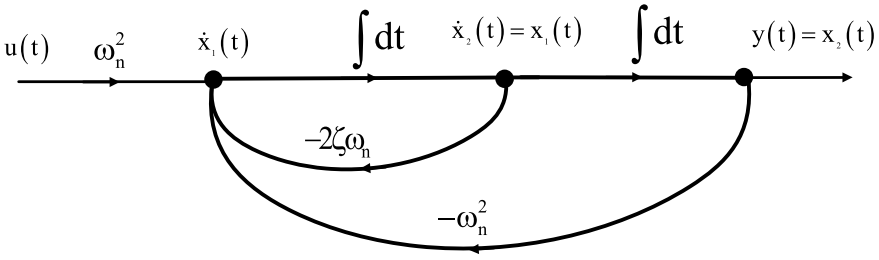


Fig. 7.3 Different realisation of states for second-order system. Second realisation (7.25), (7.26)

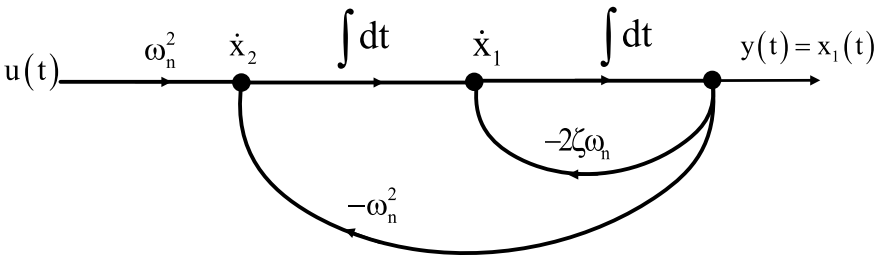


Fig. 7.4 Signal-flow graph for Eqs. (7.27), (7.28). Third realisation

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.28}$$

See Fig. 7.4 for the Signal-flow graph.

While we are at it we may as well re-label the integrator outputs on this one and it gives us a further possible realisation (Fig. 7.5).

The state-space realisation corresponding to this signal-flow graph is:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & -\omega_n^2 \\ 1 & -2\zeta\omega_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \omega_n^2 \\ 0 \end{bmatrix} u(t) \tag{7.29}$$

$$y(t) = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.30}$$

Any doubt that these are in fact all the same transfer-function from input to output can be dispelled easily by reducing (simplifying) the signal-flow graphs to get the transfer-function. This is done in almost an identical fashion to that of block-diagrams except that since the nodes add and do not subtract the formula changes slightly. This is illustrated in Fig. 7.6 and can be applied to any of these diagrams if we use "nesting" reduction. We reduce the inner loop first then outer loop.

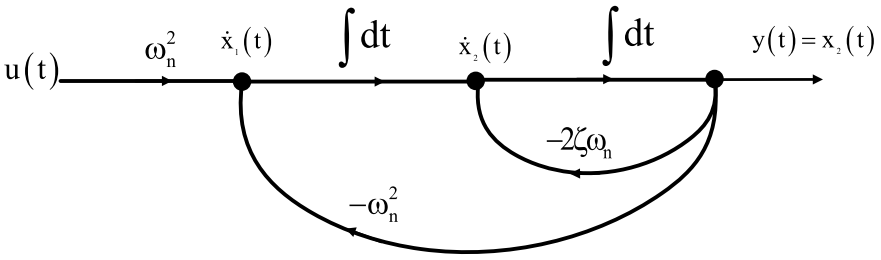


Fig. 7.5 Signal-flow graph. Fourth realisation of second-order system (7.29), (7.30)

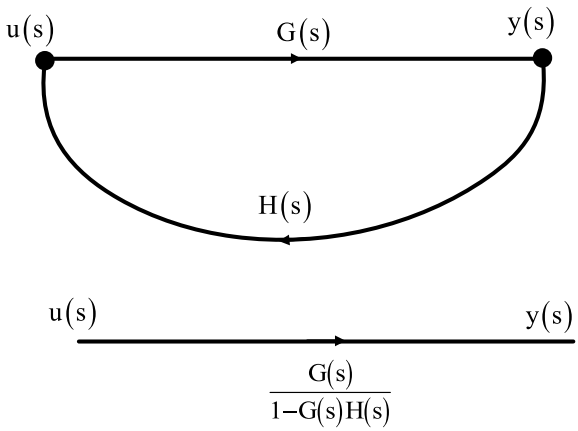


Fig. 7.6 Signal-flow-graph reduction rule

For example in Fig. 7.5 we first replace the integrator with 1/s and reduce the inner-loop. This gives us a new inner-loop transfer-function $\frac{1/s}{1/s+2\zeta\omega_n} = \frac{1}{1+2\zeta\omega_n s}$. We then as in block-diagram reduction multiply by the cascaded integrator 1/s and get a new forward-path transfer-function $\frac{1}{s(1+2\zeta\omega_n s)}$ and it has a feedback-term $-\omega_n^2$. Combining these gives us $\frac{\frac{1}{s(1+2\zeta\omega_n s)}}{1+\frac{\omega_n^2}{s(1+2\zeta\omega_n s)}} = \frac{1}{s^2+2\zeta\omega_n s+\omega_n^2}$. This is then in turn multiplied by the forward-path term ω_n^2 which returns us to the original transfer-function. There is a generalised set of signal-flow graph reduction rules (called *Mason's Rule*) which are rarely used nowadays since we have simulation tools like MATLAB at our fingertips, but occasionally they are useful as a learning experience.

7.1.2 State-Space Realisations with Zeros

Consider now a system with a zero as well as two poles.

$$G(s) = \frac{b_0 + b_1 s}{s^2 + a_1 s + a_0} \tag{7.31}$$

The trick to describe such systems is to introduce an intermediate variable in-between the poles and zeros thus (Fig. 7.7).

We can then treat the poles as before where x is the system output. One-such possibility and the easiest to see is to begin with

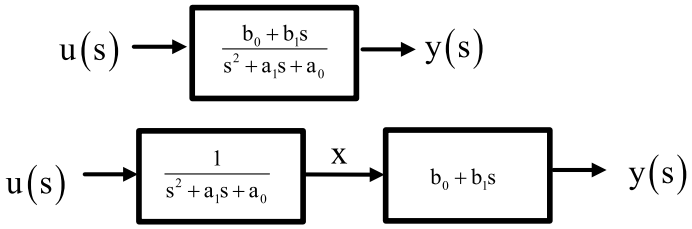


Fig. 7.7 Introduce an intermediate variable x

$$\frac{x}{u}(s) = \frac{1}{s^2 + a_1 s + a_0} \tag{7.32}$$

Working backwards to its differential equation (just cross-multiply and inverse-Laplace) we get

$$\ddot{x}(t) + a_1 \dot{x}(t) + a_0 x(t) = u(t) \tag{7.33}$$

We often drop the argument (t) for simplicity and proceed as follows

$$\begin{aligned} x_1 &= x \\ x_2 &= \dot{x} \\ \ddot{x} &= -a_1 x_2 - a_0 x_1 + u \end{aligned}$$

This gives as most of the state-space description. The rest is found from the zeros part

$$y(s) = (b_0 + b_1 s)x(s) \tag{7.34}$$

Inverse Laplace-transforming gives us

$$y = b_0 x + b_1 \dot{x}$$

Using the same state-variables $(x_1 = x, x_2 = \dot{x})$ we get

$$y = b_0 x_1 + b_1 x_2$$

The state-space realisation is therefore

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \tag{7.35}$$

$$y(t) = \begin{bmatrix} b_0 & b_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.36}$$

The signal-flow graph is shown in Fig. 7.8.

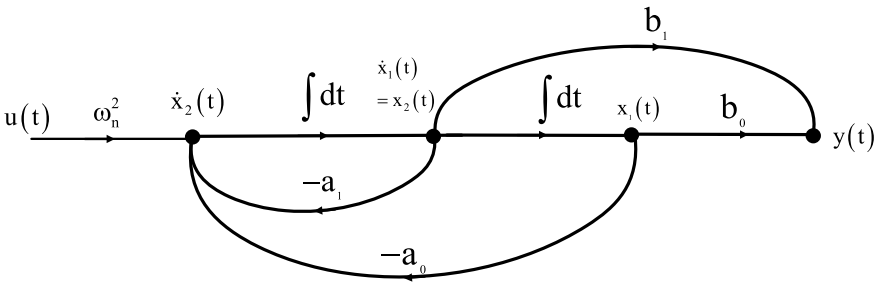


Fig. 7.8 Signal-flow graph including a zero numerator term

7.2 Canonical Forms

A canonical form, is a state-space realisation that forms a pattern and has certain properties that are in our favour. The simplest of these canonical forms is the *phase-variable* canonical form. This is sometimes also called *controllable-canonical* form. In fact we already know this from the previous section. Consider an nth-order system represented with an ODE as follows

$$\frac{d^n y}{dt^n} + a_{n-1}\frac{d^{n-1}y}{dt^{n-1}} + a_{n-2}\frac{d^{n-2}y}{dt^{n-2}} + \ldots + a_0 y = b_{n-1}\frac{d^{n-1}u}{dt^{n-1}} + b_{n-2}\frac{d^{n-2}u}{dt^{n-2}} + \ldots + b_0 u \tag{7.37}$$

Note that the coefficient of $\frac{d^n y}{dt^n}$ is unity. If it is not unity then we can divide through both sides of the equation by it and still arrive at (7.37). we can also see that the order of the b terms is one less than that of the poles. The case where they are equal will be shown later and we recall that it is not possible to have more zeros than poles for a physically realisable continuous-time system. Define an intermediate variable as before and use the pattern

$$\begin{aligned} x_1 &= x \\ x_2 &= \dot{x} \\ x_3 &= \dddot{x} \end{aligned}$$

and so on, we eventually obtain the canonical form

$$A = \begin{bmatrix} 0 & 1 & 0 & . & . & . & 0 & 0 \\ 0 & 0 & 1 & . & . & . & 0 & 0 \\ . & . & . & & & & . & . \\ . & . & . & & & & . & . \\ . & . & & & & & . & . \\ 0 & 0 & 0 & . & . & . & 0 & 1 \\ -a_0 & -a_1 & -a_2 & . & . & . & -a_{n-2} & -a_{n-1} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ . \\ . \\ . \\ 1 \end{bmatrix}, C = [\, b_0 \quad b_1 \quad . \quad . \quad . \quad b_n \quad b_{n-1} \,]$$

If we remember this pattern, then at least for SISO system we can very quickly write down the state-space description directly. The matrix $\boldsymbol{A}$ is known as a *companion* matrix.

7.2.1 Example, Transfer-Function to State-Space Controllable Canonical Form

Consider the transfer-function where the order n = 3.

$$G(s) = \frac{s^2 + 6s + 5}{s^2 + 6s^2 + 11s + 6}$$

Using our pattern, the phase-variable canonical form (also called controllable canonical form) becomes

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(t) \tag{7.38}$$

$$y(t) = [\, 5 \quad 6 \quad 1 \,] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{7.39}$$

There is of course another form of this canonical form where the bottom-row of the **A** matrix moves to the top and so on as before.

7.2.2 Observable Canonical Form

Perhaps not so common as the previous form, this form is more difficult to arrive at. Consider the system

$$G(s) = \frac{2s^2 + 6s + 5}{s^2 + 7s^2 + 14s + 8}$$

and divide numerator and denominator by the highest power of s.

$$\frac{y}{u}(s) = \frac{2s^{-1} + 6s^{-2} + 5s^{-3}}{1 + 7s^{-1} + 14s^{-2} + 8s^{-3}} \tag{7.40}$$

Cross-multiply and we get

$$y(s)\left(1 + 7s^{-1} + 14s^{-2} + 8s^{-3}\right) = \left(2s^{-1} + 6s^{-2} + 5s^{-3}\right)u(s) \tag{7.41}$$

Collect terms of negative powers of *s*

$$y = s^{-1}(2u - 7y) + s^{-2}(6u - 14y) + s^{-3}(5u - 8y) \tag{7.42}$$

This tells us that not surprisingly there are 3 integrators and therefore 3 states. We can draw a signal-flow graph and label the output of the integrators. We define $y = x_3$ (but we could label $y = x_1$ as an alternative). We observe that the output of each integrator must be the integral of its input. Label the right-most integrator output as x_3, the second integrator output as x_2 and the first x_1. Their inputs must be their respective derivatives of the outputs.

Equation (7.42) tells us that the first integrator (from the right) must have $(2u - 7y)$ as its input, the second integrator must have $(6u - 14y)$ and the third on the far left, $(5u - 8y)$ at the input (Fig. 7.9).

From the signal-flow graph we can find that

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -8 \\ 1 & 0 & -14 \\ 0 & 1 & -7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 5 \\ 6 \\ 2 \end{bmatrix} u(t) \tag{7.43}$$

$$y(t) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{7.44}$$

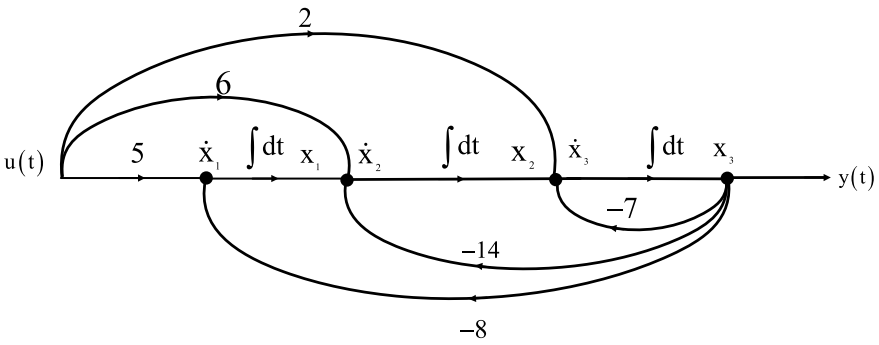


Fig. 7.9 Observable canonical form

For the general case, the observable canonical form becomes

$$A = \begin{bmatrix} 0 & 0 & 0 & . & . & . & 0 & -a_0 \\ 1 & 0 & 0 & . & . & . & 0 & -a_1 \\ . & 1 & . & & & & . & . \\ . & . & . & & & & . & . \\ . & . & & & & & . & . \\ 0 & 0 & 0 & . & . & . & 0 & -a_{n-2} \\ 0 & & & . & . & . & 1 & -a_{n-1} \end{bmatrix}$$

$$B = \begin{bmatrix} b_0 \\ b_0 \\ . \\ . \\ . \\ b_{n-2} \\ b_{n-1} \end{bmatrix}, C = [0 \quad 0 \quad 0 \quad . \quad . \quad . \quad 1]$$

If we remember this pattern, then at least for SISO system we can very quickly write down the state-space description directly.

7.2.3 Biproper Systems

A system which has the number of zeros less than the number of poles is known as *strictly-proper*. A continuous-time system which is physically unrealizable and has more zeros than pole is known as *improper*. For systems that have an equal number

of poles as zeros, we term them *biproper*. The state-space description is a little different. For example, consider the biproper transfer-function

$$G(s) = \frac{s^2 + 6s + 5}{s^2 + 3s + 2} \tag{7.45}$$

Let us put the system in controllable canonical form.

We split the poles and zeros as before defining an intermediate variable x in-between. We then have

$$\begin{aligned} x(s) &= \frac{1}{s^2 + 3s + 2} u(s) \\ y(s) &= \left(s^2 + 6s + 5\right) x(s) \end{aligned}$$

Then inverse Laplace and we get

$$\begin{aligned} \ddot{x} + 3\dot{x} + 2x &= u \\ y &= \ddot{x} + 5\dot{x} + 6x \end{aligned}$$

If define state-variables

$$\begin{aligned} x_1 &= x \\ x_2 &= \dot{x} \end{aligned}$$

Then for the pole part there is no difference and we get

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

For the zero part we have

$$\begin{aligned} y &= \ddot{x} + 6\dot{x} + 5x \\ &= \ddot{x} + 6x_2 + 5x_1 \end{aligned}$$

Now find $\ddot{x}$ from $\ddot{x} + 3\dot{x} + 2x = u$ and we get

$$\ddot{x} = -3x_2 - 2x_1 + u$$

(which we had anyway for fining the pole-part).

$$\begin{aligned} y &= \ddot{x} + 6x_2 + 5x_1 \\ &= -3x_2 - 2x_1 + u + 6x_2 + 5x_1 \end{aligned}$$

Finally

$$y = 3x_2 + 3x_1 + u$$

In vector form

$$y = \begin{bmatrix} 3 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + u$$

In general, we always get a direct path from input to output (the u term) for this kind of system. We write it in general terms

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \tag{7.46}$$

$$y(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}u(t) \tag{7.47}$$

We can solve this a different way by recognising that

$$\frac{s^2 + 6s + 5}{s^2 + 3s + 2} = \frac{as + b}{s^2 + 3s + 2} + c$$

Multiply out

$$s^2 + 6s + 5 = (as + b) + c\left(s^2 + 3s + 2\right)$$

Compare coefficients of

s^2 gives c = 1
s gives 6 = a + 3c or a = 3
s^0 gives 5 = b + 2c or b = 3

Hence

$$\frac{s^2 + 6s + 5}{s^2 + 3s + 2} = \frac{3s + 3}{s^2 + 3s + 2} + 1$$

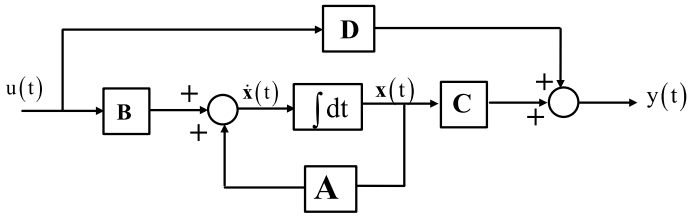


Fig. 7.10 State-space description block-diagram

Now the first term on the RHS is now a strictly-proper system and we can handle it in the usual way. The second term on the RHS is the constant feedthrough term from input to output which we attribute D = 1 in state-space. The general graphical state-space is sometimes shown graphically in Fig. 7.10.

7.3 Converting from State-Space to Transfer-Function

Once we know how to get the state-space representation we need to know how to get back from state-space to transfer-function. This is easily achieved by first writing down the state-space equations

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

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

Now take Laplace-transforms of the first of these with zero initial conditions and we get

$$s\mathbf{x}(s) = \mathbf{A}\mathbf{x}(s) + \mathbf{B}u(s)$$

Collecting terms

$$(s\mathbf{I} - \mathbf{A})\mathbf{x}(s) = \mathbf{B}u(s)$$

The identity matrix ***I*** is required because when subtracting from a matrix the two quantities must be the same dimension. Then by inverting the LHS

$$\mathbf{x}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}u(s)$$

We also note that the inverse must be from the left and not the right since matrices do not necessarily commute (except in special cases). Multiply by $\boldsymbol{C}$ from the left and adding the $\boldsymbol{D}$ term and we get the output

$$\mathbf{y}(\mathrm{s}) = \Big[\mathbf{C}(\mathrm{s}\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}\Big]\mathrm{u}(\mathrm{s})$$

By comparison with the definition for transfer-function, a relationship that is defined as output to input we must have that

$$\mathrm{G}(\mathrm{s}) = \mathbf{C}(\mathrm{s}\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \tag{7.48}$$

We have written G as a scalar here though this depends on the dimension of the input and output. For example if there are two inputs and two outputs, or if there are two inputs and one output) then G becomes either a matrix or vector of transfer-functions and is written in bold $\boldsymbol{G}$. For example for two inputs and outputs we would have a transfer-function matrix of the form

$$\mathbf{G}(\mathrm{s}) = \begin{bmatrix} \mathrm{g}_{11}(\mathrm{s}) & \mathrm{g}_{12}(\mathrm{s}) \\ \mathrm{g}_{21}(\mathrm{s}) & \mathrm{g}_{22}(\mathrm{s}) \end{bmatrix} \tag{7.49}$$

This is termed a square *multivariable* system. For 2 inputs and one output we could have the vector of transfer-functions (a non-square system)

$$\mathbf{G}(\mathrm{s}) = \begin{bmatrix} \mathrm{g}_{11}(\mathrm{s}) & \mathrm{g}_{12}(\mathrm{s}) \end{bmatrix} \tag{7.50}$$

7.3.1 Example Conversion to Transfer-Function

Convert the following state-space transfer description to a transfer-function.

$$\begin{bmatrix} \dot{\mathrm{x}}_1 \\ \dot{\mathrm{x}}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathrm{u}$$

$$\mathrm{y} = \begin{bmatrix} 1 & 3 \end{bmatrix} \begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \end{bmatrix}$$

We have $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\mathbf{C} = \begin{bmatrix} 1 & 3 \end{bmatrix}$ $\mathbf{D} = 0$

Then using (7.48)

$$s\mathbf{I} - \mathbf{A} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} = \begin{bmatrix} s & -1 \\ 2 & s+3 \end{bmatrix}$$

Inverting this matrix by hand, since a 2×2 is easy to solve

$$(s\mathbf{I} - \mathbf{A})^{-1} = \begin{bmatrix} s & -1 \\ 2 & s+3 \end{bmatrix}^{-1} = \frac{1}{\Delta}\begin{bmatrix} s+3 & 1 \\ -2 & s \end{bmatrix}$$

The determinant of the matrix $\Delta = s^2 + 3s + 2$

We then multiply this inverse matrix by $\boldsymbol{C}$ from the left and $\boldsymbol{B}$ from the right. It is easiest to multiply from the right first as there is a zero in the $\boldsymbol{B}$ vector.

$$(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} = \frac{1}{\Delta}\begin{bmatrix} 1 \\ s \end{bmatrix}$$

and

$$\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} = \frac{[1 \quad 3]}{\Delta}\begin{bmatrix} 1 \\ s \end{bmatrix}$$

We arrive at the transfer-function

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

For larger-order problems we can use MATLAB. For this example we can convert as follows.

We see that MATLAB has returned a slightly different realisation than we began with. This is just another version of controllable-canonical form.

7.4 Poles and Zeros from State-Space

If we for the time-being assume we are dealing for the most part with systems for which the $\boldsymbol{D}$ matrix is zero, this simplifies our analysis and examples.

The transfer-function was found to be

MATLAB Code 7.1

```
%MATLAB Code. State-Space to Transfer-function

A=[0 1;2 -3];

% note the dash, indication transpose for B below.Since B is a column vector.

B=[0 1]';

C=[1 3];

D=0

% num and den are numerator and denominator polynomials of G(s)

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

MATLAB returns the numerator and denominator polynomials of the transfer-function

num =

   0    3    1

den =

   1    3    2
```

We can also go from transfer-function to state-spacein MATLAB as follows using the same example. Just one line is needed.

```
%Transfer-function to state-space conversion

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

A =

  -3   -2

   1    0

B =

   1

   0

C =

   3    1

D =

   0
```

$$G(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} \tag{7.51}$$

For future reference, the matrix $(s\mathbf{I} - \mathbf{A})^{-1}$ crops up now and again in the theory and is called the *resolvent* matrix or as a shortcut, the matrix $\Phi(s)$.

The inverse

$$(s\mathbf{I} - \mathbf{A})^{-1} = \frac{\text{adj}(s\mathbf{I} - \mathbf{A})}{|s\mathbf{I} - \mathbf{A}|} \tag{7.52}$$

where *adj* is the *adjoint* matrix and $|s\mathbf{I} - \mathbf{A}|$ is the determinant. Then

$$G(s) = \mathbf{C}\frac{\text{adj}(s\mathbf{I} - \mathbf{A})}{|s\mathbf{I} - \mathbf{A}|}\mathbf{B} \tag{7.53}$$

We see that the poles of the system are given by the roots of the polynomial formed by

$$|s\mathbf{I} - \mathbf{A}| = 0 \tag{7.54}$$

From linear-algebra this is immediately recognisable as the characteristic equation of the matrix. This is formed when we find the eigenvalues of the ***A*** matrix. Therefore *the poles of the transfer-function are the eigenvalues of the A matrix.*

The zeros are a little harder to find and are found by polynomial formed by the determinant of the so-called *system matrix*.

$$\begin{vmatrix} s\mathbf{I} - \mathbf{A} & -\mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{vmatrix} = 0 \tag{7.55}$$

Using the same example, namely $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\mathbf{C} = [1 \quad 3]$, $\mathbf{D} = 0$

$$\begin{vmatrix} \begin{bmatrix} s & -1 \\ 2 & s+3 \end{bmatrix} & -\begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ [1 \quad 3] & 0 \end{vmatrix} = 0 \tag{7.56}$$

Equation (7.56) is written with a little abuse of notation just to illustrate the positioning of the individual sub-matrices. Overall we have a 3×3 matrix and its determinant can be found manually since there is a zero in a row which we can work along. Recall that for finding determinants, we must sign each element in the form $(-1)^{i+j}$ where i and j are its respective row and column position. We then form sub-matrices which we need to find the determinant of. We can work along

any row or column but we select the first row for convenience, crossing out the row and column to form a sub-matrix as per the usual method.

$$\begin{vmatrix} s & -1 & 0 \\ 2 & s+3 & -1 \\ 1 & 3 & 0 \end{vmatrix} = s\begin{vmatrix} s+3 & -1 \\ 3 & 0 \end{vmatrix} + \begin{vmatrix} 2 & -1 \\ 1 & 0 \end{vmatrix} \tag{7.57}$$

$$s\begin{vmatrix} s+3 & -1 \\ 3 & 0 \end{vmatrix} + \begin{vmatrix} 2 & -1 \\ 1 & 0 \end{vmatrix} = 3s+1=0 \tag{7.58}$$

The zero-polynomial is then $3s+1=0$ giving a zero at $s = -1/3$.
MATLAB will calculate the same result from

7.5 States as Physical Variables

The theory up to this point involves state-space descriptions with states that are not real. That is to say they hold no meaning other than a mathematical value and cannot be said to be a measurable quantity in a physical system. However, one of the great things about state-space is that we can indeed assign physical values to the states of a system.

```
z=tzero(A,B,C,D)
```

7.5.1 Example of an RLC Circuit

Consider the RLC circuit shown in Fig. 7.11. There are two energy-storage components, an inductor and capacitor telling us that the system must be second-order and hence two states need to be defined.

Write the equations of the circuit as

$$V_{in} = L\frac{di}{dt} + iR + V_c \tag{7.59}$$

$$i = C\frac{dV_c}{dt} \tag{7.60}$$

Define state-variables as the current through the circuit and the capacitor voltage. The state-vector is now

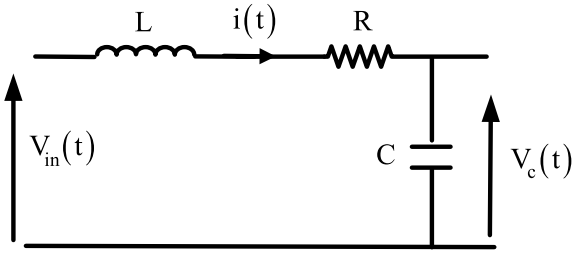


Fig. 7.11 RLC circuit

$$\mathbf{x} = \begin{bmatrix} V_c \\ i \end{bmatrix} \tag{7.61}$$

Differentiating each element in the state, we must find equations in terms of the state-variables. This can be tricky for some circuits but this problem is quite an easy one. From (7.60)

$$\frac{dV_c}{dt} = \frac{i}{C} \tag{7.62}$$

The equation relates the derivative of the first state-variable in terms of the second. We now only need the expression for the derivative of the second state-variable, the current.

From (7.59)

$$\frac{di}{dt} = -i\frac{R}{L} - \frac{V_c}{L} + \frac{V_{in}}{L}$$

This is all we need to make up the state-space equation.

$$\begin{bmatrix} \frac{dV_c}{dt} \\ \frac{di}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 1/C \\ -1/L & -R/L \end{bmatrix} \begin{bmatrix} V_c \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ 1/L \end{bmatrix} V_{in} \tag{7.63}$$

The output equation involving the ***C*** vector all depends on what we define as the output. Suppose we require the capacitor voltage. Then we have

$$y = [1 \quad 0] \begin{bmatrix} V_c \\ i \end{bmatrix} \tag{7.64}$$

If we require both then we can define y as a vector and write

$$\mathbf{y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V_c \\ i \end{bmatrix} \tag{7.65}$$

In the first case ***C*** is a row-vector and in the second ***C*** is a matrix. In addition, we write *y* in bold for the case when it becomes a vector. Equation (7.64) would be the output equation of a single-input two-output system (a so-called none-square system).

We can calculate the transfer function matrix for (7.64) as the output vector by slogging through the algebra. If we use

$$\mathbf{G}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1/C \\ -1/L & -R/L \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 1/L \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

After some algebra we get

$$\mathbf{G}(s) = \begin{bmatrix} \frac{1/LC}{s^2 + \frac{R}{L}s + \frac{1}{LC}} \\ \frac{s/L}{s^2 + \frac{R}{L}s + \frac{1}{LC}} \end{bmatrix} \tag{7.66}$$

7.5.2 Example of a Coupled Mesh Network

The circuit of Fig. 7.12 has two capacitors and one inductor. There are therefore three states. Define as the states as the voltages across both capacitors and the current through the inductor.

It is just a matter of now writing down Kirchoffs current and voltage laws for the circuit and getting the equations into the required form. The state-equation is $\mathbf{x} = \begin{bmatrix} V_{c1} \\ V_{c2} \\ i_L \end{bmatrix}$.

For the loop on the left of Fig. 7.12, sum the voltages. Take v_L to be the voltage across the inductor.

$$u = v_{c1} + v_L \tag{7.67}$$

Now sum the voltages for the second loop

$$v_L = v_{c2} + RC_2 \frac{dv_{c2}}{dt} \tag{7.68}$$

Now re-arrange (7.68) for $\frac{dv_{c2}}{dt}$ and substitute for v_L from (7.67)

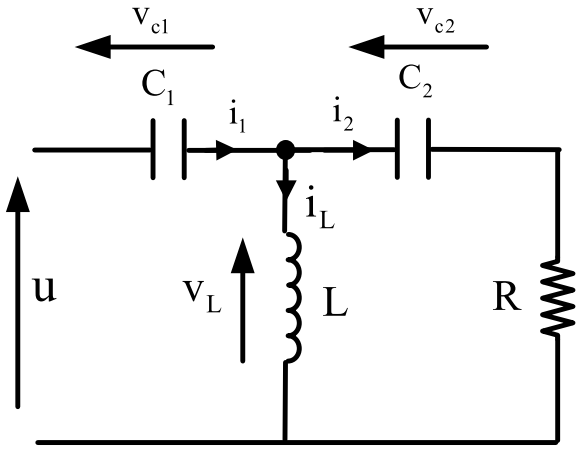


Fig. 7.12 Third-order circuit

$$\frac{dv_{c2}}{dt} = \frac{1}{RC_2}[u - v_{c1} - v_{c2}] \tag{7.69}$$

Now sum the currents at the node.

$$i_1 = i_L + i_2 \tag{7.70}$$

Re-arrange

$$\begin{aligned} i_L &= i_1 - i_2 \\ &= C_1 \frac{dv_{c1}}{dt} - C_2 \frac{dv_{c2}}{dt} \end{aligned} \tag{7.71}$$

Use (7.69) substituted into (7.71)

$$\frac{dv_{c1}}{dt} = \frac{1}{RC_1}[u - v_{c1} - v_{c2}] + \frac{i_L}{C_1} \tag{7.72}$$

From (7.67)

$$u = v_{c1} + L\frac{di_L}{dt}$$

$$\frac{di_L}{dt} = \frac{1}{L}u - \frac{1}{L}v_{c1} \tag{7.73}$$

The state-equations now become

$$\begin{bmatrix} \frac{dv_{c1}}{dt} \\ \frac{dv_{c2}}{dt} \\ \frac{di_L}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{1}{C_1 R} & -\frac{1}{C_1 R} & \frac{1}{C_1} \\ -\frac{1}{C_2 R} & -\frac{1}{C_2 R} & 0 \\ -\frac{1}{L} & 0 & 0 \end{bmatrix} \begin{bmatrix} v_{c1} \\ v_{c2} \\ i_L \end{bmatrix} + \begin{bmatrix} \frac{1}{C_1 R} \\ \frac{1}{C_2 R} \\ \frac{1}{L} \end{bmatrix} u \tag{7.74}$$

The output can be taken from any of the states by selecting the right $\boldsymbol{C}$ vector. If we require the capacitor voltage v_{c1} for instance then

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_{c1} \\ v_{c2} \\ i_L \end{bmatrix} \tag{7.75}$$

7.5.3 Example of Mass-Spring-Damper

Finally, a mechanical problem which is quite classical in that many textbooks consider this. The mass with spring and damper has two energy-storage devices, the mass and spring.

Figure 7.13 illustrates this. The position of the mass is denoted by x and its velocity by $\dot{x}$. The force applied is u. If we select the position and velocity as state-vectors then form its differential equation by summing all forces:

$$M\ddot{x} + B\dot{x} + kx = u \tag{7.76}$$

Now let $x_1 = x, x_2 = x$ and $\dot{x}_1 = x_2$. From (7.76) $\dot{x} = -xB/M - xk/M + u/M$. The state-space description becomes

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/M & -B/M \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/M \end{bmatrix} u \tag{7.77}$$

Take the output as position and

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.78}$$

This is an important example, since in many mechanical systems we take position, velocity, acceleration etc. as the states. The same goes for rotational electro-mechanical systems. The states are usually angular position, angular velocity and angular acceleration.

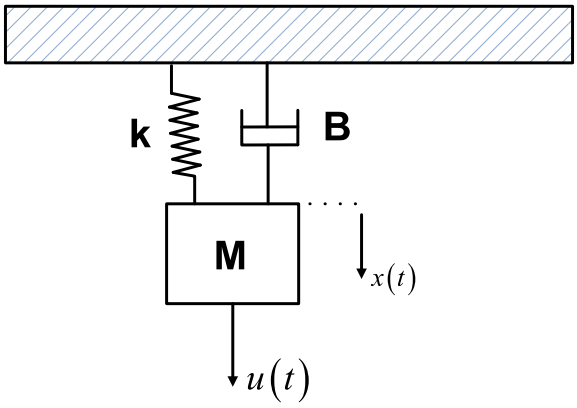


Fig. 7.13 Mass-spring-damper

7.6 Similarity Transformations

Although not essential in any design philosophy, the similarity transformation is a handy tool used in this application to diagonalise a system. We achieve this by applying a mapping from the state-vector **x** to a new state-vector **z** by means of a matrix **T**. We apply

$$\mathbf{x} = \mathbf{T}\mathbf{z} \tag{7.79}$$

Apply this transformation to the state-space equations

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

$$\mathrm{y}(\mathrm{t}) = \mathbf{C}\mathbf{x}(\mathrm{t})$$

We get

$$\mathbf{T}\dot{\mathbf{z}}(\mathrm{t}) = \mathbf{A}\mathbf{T}\mathbf{x}(\mathrm{t}) + \mathbf{B}\mathrm{u}(\mathrm{t}) \tag{7.80}$$

and

$$\mathrm{y}(\mathrm{t}) = \mathbf{C}\mathbf{T}\mathbf{x}(\mathrm{t}) \tag{7.81}$$

Define $\mathbf{\Lambda} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$, and from (7.80) obtain

$$\dot{\mathbf{z}}(\mathrm{t}) = \mathbf{\Lambda}\mathbf{x}(\mathrm{t}) + \mathbf{T}^{-1}\mathbf{B}\mathrm{u}(\mathrm{t}) \tag{7.82}$$

We say that (7.82) and the original state-space description are *similar*. We have transformed the state to a different one but from output to input the transfer-function is still the same. We call ***T*** a basis matrix. We can prove that our new **A** matrix,

which we call instead $\mathbf{\Lambda} = \mathbf{T}^{-1}\mathbf{AT}$ has the same eigenvalues as the original one $\mathbf{A}$. We first need a commonly used linear-algebra result that the determinant of the inverse of a matrix is the same as the determinant of the matrix itself.

$$|\mathbf{T}| = \left|\mathbf{T}^{-1}\right| \tag{7.83}$$

Now take the eigenvalues of $\mathbf{\Lambda} = \mathbf{T}^{-1}\mathbf{AT}$

$$\begin{aligned} &\left|\lambda\mathbf{I} - \mathbf{T}^{-1}\mathbf{AT}\right| \\ &= \left|\mathbf{T}^{-1}\right||\lambda\mathbf{T} - \mathbf{AT}| \\ &= \left|\mathbf{T}^{-1}\right||\lambda\mathbf{I} - \mathbf{A}||\mathbf{T}| \\ &= |\lambda\mathbf{I} - \mathbf{A}| \end{aligned} \tag{7.84}$$

Proving that our transformed matrix $\mathbf{\Lambda}$ has the same eigenvalues as the original matrix $\mathbf{A}$. In the above we have also used the fact that the determinant of the product of two matrices is the same as the product of the determinants $|\mathbf{XY}| = |\mathbf{X}||\mathbf{Y}|$.

This now means we can define any square matrix $\boldsymbol{T}$ provided it is invertible as a co-ordinates change and create a similar system. There are an infinitely of possibilities. Consider just the system $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\mathbf{C} = [1 \quad 0]$, $\mathbf{D} = 0$ which corresponds to the transfer-function $\mathrm{G(s)} = \frac{1}{\mathrm{S}^2 + 3\mathrm{S} + 2}$. Now apply a transformation matrix

$$\mathbf{T} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \tag{7.85}$$

From (7.82) we obtain a similar system

$$\begin{aligned} \begin{bmatrix} \dot{\mathbf{z}}_1 \\ \dot{\mathbf{z}}_2 \end{bmatrix} &= \begin{bmatrix} -17 & -24 \\ 10 & 14 \end{bmatrix}\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}\mathrm{u} \\ \mathrm{y} &= [1 \quad 2]\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} \end{aligned} \tag{7.86a, b}$$

This system is identical from input to output but the state definitions are different. Although of some theoretical interest this does not progress us much however unless we use a special transformation. We can make the transformation matrix the matrix of eigenvalues of the $\boldsymbol{A}$ matrix (or modal matrix). This has the effect of diagonalising the system and this does have certain advantages. Consider the same system. We find the eigenvalues and eigenvectors of $\boldsymbol{A}$. The eigenvalues are found from

$$|\lambda \mathbf{I} - \mathbf{A}| = 0 \tag{7.87}$$

From which we obtain the polynomial

$$\lambda^2 + 3\lambda + 2 = 0 \tag{7.88}$$

The eigenvalues are then the poles of the system or $\lambda_1 = -1, \lambda_2 = -2$. To find the matrix $\boldsymbol{T}$ we must find the eigenvectors. We solve for a vector $\mathbf{v}_1$ from

$$(\mathbf{A} - \lambda_1 \mathbf{I})\mathbf{v}_1 \tag{7.89}$$

and repeat for the second eigenvalue to find vector $\mathbf{v}_2$. We then construct the matrix $\boldsymbol{T}$ from

$$\mathbf{T} = [\mathbf{v}_1 \quad \mathbf{v}_2] \tag{7.90}$$

This is quite an easy task for a second-order system, but we must realise that the eigenvectors are not unique in their scaling and often they are normalised for simplicity. MATLAB can solve such problems for any order with the command

Where $\boldsymbol{T}$ is our matrix of eigenvectors and $\boldsymbol{D}$ is a diagonal matrix of eigenvalues. For our problem MATLAB returns

$$\mathbf{T} = \begin{bmatrix} 0.707 & -0.4472 \\ -0.707 & 0.8944 \end{bmatrix}, \mathbf{D} = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$$

The beauty of using the eigenvector matrix to transform a system is that the new matrix formed, $\mathbf{\Lambda} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$ is diagonal, consisting of the poles of the system (identical to $\boldsymbol{D}$ in MATLAB). This is the case assuming the poles of the system are all unique and not duplicated, in which a similar though more complex form is required known as the *Jordan form*. Continuing, we find $\mathbf{T}^{-1}\mathbf{B} = \begin{bmatrix} 1.4142 \\ 2.2361 \end{bmatrix}$, $\mathbf{CT} = [0.707 \quad -0.4472]$. Our new diagonal system is then

```
[T,D] = eig(A)
```

$$\begin{bmatrix} \dot{\mathbf{z}}_1 \\ \dot{\mathbf{z}}_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} + \begin{bmatrix} 1.4142 \\ 2.2361 \end{bmatrix} \mathrm{u}$$
$$\mathrm{y} = [0.707 \quad -0.4472] \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} \tag{7.91}$$

We will see later that the diagonal form has certain advantages, but for the time being we draw its signal-flow graph in Fig. 7.14.

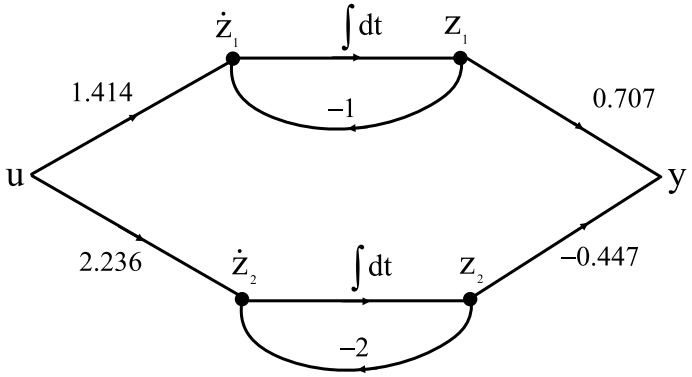


Fig. 7.14 Diagonal form of signal-flow graph

7.7 Step-Response of State-Space Systems

For a simple scalar problem we already showed at the beginning of this chapter that the step-response of systems described in this way requires a convolution integral. We can verify this by using Laplace-Transforms on the state-equations.

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \tag{7.92}$$

$$y(t) = \mathbf{C}\mathbf{x}(t) \tag{7.93}$$

Define $\mathbf{\Phi}(s) = (s\mathbf{I} - \mathbf{A})^{-1}$ and take Laplace transforms of (7.92) and we find

$$\mathbf{x}(s) = \mathbf{\Phi}(s)\mathbf{x}_0 + \mathbf{\Phi}(s)\mathbf{B}u(s) \tag{7.94}$$

Now here is where it gets interesting. If we inverse-Laplace $\mathbf{\Phi}(s) = (s\mathbf{I} - \mathbf{A})^{-1}$ we get a matrix

$$\begin{aligned} \mathbf{\Phi}(t) &= \mathcal{L}^{-1}(s\mathbf{I} - \mathbf{A})^{-1} \\ &= e^{\mathbf{A}t} \end{aligned} \tag{7.95}$$

Exponential a matrix? The matrix $\mathbf{\Phi}(t) = e^{\mathbf{A}t}$ is termed the *state-transition* matrix. It can be found either by direct application of the Laplace transform inverse on every element of the matrix, or approximated by a power-series.

$$e^{\mathbf{A}t} = \mathbf{I} + \mathbf{A}t + \frac{\mathbf{A}^2t^2}{2!} + \frac{\mathbf{A}^3t^3}{3!}.. \tag{7.96}$$

This is a messy way however, even simple examples require that each element of the matrix converges in a power-series and you must recognise the terms. We can inverse Laplace directly instead. For example, for our system above if

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

$$\begin{aligned} \mathbf{\Phi}(\mathrm{s}) &= (\mathrm{s}\mathbf{I} - \mathbf{A})^{-1} \\ &= \begin{bmatrix} \frac{\mathrm{S}+3}{(\mathrm{S}+1)(\mathrm{S}+2)} & \frac{1}{(\mathrm{S}+1)(\mathrm{S}+2)} \\ \frac{-2}{(\mathrm{S}+1)(\mathrm{S}+2)} & \frac{s}{(\mathrm{S}+1)(\mathrm{S}+2)} \end{bmatrix} \end{aligned} \tag{7.97}$$

The state-transition matrix is the inverse-Laplace of this matrix element by element. We expand each term as partial fractions then inverse-Laplace from the tables.

$$\begin{aligned} \mathbf{\Phi}(\mathrm{s}) &= \begin{bmatrix} \frac{\mathrm{S}+3}{(\mathrm{S}+1)(\mathrm{S}+2)} & \frac{1}{(\mathrm{S}+1)(\mathrm{S}+2)} \\ \frac{-2}{(\mathrm{S}+1)(\mathrm{S}+2)} & \frac{s}{(\mathrm{S}+1)(\mathrm{S}+2)} \end{bmatrix} \\ &= \begin{bmatrix} \frac{2}{\mathrm{S}+1} - \frac{1}{\mathrm{S}+2} & \frac{1}{\mathrm{S}+1} - \frac{1}{\mathrm{S}+2} \\ \frac{-2}{\mathrm{S}+1} + \frac{2}{\mathrm{S}+2} & \frac{-1}{\mathrm{S}+1} + \frac{2}{\mathrm{S}+2} \end{bmatrix} \end{aligned}$$

From tables we find

$$\mathbf{\Phi}(\mathrm{t}) = \begin{bmatrix} 2\mathrm{e}^{-\mathrm{t}} - \mathrm{e}^{-2\mathrm{t}} & \mathrm{e}^{-\mathrm{t}} - \mathrm{e}^{-2\mathrm{t}} \\ -2\mathrm{e}^{-\mathrm{t}} + 2\mathrm{e}^{-2\mathrm{t}} & -\mathrm{e}^{-\mathrm{t}} + 2\mathrm{e}^{-2\mathrm{t}} \end{bmatrix} \tag{7.98}$$

Of course for higher-order systems this will get quite complicated. An easier method is to us the modal matrix of eigenvectors.

$$\mathbf{\Phi}(\mathrm{t}) = \mathbf{T}\mathrm{e}^{\mathbf{\Lambda}t}\ \mathbf{T}^{-1} \tag{7.99}$$

This is simply because the matrix $\mathbf{\Lambda} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$ is diagonal and exponential a diagonal matrix can be found as

$$\mathrm{e}^{\mathbf{\Lambda}\mathrm{t}} = \begin{bmatrix} \mathrm{e}^{\lambda_1 \mathrm{t}} & 0 \\ 0 & \mathrm{e}^{\lambda_2 \mathrm{t}} \end{bmatrix} \tag{7.100}$$

For the same example $\mathbf{\Lambda} = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$ and clearly $\mathrm{e}^{\mathbf{\Lambda}\mathrm{t}} = \begin{bmatrix} \mathrm{e}^{-\mathrm{t}} & 0 \\ 0 & \mathrm{e}^{-2\mathrm{t}} \end{bmatrix}$ with $\mathbf{T} = \begin{bmatrix} 0.707 & -0.4472 \\ -0.707 & 0.8944 \end{bmatrix}$.

$$\begin{aligned}\boldsymbol{\Phi}(t) &= \mathbf{T}e^{\boldsymbol{\Lambda}t}\mathbf{T}^{-1} \\ &= \begin{bmatrix} 0.707 & -0.4472 \\ -0.707 & 0.8944 \end{bmatrix} \begin{bmatrix} e^{-t} & 0 \\ 0 & e^{-2t} \end{bmatrix} \begin{bmatrix} 2.8284 & 1.4142 \\ 2.2361 & 2.2361 \end{bmatrix}\end{aligned} \tag{7.101}$$

Which after multiplying out, gives the same result as (7.98).

Continuing our analysis on step-response. From (7.94), taking the inverse Laplace

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0 + \int_{\tau=0}^{\tau=t} \boldsymbol{\Phi}(t-\tau)\mathbf{B}u(\tau)d\tau \tag{7.102}$$

The RHS of (7.102) is of course the convolution integral, but here applied to a matrix quantity. The method is exactly the same as used in scalar quantities, only we convolve element by element. We don't have to evaluate such integrals much nowadays due to the use of modern software tools, yet a basic understanding is good via the following example.

7.7.1 Step-Response of a Mass with Force

Consider the mass M with force u(t) applied on a frictionless surface. Newtons law gives (Fig. 7.15)

$$M\frac{d^2x}{dt^2} = u \tag{7.103}$$

Define state-variables as position $x_1 = x$ and velocity $x_2 = \dot{x}$ and obtain the state-space realisation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/M \end{bmatrix} u \tag{7.104}$$

We wish to obtain the step-response of the states in response to a unit force of u = 1 N. We must solve the state-equations using (7.102). First we need the state-transition matrix. Actually, in this particular case it is one of the few times the power-series method comes in handy. This is because if we raise $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ to the power 2 or higher the result is a zero matrix. Therefore

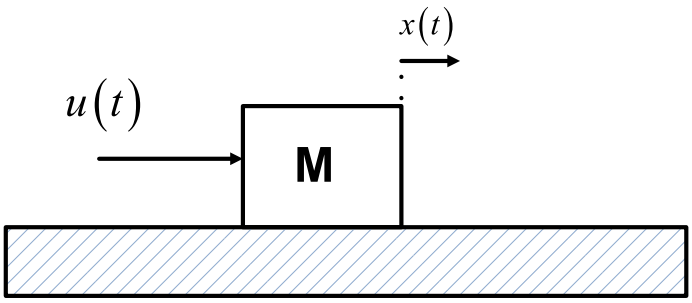


Fig. 7.15 Mass with force

$$\begin{aligned} e^{\mathbf{A}t} &= \mathbf{I} + \mathbf{A}t + \frac{\mathbf{A}^2t^2}{2!} + \frac{\mathbf{A}^3t^3}{3!} .. \\ &= \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{7.105}$$

We can check this by the Laplace method

$$\begin{aligned} \mathbf{\Phi}(t) &= \mathcal{L}^{-1}(s\,\mathbf{I} - \mathbf{A})^{-1} \\ &= \mathcal{L}^{-1} \begin{bmatrix} \frac{1}{s} & \frac{1}{s^2} \\ 0 & \frac{1}{s} \end{bmatrix} \\ &= \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \end{aligned}$$

Assuming initial conditions of position and velocity on the state-vector $\mathbf{x}_0 = [\,x_{op} \quad x_{ov}\,]^T$. We first solve for the zero-input (homogeneous) case.

$$\begin{aligned} \mathbf{x}(t) &= e^{\mathbf{A}t}\mathbf{x}_0 \\ &= \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{op} \\ x_{ov} \end{bmatrix} \end{aligned}$$

The response to initial conditions is therefore

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_{op} + tx_{ov} \\ x_{ov} \end{bmatrix} \tag{7.106}$$

Now we need to solve the force-input part when the input $u(t) = u(\tau) = 1$

$$\begin{aligned}\mathbf{x}(t) &= \int_{\tau=0}^{\tau=t} \mathbf{\Phi}(t-\tau)\mathbf{B}u(\tau)d\tau \\ &= \int_{\tau=0}^{\tau=t} \begin{bmatrix} 1 & t-\tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1/M \end{bmatrix} d\,\tau\end{aligned} \tag{7.107}$$

We must remind ourselves that we are integrating wrt τ and not time t. The t terms become constants in (7.107).

$$\begin{aligned}\mathbf{x}(t) &= \int_{\tau=0}^{\tau=t} \begin{bmatrix} 1 & t-\tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1/M \end{bmatrix} d\tau \\ &= \frac{1}{M} \int_{\tau=0}^{\tau=t} \begin{bmatrix} t-\tau \\ 1 \end{bmatrix} d\tau \\ &= \frac{1}{M} \begin{bmatrix} t\tau - \frac{1}{2}\tau^2 \\ \tau \end{bmatrix}_{\tau=0}^{\tau=t} \\ &= \frac{1}{M} \begin{bmatrix} t^2 - \frac{1}{2}t^2 \\ t \end{bmatrix} \\ &= \frac{1}{M} \begin{bmatrix} \frac{1}{2}t^2 \\ t \end{bmatrix}\end{aligned}$$

Combine this result with the initial condition response and we find

$$\begin{aligned}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} x_{op} + tx_{ov} \\ x_{ov} \end{bmatrix} + \frac{1}{M} \begin{bmatrix} \frac{1}{2}t^2 \\ t \end{bmatrix} \\ &= \begin{bmatrix} x_{op} + tx_{ov} + \frac{at^2}{2} \\ x_{ov} + at \end{bmatrix}\end{aligned}$$

where we have defined $a = 1/M$ as the acceleration of the mass since force is constant at unity. These equations have of course a familiar look about them from basic mechanics.

Chapter 8
State-Space Control

Chapter 7 looks at the basic theory underlining LTI systems as represented in state-space. The state-space approach offers us both advantages and disadvantages. The most prominent advantage is that the method is quite general and can apply to multivariable, time-variant and nonlinear systems. The disadvantage is that there is little or no concept of frequency or bandwidth and so we lose sight of one of the most important aspects of design methodology since the time of Bode. The bandwidth of a compensated systems gives us an idea of the possible maximum capabilities of a control-system. It tells us how much more we can push the performance in easily recognisable units of Hz. Amplifiers for instance are categorised in terms of various parameters such as gain-bandwidth product, and hence if a new op-amp is bought we know the performance by these standards. Any control-system which is based on just adjustments of controller settings (such as PID) leads us to the loss of a major design parameter. Unfortunately state-space also suffers from this as can be seen in the following.

8.1 State-Variable Feedback

Just as with ordinary feedback of the output of a system to create an error, we can do the same thing with the states themselves. To begin with we assume that the states have perfect measurements.

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_8

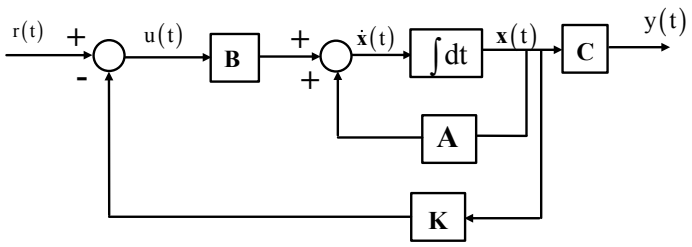


Fig. 8.1 State-feedback regulator

Figure 8.1 illustrates state-feedback in the form

$$\mathrm{u} = \mathrm{r} - \mathbf{K}\mathrm{x} \tag{8.1}$$

where for SISO systems of order n, $\boldsymbol{K}$ is a row-vector of the form.

$$\mathbf{K} = [\mathrm{k}_1 \quad \mathrm{k}_2 \quad \ldots \quad \mathrm{k}_n]$$

The setpoint is shown as $\boldsymbol{r}$. We apply this control-law to the system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathrm{u} \tag{8.2}$$

$$\mathrm{y} = \mathbf{C}\mathbf{x} \tag{8.3}$$

Substitute (8.1) into (8.2)

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}[\mathrm{r} - \mathbf{K}\mathrm{x}] \tag{8.4}$$

Re-arrange

$$\dot{\mathbf{x}} = [\mathbf{A} - \mathbf{B}\mathbf{K}]\mathbf{x} + \mathbf{B}\mathrm{r} \tag{8.5}$$

We term (8.5) the *closed-loop state-space description*. If the eigenvalues of $\mathbf{A}$ determine the poles of the open-loop system then the eigenvalues of $\mathbf{A} - \mathbf{B}\mathbf{K}$ must determine the stability in closed-loop. If all goes well then we can think immediately of a simple way of designing the performance of the closed-loop system. We can select the gains of the state-feedback vector to give desired closed-loop poles (or eigenvalues). We select the eigenvalues of

$$\mathbf{A}_c = \mathbf{A} - \mathbf{B}\mathbf{K} \tag{8.6}$$

To have desired characteristics by correct choice of gains.

8.1.1 Example of State-Feedback

Consider a transfer-function of a system that is open-loop unstable. It has a transfer-function

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

Represent the system in controllable-canonical form.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{C} = [1 \quad 0]$$

Eigenvalues of the open-loop system are at $\lambda = 1, -1$. Let the state-feedback vector be $\mathbf{K} = [k_1 \quad k_2]$ which we are required to find. Place the closed-loop poles (eigenvalues) of the system as a complex-conjugate pair at $\lambda_c = -5 \pm j5$. Recall that for complex poles the damping factor is given by the cosine of the angle at which the poles subtend the negative-real axis as shown in Fig. 8.2. If we make the angle 45° we are guaranteed a damping-factor of $\zeta = 0.707$ which is a highly damped response often considered to be optimal.

The negative real part of the pole is chosen as five times faster than the open-loop stable pole. There is of course no indication as to whether this scenario will work, since any higher-order poles are not modelled. It is a logical place to start however. With two complex poles, the closed-loop characteristic polynomial must be $(s + 5 - j5)(s + 5 + j5) = s^2 + 10s + 50$.

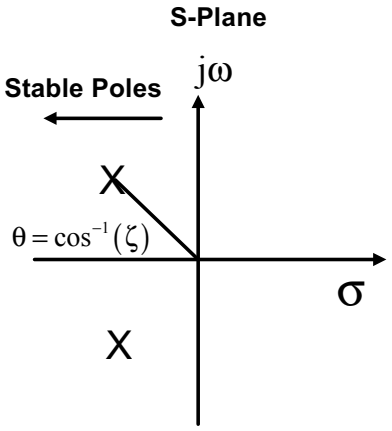


Fig. 8.2 Pole-placement for complex pole-pair

Now form the matrix $\mathbf{A}_c = \mathbf{A} - \mathbf{BK}$.

$$\mathbf{A}_c = \mathbf{A} - \mathbf{BK} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} [\mathrm{k}_1 \quad \mathrm{k}_2]$$
$$= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ \mathrm{k}_1 & \mathrm{k}_2 \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 1 \\ 1 - \mathrm{k}_1 & -\mathrm{k}_2 \end{bmatrix}$$

We now calculate its characteristic equation

$$|\mathrm{s}\mathbf{I} - \mathbf{A}_\mathrm{c}| = \begin{vmatrix} \mathrm{s} & -1 \\ \mathrm{k}_1 - 1 & \mathrm{s} + \mathrm{k}_2 \end{vmatrix}$$
$$= \mathrm{s}^2 + k_2 \mathrm{s} + \mathrm{k}_1 - 1$$

Now compare its coefficients with that of $\mathrm{s}^2 + 10\mathrm{s} + 50$ and we get $\mathrm{k}_2 = 10, \mathrm{k}_1 = 51$. We're now done! Quite a simple procedure which is made even simpler with MATLAB (Fig. 8.3).

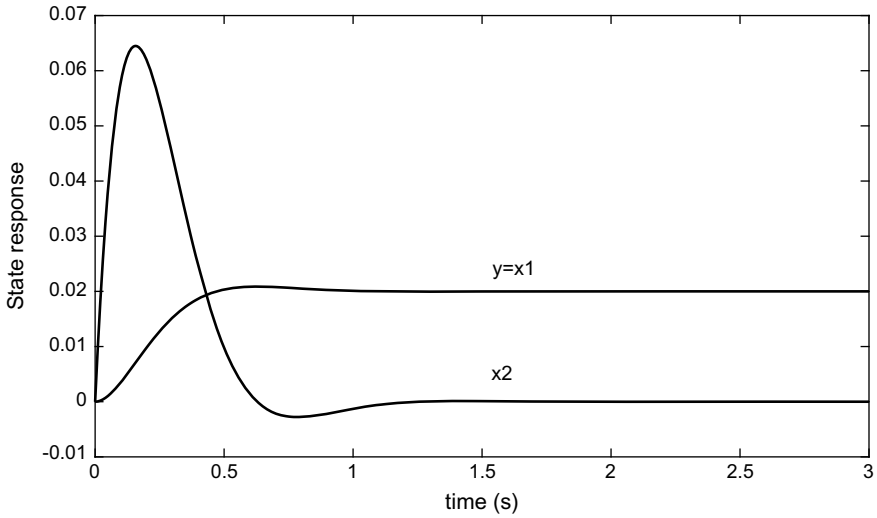


Fig. 8.3 State-variable response to unit-step of setpoint

MATLAB Code 8.1

```
% State-feedback control of open-loop unstable system
% Define system
A=[0 1;1 0]'
B=[0 1]';
C=[1 0];
p1=-5+5i;
p2=-5-5i
% Place the Poles
K=place(A,B,[p1 p2])
%Closed-loop A matrix is Ac
Ac=A-B*K
% Simulate the closed-loop system
% time vector
t=0:0.01:3;
% Step input for the reference, unit magnitude
r=ones(size(t));
% No initial conditions
x0=[0 0]';

gc=ss(Ac,B,C,0)

[y,t,x]=lsim(gc,r,t,x0);
plot(t,x)
xlabel('time(s)')
ylabel('State response')
set(gca,'FontSize',20);
```

The closed-loop system is clearly stable but the output does not rise up to unity which is demanded. Many textbooks recommend scaling the input to compensate for this, but this is not the way to treat steady-state errors. What is wrong is that the state-feedback has provided stability by pole-placement, but it has not got good tracking ability because there are no integrators in the system. However, we can add an integrator as an extra state. See Fig. 8.4.

The extra state gain is shown as a scalar k_I in Fig. 8.4 and the extra state as x_I [23]. The gain is acting on the integral of the error. The error itself becomes

$$e = \dot{x}_I = r(t) - \mathbf{Cx} \tag{8.7}$$

The state-feedback control-law now gets *augmented* by one state x_I

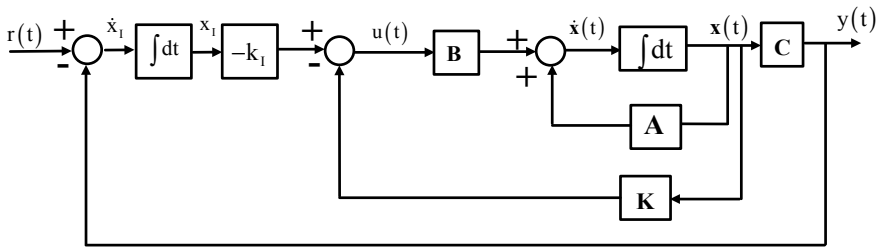


Fig. 8.4 Adding an extra-state and integral action

$$u = -[\mathbf{K} \quad k_I] \begin{bmatrix} \mathbf{x} \\ x_I \end{bmatrix} \tag{8.8}$$

Now use this new augmented control-law in (8.2)

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} - [\mathbf{B}\mathbf{K} \quad \mathbf{B}k_I] \begin{bmatrix} \mathbf{x} \\ x_I \end{bmatrix} \tag{8.9}$$

The augmented closed-loop system is now

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{x}_I \end{bmatrix} = \begin{bmatrix} \mathbf{A} - \mathbf{B}\mathbf{K} & -\mathbf{B}k_I \\ -\mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ x_I \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} r \tag{8.10}$$

$$y = [\mathbf{C} \quad \mathbf{0}] \begin{bmatrix} \mathbf{x} \\ x_I \end{bmatrix} \tag{8.11}$$

Now let us apply this to our example.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{C} = [1 \quad 0]$$

Applying the state + integral feedback

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_I \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 - k_1 & -k_2 & -k_I \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_I \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} r \tag{8.12}$$

Now find the characteristic polynomial of the closed-loop augmented matrix.

$$\begin{vmatrix} s & -1 & 0 \\ k_1 - 1 & s + k_2 & k_I \\ 1 & 0 & s \end{vmatrix} = 0$$

$$s\begin{vmatrix} s + k_2 & k_I \\ 0 & s \end{vmatrix} + \begin{vmatrix} k_1 - 1 & k_I \\ 1 & s \end{vmatrix} = 0$$

$$s^3 + k_2 s^2 + s(k_1 - 1) - k_I = 0$$

Now we must place three poles instead of the original two. If we leave the complex-conjugate pair as before but add a third pole which is faster than the other two (say at $s = -15$) then the pole-polynomial becomes $(s^2 + 10s + 50)(s + 15) = s^3 + 25s^2 + 200s + 750$. Comparing coefficients we get

$$k_2 = 25, k_1 = 201, k_I = -750$$

The closed-loop system with integral-action is now

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_I \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -200 & -25 & 750 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_I \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} r \tag{8.13}$$

and $\mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ $\mathbf{C} = [1 \quad 0 \quad 0]$ are new augmented vectors constructed from (8.10).

The closed-loop system has the desired eigenvalues as per the design.

Simulate the closed-loop system using the MATLAB code below.

MATLAB Code 8.2

```
% State-feedback control of open-loop unstable system
% with extra state - integral action
%Closed-loop A matrix
Ac=[0 1 0;-200 -25 750;-1 0 0];
B=[0 0  1]';
C=[1 0 0];

% Simulate the closed-loop system
% time vector
t=0:0.01:3;
```

```
% Step input for the reference, unit magnitude
r=ones(size(t));
% No initial conditions
x0=[0 0 0]';

gc=ss(Ac,B,C,0)

[y,t,x]=lsim(gc,r,t,x0);
plot(t,x)
xlabel('time(s)')
ylabel('State response')
set(gca,'FontSize',20);
```

We obtain the step-response of the states as shown in Fig. 8.5.

The output rises to the setpoint of unity with zero steady-state error. The second state and integrator states perform differently. The second state must go to zero as it is the derivative of the first which is constant in steady-state. Likewise the integrator state must be finite valued in steady-state in order to counter the offset.

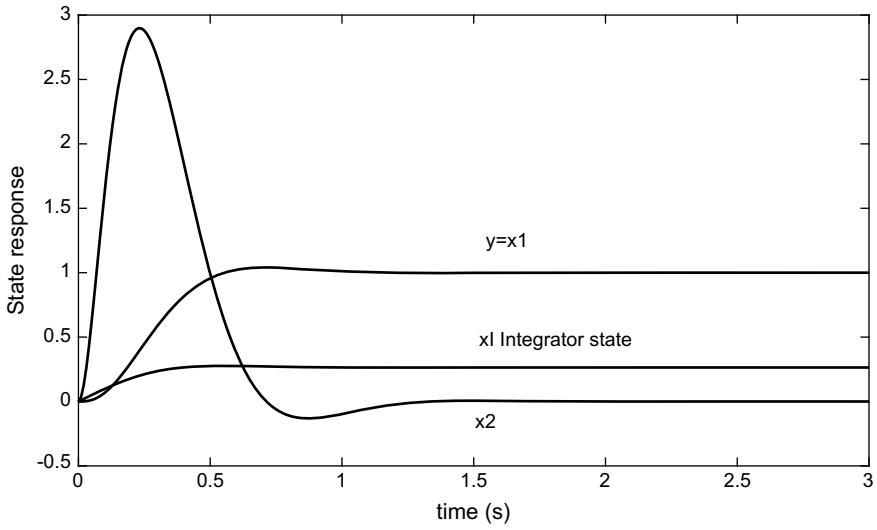


Fig. 8.5 Step-response of states with integral action

8.2 Controllability of a System

What we didn't mention in the previous section is that the state-feedback method may not always work. This occurs in occasions where we say one or more states are *uncontrollable* and therefore we cannot control that state. To illustrate this, consider the following second-order system.

$$G(s) = \frac{s+2}{s^2+3s+2} \tag{8.14}$$

Convert this to state-space. We have many choices, let us choose the one below for a change.

$$\mathbf{A} = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \mathbf{C} = [0 \quad 1]$$

Now we are going to use a similarity transformation $\mathbf{x} = \mathbf{Tz}$ to diagonalise the system. This makes it easier to see what is going on with the states. We need the eigenvalues and eigenvectors of the system to construct the *modal* matrix $\boldsymbol{T}$ of eigenvectors. Use the MATLAB command *[T, D] = eig(A)* to obtain (see earlier work in Chap. 7)

$$\mathbf{T} = \begin{bmatrix} 0.8944 & 0.707 \\ 0.4472 & 0.707 \end{bmatrix}, \Lambda = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$$

Now, the equivalent system is

$$\dot{\mathbf{z}}(t) = \Lambda \mathbf{x}(t) + \mathbf{T}^{-1}\mathbf{B}u(t)$$

$$y(t) = \mathbf{CTx}(t)$$

$$\mathbf{T}^{-1}\mathbf{B} = \begin{bmatrix} 2.236 \\ 0 \end{bmatrix}, \mathbf{CT} = [0.447 \quad 0.707]$$

Sketching a signal-flow graph of the similar system we obtain Fig. 8.6.

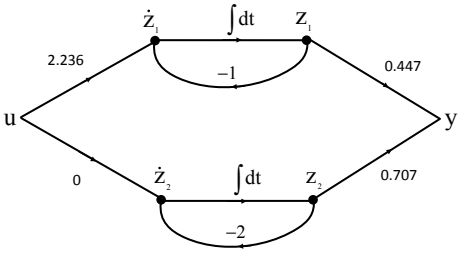


Fig. 8.6 Signal-flow graph of diagonalised (parallel) system

We notice a disturbing fact about Fig. 8.6, the second state has a zero term at the input. We may as well not connect it at all. There is no pathway from the input to the state. We would not notice this on a normal signal-flow graph, only the diagonal one (sometimes called parallel realisation). We say that the second state is uncontrollable. Often we say that the system is not completely controllable. Other than having to diagonalise every system we come across, there is a much better approach using the rank of a matrix.

8.2.1 Rank of a Matrix

The rank of a matrix is a measure of the sparsity of the matrix and represents the maximum number of linearly independent rows or columns. Since the rank of a matrix can involve matrices which are not square, the maximum rank cannot be greater than the smallest number of the rows or columns. For example, a 3×4 matrix would have rank 3 if it were full rank. A 4×2 matrix can only have rank 2 as its maximum rank. For square matrices the rank is easy to calculate. For example

$$\mathbf{M} = \begin{bmatrix} 2 & 3 \\ 4 & 6 \end{bmatrix}$$

We can look at it closely and we see that if we divide the first column by 2 and multiply by 3 we obtain the second column. We could do something similar with the rows. The second row is twice the first row. Hence the vectors that comprise the matrix are not linearly independent and we say that ***M** is not full rank*. There is only one linearly independent row or column so it has rank 1. For square matrices though an easier method is to find the determinant of the matrix. If $|\mathbf{M}| = 0$ then the matrix is rank deficient (another way of saying it is not full rank). This doesn't tell us what the rank is of course, only that it is not full rank if its determinant is zero. For more complicated problems we can use MATLAB. The command is `rank(M)`.

The non-square matrix

$$\mathbf{M} = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 6 & 4 \end{bmatrix}$$

Has rank one because the second row is twice the first.

8.2.2 Rank Test for Controllability

A system of order n is fully controllable if the following controllability matrix has full rank.

$$\mathcal{C} = \left[\mathbf{B}, \mathbf{AB}, \mathbf{A^2B} \ldots \mathbf{A^{n-1}B}\right] \tag{8.15}$$

For $\mathbf{A} = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \mathbf{C} = [0 \quad 1]$ (note the rank test does not require the $\boldsymbol{C}$ vector). There are two states, n = 2

$$\mathbf{AB} = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix}$$

$$\mathcal{C} = [\mathbf{B}, \mathbf{AB}] = \begin{bmatrix} 2 & -2 \\ 1 & -1 \end{bmatrix}$$

This matrix is singular so it has rank one indicating it is not fully controllable. One state is uncontrollable as we have seen by diagonalising the system.

The system $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{C} = [1 \quad 0]$ also has 2 states.

$$\mathbf{AB} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mathcal{C} = [\mathbf{B}, \mathbf{AB}] = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

It is full rank indicating the system we simulated for state-feedback is completely controllable. In fact the trick is to represent a system in controllable canonical form as the name suggests. It will always be controllable (but not necessarily observable which we also may need later). So back to this system

$$\mathrm{G(s)} = \frac{\mathrm{s}+2}{\mathrm{s}^2+3\mathrm{s}+2} \tag{8.16}$$

and the first thing we notice is that we can factorise the denominator

$$\mathrm{G(s)} = \frac{\mathrm{s}+2}{(\mathrm{s}+1)(\mathrm{s}+2)} \tag{8.17}$$

indicating that the system is in fact first-order after all and not second-order. This is the cause of the uncontrollability, one of the states doesn't really exist at all! However, we can write (8.16) in controllable canonical form

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 2 & 1 \end{bmatrix}$$

and do the controllability test

$$\mathcal{C} = [\mathbf{B}, \mathbf{AB}] = \begin{bmatrix} 0 & 1 \\ 1 & -3 \end{bmatrix}$$

We find that at least for this canonical form the system is completely controllable as the matrix is full rank.

8.3 Tuning a State-Space System

If we write a system in controllable canonical form, it will always be controllable and we can apply state-feedback. One thing that is often missed in textbooks we will not study briefly is the equivalence of state-space to PID control. Of course integrators do not appear naturally with state-feedback as we have seen, we have to add them, but this leads us to some interesting observations. For any system in controllable-canonical form we define the states as

$$\begin{aligned} x_1 &= x \\ x_2 &= \dot{x} \\ x_3 &= \ddot{x} \end{aligned}$$

etc. The next state is therefore the derivative of the previous. If we consider a third-order case with transfer-function $G(s) = \frac{b_2 s^2 + b_1 s + b_0}{s^3 + a_2 s^2 + a_1 s + a_0}$, then its controllable-canonical form becomes

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix}$$

Now by taking Laplace-transforms we can show for this canonical form

$$\mathbf{x}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}u(s) = \frac{1}{s^3 + a_2 s^2 + a_1 s + a_0} \begin{bmatrix} 1 \\ s \\ s^2 \end{bmatrix} u(s) \tag{8.18}$$

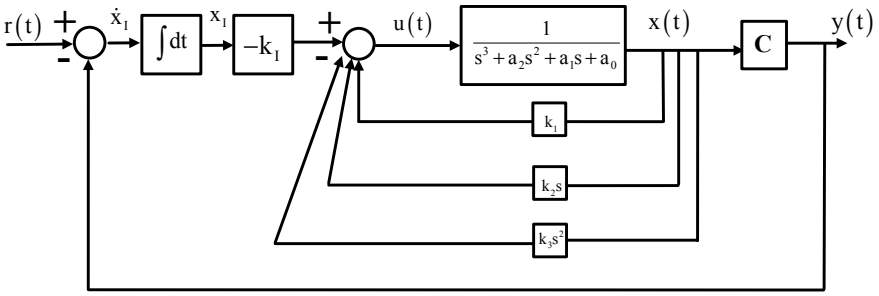


Fig. 8.7 Simplified state-feedback with three states plus an integrator for controllable-canonical form

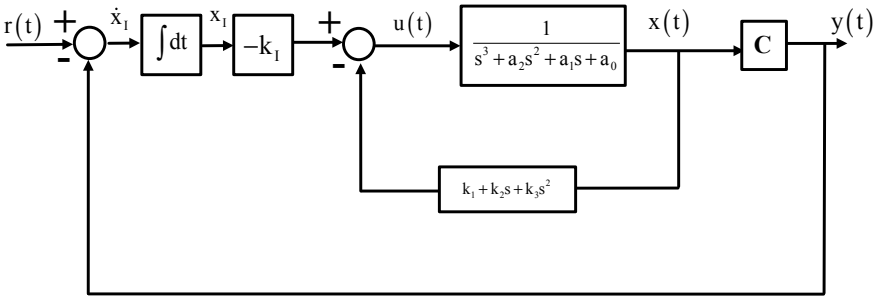


Fig. 8.8 Simplified block-diagram of state-feedback plus integral action for controllable-canonical form

This pattern holds for any order system. When multiplied from the left by the $\boldsymbol{C}$ vector it turns back into the transfer-function. $y(s) = \mathbf{C}x(s) = G(s)u(s)$. So we can see that at the states we have the first state followed by its derivative, second derivative and so on. These in turn get multiplied by the state-feedback gains k_1, k_2, k_3. Drawing a block-diagram of this and including an extra integrator state as in Fig. 8.7.

The differentiators in the feedback path an be simplified to one loop since they both start and finish at the same point. This is shown in Fig. 8.8.

For a second-order system we see that there is a zero in the feedback path from $k_1 + k_2 s$. In fact this is no more than a classical method known as rate-feedback (or derivative-feedback) where a sensor to pick up the derivative of the output is fed back to the input and added to the output itself. It has much the same effect as a phase-advance or derivative term in a PID controller and provides damping. We can think of this in terms of a mechanical system with position as the first state. We would feedback position and velocity in an inner loop. In position-control of dc motors the derivative term is often called Tacho-feedback as a Tacho which measures speed is used as a sensor. In autopilot systems a rate-gyro was used for the

same purpose. For the third-order system this theory is clearly extended so we also feedback acceleration as well as position and velocity. Just as with PID we can tune the gains manually for such systems starting with the position gain until it becomes unstable and then adding derivative. Finally, we add the integrator gain. This is just a form of PID under a different name but with a second derivative to control acceleration.

8.4 Observers

The state-feedback approach is both simple and intuitive but one thing is missing. We make the bold assumption that the states can be measured directly and fed back. Whilst there may well be occasions where a sensor is employed for each state, it is more likely in many cases that the states cannot be measured and are contaminated with additive noise. When there is additive noise we term the system a *stochastic* system and without noise a *deterministic* system. We consider in this chapter the deterministic case only. If we cannot directly measure the states then we can do the next best thing, reconstruct them from a mathematical model of the system. Before we do that however, we must first make sure that we can access the states from observations on the output. This is known as *observability*.

8.4.1 Observability

The states of an nth order system are said to be observable if the observability matrix

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA^2} \\ . \\ . \\ \mathbf{CA^{n-1}} \end{bmatrix}$$

has full rank n. For example, consider the system

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 2 & 1 \end{bmatrix}$$

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ -2 & -1 \end{bmatrix}$$

This matrix has rank 1 and therefore not full rank. (Its determinant is zero and/or the columns are not independent. Multiply column 2 by 2 and we get column 1. Multiply row 1 by −1 and we get row 2.) As with the controllable canonical form, if we put our states in *observable* canonical form they will always be observable (but not necessarily controllable).

For the same system but in *observable* canonical form $\mathbf{A} = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \mathbf{C} = [0 \quad 1]$.

We form the observability matrix

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -3 \end{bmatrix}$$

This is full rank and therefore both states are fully observable. We cannot of course use one canonical form for state-feedback and another one for an observer just to ensure controllability and observability. The two states descriptions are different and if we did this we would need to convert the estimated states from the observer to the other canonical form with a special similarity transformation.

8.4.2 Theory of the Observer

Consider a system of order n

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathrm{u} \tag{8.19}$$

$$\mathrm{y} = \mathbf{C}\mathbf{x} \tag{8.20}$$

and a second dynamic system

$$\dot{\hat{\mathbf{x}}} = \hat{\mathbf{A}}\hat{\mathbf{x}} + \hat{\mathbf{B}}\mathrm{u} + \mathbf{L}\mathrm{y} \tag{8.21}$$

where the symbol on top of the state denoted $\hat{\mathbf{x}}$ is termed x-hat and represents an estimate.

$\mathbf{L}$ is a *column* vector of gains of order n. $\mathbf{L} = [\ell_1 \quad \ell_2 \quad . \quad . \quad \ell_\mathrm{n}]^\mathrm{T}$. Define an error vector between the true state and the estimated state thus

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}} \tag{8.22}$$

Luenberger's method is as follows. Substitute the states into the differentiated error.

$$\dot{\mathbf{e}} = \dot{\mathbf{x}} - \dot{\hat{\mathbf{x}}} \tag{8.23}$$

$$\dot{\mathbf{e}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathrm{u} - \hat{\mathbf{A}}\hat{\mathbf{x}} - \hat{\mathbf{B}}\mathrm{u} - \mathbf{L}\mathrm{y}$$

Using (8.20)

$$\dot{\mathbf{e}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathrm{u} - \hat{\mathbf{A}}\hat{\mathbf{x}} - \hat{\mathbf{B}}\mathrm{u} - \mathbf{L}\mathbf{C}\mathbf{x} \tag{8.24}$$

Now from (8.22) $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$ so that the state-estimate

$$\hat{\mathbf{x}} = \mathbf{x} - \mathbf{e} \tag{8.25}$$

Substitute the state-estimate (8.25) in terms of the true state and error into (8.24)

$$\begin{aligned}\dot{\mathbf{e}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathrm{u} - \hat{\mathbf{A}}(\mathbf{x} - \mathbf{e}) - \hat{\mathbf{B}}\mathrm{u} - \mathbf{L}\mathbf{C}\mathbf{x} \\ &= \hat{\mathbf{A}}\mathbf{e} + \left(\mathbf{A} - \hat{\mathbf{A}} - \mathbf{L}\mathbf{C}\right)\mathbf{x} + \left(\mathbf{B} - \hat{\mathbf{B}}\right)\mathrm{u}\end{aligned}$$

Now in the above if $\hat{\mathbf{A}} \rightarrow \mathbf{A} - \mathbf{L}\mathbf{C}$ and $\hat{\mathbf{B}} \rightarrow \mathbf{B}$ we get

$$\dot{\mathbf{e}} = \hat{\mathbf{A}}\mathbf{e} \tag{8.26}$$

Equation (8.26) must die out to zero as the solution provided the eigenvalues of the matrix

$$\hat{\mathbf{A}} = \mathbf{A} - \mathbf{L}\mathbf{C} \tag{8.27}$$

all have negative real parts.

Substitute (8.27) back into (8.21) and we obtain

$$\dot{\hat{\mathbf{x}}} = (\mathbf{A} - \mathbf{L}\mathbf{C})\hat{\mathbf{x}} + \hat{\mathbf{B}}\mathrm{u} + \mathbf{L}\mathrm{y} \tag{8.28}$$

Giving

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathrm{u} + \mathbf{L}(\mathrm{y} - \mathbf{C}\hat{\mathbf{x}}) \tag{8.29}$$

The observer has two inputs, the system output and input. Its output is the state-estimate which is used as a replacement for the real states. This is illustrated in Fig. 8.9.

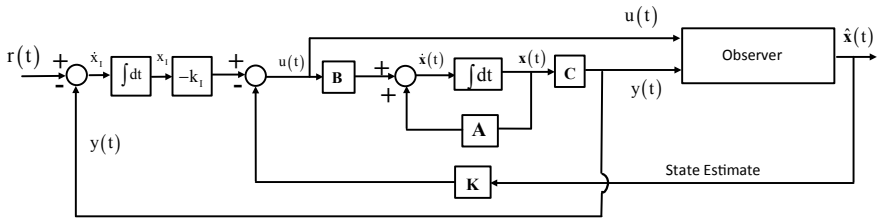


Fig. 8.9 State-feedback using an observer to estimate the states

We see that the integrator state is known, so we need not estimate it via the observer. The observer sits in parallel to the real system measuring the input and output signals and generates the states from the model. It is entirely separate from the state-feedback gain in terms of the calculations. Both require separate equations to get their respective gain vectors. The state-feedback gain vector $\boldsymbol{K}$ is a row vector and the observer gain vector $\boldsymbol{L}$ is a column vector. The calculation of the observer gain vector is done via (8.27) by placing its poles (eigenvalues) to the far left in the s-plane of the closed-loop poles of the system. We require the observer to converge quickly to the true states otherwise this will in turn degrade the performance of the closed-loop system. We pick a simple example to illustrate the operation of the observer. Consider the system with state-space matrices

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{C} = [3 \quad 2]$$

We first check for observability of the states since this system is in controllable canonical form.

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ -12 & -7 \end{bmatrix}$$

This matrix has a determinant which is non-zero and so it is non-singular and full rank. Both states are therefore completely observable.

The eigenvalues of the $\boldsymbol{A}$ matrix are at -1 and -5. Therefore our observer needs to be even faster than this. Place the poles of the observer at s = -10, -20. This gives us a polynomial $s^2 + 30s + 200$. With 2 states the observer gain vector must have two gain terms $\mathbf{L} = \begin{bmatrix} \ell_1 \\ \ell_2 \end{bmatrix}$ and we form (8.27)

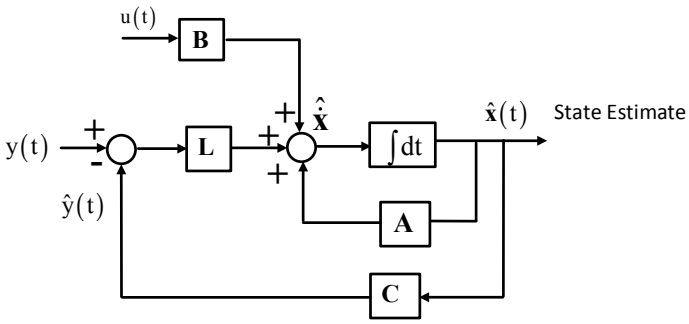


Fig. 8.10 Block diagram of observer

$$\hat{\mathbf{A}} = \mathbf{A} - \mathbf{LC} = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix} - \begin{bmatrix} 3\ell_1 & 2\ell_1 \\ 3\ell_2 & 2\ell_2 \end{bmatrix}$$
$$= \begin{bmatrix} -3\ell_1 & 1-2\ell_1 \\ -6-3\ell_2 & -5-2\ell_2 \end{bmatrix}$$

Now find the characteristic equation of this matrix by forming

$$\left| \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} -3\ell_1 & 1-2\ell_1 \\ -6-3\ell_2 & -5-2\ell_2 \end{bmatrix} \right| = 0$$

or

$$\left| \begin{bmatrix} s+3\ell_1 & -1+2\ell_1 \\ 3\ell_2+6 & s+5+2\ell_2 \end{bmatrix} \right| = 0$$

The polynomial of the observer is then found to be

$$s^2 + (3\ell_1 + 2\ell_2 + 5)s + 3\ell_2 + 6 + 3\ell_1$$

Compare coefficients of s with $s^2 + 30s + 200$ and we obtain

$$\mathbf{L} = \begin{bmatrix} -104.33 \\ 69 \end{bmatrix}$$

The block-diagram of such an observer is shown in Fig. 8.10. We assume that all the matrices are known.

To simulate the observer in MATLAB is a little tricky. In fact *Simulink* would be a better approach. With a little ingenuity however we can still use MATLAB. We do this by augmenting the original system to include the observer.

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\hat{\mathbf{x}}} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{LC} & \mathbf{A}-\mathbf{LC} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \hat{\mathbf{x}} \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ \mathbf{B} \end{bmatrix} \mathrm{u} \tag{8.30}$$

$$\begin{bmatrix} \mathrm{y} \\ \hat{\mathrm{y}} \end{bmatrix} = \begin{bmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \hat{\mathbf{x}} \end{bmatrix} \tag{8.31}$$

We can verify that this system is correct by calculating the eigenvalues of the augmented matrix which gives us the expression

$$\begin{vmatrix} s\mathbf{I}-\mathbf{A} & \mathbf{0} \\ \mathbf{LC} & s\mathbf{I}-\mathbf{A}+\mathbf{LC} \end{vmatrix} = |s\mathbf{I}-\mathbf{A}||s\mathbf{I}-\mathbf{A}+\mathbf{LC}| = 0 \tag{8.32}$$

This is the product of two polynomials, the system itself and the closed-loop observer.

For our second-order example, the first two states will be the true states and the last two the estimates. We require a state-vector or dimension 4. We have two outputs, the true output of the system and the reconstructed estimate from the observer. The MATLAB code is as follows.

MATLAB Code 8.3

```
%State observer simulation
%System
A=[0 1; -6 -5];
B=[0 1]';
C=[3 2];
%Place Poles of observer
po=[-10 -20];
%Find the gain vector
L=place(A',C',po)';
eig(A-L*C)
Ao=A-L*C;
% Observer gain vector
```

```
L
% Simulate the  observer
% time vector
t=0:0.01:3;
% step input for the reference, unit magnitude
r=ones(size(t));
% no initial conditions - 4 states. Actual + estimated
% This is the augmented system of system + observer
I2=eye(2);% identity matrix order 2
% there are 4 states. Two true ones and two estimates
x0=[0 0 0 0]';
Aa=[A 0*I2;L*C Ao]
Ba=[B; B];
Ca=[C 0 0 ;0 0  C]
gc=ss(Aa,Ba,Ca,0)

[y,t,x]=lsim(gc,r,t,x0);
figure
plot(t,x(:,1),t,x(:,2))
xlabel('time(s)')
ylabel('True State response')
set(gca,'FontSize',20);
figure
plot(t,x(:,3),t,x(:,4))
xlabel('time(s)')
ylabel('Estimated State response')
set(gca,'FontSize',20);
```

We find that the two states and the estimates are so close that it is impossible to discern the difference as can be seen in Figs. 8.11 and 8.12.

The state which goes to zero is the second state since it is the derivative of the first. This looks great until we consider what will happen if we don't know the system as well as we think we do. Let us suppose the true system is the same as this current example but the observer uses matrices which are slightly in error, which we denote with subscript m.

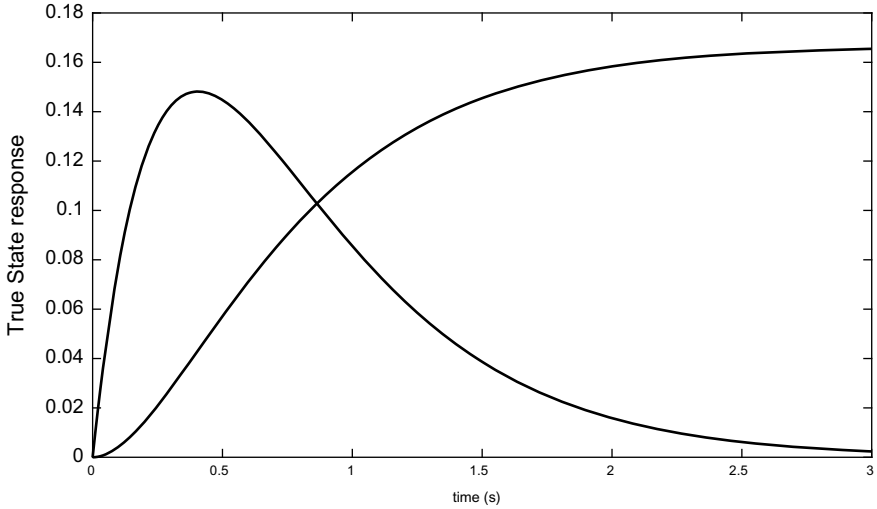


Fig. 8.11 True state response to a step-response of magnitude unity

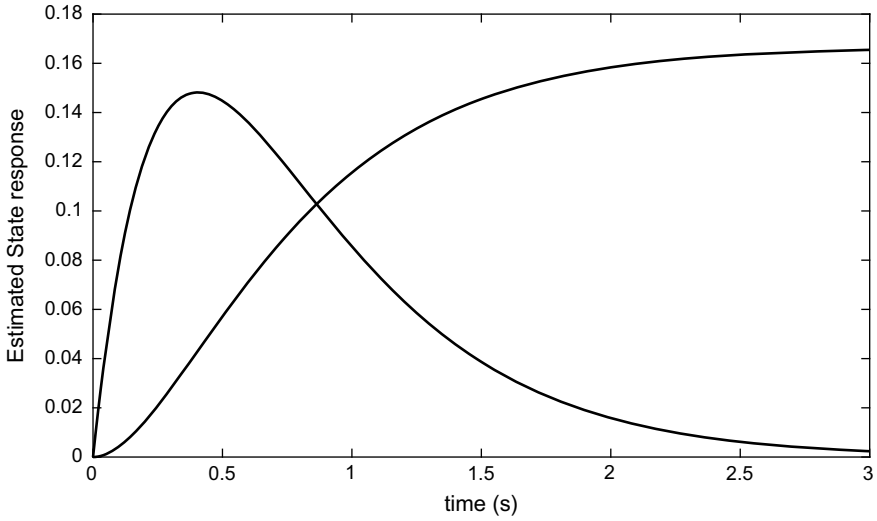


Fig. 8.12 Estimated state response obtained from observer

$$\mathbf{A}_{\mathrm{m}} = \begin{bmatrix} 0 & 1 \\ -6.01 & -5.01 \end{bmatrix}, \mathbf{B}_{\mathrm{m}} = \begin{bmatrix} 0 \\ 1.01 \end{bmatrix} \mathbf{C}_{\mathrm{m}} = [3.01 \quad 2.01]$$

Our augmented system becomes

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\hat{\mathbf{x}}} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{LC} & \mathbf{A}_{\mathrm{m}} - \mathbf{LC}_{\mathrm{m}} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \hat{\mathbf{x}} \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_{\mathrm{m}} \end{bmatrix} \mathbf{u} \tag{8.33}$$

$$\begin{bmatrix} \mathrm{y} \\ \hat{\mathrm{y}} \end{bmatrix} = \begin{bmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{\mathrm{m}} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \hat{\mathbf{x}} \end{bmatrix} \tag{8.34}$$

The poles of the system are the eigenvalues of the augmented matrix and MATLAB returns s = −21.00953, −2, −3 and −9.5614. These are not too far away from the ideal and will not make a difference to the speed that the observer converges in steady-state. However, if we examine the estimates of the two states we see that there is an error between the two. This rapidly increases if the mismatch between system and model becomes larger (Fig. 8.13).

Therefore we must be careful when implementing this kind of design methodology since it requires an accurate model of the system dynamics.

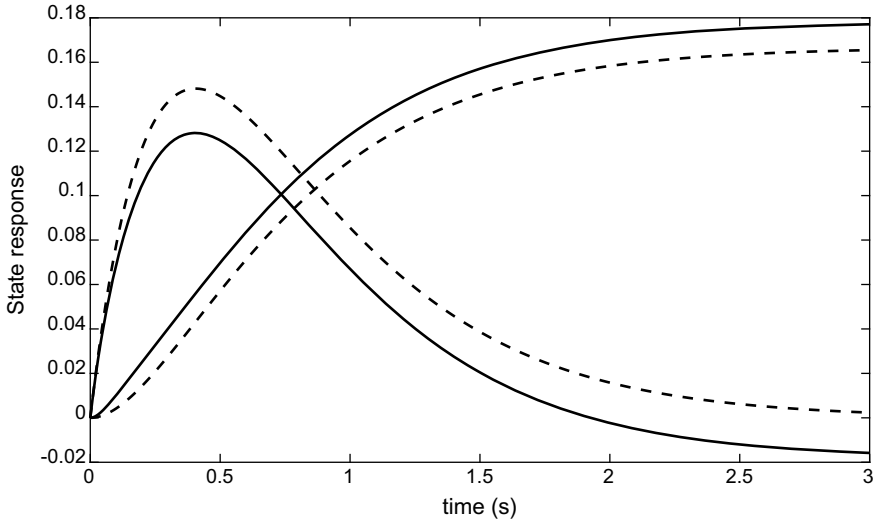


Fig. 8.13 Mismatch between true and estimated states (broken-lines represent the observer estimates)

8.5 The Separation Principle

We have derived the principle for state-feedback and a method for estimating the states. Now we must combine the two approaches, and when we do we notice a quite fascinating result. To proceed further we write down the equations of the open-loop system.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathrm{u} \tag{8.35}$$

and state-feedback *from the observer* (note the states are estimates)

$$\mathrm{u} = -\mathbf{K}\hat{\mathbf{x}} \tag{8.36}$$

Combining

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}\hat{\mathbf{x}} \tag{8.37}$$

Now define the error in the observer as

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}} \tag{8.38}$$

We write (8.37) as

$$\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} + \mathbf{B}\mathbf{K}\mathbf{e} \tag{8.39}$$

Recall that for the observer

$$\dot{\mathbf{e}} = (\mathbf{A} - \mathbf{L}\mathbf{C})\mathbf{e} \tag{8.40}$$

If we write an augmented system by stacking the error below the state then we have from (8.39), (8.40)

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{e}} \end{bmatrix} = \begin{bmatrix} \mathbf{A} - \mathbf{B}\mathbf{K} & \mathbf{B}\mathbf{K} \\ \mathbf{0} & \mathbf{A} - \mathbf{L}\mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{e} \end{bmatrix} \tag{8.41}$$

This describes the dynamics of the closed-loop system, state-feedback and observer combined.

We can find the poles by finding the eigenvalues according to

$$\begin{vmatrix} s\mathbf{I} - (\mathbf{A} - \mathbf{B}\mathbf{K}) & \mathbf{B}\mathbf{K} \\ \mathbf{0} & s\mathbf{I} - (\mathbf{A} - \mathbf{L}\mathbf{C}) \end{vmatrix} = 0 \tag{8.42}$$

From which

$$|s\mathbf{I} - (\mathbf{A} - \mathbf{BK})||s\mathbf{I} - (\mathbf{A} - \mathbf{LC})| \tag{8.43}$$

This is the product of the two characteristic polynomials of the controller and the observer. They are not in any way linked to one-another. The closed-loop poles of the system are composed of both of them independently. Therefore we say that the problem of control can be treated as if the states were the true values and design the controller gain matrix (or vector) separately from that of the observer. We then design the observer independently of the controller gains. Hence the term *separation principle*. The two problems of filtering (essentially an observer is a filter for the states) and control are duals of one-another, Both have very similar equations for selecting the gains.

This is not the end of the line for control-theory by any means, but we have covered a great deal of the most fundamental necessary material. There are a great many other things we could study including for example *optimal-control* and *optimal filtering*. These are covered in later chapters. The basic idea of the separation principle holds for optimal systems too, but instead of placing the poles of the system or observer it is done automatically as the solution to a performance index or cost-function.

Chapter 9
Digital Sampled Systems

In the 21st century, it is a fair assumption that there has to be great advantages in moving from analogue to digital. The change has been gradual, though the hardware took much longer to catch up with the theory, which has been known from around the 1950s. Digital-control has many names which are interchangeable. These include Sampled-Data Systems, a name coined by Jury, one of the fathers of the digital theory [24]. Then we have Discrete-time control-systems (in analogy to continuous-time). This name is more mathematical but more often used among academic engineers. We talk of the continuous-time and discrete-time cases to distinguish the two theories. Finally, there are more colloquial names such as *Computer-controlled systems* or *Digital Control*. It is not a giant step to move from continuous to discrete-time systems. The laws of physics do not change just because we have a computer in the loop! What does change is the versatility and power-consumption of modern control-systems. Just as with linear power-supplies which are very rare indeed nowadays for home appliances, (having been replaced with switching regulators) linear amplifiers have been replaced in control-systems with pulse-width modulated versions. The nearest we could compare with in audio-amplifier terms is class-D replacing class A, B amplifiers. At the time of writing, these switching methods are highly efficient and outperform the linear methods for most of the range of frequencies of interest. The humble dc-motor is all but nearly obsolete in many areas of industry that require speed-control, being replaced by ac-induction or brushless dc motors. For smaller position servo applications however the dc motor has survived for now. Consider a comparison of analogue and digital in terms of hardware.

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_9

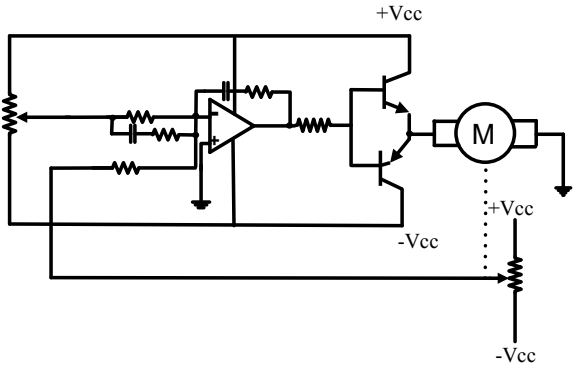


Fig. 9.1 Analogue position servo

9.1 Analogue Versus Digital Hardware

Consider the analogue position servo shown in Fig. 9.1.

This is maybe quite a crude design in many aspects but illustrates the original analogue hardware. We need potentiometers for the setpoint and for the sensor to measure angle of the shaft. These suffer from friction (though there are expensive so-called servo pots in existence with low friction) and a limited turn angle as they hit an end-stop. We would be lucky if they can control 180° and many servos off-the shelf only turn 120°. Then we have a summing and compensation amplifier. This is not too bad but when designing the controller we need to change components on the op-amp to change poles and zeros to get the right performance. In many textbooks, these systems are often shown with a dual supply which gives positive to negative voltage, but this is more a convenience and not a major hurdle as any power-supply can be split to give a pseudo negative terminal. The part that suffers most from increased temperature is the linear power-amplifier. This is due to standing quiescent currents in the Bipolar transistors. In small hobby servos the potentiometer survives but the amplifier is replaced by an H-bridge and MOSFET switching.

Now consider a digital-control servo for a dc-motor.

The potentiometers of the analogue system have been replaced by quad-detectors. These give out a string of pulses. By counting the pulses in a given time-interval we know the velocity. Just be counting (adding or subtracting pulses) we can find the position of the shaft. The resolution depends on the number of pulses they give out in a revolution. Some have as many as 1024 ppr (pulses per revolution). For example, for a full 360° turn with 1024 ppr we have 0.35 of a degree per pulse. Good for many applications. The direction of travel however is ambiguous with only one set of pulses, and so a second of pulses set in quadrature with the first, is also sent out. The two outputs are shown for the quad-detector in Fig. 9.2 (A and B). By detecting the phase between the two we can deduce the direction of travel of the shaft. The encoder does not have any restriction on travel, so it can go right past 360° and on and on for as many turns as required. We can have two identical encoders, one for

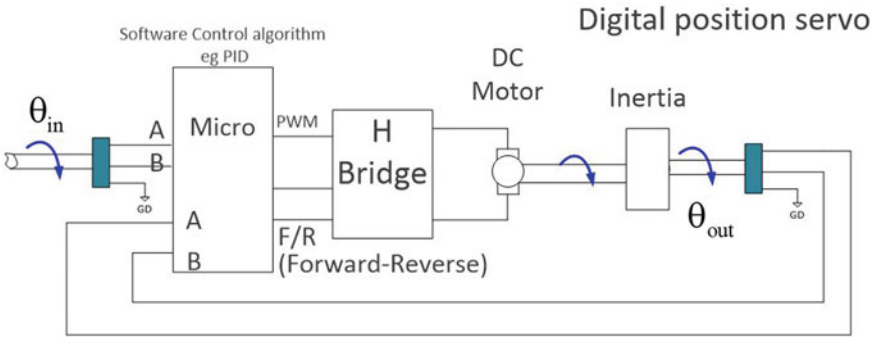


Fig. 9.2 Digital control position servo

setpoint and one for the shaft angle sensor. The only problem is that when the system is first switched on we do not know where the shaft is as there is no absolute reference. In the analogue case we know that mid-way travel on a potentiometer is say 0v and +v max positive, −v negative. Therefore, in the digital case it may become necessary to do a quick setup to position the initial alignment point. If this was part of a robotic arm, we would need to start the arm in a special rest position and take the reference from that position. Indeed, without this setup the servo will still work, but it will start at an arbitrary angle.

The H-bridge is available off-the-shelf in various ratings up to around 100A. Usually the H bridges have inputs which are PWM (the pulse-width modulation input) and two others. Essentially the H-bridge just amplifies the PWM which is produced by the computer (since it cannot drive a motor directly due to too low a current). To reverse the motor we need two binary lines also coming from the computer. Usually they are set so that say 10 in binary represents forward and 01 reverse (00 can be stop). The programmer must therefore enable the correct digital outputs when the output control signal needs to go in the opposite direction. This is usually achieved with a simple *if then else* type instruction.

There is no visible controller in a digital servo so where has it gone? The controller has been replaced by software. Our PID, lag-lead, state-space or whatever type of controller we use in the continuous-time world is replaced by software. This of course is the essence of digital control-system design.

9.2 Sampled-Data

In the early text-books on digital control, it was always assumed that the speed or position of a servo was analogue and we would simply sample this signal via an analogue-to digital convertor (A-D) to convert it to digital. Once in digital form we would perform our software magic to give our digital control (microprocessor) and sent it back put a digital-to-analogue (D-A) convertor. See Fig. 9.3.

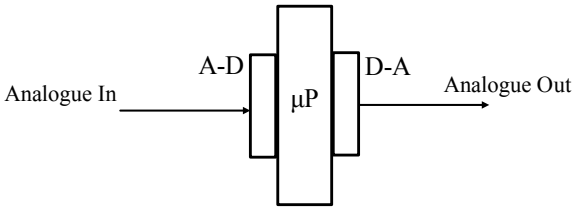


Fig. 9.3 A/D and D/A sampling

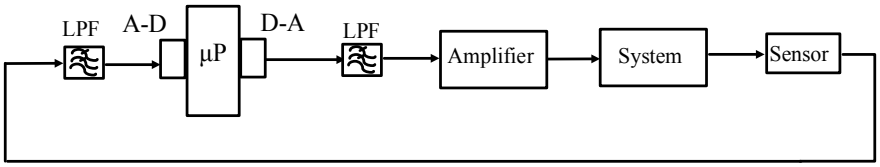


Fig. 9.4 Block-diagram of a generic digital-control system

When we look at Fig. 9.2 however, we do not see an A-D or D-A convertor in sight! This is because the sampling is done by the sensor itself, so there is still a sampling process but it is not explicitly in the form of an A-D convertor. Many sensors we use give out direct digital signals in terms of say an I2C output and the A-D part is transparent to the design process. This is not to say that there is no sampling happening of course, only we do not have to handle it directly. At the other end, where we go to the real-world we use a D-A, but once again in the case of the digital servo there is no D-A in sight. This is replaced by PWM from the computer, which gets filtered by the low-pass characteristic of the motor itself and converted back in this way to dc. However, in the more general case when we have analogue only sensors and do not use PWM directly we can draw a block-diagram of the system as shown in Fig. 9.4.

The purpose of the low-pass filters before the A-D and after the D-A will be explained later. In most microprocessors used in such application we already have built-in A-D and D-A so we do not need external devices to perform these actions. Similarly, PWM is easily generated from any microprocessor.

The microprocessor will need either a *real-time operating system* (RTOS) or exact timing as with *field-programmable gate-arrays* (FPGAs). We cannot use a general-purpose computer such as a personal computer (PC). This is because our processor must be *deterministic* in nature. It must be 100% dedicated to performing the task. A PC may stop half way through and decide to clean up some files or do some general housekeeping tasks! There is no restriction on the language we use as long as it operates in real-time. Usually this means we have an infinite *while()* type loop which runs at start-up and never stops. Of course, it may well interrupt its work to grab data from an I2C bus or sensor via an interrupt, but this then becomes the sampling process itself. What we do not want is a loop that runs at different

speeds, it must run at the same rate every iteration of the loop. We call this rate the sampling-time (or *sampling interval*) and denote it as T_s seconds. This of course gives rise to a second term called the sampling frequency f_s Hz or ω_s rad/s.

9.3 Sampling Theory

Let a signal $f(t)$ be band-limited to ω_m rad/s and sampled every T_s seconds. The action of sampling is modelled by a train of impulses at the required repetition rate multiplied by the signal. This is the generally accepted method of describing the sampling process and works well in practice. We usually assume the impulses have magnitude unity and call them unit-impulses. In practice, the sampling process takes a finite time but this only marginally effects our results. See Fig. 9.5.

In a real A-D convertor, there are no impulses or multiplication. Instead, this is a model of what happens rather than exact hardware description. The train of impulses is given by:

$$\delta_T = \sum_{k=-\infty}^{\infty} \delta(t - kT_s) \tag{9.1}$$

When multiplied by the signal we get the product

$$f^*(kT_s) = f(t) \sum_{k=-\infty}^{\infty} \delta(t - kT_s) \tag{9.2}$$

where the star * superscript notation represents sampled-data as it only exists at regular intervals of time. We are viewing a snapshot at time $t = 0, T_s, 2T_s, 3T_s \ldots$ etc. Equation (9.2) is not really much to look at and we cannot discern much from it. Therefore, we look at the frequency-domain instead. We use the *Fourier-Transform* (FT). The FT has many of the properties of the Laplace-Transform we have already studied except instead of the variable being $s = \sigma + j\omega$, we consider the imaginary part only which is frequency. Therefore a FT cannot be used for

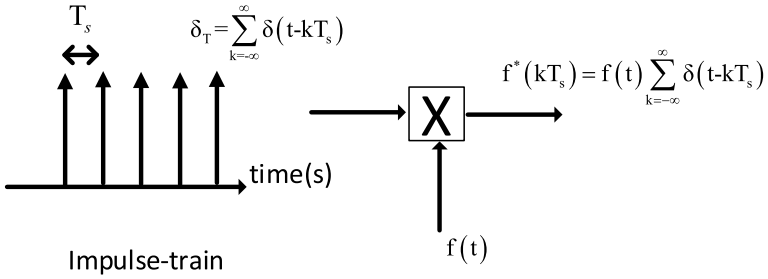


Fig. 9.5 Sampling process

transient analysis (step-response etc.) as it only can see frequency-domain properties of the signal. The magnitude of the FT is known as the *Spectrum* of the signal. We do not require a detailed explanation of the FT in order to use it and there is a table of its properties and common FT examples available in textbooks on signal-processing. The FT is defined as

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \tag{9.3}$$

Or in shorthand notation

$$F(\omega) = \mathcal{F}[f(t)] \tag{9.4}$$

One of the properties of the FT is the same as we obtained for the Laplace Transform, namely

The Fourier-transform of the convolution of two time-domain signals is the same is the product of their Fourier-Transforms. Mathematically we can write (where * denotes convolution)

$$\mathcal{F}[f_1(t) * f_1(t)] = \mathcal{F}[f_1(t)]\mathcal{F}[f_2(t)] \tag{9.5}$$

In the case of the Laplace-Transform, we used this to prove that we can multiply the transfer-functions of LTI systems and is the foundation for most of our control-theory analysis. We can also write this as

$$\mathcal{F}[f_1(t) * f_1(t)] = F_1(\omega)F_2(\omega) \tag{9.6}$$

The second property which we use is:

The Fourier-Transform of the product of two time-domain signals is the same as the convolution of their individual Fourier-Transforms.

In other words

$$\mathcal{F}[f_1(t)f_1(t)] = \mathcal{F}[f_1(t)] * \mathcal{F}[f_2(t)] \tag{9.7}$$

where * represents convolution. We can also write (9.7) as

$$\mathcal{F}[f_1(t)f_1(t)] = F_1(\omega) * F_2(\omega) \tag{9.8}$$

Now we can have a peek and see what effect multiplying a signal by a train of impulses has to a signal. We must take the FT of (9.2) and use the FT property of convolution in the frequency-domain.

$$\mathcal{F}[f^*(kT_s)] = \mathcal{F}\left[\sum_{k=-\infty}^{\infty} \delta(t - kT_s)\right] * \mathcal{F}[f(t)] \tag{9.9}$$

The FT of an impulse-train $r(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT_s)$ is a commonly met in the literature. Let

$$\begin{aligned} \mathcal{F}[r(t)] &= R(\omega) \\ &= \mathcal{F}\left[\sum_{k=-\infty}^{\infty} \delta(t - kT_s)\right] \end{aligned} \tag{9.10}$$

This has FT

$$R(\omega) = \frac{2\pi}{T_s} \sum_{k=-\infty}^{\infty} \delta(\omega - k\omega_s) \tag{9.11}$$

where the sampling frequency in rad/s is defined as $\omega_s = \frac{2\pi}{T_s}$ rad/s.

The details are not important but require getting the Fourier-series of the train of impulses and then taking the FT. This well-known result states that

The Fourier-Transform of a train of impulses in the time-domain is a train of impulses in the frequency-domain.

This is quite a fascinating result. We have impulses of energy in the frequency-domain spaced-apart by the sampling-frequency ω_s. Now the FT of the signal itself we can call

$$F(\omega) = \mathcal{F}[f(t)] \tag{9.12}$$

We must now convolve in the frequency-domain.

$$\begin{aligned} \mathcal{F}[f^*(kT_s)] &= \mathcal{F}\left[\sum_{k=-\infty}^{\infty} \delta(t - kT_s)\right] * \mathcal{F}[f(t)] \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} R(\beta) F(\omega - \beta) d\beta \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{2\pi}{T_s} \sum_{k=-\infty}^{\infty} \delta(\beta - k\omega_s) F(\omega - \beta) d\beta \end{aligned} \tag{9.13}$$

Finally

$$\mathcal{F}[f^*(kT_s)] = \frac{1}{T_s} \sum_{k=-\infty}^{\infty} F(\omega - k\omega_s) \tag{9.14}$$

Equation (9.14) is a famous formula derived by several people. It was originally derived by an English Statistician E. T. Whittaker in 1915 [25] and Kotelnikov in

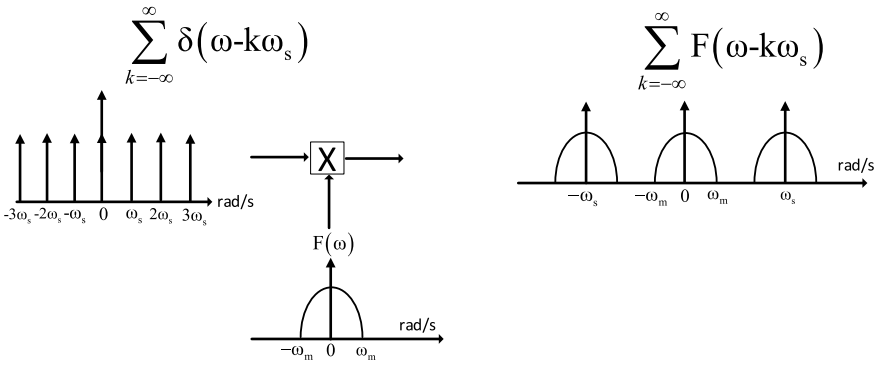


Fig. 9.6 Sampling represented in the frequency-domain showing the spectra

1933 [26]. However, in the West, Claude Shannon published his version which is usually taken as the most relevant to electrical engineering [27]. Nyquist who made great contributions to stability theory also played a part in the story of sampling [28] and often the sampling theory is referred to as the Nyquist-Shannon theory. It appears to be still unclear of the contribution made by Nyquist, although the half-sampling frequency is named in his honour as the Nyquist frequency. Kotelnikov and Whittaker are all but a memory.

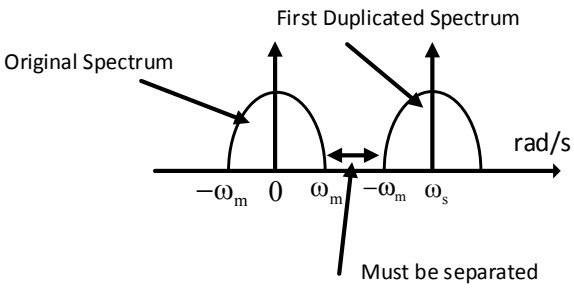
Equation (9.14) represents the original spectrum of the signal repeated indefinitely at multiple integer multiples of the sampling frequency. This is illustrated in Fig. 9.6.

In Fig. 9.6 we see the multiple repeated spectra that occurs. The crux of the multiple spectra however, is to see that for the first one its maximum frequency is ω_m, since that is its band-limited upper frequency. However, the next repeated spectrum must have as its lower-most frequency $\omega_s - \omega_m$. This is shown in Fig. 9.7.

Therefore, to avoid a collision we must have that

$$\omega_s - \omega_m > \omega_m \tag{9.15}$$

Fig. 9.7 Separation needed between spectra



or

$$\omega_s > 2\omega_m \tag{9.16}$$

Equation (9.16) is referred to as the *Sampling Theorem* and tells us that to reconstruct the signal to its original, there must be separation between the spectra. Of course, this repeats itself for all of the other multiple spectra and the same theorem holds. In words (and converted to Hz instead of rad/s)

A band-limited continuous-time signal, with highest frequency B Hz can be uniquely recovered from its samples provided that the sampling rate is greater than 2B samples per second.

The sampling-theorem is often miss-quoted as the sampling frequency needed to be twice the *maximum frequency* of the signal. In fact, this is incorrect, it should be more than twice the *bandwidth* of the signal. This miss-quote however is often true because for control-systems the baseband signals (or frequency-response of the systems) nearly always have a frequency-response down to dc, so the bandwidth *is also* the maximum frequency! In communication systems though we cannot get away with this because there are signals centred at high frequencies (say 1 GHz) with bandwidths of 5 kHz. They need only be sampled at a frequency greater than 10 kHz since the bandwidth of the signal does not reach down to dc. We concentrate in this text on signals with frequency-response that go down to dc. This has to be the case for all dc servos at least. A motor must be able to respond down to dc in order to operate in steady-state. Also, be aware that here signals and systems are interchangeable since by convolution and LTI systems-theory they can be treated the same. Hence, we look at the bandwidth of the overall control-system and sample higher than twice the bandwidth. The bandwidth is usually defined as the unity-gain crossover frequency (by design at least) or a structural-resonance of some sort that defines the absolute upper-limit on bandwidth. We shall see later that sampling twice the bandwidth is not nearly enough when feedback is applied around a system. In open-loop signal processing, often we can just about get away with sampling three to four times the bandwidth, but not when feedback is applied. However, for the time being we look at what happens when we do not satisfy the sampling theorem.

9.4 Aliasing

If the sampling theorem is not satisfied then we get an overlap of spectra as shown in Fig. 9.8.

Clearly, the two spectra are now inseparable and the original one cannot be recovered without a part of the low-frequencies of the second one mixed within it. This is known as *aliasing* and cannot be allowed to happen. Therefore we must apply a low-pass filter before the A-D conversion (See Fig. 9.4) to limit the bandwidth of the signal so no power appears after half the sampling frequency. Half

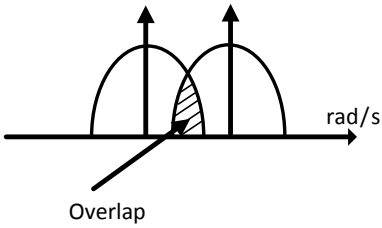


Fig. 9.8 Overlap of spectra

sampling frequency $\omega_s/2$ or $f_s/2$ is known as the *Nyquist-Frequency*. We must limit as far as possible any signal that has power above the Nyquist frequency. In practice it is not possible to eliminate all power above the Nyquist frequency. We must reduce it enough so that it is no longer a problem. In theory this would mean reducing the power of the signal above $f_s/2$ so that it is below the quantization level of the A-D convertor. Then we have a situation where any aliasing cannot be discerned from quantization noise. This too can often be a difficult task to satisfy since it will often require very high-order analogue filters. Note that aliasing cannot be fixed after it has occurred! Once a signal is aliased, you cannot recover a clean version of it. Usually we use a Butterworth filter since it has a flat passband with no ripple and does not change the amplitude characteristics in any way until the roll-off frequency. We can view this better by considering the aliasing of a pure sine wave and thinking of a virtual mirror at half-sampling frequency (Fig. 9.9).

We can see that the Nyquist frequency (half-sampling) can be thought of as a virtual mirror. Any signal above this that is sampled (assuming no filter) gets reflected back by the same 'distance" before the mirror as it appears after the mirror. For example, if we sample at 10 kHz and have a sine wave at 6 kHz, then this is 1 kHz *above* the Nyquist frequency (which is 5 kHz). It therefore gets reflected back by 1 kHz *before* the Nyquist frequency or 4 kHz. The amplitude is unaffected and the phase we don't care in any case. We can see that by filtering this unwanted signal above half-sampling, we can at least chop it down in size a bit, so that when it does get reflected back, its size (amplitude) will be reduced. This is the concept of an anti-aliasing filter (Fig. 9.10).

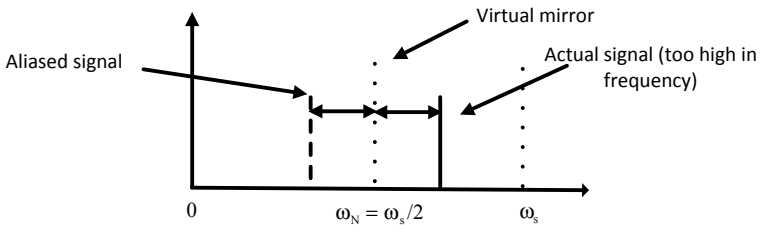


Fig. 9.9 Aliasing of a sine wave

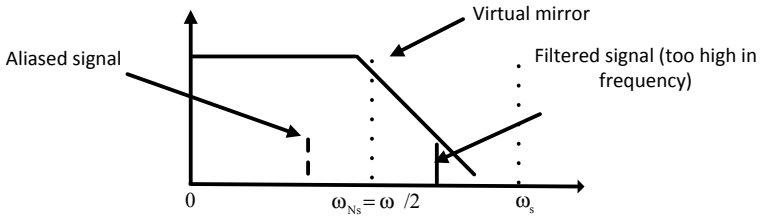


Fig. 9.10 Anti-aliasing filter concept

Recall from analogue control that phase-margin decides the stability of a closed-loop system. Therefore, any phase-shift that is introduced by this low-pass filter will de-stabilise the closed-loop system. We must therefore be careful to sample high-enough so that half-sampling is also well above our unity-gain closing frequency of our Bode-plot. In this way, any phase-shift will be kept as low as possible. It is inevitable that our anti-aliasing filter will do some harm to phase-margin in a real system unless we can sample high-enough.

Aliasing appears in other scientific areas other than control-systems and signal processing. Anybody who has watched a movie of waggon-wheels turning in an old western will have noticed that the wheels start off moving in the right direction and then move backwards! This is because a film is a sampling process of images taken at regular instances in time. Once the speed of the wheel turns at a greater rate than the Nyquist rate of the frame shutter speed, we get false information and the wheel looks as if it turns backwords. The same effect can be seen on helicopter blades when filmed. Aliasing is also a problem in image compression.

9.5 The D-A Reconstruction

At the D-A we can find out the contribution made by the device if we assume its impulse-response has the shape shown in Fig. 9.11.

Is you were to send a blip of data out the D-A it would hold the output at some value (here normalised to unity) for one sampling interval only and then go to zero. This is its impulse response $h(t)$ and if we take the Fourier Transform, we can essentially obtain its Bode-plot. The FT is

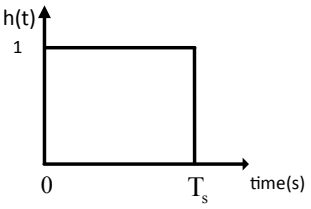


Fig. 9.11 Impulse-response of D-A (zero-order hold)

$$H(\omega) = \int_0^{T_s} h(t)e^{-j\omega t}dt \tag{9.17}$$

Since the amplitude is unity

$$\begin{aligned} H(\omega) &= \int_0^{T_s} e^{-j\omega t}dt \\ &= -\left[\frac{1}{j\omega}e^{-j\omega t}\right]_{t=0}^{t=T_s} \\ &= \frac{1 - e^{-j\omega T_s}}{j\omega} \end{aligned}$$

Now we use a trick to get this into a familiar format by introducing an exponential Euler term

$$\begin{aligned} H(\omega) &= \frac{1 - e^{-j\omega T_s}}{j\omega} \\ &= \frac{e^{-j\omega T_s/2}\left(e^{j\omega T_s/2} - e^{-j\omega T_s/2}\right)}{j\omega} \\ &= \frac{2}{\omega}\frac{e^{-j\omega T_s/2}\left(e^{j\omega T_s/2} - e^{-j\omega T_s/2}\right)}{2j} \\ &= e^{-j\omega T_s/2}\left(\frac{2}{\omega}\right)\sin(\omega T_s/2) \end{aligned}$$

Now we divide and multiply by T_s which gives us

$$H(\omega) = e^{-j\omega T_s/2}T_s\frac{\sin(\omega T_s/2)}{\omega T_s/2} \tag{9.18}$$

This is the famous $\frac{\sin(X)}{X}$ function which crosses the horizontal axis at $x = \pm\pi, \pm 2\pi, \pm 3\pi \ldots$. Our expression (9.18) must also cross the frequency axis at $\omega T_s/2 = \pi, 2\pi, 3\pi \ldots$ or in terms of frequency $\omega = \pm\left(\frac{2\pi}{T_s}, \frac{4\pi}{T_s}, \frac{6\pi}{T_s} \ldots\right) = \pm(\omega_s, 2\omega_s, 3\omega_s \ldots)$. Taking the magnitude of (9.18)

$$|H(\omega)| = \left|T_s\frac{\sin(\omega T_s/2)}{\omega T_s/2}\right| \tag{9.19}$$

and the phase (we are only interested in the phase up to the Nyquist frequency anyway

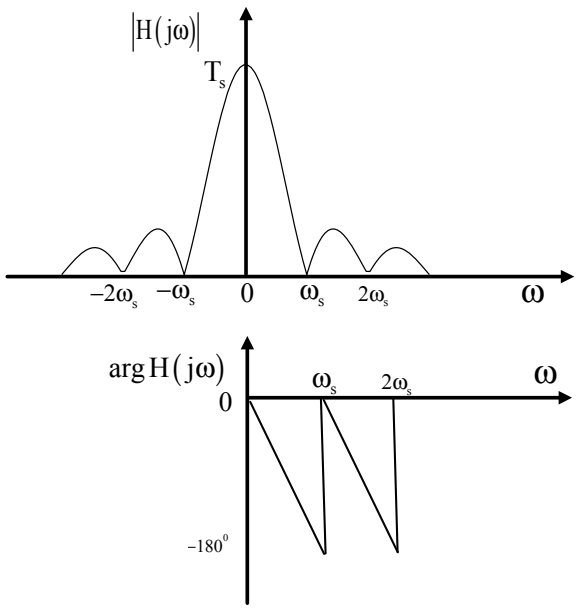


Fig. 9.12 Zero-order hold frequency-response

$$\arg(H(\omega)) = -\omega T_s/2 \tag{9.20}$$

We can write (9.20) more conveniently as

$$\arg(H(\omega)) = -\left(\frac{\omega}{\omega_s}\right)\pi \tag{9.21}$$

So at the sampling frequency the phase is −180° and zero at dc. It is a linear slope between these frequencies. The magnitude and phase are shown in Fig. 9.12.

It is called a zero-order hold because it holds the signal at whatever level it is given out at for one sample interval. The importance of this Bode-Plot (up to half-sampling frequency only) is that the magnitude drop (or droop as it is termed) is not that significant, but there is a phase-shift of −90° at half-sampling frequency. Therefore, an such digital-control-system will have this delay embedded into the loop which is not a good thing. It means that we must sample much higher than the two to three times which is the bare minimum for an open-loop signal or system. We need to sample at least ten times the bandwidth of our open-loop system to be sure that this phase is low-enough to not cause any significant harm to our phase-margin. At/10th sampling frequency we will have a phase-shift of −18° and we would need to make the bandwidth of our loop 1/10th smaller than this again so that we would only then be effected by a few degrees of phase-shift. If we sample high-enough, which is the case with today's computing power, then we can all but ignore the effects of the zero-order hold (ZOH). If we cannot get a fast enough

sampling frequency (and this was the case in the early days of digital control) then we must take account of the transfer-function of the ZOH.

It is east to show that the Laplace-Transform of the ZOH is

$$\begin{aligned} H(s) &= \int_0^{T_s} h(t)e^{-st}dt \\ &= \frac{1 - e^{-sT_s}}{s} \end{aligned} \tag{9.22}$$

This transfer-function was cascaded with the transfer-function of the system and used instead of the actual system transfer-function. With the power of modern computing we don't need this stage of analysis any more as the effect of the ZOH becomes insignificant at high sampling rates. Of course having a high sample rate can unfortunately also introduce other problems. The problem is caused because the output of the D-A feeds back into the system, but for open-loop systems (for example digital filters) we are not so troubled since a linear phase-shift is fairly benign. If you have to have a phase-shift then a linear phase-shift is better than ordinary non-linear phase since the shape of a signal is unaffected. In fact a linear phase-shift acts a bit like a time-delay and only really effects the phase. As damaging as this is in closed-loop, we do not care too much in open-loop filtering problems.

The D-A output is shown in Fig. 9.13 for a typical signal.

The envelope of this staircase waveform is obtained by filtering after the D-A. This is another low-pass filter called the *interpolation* filter. With the frequent use of H bridges at least for motor drivers, they too have a pulsed output that needs smoothing. However, this is done by the low-pass characteristics of the motor itself, so no extra filter is needed. Adding any extra filter to a control-loop will reduce phase-margin and so a phase-budget must be obtained to determine how much phase-shift we can withstand or what loss of performance is considered negligible for our design. The crucial frequency is the unity-gain crossover frequency, which

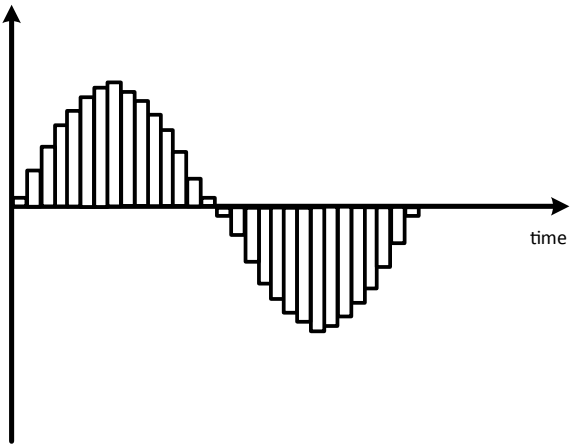


Fig. 9.13 D-A output or ZOH output

defines the phase-margin. This is another reason why for small benchtop servos at least, designing in the frequency-domain has many advantages over time-domain methods or state-space.

9.6 The z-Transform

The z-transform is our Laplace-transform for sampled-data systems. Its origins are a little sketchy, but we do know that Ragazzini and Zadeh [29] popularised the method in the early 1950s, long before digital computers were in common use. Jury [24] later published a book on sampled-data systems which became one of the founding textbooks on the subject. To derive the z-transform we return to (9.2) but take the Laplace transform rather than the Fourier transform.

$$\mathcal{L}[\mathrm{f}^*(\mathrm{kT_s})] = \mathcal{L}\left[\mathrm{f(t)} \sum_{\mathrm{k}=-\infty}^{\infty} \delta(\mathrm{t} - \mathrm{kT_s})\right]$$

Obtain

$$\begin{aligned} &\int_0^t \mathrm{f(t)} \sum_{\mathrm{k}=-\infty}^{\infty} \delta(\mathrm{t} - \mathrm{kT_s}) e^{-st} dt \\ &= \int_0^t \sum_{\mathrm{k}=-\infty}^{\infty} \mathrm{f(t)}\delta(\mathrm{t} - \mathrm{kT_s}) e^{-st} dt \end{aligned} \tag{9.23}$$

This integral is zero everywhere except at the time the impulses occur. We also do not need to sum from minus infinity because our signals and systems will be causal. That is, their impulse-response (or waveform) begins at time zero or later and does not exist for negative-time. In addition, the output must follow after the input is applied. It is possible to use the two-sided Laplace transform and end up with the so-called two-sided z-transform but we do not need this level of complexity at this stage. Therefore, it is simpler to write

$$\int_0^t \sum_{\mathrm{k}=0}^{\infty} \mathrm{f(t)}\delta(\mathrm{t} - \mathrm{kT_s}) e^{-st} dt = \sum_{\mathrm{k}=0}^{\infty} \mathrm{f(kT_s)} e^{-s\mathrm{kT_s}} \tag{9.24}$$

For discrete-time signals, we use many ways of writing discrete signals in shorthand without having to write explicitly the sampling interval as part of the time index. Instead of writing $\mathrm{f(kT_s)}$ we can simply just write $\mathrm{f(k)}$ instead or as a subscript f_k. It is taken as known that the index k occurs at each sampling interval. Therefore we write $f(kT_s) = f_k$ as our notation. However, (9.24) has a an

exponential term within it which is irrational and difficult to deal with. So instead we define a new variable

$$z = e^{sT_s} \tag{9.25}$$

We call this the z-transform operator (just like s is for Laplace and continuous-time). It is a complex-valued operator just as with the Laplace case. Using (9.25) in (9.24) we get

$$\sum_{k=0}^{\infty} f_k z^{-k} = F(z) \tag{9.26}$$

where $F(z)$ is defined as the *one-sided* z-transform of the time-domain sequence f_k. This is the game-changer for discrete-systems just as Laplace was for the continuous-time. The *two-sided* z-transform is the same except the sum is taken from minus infinity. We only use the one-sided case in this chapter.

Small digression to study a *Geometric-Sequence*

In order to look at some examples of the z-transform we first need to review some basic mathematics of geometric progressions (or geometric sequences). This is a bit like a maths puzzle where we are given the first few numbers and told to find the next one. For example 1, 2, 4, 8… The next number is 16 since each consecutive number is multiplies the previous one by 2. If we were to add the numbers to infinity we would get infinity as the answer, but what about 1, 0.5, 0.25, 0.125… In this case the previous number is multiplied by 0.5 to get the new number. The next number gets progressively smaller than the previous. The 0.5 multiplier is known as the *growth factor* and if the numbers are written as a sum we could write the sum to *n* terms as

$$S_n = a + ar + ar^2 + ar^3 + \ldots ar^n \tag{9.27}$$

where *a* is the first term and *r* is the growth factor. We can immediately see that if the growth factor is less than one, that the numbers in the sequence progress to be smaller and smaller. In fact the sequence converges as in any Taylor or Maclaurin series. If the growth-rate has value greater than one the series (or sequence) diverges or grows infinite. We therefore assume that

$$|r| < 1 \tag{9.28}$$

We require the sum to infinity as a closed expression so we multiply (9.27) by the growth-rate to get

$$rS_n = ar + ar^2 + ar^3 + ar^4 + \ldots ar^{n+1} \tag{9.29}$$

Now subtract (9.27)–(9.29)

$$S_n - rS_n = a - ar^{n+1} \tag{9.30}$$

Collect terms and solve for the sum gives

$$S_n = \frac{a(1 - r^{n+1})}{1 - r} \tag{9.31}$$

This is not the sum to infinity of course so we need to let $n \to \infty$ in (9.31) and we get the final result that the sum to infinity of a geometric progression or sequence is

$$S_\infty = \frac{a}{1 - r} \tag{9.32}$$

This holds provided $|r| < 1$. In reverse, we can also say that

$$\frac{a}{1 - r} = ar + ar^2 + ar^3 + \ldots \tag{9.33}$$

which we could have found be expanding using a Maclaurin-series or even long division.

We can now return to the z-transform.

9.6.1 z-Transform of a Decaying Sequence

We do not really get exponential decay in sampled-data systems, the next-best think are terms such as

$$f_k = \alpha^k \tag{9.34}$$

If the alpha term is less than unity then at each consecutive sample the terms die-out a bit like exponential decay (or as near as we will get for a simple expression). Take its z-transform

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

which is an infinite some or geometric progression with growth factor αz^{-1}. The first-term is unity so we write the sum to infinity by using (9.32) as

$$\begin{aligned} S_\infty &= \frac{a}{1-r} \\ &= \frac{1}{1-\alpha z^{-1}} \end{aligned} \tag{9.35}$$

Therefore

$$\begin{aligned} F(z) &= \frac{1}{1-\alpha z^{-1}} \\ &= \frac{z}{z-\alpha} \end{aligned} \tag{9.36}$$

This is its z-transform and we note that for convergence we must have

$$\left|\alpha z^{-1}\right| < 1$$

Or alternatively in terms of z

$$|z| > |\alpha|$$

This is known as the *region of convergence.* We usually quote it in problems that have both so called left and right-handed sequences which occur in the two-sided z-transform [30]. Since we are only dealing with the one-sided case, it is not so much of a concern. The reason why it is important in more general problems with the two-sided transform, is that two sequences can have the same z-transform, and confusion will arise. The sort of sequences we are talking about though usually have impulse-responses which go backwards in time or are known as non-causal. In fact, any Talyor-series or power-series can be tested for convergence, a topic covered in basic linear algebra. This is done usually with a ratio test. We meet such problems in optimal filters but not so commonly in ordinary classical control. For short-hand, we can write the z-transform result as

$$\mathcal{Z}\left[\alpha^k\right] = \frac{z}{z-\alpha} \tag{9.37}$$

Or in reverse, the inverse-z-transform as

$$\mathcal{Z}^{-1}\left[\frac{z}{z-\alpha}\right] = \alpha^k \tag{9.38}$$

Figure 9.14 shows such a decaying sequence for $\alpha = 0.5$ and for when $\alpha = 1$, a sampled unit-step sequence.

As with the Laplace-Transform, we can have the z-transform of signals or systems and often there is a signal which has the same transform as a system. Just as

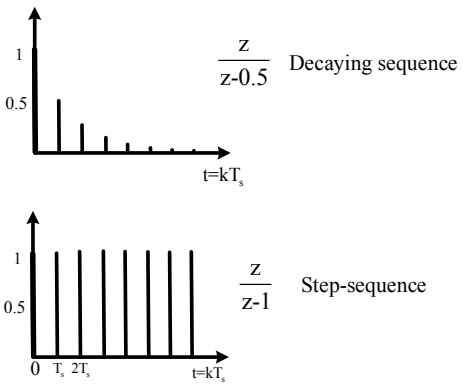


Fig. 9.14 Decaying sequence 0.5^k and sampled unit sequence (step)

Table 9.1 Common one-sided z-transforms

Time-domain f_k	z-domain $F(z)$
δ_k unit impulse at k = 0	1
$f_k = 1$, unit step function	$\frac{Z}{Z-1}$
$f_k = k$ unit ramp	$\frac{Z^2}{(Z-1)^2}$
$f_k = (k)^2$ parabolic input	$\frac{Z^2}{(Z-1)^2}$
$f_k = e^{-ak}$	$\frac{Z}{Z-e^{-a}}$
$f_k = a^k$	$\frac{Z}{Z-a}$
$\cos(\Omega k)$	$\frac{Z^2-Z\cos(\Omega)}{Z^2-2Z\cos(\Omega)+1}$
$\sin(\Omega k)$	$\frac{\sin(\Omega)Z}{Z^2-2\cos(\Omega)Z+1}$

with the Laplace case, the z-transform of a unit-step (signal) is the same as that of an integrator (system). Nothing significant about this but a simple way of remembering the transform without looking it up for both cases. Of course we can do some mental algebra, to integrate a step we should get a ramp and therefore apply the z-transform for a step and integrate it with the z-transform of an integrator. So clearly the z-transform of a ramping signal must be $\left[\frac{Z}{Z-1}\right]^2$ just as by analogy with the continuous-time case we had $\frac{1}{s^2}$. We don't really need to know many z-transforms, but there is a table of commonly met ones shown below for unity sample-interval (Table 9.1).

9.7 Step-Response Example

The z-transform satisfies all the properties of LTI systems with analogy to continuous-time systems. We can apply similar rules. Take for example a first-order system given by $G(z) = \frac{Z}{Z-0.2}$. Suppose we are required to find the step-response of the system. We write

$$\begin{aligned} y(z) &= G(z)u(z) \\ &= \left(\frac{z}{z-0.2}\right)\left(\frac{z}{z-1}\right) \end{aligned}$$

Just as with the Laplace-transform, we see that the combination of the step with the system produces a product which is not in the tables. We therefore use partial-fractions to separate this product thus:

$$\left(\frac{z}{z-0.2}\right)\left(\frac{z}{z-1}\right) = \frac{A}{z-0.2} + \frac{B}{z-1} \tag{9.39}$$

and find constants *A* and *B*. Is we do this however we still find that there is no term in the tables with the form $\frac{1}{z-a}$ though we do have $\frac{z}{z-a}$. (The constants *A* and *B* only form multiplier constants.) We therefore first divide by *z*.

$$y(z)/z = \left(\frac{z}{z-0.2}\right)\left(\frac{1}{z-1}\right) = \frac{A}{z-0.2} + \frac{B}{z-1} \tag{9.40}$$

Now find the constants as $A = -0.25$, $B = 1.25$. Now multiply (9.40) by *z* and we get

$$y(z) = \left(\frac{z}{z-0.2}\right)\left(\frac{z}{z-1}\right) = \frac{-0.25z}{z-0.2} + \frac{1.25z}{z-1} \tag{9.41}$$

Use the tables and take the inverse z-transform of each term.

$$\begin{aligned} y_k &= \mathcal{Z}^{-1}G(z)u(z) \\ &= \mathcal{Z}^{-1}\left(\frac{-0.25z}{z-0.2} + \frac{1.25z}{z-1}\right) \\ &= 1.25 - 0.25(0.2)^k \end{aligned} \tag{9.42}$$

We can see that as k gets large the second term dies out giving us a steady-state solution of *1.25*. This is a similar example to a continuous-time first-order system response to a step. It rises up to a steady-state value. A quicker way to check the working is to use the discrete-time version of the *final-value theorem* which states that provided the system is stable (we fully describe stability later), its final value can be found from:

Discrete-time final-value theorem

$$y_{\substack{k \\ \lim k\to\infty}} = \lim_{z\to 1}(z-1)y(z) \tag{9.43}$$

Applying this to

$$y(z) = \left(\frac{z}{z - 0.2}\right)\left(\frac{z}{z - 1}\right)$$

we get

$$y_{\substack{k \\ \lim k \to \infty}} = \lim z \to 1 (z - 1)\left(\frac{z}{z - 0.2}\right)\left(\frac{z}{z - 1}\right)$$

Now the terms in *(z − 1)* cancel and the result becomes *1/(1 − 0.2) = 1.25*.

9.8 Stability of Discrete-Time Systems

We use continuous-time as the standard for stability in order to discover what happens in discrete-time. From (9.25), $z = e^{sT_s}$ is the fundamental relationship between s and z. This is a mapping from one-plane (Laplace) to another one which we name the z-plane. To figure out what is happening, it is best to first consider the continuous-time case and look at the dividing line between stability and instability. Instability happens when poles are in the right-half s-plane and stability when the poles are in the left-half plane. Therefore the imaginary axis is the dividing line between stability and instability. Substitute $s = j\omega$ into (9.25) and we obtain

$$z = e^{j\omega T_s} \tag{9.44}$$

Taking its magnitude we get

$$|z| = \left|e^{j\omega T_s}\right| = 1 \tag{9.45}$$

This is a circle of radius one centred at the origin. We see that the imaginary axis of the continuous-time case maps to a circle of radius one in the z-domain. But what about the region of stability? To find this substitute a stable pole in the form $s = -\sigma + j\omega$ where $\sigma > 0$ into (9.25) and get

$$z = e^{-\sigma T_s + j\omega T_s} \tag{9.46}$$

Find the magnitude

$$\begin{aligned} |z| &= \left|e^{-\sigma T_s + j\omega T_s}\right| \\ &= \left|e^{-\sigma T_s}\right|\left|e^{j\omega T_s}\right| \\ &= \left|e^{-\sigma T_s}\right|.1 \end{aligned} \tag{9.47}$$

Now $|e^{-\sigma T_s}| < 1$ indicating that all of the left-hand plane must map *within* a circle of radius one in the z-plane. The situation is slightly more complicated because we are of course as always limited by the sampling-frequency. We know $\omega_s = \frac{2\pi}{T_s}$ as the sampling frequency in rad/s. Substitute $T_s = \frac{2\pi}{\omega_s}$ into (9.44). In (9.44) $z = e^{j\omega T_s}$ considers the imaginary axis only since we are only interested in frequency.

$$\begin{aligned} z &= e^{j\omega T_s} \\ &= e^{2\pi j \frac{\omega}{\omega_s}} \end{aligned} \tag{9.48}$$

In (9.48) we can select various values of analogue frequency and find the equivalent discrete version. For instance, beginning with dc, when $\omega = 0, z = 1$. This is the edge of the unit-circle at a point *1 + j0*. Now consider the Nyquist frequency or half-sampling frequency. This is the highest frequency we can achieve for a given sampling-interval. When $\omega = \omega_s/2, z = e^{j\pi} = -1 + j0$. This is again on the unit circle half way round if we were to move on the edge of the circle in an anti-clockwise rotation. Similarly, quarter sampling frequency appears at *0 + j1*.

The shaded areas in Fig. 9.15 are equivalent in continuous and discrete-time. Notice that for the s-plane we can only consider up to half-sampling. As far as the discrete sampled system is concerned, no frequency above half-sampling can ever exist. It is blissfully unaware of the real analogue world! We move around the circle from dc at 1 + j0 in a semi-circle anti-clockwise through 0 + j1 to −1 + j0 and that is the upper limit that exists. Any poles (can be real or complex) that are in the shaded area on the left of Fig. 9.15 must appear inside the unit circle of the z-plane. Looking at this in reverse:

For any z transfer-function, its poles must lie within the unit circle of the z-plane in order to be stable.

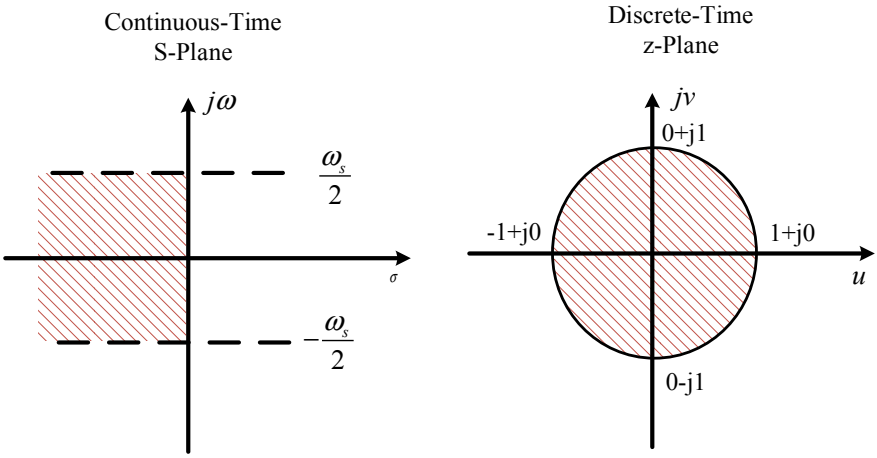


Fig. 9.15 Mapping from s to z

$G(z) = \frac{\mathrm{Z}}{\mathrm{Z}-0.2}$ is therefore a z-transfer function with a pole at z = 0.2 has magnitude 0.2 which is less than one and therefore stable. It also has a zero at the origin.

$G(z) = \frac{\mathrm{Z}}{\mathrm{Z}^2-z+0.5}$ has a complex-pair of poles at $z = 0.5 \pm j0.5$. Their magnitudes are $|z_1|, |z_2| = \sqrt{0.5^2 + 0.5^2} = 0.707 < 1$ which are both stable. There is also a zero at the origin. The transfer-function $G(z) = \frac{(z-2)}{(z-0.5)(z-0.1)}$ has two stable poles at 0.5 and 0.1 and is therefore stable. It has a zero at z = 2 which is outside of the unit circle (and therefore non-minimum phase). This does not effect the stability of an open-loop system however. Just as with the Routh-test for continuous-time there is a test for discrete-time polynomials in z to determine whether they have magnitudes less than unity. It differs from the Routh method and is known as a Jury stability test. As with the continuous-time case it is rarely used nowadays since root-finders are so commonly available.

9.9 Normalised Frequency and Frequency-Response

It makes sense to often consider the mapping as being independent of sampling frequency by inventing a new type of frequency called *normalised-frequency*. Normalised frequency is defined as

$$\theta = \omega \mathrm{T_s} \tag{9.49}$$

Using normalised frequency we don't care about the sampling-frequency and take its normalised version in the form of (9.49) to range from 0 to π. Our new representation is then $z = e^{j\theta}$. If we go beyond pi, we just repeat things over and over again. There are no nice asymptotic frequency-response methods for discrete-time systems that exist for continuous-time. We can estimate the shape of simple ones, but graphing software is usually employed instead of hand calculations. As an example consider the first-order z transfer function $G(z) = \frac{1}{\mathrm{Z}-0.5}$. To find its frequency-response we simply substitute $z = e^{j\theta}$ and find its magnitude and phase. A hand calculation yields

$$\mathrm{G}\left(\mathrm{e}^{\mathrm{j}\theta}\right) = \frac{1}{\mathrm{e}^{\mathrm{j}\theta} - 0.5} \tag{9.50}$$

Using Euler's identity

$$\mathrm{G}\left(\mathrm{e}^{\mathrm{j}\theta}\right) = \frac{1}{\cos(\theta) + \mathrm{j}\sin(\theta) - 0.5} \tag{9.51}$$

The magnitude is

$$\left|\mathrm{G}\left(\mathrm{e}^{\mathrm{j}\theta}\right)\right| = \frac{1}{\sqrt{(\cos(\theta) - 0.5)^2 + \sin^2(\theta)}} = \frac{1}{\sqrt{1.25 - \cos(\theta)}}$$

At dc, $\theta = 0$ and the magnitude is $\left|G\left(e^{j\theta}\right)\right| = \frac{1}{\sqrt{1.25-1}} = 2$.

At half-sampling frequency $\theta = \pi$ and the magnitude becomes $\left|G\left(e^{j\theta}\right)\right| = \frac{1}{\sqrt{(\cos(\pi)-0.5)^2 + \sin^2(\pi)}} = \frac{1}{\sqrt{(-1-0.5)^2+0}} = 0.667$. If we go beyond half sampling frequency then the frequency-response just repeats itself over and over again. The phase can be calculated from (9.51)

$\arg\left(G\left(e^{j\theta}\right)\right) = -\tan^{-1}\left(\frac{\sin(\theta)}{\cos(\theta)-0.5}\right)$. The phase is zero at dc and at half-sampling frequency. It has a maximum negative phase-shift which could be found by differentiating the expression for phase, but it is much easier to use MATLAB!

The MATLAB program for dB magnitude is as follows and produces the frequency-response of Fig. 9.16.

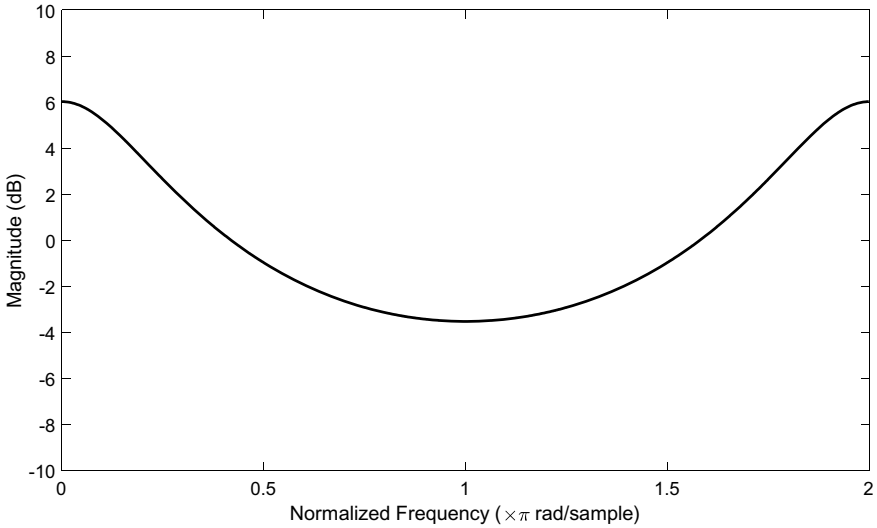


Fig. 9.16 Frequency-response (magnitude in dB) of first-order discrete-time system

MATLAB Code 9.1

```
%dB Magnitude frequency response of first-order system
b=1;
a=[1 -0.5];
[h,w] = freqz(b,a,'whole',2001);
plot(w/pi,20*log10(abs(h)))
ax = gca;
ax.YLim = [-10 10];
ax.XTick = 0:.5:2;
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Magnitude (dB)')
```

Note that the frequency-axis is multiplied by pi and so it goes from dc to pi and then 2pi. After pi it is just a repeat of before pi and if we keep going it repeats endlessly. It is a typical low-pass characteristic of a first-order system with no zeros. Of course this system does have one zero but zeros at the origin have no effect on frequency response magnitude.

To find the phase we use the following code:

MATLAB Code 9.2

```
%Phase response of first-order system

b=1;
a=[1 -0.5];
[h,w] = freqz(b,a,'whole',2001);
plot(w/pi,(angle(h)*(180/pi)))
ax = gca;
ax.YLim = [-50 50];
ax.XTick = 0:.5:2;
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Phase (degrees)')
```

Figure 9.17 shows the phase-response for the same system.

The phase up to half-sampling (or pi) is the actual phase where it is lagging. Beyond that the phase the sign changes and the phase leads, but beyond half-sampling we cannot venture since such frequencies do not exist. Also of interest is that the phase does not go down to $-90°$ but returns to zero at

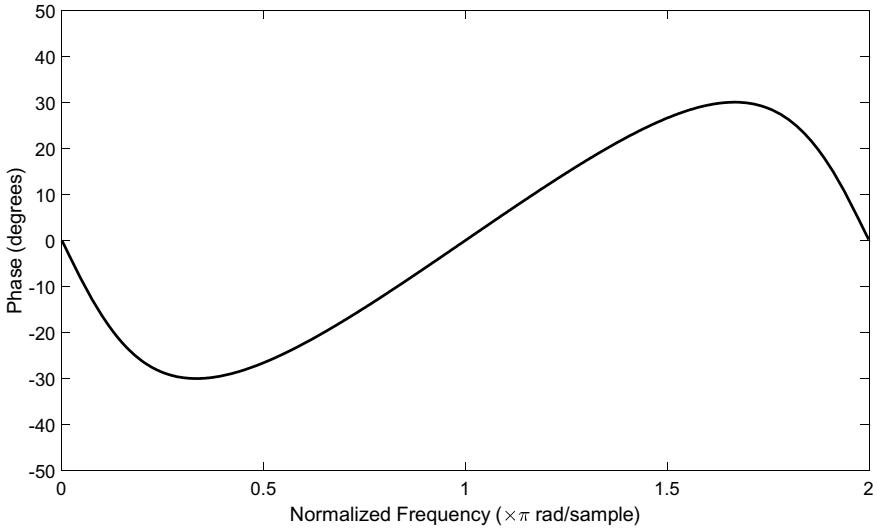


Fig. 9.17 Phase response of first-order example

half-sampling. This is because this is not an accurate representation of a pure first-order analogue system. We can get better approximations as we will see later.

9.10 Impulse-Response and Convolution

Just as with continuous-time systems, discrete-time systems also have an impulse-response. It can be found either numerically by putting a unit impulse at time zero into the discrete-system, or more usually by simply the inverse z-transform of its transfer-function. For example, for the system $G(z) = \frac{z}{z-a}$ we define its impulse-response sequence as its inverse z-transform. Thus

$$\mathcal{Z}^{-1}[G(z)] = g_k, k = 0, 1, 2\ldots \tag{9.52}$$

For our example $\mathcal{Z}^{-1}\left[\frac{z}{z-a}\right] = a^k$ (from the z-transform table). Of course our earlier example started from this sequence to get the z-transform so we are already familiar with this problem. The thing to notice however is that the number of points in the impulse-response is in theory infinite. For our example, the terms in the impulse-response are

$$a^k = \left\{1, a, a^2, a^3\ldots\right\} \tag{9.53}$$

Of course they get so small after only a small number of terms as to be insignificant, but the impulse-response is said to be an *infinite impulse-response*. This is usually shortened to IIR or IIR filters when referring to the systems. For a stable system its impulse-response sequence will die out and for an unstable-system it will grow. Given two such impulse-responses we can show in a similar way to continuous-time, that by convolution, we get the same result as multiplying the z transfer-functions together. So for two IIR sequences g_k and h_k their discrete-time convolution is defined as a third sequence—say f_k

$$f_k = \sum_{i=0}^{k} g_{k-i} h_i, k = 0, 1, 2 \ldots \tag{9.54}$$

The summation can have an upper limit of infinity where necessary.

By taking z transforms of (9.54) we simply arrive at

$$F(z) = G(z)H(z) \tag{9.55}$$

Which is two transfer-functions in cascade. We say that

Convolution in the time-domain is the same as multiplication in the frequency-domain.

This can also be proven using the discrete-time version of the Fourier-transform and is covered in most textbooks on digital-signal processing (DSP). The convolution sum itself is time-consuming however for large impulse-responses. More than often the fast-Fourier-transform (FFT) is used instead to perform such computations by multiplying in the frequency-domain.

There is a slight and important difference between discrete and continuous-time here however. We know that it is physically unrealisable to have continuous-time systems with more zeros than poles or just with zeros alone. This is not the case for discrete-systems. We can cheat and design filters with only zeros making them always stable. The cheat is that in actual fact they do have poles but the poles are usually neglected since they are all at the origin and have no effect. For example, the impulse response

$$h_k = \{1, 2, 3, 4\} \tag{9.56}$$

has only 4 values and of finite duration. Therefore we call it a *finite-impulse-response* or FIR.

To see what its transfer-function looks like we take z-transforms. This is particularly simple since each term just gets multiplied by a z^{-1} term depending on its position. So

$$H(z) = 1 + 2z^{-1} + 3z^{-2} + 4z^{-3} \tag{9.57}$$

For reasons we explore later, we usually leave FIR such transfer-functions in this form, but we can convert it to a more familiar form in positive powers of z just to see what is happening.

$$H(z) = \frac{z^3 + 2z^3 + 3z^1 + 4}{z^3} \tag{9.58}$$

We see 3 poles all at zero which are pretty benign and can never be unstable. The zeros can lie anywhere depending on the design. In this case the three zeros have no meaning, but do not effect stability in an open-loop system (or filter). Therefore we can say that FIR filters are always stable. This is a great property they have and are therefore used more frequently than IIR filters in DSP applications. They have the disadvantage that their length can be quite long to approximate an IIR impulse response. For example, for any IIR impulse response we could truncate it and approximate it to be FIR. This depends on how accurate the FIR response is as compared with its IIR counterpart. The other advantage that FIR filters have is that their phase can be linear. Having linear-phase means that they do not effect the shape of pulses and act a bit like a time-delay. Of course, the word time-delay rings alarm bells in control-systems since any excess negative phase-shift destabilises a closed-loop system. Besides, some control-systems have system models that are open-loop unstable and so their equivalent IIR impulse-response would be divergent when used in any design. This is probably the main reason that the main application of FIR filters are in open-loop DSP applications where high-order impulse-response filters are designed in such fields as software radio and general communications or speech-processing. Fortunately in control, we are usually content to use very low-order filters (IIR) and for high-order we simply cascade the low-order ones. A PID controller is a very simple filter in the DSP world and almost trivial to implement in software as we shall see later. Given that, it is still of some use to look at an example of a real FIR filter to see what advantages it can have.

9.10.1 Example, FIR Notch Filter

Suppose we have a resonance on an electro-mechanical system which occurs at 1 kHz and we want to try and notch it out with a notch-filter. This can be a risky move because you may not gain much (there being other resonance terms nearby which come into play). An infinite notch can be created by a pair of complex-conjugate zeros exactly on the unit-circle at the required frequency [16]. This is after all what a zero does, it attenuates a particular complex frequency. Let us put two zeros on the unit circle at $0 + j$ *and* $0 - j$. This will give us a z-transfer function of

$$G(z) = \frac{z^2 + 1}{z^2} = 1 + z^{-2} \tag{9.59}$$

This is an FIR filter with impulse-response $g_k = \{1, 0, 1\}$.

Figure 9.18 shows the z-plane placement of the complex zeros.

First let us plot the frequency-response. The MATLAB code is shown below and the frequency-response in Fig. 9.19.

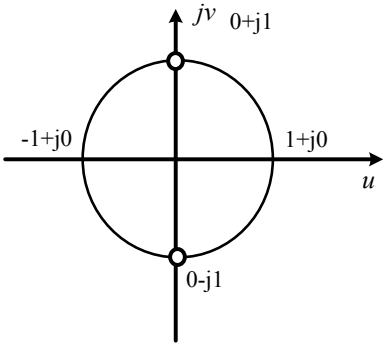


Fig. 9.18 z-plane placement of complex zeros

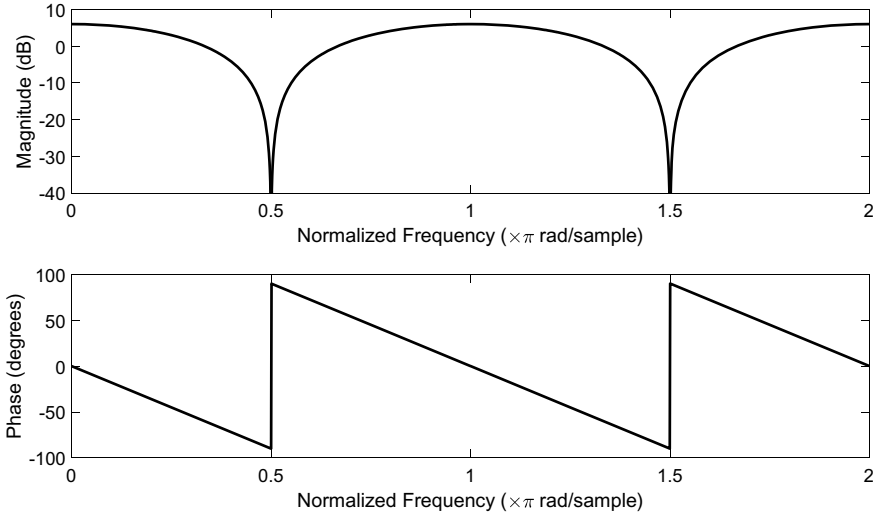


Fig. 9.19 Magnitude (dB) and phase-shift of digital FIR notch-filter

MATLAB Code 9.3

```
%dB Magnitude frequency response of second-order FIR notch-filter
b=[1 0 1];
a=1;
[h,w] = freqz(b,a,'whole',2001);
subplot(2,1,1)
plot(w/pi,20*log10(abs(h)))
ax = gca;
ax.YLim = [-40 10];
ax.XTick = 0:.5:2;
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Magnitude (dB)')
% Phase part of Notch-filter
subplot(2,1,2)
plot(w/pi,(angle(h)*(180/pi)))
ax = gca;
ax.YLim = [-100 100];
ax.XTick = 0:.5:2;
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Phase (degrees)')
```

The notch appears at 0.5 which corresponds to one quarter sampling frequency in terms of normalised frequency. To make the notch appear at 1 kHz would mean our sampling frequency should be 4 kHz. Any variation of this and the notch would be at the wrong frequency. The phase is perfectly linear, but to use such a filter in a control-loop would require the notch to be far away from the unity-gain crossover frequency where phase-margin is created. The phase shift at 1 kHz is $-90°$ so $\times 20$ fold down in frequency from this is 50 Hz. If this was to be the crossover frequency then the phase-shift at this frequency will be $-4.5°$ which will have an effect on stability. Besides using MATLAB, we can work out the frequency-response theoretically as follows. Substitute $z^{-1} = e^{-j\theta}$ into $\mathrm{G(z)} = 1 + \mathrm{z}^{-2}$ gives

$$\mathrm{G}\left(\mathrm{e}^{\mathrm{j}\theta}\right) = 1 + \mathrm{e}^{-\mathrm{j}2\theta} \tag{9.60}$$

$$\begin{aligned}\left|\mathrm{G}\left(\mathrm{e}^{\mathrm{j}\theta}\right)\right| &= \left|1 + \mathrm{e}^{-\mathrm{j}2\theta}\right| \\ &= \sqrt{(1 + \cos(2\theta))^2 + \sin^2(2\theta)} \\ &= \sqrt{2}\sqrt{1 + \cos(2\theta)}\end{aligned}$$

When the normalised frequency is at quarter-sampling we have $\theta = \frac{\pi}{2}$ (since π is half-sampling frequency). Then the magnitude at this frequency must be zero since $1 + \cos(2\theta)|_{\theta=\pi/2} = 0$.

The magnitude at dc is found when $\theta = 0$ which gives a gain of 2 or 6 dB.

The phase is found from (9.60) by expanding

$$G(e^{j\theta}) = 1 + \cos(2\theta) - j\sin(2\theta) \tag{9.61}$$

The phase expression is therefore

$$\arg(G(e^{j\theta})) = -\tan^{-1}\left(\frac{\sin(2\theta)}{1 + \cos(2\theta)}\right) \tag{9.62}$$

The phase-shift at dc is therefore zero and at $\theta = \frac{\pi}{2}$ becomes $-90°$.

9.11 A Note on Two-Sided Sequences and the z-Transform

Now that the basic one-sided z-transform is covered, it is best now to consider the two-sided transform. It is not used that often in the classical theory of digital-control but is important to know the distinction. The two-sided z-transform is defines as

$$F(z) = \sum_{k=-\infty}^{\infty} f_k z^{-k} \tag{9.63}$$

We see that the lower summation limit is minus infinity. This means that such a sum contains both negative and positive powers of z. For example

$$F(z) = \ldots + b_3 z^3 + b_2 z^2 + b_1 z^1 + a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + \ldots \tag{9.64}$$

is known as a Laurent-series. Terms which include negative powers of z (and a_0) are considered causal, whereas terms in positive powers of z are non-causal. Although we may never actually meet a non-causal system in person (because its step-response appears before an input is applied!), it is a mathematical concept that is used in optimal filtering and control (and even some FIR filter design) to represent what is known as two-sided sequences. We make them causal again by introducing a time-delay. So if (9.64) had only three terms in positive powers of z, by introducing a three-step delay z^{-3}, we could shift the impulse-response to the right and make it causal. We will of course have introduces a time-delay and at least for control-systems this is a dangerous thing. We can get away with it in open-loop DSP problems however at least since it only means to wait a little longer for the output. To delve a little more deeply consider the following first-order transfer-function

$$G(z) = \frac{1}{1 - az^{-1}}, |a| < 1 \tag{9.65}$$

By long-division or a Maclaurin power-series we can write (9.65) as

$$\frac{1}{1 - az^{-1}} = 1 + az^{-1} + a^2z^{-2} + \ldots \tag{9.66}$$

which converges with region of convergence $\left|az^{-1}\right| < 1$ or $|z| > |a|$. The region of convergence is greater than the pole and stretches to infinity in the region $|z| > |a|$. This is our ordinary or one-sided z-transform. However we can also write by long-division

$$\frac{1}{1 - az^{-1}} = \frac{z}{z - a} = -a^{-1}z - a^{-2}z^2 - a^{-3}z^3 - \ldots \tag{9.67}$$

which converges provided $\left|a^{-1}z\right| < 1$ or $|z| < |a|$. Therefore the region of convergence differs for both cases but the transfer-functions look the same in both cases. This is why it is important to state the region of convergence if we have problems with both right and left-handed sequences. For more in depth coverage the reader is referred to the classic text on DSP by Oppenheim and Schafer [30].

Chapter 10
Implementation of Digital Controllers

The previous chapter examines the basic theory of sampling and the z-transform. The following chapter will continue on this theme but consider the actual implementation of digital controllers. Analogue control requires circuitry to implement compensators or controllers. To design the hardware there are a number of methods available, but usually the controller is made from very basic compensators such as integrators and phase-lead terms and these are easy to realise in circuitry with standard operational amplifier circuits. We have seen that digital transfer-functions are based on z-transforms, but we require a way of implementing the z-transfer-function.

10.1 Difference Equations

Analogue systems are based on electronic hardware. Modelling is done using differential equations and transfer-functions, design using say the Bode method or State-Space, and then the compensator needs to be physically realised in circuitry. In sampled-data systems we need a method of realising a z-transfer function. It would be possible to do this in hardware using delay circuitry but this is uncommon nowadays and the job is performed fairly easily by using ordinary difference equations. If we return to the z-transform, it is related to the Laplace via $z = e^{sT_s}$ which for our methods here we prefer to write in terms of negative powers i.e. $z^{-1} = e^{-s\mathrm{T_s}}$. From our analogue work we immediately recognise the exponential term as a pure time-delay of one sampling interval. Therefore the negative power of z (called the backwords shift-operator) can be incorporated in an equation which consists mainly of delayed versions of input and output. For example, for a z-transfer-function written in the backwords form

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_10

$$G(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{10.1}$$

We assume the poles of the system lie within the unit-circle of the z-plane. That is to say that the two roots of the polynomial $1 + a_1 z^{-1} + a_2 z^{-2} = 0$ have magnitude less than unity. We of course solve for positive powers of z and not negative i.e. solve $z^2 + a_1 z + a_2 = 0$ for z and ensure the z values have magnitude less than unity. If we now use z as an operator we can write that for some signal say y_k when acted on by z^{-1}, it must shift the signal back in time by one sample interval and give

$$z^{-1}(y_k) = y_{k-1} \tag{10.2}$$

In fact (10.2) is a little abusive of notation since the k subscript represents time and the z term frequency. We therefore are illegally mixing time with frequency and this is like mixing s with time in continuous-time. Because of this, in some texts we replace z^{-1} with another variable (usually called q^{-1}, the backwards-shift operator). However, in the fields of engineering we usually relax this rule and use z^{-1} directly as an operator as in (10.2). Likewise we can write

$$z^{-2}(y_k) = y_{k-2} \tag{10.3}$$

Or even

$$z^{-n}(y_k) = y_{k-n} \tag{10.4}$$

It is also possible to use positive powers of z to shift forward in time instead of back by as many samples as necessary. It is more convenient to use the backwards-shift form. Taking (10.1) we write it as a transfer-function

$$y_k = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} u_k \tag{10.5}$$

In (10.5) ensure that the leading term of the denominator is unity. If it is not then we can divide every term by that value and convert it so it is in this format. Then by cross-multiplying we get

$$y_k\left(1 + a_1 z^{-1} + a_2 z^{-2}\right) = \left(b_0 + b_1 z^{-1}\right) u_k \tag{10.6}$$

Multiply out (10.6) and apply the operator technique

$$y_k + a_1 y_{k-1} + a_2 y_{k-2} = b_0 u_k + b_1 u_{k-1} \tag{10.7}$$

Move anything less than the current time sample k to the RHS of (10.7)

$$y_k = -a_1 y_{k-1} - a_2 y_{k-2} + b_0 u_k + b_1 u_{k-1} \tag{10.8}$$

This is termed the finite difference equation or just simply difference equation. To make use of it we substitute increasing integer values of k starting ta $k = 0$. The current output is then expressed in terms of past inputs and outputs. For this second-order difference-equation we need to store the past two values of output and one last value of input. We process sample by sample in digital-controllers. We do not take a huge buffer of data and then pass it through the difference-equation although this can be done in many DSP applications that use FIR filters. To be able to process such equations we require a microprocessor (or less commonly field-programmable gate-array FPGA) that executes an endless loop *at the sample-interval*. In other-words, if a particular controller has been designed for a sample-interval T_s, then this same time-interval must be used to process one loop of the difference-equation. If it is processed faster or slower then we have changed the sampling-interval and the difference-equation will give the wrong output since its z-transfer-function will have changed from the proposed one. For some system with n poles and m zeros we can write a more general form of (10.8)

$$y_k = -a_1 y_{k-1} - a_2 y_{k-2} \ldots - a_n y_{k-n} + b_0 u_k + b_1 u_{k-1} + \ldots b_m u_{k-m} \tag{10.9}$$

10.2 Pseudo-code for Implementing Difference-Equations

Given that we now have the method of implementation of a z-transfer-function, how do we write the software for the difference-equation? We require a processor with a real-time operating system (RTOS) that provides precise timing. Personal-computers (PCs) are not suitable for real-time control since the operating system is a general operating system which is designed for a whole range of applications from shifting files about to accessing the internet and house-keeping work. We need an operating system which is accurately controlled and dedicated to the task we set it to do and no interruptions except when requested to get data from the outside-world. Many microprocessors do not have a RTOS as such but just endlessly execute the code you present to them with precise timing within a margin of error. Such processors are termed *deterministic*. A computer that cannot guarantee accurate timing is termed *non-deterministic*. The amount of time that the loop time varies from each execution to the next is known as *jitter*. A good RTOS will have minimal jitter. The most common language that is used at the time of writing is c-language, but this is not a book on programming methods and so we will use our own language which we term Pseudo-code which has been made readable in such a way so it can be converted into any language. The language doesn't really matter, the accuracy of the timing of the real-time loop is important. Take the first example from Sect. 10.1. The difference equation is

$$y_k = -a_1 y_{k-1} - a_2 y_{k-2} + b_0 u_k + b_1 u_{k-1}$$

We require a loop that is accurately timed to the sample-interval and which must loop forever. In the code // represents a comment. Hence

```
// Do all definitions before the real-time loop
//define a1,a2,b0,b1 from design calculations of z transfer-function
While Loop Forever:
// curly brackets denote the beginning and end of infinite loop
{
// store past value of input
u1=u0
// read current input
Read u0
//shuffle past two values of output
y2=y1
y1=y0
//calculate the new output and send it to the analogue world.
y0=-a1*y1-a2*y2+b0*u0+b1*u1
Output y0
}
```

We have labelled a past value of output or input with the number next to it. Hence y1 is the past value of output (a sampling-interval before) and y2 is the value of output one sampling-interval before y1. Similarly, u1 is the input delayed by one-sampling interval. What we have done is to put our z-transfer function into software just as in analogue systems we implement in circuitry.

10.3 Converting from s-Domain to z-Domain

We have seen how we can take any z transfer-function and convert it to its difference equation and then into software. But how do we get the z-transfer function in the first-place? We always start with an analogue model, since the real-world is analogue or continuous-time. We therefore can also design an analogue controller but convert it into a digital controller instead of analogue hardware. This is our

preferred method of design though there are other methods such as converting the analogue system model to discrete and then designing a discrete-controller entirely in the z-domain. If we work in discrete-time too much we tend to lose track of important parameters such as the bandwidth of a system. Often it can become a mathematical exercise and we can lose control of the problem in hand. There are a number of ways to convert from continuous-time s-domain transfer functions to discrete-time z transfer-functions. We start with the fundamental equation that relates s and z that we have used several times before.

$$z = e^{sT_s} \tag{10.10}$$

There is a common power-series expansion for $f(x) = e^x$ which comes from the Maclaurin series. We can write

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \ldots \tag{10.11}$$

where the differentials are with respect to x.We get $f(0) = 1, f'(0) = 1$ and so to a first-order approximation we have our first approximation:

$$\begin{aligned} z &= e^{sT_s} \\ &\approx 1 + sT_s \end{aligned}$$

Re-arranging, we get the approximation

$$s \approx \frac{z-1}{T_s} \tag{10.12}$$

We now have an approximate fit between the s and z-domain instead of the irrational exponential term which we cannot use. However, there are other possibilities. We can also write

$$\begin{aligned} z &= \frac{1}{e^{-sT_s}} \\ &\approx \frac{1}{1 - sT_s} \end{aligned}$$

Re-arranging we get a second approximation

$$s \approx \frac{z-1}{zT_s} \tag{10.13}$$

This doesn't look too different from (10.12), but there are subtle differences when we do some analysis.

Finally we can write

$$\begin{aligned} z &= e^{sT_s} \\ &= \frac{e^{sT_s/2}}{e^{-sT_s/2}} \\ &\approx \frac{1 + sT_s/2}{1 - sT_s/2} \end{aligned}$$

Re-arranging for s we get a third approximation

$$s \approx \frac{2}{T_s}\left(\frac{z-1}{z+1}\right) \tag{10.14}$$

To be a good match, the left-half of the s-plane must map into the unit-circle on the z-plane.

Begin with our first approximation, which we now write as an equality $s = \frac{z-1}{T_s}$. We have $z = 1 + sT_s$ and substitute $s = \sigma + j\omega$. This results in

$z = 1 + \sigma T_s + j\omega T_s$. First thing we notice is that the imaginary axis in s occurs when $\sigma = 0$ and this corresponds to when $z = 1 + j\omega T_s$. This is a vertical line at $z = 1$. Likewise for values of $\sigma < 0$, this corresponds to anything left of this line on the z-plane. See Fig. 10.1.

Clearly this is not a good mapping to use since a stable continuous-time system could map onto an unstable region of the z-plane outside of the unit circle. We

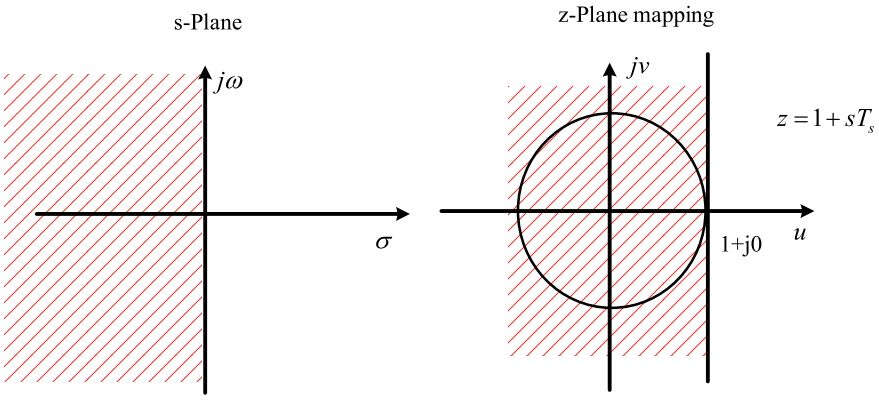


Fig. 10.1 Mapping from s to z for $s = \frac{z-1}{T_s}$

might get lucky of course but we are best to avoid such an approach and try the second approximation. For the second approximation we had $z = \frac{1}{1-sT_s}$. We can write this as

$$\begin{aligned} \mathrm{z} &= \frac{1}{1-\mathrm{sT_s}} \\ &= \frac{1}{2} + \frac{1}{2}\left(\frac{1+\mathrm{sT_s}}{1-\mathrm{sT_s}}\right) \end{aligned}$$

Now when $\sigma = 0$ we have

$$\left|\mathrm{z} - \frac{1}{2}\right| = \left|\frac{1}{2}\left(\frac{1+\mathrm{j}\omega\mathrm{T_s}}{1-\mathrm{j}\omega\mathrm{T_s}}\right)\right| \tag{10.15}$$

And since $\left|\frac{1}{2}\left(\frac{1+j\omega T_s}{1-j\omega T_s}\right)\right| = 0.5$, this means that the imaginary axis in the s-domain maps onto a circle in the z-domain, but unfortunately not the one we want. It is a circle centred at 0.5 + j0 radius 0.5. See Fig. 10.2.

This mapping will always give a stable discrete-system, but not necessarily an accurate representation of its continuous-time counterpart. If the sampling interval is small enough it may give rise to near-accurate models. This leads us to our third approximation $s = \frac{2}{T_s}\left(\frac{z-1}{z+1}\right)$. Re-arrange and we have

$$z = \frac{2+\mathrm{sT_s}}{2-\mathrm{sT_s}} \tag{10.16}$$

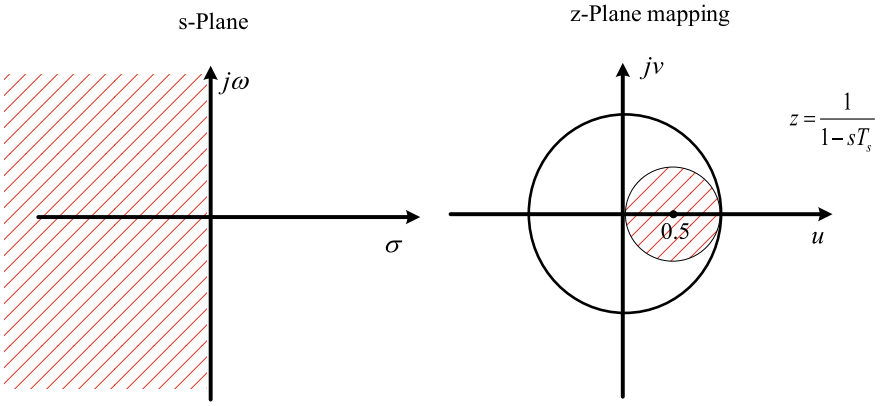


Fig. 10.2 $\mathrm{s} = \frac{\mathrm{Z}-1}{\mathrm{ZT_s}}$

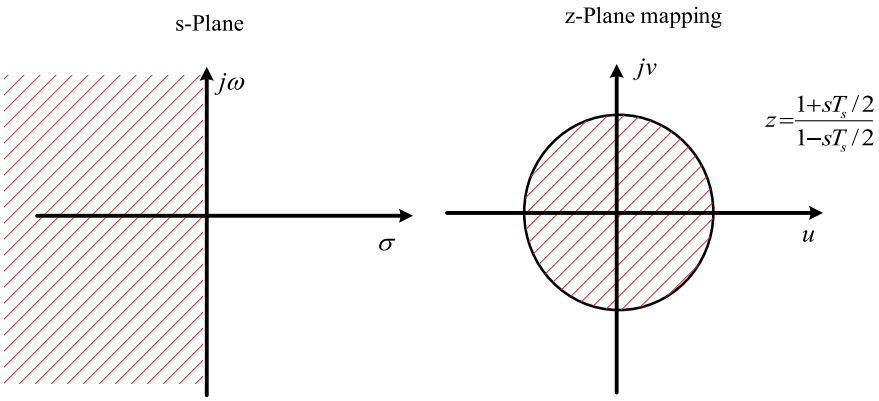


Fig. 10.3 $s = \frac{2}{T_s}\left(\frac{z-1}{z+1}\right)$

The imaginary axis on the s-plane must map to a circle of magnitude unity since when $\sigma = 0$

$$|z| = \left|\frac{2 + \mathrm{j}\omega \mathrm{T_s}}{2 - \mathrm{j}\omega \mathrm{T_s}}\right| = 1 \tag{10.17}$$

Similarly we can show that the left-hand plane maps directly into the unit-circle in the z-domain. See Fig. 10.3.

Finally we have a transformation which will map the left-half plane to the unit-circle. This transformation is also known as *Tustin's method* or the *Bilinear transform*.

10.4 Analysis of the Bilinear Transform

Now we have found the best transformation in the form of $s = \frac{2}{T_s}\left(\frac{z-1}{z+1}\right)$, we further analyse the transform as follows. We examine the frequency axis only and the relationship between continuous-time frequency and discrete-time frequency. For simplicity we normalise frequencies as before but define $\theta_c = \omega T_s$ as ideal normalised frequency and θ_d as the discrete-time actual counterpart. This latter frequency comes from $z = e^{j\theta_d}$. Of course, both frequencies need to be the same for an accurate representation.

With the real-part of s equal to zero, $\sigma = 0$, we have from (10.14)

$$\begin{aligned} j\omega T_s &= 2\left(\frac{e^{j\theta_d} - 1}{e^{j\theta_d} + 1}\right) \\ &= j\theta_c \end{aligned}$$

Write the above as

$$\begin{aligned} j\theta_c &= 2\left(\frac{1 - e^{-j\theta_d}}{1 + e^{-j\theta_d}}\right) \\ &= 2\left(\frac{e^{-j\theta_d/2}\left(e^{j\theta_d/2} - e^{-j\theta_d/2}\right)}{e^{-j\theta_d/2}\left(e^{j\theta_d/2} + e^{-j\theta_d/2}\right)}\right) \\ &= 2\left(\frac{\left(e^{j\theta_d/2} - e^{-j\theta_d/2}\right)}{\left(e^{j\theta_d/2} + e^{-j\theta_d/2}\right)}\right) \end{aligned}$$

But $\sin(\theta_d/2) = \left(e^{j\theta_d/2} - e^{-j\theta_d/2}\right)/2j$, $\cos(\theta_d/2) = \left(e^{j\theta_d/2} + e^{-j\theta_d/2}\right)/2$. Therefore

$$\begin{aligned} j\theta_c &= 2j\left(\frac{\left(e^{j\theta_d/2} - e^{-j\theta_d/2}\right)/2j}{\left(e^{j\theta_d/2} + e^{-j\theta_d/2}\right)/2}\right) \\ &= 2j\frac{\sin(\theta_d/2)}{\cos(\theta_d/2)} \\ &= 2j\tan(\theta_d/2) \end{aligned}$$

Now the j operators cancel and we get

$$\theta_c = 2\tan(\theta_d/2) \tag{10.18}$$

Or in terms of digital normalised frequency

$$\theta_d = 2\tan^{-1}(\theta_c/2) \tag{10.19}$$

For small angles $\tan(\theta_d/2) \approx \theta_d/2$ and it becomes clear that the frequencies for continuous and discrete-time are the same. When the frequency increases however we have a tan term to contend with. We can plot a graph comparing the continuous-time with the discrete-time frequency and we see from Fig. 10.4 that the two do not correspond exactly.

The higher is the frequency, the worse is the match and usually for a good match we can need a high sampling frequency compared with the frequency of interest. We choose a factor of 10 because we sample ten times higher in any case for sampled systems with feedback. In DSP applications which are open-loop the sampling rate can be relaxed and 3 to 4 times the frequency of interest can be

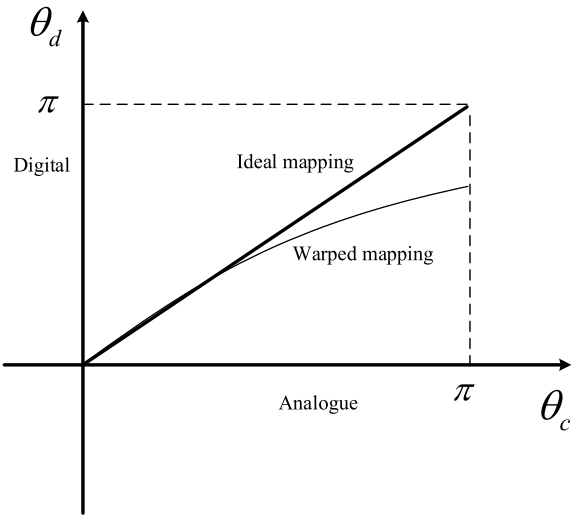


Fig. 10.4 Frequency warping effect of Bilinear-transform

assumed. However, for these lower sampling rates the frequencies get warped. A clever way around this is to pre-warp the frequencies using our derived formula and when it is converted to digital via the transform it gets warped back to the correct frequency. We rarely use this approach in feedback systems, preferring to use a 10 times sampling rate at least.

10.5 Examples: Use of Bilinear Transform

We consider some examples as follows.

10.5.1 Example 1: First-Order System

Consider a first-order system described by

$$G(s) = \frac{1}{1 + sT} \tag{10.20}$$

To convert (10.20) to its z-transform equivalent we use the Bilinear transform and substitute

$s = \frac{2}{T_s}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)$ into (10.20). Note we use the backwards shift version just for convenience.

$$G(z) = \frac{1}{1 + \frac{2T}{T_s}\left(\frac{1-Z^{-1}}{1+Z^{-1}}\right)} \tag{10.21}$$

Re-arranging

$$G(z) = \frac{T_s(1+z^{-1})}{T_s(1+z^{-1}) + 2T(1-z^{-1})} \tag{10.22}$$

Collecting terms of z

$$G(z) = \frac{T_s(1+z^{-1})}{(T_s+2T) + (T_s-2T)z^{-1}}$$

$$G(z) = \frac{T_s}{(T_s+2T)} \frac{(1+z^{-1})}{\left(1 + \frac{(T_s-2T)z^{-1}}{(T_s+2T)}\right)} \tag{10.22}$$

By defining $K = \frac{T_s}{(T_s+2T)}$ and $a = \frac{(T_s-2T)}{(T_s+2T)}$ in (10.22) we can simplify the expression down to

$$G(z) = K\frac{(1+z^{-1})}{(1+az^{-1})} \tag{10.23}$$

The reason we used the backwards-shift method for z rather than forwards is that the last step where we calculate the difference equation is easier. Leaving a "one" term and writing in backwards i.e. negative powers of z gives us an output which is at time k rather than some future time say $k+1$ in this case. To find the difference-equation

$$y_k = K\frac{(1+z^{-1})}{(1+az^{-1})}u_k \tag{10.24}$$

Cross-multiplying and using the backwards-shift operator leaves us

$$y_k(1+az^{-1}) = K(1+z^{-1})u_k$$
$$y_k = -ay_{k-1} + Ku_k + Ku_{k-1}$$

If we had used the forwards-shift operator for z we would have ended up with $y_{k+1} = -ay_k + Ku_{k+1} + Ku_k$, which is still perfectly valid, but when we do the software it feels better to have the current output at time k rather than a step ahead.

To put some numbers for realism, suppose we need a low-pass filter (first-order) with cut-off frequency $f_c = 1\,\text{kHz}$. Then $T = 159\,\mu\text{s}$ time-constant for the filter. (cut-off frequency is $f_c = \frac{1}{2\pi T}$ so $T = \frac{1}{2\pi f_c}$). We need to sample at least ten times higher than this frequency to use the Bilinear-transform. We therefore sample at $f_s = 10\,\text{kHz} = 1/T_s$ making $T_s = 1/10000 = 100\,\mu\text{s}$. This must be as accurate as possible for the timed-loop to execute in the software. We then find the constants $K = \frac{T_s}{(T_s+2T)} = 0.239$, $a = \frac{(T_s-2T)}{(T_s+2T)} = -0.521$. (note the pole of the digital filter is at 0.521). Our digital filter is $G(z) = 0.239\frac{\left(1+z^{-1}\right)}{(1-0.521z^{-1})}$. Another good check to make is to ensure that the dc gain of the original analogue filter is the same for the digital version. For $G(s) = \frac{1}{1+sT}$, then $G(0) = 1$. In the z-domain dc occurs when $z = 1$. Therefore $G(z) = 0.239\frac{\left(1+z^{-1}\right)}{(1-0.521z^{-1})}$ and $G(1) = 0.239\frac{(1+1)}{(1-0.521)} \approx 1$.

The Pseudo-code for implementation becomes.

```
// First-order low-pass filter 1/(1+sT)

//define K and a from design calculations of z transfer-function in terms of Ts sampling interval

// This loop must execute at Ts seconds

While Loop Forever:

// curly brackets denote the beginning and end of infinite loop

{

// store past value of input

u1=u0

// read current input

Read u0

//shuffle past value of output

y1=y0

//calculate the new output and send it to the analogue world.

y0=-a1*y1+K*u0+K*u1

Output y0

}
```

10.5.2 Integral Compensator

Suppose we have an integral compensator that we must implement (this is an integrator at low-frequencies i.e. −20 dB/decade and then goes flat at a frequency $\frac{1}{2\pi T_I}$ Hz in the dB Bode response)

$$\mathrm{G(s)} = \frac{1 + \mathrm{sT_I}}{\mathrm{s}} \tag{10.25}$$

as a digital filter (compensator).

Now substitute $s = \frac{2}{T_s}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)$ into (10.25)

$$\mathrm{G(z)} = \frac{1 + \frac{2\mathrm{T_I}}{\mathrm{T_s}}\left(\frac{1-\mathrm{Z}^{-1}}{1+\mathrm{Z}^{-1}}\right)}{\frac{2}{\mathrm{T_s}}\left(\frac{1-\mathrm{Z}^{-1}}{1+\mathrm{Z}^{-1}}\right)} \tag{10.26}$$

After a little algebra we get

$$\mathrm{G(z)} = \frac{\mathrm{c} + \mathrm{dz}^{-1}}{1 - \mathrm{z}^{-1}} \tag{10.27}$$

where $c = T_s/2 + T_I, d = T_s/2 - T_I$.

The difference equation is found from

$$\mathrm{y_k}\left(1 - \mathrm{z}^{-1}\right) = \left(\mathrm{c} + \mathrm{dz}^{-1}\right)\mathrm{u_k}$$

to be

$$\mathrm{y_k} = \mathrm{y}_{k-1} + \mathrm{cu_k} + \mathrm{du}_{k-1} \tag{10.28}$$

The familiar $y_k = y_{k-1} + ..$ expression is always the sign of a pure integrator. Many mathematical algorithms have them since an update (new value = old value + input) have this classic form.

The Pseudo-code for implementation becomes.

```
// Integrator compensator at low frequencies (1+sT1)/s

//define c and d from design calculations of z transfer-function in terms of Ts sampling interval

// This loop must execute at Ts seconds

While Loop Forever:

// curly brackets denote the beginning and end of infinite loop

{

// store past value of input

u1=u0

// read current input

Read u0

//shuffle past value of output

y1=y0

//calculate the new output and send it to the analogue world.

y0= y1+c*u0+d*u1

Output y0

}
```

10.5.3 Phase-Lead (Advance) Compensator

Phase-lead compensator.

$$G(s) = \frac{1 + sT_1}{1 + sT_2}, T_1 > T_2 \tag{10.29}$$

Now substitute $s = \frac{2}{T_s}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)$ into (10.29)

$$G(z) = \left[\frac{1 + sT_1}{1 + sT_2}\right]_{s=\frac{2}{T_s}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)} \tag{10.30}$$

$$G(z) = \left[\frac{T_s + 2T_1}{T_s + 2T_2}\right]\left[\frac{1 + z^{-1}\left(\frac{T_s - 2T_1}{T_s + 2T_1}\right)}{1 + z^{-1}\left(\frac{T_s - 2T_2}{T_s + 2T_2}\right)}\right] = K\left[\frac{1 + bz^{-1}}{1 + az^{-1}}\right] \tag{10.31}$$

where

$K = \left[\frac{T_s + 2T_1}{T_s + 2T_2}\right]$, $b = \left(\frac{T_s - 2T_1}{T_s + 2T_1}\right)$, $a = \left(\frac{T_s - 2T_2}{T_s + 2T_2}\right)$

The difference-equation is

$$y_k = -ay_{k-1} + Ku_k + Kbu_{k-1} \tag{10.32}$$

```
// Phase=lead compensator (1+sT1)/(1+sT2),T1>T2
//define a and b and K from design calculations of z transfer-function in terms of Ts sampling interval
// This loop must execute at Ts seconds
While Loop Forever:
// curly brackets denote the beginning and end of infinite loop
{
// store past value of input
u1=u0
// read current input
Read u0
//shuffle past value of output
y1=y0
//calculate the new output and send it to the analogue world.
y0= -a*y1+K*u0+K*b*u1
Output y0
}
```

10.5.4 Digitally Cascading Compensators

Cascading compensators in software.

In analogue systems compensators are usually made from operational amplifiers, and so to cascade them together simply means to link the output of the first into the input of the second. We can apply the same rule in software. For example. Suppose we have the previous integrator at low-frequencies followed by the phase-lead compensator.

$$G(s) = \left(\frac{1 + sT_I}{s}\right)\left(\frac{1 + sT_I}{1 + sT_2}\right), T_I > T_1 > T_2 \tag{10.33}$$

We already know the individual z-transfer-functions and so we can cascade them as follows

$$G(z) = K\left(\frac{c + dz^{-1}}{1 - z^{-1}}\right)\left(\left[\frac{1 + bz^{-1}}{1 + az^{-1}}\right]\right) \tag{10.34}$$

In software this becomes

```
// Integrator compensator at low frequencies (1+sT1)/s and phase-lead
// in cascade(1+sT1)/(1+sT2)
//define a, b, c ,d and K from design calculations of z transfer-function in terms of Ts sampling interval
// This loop must execute at Ts seconds
While Loop Forever:
//Integrator part first
// curly brackets denote the beginning and end of infinite loop
{
// store past value of input
u1=u0
// read current input
Read u0
//shuffle past value of output
y1=y0
//calculate the new output.
y0= y1+c*u0+d*u1
// y0 becomes input to phase-lead. Use it in next section

//Phase-lead part
//shuffle output
z1=z0
//calculate the new output and send it to the analogue world.
//note that the input to the phase-lead is the output from previous stage
z0= -a*z1+K*y0+K*b*y1
Output z0
}
```

The code is fairly simple, whatever language is used to implement it since controllers using classical control-theory do not generally have more than cascaded first-order systems. However, the code must execute in less time than the timed-loop (the infinite loop). Therefore, for more complex algorithms in digital-control (or filtering), the more powerful the processor gives us more freedom to fit the code execution time within the boundary set by the sampling-interval.

10.6 The Link with Numerical Integration

There is a close link between the z-transform way of discretising continuous-time systems and classical methods of numerical integration. Integration is merely finding the area under a curve. Consider Fig. 10.5. There are three common methods shown here for finding area.

In the first two of these we use rectangles at regularly spaced intervals. By summing the areas under the rectangles we can find the area under the curve. It is best to write the area in a recursive form. This means finding the new area in terms of the old area. We can write for any of the rectangular methods

$$new\ area = old\ area + area\ under\ rectangle$$

This is moving left to right. For the first method (Forward Difference or Euler's Forward difference method) the area under the rectangle is $u_{kT_s}T_s$. For the second method (Euler's Backwards method) we have the area under the triangle as $u_{(k+1)T_s}T_s$. The only difference between the two is quite subtle and for the same curve we can think of one method of approximating by taking too much area versus taking too little, since the rectangle cannot fit in where there is a slope and there is always a residual left over. If the area at time k is y_{kT_s} and we ignore the dependence of the sampling interval for simplicity (hence we write y_k instead) then for the Euler's forward-difference method:

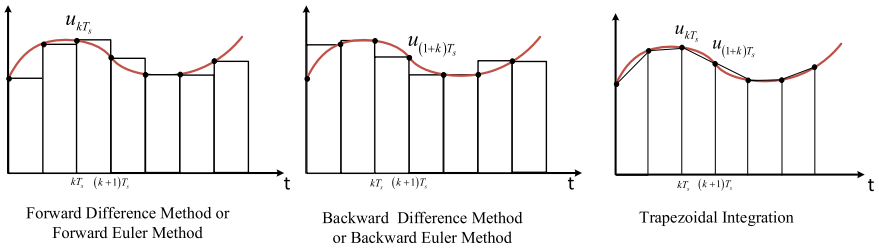


Fig. 10.5 Numerical integration methods

$$y_{k+1} = y_k + T_s u_k \tag{10.35}$$

and for the Euler's backwards-difference method:

$$y_{k+1} = y_k + T_s u_{k+1} \tag{10.36}$$

Now take z-transforms since (10.35) and (10.36) are both difference-equations. Equation (10.35) gives us

$$y(z) = \frac{T_s}{z-1} u_k \tag{10.37}$$

For (10.36) we get

$$y(z) = \frac{T_s z}{z-1} u_k \tag{10.38}$$

We can define integrators as these transfer-functions so for both cases

$$\frac{1}{s} \approx \frac{T_s}{z-1} \tag{10.39}$$

$$\frac{1}{s} \approx \frac{T_s z}{z-1} \tag{10.40}$$

Both of these we have already encountered when expanding $z = e^{sT_s}$. Referring back to Fig. 10.5, the rightmost method uses a Trapezium rather than a rectangle to approximate the area. We get the update for area as (area of a Trapezium is half the sum of the parallel sides times the perpendicular distance)

$$y_{k+1} = y_k + \frac{T_s}{2}(u_{k+1} + u_k) \tag{10.41}$$

This is of course Tustin's method or *Trapezoidal integration*. Of course (10.41) can be written as the relationship

$$\frac{1}{s} \approx \frac{T_s}{2}\left(\frac{z+1}{z-1}\right) \tag{10.42}$$

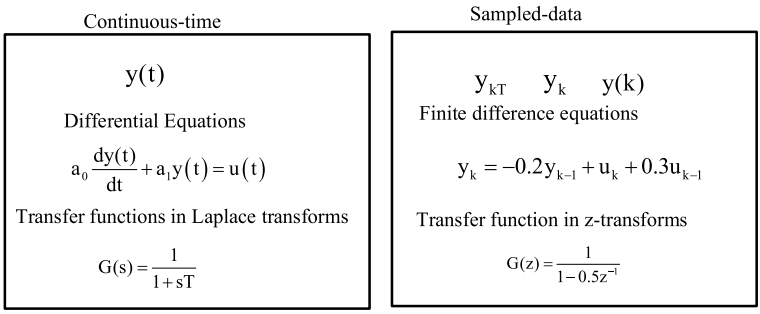


Fig. 10.6 Comparison of discrete-time and continuous-time

This is the most accurate of these three methods. There are more accurate methods of numerical integration such as Simpsons rule, but the added complexity and little gained by such an approach means it is never used for obtaining z-transfer functions from their Laplace equivalents. A comparison of discrete-time and continuous-time is shown in Fig. 10.6.

10.7 Discrete-Time PID Controllers

Theory tells us that Trapezoidal integration is the best, but for PID it is rarely used and instead simple rectangular integration is usually sufficient. The correct terminology for the discrete-time case should be proportional plus integral plus difference controller. This is shown in z-transfer-function format in Fig. 10.7. It is shown in backwards-shift notation for programming ease.

The sampling interval is normalised to unity since it only re-scales the individual PID gains. For example for the integrator we would have $\frac{K_I}{s} \approx \frac{K_I T_s}{1-z^{-1}}$ and $K_I T_s$ can then be absorbed together into a new constant. We note that the difference (differentiator) is also a Euler type, but whatever type we use, a pure differentiator is not good in a control-system as it amplifies high-frequencies and can cause trouble with mechanical resonances and noise. Therefore a low-pass digital filter (Eq. (10.23)) is usually put in cascade with the differentiator or for simplicity outside of the PID as shown in Fig. 10.8.

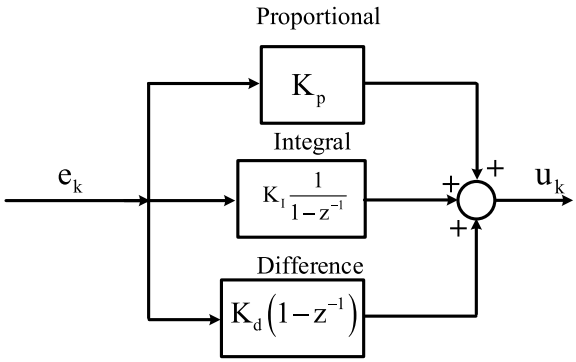


Fig. 10.7 Simplified discrete-time PID controller

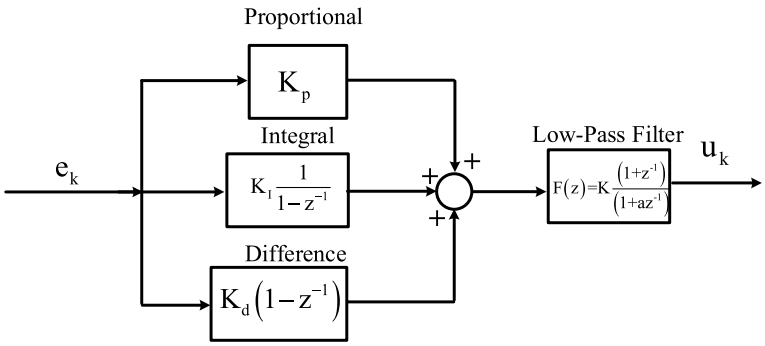


Fig. 10.8 Discrete-time PID with low-pass filter in cascade

The pseudo-code for the controller is easily written as the sum of the three terms.

```
// Discrete-time PID
// Do all definitions before the real-time loop
//define Kp,Ki and Kd (prop, integral and difference gain terms)
While Loop Forever:
// curly brackets denote the beginning and end of infinite loop
{
//read setpoint r:  either from analogue world or software defined
Read r
// store past value of error
e1=e0
//Read output of overall system (e.g. could be current speed or position of a motor from a
//transducer)
Read z0
// Create current error for control-system
e0=r-z0
//shuffle last value of integrator output
y1=y0
//calculate the three terms.
//Proportional
p=Kp*e0
//Difference term
d=Kd*(e0-e1)
//Integrator term ouput
y0=y1+Ki*e0
//sum the outputs
PID0=p+d+y0
Output PID0 to analogue world (or perhaps PWM for a motor drive)
}
```

To add the filter we need to add code after PID0 which takes PID0 as input and creates a digitally filtered output (see example 1 in Sect. 10.5). Tuning a digital PID is not so easily done as in the analogue case. For analogue we usually have three potentiometers, but here we have three gains and they are difficult to change in

real-time in an embedded system. We could for example have three temporary potentiometers which are read by three A/D convertors and the values converted to digital values and fed to this algorithm. Otherwise we may have to change values and re-compile each time. A digital keypad could also be used or really any method that allows real-time changing of numerical values within a loop. The rules of tuning are identical to that of an analogue controller. We start with a low gain and turn it up until it just begins to oscillate. Then we add the difference term which should stop it oscillating. Finally we add integral action to remove steady-state error. We then go back and change whatever gains are necessary for desired performance. We usually put the low-pass filter well out of the way at a higher frequency to attenuate any noise that gets through from the differentiator.

Chapter 11
Discrete-Time State-Space

This chapter derives the discrete-time version of state-space which was covered in some detail in Chap. 7. It will be shown that most of the already-known properties of state-space carry into the discrete-time case without any difficulties and moving from one to the other is relatively easy.

11.1 The Discrete-Time State-Space from the Continuous-Time Case

Recall that the continuous-time state-space description of a LTI system is given by

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \tag{11.1}$$

$$y(t) = \mathbf{C}\mathbf{x}(t) \tag{11.2}$$

For simplicity we consider SISO systems only without losing any generality as this is easily extended to the multivariable case.

Furthermore, we also know that the response of a system to arbitrary input is given in continuous-time as the convolution integral:

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0 + \int_{\tau=0}^{\tau=t} e^{\mathbf{A}(t-\tau)}\mathbf{B}u(\tau)d\,\tau \tag{11.3}$$

Now substitute $t = kT_s$ into (11.3)

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_11

$$\mathbf{x}(\mathrm{kT_s}) = \mathrm{e}^{\mathbf{A}\mathrm{kT_s}}\mathbf{x}_0 + \int_{\tau=0}^{\tau=\mathrm{kT_s}} \mathrm{e}^{\mathbf{A}(\mathrm{kT_s}-\tau)}\mathbf{B}\mathrm{u}(\tau)\mathrm{d}\,\tau \tag{11.4}$$

Now increment the discrete-time index by one and drop the dependence on the sampling interval for the state. This is just for clarity i.e. $\mathbf{x}(kT_s) = \mathbf{x}(k)$

$$\mathbf{x}(\mathrm{k}+1) = \mathrm{e}^{\mathbf{A}(\mathrm{k}+1)\mathrm{T_s}}\mathbf{x}_0 + \int_{\tau=0}^{\tau=(\mathrm{k}+1)\mathrm{T_s}} \mathrm{e}^{\mathbf{A}((\mathrm{k}+1)\mathrm{T_s}-\tau)}\mathbf{B}\mathrm{u}(\tau)\mathrm{d}\,\tau \tag{11.5}$$

Multiply (11.4) throughout by $e^{\mathbf{A}T_s}$, subtract from (11.5) and rearrange. We get

$$\mathbf{x}(\mathrm{k}+1) = \mathrm{e}^{\mathbf{A}\mathrm{T_s}}\mathbf{x}(\mathrm{k}) + \int_{\tau=\mathrm{kT_s}}^{\tau=(\mathrm{k}+1)\mathrm{T_s}} \mathrm{e}^{\mathbf{A}(\mathrm{kT_s}+\mathrm{T_s}-\tau)}\mathbf{B}\mathrm{u}(\tau)\mathrm{d}\,\tau \tag{11.6}$$

where we will define

$$\mathbf{x}(\mathrm{k}) = \mathrm{e}^{\mathbf{A}\mathrm{kT_s}}\mathbf{x}_0 \tag{11.7}$$

Substitute a new variable $\sigma = (k+1)T_s - \tau$. When $\sigma = 0, \tau = (k+1)T_s$. When $\sigma = T_s, \tau = kT_s$ and $d\sigma = -d\tau$. The RHS of (11.6) is

$$\begin{aligned} &-\int_{\mathrm{T_s}}^{0} \mathrm{e}^{\mathbf{A}(\mathrm{kT_s}+\mathrm{T_s}-\tau)}\mathbf{B}\mathrm{u}(\mathrm{kT_s})\mathrm{d}\,\sigma = \int_{0}^{\mathrm{T_s}} \mathrm{e}^{\mathbf{A}\sigma}\mathbf{B}\mathrm{u}(\mathrm{kT_s})\mathrm{d}\,\sigma \\ &= \int_{0}^{\mathrm{T_s}} \mathrm{e}^{\mathbf{A}\sigma}\mathbf{B}\mathrm{d}\,\sigma\,\mathrm{u}(\mathrm{kT_s}) \end{aligned} \tag{11.8}$$

Usually we write $u(kT_s) = u(k)$ since we know each step is dependent on the sampling interval anyway.

The discrete-time state-space is hence

$$\mathbf{x}(\mathrm{k}+1) = \mathrm{e}^{\mathbf{A}\mathrm{T_s}}\mathbf{x}(\mathrm{k}) + \int_{0}^{\mathrm{T_s}} \mathrm{e}^{\mathbf{A}\,\sigma}\mathbf{B}\mathrm{d}\,\sigma\,\mathrm{u}(\mathrm{k}) \tag{11.9}$$

Define

$$\mathbf{F} = e^{\mathbf{A}T_s} \tag{11.10}$$

$$\mathbf{G} = \int_0^{T_s} e^{\mathbf{A}\sigma}\mathbf{B}d\sigma \tag{11.11}$$

and we have

$$\mathbf{x}(k+1) = \mathbf{F}\mathbf{x}(k) + \mathbf{G}u(k) \tag{11.12}$$

For the output equation we define $\boldsymbol{H} = \boldsymbol{C}$ and get

$$y(k) = \mathbf{H}\mathbf{x}(k) \tag{11.13}$$

Figure 11.1 shows the discrete-time state-space block-diagram.
We can also write

$$y(k) = \mathbf{H}\mathbf{x}(k) + \mathbf{D}x(k)$$

where appropriate, when the numerator and denominator z transfer-functions have the same order.

A few other convenient ways of writing discrete-time state-space exist. For example with subscripts as in Fig. 11.1.

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}u_k, y_k = \mathbf{H}\mathbf{x}_k \tag{11.14}$$

Or shifted back in time by one sample

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{G}u_{k-1}, y_k = \mathbf{H}\mathbf{x}_k \tag{11.15}$$

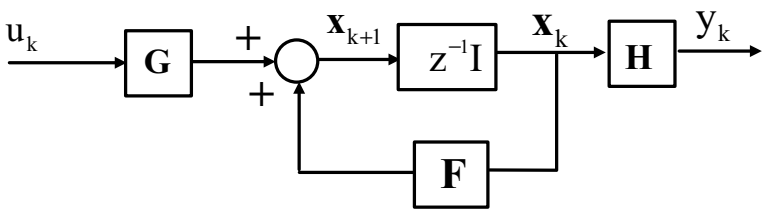


Fig. 11.1 Representation of discrete-time state-space

But how do we compute the discrete-time matrices given the continuous-time ones? There are a number of methods, the simplest of which is the power-series expansion method. We write

$$\begin{aligned}\mathbf{F} &= e^{\mathbf{A}T_s} = \mathbf{I} + \mathbf{A}T_s + \frac{1}{2!}\mathbf{A}^2T_s^2 + \frac{1}{3!}\mathbf{A}^3T_s^3 + \cdots \\ &= \mathbf{I} + \mathbf{A}T_s\boldsymbol{\Psi}\end{aligned} \tag{11.16}$$

where we define

$$\begin{aligned}\boldsymbol{\Psi} &= \mathbf{I} + \frac{1}{2!}\mathbf{A}T_s + \frac{1}{3!}\mathbf{A}^2T_s^2 + \cdots \\ &= \sum_{k=0}^{\infty}\frac{\mathbf{A}^kT_s^k}{(k+1)!}\end{aligned} \tag{11.17}$$

This can be computed to a finite number of terms to find the discrete-time $\boldsymbol{F}$ matrix. Depending on the system eigenvalues will dictate the number of terms for convergence. Usually about 12 terms is enough for most problems, but with today's computation power it presents no difficulties.

We can likewise integrate (11.1) term by term and show that

$$\mathbf{G} = T_s\boldsymbol{\Psi}\mathbf{B} \tag{11.18}$$

A long time ago these discrete-time matrices were known as the Greek letters Phi $\boldsymbol{\Phi}$ and Del Δ, but $\boldsymbol{F}$ and $\boldsymbol{G}$ are more frequently used today.

11.1.1 Example. Newton's Law in Discrete State-Space

Consider a mass M with a force $f(t)$ applied. We have for displacement $x(t)$ and no friction:

$$f(t) = M\frac{d^2x(t)}{dt^2} \tag{11.19}$$

Define states as position and velocity and we arrive at

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{M}\end{bmatrix}f(t) \tag{11.20}$$

We have $\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} 0 \\ \frac{1}{\mathrm{M}} \end{bmatrix}$. We are fortunate that this example can be calculated manually since when the A matrix is squared or raised to high powers, the higher powers are zero. Hence from (11.17)

$$\boldsymbol{\Psi} = \mathbf{I} + \frac{1}{2!}\mathbf{A}T + \frac{1}{3!}\mathbf{A}^2T^2 + \cdots = \begin{bmatrix} 1 & T/2 \\ 0 & 1 \end{bmatrix} \tag{11.21}$$

$$\begin{aligned} \mathbf{F} &= \mathrm{e}^{\mathbf{A}\mathrm{T_s}} = \mathbf{I} + \mathbf{A}\mathrm{T_s}\boldsymbol{\Psi} \\ &= \begin{bmatrix} 1 & \mathrm{T} \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{11.22}$$

$$\begin{aligned} \mathbf{G} &= \mathrm{T_s}\boldsymbol{\Psi}\mathrm{B} \\ &= \begin{bmatrix} \mathrm{T_s^2}/2 \\ \mathrm{T_s} \end{bmatrix} \frac{1}{\mathrm{M}} \end{aligned} \tag{11.23}$$

The discrete-time state-space equations become

$$\begin{bmatrix} \mathrm{x_1(k+1)} \\ \mathrm{x_2(k+1)} \end{bmatrix} = \begin{bmatrix} 1 & \mathrm{T}_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{x_1(k)} \\ \mathrm{x_2(k)} \end{bmatrix} + \begin{bmatrix} \mathrm{T}_s^2/2 \\ \mathrm{T}_s \end{bmatrix} \frac{1}{\mathrm{M}} \mathrm{f(k)} \tag{11.24}$$

We can use MATLAB to convert numerically. To show it working, suppose M = 1, $\mathrm{T}_s = 1$.

MATLAB Code 11.1

```
%Matlab code to convert continuous-time to discrete-time state-space
%Define A,B,C,D first in MATLAB and sampling interval T.
A=[0 1;0 0];
B=[0 1]';
C=[1 0];
D=0;
T=1;
% Then find continuous-time state-space description in terms of those matrices
sys=ss(A,B,C,D)
% Convert to discrete is one line command
sysd=c2d(sys,T)
```

The program returns

Continuous-time state-space model.

```
sysd =

 A =
      x1  x2
  x1   1   1
  x2   0   1

 B =
       u1
  x1  0.5
  x2    1

 C =
      x1  x2
  y1   1   0

 D =
      u1
  y1   0

Sample time: 1 seconds
Discrete-time state-space model.
```

11.2 State-Space to Transfer-Function

Much of the discrete-time theory is very similar in form to continuous-time except for regions of stability. To convert from discrete-time state-space we take z-transforms of

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{F}\mathbf{x}_k + \mathbf{G}u_k \\ y_k &= \mathbf{H}\mathbf{x}_k \end{aligned} \tag{11.25}$$

We get

$$z\mathbf{x}(z) = \mathbf{F}\mathbf{x}(z) + \mathbf{G}u(z)$$

Collecting terms

$$(zI - \mathbf{F})\mathbf{x}(z) = \mathbf{G}u(z) \tag{11.26}$$

Then

$$\mathbf{x}(z) = (zI - \mathbf{F})^{-1}\mathbf{G}u(z) \tag{11.27}$$

and

$$y(z) = \mathbf{H}(zI - \mathbf{F})^{-1}\mathbf{G}u(z) \tag{11.28}$$

The transfer-function is then

$$E(z) = \mathbf{H}(zI - \mathbf{F})^{-1}\mathbf{G} \tag{11.29}$$

Here the z-transfer function is written as being scalar, but it could be a transfer-function matrix or vector of transfer-functions, depending on the dimensions of the various matrices.

11.2.1 Example. Second-Order System. Discrete State-Space to Transfer-Function

Consider the second-order state-space system

$$\begin{aligned} \mathbf{F} &= \begin{bmatrix} 0 & 1 \\ -0.5 & -1 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \\ \mathbf{H} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \end{aligned}$$

$$(zI - \mathbf{F})^{-1} = \begin{bmatrix} z & -1 \\ 0.5 & z+1 \end{bmatrix}^{-1}$$
$$= \frac{1}{z^2 + z + 0.5} \begin{bmatrix} z+1 & 1 \\ -0.5 & z \end{bmatrix}$$

$$E(z) = \begin{bmatrix} 1 & 0 \end{bmatrix} \frac{1}{z^2 + z + 0.5} \begin{bmatrix} z+1 & 1 \\ -0.5 & z \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\frac{1}{z^2 + z + 0.5}$$

The stability of the state-space approach is determined by the eigenvalues of the determinant

$$|zI - \mathbf{F}| = 0 \tag{11.30}$$

The eigenvalues of $\boldsymbol{F}$ are the roots of the polynomial formed by (11.30) (and the poles), and must lie within the unit-circle of the z-plane.

11.3 Transfer-Function to State-Space

Given a transfer-function which is all-pole.

$$E(z) = \frac{1}{z^n + a_1 z^{n-1} + a_2 z^{n-2} + \ldots + a_n} \tag{11.31}$$

we can write this as a difference-equation

$$y_{k+n} = -a_1 y_{k+n-1} - a_2 y_{k+n-2} - \ldots - a_n y_k + u_k \tag{11.32}$$

Define $\mathbf{x_k} = \begin{bmatrix} x_k^1 & x_k^2 & x_k^3 & . & . & x_k^{n-1} & x_k^n \end{bmatrix}^T$. Just as with the continuous-time case, there are an infinite number of state-space descriptions. Let us pick one common one in phase-variable canonical form. (note, do not confuse the super-script notation here with powers of variables, they are merely labels)

$$\begin{aligned} x_k^1 &= y_k \\ x_k^2 &= y_{k+1} \\ x_k^3 &= y_{k+2} \\ &. \\ &. \\ x_k^n &= y_{k+n-1} \end{aligned} \tag{11.33}$$

etc.

It follows from (11.33) that

$$x_{k+1}^1 = y_{k+1} = x_k^2$$
$$x_{k+1}^2 = y_{k+2} = x_k^3$$

up to

$$x_{k+1}^n = y_{k+n} = -a_1 y_{k+n-1} - a_2 y_{k+n-2} - \ldots - a_n y_k + u_k \tag{11.34}$$

The above follows from (11.32). We can now write

$$x_{k+1}^n = -a_1 x_k^n - a_2 x_k^{n-1} - \ldots - a_n x_k^1 + u_k$$

In state-space format this becomes

$$\begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \\ x_{k+1}^3 \\ . \\ . \\ x_{k+1}^{n-1} \\ x_{k+1}^n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & . & . & 0 & 0 \\ 0 & 0 & 1 & . & . & 0 & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & . & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & . & . & -a_2 & -a_1 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \\ x_k^3 \\ . \\ . \\ x_k^{n-1} \\ x_k^n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 0 \\ 1 \end{bmatrix} u_k \tag{11.35}$$

$$\begin{aligned} y_k &= x_k^1 \\ &= \begin{bmatrix} 1 & 0 & 0 & . & . & . & 0 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \\ x_k^3 \\ . \\ . \\ x_k^{n-1} \\ x_k^n \end{bmatrix} \end{aligned} \tag{11.36}$$

That is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & . & . & . & 0 \end{bmatrix}$$

It is interesting to see that the state-vector is just shifted values of the system output. In this case each consecutive state is shifted one step in time forwards.

$$\mathbf{x}_k = \begin{bmatrix} y_k \\ y_{k+1} \\ y_{k+2} \\ . \\ . \\ y_{k+n-2} \\ y_{k+n-1} \end{bmatrix}$$

Then (11.35) is also

$$\begin{bmatrix} y_{k+1} \\ y_{k+2} \\ y_{k+3} \\ . \\ . \\ y_{k+n-1} \\ y_{k+n} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & . & . & 0 & 0 \\ 0 & 0 & 1 & . & . & 0 & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & . & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & . & . & -a_2 & -a_1 \end{bmatrix} \begin{bmatrix} y_k \\ y_{k+1} \\ y_{k+2} \\ . \\ . \\ y_{k+n-2} \\ y_{k+n-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 0 \\ 1 \end{bmatrix} u_k \tag{11.37}$$

An alternative description can be found by analogy to continuous-time canonical forms.

For example, we can have instead,

$$\mathbf{F} = \begin{bmatrix} -a_1 & -a_2 & -a_3 & . & . & -a_{n-1} & -a_n \\ 1 & 0 & 0 & . & . & 0 & 0 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & . & . & 1 & 0 & 0 \\ 0 & 0 & . & . & . & 1 & 0 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ . \\ . \\ . \\ 0 \end{bmatrix},$$

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & . & . & . & 1 \end{bmatrix}$$

11.3.1 Example. Second-Order System to Discrete State-Space

From first principles convert the following z-transfer function to discrete-time state-space.

$$E(z) = \frac{b_1 z + b_2}{z^2 + a_1 z + a_2} \tag{11.38}$$

Split the numerator and denominator and introduce an intermediate variable x_k in-between. Thus

$$x_k = \frac{1}{z^2 + a_1 z + a_2} u_k$$
$$y_k = (b_1 z + b_2) x_k$$

The first of these is already in a familiar format but we write the difference equation

$$x_{k+2} = -a_1 x_{k+1} - a_2 x_k + u_k \tag{11.39}$$

Define

$$x_k^1 = x_k$$
$$x_k^2 = x_{k+1}$$

Then clearly by shifting by one-step forwards

$$x_{k+1}^1 = x_{k+1} = x_k^2$$

$$x_{k+1}^2 = x_{k+2} = -a_1 x_{k+1} - a_2 x_k + u_k$$

So that

$$x_{k+1}^2 = -a_1 x_k^2 - a_2 x_k^1 + u_k$$

Writing in matrix form

$$\begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k \tag{11.40}$$

We have also

$$y_k = (b_1 z + b_2) x_k$$
$$= b_1 x_{k+1} + b_2 x_k$$

Giving

$$y_k = b_1 x_k^2 + b_2 x_k^1 \tag{11.41}$$

$$y_k = \begin{bmatrix} b_2 & b_1 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix} \tag{11.42}$$

11.3.2 Second Realisation

Define

$$x_k^1 = x_{k+1}$$
$$x_k^2 = x_k$$

Then clearly by shifting by one-step forwards

$$x_{k+1}^2 = x_{k+1} = x_k^1$$

$$\begin{aligned} x_{k+1}^1 = x_{k+2} &= -a_1 x_{k+1} - a_2 x_k + u_k \\ &= -a_1 x_k^1 - a_2 x_k^2 + u_k \end{aligned}$$

Hence

$$\begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_k \tag{11.43}$$

$$\begin{aligned} y_k &= b_1 x_{k+1} + b_2 x_k \\ &= b_1 x_k^1 + b_2 x_k^2 \end{aligned}$$

$$y_k = \begin{bmatrix} b_1 & b_2 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix} \tag{11.44}$$

Let us check the second method by returning to the transfer-function.

$$E(z) = \mathbf{H}(zI - \mathbf{F})^{-1}\mathbf{G}$$

$$\begin{aligned} (zI - \mathbf{F})^{-1} &= \begin{bmatrix} z + a_1 & a_2 \\ -1 & z \end{bmatrix}^{-1} \\ &= \frac{1}{z^2 + a_1 z + a_2} \begin{bmatrix} z & -a_2 \\ 1 & z + a_1 \end{bmatrix} \end{aligned}$$

Multiply by ***G*** from the right

$$\frac{1}{z^2+a_1z+a_2}\begin{bmatrix} z & -a_2 \\ 1 & z+a_1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{z^2+a_1z+a_2}\begin{bmatrix} z \\ 1 \end{bmatrix}$$

Now multiply from the left by $\boldsymbol{H}$

$$E(z) = \frac{1}{z^2+a_1z+a_2}\begin{bmatrix} b_1 & b_2 \end{bmatrix}\begin{bmatrix} z \\ 1 \end{bmatrix}$$

We get (11.38).

11.4 Signal-Flow Graphs

Signal-flow graphs are very prevalent in digital-signal processing and control. The only real difference is that sometimes they are drawn with square-like edges rather than curved, but otherwise apart from aesthetics they look similar. Of course, we have no integrators and have delay-terms instead. Consider (11.43) and (11.44)

$$\begin{bmatrix} x^1_{k+1} \\ x^2_{k+1} \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x^1_k \\ x^2_k \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_k$$

$$y_k = \begin{bmatrix} b_1 & b_2 \end{bmatrix}\begin{bmatrix} x^1_k \\ x^2_k \end{bmatrix}$$

Figure 11.2 shows the signal-flow graph. Note that in this figure $x^2_{k+1} = x_{k+1} = x^1_k$.

In the related discipline of digital signal processing (DSP) we often have filters without poles (other than at the origin). For example, the third-order FIR filter has difference equation

$$y_k = b_0u_k + b_1u_{k-1} + b_2u_{k-2} + b_3u_{k-3} \tag{11.45}$$

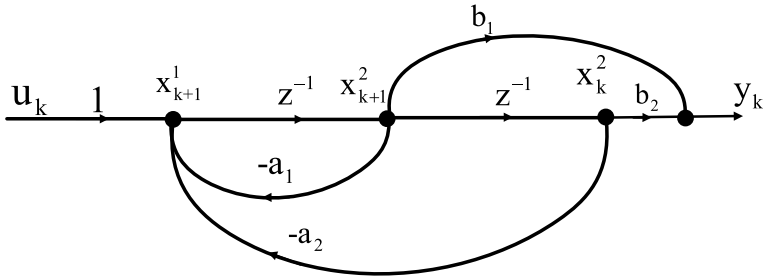


Fig. 11.2 Signal-flow graph for discrete-state-space realisation

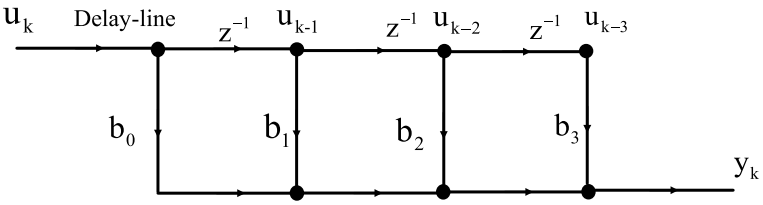


Fig. 11.3 Signal-flow graph of 3rd order FIR filter

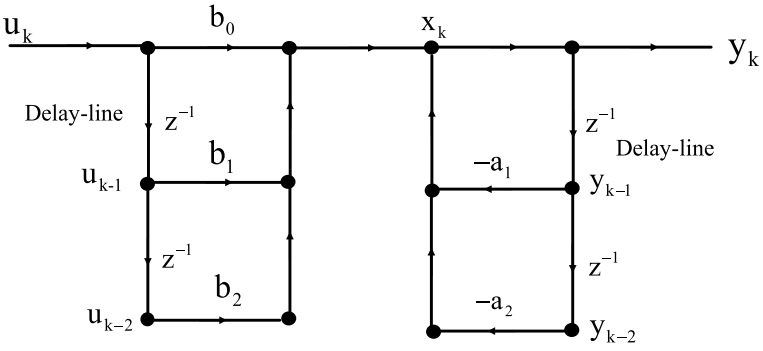


Fig. 11.4 Signal-flow graph for pole-zero digital filter of order 2

The signal-flow is drawn as in Fig. 11.3. Note that any flow (arrow) that has no value is defaulted to unity.

The so-called delay-line is historic and comes from an era when digital-filters were implemented in digital hardware since microprocessors were not fast enough for real-time computation. Note also the square edges of the signal-flow unlike the control-system convention. We can also draw a pole-zero system in the same manner by introducing an intermediate variable as we usually do in control-theory for state-space realisations. For example

$$\mathrm{E}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{11.46}$$

One realisation for the signal-flow graph is shown in Fig. 11.4.

When dealing with filters which have poles, it is interesting to see that the terms with poles have feedback whereas the zero part does not. This is only significant when implementing filters with poles (often called recursive digital filters as opposed to non-recursive for FIR filters). Finite arithmetic on some processors can mean that with rounding errors it is possible to get instability when implementing such filters, even though the filter is theoretically stable and has all its poles within

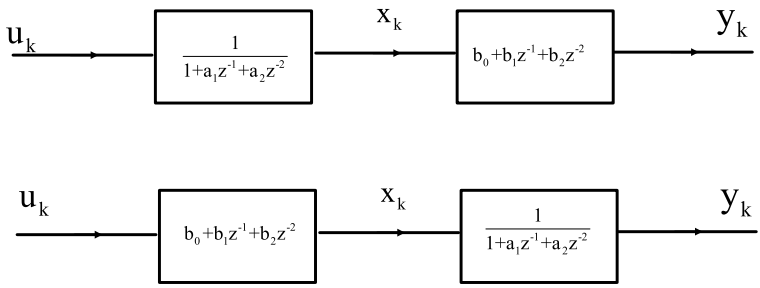


Fig. 11.5 Two different intermediate variables

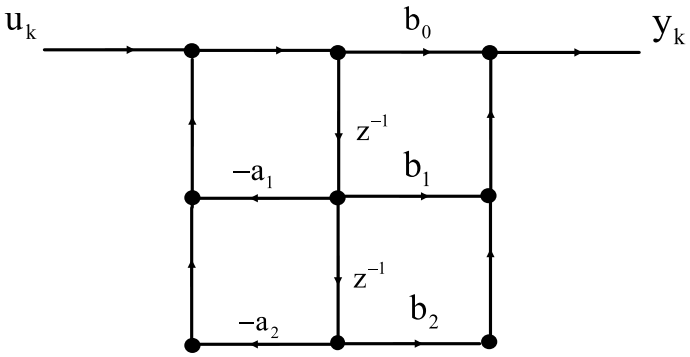


Fig. 11.6 Merging left and right sections into one

the unit-circle. There is a big literature available on this subject and often in DSP applications (open-loop filters), only FIR filters are used and filters with poles are avoided. There is another reason and this is because FIR filters can have linear-phase and filters with poles generally do not. For control-systems however we generally stick to pole-zero type filters where possible. Figure 11.4 is by far not the only realisation. For example, the left and right sections can be reversed. This is because the intermediate variable, although always between the pole and zero sections can depend on which part of the split of the filter happens first at the input. Figure 11.5 illustrates this point.

We can also combine the two half-sections into one in signal-flow graph. For the same example as above we can draw the diagram as shown in Fig. 11.6.

11.5 Solution of the Discrete-Time State-Equations

Consider the state-equation only, since the output equation is just a multiply by a vector or matrix.

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k$$

Suppose there is an initial-condition of the state $\mathbf{x}_0$ and the input at time zero is u_0. Now we can increment the time-index

$$\mathbf{x}_1 = \mathbf{F}\mathbf{x}_0 + \mathbf{G}\mathbf{u}_0 \tag{11.47}$$

Increment again

$$\mathbf{x}_2 = \mathbf{F}\mathbf{x}_1 + \mathbf{G}\mathbf{u}_1 \tag{11.48}$$

Substitute (11.47) into (11.48)

$$\mathbf{x}_2 = \mathbf{F}^2\mathbf{x}_0 + \mathbf{G}\mathbf{u}_1 + \mathbf{F}\mathbf{G}\mathbf{u}_0 \tag{11.49}$$

Now increment again

$$\mathbf{x}_3 = \mathbf{F}\mathbf{x}_2 + \mathbf{G}\mathbf{u}_2 \tag{11.50}$$

Substitute (11.49) into (11.5)

$$\mathbf{x}_3 = \mathbf{F}^3\mathbf{x}_0 + \mathbf{G}\mathbf{u}_2 + \mathbf{F}\mathbf{G}\mathbf{u}_1 + \mathbf{G}\mathbf{u}_2 + \mathbf{F}^2\mathbf{G}\mathbf{u}_0 \tag{11.51}$$

A pattern emerges

$$\mathbf{x}_k = \mathbf{F}^k\mathbf{x}_0 + \sum_{n=0}^{k-1} \mathbf{F}^{k-n-1}\mathbf{G}\mathbf{u}_n \tag{11.52}$$

The first term on the LHS is a matrix raised to the power k. To find out what happens in such a problem we write the matrix in its so-called *spectral-form* in terms of eigenvalues and modal-matrix.

$$\mathbf{F} = \mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1} \tag{11.53}$$

where $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues and $\boldsymbol{T}$ is the modal-matrix of eigenvectors. If we square the matrix $\boldsymbol{F}$ we can write

$$\begin{aligned}\mathbf{F}^2 &= \mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1}\mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1} \\ &= \mathbf{T}\boldsymbol{\Lambda}^2\mathbf{T}^{-1}\end{aligned}$$

Similarly, raising to any other power gives us

$$\mathbf{F}^{\mathrm{k}} = \mathbf{T}\boldsymbol{\Lambda}^{\mathrm{k}}\mathbf{T}^{-1} \tag{11.54}$$

If all the eigenvalues of the matrix have magnitude less than unity (which is the condition for stability of a LTI discrete-time system) then $\boldsymbol{F}^{k} \to 0$ for large k. Therefore (11.52) is an expression where the LHS will die out towards zero just as a stable LTI continuous-time system will do in terms of exponential decay. This leaves the RHS of (11.52) which is a *discrete-time convolution* summation term. For the special case of a constant step input $\mathrm{u_k} = \mathrm{u_c}$. We can calculate the steady-state value of the state-vector. We assume that the transient term has died out first and we are left with

$$\begin{aligned}\mathbf{x}_{\infty} &= \sum_{\mathrm{n}=0}^{\mathrm{k}-1} \mathbf{F}^{\mathrm{k}-\mathrm{n}-1}\mathbf{G}\mathrm{u_c} \\ &= \sum_{\mathrm{n}=0}^{\infty} \mathbf{F}^{\mathrm{n}}\mathbf{G}\mathrm{u_c} \\ &= \left(\mathbf{I} + \mathbf{F} + \mathbf{F}^2 + \ldots\right)\mathbf{G}\mathrm{u_c}\end{aligned}$$

Now $\left(\boldsymbol{I} + \boldsymbol{F} + \boldsymbol{F}^2 + \ldots\right)$ is a *matrix* geometric-progression and has a closed-form solution

$$\left(\mathbf{I} + \mathbf{F} + \mathbf{F}^2 + \ldots\right) = (\mathbf{I} - \mathbf{F})^{-1} \tag{11.55}$$

This gives a steady-state value for a step-input of

$$\mathbf{x}_{\infty} = (\mathbf{I} - \mathbf{F})^{-1}\mathbf{G}\mathrm{u_c} \tag{11.56}$$

This could also have been found using the discrete-time final-value theorem as applied to the z transfer-function

$$\mathbf{x}(\mathrm{z}) = (\mathrm{zI} - \mathbf{F})^{-1}\mathbf{G}\mathrm{u}(\mathrm{z})$$

$$\mathbf{x}(\mathrm{z}) = (\mathrm{zI} - \mathbf{F})^{-1}\mathbf{G}\frac{\mathrm{u_c z}}{(\mathrm{z}-1)} \tag{11.57}$$

This last equation follows from the transform of a step of magnitude $\mathrm{u_c}$

$$\mathrm{u}(\mathrm{z}) = \frac{\mathrm{u_c z}}{(\mathrm{z}-1)}$$

Apply the discrete-time final-value theorem when applied to (11.57) by multiplying by $(z-1)$ and letting $z \to 1$ in the limit.

This gives us, the same result, namely by

$$\mathbf{x}_\infty \rightarrow (z-1)(z\mathbf{I}-\mathbf{F})^{-1}\mathbf{G}\left.\frac{u_c z}{(z-1)}\right|_{z=1}$$
$$\rightarrow (z\mathbf{I}-\mathbf{F})^{-1}\mathbf{G}u_c$$

11.6 Discrete-Time State-Feedback

The equations for state-feedback follow exactly the same methodology as for continuous-time. We must first ensure that the system is completely controllable. A system of order n is fully controllable if the following controllability matrix has full rank.

$$\mathcal{C} = \left[\mathbf{G}, \mathbf{FG}, \mathbf{F}^2\mathbf{G}\ldots\mathbf{F}^{n-1}\mathbf{G}\right] \tag{11.58}$$

As a simple example, for $\mathbf{F} = \begin{bmatrix} 0 & 1 \\ -0.5 & 1 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ (note the rank test does not require the $\boldsymbol{H}$ vector or matrix). There are two states, n = 2

$$\mathbf{FG} = \begin{bmatrix} 0 & 1 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\mathcal{C} = [\mathbf{G}, \mathbf{FG}] = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

This matrix has full rank 2 indicating that both discrete-time states are controllable. We also at this stage assume that we can measure both states directly. The state-feedback control is then found from

$$u_k = r_k - \mathbf{K}\mathbf{x}_k \tag{11.59}$$

where r_k is the setpoint and the gain vector

$$\mathbf{K} = [k_1 \quad k_2 \quad . \quad . \quad k_n] \tag{11.60}$$

Substitute the control-signal into the system

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}u_k$$

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{F}\mathbf{x}_k + \mathbf{G}[r_k - \mathbf{K}\mathbf{x}_k] \\ &= [\mathbf{F} - \mathbf{GK}]\mathbf{x}_k + \mathbf{G}r_k \end{aligned} \tag{11.61}$$

For stability, we require the eigenvalues of $\boldsymbol{F} - \boldsymbol{GK}$ to lie within the unit-circle. Example. The double-integrator system

$$E(z) = \frac{1}{(z-1)^2} \tag{11.62}$$

Place the closed-loop poles at $z = -0.5, -0.6$ in the z-plane. First, find a state-space realisation for (11.62).

$$E(z) = \frac{1}{z^2 - 2z + 1} \tag{11.63}$$

Using one of our previously derived state-space realisations, we can write

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{F}\mathbf{x}_k + \mathbf{G}u_k \\ y_k &= \mathbf{H}\mathbf{x}_k \end{aligned}$$

$$\mathbf{F} = \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Check for controllability

$$\begin{aligned} \mathcal{C} &= \left[\mathbf{G}, \mathbf{FG}, \mathbf{F^2G} \ldots \mathbf{F^{n-1}G}\right] \\ &= \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \end{aligned}$$

This is full rank so both states are completely controllable. Now form the closed-loop matrix

$$\begin{aligned} \mathbf{F} - \mathbf{GK} &= \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ \mathbf{k}_1 & \mathbf{k}_2 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \\ -1 - \mathbf{k}_1 & 2 - \mathbf{k}_2 \end{bmatrix} \end{aligned}$$

The characteristic polynomial corresponding to this matrix is found from

$$\left| \begin{pmatrix} z & 0 \\ 0 & z \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ -1 - \mathbf{k}_1 & 2 - \mathbf{k}_2 \end{pmatrix} \right| = 0$$

or

$$z^2 + (k_2 - 2)z + 1 + k_1 = 0 \tag{11.64}$$

But we require a polynomial be design where our two closed-loop poles are placed, namely

$$\begin{aligned}(z+0.5)(z+0.6)&=0\\ z^2+1.1z+0.3&=0\end{aligned}\tag{11.65}$$

Now compare coefficients of (11.65) with (11.64)

$$\begin{aligned}k_2-2&=1.1\\ 1+k_1&=0.3\end{aligned}$$

Giving $k_1=-0.7, k_2=3.1$.

Now suppose we require the closed-loop transfer-function. We find this from the state-space description

$$\begin{aligned}\mathbf{x}_{k+1}&=[\mathbf{F}-\mathbf{GK}]\mathbf{x}_k+\mathbf{G}r_k\\ &=\begin{bmatrix}0 & 1\\ -0.3 & -1.1\end{bmatrix}\mathbf{x}_k+\begin{bmatrix}0\\ 1\end{bmatrix}r_k, \mathbf{H}=[1\quad 0]\end{aligned}$$

Take z-transforms

$$\begin{aligned}(z\mathbf{I}-[\mathbf{F}-\mathbf{GK}])\mathbf{x}(z)&=\begin{bmatrix}0\\ 1\end{bmatrix}r(z)\\ \mathbf{x}(z)&=(z\mathbf{I}-[\mathbf{F}-\mathbf{GK}])^{-1}\begin{bmatrix}0\\ 1\end{bmatrix}r(z)\\ \mathbf{y}(z)&=[1\quad 0](z\mathbf{I}-[\mathbf{F}-\mathbf{GK}])^{-1}\begin{bmatrix}0\\ 1\end{bmatrix}r(z)\end{aligned}$$

We get

$$\mathbf{y}(z)=\frac{1}{z^2+1.1z+0.3}r(z)\tag{11.66}$$

We can add integral action as we did in continuous time. An integrator has a z-transform given by $\frac{1}{1-z^{-1}}$. When applied to an error, we can define a new state as its output as follows:

$$x_k^I=\frac{1}{1-z^{-1}}e_k\tag{11.67}$$

where

$$e_k=r_k-\mathbf{H}\mathbf{x}_k\tag{11.68}$$

Our integrator transfer-function has a corresponding difference-equation given by

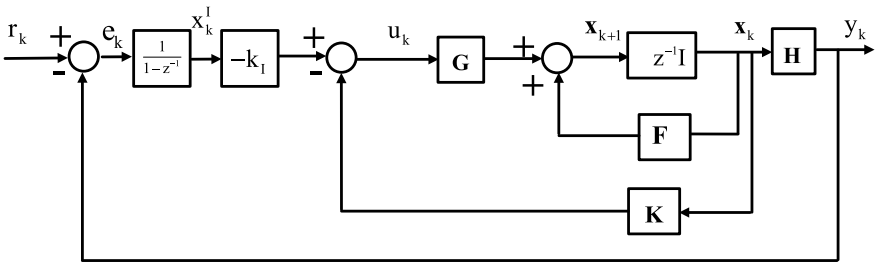


Fig. 11.7 Discrete-time state-feedback and integral action

$$x^{I}_{k+1} = x^{I}_{k} + e_k \tag{11.69}$$

This is an Euler integrator but is a good enough approximation and is the type of integrator used for nearly all such similar problems.

Now apply the state-feedback by adding the extra state and an extra gain k^I to act on it.

$$u_k = r_k - \begin{bmatrix} \mathbf{K} & k^I \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ x^I_k \end{bmatrix} \tag{11.70}$$

Now write the state-equation of an augmented system to include this extra state variable (Fig. 11.7).

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ x^I_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{F} - \mathbf{GK} & -\mathbf{G}k^I \\ -\mathbf{H} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ x^I_k \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k \tag{11.71a}$$

$$y_{\mathbf{k}} = \begin{bmatrix} \mathbf{H} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ x^I_k \end{bmatrix} \tag{11.71b}$$

Example.
Discrete-time state-feedback and integral action.

A system has z-transfer function $E(z) = \frac{1}{Z^2 - 2Z + 2}$. A quick check indicates this system is open-loop unstable with poles outside the unit-circle at $z = 1 \pm j$. A controllable-canonical form state-space realisation becomes:

$$\mathbf{F} = \begin{bmatrix} 0 & 1 \\ -2 & 2 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The augmented matrix is

$$\begin{bmatrix} \mathbf{F}-\mathbf{GK} & -\mathbf{G}k^I \\ -\mathbf{H} & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -2-k_1 & 2-k_2 & -k^I \\ -1 & 0 & 1 \end{bmatrix} \tag{11.72}$$

We have three gains to find and the eigenvalues of (11.72) are also three. First find the characteristic polynomial corresponding to (11.72)

$$\left| \begin{bmatrix} z & 0 & 0 \\ 0 & z & 0 \\ 0 & 0 & z \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ -2-k_1 & 2-k_2 & -k^I \\ -1 & 0 & 1 \end{bmatrix} \right| = 0 \tag{11.73}$$

$$\left| \begin{bmatrix} z & -1 & 0 \\ 2+k_1 & z-2+k_2 & k^I \\ 1 & 0 & z-1 \end{bmatrix} \right| = 0 \tag{11.74}$$

It is best to work along the top row when finding the determinant of (11.74). We get

$$z\begin{vmatrix} z-2+k_2 & k^I \\ 0 & z-1 \end{vmatrix} + \begin{vmatrix} 2+k_1 & k^I \\ 1 & z-1 \end{vmatrix} = 0$$

$$z(z-1)(z-2+k_2) + (z-1)(2+k_1) - k^I = 0$$

$$z^3 + z^2(k_2-3) + z(4+k_1-k_2) - 2 - k_1 - k^I = 0 \tag{11.75}$$

Now we place the poles. Let them be at $z = 0.1, 0.2, 0.3$ which is within the unit-circle. This makes a damped system since there are no complex closed-loop poles in our choice. The pole-polynomial becomes $(z-0.1)(z-0.2)(z-0.3)$ or

$$z^3 - 0.6z^2 + 0.11z - 0.006 = 0 \tag{11.76}$$

Compare coefficients of the two polynomials (11.76) and (11.75).

$$k_2 - 3 = -0.6, k_2 = 2.4$$

$$4 + k_1 - k_2 = 0.11 \text{ so } k_1 = -1.49$$
$$-2 - k_1 - k^I = -0.006 \text{ so } k^I = -0.504$$

Working back, our augmented matrices become

$$\begin{bmatrix} \mathbf{F}-\mathbf{GK} & -\mathbf{G}k^{\mathrm{I}} \\ -\mathbf{H} & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -0.51 & -0.4 & 0.504 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

$$[\mathbf{H} \quad 0] = [1 \quad 0 \quad 0]$$

We can convert this to a z transfer-function in MATLAB and then take the step-response by using the following code.

MATLAB Code 11.2

```
% Simulate discrete-time state-space plus integral action
%Closed-loop matrices follow
F=[0 1 0
  -0.51 -0.4 0.5040
  -1 0 1]
G=[0 0 1]'
H=[1 0 0]
%Convert to polynomial form z- domain
[num,den]=ss2tf(F,G,H,0)
%Convert to z transfer function
%Let sample interval be unity
e=tf(num,den,1)
step(e)
```

Figure 11.8 shows the step-response of the discrete-time system.

The closed-loop system is stable and well damped. The delay in the output is the inherent delay that takes a third-order system to respond in discrete-time (3 steps). Note there is no steady-state error due to the integral gain term. We can show the state-response by converting the discrete-time matrices to a dynamic system in MATLAB with the sys=ss(F,G,H,0,1) command. This is for unity sampling interval. The step-response of the states is then found from [y,t,x]=step(sys) from which we can then plot the states with plot(x). The state-response to a unit-step is shown in Fig. 11.9.

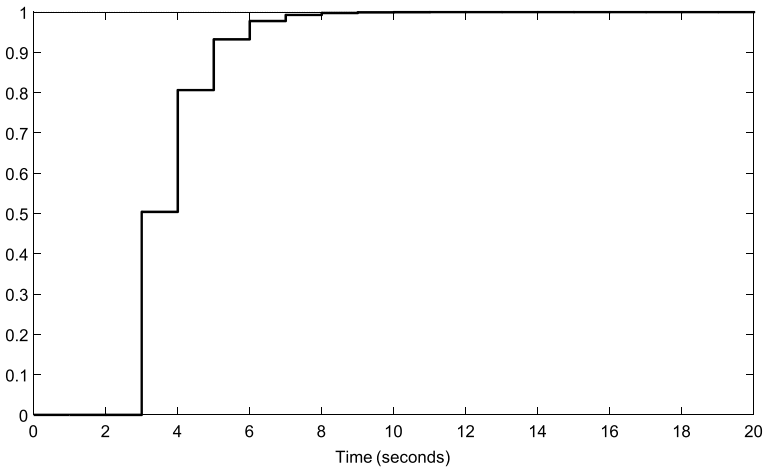


Fig. 11.8 Unit step-response of closed-loop system

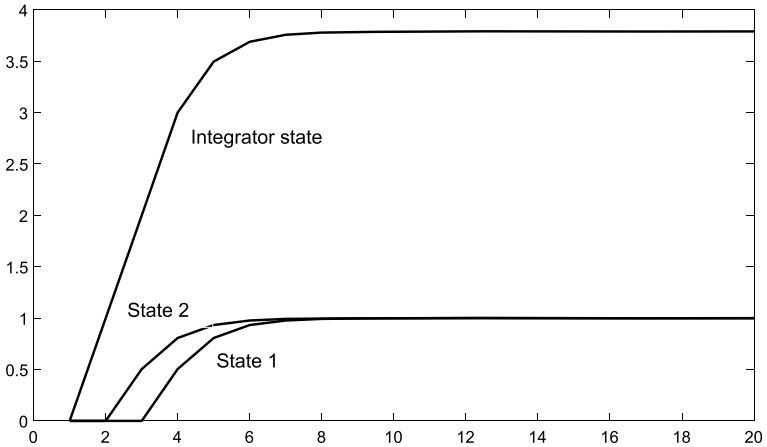


Fig. 11.9 Response of the states to a unit step

11.7 Discrete-Time Observers

Much of the theory has already been covered in Chap. 8. We need only convert to discrete-time. A discrete-time observer will reconstruct the discrete states when we cannot measure them directly with sensors. The state-observer is found from

$$\hat{\mathbf{x}}_{k+1} = \mathbf{F}\hat{\mathbf{x}}_k + \mathbf{G}u_k + \mathbf{L}(y_k - \mathbf{H}\hat{\mathbf{x}}_k) \tag{11.77}$$

where $\mathbf{L} = [\ell_1 \quad \ell_2 \quad . \quad . \quad \ell_n]^T$ is a column vector of gains which is to be determined by placement of the observer poles and the hat symbol above the state-vector denotes an estimate. (In other words, the state is not exactly known.) Before proceeding, we must first ensure that the states are fully observable.

The states of an nth order system are said to be observable if the observability matrix

$$\mathcal{O} = \begin{bmatrix} \mathbf{H} \\ \mathbf{HF} \\ \mathbf{HF}^2 \\ . \\ . \\ \mathbf{HF}^{n-1} \end{bmatrix}$$

has full rank n. For example, consider the system

$$\mathbf{F} = \begin{bmatrix} 0 & 1 \\ -0.5 & 1 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{H} = [2 \quad 1]$$

$$\mathcal{O} = \begin{bmatrix} \mathbf{H} \\ \mathbf{HF} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ -0.5 & 3 \end{bmatrix}$$

This has rank 2 so we say that both states are fully observable.

Now, re-arranging (11.77)

$$\hat{\mathbf{x}}_{k+1} = (\mathbf{F} - \mathbf{LH})\hat{\mathbf{x}}_k + \mathbf{G}u_k + \mathbf{L}y_k \tag{11.78}$$

The closed-loop poles of the observer are then found by the eigenvalues of $\mathbf{F} - \mathbf{LH}$. The block-diagram of a discrete-time observer is shown in Fig. 11.10.

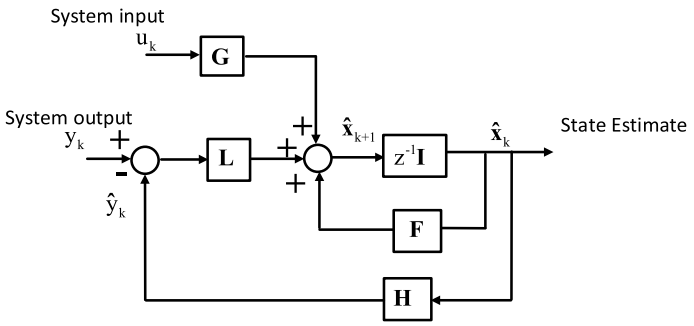


Fig. 11.10 Discrete-time observer

The unknown gain of the vector (or matrix if multivariable) $\boldsymbol{L}$ are found by placing the eigenvalues of $\mathbf{F} - \mathbf{LH}$ to be faster than the open-loop eigenvalues of the system. The error between the estimated and true state will then converge fast enough so that the two are indistinguishable from one-another. This is of course provided that an accurate enough model of the system has first been obtained.

11.7.1 Example of a Discrete-Time Observer

Consider a second-order discrete-time system given by $\mathbf{F} = \begin{bmatrix} 0 & 1 \\ -0.5 & 0.7 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$. The system is open-loop stable. We can check if it is observable by forming $\mathcal{O} = \begin{bmatrix} \mathbf{H} \\ \mathbf{HF} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. The matrix is full-rank indicating that both states are fully observable. Form the matrix $\mathbf{F} - \mathbf{LH}$, where $\mathbf{L} = \begin{bmatrix} \ell_1 \\ \ell_2 \end{bmatrix}$. We get

$$\mathbf{F} - \mathbf{LH} = \begin{bmatrix} 0 & 1 \\ -0.5 & 0.7 \end{bmatrix} - \begin{bmatrix} \ell_1 & 0 \\ \ell_2 & 0 \end{bmatrix} = \begin{bmatrix} -\ell_1 & 1 \\ -0.5 - \ell_2 & 0.7 \end{bmatrix}.$$

The characteristic polynomial becomes $\mathrm{z}^2 + \mathrm{z}(\ell_1 - 0.7) - 0.7\ell_1 + \ell_2 + 0.5$. We place the closed-loop poles at $(\mathrm{z} - 0.1)(\mathrm{z} - 0.2) = \mathrm{z}^2 - 0.3\mathrm{z} + 0.02$. Comparing coefficients we get $\ell_1 - 0.7 = -0.3, \ell_1 = 0.4$ and $-0.7\ell_1 + \ell_2 + 0.5 = 0.02, \ell_2 = -0.2$. This gives us a closed-loop matrix for the observer of $\mathbf{F} - \mathbf{LH} = \begin{bmatrix} -0.4 & 1 \\ -0.3 & 0.7 \end{bmatrix}$ which has the desired eigenvalues.

Chapter 12
Systems with Random Noise

Up to this chapter we have considered signals that are noise-free or *deterministic*. In reality, since all control-systems involve measurements, these measurements must always have inherent contaminated noise which adds to the uncertainty of the measurement. The noise itself comes from the physics of the device at the atomic level and cannot be removed. We therefore cannot say with exact certainty what the value of the output of a system will be, but we can define its probability. We can say that a signal has a certain probability, and this in itself is a useful mathematical tool to have. We name systems with noise *stochastic.* In a deterministic system, if we know the mathematical model of the system, then we can calculate exactly what the output will be at any time for a particular input. For a stochastic system we still can calculate the output but added on top we can say there is a probability that the value will be greater or less than a certain value. There is an exception to these rules found in nonlinear systems that exhibit chaotic outputs. Such systems can be deterministic yet look as if the output is random. In control-systems however we usually do not have to face such problems. Note that in this chapter we often move with little warning between continuous and discrete-time. We denote expressions with arguments such as $y(t)$ as being continuous-time and with integer subscripts thus—y_k, as discrete-time.

12.1 White Noise and Probability

The purest form of noise that exists is white noise. If we take even a resistor with an applied voltage and look at the waveform on an oscilloscope we will see that there is additive noise to that signal. It doesn't matter if the signal is dc or ac. More than often the noise will not be exactly white-noise because white-noise does not exist in real life, only in theory. In theory, white-noise has an equal probability of having any frequency present. It would stretch from dc to daylight in frequency-response terms. Therefore, an engineering definition is much favoured where we consider

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_12

bandlimited white-noise. Bandlimited white-noise is just white-noise with every frequency (of equal probability existing) from a very low frequency to a high frequency which is within the bandwidth of the system you are working on. Figure 12.1 shows a plot of simulated white-noise.

To understand white-noise, we first must recall work on basic statistics. White-noise has a probability-density function (PDF) which is Gaussian (or Normal) in nature. The mathematical expression for a Gaussian PDF is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \tag{12.1}$$

In (12.1) $p(x)$ is the probability that a random signal x has with *mean* (or average) value μ and *standard-deviation* σ. If we look closely at Fig. 12.1 we see that we cannot define its amplitude like we do with sine-waves of deterministic signals. We can see that most of the signal is concentrated in a band which stretches from 3 standard-deviations to minus three standard-deviations. There are a few values outside of this, but they are not significant for the 1000 samples in the plot. One thing which simplifies a lot of our work in electronics is that we can assume that the noise has zero average value or mean. The mean of a signal is the easiest parameter to understand, but the standard-deviation is a little more different. To understand it we must look at the typical PDF of a Gaussian waveform as shown in Fig. 12.2.

A PDF is just a histogram in the practical sense. If we had a sample of a thousand values and counted how many of each we had, and put them in a graph, we would get a histogram. The PDF is the theoretical histogram. Most of the values are centred around the mean, which is conveniently zero for all for our work. We see that at one standard-deviation (including negative values too), approximately 68% of all the values are present. Moving out to two standard-deviations we see that around 95% of all the values exist. Finally, to three standard-deviations we find 99.7% of all the values are present. Therefore, it is very unlikely to see any value beyond three sigma. Of course, there will be values beyond three sigma but they

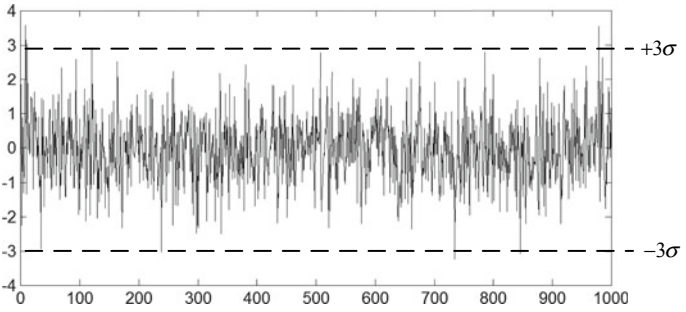


Fig. 12.1 Simulated white-noise for 1000 samples $\sigma = 1, \mu = 0$

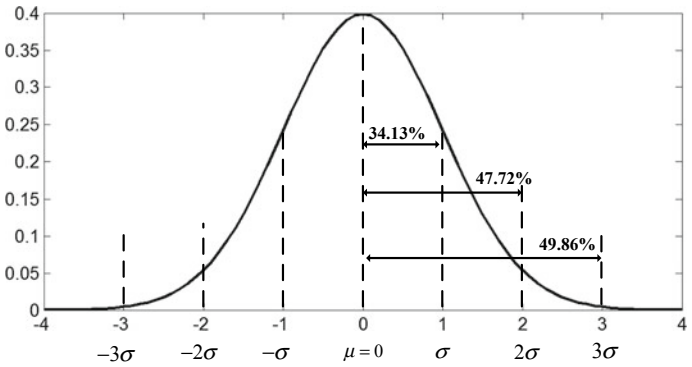


Fig. 12.2 PDF of Gaussian distribution

have a very low probability. Likewise, even more so for 4 standard-deviations and beyond. Figure 12.2 is drawn for a standard-deviation of unity (for reference). Other PDFs would look the same but far narrower or wider if the standard-deviation were smaller or larger respectively. Referring to Fig. 12.1, we can measure the standard deviation crudely on an oscilloscope. Turning the signal to zero we are always left with noise (provided the noise is additive of course). We can then make an estimate of how many standard-deviations it is from the mean which is the dc value and usually zero. If we have the samples themselves we can calculate them from the sample mean and variance.

The mean for N samples of a sampled random signal (discrete-time) $\xi_k, k = 1, 2, 3 \ldots N$ is

$$\mu = \frac{1}{N}\sum_{k=1}^{N} \xi_k \tag{12.2}$$

This would be close to zero if N was large enough (assuming no dc—most electronic measurements are ac coupled but not all of course so there could be situations where there is dc (i.e. a mean value which is not zero).

The sampled variance for zero mean is given by

$$\sigma^2 = \frac{1}{N}\sum_{k=1}^{N} \xi_k^2 \tag{12.3}$$

Its square root is the standard deviation (or root-mean square (rms)) σ. We can also theoretically relate this to the PDF itself.

The mean (average or dc in electronic engineering) is mathematically defined as the first moment. In all work we assume the operator $E\{.\}$ represents the statistical *expected* value of an expression.

$$E\{x\} = \int_{-\infty}^{\infty} xp(x)dx = \mu \tag{12.4}$$

This also represents *ensemble* average. This is the expected outcome if we had numerous trials.

The mean-square (or second moment)

$$E\{x^2\} = \int_{-\infty}^{\infty} x^2 p(x)dx \tag{12.5}$$

The above Eq. (12.5) is also the variance (turns out to be average power) provided the mean is zero. No moments exist beyond the variance for Gaussian distributions. For other PDFs there can be higher-order moments which describe the skewness and other properties of the PDF. A good example of a PDF which is not Gaussian is the PDF of a speech waveform. It has the Laplace PDF instead.

If a white noise signal (continuous-time variable) on an oscilloscope has zero-dc level, then its expected value (or mean) will be zero thus

$$E\{\xi(t)\} = 0 \tag{12.6}$$

where $\xi(t)$ is the white noise signal.

Its variance or average power (in engineering terminology) is the second statistical moment and given by

$$E\{\xi^2(t)\} = \sigma^2 \tag{12.7}$$

where σ^2 is variance and σ is standard deviation or rms. White noise is uncorrelated from any time t_1 to some other time t_2. Its autocorrelation function is therefore an impulse:

$$E\{\xi(t_1)\xi(t_2)\} = \sigma^2\delta(t_1 - t_2) \tag{12.8}$$

The impulse-function tells us that the impulse occurs only when time $t_1 = t_2$ and nowhere else. Hence if you take a sample of white noise and another sample at another time there will be no relationship between them. They are said to be *uncorrelated*.

Similarly, *Autocorrelation* of white-noise is defined as:

$$R_{\xi\xi}(\ell) = E\{\xi_k \xi_{k+\ell}\} = \sigma^2\delta_\ell \tag{12.9}$$

$$= \frac{1}{N}\sum_{k=1}^{N} \xi_k \xi_{k+\ell}, \ell = 0, 1, 2, 3, \ldots \tag{12.10}$$

The above formulae are given with a slight abuse of notation since the Dirac delta is used for continuous-time and Kronecker delta for discrete-time.

Autocorrelation is at a maximum when the delay $\ell = 0$. In fact, $R_{\xi\xi}(0) = \sigma^2$, the variance. Instead of correlating with the same signal we can correlate with an entirely different signal and we get *cross-correlation.* Autocorrelation is like a time-domain version of power-spectrum.

Stationary and Non-Stationary
A stationary random signal is one whose statistics do not change with time. For example, the mean and variance stay the same. The PDF must also be the same. A good example of a random signal that is non-stationary is a sample of a speech waveform. Its variance changes over time (though its mean will stay unaffected).

12.2 Coloured Noise

Wiener–Khinchen Theorem
In theory, there are two further important functions which can be calculated—these are the Fourier Transforms of Autocorrelation and Fourier Transform of Cross-Correlation. There is a well-known formula allegedly first found by Einstein. However, it is more commonly known as the Wiener-Khinchin Theorem. This states that the Fourier transform of Autocorrelation is Power Spectral Density and the Fourier transform of Cross-correlation is Cross-Power Spectral Density, i.e. for Power Spectrum (Power spectral density—PSD) $S_{\xi\xi}(\omega)$ in continuous time for some time-delay τ.

$$\mathrm{S}_{\xi\xi}(\omega) = \int_{-\infty}^{\infty} \mathrm{R}_{\xi\xi}(\tau)\mathrm{e}^{-\mathrm{j}\omega\tau}\mathrm{d}\tau \tag{12.11}$$

For Cross Power Spectrum $S_{\xi\eta}(\omega)$

$$\mathrm{S}_{\xi\eta}(\omega) = \int_{-\infty}^{\infty} \mathrm{R}_{\xi\eta}(\tau)\mathrm{e}^{-\mathrm{j}\omega\tau}\mathrm{d}\tau \tag{12.12}$$

But despite these relationships, they are seldom used directly i.e. if $S_{\xi\xi}(\omega)$ is needed, it can be computed directly using a Fast-Fourier Transform (FFT) and the Periodogram (sampled estimate of Power Spectral Density) is used. Autocorrelation is not computed (or is avoided where necessary) because it is slow to compute for different lags $\ell = 0, 1, 2\ldots$. If however, autocorrelation is known for some reason. then simply by taking its FFT the PSD can be found. PSD is a plot of Power versus frequency. We usually see spectrum plots showing peaks and troughs etc. indicating where a signal has most or least power. Here is a typical PSD plot generated

in National-Instruments (NI) LabView. It is created by passing unit-variance white-noise through a second-order continuous-time system with damping-factor 0.4 and undamped natural frequency 0.1 (normalised frequency for unity sampling). For pure white-noise the spectrum will be flat. Here however the white-noise is shaped by the spectrum of the system (Fig. 12.3).

It can be seen that the power-spectrum has not been averaged, so it is quite noisy. By averaging over successive batches, we can smooth this out. More important is the peak that occurs at 0.1 Hz. The white-noise spectrum has been shaped by the filter and hence the name coloured-noise. In other words, the power around 0.1 Hz is higher than other frequencies. The coloured-noise signal has samples which are now correlated with each other, unlike white-noise. It has also a well-defined autocorrelation and not an impulse as with the white-noise case. With such real-time analysis of signals available, passing white-noise through an unknown

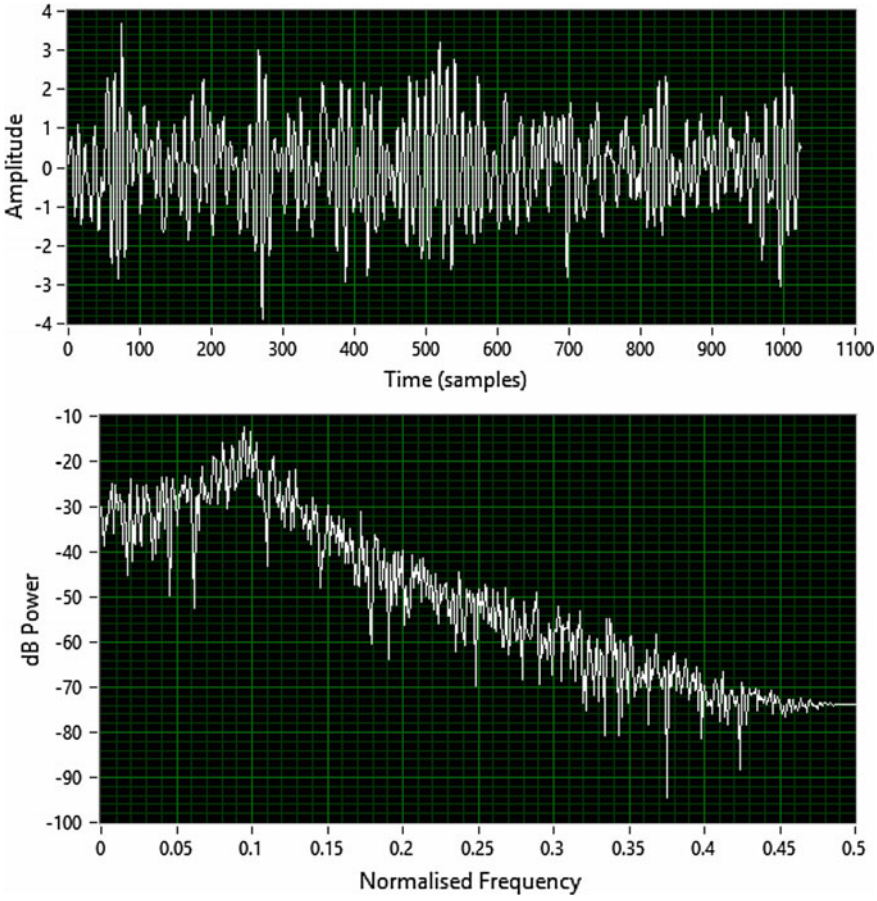


Fig. 12.3 Time-domain and frequency-domain for coloured-noise

system can result in the identification of the system itself. This is a bit like a Bode-plot but much noisier. After averaging over 100 batches of data, each set 1024 samples, we get a spectrum as shown in Fig. 12.4. This is plotted on a log frequency axis like a Bode-plot to make a comparison.

Although the peak in the spectrum does not have the exact maximum value, it is by far an impressive way of quickly obtaining a frequency-response. The mathematics goes as follows. It can be shown that if we pass zero-mean white noise $u(t)$ of variance σ_u^2 through a filter with a known transfer function, say $W(j\omega)$ (see Fig. 12.5): then the PSD of the signal (say y(t)) that comes out of the filter (i.e. a coloured-noise signal) is given by $S_{yy}(\omega)$ where

$$\mathrm{S_{yy}}(\omega) = |\mathrm{H}(\mathrm{j}\omega)|^2 \sigma_\mathrm{u}^2 \tag{12.13}$$

This can also be written as

$$\mathrm{S_{yy}}(\omega) = \sigma_\mathrm{u}^2 \mathrm{H}(\mathrm{j}\omega)\mathrm{H}(-\mathrm{j}\omega) \tag{12.14}$$

Example: Power-Spectral Density (PSD)
Suppose the variance is unity i.e. $\sigma_u^2 = 1$ and the filter has transfer function $H(j\omega) = \frac{1}{(1+j\omega)}$. Then the PSD of the coloured signal at the output will be

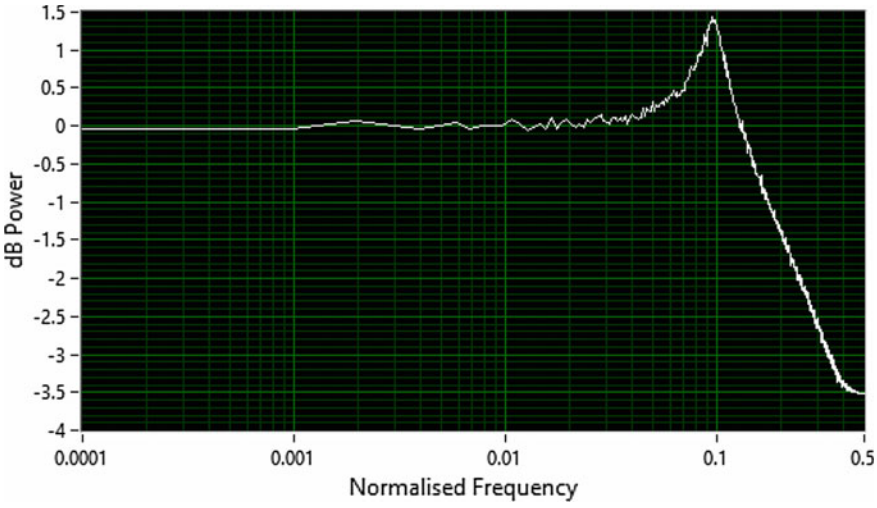


Fig. 12.4 Bode-plot obtained after averaging spectrum

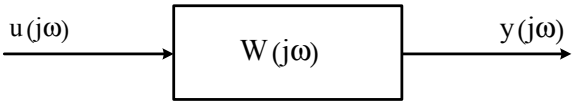


Fig. 12.5 A colouring filter. White noise in, coloured-noise out

$S_{yy}(\omega) = |H(j\omega)|^2\sigma_u^2 = \frac{1}{(1+\omega^2)}$. This can be plotted for positive and negative values of ω. If we were bold enough and were to take the inverse Fourier transform of $\frac{1}{(1+\omega^2)}$ i.e.

$$R_{\xi\xi}(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_{yy}(\omega)e^{j\omega\tau}d\omega \tag{12.15}$$

Then we would arrive at the autocorrelation of the coloured signal, i.e. the Wiener Khinchen theorem in reverse! This is seldom done except in theoretical work. The FFT is always used for practical work and numerical rather than analytical methods are more the norm. However, from tables we can easily find the inverse transform for this example to be $R_{yy}(\tau) = 0.5\exp(-|\tau|)$. Another important point follows from the inverse Fourier Transform of the PSD by putting $\tau = 0$. We arrive at

$$R_{yy}(0) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_{yy}(\omega)d\omega = \sigma_y^2 = E[y^2(t)] \tag{12.16}$$

The average power σ_y^2 is the area under the PSD graph and is also the *mean-square* in the statistical sense of the output of the colouring filter. This can be written in an alternative way in terms of the transfer function of the colouring filter

$$R_{yy}(0) = \frac{\sigma_u^2}{2\pi}\int_{-\infty}^{\infty} |H(j\omega)|^2 d\omega = \sigma_y^2 = E[y^2(t)] \tag{12.17}$$

If we replace the mean-square (i.e. variance if the dc is zero) $E[y^2(t)]$ by its time average in terms of sampled-data values we also get

$$\sigma_y^2 = \frac{1}{N}\sum_{k=1}^{N} y_k^2 = \frac{\sigma_u^2}{2\pi}\int_{-\infty}^{\infty} |H(j\omega)|^2 d\omega \tag{12.18}$$

This is a statement that says that the average power in the *time-domain* is the same as in the *frequency domain*. It is known as *Parsevals Theorem*.

12.3 Whitening-Filter

Certain problems, particularly in optimal filtering require the opposite of a colouring filter, namely a whitening-filter. Coloured-noise goes in and white-noise comes out. In the simplest of cases we have the situation shown in Fig. 12.6.

We ensure that the inverse filter is of course stable before we invert it. This is ensured because such problems usually have minimum-phase zeros before they are inverted making stable poles.

In reality, the problem is a little harder because y(t) normally has additive white noise (uncorrelated with the original noise u(t)). The whitening filter is no longer the inverse of *W* but must be found by what is known as *spectral-factorization*. By uncorrelated additive zero-mean white noise (variance σ_v^2) we mean an extra noise source say v(t) which satisfies

$$\mathrm{E}\{\mathrm{u}(\mathrm{t})\mathrm{v}(\mathrm{t})\} = 0 \tag{12.19}$$

That is, they bear no relationship to one another in the statistical sense. For a coloured signal y(t) plus white-noise v(t) we can write in continuous-time its sum

$$\mathrm{s}(\mathrm{t}) = \mathrm{y}(\mathrm{t}) + \mathrm{v}(\mathrm{t}) \tag{12.20}$$

Then the spectrum of the sum becomes

$$\mathrm{S}_{\mathrm{ss}}(\omega) = \mathrm{S}_{\mathrm{yy}}(\omega) + \sigma_{\mathrm{v}}^2 \tag{12.21}$$

Then we can define another filter *F* which will give us the same spectrum as found from signal plus noise

$$\mathrm{S}_{\mathrm{ss}}(\omega) = |\mathrm{F}(\mathrm{j}\omega)|^2 \sigma_{\varepsilon}^2 \tag{12.22}$$

This tells us that a LTI filter when driven by yet another white-noise signal with variance σ_{ε}^2 (let is call this signal $\varepsilon(t)$) will generate the same spectrum provided it satisfies

$$|\mathrm{F}(\mathrm{j}\omega)|^2 \sigma_{\varepsilon}^2 = \mathrm{S}_{\mathrm{yy}}(\omega) + \sigma_{\mathrm{v}}^2 \tag{12.23}$$

Solving (12.23) is known as the spectral-factorization problem and appears in the literature on Wiener-filters. Therefore, the inverse of *F* once we have found it is the whitening filter.

See Fig. 12.7.

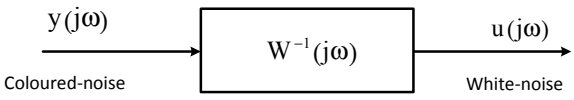


Fig. 12.6 Whitening-filter

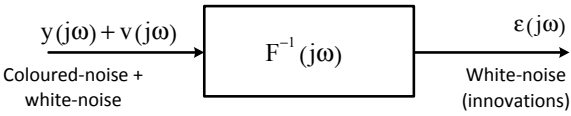


Fig. 12.7 Whitening filter for signal plus uncorrelated white-noise

The white-noise that comes out is correlated with the signals that go in but is not the same white-noise as found at the input. The white-noise is named the *innovations* and discovered by Norbert-Wiener. It plays a major role along with the whitening-filter in optimal filter theory. The filter can always be inverted due to the setup of the spectral-factorization problem which ensures that F always has minimum-phase zeros.

12.4 Discrete-Time Spectrum

For sampled signals the same sort of thing applies, but in some cases the theory is slightly easier if we assume that discrete coloured noise is defined as white noise passing through a digital filter which has no poles (i.e. zeros only). Such a filter we know to be a finite-impulse response filter (FIR). We need z-transforms to describe such a filter. For discrete white noise $u_k, k = 0, 1, 2, \ldots$ of variance σ^2, it can be passed through an FIR filter of transfer function $H(z)$ and the spectrum of the signal that comes out (PSD but it is normally the called z-transform spectral density in this case) is given by

$$\mathrm{S_{yy}(z) = H(z)H(z^{-1})\sigma^2} \tag{12.24}$$

or alternatively

$$\mathrm{S_{yy}(z) = |H(z)|^2\sigma^2} \tag{12.25}$$

12.4.1 Discrete-Time Spectrum and Autocorrelation Example

White noise of unit variance is passed through a linear FIR filter with transfer function $H(z) = 1 + 0.5z^{-1}$, find the autocorrelation of the output signal and find the average power of the signal at the output.

The z-transform PSD is found from

$$\begin{aligned} S_{yy}(z) &= H(z)H(z^{-1})\sigma^2 \\ &= (1+0.5z^{-1})(1+0.5z) \\ &= 0.5z+1.25+0.5z^{-1} \end{aligned}$$

(*Note—at this point it is worth mentioning that there is a separate problem where we are given* $0.5z+1.25+0.5z^{-1}$ *and we have to find the colouring filter* $H(z)$*—this is called spectral factorization and we already mentioned it above.*)

The autocorrelation is the inverse discrete Fourier transform of the above but luckily, since the PSD is in z-transform form (and not in the form $z = e^{j\theta}$), we only need take the inverse z-transform. For such a case when there are no poles, the coefficients of z become the discrete autocorrelation (this is just the discrete form of the Wiener-Khinchen theorem). The autocorrelation terms are simply

$$\begin{aligned} \phi_{yy}(-1) &= 0.5 \\ \phi_{yy}(0) &= 1.25 \\ \phi_{yy}(1) &= 0.5 \end{aligned}$$

There are no other terms. The average power is the zeroth (no delay) autocorrelation term (as in the analogue case) giving $\sigma_y^2 = 1.25$ or an rms value of $\sigma_y = \sqrt{1.25}$. If $H(z)$ had poles, then this requires a look-up table to solve, since it needs a complex contour integration in the z-plane. The simple case above is for FIR filters only.

Example—Time Delay Estimation

Suppose a signal (not moving in space) is sent to two sensors spaced apart. Provided the signal is not directly in front or behind the two sensors the signal must take a different time to reach each sensor. Suppose the signal takes 5 samples to reach one sensor with magnitude 0.9 and 7 samples to reach the other sensor with magnitude 0.5. Can we from direct measurements of the two signals estimate the time-delay between the two signals? (called the time-difference of arrival TDOA). We model the two signals as y1 and y2 thus (note the attenuations of the signal)

$$\begin{aligned} y1_k &= 0.9s_{k-5} \\ y2_k &= 0.5s_{k-7} \end{aligned}$$

where s_k is the unknown signal at time k = 0, 1, 2…. Let us assume for simplicity that the signal is zero-mean white-noise with variance σ^2. If it isn't white, we can still proceed but it is a little more complicated.

We define (by taking z-transforms of y1 and y2)

$$\mathrm{y1_k} = \mathrm{H_1(z)s_k}$$
$$\mathrm{y2_k} = \mathrm{H_2(z)s_k}$$

where $H_1(z) = 0.5z^{-5}$ and $H_2(z) = 0.9z^{-7}$.

Then the cross-spectral density between y1 and y2 is

$$\mathrm{S_{y1y2}(z) = H_1(z)H_2(z^{-1})\sigma^2}$$

Note the complex-conjugate for H_2. We could take the conjugate of H_1 instead (i.e. $S_{y2y1} = H_2(z)H_1(z^{-1})\sigma^2$) but we would end up with a positive delay—this is not-causal i.e. it is in the future—but we could still get away with this as we see below.

Hence, we have the z-transform cross-PSD as

$$\begin{aligned}\mathrm{S_{y1y2}(z)} &= \mathrm{0.5z^{-7}X0.9z^5\sigma^2} \\ &= \mathrm{0.45z^{-2}\sigma^2}\end{aligned}$$

Taking inverse z-transforms gives us the *cross-correlation* and it has only one term, the rest are zero. It is therefore an impulse at time-index −2 with amplitude 0.45.

We have $\phi_{y1y2}(-2) = 0.45$ and zero elsewhere. If the sensors were reversed we would get $\phi_{y2y1}(2) = 0.45$, a positive delay at time index +2. We must therefore calibrate things to know which way around. A positive delay (i.e. pure time-advance) only means the sensors are the wrong way around.

Now suppose that the signal is not white but coloured. Any coloured signal or noise can be modelled as white noise passing through a colouring filter. If we know the colouring filter the maths is altered accordingly. Essentially, for the white-noise case above the z-transform cross-PSD $S_{y1y2}(z) = H_1(z)H_2(z^{-1})\sigma^2$ gets altered to be $S_{y1y2}(z) = H_1(z)H_2(z^{-1})S_{ss}(z)$ where $S_{ss}(z)$ is the PSD of the coloured signal and this in turn can be broken down into a colouring filter say $H_3(z)$. We would then get back to another case driven by white-noise i.e. the PSD of the signal would become $S_{ss}(z) = H_3(z)H_3(z^{-1})\sigma^2$ and the cross-PSD would becomes

$$\mathrm{S_{y1y2}(z) = H_1(z)H_2(z^{-1})H_3(z)H_3(z^{-1})\sigma^2}$$

a bit more complicated to work out but the same idea as before.

12.5 The Wiener Filter

Norbert Wiener was a mathematical prodigy who derived the now famous approach to estimating random signals contaminated by random noise [31]. He only looked at the continuous-time version and published the work in a book that was classified during world-war II. A Russian mathematician by the name of Kolmogorov solved the discrete-time case around the same time [32] and the theory is often referred to as the Wiener–Kolmogorov filter. The simplest of approaches leads us to consider a coloured-noise signal (although we refer to it as noise, it can in fact be a signal and generated in the same way). First, we consider the continuous-time case. The signal is generated by passing white-noise through a continuous-time LTI filter and the resulting output is then further contaminated by additive white-noise. We assume that the additive white-noise is uncorrelated with the noise driving the colouring filter. Figure 12.8 illustrates this.

We have zero-mean white-noise $\xi(t)$ driving the LTI system with variance σ_ξ^2. The LTI (stable) system has transfer-function $W(s)$. The additive white-noise $v(t)$ also has zero-mean and variance σ_v^2. The object of the method is to have a filter $H(s)$ which takes signal plus noise as the input and produces an estimate of the signal at its output. We define a least-squares problem that minimises a cost-function J where

$$\mathrm{J} = \mathrm{E}(\mathrm{y}(\mathrm{t}) - \widehat{\mathrm{y}}(\mathrm{t}|\mathrm{t}))^2 \tag{12.26}$$

The convention that $\widehat{\mathrm{y}}(\mathrm{t}|\mathrm{t})$ is used, which means the linear least-squares estimate of the signal at time t given observations up to time t. If the LTI system has an impulse-response $w(t) = \mathcal{L}^{-1}W(s)$, and the filter has impulse-response $h(t) = \mathcal{L}^{-1}H(s)$. Then we assume the convolution

$$\mathrm{y}(\mathrm{t}) = \int_{-\infty}^{\mathrm{t}} \mathrm{w}(\mathrm{t}-\tau)\xi(\tau)\mathrm{d}\tau \tag{12.27}$$

gives rise to the signal, and the signal plus noise is found from

$$\mathrm{s}(\mathrm{t}) = \mathrm{y}(\mathrm{t}) + \mathrm{v}(\mathrm{t}) \tag{12.28}$$

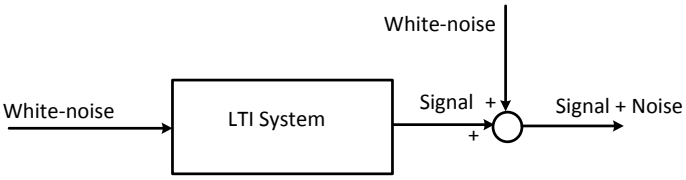


Fig. 12.8 Random signal plus noise

To indicate no statistical relationship between the two white-noise terms, statistically we must have

$$\mathrm{E}(\xi(\mathrm{t})\mathrm{v}(\mathrm{t})) = 0 \tag{12.29}$$

The filter-output becomes the estimated signal

$$\widehat{\mathrm{y}}(\mathrm{t}|\mathrm{t}) = \int_{-\infty}^{\mathrm{t}} \mathrm{h}(\mathrm{t}-\tau)\mathrm{s}(\tau)\mathrm{d}\tau \tag{12.30}$$

Fortunately, the Wiener-filter has a transfer-function solution that minimises (12.26). It is given by

$$\mathrm{H}(\mathrm{s}) = \left[\frac{\mathrm{W}(\mathrm{s})\sigma_\xi^2\mathrm{W}(-\mathrm{s})}{\Delta(-\mathrm{s})}\right]_+ \frac{1}{\Delta(\mathrm{s})} \tag{12.31}$$

The stable and minimum-phase spectral-factor $\Delta(\mathrm{s})$ is found from

$$\Delta(\mathrm{s})\Delta(-\mathrm{s}) = \mathrm{W}(\mathrm{s})\sigma_\xi^2\mathrm{W}(-\mathrm{s}) + \sigma_v^2 \tag{12.32}$$

Its mirror-image $\Delta(-\mathrm{s})$ is unstable and nonminimum-phase. This comes about because the optimal filter has impulse-response that runs both forwards and backwards in time. We must only take the part which is causal in positive-time. The special Wiener notation in (12.31) with the plus sign tells us to do this. Put concisely, $\mathcal{L}^{-1}\left[\frac{\mathrm{W}(\mathrm{s})\sigma_\xi^2\mathrm{W}(-\mathrm{s})}{\Delta(-\mathrm{s})}\right]_+$ exists for $\mathrm{t} > 0$ only. If we do not perform this step, our filter will be noncausal and hence unstable. In (12.32), we are essentially splitting the spectrum of signal plus noise so that a LTI system models it. This is the same idea as our theory above with the whitening filter (see 12.23). We illustrate the calculation of the Wiener-filter using an interesting example.

12.5.1 Wiener-Filter Example, Continuous-Time

Suppose a sensor measures acceleration, but we require position. Then we would have to integrate twice in order to get our result. Integration however in an open-loop system is fraught with problems. The slightest dc and the integrator output will drift. Therefore, we use an optimal filter to see what answer it gives. The block-diagram of the problem is shown in Fig. 12.9.

We are using a model of the equations of motion. If the noise measures acceleration then by integrating twice we arrive at position. Assume for simplicity that

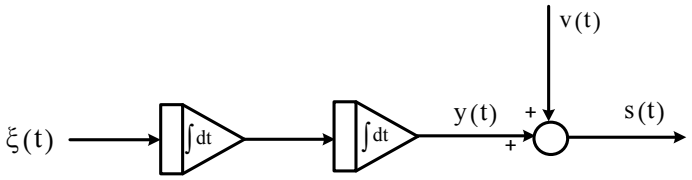


Fig. 12.9 Estimator for double-integration of white-noise

both white-noise terms have unit-variance. $\sigma_v^2 = \sigma_\xi^2 = 1$. We must also have that $\mathrm{W(s)} = \frac{1}{\mathrm{s}^2}$. Now form the spectral factorisation problem

$$\begin{aligned}\Delta(\mathrm{s})\Delta(-\mathrm{s}) &= \mathrm{W(s)}\sigma_\xi^2\mathrm{W}(-\mathrm{s}) + \sigma_v^2 \\ &= \frac{1}{\mathrm{s}^4} + 1 \\ &= \frac{\mathrm{s}^4 + 1}{\mathrm{s}^4}\end{aligned}$$

Factorising the numerator

$$\mathrm{s}^4 + 1 = (\mathrm{s}^2 + \sqrt{2}\mathrm{s} + 1)(\mathrm{s}^2 - \sqrt{2}\mathrm{s} + 1)$$

Giving a spectral factor of

$$\Delta(\mathrm{s}) = \frac{\left(\mathrm{s}^2 + \sqrt{2}\mathrm{s} + 1\right)}{\mathrm{s}^2}$$

Now we use

$$\left[\frac{\mathrm{W(s)}\sigma_\xi^2\mathrm{W}(-\mathrm{s})}{\Delta(-\mathrm{s})}\right]_+ = \left[\frac{1}{\mathrm{s}^2(\mathrm{s}^2 - \sqrt{\mathrm{s}} + 1)}\right]_+$$

Expand using partial-fractions

$$\begin{aligned}\left[\frac{1}{\mathrm{s}^2(\mathrm{s}^2 - \sqrt{\mathrm{s}} + 1)}\right]_+ &= \left[\frac{\mathrm{A}}{\mathrm{s}^2} + \frac{\mathrm{B}}{\mathrm{s}} + \frac{\mathrm{Cs} + \mathrm{D}}{\mathrm{s}^2 - \sqrt{\mathrm{s}} + 1}\right]_+ \\ &= \left[\frac{1}{\mathrm{s}^2} + \frac{\sqrt{2}}{\mathrm{s}} + \frac{1 - \sqrt{2}s}{\mathrm{s}^2 - \sqrt{\mathrm{s}} + 1}\right]_+\end{aligned}$$

Since the right-most term $\frac{1-\sqrt{2}s}{s^2-\sqrt{s}+1}$ is noncausal (actually it is unstable, but an unstable system is also a stable un-causal system whose impulse-response goes backwards in time!), we reject it and take only the other two terms.

$$\left[\frac{W(s)\sigma_\xi^2 W(-s)}{\Delta(-s)}\right]_+ = \left[\frac{1}{s^2} + \frac{\sqrt{2}}{s}\right]$$
$$= \frac{s^2 + \sqrt{2}s^2}{s^3}$$

The reason we rejected the last term is because its impulse-response goes backwards in time. From our point of view, it looks just like an unstable system. An unstable system is stable if time runs backwards! We were looking at the noncausal part and we rejected it.

Substitute all terms into

$$H(s) = \left[\frac{W(s)\sigma_\xi^2 W(-s)}{\Delta(-s)}\right]_+ \frac{1}{\Delta(s)}$$

and our Wiener filter becomes

$$H(s) = \frac{1 + \sqrt{2}s}{s^2 + \sqrt{2}s + 1}$$

Examining the poles, we see this is a type of Butterworth filter. Of course, the filter has only been calculated for the particular noise variances and will change if say the measurement noise variance is increased. The Bode-plot is shown in Fig. 12.10. Although the dc gain is unity, there is a zero which takes effect at 0.707 rad/s and gives a slight rise in gain before the second-order poles take over.

If we increase the measurement noise so that $\sigma_v^2 = 10, \sigma_\xi^2 = 1$ then

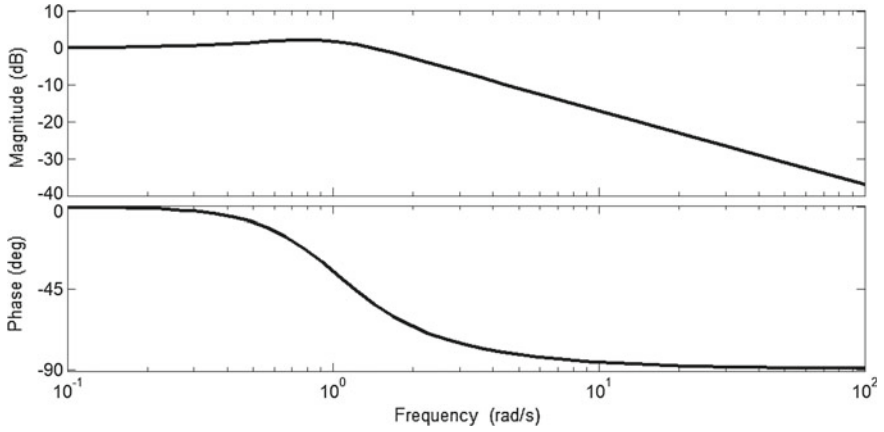


Fig. 12.10 Bode-plot of Wiener-filter $\sigma_v^2 = 1$

$$\begin{aligned}\Delta(s)\Delta(-s) &= W(s)\sigma_\xi^2 W(-s) + \sigma_v^2 \\ &= \frac{1}{s^4} + 10 \\ &= \frac{s^4 + 10}{s^4}\end{aligned}$$

Factorising the numerator

$$s^4 + 10 = (s^2 + 2.515s + 3.1623)(s^2 - 2.515s + 3.1623)$$

Giving a spectral factor of

$$\Delta(s) = \frac{(s^2 + 2.515s + 3.1623)}{s^2}$$

Now we use

$$\left[\frac{W(s)\sigma_\xi^2 W(-s)}{\Delta(-s)}\right]_+ = \left[\frac{1}{s^2(s^2 - 2.515s + 3.1623)}\right]_+$$

Continuing as before we find

$$\begin{aligned}\left[\frac{1}{s^2(s^2 - 2.515s + 3.1623)}\right]_+ &= \left[\frac{0.3162}{s^2} + \frac{0.2515}{s}\right] \\ &= \frac{0.3162s^2 + 0.2515s}{s^3}\end{aligned}$$

The Wiener-filter is now

$$H(s) = \frac{0.3162s + 0.2515}{s^2 + 2.515s + 3.1623}$$

The Bode-plot in Fig. 12.11 shows the zero has more effect as the additive noise increases. A very small bandpass characteristic is showing.

12.5.2 Discrete-Time Wiener-Filter

The continuous-time Wiener-filter is a useful theory for examining the properties of such filters, but nowadays the discrete-time case is more relevant due to the ease of implementation. In the discrete-time case the derivation needs the two-sided z-transform which was touched- upon in an earlier chapter. Without too much detail, we can avoid confusion by assuming that all causal system have z-transforms

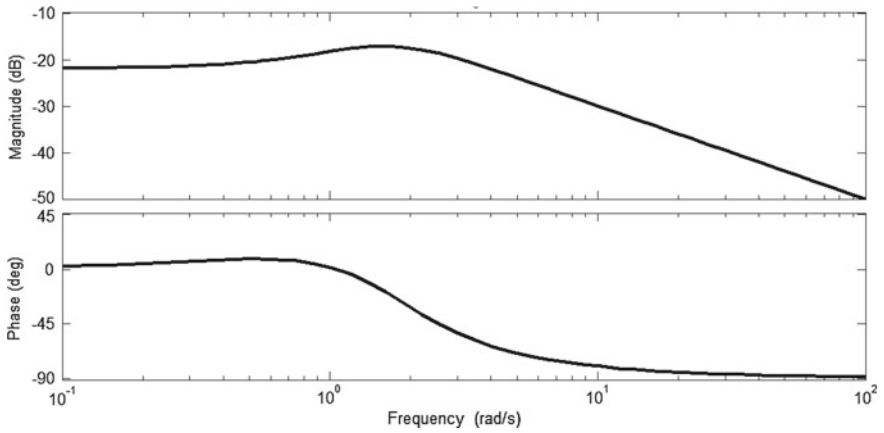


Fig. 12.11 Bode plot of Wiener filter with $\sigma_v^2 = 10$

in negative powers of z including zero, and all noncausal systems use positive powers of z. Hence, the following stable z-transfer-function $G(z^{-1}) = z^{-1} + 0.5z^{-2} + 0.25z^{-3} + 0.125z^{-4} + \ldots = \frac{z^{-1}}{1-0.5z^{-1}}$ will be considered causal and have an impulse response which exists for time $k > 0$. Its pole is at $z = 0.5$. However, a z-transfer-function $G(z) = z + 0.5z^2 + 0.25z^3 + 0.125z^4 \ldots = \frac{z}{1-0.5z}$ is considered noncausal and has an impulse-response which exists only for time k < 0 . Its pole is at $z = 2$. Each has a different region of convergence in terms of their derivation as an infinite summation. The poles of $G(z^{-1})$ lie within the unit-circle for a stable system, and the poles of $G(z)$ are mirror images which lie outside the unit-circle. When we multiply $G(z^{-1})G(z)$ we get a power-series which has *both* positive and negative powers of z and a term in z^0. This is called a *Laurent series*. We take the causal part of a Laurent series to include the term with coefficient z^0. The reason the power-series for $G(z^{-1})$ does not start at zero is because technically a digital system cannot respond instantaneously and needs a one-step delay minimum. In earlier chapters we used transfer-function such as $G(z) = \frac{z}{z-0.5}$. These were deemed to be causal, and they have impulse-responses starting at time zero. Likewise, for FIR systems we often define an FIR filter such as $G(z) = 1 - z^{-4}$, which also has an impulse-response starting at time zero. However, although mathematically this is the case, when we implement the difference equation we get $y_k = u_k - u_{k-4}$ and we see that at time zero $y_0 = u_0$. We can write this on paper, but when we implement it the computer must execute for one sampling interval before an output can occur. Therefore, in reality there is always one-step delay.

To avoid confusion, we use polynomial form instead. Write our signal as shown in Fig. 12.12

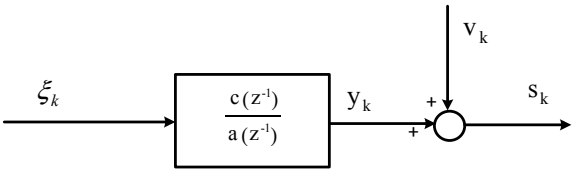


Fig. 12.12 Polynomial representation of random signal plus noise

$$y_k = \frac{c(z^{-1})}{a(z^{-1})}\xi_k \tag{12.33}$$

$$s_k = y_k + v_k \tag{12.34}$$

where

$$c(z^{-1}) = c_1 z^{-1} + c_2 z^{-2} + \ldots + c_n z^{-n} \tag{12.35}$$

$$a(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n} \tag{12.36}$$

and $a(z^{-1})$ has all its roots inside the unit circle. We define $a(z)$ as having its roots outside the unit-circle.

As a short cut we make $a(z^{-1}) \equiv a$ and $a(z) \equiv a^*$. Polynomial methods have proven popular in the past 30 years and are also an elegant way of solving several estimation and control problems. Equation (12.33) is also known as an *autoregressive-moving-average* model or ARMA for short. This pre-dates the engineering applications and was used in statistics for time-series analysis [33]. A pole-zero model to an engineer is ARMA to a statistician. Essentially the analysis is much the same, except engineers usually work in the areas of filtering and control, whereas statisticians work in areas such as forecasting and time-series analysis from a range of scientific disciplines. An added complication for engineers, is that more than often our application must run in real-time whereas statistical analysis can be done entirely off-line. The discrete-time Wiener-filter transfer function is found from [34].

$$H = \left[\frac{cc^*}{ad^*}\right]_+ \frac{a}{d}\left(\frac{\sigma_\xi^2}{\sigma_\varepsilon^2}\right) \tag{12.37}$$

where the minimum-phase polynomial d of order n satisfies the spectral factorization.

$$dd^*\sigma_\varepsilon^2 = aa^*\sigma_v^2 + cc^*\sigma_\xi^2 \tag{12.38}$$

We normalise the spectral-factor such that $d(0) = 1$. A simpler form of (12.37) exists by substituting from the spectral-factor (12.38).

$$\frac{cc^*}{aa^*}\sigma_\xi^2 = \frac{dd^*}{aa^*}\sigma_\varepsilon^2 - \sigma_v^2$$

This results in

$$H = \left[\left(\frac{dd^*}{aa^*}\sigma_\varepsilon^2 - \sigma_v^2\right)\frac{a^*}{d^*}\right]_+ \frac{a}{d}\left(\frac{1}{\sigma_\varepsilon^2}\right) \tag{12.39}$$

Which becomes

$$H = \left[\left(\frac{d}{a}\sigma_\varepsilon^2 - \sigma_v^2\frac{a^*}{d^*}\right)\right]_+ \frac{a}{d}\left(\frac{1}{\sigma_\varepsilon^2}\right) \tag{12.40}$$

Now

$$\left[\left(\frac{d}{a}\sigma_\varepsilon^2 - \sigma_v^2\frac{a^*}{d^*}\right)\right]_+ = \frac{d}{a}\sigma_\varepsilon^2 - \sigma_v^2 \tag{12.41}$$

Hence (12.40) becomes

$$H = 1 - \frac{a}{d}\left(\frac{\sigma_v^2}{\sigma_\varepsilon^2}\right) \tag{12.42}$$

In (12.41) above, we can expand in a power-series of positive powers of z

$$\left[\frac{a^*}{d^*}\right]_+ = \left[1 + \alpha_1 z + \alpha_2 z^2 + \ldots\right]_+ = 1$$

We also have that $\left[\frac{d}{a}\right]_+ = \frac{d}{a}$ is causal, as its power-series would have a zeroth power and only negative powers of z.

12.5.3 *Illustrative Example of Discrete-Time Wiener-Filter*

Suppose $c(z^{-1}) = z^{-1}, a(z^{-1}) = 1 - 1.6z^{-1} + 0.9z^{-2}, \sigma_\xi^2 = 1, \sigma_v^2 = 10$. We check that the polynomial a is stable, its roots have magnitude less than unity. The spectral-factor is found from

$$\begin{aligned} dd^*\sigma_\varepsilon^2 &= aa^*\sigma_v^2 + cc^*\sigma_\xi^2 \\ &= (1 - 1.6z^{-1} + 0.9z^{-2})(1 - 1.6z + 0.9z)x10 + 1 \end{aligned}$$

There are many methods available for solving such equations. We use the method of Cholesky factorisation in the MATLAB code below. There is a close relationship between polynomials and lower-triangular Toeplitz matrices, and this is exploited here.

MATLAB Code 12.1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Spectral Factorisation using LDL' Cholesky method
%Uses the Toeplitz matrix approach
%T.J.Moir 28-12-2018
%Finds polynomial d from dd*si=aa*sv+cc*sq
%Book example
%c=[1]; a=[1 -1.6 0.9]; sq=1;sv=10;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
output=('Signal Model is c/a')
c=input('Input c polynomial coefficients as a vector [...], eg [c0,c1] etc with square-
 brackets')
a=input('Input a polynomial coefficients as a vector [...], eg [a0,a1] etc with square-
 brackets')
n=length(a);
% m relatesto accuracy. Needs to be fairly big
m=20;
%Form A Toeplitz matrix from a polynomial
tr=zeros(1,m);
tr(1)=a(1);
nul=zeros(1,(m-length(a)));
tc=horzcat(a,nul);
A=toeplitz(tc,tr);
%%%%%%%%%%%%%%%%%%%%%
%Form C Toeplitz matrix from c polynomial
tr=zeros(1,m);
tr(1)=c(1);
nul=zeros(1,(m-length(c)));
tc=horzcat(c,nul);
C=toeplitz(tc,tr);
%%%%%%%%%%%%%%%%%%%
sq=input('Input driving noise variance for signal sq')
sv=input('Input additive white noise variance sv')
S=sq*C*C'+A*A'*sv;

[L,D]=ldl(S)

d=fliplr(L(m,:));
output=('Spectral factor coefficients of dpolynomial in ascending order)')
d(1:n)
%Innovationas Variance si
si=D(m,m)
```

The program returns

Spectral factor coefficients of d polynomial in ascending order)

ans =

1.0000 −1.2198 0.5653

si =

15.921

Substituting these values we get $d = (1 - 1.2198z^{-1} + 0.5653z^{-2})$, $\sigma_\varepsilon^2 = 15.921$, $\frac{\sigma_v^2}{\sigma_\varepsilon^2} = 0.6281$.

$$H = 1 - \frac{a}{d}\left(\frac{\sigma_v^2}{\sigma_\varepsilon^2}\right)$$

Or alternatively

$$\begin{aligned} H &= \frac{d - a\left(\frac{\sigma_v^2}{\sigma_\varepsilon^2}\right)}{d} \\ &= \frac{1 - 1.2198z^{-1} + 0.56526z^{-2} - (1 - 1.6z^{-1} + 0.9z^{-2}).0.6281}{1 - 1.2198z^{-1} + 0.5653z^{-2}} \\ &= \frac{0.3719 - 0.2148z^{-1}}{1 - 1.2198z^{-1} + 0.5653z^{-2}} \end{aligned}$$

The filter output can be written as

$$\widehat{y}_{k|k} = Hs_k$$

The notation $\hat{y}_{k|k}$ means the linear least-squares estimate of the signal y at time k given observations up to and including time k. It is possible to get a slightly worse estimate by finding the estimate with observations up to time $k - 1$. This we call $\hat{y}_{k|k-1}$ and gives us a marginally worse estimate of the signal [34]. It is known as a one-step ahead predicted estimate as opposed to ours.

The filter is simulated using MATLAB.

MATLAB Code 12.2

```
% Signal Model plus noise variances
c=[1]; a=[1 -1.6 0.9]; sq=1;sv=10;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Wiener Filter from calculation
num=[0.3719  -0.2148];
den=[1 -1.2198 0.5626];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Wiener filter simulation
% Use 100 points for simulation
t=[0:1:99]';

%Generate Random Noise Variance sq=1 driving c/a
rn=randn(100,1);
%Filter it

%Generate Signal using c/a filter with white-noise input
y=filter(c,a,rn);
%Re-seed wnoise generator
rng(100);
% Additive uncorrelated white-noise variance sv=10
rv=sqrt(sv)*randn(100,1);
% add white-noise
s=y+rv;
%Wiener filter  with s as input
yh=filter(num,den,s);
subplot(2,1,1)
plot(t,y,'-r',t,yh,'-.b')
legend('Signal','Estimate','Location','northwest')
title('Signal and Estimate')
xlabel({'Time in samples'})
subplot(2,1,2)
plot(t,s)
title('Signal plus Noise')
xlabel({'Time in samples'})
sy=var(y);
sn=var(rv);
output=('Signal to Noise ratio (dB) = ')
snr=10*log10(sy/sn)
```

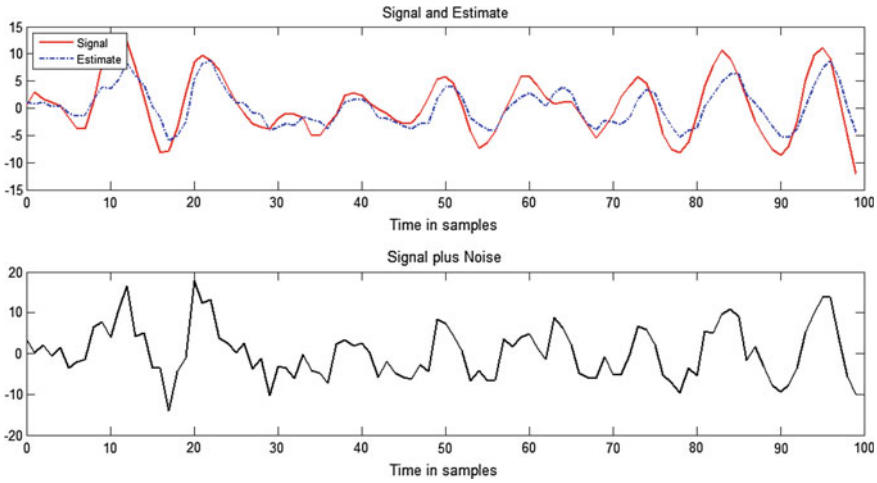


Fig. 12.13 Signal plus noise, true signal and estimate using discrete-time Wiener filter

It is important to note that the above filter is the instantaneous estimate or filtered estimate. A similar one-step predicted signal estimate [39] is found from $H_p = \frac{d-a}{d} = \frac{0.3802z^{-1}-0.3348z^{-2}}{1-1.2198z^{-1}+0.5653z^{-2}}$ with $\hat{y}_{k|k-1} = H_p s_k$. This estimate is slightly worse because there is less observational data available. It has data up to and including time $k-1$ instead of k. Note that the numerator has no term in z^0, but starts with a delay (Fig. 12.13).

12.6 State-Space Systems with Noise

Our descriptions of noisy systems have excluded problems with a control signal. In fact it is quite straight-forward to represent noisy systems in state-space, either in continuous or discrete-time. We assume, just as with the previous cases that there is a driving noise-term but also assume there is a control term and additive noise. Consider first the continuous-time case.

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) + \mathbf{D}\xi(t) \tag{12.43}$$

$$y(t) = \mathbf{C}\mathbf{x}(t) \tag{12.44}$$

$$s(t) = y(t) + v(t) \tag{12.45}$$

We assumed one in out and one output only, but this can be generalised to vector inputs and outputs. The noise driving the states $\xi(t)$ is a scalar. The vector $\boldsymbol{D}$ is sometimes omitted altogether and the noise term written as a vector units place. It

should not be confused with the feed through matrix used for systems with the same order of numerator and denominator.

An interesting model arises when we take Laplace transforms of (12.43). We get

$$\mathbf{x}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}u(s) + (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{D}\xi(s) \tag{12.46}$$

Using (12.44)

$$\begin{aligned} y(s) &= \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}u(s) + \mathbf{C}(s\mathbf{I} - A)^{-1}\mathbf{D}\xi(s) \\ &= W(s)u(s) + W_0(s)\xi(s) \end{aligned} \tag{12.47}$$

We have defined the system transfer-function in the usual way

$$W(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} \tag{12.48}$$

However, we have an extra term which we call the disturbance term found from

$$W_0(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{D} \tag{12.49}$$

We can then write the noisy measurement as

$$s(t) = W(s)u(s) + W_0(s)\xi(s) + v(t) \tag{12.50}$$

Doing the same for the discrete-time case

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}u_k + \mathbf{E}\xi_k \tag{12.51}$$

$$y_k = \mathbf{H}\mathbf{x}_k \tag{12.52}$$

$$s_k = y_k + v_k \tag{12.53}$$

The relationship to continuous-time is given by

$$\mathbf{F} = e^{\mathbf{A}T_s} \tag{12.54}$$

$$\mathbf{G} = \int_0^{T_s} e^{\mathbf{A}\sigma}\mathbf{B}\,d\sigma \tag{12.55}$$

$$\mathbf{E} = \int_0^{T_s} e^{\mathbf{A}\sigma}\mathbf{D}\,d\sigma \tag{12.56}$$

These can all be found with a power-series expansion as in a previous chapter. We define the system z-transfer-function

$$\mathrm{W}(\mathrm{z}) = \mathbf{H}(\mathrm{z}\mathbf{I} - \mathbf{F})^{-1}\mathbf{G} \tag{12.57}$$

An extra term which we call the disturbance term is found from

$$\mathrm{W}_0(\mathrm{z}) = \mathbf{H}(\mathrm{z}\mathbf{I} - \mathbf{F})^{-1}\mathbf{E} \tag{12.58}$$

We can then write the noisy measurement as

$$\mathrm{s_k} = \mathrm{W}(\mathrm{z})\mathrm{u_k} + \mathrm{W}_0(\mathrm{z})\xi_\mathrm{k} + \mathrm{v_k} \tag{12.59}$$

Now draw a block-diagram of the stochastic system in Fig. 12.14. We see that it appears as a LTI system with an additive coloured-noise disturbance $W_0(z)$. We can denote this coloured-noise term as

$$\eta_\mathrm{k} = \mathrm{W}_0(\mathrm{z})\xi_\mathrm{k} \tag{12.60}$$

Positive-definite Covariance matrices

When the control-signal, which is deterministic, is zero, we get back to our model we used in Wiener filtering of coloured-noise plus additive white-noise. We also note that there is an alternative description which is often used whereby the driving noise is taken as a vector and not a scalar. Also, there can be more than one driving noise due to the nature of the problem itself.

$$\mathbf{x}_{\mathrm{k}+1} = \mathbf{F}\mathbf{x}_\mathrm{k} + \mathbf{G}\mathrm{u_k} + \boldsymbol{\xi}_\mathrm{k} \tag{12.61}$$

We see that the ***E*** vector is no longer needed. However, if noise is written as a column vector

$$\boldsymbol{\xi}_\mathrm{k} = \begin{bmatrix} \xi_\mathrm{k}^1 & \xi_\mathrm{k}^2 & . & . & \xi_\mathrm{k}^\mathrm{n} \end{bmatrix}^\mathrm{T}$$

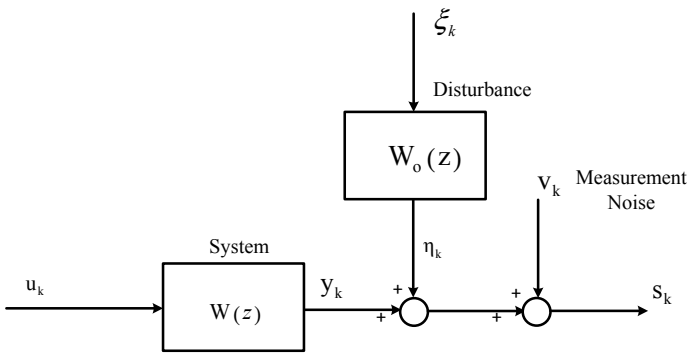


Fig. 12.14 Block-diagram of discrete-time stochastic system

then we can no longer define its variance. Instead we have a matrix of variances. We usually call this the covariance matrix. The covariances matrix is defined for a vector of white-noise terms as

$$\mathrm{E}\left[\boldsymbol{\xi}_{\mathrm{k}}\boldsymbol{\xi}_{\mathrm{k}-\ell}^{\mathrm{T}}\right] = \mathbf{Q}\delta_{\ell} \tag{12.62}$$

This matrix is usually diagonal, since individual white-noise terms are unlikely to be correlated with each other. In (12.51), we can also see that this produces a vector $\boldsymbol{E}\xi_k$ from a scalar ξ_k. (note $\mathbf{E}$ is not expectation here but a matrix) We can define its combined covariance matrix as

$$\mathrm{E}\left[\mathbf{E}\xi_{\mathrm{k}}\mathbf{E}^{\mathbf{T}}\xi_{\mathrm{k}}\right] = \mathbf{E}\mathbf{E}^{\mathbf{T}}\boldsymbol{\sigma}_{\xi}^{2}\delta_{\ell} \tag{12.63}$$

A covariance matrix must be *positive-definite*. We define positive-definite for a matrix $\boldsymbol{Q}$ as $\boldsymbol{Q} > 0$. A generalisation of a scalar being positive, negative or zero means we can have *positive-definite, negative-definite* or *positive semi-definite*. A positive-definite matrix is one where its *leading* principle minors are all positive. A diagonal matrix of variances must always be positive-definite since all elements in the diagonal are positive. Variance cannot be negative. For example, the matrix

$$\mathrm{A} = \begin{bmatrix} \mathrm{a}_{11} & \mathrm{a}_{12} & \mathrm{a}_{13} \\ \mathrm{a}_{21} & \mathrm{a}_{22} & \mathrm{a}_{23} \\ \mathrm{a}_{31} & \mathrm{a}_{32} & \mathrm{a}_{33} \end{bmatrix}$$

Its principle minors are

$$|\mathrm{A}|, \begin{vmatrix} \mathrm{a}_{11} & \mathrm{a}_{12} \\ \mathrm{a}_{21} & \mathrm{a}_{22} \end{vmatrix}, \begin{vmatrix} \mathrm{a}_{22} & \mathrm{a}_{23} \\ \mathrm{a}_{32} & \mathrm{a}_{33} \end{vmatrix}, \begin{vmatrix} \mathrm{a}_{11} & \mathrm{a}_{13} \\ \mathrm{a}_{31} & \mathrm{a}_{33} \end{vmatrix}, \mathrm{a}_{11}, \mathrm{a}_{22}, \mathrm{a}_{33}$$

Its *leading* principle minors are

$$\mathrm{a}_{11}, \begin{vmatrix} \mathrm{a}_{11} & \mathrm{a}_{12} \\ \mathrm{a}_{21} & \mathrm{a}_{22} \end{vmatrix}, |\mathrm{A}|$$

These must be all positive for a positive-definite matrix. If say one leading principle minor is zero and the rest positive, then it is *positive semi-definite*. If all are negative, then the matrix is *negative-definite* and so on.

12.7 ARMAX Models

Finally, we look at a different kind of model, the ARMAX model. This stands for exogenous input autoregressive-moving-average model. The idea comes from (12.59), namely $s_k = W(z)u_k + W_0(z)\xi_k + v_k$. We can represent each of the transfer-function with ratios of polynomials instead. Thus

$$s_k = \frac{z^{-d}b(z^{-1})}{a(z^{-1})}u_k + \frac{c(z^{-1})}{a(z^{-1})}\xi_k + v_k \tag{12.64}$$

where the polynomials are defined as

$$c(z^{-1}) = c_1 z^{-1} + c_2 z^{-2} + \ldots + c_n z^{-n} \tag{12.65}$$

$$\begin{aligned} a(z^{-1}) &= 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n} \\ b(z^{-1}) &= b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-m} \end{aligned} \tag{12.66}$$

And d is an integer time-delay which for many process-control systems is an important facet of the model. Even in the simplest of models the delay should be at least one-step. Usually the degree of b is less than or equal to the degree of a, so that $m \leq n$. The b polynomial can also be nonminimum phase. The c polynomial is usually minimum phase for many control-problems but for certain equalisation and filtering problems could also be nonminimum-phase. It is usual to combine the additive noise with the disturbance term and treat them as a combined system. We can write

$$\frac{c(z^{-1})}{a(z^{-1})}\xi_k + v_k = \frac{d(z^{-1})}{a(z^{-1})}\varepsilon_k \tag{12.67}$$

where the minimum-phase polynomial d of order n satisfies the spectral factorization

$$dd^*\sigma_\varepsilon^2 = aa^*\sigma_v^2 + cc^*\sigma_\xi^2 \tag{12.68}$$

We normalise the spectral-factor such that $d(0) = 1$. The term on the RHS of (12.67) is then known as an innovations model, where ε_k is a white-noise signal. We used a similar idea in discrete-time Wiener-filtering. Our new ARMAX model then becomes

$$s_k = \frac{z^{-d}b(z^{-1})}{a(z^{-1})}u_k + \frac{d(z^{-1})}{a(z^{-1})}\varepsilon_k \tag{12.69}$$

This is a useful model for stochastic systems that avoids using state-space altogether. It has the added advantage that the parameters of the polynomials can be estimated using least-squares methods in real-time, hence giving the possibility of system identification and control in an adaptive combined format. For systems with multiple inputs and outputs, the model can be extended so that the outputs and inputs are vectors, and the polynomials are replaced with polynomial-matrices. That is, a polynomial whose coefficients are matrices rather than scalars. This method has proven valuable in many aspects of adaptive or self-tuning control-systems, particularly with process-control systems that currently use PID controllers.

Chapter 13
Kalman Filtering

The Kalman filter is without doubt, one of the most influential theoretical advancements in signal-processing and control-systems in the latter half of the 20th century. In essence, it is no more than an observer for the states of a dynamic system contaminated with noise. However, because it uses state-space, it has applications in nonlinear estimation and systems that are time-varying. It has been used in signal-processing applications and equally employed in control-systems design. In signal processing it has been applied to almost anything that can be described as a random system. It has applications from image processing, speech-processing, system identification, optimal-control and inertial guidance. It was inertial guidance that first made the Kalman filter popular, having been implemented on many missile systems and even programmed into the lunar landing computer for the Apollo mission to the moon. Kalman's first paper [35] was on the discrete-time case, which is quite surprising considering that nearly all systems were analogue in that time-period. This was followed up a year later with the continuous-time case [36], co-authored with Bucy. It has become known as the Kalman-Bucy filter. Even more interesting from a historic point of view is that the Kalman filter is a special case of work by a Soviet mathematician Stratonovich [37] who considered a more general nonlinear problem. We begin however with the continuous-time case.

13.1 Kalman-Bucy Filter

Consider a state-space stochastic system which has *completely observable* states.

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) + \mathbf{D}\xi(t) \tag{13.1}$$

$$y(t) = \mathbf{C}\mathbf{x}(t) \tag{13.2}$$

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_13

$$s(t) = y(t) + v(t) \tag{13.3}$$

More often the scalar white process noise term in (13.1) is replaced by a vector of the same dimension as the number of states.

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) + \boldsymbol{\xi}(t) \tag{13.4}$$

The positive-definite covariance matrix is defined as $\mathbf{Q} = E\left[\boldsymbol{\xi}(t)\boldsymbol{\xi}^T(t)\right] > 0$. The observation (13.3) can be scalar or vector based depending on the problem. It is shown as scalar, but in the more general case the measurement noise is shown as a vector, uncorrelated with the process noise and with covariance matrix $\mathbf{R} = E[\mathbf{v}(t)\mathbf{v}^T(t)] > 0$. In the scalar measurement case above, the measurement noise has a variance $\boldsymbol{\sigma}_v^2$ and we substitute with a slight abuse of notation $\boldsymbol{\sigma}_v^2$ for $\mathbf{R}$. All noise terms are assumed to have zero-mean. The convention is to name the driving noise as process-noise and additive noise as measurement- noise. We denote the linear least-squares estimate of the state at time t given information up to time t as $\hat{\mathbf{x}}(t|t)$. Sometimes it is written as $\hat{\mathbf{x}}(t_1|t)$ for the more general case. In such cases if $t_1 > t$ then we have a prediction problem and if $t_1 < t$ we have a smoothing problem. These cases for transfer-function descriptions had already been considered by Norbert Wiener. For control-system purposes a smoother would give a better estimate of the state (hence the name), but it introduces a time-delay which is detrimental to closed-loop stability. Prediction gives a more noisy estimate than filtering. For example in the discrete-time case that follows, we can use the one-step ahead prediction of the state. Define an error-vector

$$\mathbf{e}(t) = (\mathbf{x}(t) - \hat{\mathbf{x}}(t|t)) \tag{13.5}$$

The Kalman-filter minimises the following cost-function

$$J = E\left[\mathbf{e}^T(t)\mathbf{e}(t)\right] \tag{13.6}$$

This is a scalar cost and is the sum of individual mean-square errors in the error vector.

The solution to this optimisation problem is the linear-dynamic system (the filter itself)

$$\begin{aligned} \hat{\dot{\mathbf{x}}}(t|t) &= \mathbf{A}\hat{\mathbf{x}}(t|t) + \mathbf{B}u(t) + \mathbf{K}(t)[y(t) - \mathbf{C}\hat{\mathbf{x}}(t|t)] \\ \hat{y}(t|t) &= \mathbf{C}\hat{\mathbf{x}}(t|t) \end{aligned} \tag{13.7}$$

where the Kalman gain vector (or matrix depending on whether the problem is SISO or multivariable) is given by

$$\mathbf{K}(\mathrm{t}) = \mathbf{P}(\mathrm{t})\mathbf{C}^{\mathrm{T}}\mathbf{R}^{-1} \tag{13.8}$$

The matrix $\mathbf{P}(t)$ is the symmetrical *positive definite* estimation error covariance matrix

$$\mathbf{P}^{\mathrm{T}}(\mathrm{t}) = \mathbf{P}(\mathrm{t}) = \mathrm{E}\left[\mathbf{e}(\mathrm{t})\mathbf{e}^{\mathrm{T}}(\mathrm{t})\right] \tag{13.9}$$

Which is found from a matrix Riccati equation

$$\dot{\mathbf{P}}(\mathrm{t}) = \mathbf{A}\mathbf{P}(\mathrm{t}) + \mathbf{P}(\mathrm{t})\mathbf{A}^{\mathrm{T}} - \mathbf{P}(\mathrm{t})\mathbf{C}^{\mathrm{T}}\mathbf{R}^{-1}\mathbf{C}\mathbf{P}(\mathrm{t}) + \mathbf{Q} \tag{13.10}$$

Subject to the initial condition

$$\mathbf{P}(0) = \mathbf{P}_0 \tag{13.11}$$

Examining (13.7) shows that the Kalman filter is a model of the system with a feedback path around it and a gain vector $\mathbf{K}(t)$ acting on the error signal. This is better illustrated in Fig. 13.1.

In most cases it is not possible to find a closed-form solution to (13.10) and a numerical solution is found instead. However, for some smaller examples the steady-state solution can be found instead. This is known as the *algebraic* Riccati equation with steady-state solution $\boldsymbol{P}$.

$$\mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^{\mathrm{T}} - \mathbf{P}\mathbf{C}^{\mathrm{T}}\mathbf{R}^{-1}\mathbf{C}\mathbf{P} + \mathbf{Q} = \mathbf{0} \tag{13.12}$$

The steady-state solution is the Wiener-filter for the states of the system. In fact for stationary noise and an LTI system, the Riccati equation converges to the steady-state solution in any case. Solving (13.12) is only possible for the simplest of cases, since the algebra quickly gets out of hand.

The transfer-function of the Kalman-filter is found by taking Laplace-transforms of (13.7) in steady-state, with $\mathbf{K}(\mathrm{t}) = \mathbf{K}$.

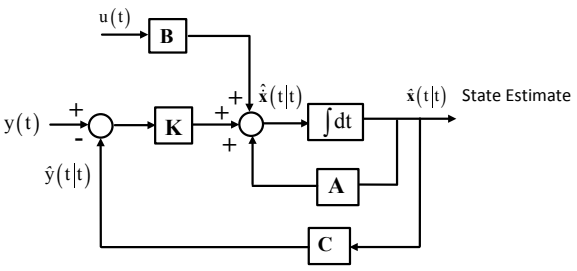


Fig. 13.1 Block diagram of Kalman-Bucy filter

$$\hat{\mathbf{x}}(s) = (s\mathbf{I} - \mathbf{A} + \mathbf{KC})^{-1}\mathbf{K}y(s) + (s\mathbf{I} - \mathbf{A} + \mathbf{KC})^{-1}\mathbf{B}u(s)$$
$$\hat{y}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A} + \mathbf{KC})^{-1}\mathbf{K}y(s) + \mathbf{C}(s\mathbf{I} - \mathbf{A} + \mathbf{KC})^{-1}\mathbf{B}u(s)$$

13.1.1 Kalman-Filter Example Continuous-Time

Consider the same example from Chap. 12. This is double-integrated white-noise as shown in Fig. 13.2.

We define the state-variables as

$$x_1(t) = y(t)$$
$$x_2(t) = \dot{y}(t)$$

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix}\xi(t)$$
$$y(t) = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix}$$

We have $\sigma_v^2 = 1, \sigma_\xi^2 = 1$ so that $\mathbf{Q} = \mathbf{D}\mathbf{D}^T\sigma_\xi^2 = \begin{bmatrix}0\\ 1\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix}\sigma_\xi^2 = \begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}$. Also note that we must replace the measurement noise covariance matrix $\mathbf{R}$ with the scalar value of unity.

Define $\mathbf{P} = \begin{bmatrix}p_1 & p_2\\ p_2 & p_3\end{bmatrix} = \mathbf{P}^T$ and substitute into the steady-state Riccati equation.

$$\mathbf{AP} + \mathbf{PA}^T - \mathbf{PC}^T\mathbf{R}^{-1}\mathbf{CP} + \mathbf{Q} = \mathbf{0}$$

(Be aware that some texts give the algebraic Riccati equation as $\mathbf{AP} + \mathbf{PA}^T - \mathbf{PC}^T\mathbf{R}^{-1}\mathbf{CP} + \mathbf{DQD}^T = \mathbf{0}$ which includes the $\mathbf{D}$ matrix. Here we calculate it as being combined with the noise. The result is the same for both cases.)

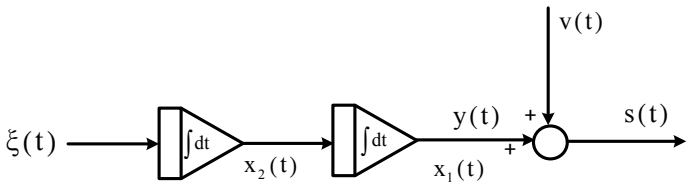


Fig. 13.2 Illustrative example

$$\begin{bmatrix} p_2 & p_3 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} p_2 & 0 \\ p_3 & 0 \end{bmatrix} - \begin{bmatrix} p_1 & p_2 \\ p_2 & p_3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 & p_2 \\ p_2 & p_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 2p_2 & p_3 \\ p_3 & 0 \end{bmatrix} - \begin{bmatrix} p_1^2 & p_1 p_2 \\ p_1 p_2 & p_2^2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 2p_2 - p_1^2 & p_3 - p_1 p_2 \\ p_3 - p_1 p_2 & -p_2^2 + 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Solving the three equations from the matrix above we have

$$-p_2^2 + 1 = 0, p_2 = 1$$
$$2p_2 - p_1^2 = 0, p_1 = \sqrt{2}$$
$$p_3 - p_1 p_2, p_3 = \sqrt{2}$$

$$\mathbf{P} = \begin{bmatrix} p_1 & p_2 \\ p_2 & p_3 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 1 \\ 1 & \sqrt{2} \end{bmatrix} > 0$$

We check that this is positive-definite. All leading principle minors are positive. The steady-state Kalman-gain vector is then

$$\mathbf{K} = \mathbf{P}\mathbf{C}^{\mathbf{T}}\mathbf{R}^{-1} = \begin{bmatrix} \sqrt{2} \\ 1 \end{bmatrix}$$

We can solve with MATLAB to check the result.

MATLAB Code 13.1

```
% Kalman-Bucy filter
%Enter state-space description
A=[0 1;0 0]; D=[0 1]';C=[1 0];
sys=ss(A,D,C,0)
Q=1;R=1
[kest,K,P]=kalman(sys,Q,R,0)
Ak=(A-K*C);
[num,den]=ss2tf(Ak,K,C,0)
%Transfer-function of Kalman-Bucy filter
gk=tf(num,den)
```

It returns

```
K =

   1.4142
   1.0000


P =

   1.4142   1.0000
   1.0000   1.4142


gk =

    1.414 s + 1
 -----------------
 s^2 + 1.414 s + 1
```

This corresponds to the Wiener-filter we calculated in Chap. 12.

13.2 Discrete-Time Kalman-Filter

Although the continuous-time version of the optimal state-estimator is of some theoretical importance, for implementation we need to examine the discrete-time version. Consider a discrete-time stochastic system:

Doing the same for the discrete-time case, we assume the following system is completely observable:

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}u_k + \mathbf{E}\xi_k \tag{13.13}$$

$$y_k = \mathbf{H}\mathbf{x}_k \tag{13.14}$$

$$s_k = y_k + v_k \tag{13.15}$$

Define the estimation error vector vector

$$\mathbf{e}_k = \left(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1}\right) \tag{13.16}$$

The discrete-time Kalman-filter minimises the following cost-function

$$J = E\left[\mathbf{e}_k^T \mathbf{e}_k\right] \tag{13.17}$$

This is a scalar cost and is the sum of individual mean-square errors in the error vector. We observe that the error in (13.16) is in fact the one-step prediction error. It is still usually referred to as a filter even though it is a one-step predictor. We could also have used $\mathbf{e}_k = \left(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k}\right)$ instead.

The solution to this optimisation problem is the linear-dynamic system

$$\begin{aligned} \hat{\mathbf{x}}_{k+1|k} &= \mathbf{F}\hat{\mathbf{x}}_{k|k-1} + \mathbf{G}u_k + \mathbf{K}_k\left[y_k - \mathbf{C}\hat{\mathbf{x}}_{k|k-1}\right] \\ \hat{y}_{k|k-1} &= \mathbf{H}\hat{\mathbf{x}}_{k|k-1} \end{aligned} \tag{13.18}$$

The noise covariance matrices have their usual meaning and are defined as $\boldsymbol{Q}$ for process-noise and $\boldsymbol{R}$ for measurement-noise.

A matrix $\boldsymbol{P}_{k|k-1} = E\left[\boldsymbol{e}_k \boldsymbol{e}_k^T\right]$ where $\boldsymbol{e}_k = \left(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1}\right)$, is defined as the symmetric positive-definite error-covariance matrix. This form is usually referred to as the *a priori* error-covariance matrix due to the fact that there is a second one $\boldsymbol{P}_{k|k}$ (associated with filtering rather than one-step prediction) known as the *a posteriori* error-covariance matrix and uses the error $\boldsymbol{e}_k = \left(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k}\right)$.

This is all a bit confusing with different $\boldsymbol{P}$ matrices, and so a much simpler implementation is shown below which works well in practice. We drop the conditional expectations on the $\boldsymbol{P}$ matrix and simply refer to it as $\boldsymbol{P}_k$. We implement the loop shown below until the gain converges:

Initialise $\mathbf{P}_0 = \alpha\mathbf{I}$ for some positive constant α

$$
\begin{aligned}
&\{\\
&\mathbf{K}_k = \mathbf{F}\mathbf{P}_k\mathbf{H}^\mathbf{T}(\mathbf{R} + \mathbf{H}\mathbf{P}_k\mathbf{H}^\mathbf{T})^{-1}\\
&\mathbf{P}_{k+1} = \mathbf{F}\mathbf{P}_k\mathbf{F}^\mathbf{T} + \mathbf{Q} - \mathbf{K}_k\mathbf{H}\mathbf{P}_k\mathbf{F}^\mathbf{T}\\
&\hat{\mathbf{x}}_{k+1|k} = \mathbf{F}\hat{\mathbf{x}}_{k|k-1} + \mathbf{G}u_k + \mathbf{K}_k\left[y_k - \mathbf{C}\hat{\mathbf{x}}_{k|k-1}\right]\\
&\hat{y}_{k|k-1} = \mathbf{H}\hat{\mathbf{x}}_{k|k-1}\\
&\}
\end{aligned}
$$

With todays computing power, such equations are relatively easy to compute. However, it is still an overhead to have to constantly compute the Riccati equation each iteration of a real-time loop. For stationary noise and a LTI system, this equation will converge to a steady-state value. Once computed we can then implement the steady-state Kalman-filter in say a real-time loop having computed the gain-vector off-line. (assuming the model of the system is known and the noise covariance matrices). More than often (13.18) is implemented with a constant Kalman gain vector. In this form it is really a discrete-time Wiener filter. However, since Wiener or Kolmogorov (who discovered the discrete-time estimator) never researched state-estimation, it is a fair assumption to still call it a Kalman filter. The algebraic or steady-state Riccati equation is seldom solved analytically but is found from

$$\mathbf{P} = \mathbf{F}\mathbf{P}\mathbf{F}^\mathbf{T} + \mathbf{Q} - \mathbf{F}\mathbf{P}\mathbf{H}^\mathbf{T}(\mathbf{R} + \mathbf{H}\mathbf{P}\mathbf{H}^\mathbf{T})^{-1}\mathbf{H}\mathbf{P}\mathbf{F}^\mathbf{T} \tag{13.19}$$

The Kalman gain vector we find in steady-state is for a predictor (i.e. (13.18))

$$\mathbf{K} = \mathbf{F}\mathbf{P}\mathbf{H}^\mathbf{T}(\mathbf{R} + \mathbf{H}\mathbf{P}\mathbf{H}^\mathbf{T})^{-1} \tag{13.20}$$

The other form of the Kalman filter extends this to a filter where observations are available up to and including time k and not $k - 1$.

$$
\begin{aligned}
\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{G}u_k + \mathbf{K}^f\left[y_k - \mathbf{C}\hat{\mathbf{x}}_{k|k-1}\right]\\
\hat{y}_{k|k-1} &= \mathbf{H}\hat{\mathbf{x}}_{k|k-1}
\end{aligned}
\tag{13.21}
$$

where in steady-state

$$\mathbf{K}^f = \mathbf{P}\mathbf{H}^\mathbf{T}(\mathbf{R} + \mathbf{H}\mathbf{P}\mathbf{H}^\mathbf{T})^{-1} \tag{13.22}$$

For this method we require $\hat{\mathbf{x}}_{k|k-1}$ before we can update to get the update. This is a predictor update to a filter, a correction term which gives slightly smaller estimation error but is more complicated to implement. Usually we employ only the predictor state-estimator.

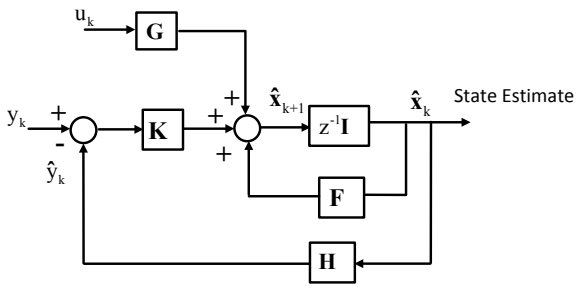


Fig. 13.3 Discrete-time Kalman-filter (one-step predictor)

The stationary Kalman-filter is shown in block-diagram form in Fig. 13.3. It is nothing more than an observer, but the poles are not placed, they are found by optimisation methods.

13.2.1 Illustrative Example for Discrete-Time Kalman Filter

We use the same example as used for the discrete-time Wiener filter in Chap. 12, namely in polynomial form, $c(z^{-1}) = z^{-1}, a(z^{-1}) = 1 - 1.6z^{-1} + 0.9z^{-2}, \sigma_{\xi}^2 = 1, \sigma_{v}^2 = 10.$ Converting to state-space

$$\begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.9 & 1.6 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \xi_k$$

$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix}$$

Now we find the Kalman gain vector by programming the equations from the loop above. MATLAB has its own function to find the gain vector, so we compare the two results. As the loop executes, we simulate the noisy dynamic system and feed the output to a simulated discrete-rime Kalman filter. The code also calculates the transfer-function of the Kalman filter.

MATLAB Code 13.1

```
% Finds Kalman Filter Gain and closed-loop transfer-function.
%Plots states and estimates
x0=[0;0] % Initial conditions on state vector
xh0=[0;0]; % Init conditions for discrete-time  KF
y=zeros(200,1);
s=zeros(200,1);
x1=zeros(200,1);
x2=zeros(200,1);
xh1=zeros(200,1);
xh2=zeros(200,1);
t=[1:1:200];
% Define the system
N = 200; % number of time steps
F = [0, 1; -0.9 1.6]; % system matrix - state
E = [0 1]'; % system process noise vector - input
H = [1 0]; % observation matrix
Q = E*E'*1; %Process noise covariance
 R=10    %Measurement noise variance
% Initialize the covariance matrix
 P = [10, 0; 0, 10]; % Covariance for initial state error
%Generate Random Noise Variance sq=1 driving process
zeta=randn(200,1);
%re-seed noise generator
rng(100);
% Additive uncorrelated white-noise variance sv=10
rv=sqrt(10)*randn(200,1);
zz=var(rv)
% Loop through and perform the Kalman filter equations recursively
x=x0;
xh=xh0;
for k = 1:N
% State outputs of system, no control term G=0;
x=F*x+E*zeta(k);
y(k)=H*x;
s(k)=y(k)+rv(k);
% Store states for plotting later
x1(k)=x(1);
x2(k)=x(2);
%Update Kalman gain vector - one step ahead version
K=F*P*H'/(H*P*H'+R);
% Update the covariance
P = F*P*F' + Q-K*H*P*F';
 %Kalman Filter
 xh=F*xh+K*[s(k)-H*xh];
 %Store estimated states for plotting
 xh1(k)=xh(1);
```

```
 xh2(k)=xh(2);
end
% Check with MATLAB result. Should be the same Kalman gain vector and P
sysd=ss(F,E,H,0,1);
[kest,kd,p2]=kalman(sysd,1,10,0)
%Display both for comparison
kd
K
%Find Closed-loop Kalman filter Transfer-function
Fc=F-K*H;
[num,den]=ss2tf(Fc,K,H,0,1)
%Transfer function of Steady-state Kalman filter
gkfz=tf(num,den,1)
%Now plot the states and their estimates
subplot(2,1,1)
plot(t,x1,'-r',t,xh1,'-.b')
subplot(2,1,1)
legend('State 1','State 1 estimate','Location','northwest')
title('State 1 and KF Estimate')
xlabel({'Time in samples'})
subplot(2,1,2)
plot(t,x2,'r',t,xh2,'-.b')
legend('State 2','State 2 estimate','Location','northwest')
title('State 2  and KF Estimate')
xlabel({'Time in samples'})
```

MATLAB returns

```
kd =

   0.3802
   0.2736

K =
```

```
   0.3802
   0.2736
```

Where the second result is the MATLAB function result and the first is our code. It then produces the z-transfer-function of the filter.

```
gkfz =

   0.3802 z - 0.3347
 ---------—--------
 z^2 - 1.22 z + 0.5653
```

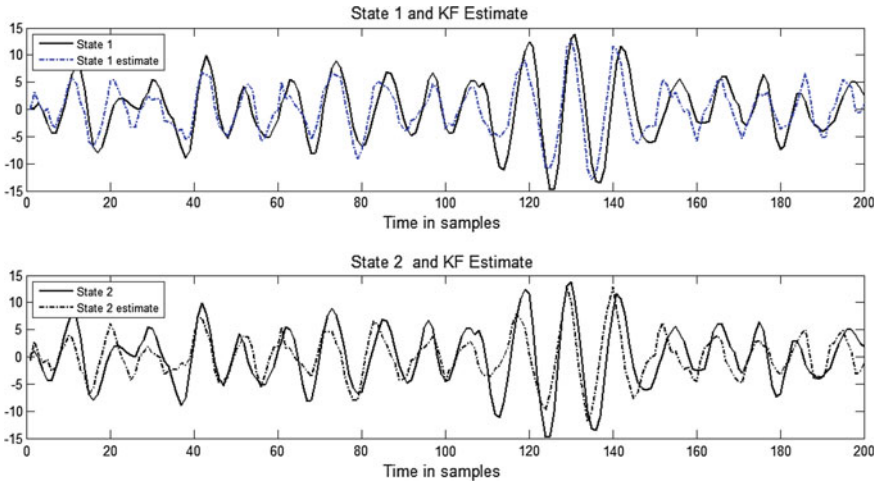


Fig. 13.4 States and estimates for discrete-time Kalman filter example

Examining this result, if we compare with the Wiener-filter result we see the numerator differs. This is because the main result of the Wiener-filter was a filter based on observations up to and including time k. If we now compare with the one-step predictor we find the two are identical.

The states and their estimates are shown in Fig. 13.4.

13.3 The Kalman-Filter for Inertial Measurements

Kalman-filters have had numerous applications since the original papers were published, but perhaps the earliest were applications to the guidance of missiles. Nowadays that application is still there, but we can add the *Inertial Measurement Unit (IMU)* to the list. These are small solid-state devices which have built in accelerometers and gyros acting in three-axis each. They are commonly found in every smart-phone and increasingly in quadcopters and robotics. They can find the orientation of a device. In the case of a phone or mobile computer, only the orientation is needed. For our control purposes we use the estimated angle to control the pitch or roll of a balancing robot or a machine in flight. The obvious question we ask is: *Why do we need both a Gyro and an Accelerometer to estimate angular position?* Surely if we have say acceleration, then we need only integrate twice and get position, and if we have velocity from a Gyro we can integrate once and get position. The answer is firstly integration in itself is a risky business. Any small amount of dc will cause a huge error in the output reading. Secondly, an

accelerometer is very noisy at high-frequencies but ok at low-frequencies. Thirdly, a gyro gives good reading at high frequencies but drifts over time. Some form of fusion is in order.

The accelerometer sensor gives acceleration in all three axis (x, y, z). Therefore some trigonometry is necessary to convert to the angle we need (pitch or roll). This is usually an arctan function and is shown in the next chapter, where one is implemented in real-time. We assume we can measure the noisy pitch or roll angle from the accelerometer after some minor calculations. The gyro gives us angular velocity and must be integrated to get position. This we cannot do directly without large offsets forming. Therefore, we measure gyro velocity also with additive noise. We model the two combined thus

$$\begin{aligned}
\begin{bmatrix} \dot{\theta}_p(t) \\ \dot{\omega}_p(t) \end{bmatrix} &= \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_p(t) \\ \omega_p(t) \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \xi_g(t) \\ \xi_a(t) \end{bmatrix} \\
\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_p(t) \\ \omega_p(t) \end{bmatrix} = \begin{bmatrix} \theta_p(t) \\ \omega_p(t) \end{bmatrix} \\
\begin{bmatrix} s_1(t) \\ s_2(t) \end{bmatrix} &= \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} + \begin{bmatrix} v_g(t) \\ v_a(t) \end{bmatrix}
\end{aligned} \tag{13.23}$$

This is a multivariable stochastic state-space description. We model the velocity and acceleration themselves as zero-mean white-noise processes $\xi_g(t)$ and $\xi_a(t)$ respectively. The first state is angular position, and the second angular velocity. Essentially, the two state equations of interest are just

$$\dot{\theta}_p(t) = -\omega_p(t) + \xi_g(t) \tag{13.24}$$

$$\dot{\omega}_p(t) = \xi_a(t) \tag{13.25}$$

Equation (13.24) combines the gyro input (white-noise) with an integrated accelerometer reading (velocity) and gives a net output of velocity. Notice that the gyro acts in the negative direction due to gravity. A gyro is not affected by gravity, but an accelerometer measures the effect of gravity and acceleration combined. Equation (13.25) is just a description of acceleration in terms of a white-noise process that models it. The state-matrices are $\boldsymbol{A} = \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix}$, $\boldsymbol{D} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\boldsymbol{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. (we could omit $\boldsymbol{D}$ altogether, but it is useful for calculating the transfer-function). The measurement noise is modelled for the gyro as $v_g(t)$ and $v_a(t)$ for the accelerometer. If we normalise the process noise variances to unity, we can make $\boldsymbol{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Now we find the transfer-function matrix relating the white-noise inputs to the output vector $\boldsymbol{y}$. We find

$$\begin{bmatrix} \theta_p(s) \\ \omega_p(s) \end{bmatrix} = \begin{bmatrix} \frac{1}{s} & -\frac{1}{s^2} \\ 0 & \frac{1}{s} \end{bmatrix} \begin{bmatrix} \xi_g(s) \\ \xi_a(s) \end{bmatrix} \tag{13.26}$$

In words:

Angular position is the integral of gyro velocity minus the double integral of acceleration.
Angular velocity is the integral of acceleration.

These are our two measurements from the sensor that have in turn measurement noise added. It is important to note that our measurements are not velocity and acceleration, but position (from the accelerometer) and velocity from the gyro. This is because we can calculate the angle of acceleration from the accelerometer directly. Of course, we could just use this and we are done! However, the accelerometer position calculation is noisy in itself and needs combining with the other gyro sensor. Now we can calculate the transfer-function of a Kalman filter if we put some numbers for the measurement noise. Arbitrarily, we put $\boldsymbol{R} = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.001 \end{bmatrix}$. MATLAB gives us for the continuous-time Kalman filter gain matrix

```
K =

  31.6346   -0.4999

  -0.4999   31.6188
```

And the steady-state error-covariance matrix

```
P =

   0.0316   -0.0005

  -0.0005    0.0316
```

Then we can find the transfer-function of the Kalman-filter

$$\begin{aligned} G_{KF}(s) &= C(sI - (A - KC))^{-1}K \\ &= \frac{1}{s^2 + 63.2s + 1000} \begin{bmatrix} 31.6s + 1000 & -(0.5s + 31.62) \\ -0.5s & 31.6s + 1000 \end{bmatrix} \end{aligned} \tag{13.27}$$

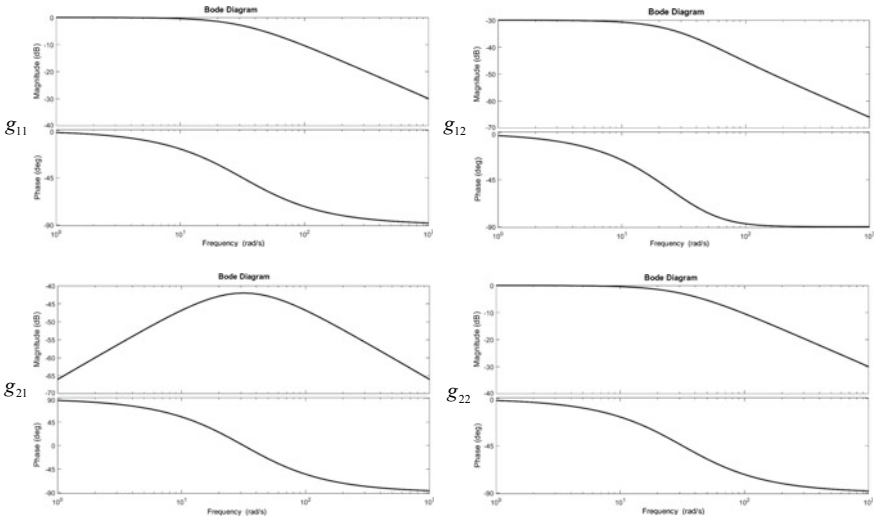


Fig. 13.5 Bode-plots of Kalman-filter

It can be seen from Fig. 13.5 that the individual elements of the transfer-function matrix has the form shown. The shape is of more interest than the scales. We can conclude the optimal filter does the following:

Angular position estimate = low-pass filtered accelerometer angle position measurement − low-pass filtered gyro velocity
Angular velocity estimate = −band-pass filtered accelerometer position estimate + low-pass filtered gyro velocity estimate

There are several other ways to model the combined effect including modelling gyro drift itself as a state-variable. Sometimes only one white-noise source is used to model both the gyro and accelerometer driving signals.

13.4 System Identification Using the Kalman-Filter and Recursive-Least-Squares

Unlike the Wiener-filter, the Kalman-filter works on time-varying systems as well as LTI systems. This is a major advantage for some applications. We can use a Kalman-filter to estimate the parameters of an unknown system. The only thing we need assume, is the order of the system. This is best illustrated with a simple example. Let us take a third-order system and write it in polynomial form.

$$y_k = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}} u_k + v_k \tag{13.28}$$

For this case the parameters are constant but assumed unknown. They could be time-varying as well, but we keep it simple at this stage. In (13.28) v_k is additive white-noise and u_k is assumed to be known. In fact we create u_k to drive the real system and usually it must be rich in harmonics. It can be a square-wave or even white-noise. We cannot use constant dc as an input. Write (13.28) in a different form from its difference-equation.

$$y_k = -a_1 y_{k-1} - a_2 y_{k-2} - a_3 y_{k-3} + b_0 u_k + b_1 u_{k-1} + v_k \tag{13.29}$$

Now write in vector form

$$y_k = \mathbf{X}_k^T \boldsymbol{\theta}_k + v_k \tag{13.30}$$

where $\boldsymbol{X}_k^T = [-y_{k-1} - y_{k-2}, -y_{k-3}, u_k, u_{k-1}]$ is termed the regressor-vector and $\boldsymbol{\theta}_k = [a_1, a_2, a_3, b_0, b_1]^T$ is the parameter vector that we must estimate. If the weights are constant, then we can assume a state-model of the form

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} \tag{13.31}$$

If they are time-varying we can assume

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \xi_k \tag{13.32}$$

This latter equation is known as a random-walk or integrated white-noise vector ξ_k. It is a discrete-time state-space model with $\boldsymbol{F} = \boldsymbol{I}$, $\boldsymbol{H} = \boldsymbol{X}_k^T$ and $\boldsymbol{G} = \boldsymbol{I}$. For the constant-parameter case we can write the best estimate in the least-squares sense recursively as

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \frac{\mathbf{P}_k \mathbf{X}_k}{1 + \mathbf{X}_k^T \mathbf{P}_k \mathbf{X}_k} \left[y_k - \mathbf{X}_k^T \boldsymbol{\theta}_k \right] \tag{13.33}$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \frac{\mathbf{P}_k \mathbf{X}_k \mathbf{X}_k^T \mathbf{P}_k}{1 + \mathbf{X}_k^T \mathbf{P}_k \boldsymbol{X}_k} \tag{13.34}$$

where $\mathbf{P}_k > 0$ is the usual error-covariance matrix. The above algorithm looks very similar to the Kalman-filter but is not exactly the same. In fact it pre-dates the Kalman-filter by over 100 years. It is known as deterministic recursive-least-squares (RLS) and became popular from the 1970s as a method of on-line identification of a system. Such an approach is known as an adaptive or explicit self-tuning control-system. If we can estimate the parameter-vector, then we have the system and can compute the controller. According to Plackett [38], RLS was discovered by Gauss in 1821. It has fast convergence and can be extended to

ARMAX models. The more general approach can be used for systems with time-varying stochastic parameters and uses a Kalman-filter [39].

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \frac{\mathbf{P}_k \mathbf{X}_k}{\sigma_v^2 + \mathbf{X}_k^T \mathbf{P}_k \mathbf{X}_k} \left[y_k - \mathbf{X}_k^T \boldsymbol{\theta}_k \right] \tag{13.35}$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \frac{\mathbf{P}_k \mathbf{X}_k \mathbf{X}_k^T \mathbf{P}_k}{\sigma_v^2 + \mathbf{X}_k^T \mathbf{P}_k \mathbf{X}_k} + \mathbf{Q}$$

$\boldsymbol{Q} = \sigma_\xi^2 \boldsymbol{I}$ and σ_v^2 can be assumed to be the variances of the process and measurement-noise respectively. The above algorithms are for the case of SISO systems only. For multivariable systems we can easily extent the theory or even use the scalar cases multiple times for each input and output.

13.4.1 RLS Estimation Example

Consider the system

$$y_k = \frac{0.8 - 3z^{-1}}{1 - 1.5z^{-1} + 0.6z^{-2} + 0.2z^{-3}} u_k \tag{13.36}$$

A program was written in LabView to implement the RLS algorithm. The input was chosen as white-noise unit-variance. The results for the evolution of the parameters versus time were as shown in Fig. 13.6.

The convergence is very fast, but the algorithm is quite complex in terms of multiplications. Because of this it has been used more in slow process-control type applications rather than faster electro-mechanical systems. With the advent of powerful and small computers, we may well see more of this algorithm. Unlike many other cases, we have a time-varying vector in the algorithm, so we cannot implement the steady-state solution. The gain vector must be allowed to vary with time towards convergence. Other less complex algorithms are available which are less expensive computationally yet are not as fast. For example, the least-mean squares algorithm (LMS) which is much favoured in adaptive signal-processing [40]. LMS uses the method of steepest-descent which is quite a slow convergent algorithm, but has great stability properties if used correctly.

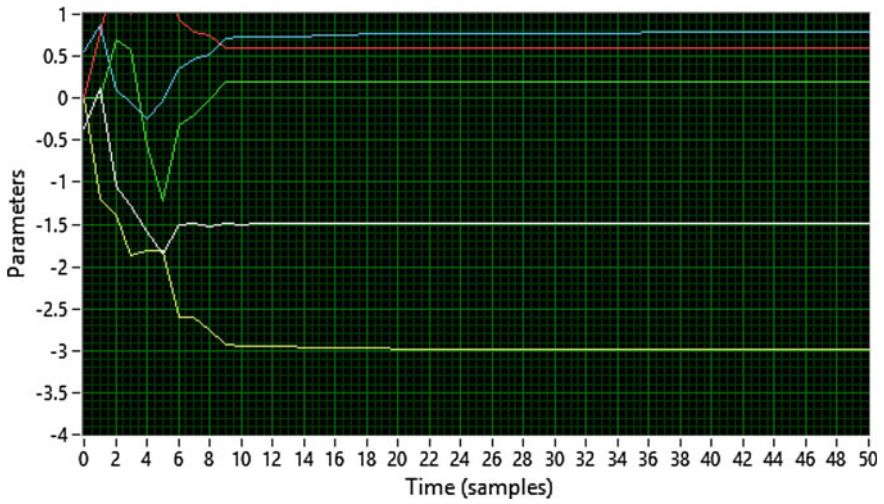


Fig. 13.6 Evolution to convergence of unknown weights using RLS algorithm

13.4.2 RLS Estimation of System Transfer-Function

Consider a continuous-time system given by a second-order transfer-function, slightly resonant.

$$\mathrm{G(s)} = \frac{10000}{\mathrm{s}^2 + 40\mathrm{s} + 10000} \tag{13.37}$$

The system has an undamped natural frequency $\omega = 100\,\mathrm{rad/s}$ and a damping factor $\zeta = 0.2$. Such a system has a frequency-response with a roll-off of -40 dB/decade, a phase-shift that starts at zero and goes to $-180°$, and a peak overshoot in the frequency-response of 8.14 dB. This happens at around the resonance frequency which is close to the undamped natural frequency. Converting to discrete-time, we use the Bilinear transform.

MATLAB Code 13.2

```
% Define continuous-time 2nd order system
wn=100;
zeta=0.2;
num=[wn^2];
den=[1 2*zeta*wn  wn^2];
g=tf(num,den)
%Convert to discrete-time
%Sampling time 1kHz
ts=1/1000;
sysd=c2d(g,ts,  'Tustin'
```

For a sampling frequency of 1 kHz, MATLAB returns the discrete-time system

$$y_k = \frac{2.443 + 4.89z^{-1} + 2.44z^{-2}}{1 - 1.951z^{-1} + 0.9609z^{-2}} u_k \tag{13.38}$$

We have added an extra gain of 1000 (60 dB) to give larger numerator parameters (otherwise the convergence is hard to see on a graph). First, we find the parameters using RLS. The parameter initial convergence is shown using LabView in Fig. 13.7.

Having found the discrete-time transfer-function, we can calculate its frequency-response from its z-transfer-function. The magnitude and phase is shown in Figs. 13.8 and 13.9 respectively.

We see that a second-order system with the desired frequency-response is easily identifiable from these graphs. The magnitude estimate gets worse as the plot nears half-sampling frequency of 500 Hz, due to zeros causing a notch-effect at half-sampling. The peak in the magnitude is around 8 dB as predicted, which occurs around 15 Hz (100 rad/s is 15.9 Hz). Furthermore, the gain is 60 dB at

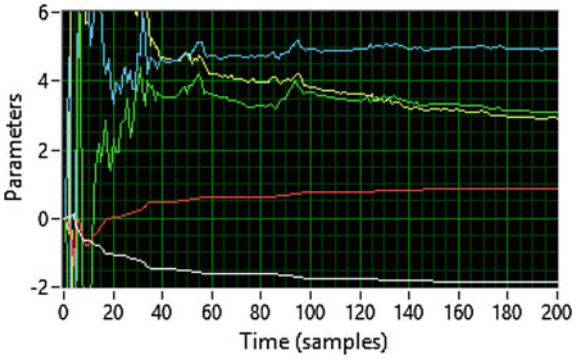


Fig. 13.7 Evolution of the parameters towards convergence

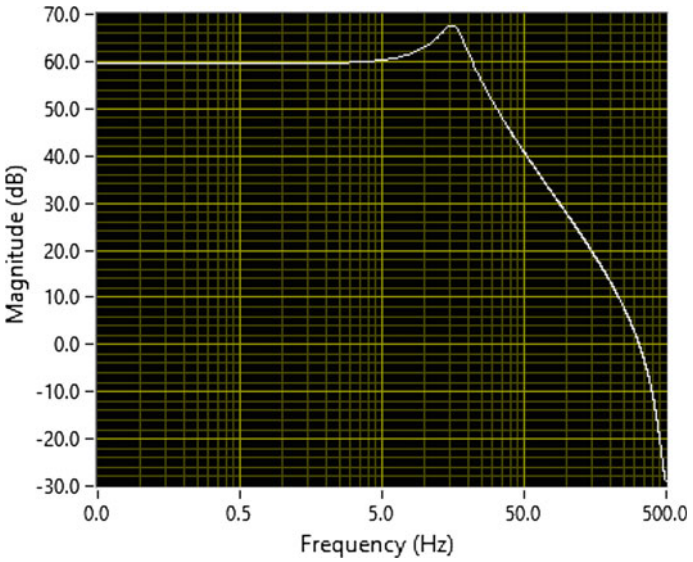


Fig. 13.8 Bode-plot calculated (magnitude in dB) from estimated model

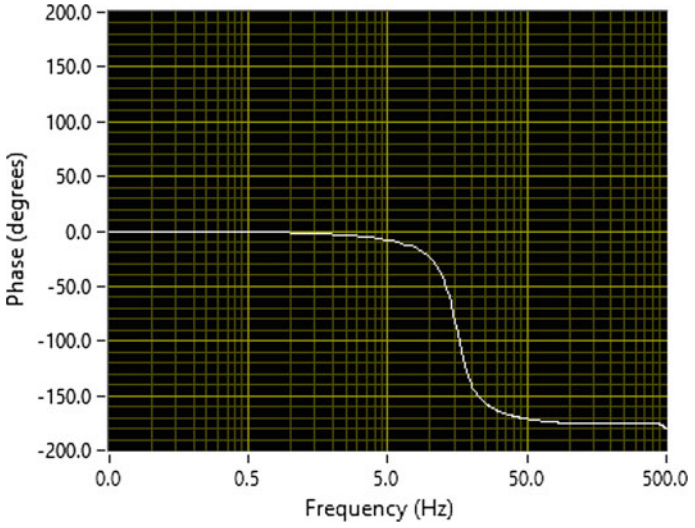


Fig. 13.9 Phase-plot calculated from estimated model

20 Hz and 20 dB at 200 Hz, a decade above. This indicates a slope of −40 dB/decade. One of the problems with parameter estimation is that the order of the system is unknown. We can overestimate the order and the algorithm usually gives us a stable pole and a zero which are very close to one-another and cancel. We must

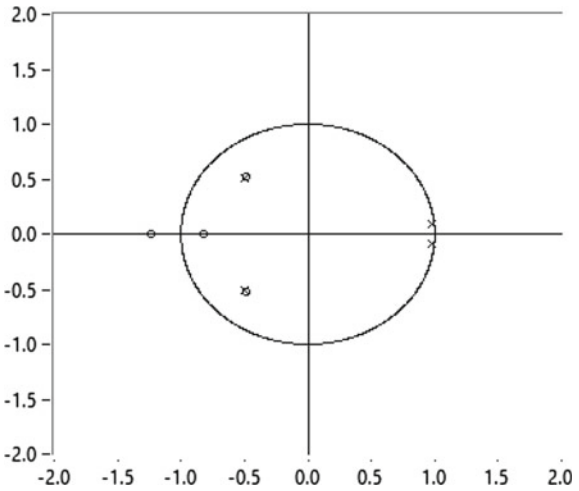


Fig. 13.10 Gives 4th order estimation of second-order system and pole-zero cancelations. (note, the zeros at half-sampling are not too much cause for concern as they only give a notch type effect at half-sampling and effect the phase there)

leave enough freedom in the order of the numerator and denominator for this to happen. We can calculate the poles and zeros and plot them on the z-plane. Observe in Fig. 13.10 what happens when we assume the order is 4 on both numerator and denominator. There is a complex pole-zero cancellation. The poles are stable. Note, if we do just increase the number of poles and not the zeros, then the RLS algorithm will do its best to fit the best all-pole model to the system. We can model systems as all-pole, pole-zero, or even all-zero (FIR). Having stable pole-zero cancellations is not of great importance unless these estimated polynomials are being used in an equation such as a polynomial *Diophantine* equation. Then the polynomials must be relative-prime (free from any stable or unstable common factors).

Chapter 14
Implementing Digital Real-Time Servos

There are a great many books on control-systems which extensively cover the theory, often in much more detail than this textbook. However, few if any discuss the practical aspects of implementation. This chapter will fill the gap from an experimenter's point of view. There is no better way to learn than hands-on experience and constant practice with the real hardware. There is a great deal of choice when it comes to the processors available in today's markets. However, there can be a tendency to become enthralled in the software itself rather than the problems of feedback. It is for this reason that the National Instruments myRio is chosen for the purpose of implementing the software in this book. It is programmed in LabView graphical language known as data-flow and is far easier to understand than the traditional text-based languages. It still requires a little experience, but the end result is arguably easier to follow. The myRio is a student embedded device which has built in FPGA. We don't use the FPGA part in this book, but only employ the real-time processor. It has an ARM Cortex dual-core processor with adequate computing power for our purposes. It is connected to the PC via a USB link, but can run stand-alone on batteries. The great advantage of using LabView is the built-in graphical user interface (GUI) and extensive libraries. The problem of interfacing has been largely taken care of. It has a great many applications, but since it has a real-time processor it is ideal for creating a timed-loop in which we can design anything ranging from PID, lag-lead or Kalman-filters. Our first experimenters rig is shown in Fig. 14.1.

It consists of a 24v 350W dc-motor, two incremental rotary-encoders, a myRio and a 100A H-Bridge. Other than the myRio, the rest of the hardware is not critical. The motor was chosen since it has a great deal of torque and is inexpensive (it is a motor intended for electric bicycles). We run the servo at 12v simply for convenience of the bench power-supply. We have made an aluminium inertial load, but this is not a necessary requirement. It was used as a large pulley for winding a metal wire and weight on various tests. The H-bridge is much larger than necessary, since the motor only draws about 30 A when at full load. Experience however has taught that when experimenting with feedback, that the oscillations can quite easily

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_14

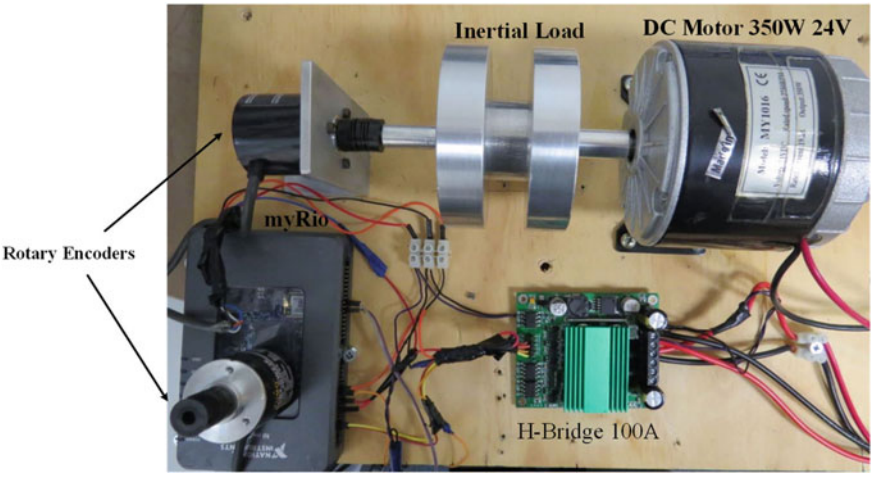


Fig. 14.1 Experimenters servo kit using a myRio

overload an H-bridge and burn it out. Therefore, a much larger one can easily cope with any overloading problems. Two quad-detectors are used. One is coupled to the shaft of the motor giving a reading of position as the shaft rotates. A second identical one can be used as a setpoint. Unlike an analogue potentiometer, the quad-detectors can rotate a full 360° and keep going almost indefinitely. The only thing that stops an infinite number of rotations is the word length of the integer used to count the number of pulses. The rotary-encoders used have 1024 pulses per revolution (see Chap. 4 for more details on rotary-encoders). One of the advantages of having full 360° control, is that a speed-controller and position-controller are almost indistinguishable. If we design a position controller (i.e. a position-servo) and apply a ramp waveform as the input, it becomes a speed-controller by varying the slope of the ramp. The applied waveform determines whether we have a position or speed controller. This is something that has never been possible in servos that use potentiometers. However, old-style ac-servos would have been capable of this, since they used synchro-resolvers. A synchro-resolver is a rotating transformer with no wiper movement unlike a potentiometer. Synchros were used in many heavy industry and military applications and will probably be largely replaced by *inductive encoders*. Our rotary-encoder is inexpensive, readily available and great in the lab environment, but for some rough terrains, aerospace and medical applications we may require a more robust or higher accuracy sensor.

There are no gears used. Usually a gearbox is used when a motor is insufficient in torque and extra torque is needed via the gearbox. Small servos can use cheap motors that are small and still deliver torque via trading off speed. Gears have a habit of wearing out with frequent use however. Essentially the rig shown in Fig. 14.1 is akin to many old-style analogue servos. In place of the dc amplifier lies the H-bridge. In place of the potentiometers are the rotary-encoders and in place of

the control circuitry is software in the form of the myRio. The rotary-encoders require a 5v supply which, (due to the low current they require) can be supplied from the myRio. An alternative approach which is commonly in use by many home constructors is to use the Arduino platform. It has extensive open-source libraries and online discussion forums and is relatively easy to program. The hardware we use here is suitable for non-electronic engineers, since feedback is also a topic of much interest in mechanical engineering and mechatronics.

14.1 Implementing Digital PID Control

The equations we need to implement are straight forward. First we create an error signal based on: *error = setpoint-measured position.* This we call e_k. Then we implement the PID elements with the error as an input. Finding the correct sign for the error is important, we require negative feedback and not positive. This is easily tested by ensuring that the sign of the setpoint and the sensor are both positive. In steady-state in an ideal situation, they should subtract and give zero error. The proportional gain term is then found by using a gain term K_p.

$$P_k = K_p e_k \tag{14.1}$$

The integral term is just a forward Euler integrator as follows:

$$I_k = I_{k-1} + K_I T_s e_k \tag{14.2}$$

where T_s is the sampling interval in seconds and K_I is the integral gain which we adjust.

Finally, the derivative term is found from a first-difference

$$D_k = \left(\frac{e_k - e_{k-1}}{T_s}\right) K_D \tag{14.3}$$

Then we sum them all up and send the output to the PWM generator.

$$PID_k = P_k + I_k + D_k \tag{14.4}$$

A few simplifications can be made. For example, in (14.2) we have $K_I T_s$ which can be merged together into another variable to avoid an extra multiplication. In the same way for (14.3) we can avoid a division by merging $\left(\frac{K_D}{T_s}\right)$ as another variable.

We require an endless loop in LabView to implement all these equations. We also can see that we require stored values. For example, in the integrator we have used the past value of the integrator. In the differentiator we need the past value of the error. We store these past values in *shift registers*. Figure 14.2 shows a while-loop with a shift register. The while-loop runs forever until the stop button is

Fig. 14.2 While-loop with a shift register. Rightmost is current value we send data to, left is past values we retrieve data from

pressed. The stop button appears on the front-panel of the LabView VI. The terms VI stands for *virtual-instrument* and is an old name for a LabView program. LabView began its days as a software tool for creating virtual test instrumentation. Since that time however, it has grown outside all expectation and we can think of it as a general dataflow programming language. We have all the usual syntax definitions that any other language has except they take the form of shape and colour. For example, thin blue lines (wires) are integers, thick blue wires are integer arrays. Thin orange lines are floating-point variables or constants and the thick versions are arrays. Similarly, green is a Boolean variable. A shift register is the small box with the black arrow pointing upwards at the right and left of the while-loop in Fig. 14.2.

Writing a LabView program is analogous to wiring a circuit. A number comes in, it gets multiplied by another number or a variable from a dial on the front panel, and it results in the output. We wire all the blocks together and we have a program. We must work left to right however. For example, Fig. 14.3 shows the computation of 3×2 divided by 6. It has no loop around it since none is required for a one-off calculation.

The answer appears on another page, the front panel. The front panel is the control panel of the virtual instrument. It can have dials, controls or graphs to enter or display data. To enter data we create a *control* and to display it we create an *indicator*. We can instead of having a control just enter a *constant* as shown in Fig. 14.3, where the numbers 3, 2 and 6 are constants. We can have constant integers, floating-points, arrays, matrices and so-on.

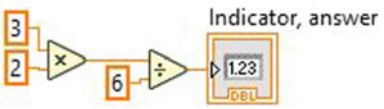


Fig. 14.3 Simple arithmetic in LabView

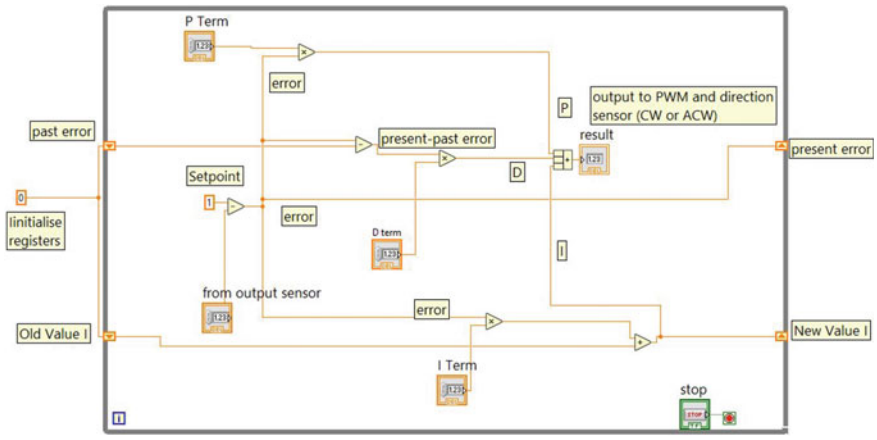


Fig. 14.4 Basic concept behind PID LabView program

We use multiplier blocks or addition or subtraction to perform basic arithmetic. The basic concept behind our PID controller is shown in Fig. 14.4.

If we were to implement such a program, it would just execute at the maximum speed possible. There is nothing to tell the loop how fast to run. Therefore, in real-time LabView (an extension of LabView for real-time systems), we have a *timed-loop* which we can define the sampling-time. Figure 14.5 Shows our real-time program. An addition is that we have added a lowpass filter after the derivative term. Without it the system can be prone to all sorts of noise and high-frequency rattle. It changes the slope from +20 to −20 dB/decade starting at 500 Hz.

There is not much difference except the loop is set up to sample at 10 kHz. The code must execute within that time-period. In Fig. 14.5 we see other blocks such as the PWM library function which gives out the PWM waveform for an input

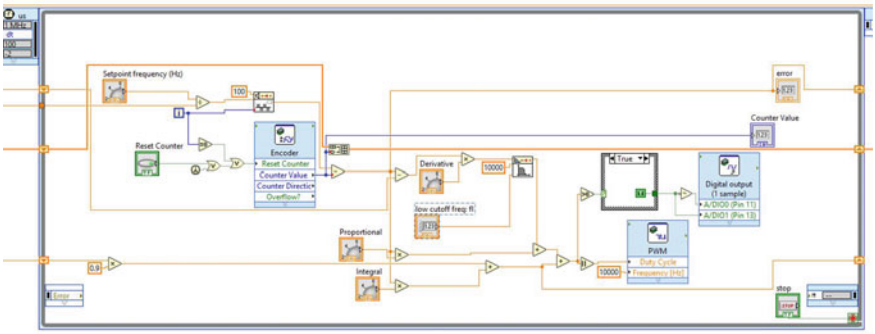


Fig. 14.5 Real-time PID in real-time LabView using a timed-loop

proportional in the range 0–1. The Encoder library block decodes the quadrature signals from the rotary encoder. Here we have included a square-wave for testing purposes because we do need some form of input that varies, and a step is the best test. We also use the digital ports for two logic levels. This is to switch the H-bridge from forward to reverse when a negative going signal appears. This is a facet that of course is not necessary in an analogue amplifier since the voltage can be made to swing from positive to negative using a bipolar power-supply. In the digital world we need only a single positive supply and we reverse the polarity to the motor when necessary. The PWM has a clock frequency of 10 kHz. If a lower frequency is used an annoying whine is heard from the motor since it is in the audible range. There are three shift registers used. The top two are for the derivative and the integral terms. The last one with the thick line going towards it (an array) is used for testing purposes and stores values of the output for later display. We run the program for a few seconds and then stop it. The stored values are then plotted outside of the loop and appear on the front panel. We could plot in real-time, but graphics is an overhead and if we can do without it we leave it to the non-real-time part of the software. The front-panel of the PID program is shown in Fig. 14.6.

We have added knobs to control all three terms and the setpoint frequency of the square-wave. The output amplitude is the number of pulses and is related to the angle turned by the shaft of the motor. Although our sensor is 1024 pulses per revolution, the myRio hardware quadruples this by counting each edge in both of the A and B outputs of the encoder. (a standard technique) We can easily scale this to convert to degrees as required. This gives us 0.0879° per pulse. We could have

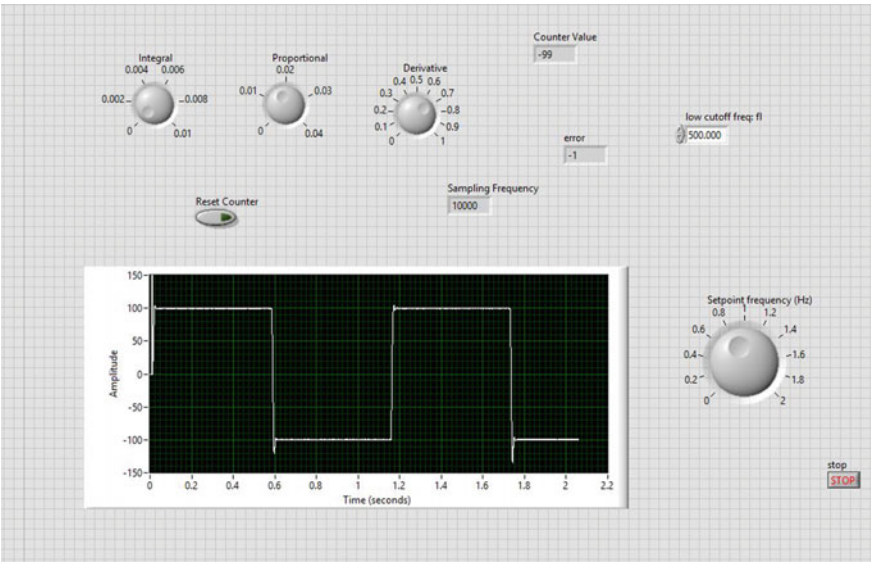


Fig. 14.6 Front panel of PID program

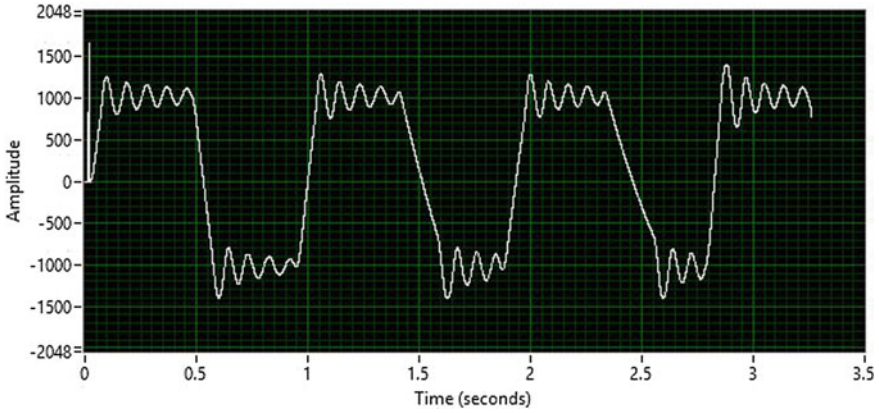


Fig. 14.7 Proportional gain only. Shaft rotation response

used any processor and language to implement a PID, but the tuning of the values without a GUI will be far more difficult. The extensive LabView library significantly simplifies the task in hand.

It is best to start with just proportional control. Turn down the two other terms of I and D and just wind up the proportional gain slowly. The setpoint is plus or minus 1024 pulses which is plus or minus 90° rotation of the shaft (since 4096 pulses is one revolution). We see a response from the shaft of the motor as shown in Fig. 14.7.

The motor begins to oscillate since it has an integrator (the mechanical integration from going to position from velocity by the sensor) and a motor time-constant. It will oscillate if the gain is held too high as shown. This is a good sign that we have negative feedback and the oscillation is just the classic lack of phase-margin (if we think frequency-domain). It is at this stage that an H-bridge can overload, but our one is rated to be well within safety limits.

The next stage is to introduce the derivative term. In doing so, we introduce damping to the system. The result is shown in Fig. 14.8.

We see a lovely well-damped response and at this stage we could stop and just admire the result! However, there is an old servo-designers saying that two integrators are always better than one! We add the integral term with care as too much will cause instability. Two integrators give a lot of gain at low-frequencies and make the shaft of the motor hard to budge when stationary. It feels locked tight in position (with this size of motor and current). There are no gears holding it in place. The other advantage of two integrators is that the servo can follow a ramp input with zero steady-state error. It becomes a type II system. This is also important if we are going to put a ramp into the system to make the servo a speed-controller. We want it to be able to track speed-changes with no error. Figure 14.9 shows the full PID controller result.

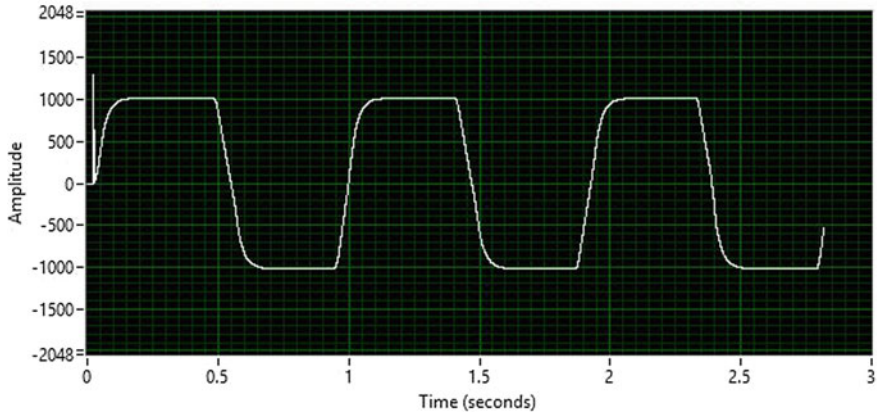


Fig. 14.8 P+D digital control

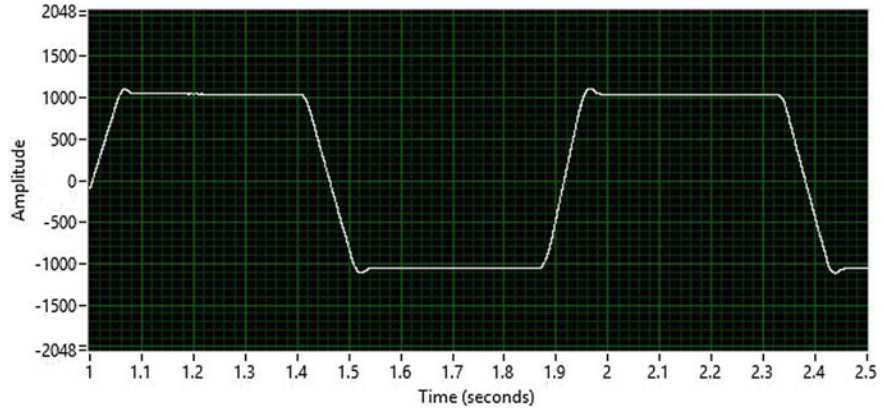


Fig. 14.9 PID digital control

The overshoot is quite minimal, and the system is well-damped. Many hours of tuning can alter this to whatever your choice of output is within the limits of the hardware. There are of course formal methods of tuning, but usually for motors this is not necessary as the result is immediately visible. For slow process-control systems tuning can be much more difficult a task. While tuning large dc-motors such as this, it is a good idea to anchor the motor to a table, so it doesn't move. It is well known in servo folklore of motors jumping off the bench! If we make the integral action too big, we end up with ringing as shown in Fig. 14.10.

Increasing even more results in continuous oscillations as shown in Fig. 14.11.

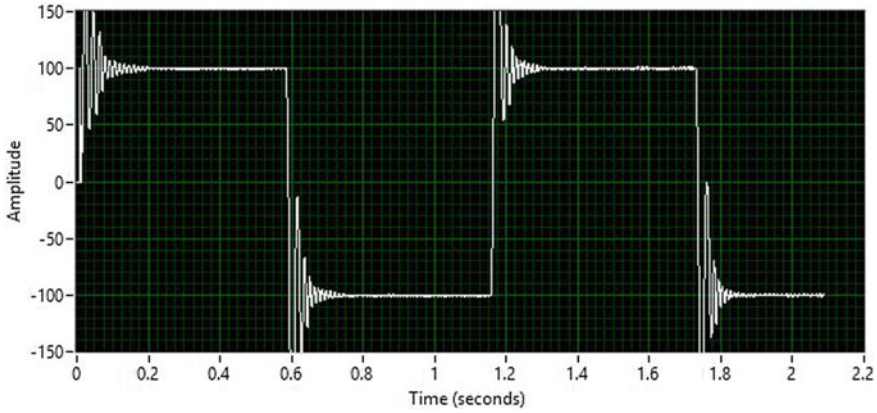


Fig. 14.10 Too much integral action

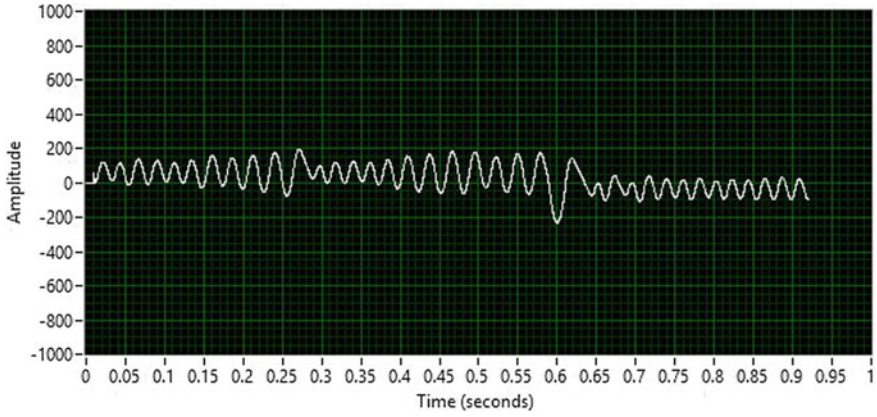


Fig. 14.11 Oscillations due to too much integral action. Approx. 40 Hz oscillation

14.1.1 Integrator Windup

When implementing integrators of the form

$$I_k = I_{k-1} + K_I T_s e_k$$

we often run into a problem where the integrator can keep integrating to very large numbers and when the setpoint changes in the other direction the integrator somehow must get from its present state to a new one some distance away. This can cause huge swings in the output and unsatisfactory results. The general term for this is *integrator windup*. A simple way around this is to use a *soft* integrator. The other name for this is a *leaky* integrator. We can switch from the above equation to

$$I_k = \alpha I_{k-1} + K_I T_s e_k$$

where α is unity for a pure integrator or 0.99 for a soft integrator. In theory the steady-state error should suffer by doing this, but in reality the steady-state error is only marginally affected. Besides this problem, starting some systems can be difficult as pure integrators can give problems with wide variations in output, so a soft-start option is preferable and switching to a pure integrator soon afterwards. There are a variety of tricks in the literature that can be used including limiting the range (saturating) that the integrator can reach with a hard limit in positive and negative directions. This also makes a good safety measure and is easy to implement in software. In analogue circuits, Zener diodes are sometimes used in the feedback path to clamp a signal in the same way. If some kind of limit is not put on an integrator, then a dangerous situation can occur when the motor is attached to some other piece of critical equipment. Safety features do need to be built in after the servo stability design is decided upon.

14.2 Implementing Digital Lag-Lead Control

Recall from Chap. 10, that we usually use two main types of compensator. The first is the PI, similar to the PI part of a PID controller, but the P and I terms are combined and not implemented in parallel. We use

$$G_{c1}(s) = \frac{1 + sT_I}{s} \tag{14.5}$$

as a digital filter (compensator). We name the corner-frequency in Hz as $f_c = \frac{1}{2\pi T_I}$ for future reference.

Using the Bilinear transform, we get

$$G_{c1}(z) = \frac{c + dz^{-1}}{1 - z^{-1}} \tag{14.6}$$

with constants $c = T_s/2 + T_I, d = T_s/2 - T_I$.

We also need a phase-lead compensator (the lead part of lag-lead)

$$G(s) = \frac{1 + sT_1}{1 + sT_2}, T_1 > T_2 \tag{14.7}$$

Recall also that the span-ratio is given by $p = \frac{\omega_h}{\omega_\ell} = \frac{T_1}{T_2}$. Usually a span of 10 is sufficient as a good trade-off. It can result in a 55° phase-margin. The frequency in Hz that gives maximum phase-shift is found from $f_\phi = \frac{1}{2\pi}\sqrt{\omega_h \omega_\ell}$. At this

frequency, the phase-lead has a gain equal to the square-root of the span-ratio for Eq. (14.7). At high-frequencies the gain of the phase-lead is the span-ratio itself.

Using the Bilinear transform on (14.7) we get

$$G_{c2}(z) = K\left[\frac{1 + bz^{-1}}{1 + az^{-1}}\right] \tag{14.8}$$

with constants $K = \left[\frac{T_s + 2T_1}{T_s + 2T_2}\right]$, $b = \left(\frac{T_s - 2T_1}{T_s + 2T_1}\right)$, $a = \left(\frac{T_s - 2T_2}{T_s + 2T_2}\right)$

We select a sampling interval T_s = 100 μs corresponding to 10kHz. This high sample-rate was chosen since we need at least ten times the highest frequency of interest for the Bilinear transform to deliver accurate results, and we also would like the closed-loop bandwidth to be as high as possible. We should be able to work as high as 1kHz bandwidth in our experiments, although that is very unlikely to be achieved. The advent of modern processors with fast sample-rates gives us this freedom. There is a danger that the compensator poles could get close to the edge of the unit-circle with too high a sample-rate, but this is only if there are rounding problems and we are here using full floating-point arithmetic.

To implement the arithmetic for the constants in the z transfer-functions we can use ordinary arithmetic blocks from the LabView library. This can lead to a large block-diagram however and we prefer a different method. Instead, we return to text-based solutions and use a formula node. Figure 14.12 shows how this is implemented for both compensators. The formula node takes input values on the left and output values on the right. In between sits the formulae themselves.

For the phase-lead compensator, the upper and lower corner-frequencies are also calculated for a given span-ratio and frequency for maximum phase. The difference-equations for the compensators are then implemented in a while-loop as

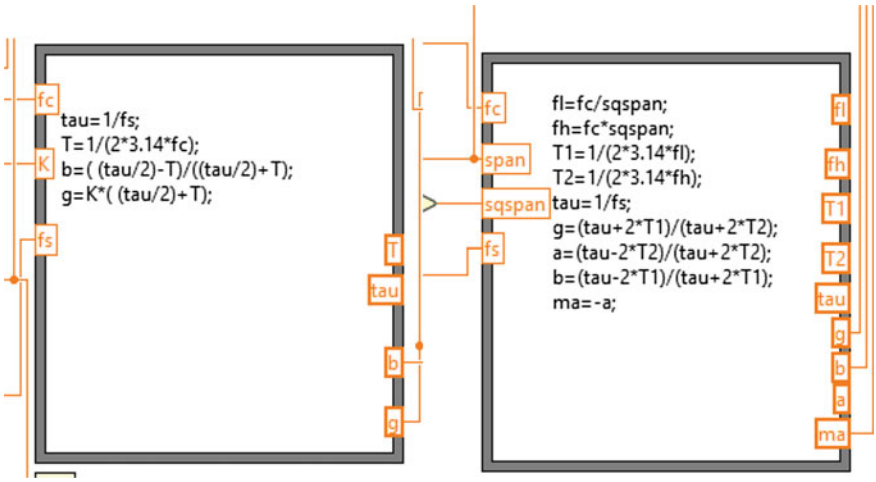


Fig. 14.12 Formula nodes for calculating PI and phase-lead compensator constants

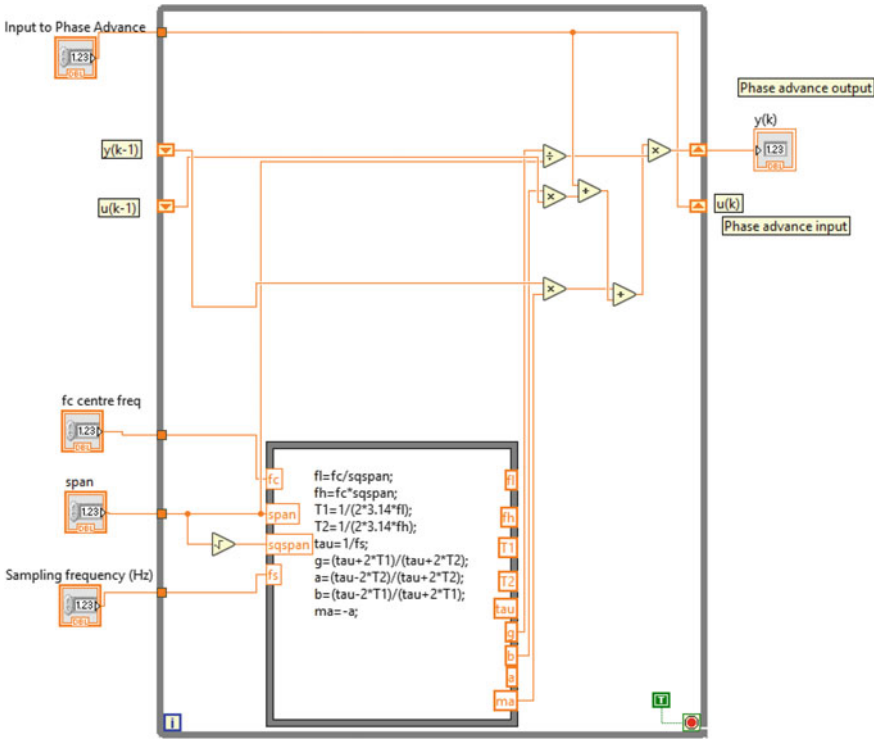


Fig. 14.13 Phase-advance VI sub

before, using registers to store past values of input or output. It is good practice to put each compensator separate in a VI *sub* (like a subroutine, a separate block) so that the code is readable. Many novice LabView programmers programme a mass of connecting wires on one large VI and find it hard to later remember where all the mass of wires is going. They end up having to scroll large distances on the PC to find the destination, or connect a new wire. The use of *subs* is recommended where necessary to avoid this problem. The sub for the phase-advance is shown in Fig. 14.13. It executes only once every time it is called and stores the required input and output values in the shift registers for the next iteration of the outer timed while-loop.

Finally, the whole VI is shown in Fig. 14.14.

The lag-lead controller is compressed into one sub VI which in turn consists of the cascaded PI and lag-lead controllers. It is as if we have created our own library VI. This makes the graphical program much easier to follow. The front-panel which corresponds to this program is shown in Fig. 14.15. An extra lowpass filter was added after the controller with roll-off frequency 1kHz. This just ensures a steep roll-off to attenuate high-frequency noise (or resonances) and it is far away from the designed unity-gain crossover frequency.

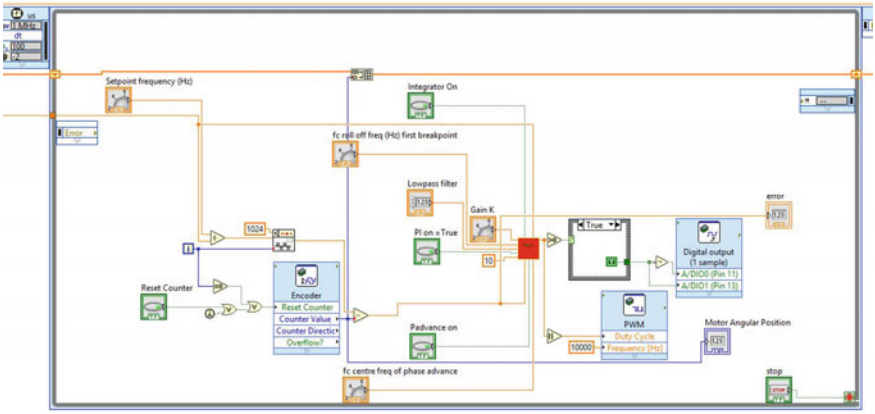


Fig. 14.14 VI for lag-lead control

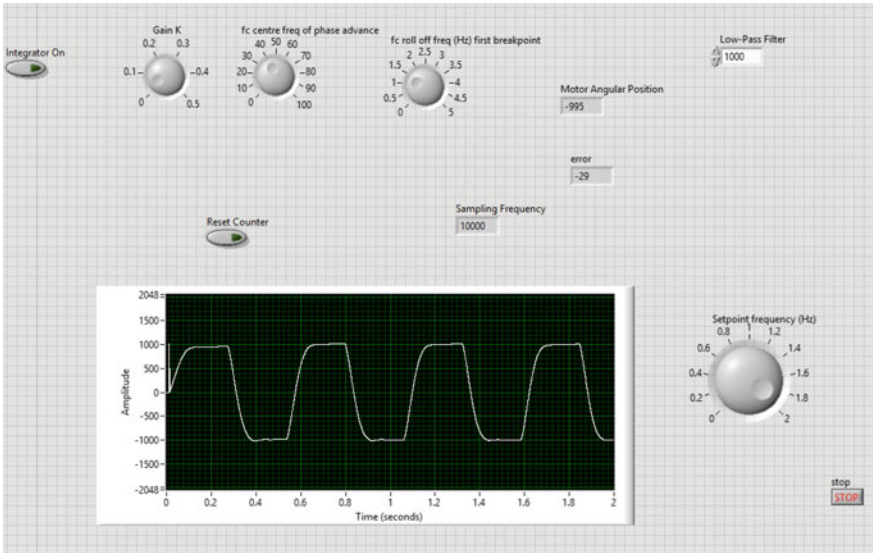


Fig. 14.15 Front panel for lag-lead control-system

The three parameters that can be set from the front panel is the overall gain K, the PI corner-frequency f_c (a zero that flattens the slope of the integrator to make it flat) and the centre frequency of the phase-advance f_ϕ. By phase-advance centre frequency, we mean the frequency that gives maximum phase. The span-ratio is fixed at 10. These settings can be varied in real-time. This is in complete contrast to old-fashioned analogue control where every design change needs new resistor or capacitor values. It is tuned like a PID controller, only the tuning is done in the

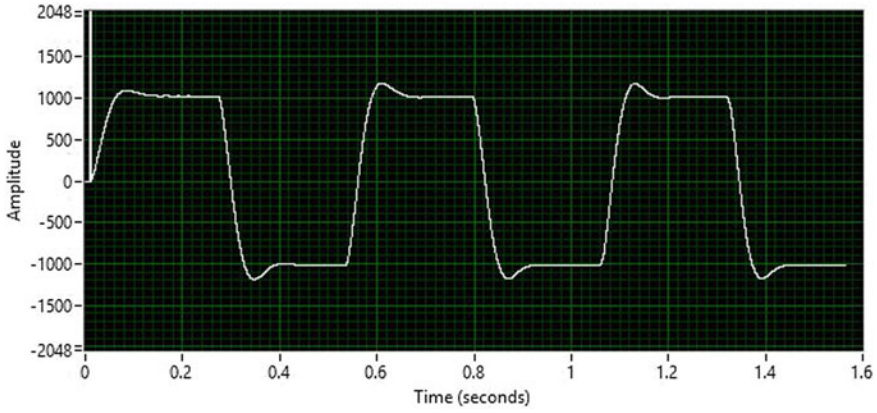


Fig. 14.16 Lag-lead type control

frequency-domain. We already know from previous experiments that oscillations occur when the gain is too high, and the frequency of these oscillations is around 40Hz. This is a great clue and tells us that phase-advance is needed at this frequency. The initial setting for the centre frequency of the phase-advance should therefore be 40Hz. This will be our unity-gain crossover frequency. We first show the response of the system in Fig. 14.16 when the span-ration is 10, $f_c = 1\,\text{Hz}$ and $f_\phi = 40\,\text{Hz}$.

Now we show the effect when f_c is set too high (Fig. 14.17).

The effect of reducing the span-ratio to 4 can be seen in Fig. 14.18. All of these captured waveforms are for a 40 Hz unity-gain bandwidth.

We can use LabView like an oscilloscope. Once we have captured a waveform we can easily zoom in and measure the step-properties as in Fig. 14.19.

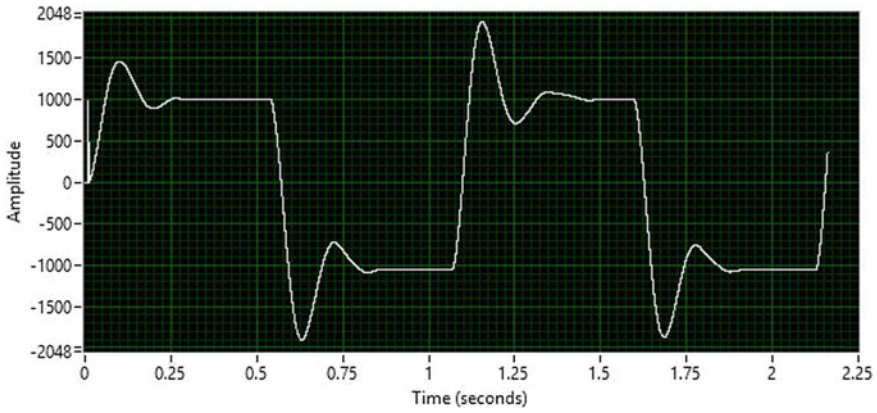


Fig. 14.17 f_c is set too high

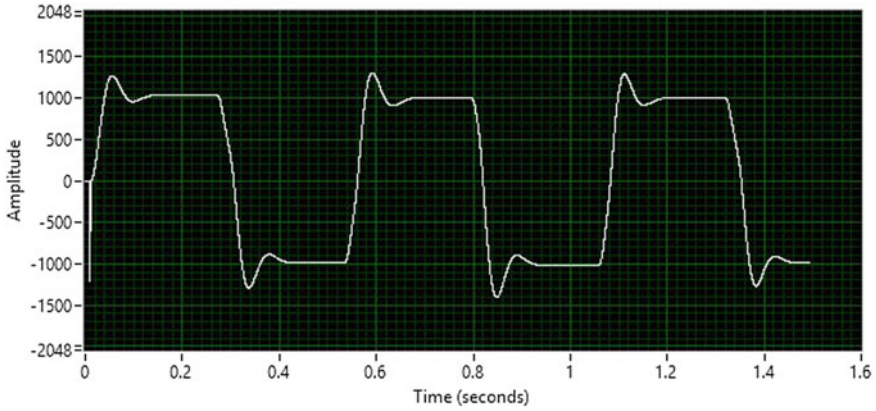


Fig. 14.18 Span-ratio of 4

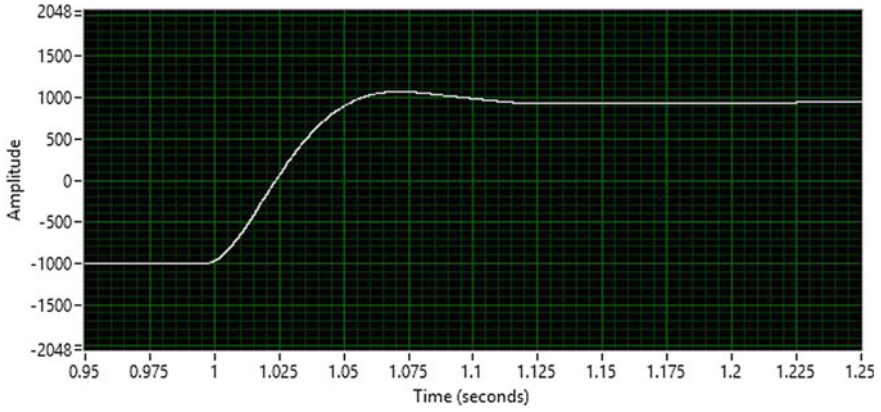


Fig. 14.19 Close-up of step-response using LabView. Rise-time approx. <50 ms, settling-time approx. 100 ms

We can achieve higher bandwidths if necessary, it is an iterative process. We can see that the design is done entirely in continuous-time, even though the controller is digital. Whilst not the only approach, it is by far the easiest and most intuitive way.

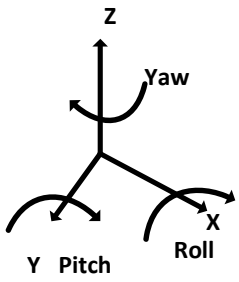
14.3 Sensor Fusion and Embedded Control of a Camera Stabilising System

In the world of control-systems and mechatronics there are a great many problems encountered which rely on balancing systems. Some are relatively low bandwidth such as personal mobility systems (for example Segway [1] and so-called hover-boards), and others require faster response-times as encountered with some UAVs and stabilised gimbaled platforms. [2–4]. Usually the control-system consists of PID type algorithms or variants thereof [5, 6], or LQR (this is an optimal-control topic) state-feedback approaches [7, 8]. The main advantage of a PID approach is that an accurate mathematical model is not needed and instead manual tuning of the stability of the loop is necessary.

The application here is a teaching rig consisting of a servo gimbal with three degrees of freedom, pitch, roll and yaw. At present we do not use the yaw as we are interested primarily on the stabilisation of a platform and not its rotation in a particular direction, since this can be done manually if a camera is mounted. Of course, other applications do require a camera to be fixed at any angle in space and we can also achieve this too by changing our setpoints from a level platform with zero-degrees pitch and roll to any chosen angle. The basic gimbal rig used is shown in Fig. 14.20. It uses off-the-shelf components as a prototype experimental design.

Although it can pitch, roll and yaw, we restrict ourselves to only Pitch and Roll of the small FPV camera which is mounted on the top (to yaw requires a different method, a digital compass). Just below the camera the MPU-6050 MEMS device is mounted which is a 6-axis Gyro and Accelerometer used in many modern devices such as smart phone technology and is here used as a motion tracking device for pitch and roll. The MEMS device is placed just below the small camera so as to measure the cameras pitch and roll. The whole unit is anchored to a plate for testing, but in real-life would be mounted to some form of hand-held pole for photography. The motors used are model servo types which run in velocity mode only. They *are not* position servos. The advantage here in using them is that we do not need an H bridge to drive them and it makes the unit more portable. Although servomotors can be used, and the design of the control-system is easier, the speed of response was found to be rather sluggish. This is because the bandwidth of the servos is fixed, whereas with motors we have faster speed of response by our own design for the same size of motors and current. We can convert any off-the-shelf servo to just a dc motor plus gearbox (and H-bridge driver), by simply breaking the feedback loop from the servo-pot and getting rid of a mechanical end stop which is put into all such servos in the factory. This is all well documented in online forums. Many hobbyists use a servo (on open-loop) as a motor for driving the wheels of small robots. In this way they avoid having to have a separate H-bridge.

The aim of the design, is to produce a self-stabilising camera mount that stays level as the photographer's hands move. The embedded processor used is a National Instruments myRio. The entire control-system is written in LabView graphical code. The MEMS device has an I2C output so no analogue-to-digital



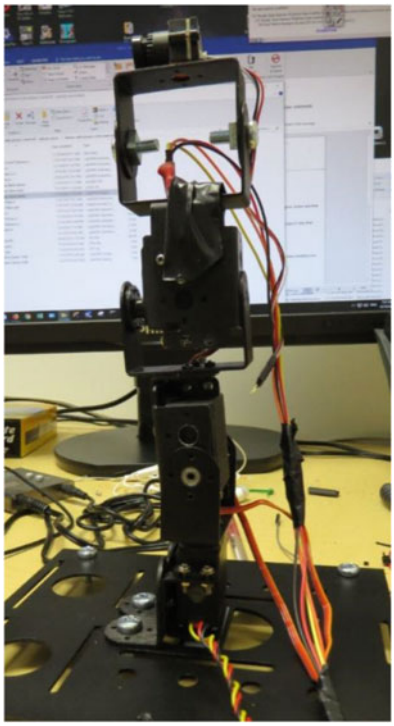


Fig. 14.20 Prototype stabilised platform rig

sampling or anti-aliasing is necessary. Similarly, the output to the motors is standard model-servo PWM type so no digital to analogue conversion is necessary. The power amplifiers which in such a system would be an H bridge, are not needed because the servomotors drive directly from PWM and have their own power electronics built within. We first consider the problem of sensor fusion between the accelerometer and gyro.

14.3.1 Sensor Fusion Using the Kalman Filter

The MPU-650 sensor is used in a variety of real-world applications including robotics, gaming (virtual-reality) and UAVs. We only use the three-axis accelerometer and 3-axis gyro. We can easily derive the state-variable description of the equations of motion relating angle, angular velocity (gyro) and angular acceleration (accelerometer). Much of the analysis has already been covered in Chap. 13. Consider the pitch angle only (and Roll follows identically with minor changes):

$$\begin{bmatrix} \dot{\theta}_p \\ \dot{\omega}_p \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_p \\ \omega_p \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \xi_g \\ \xi_a \end{bmatrix}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_p \\ \omega_p \end{bmatrix} + \begin{bmatrix} v_g \\ v_a \end{bmatrix} \tag{14.9}$$

ξ_g and ξ_a represent the random *inputs* to the gyro and accelerometer respectively i.e. velocity and acceleration process noise. v_g and v_a represent the random uncorrelated *measurement* noise to the gyro and accelerometer measurements. Note that the observations vector is related to velocity (gyro) and acceleration *angle*, not acceleration itself (The acceleration angle here means the component of acceleration which acts at a particular angle in each axis). The minus sign is because of negative acceleration due to gravity. The above equations represent the differential equations of a double integration to get from acceleration to angle. The double integrator is driven by white-noise making the system model a kind of random-walk.

Discretising with sample interval T_s (we can use the power-series method here as only a few terms are needed and the higher-order terms become zero) gives us:

$$\begin{bmatrix} \theta^p_{k+1} \\ \omega^p_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & -T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta^p_k \\ \omega^p_k \end{bmatrix} + \begin{bmatrix} T_s & -T_s^2/2 \\ 0 & T_s \end{bmatrix} \begin{bmatrix} \xi_g \\ \xi_a \end{bmatrix}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta^p_k \\ \omega^p_k \end{bmatrix} + \begin{bmatrix} v_g \\ v_a \end{bmatrix} \tag{14.10}$$

and we define the discrete-time matrices as follows

$$\boldsymbol{F} = \begin{bmatrix} 1 & -T_s \\ 0 & 1 \end{bmatrix}, \boldsymbol{G} = \begin{bmatrix} T_s & -T_s^2/2 \\ 0 & T_s \end{bmatrix}, \boldsymbol{H} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

For the Kalman-filter, here we use the one-step ahead prediction estimate, which is is given as

$$\hat{\mathbf{x}}_{k+1/k} = \mathbf{F}\hat{\mathbf{x}}_{k/k-1} + \mathbf{K}e_k$$
$$e_k = \mathbf{y}_k - \mathbf{H}\hat{\mathbf{x}}_{k/k-1} \tag{14.11}$$

The Kalman-gain matrix $\boldsymbol{K}$ found from the usual discrete-time Riccati equation shown below, with measurement vector $\boldsymbol{y}_k = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ and state-vector $\hat{\boldsymbol{x}}_{k+1/k} = \begin{bmatrix} \hat{\theta}^p_k \\ \hat{\omega}^p_k \end{bmatrix}$.

$$K_k = FP_kH^T(R + HP_kH^T)^{-1}$$
$$P_{k+1} = FP_kF^T + GQG^T - K_kHP_kF^T \tag{14.12}$$

In the above the matrix $\boldsymbol{P}$ is the positive-definite symmetric error-covariance matrix, the trace of which converges to some small value with time. For simplicity we can normalise the process noise covariance matrix $\boldsymbol{Q}$ to be the unit covariance matrix. We then select a suitable measurement noise covariance matrix $\boldsymbol{R}=\begin{bmatrix}\sigma_a^2 & 0\\ 0 & \sigma_g^2\end{bmatrix}$ consisting of the variance of the accelerometer and gyro noise. It is not a simple task finding the additive noise covariance matrix. We choose $\boldsymbol{R}=\begin{bmatrix}0.001 & 0\\ 0 & 0.001\end{bmatrix}$, which appears to give good results. Others have done detailed measurements on the MEMS device to find these values for various conditions and the results are available on various web-pages and forums. We iterate (14.12), but there is no need to do this in real-time because the matrices do not change with time and hence the Kalman gain matrix will converge to a steady-state value. We then only implement (14.11) for converged gains, which is a Wiener-filter for state-estimation. There are some applications using Kalman filters where the noise variances change over time, e.g. in GPS coordinates this can be acquired information and fed to the filter. However, for this application we only need the steady-state Kalman state estimator. Of importance is that the acceleration measurement is not acceleration but *acceleration angle*. Simple geometry can tell us that for a three-axis accelerometer with acceleration at each axis given by R_x, R_y, R_z, that the vector sum gives an angle as derived from the three-axis accelerometer as:

$$\theta_{\mathrm{p}} = \tan^{-1}\left(\frac{\mathrm{R_x}}{\sqrt{\mathrm{R_z^2} + \mathrm{R_y^2}}}\right) \tag{14.13}$$

likewise, for Roll we have

$$\theta_{\mathrm{r}} = \tan^{-1}\left(\frac{\mathrm{R_y}}{\sqrt{\mathrm{R_z^2} + \mathrm{R_x^2}}}\right) \tag{14.14}$$

We could of course use (14.13) and (14.14) directly as a measure of angle, but it will have more noise than a Kalman estimate when combined with the Gyro estimate.

14.3.2 Method of Control

The Kalman-filter will give an estimate of the position and velocity in pitch and roll. These could be used to apply state-feedback. However, state-feedback (see earlier chapter) does not have inherent integral action, so a steady-state error would

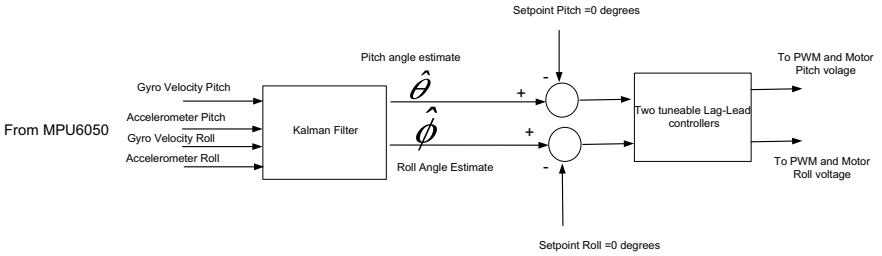


Fig. 14.21 Pitch and roll control

result. We could add an extra integrator state, but the approach here is to use the position estimate only. We use the position estimate as if it were the real estimate (the separation principle). We need only then design a stabilising lag-lead controller. By lag-lead here we mean a PI and a phase-lead in cascade as in the previous section. Figure 14.21 shows the block-diagram of the design philosophy.

We see that both setpoints are zero, representing zero degrees for a level platform. With feedback around it the platform should stay in the same position even though the whole mechanism is tilted in either direction. The Kalman-filter gain vector could be calculated in a separate program altogether, but a decision was made to calculate the vector in the same program outside of the main loop. The gain values are then fed to the real-time while-loop which executes the Kalman-filter first and then the lag-lead controller. The steady-state Kalman-filter is just a vector difference equation of the first-order. LabView has facility to define matrix constants and we implement the Kalman-filter in a similar way as we implement any difference-equation. For this we use shift registers to store past and present values of the states. The same method is used as storing scalars, since some LabView VIs are *polymorphic*. This means they change their size shape or colour to accommodate the type of input. For example, if we move from constant to array to two-dimensional array, a register can accommodate this and change accordingly. Figure 14.22 shows the Kalman-filter as implemented in LabView. The extensive matrix library in LabView real-time enables us to easily implement such equations.

The shift register in the loop recognises that an array is being used, and its floating-point type. An array and a vector are interchangeable for this application. If an integer array is wired to a register it will turn blue in recognition that it is storing integers. A vector with two values can also be thought of as an array with two values. The shift register stores both values at the current iteration on the RHS of the loop. We can do the same for a matrix if ever needed. A matrix of size 2×2 becomes a two-dimensional array when it is connected via a wire. Any values stored by the register can then be retrieved at the next iteration on the LHS of the

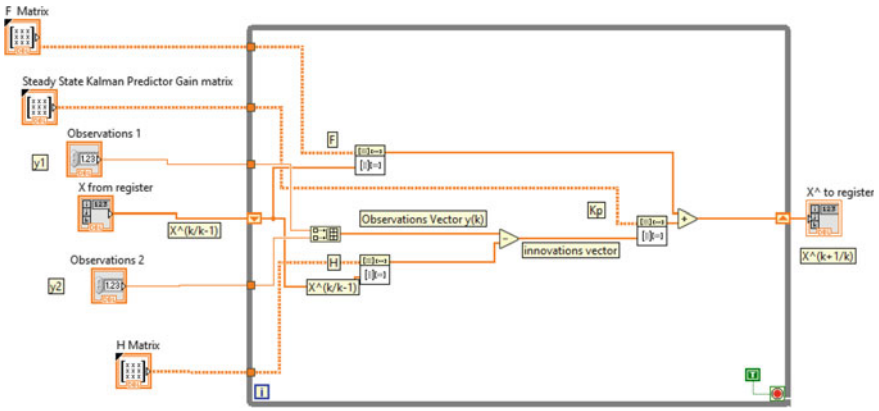


Fig. 14.22 Steady-state Kalman-filter

loop. Although the while-loop is not timed, it is placed in a sub within a timed-loop. The Kalman-filter was implemented first and tested on its own by moving the sensor to various angles in pitch and roll. A test was carried out that the estimated angles were the same within reasonable tolerances and that these estimates did not drift over time. Once the Kalman-filter design is assured, we can design the controller.

The controller is the same as used in the previous section and implemented in exactly the same way. The only difference is that we have two controllers, one for pitch and one for roll. We begin with pitch and close the loop around it first. We begin my applying only negative feedback with a gain which we set to a low value. This way we can safely see that we have indeed got negative feedback (and not positive!) as it will respond rather sluggishly and try to reduce the error. We then increase the gain in steps until it begins to oscillate. At this point we can add and adust the phase-lead which should stop it oscillating. Finally we an add an extra integrator. The methodology sounds simple, but can take some time to get right. The mechanical structure should be as firm as possible to avoid any resonant frequencies and the whole device vibrating.

Figures 14.23 and 14.24 show the captures responses to a rapid change in angle for pitch and roll respectively. The movement is done by suddenly moving the platform by 45° and recording the response For pitch, the overall time to correct itself is around 40 ms, whilst for roll it was around 50 ms. The roll response has a small residual oscillation (a parasitic) which is ever present. It can be reduced by reducing the bandwidth of the loop, but this slows down the response. This is an

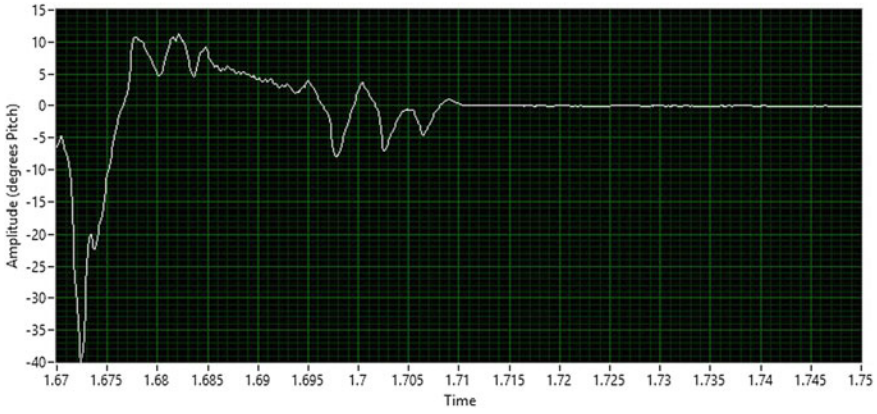


Fig. 14.23 Response to a change in pitch

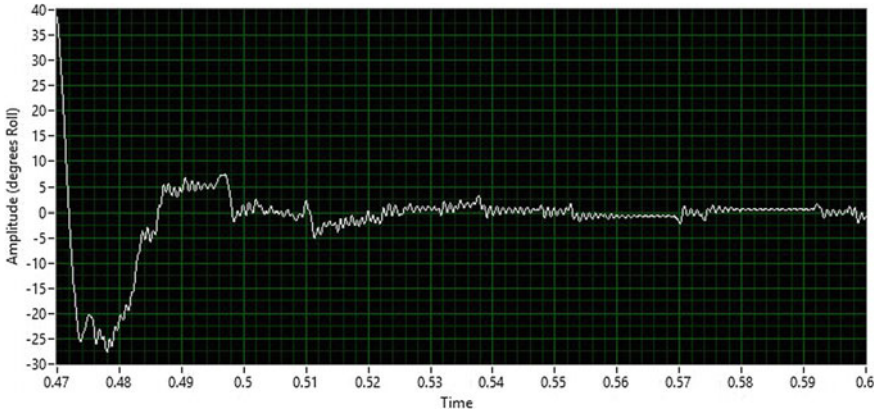


Fig. 14.24 Response to a change in roll

excellent rig to study and improve ones skill in digital-control. A later add-on to the program was the ebility to plot the position of the platform in 3D. We cannot use 3D graphics on a myRio as it is not supported in such a small real-time device. However, we can send the data to the PC and using the host computer plot it in non real-time. We do this by using *network shared-variables*. This is like having a memory location which is shared by the host computer and the target myRio. The

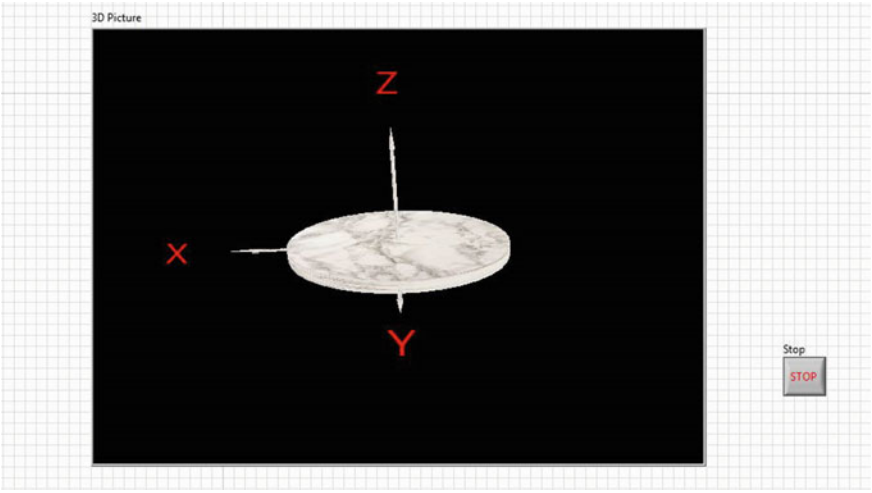


Fig. 14.25 3D plot of stabilized platform position

position of pitch and roll are then shared with the PC. We then use the extensive LabView 3D graphics to plot a platform that moves with the real platform. Figure 14.25 shows the 3D picture obtained. This is a cylindar drawn in 3D graphics with a marble texture put on top.

Chapter 15
Nonlinear Systems

Having spent 14 chapters on linear control-systems, it is hard to do justice in one short chapter of the nonlinear problem. We can only hope at most to give a brief introduction to the salient points of the theory. There are a great many textbooks on this subject, many of which dwell on the mathematical side rather than the more practical approach to the problem. Here we try and steer a middle ground since a firm mathematical grounding is necessary to follow the theory.

15.1 Linear and Nonlinear

Recall from an early chapter that a linear system is a system that satisfies *superposition*. The superposition theory is used in basic electrical theory as well as signal processing and is well known to engineers. Most practicing engineers would inject a sinewave into a system and have a look at the oscilloscope of the measured output as an initial test. For example, with an amplifier, if it is operating in the linear region we expect that for an amplifier with passband gain K, that if we inject $x(t) = \cos(\omega t)$, we get out $y(t) = K\cos(\omega t - \phi)$. This is a change in amplitude (usually a greater value in the case of an amplifier), and a shift in phase. A cursory look on the oscilloscope will tell the engineer if the waveform is pure. A quick glance for any clipping of the waveform or distortion may well indicate that the amplifier is operating at the limits of its dynamic range for instance and the input signal needs to be reduced in amplitude. A spectrum analyser can be used for more accurate results.

A more mathematical approach is to apply two sinewaves one at a time and then together. This takes more time, since a circuit needs to be found to add two sinewaves together before it is injected. However, this is not a major difficulty and can be done if necessary. So we inject $x_1(t) = a\cos(\omega_1 t)$ and measure the output as $y_1(t) = Ka\cos(\omega_1 t - \phi_1)$. We then take another frequency and inject $x_2(t) = b\cos(\omega_2 t)$ and get out $y_2(t) = Kb\cos(\omega_2 t - \phi_2)$. We then predict that if both are

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_15

injected together as a sum that we should get the sum of the individual outputs. We should get an output $Kb\cos(\omega_2 t - \phi_2) + Ka\cos(\omega_1 t - \phi_1)$ on the oscilloscope. For a low-frequency and high-frequency sinewave the sum of the two is easy to spot, but a spectrum analyser can be used for more accurate results. In particular we must not have cross-modulation terms in the output. That is, frequencies which appear that are the sum or difference of the two added frequencies. For example, if the system under test had a square-law instead of a linear gain, then we would get out a waveform similar to $(a\cos(\omega_1 t) + b\cos(\omega_2 t))^2$. Expanding this waveform we get

$$(\mathrm{a}\ \cos(\omega_1 \mathrm{t}) + \mathrm{b}\ \cos(\omega_2 \mathrm{t}))^2 = \mathrm{a}^2\cos^2(\omega_1 \mathrm{t}) + \mathrm{b}^2\cos^2(\omega_2 \mathrm{t}) + 2\mathrm{ab}\ \cos(\omega_1 \mathrm{t})\cos(\omega_2 \mathrm{t}) \tag{15.1}$$

Now from basic trigonometric formulae

$$\cos^2(\omega_1 \mathrm{t}) = \frac{1}{2}(1 + \cos(2\omega_1 \mathrm{t})) \tag{15.2}$$

$$\cos(\omega_1 \mathrm{t})\cos(\omega_2 \mathrm{t}) = \frac{1}{2}(\cos((\omega_1 + \omega_2)\mathrm{t}) + \cos((\omega_1 - \omega_2)\mathrm{t})) \tag{15.3}$$

Equation (15.2) is a dc term plus a frequency which his double the frequency of the input.

Equation (15.3) is the sum and difference terms (usually found in amplitude-modulated waveforms). If we use these expressions and simplify, we find that we get

$$\begin{aligned}
&(\mathrm{a}\ \cos(\omega_1 \mathrm{t}) + \mathrm{b}\ \cos(\omega_2 \mathrm{t}))^2 \\
&= \frac{\mathrm{a}^2}{2} + \frac{\mathrm{b}^2}{2} + 2\mathrm{ab}\ \cos(\omega_1 \mathrm{t})\cos(\omega_2 \mathrm{t}) + \frac{\mathrm{a}^2}{2}\cos(2\omega_1 \mathrm{t}) + \frac{\mathrm{b}^2}{2}\cos(2\omega_2 \mathrm{t}) \\
&= \frac{\mathrm{a}^2}{2} + \frac{\mathrm{b}^2}{2} + \mathrm{ab}(\cos((\omega_1 + \omega_2)\mathrm{t}) + \cos((\omega_1 - \omega_2)\mathrm{t})) + \frac{\mathrm{a}^2}{2}\cos(2\omega_1 \mathrm{t}) + \frac{\mathrm{b}^2}{2}\cos(2\omega_2 \mathrm{t})
\end{aligned} \tag{15.4}$$

We have a dc term plus two frequencies at twice the corresponding input frequencies and two more frequencies which are the sum and difference of the two input frequencies. This is quite a mix of frequencies for such a simple square-law and illustrates how complex are the waveform harmonics at the output of a nonlinear system. If the amplifier was linear but the waveforms clipped at the power-supply (perhaps +5v and 0v or values close to that), then we would have something close to a square-waveform at the output and we would expect odd harmonics to be present. Fortunately there are commonly met nonlinear shaping functions. This is illustrated in Fig. 15.1.

To simplify the mathematics as much as possible, traditionally we can model a nonlinear system as a linear system followed by a static nonlinearity as in Fig. 15.1.

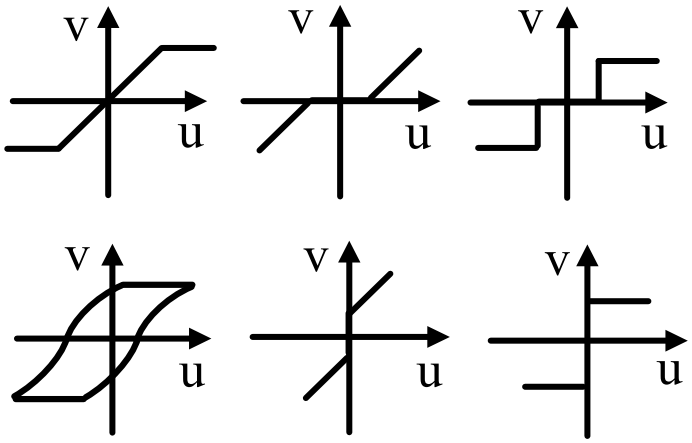
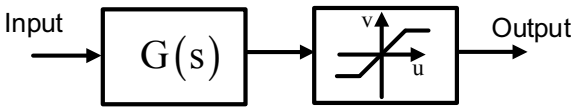


Fig. 15.1 Common static nonlinearity functions. Top (left to right): linear plus saturation, dead-zone, dead-zone plus saturation. Bottom (left to right): hysteresis, active-zone (preload), switching relay (also known as infinite gain limiter or sign function)

Fig. 15.2 LN model of a nonlinear system



This is known as the LN model and shown in Fig. 15.2. There is another model whereby the nonlinearity comes before the transfer-function. This is known as the *Hammerstein model* or NL model. In Fig. 15.2 we show a continuous-time system, but the approach is more than often applied in the discrete-domain too. Of the static nonlinearities in Fig. 15.1, perhaps the best known and most frequently encountered is the linear system plus saturation, as this is encountered in all amplifier or electrical/mechanical systems that have a limit to the travel of the output of say a thruster or mechanical linkage. Dead-zone is encountered in many electro-mechanical systems and signifies a system that has no output until the input reaches a certain level. This can happen with play in worn gears. The more exact form of dead-zone is when saturation is added with it. Hysteresis is often introduced deliberately as a nonlinearity for some crude control-systems such as temperature control of heating in a room. It is used for similar reasons that some digital logic inverters have hysteresis. For any kind of hard limiter that reacts to zero-crossings, when noise is taken account the output can chatter back and forth in an undesirable manner (rapid switching). Hysteresis introduced a memory that stops this phenomenon. In the case of heating control, the threshold for switching on and off can be different. It is for this reason that switching without hysteresis is seldom used on its own in some control-systems of this kind.

15.1.1 Equilibrium Points

In an LTI system, the states of the system have unique steady-state values. In an unforced system the states usually die out towards zero as the final resting place and if the system is forced, then the states end up in positions which are a function of the setpoint. The final values of the states (or the system output) are known as the equilibrium points. Unlike LTI systems, a nonlinear system can have equilibrium points which are both stable and unstable. Therefore, the initial starting points play a role as to where the final states end up (in the case of an unforced system). To get an idea of what is meant by equilibrium points, consider the often used ball example rolling freely as shown in Fig. 15.3.

In the case of stable equilibrium the ball must roll down to the lowest point of the curved container. In the unstable case the ball cannot balance on its own and must descend down either side. For a flat surface the equilibrium is said to be neutral. Figure 15.4 shows a collection of equilibrium points as a ball progresses along the surface. In the first unstable level on the left, the stable point draws the ball downwards. Being in certain regions therefore has predictable outcomes.

To find the equilibrium states or output of a system, we need to consider a general nonlinear SISO state-space model.

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x}(t), u(t)) \\ y(t) &= g(x(t), u(t)) \end{aligned} \tag{15.5}$$

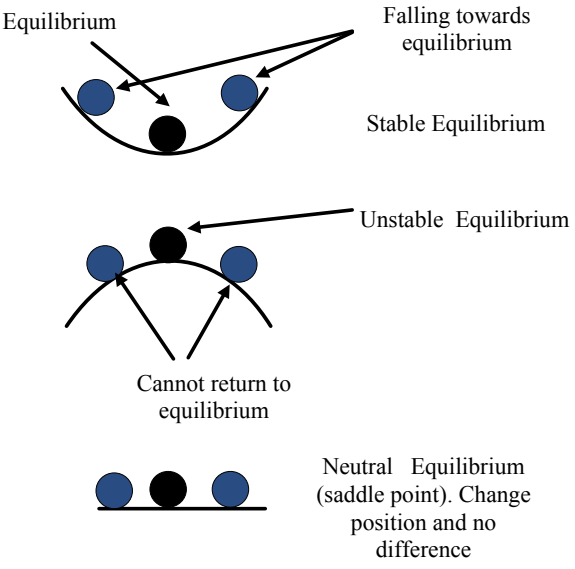


Fig. 15.3 Stable and unstable equilibrium

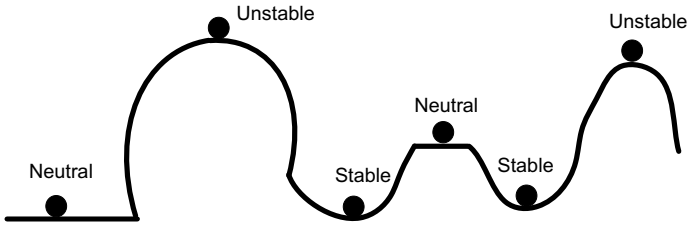


Fig. 15.4 Stable and unstable equilibrium at different points

The equilibrium points are algebraic solutions (x^*, u^*) of

$$\dot{\mathbf{x}}(\mathrm{t}) = \mathrm{f}(\mathbf{x}(\mathrm{t}), \mathrm{u}(\mathrm{t})) = 0 \tag{15.6}$$

Likewise for a discrete-time system [49] we have

$$\begin{aligned} \mathbf{x}_{\mathrm{k}+1} &= \mathrm{f}(\mathbf{x}_{\mathrm{k}}, \mathrm{u}_{\mathrm{k}}) \\ \mathrm{y}_{\mathrm{k}} &= \mathrm{g}(\mathbf{x}_{\mathrm{k}}, \mathrm{u}_{\mathrm{k}}) \end{aligned} \tag{15.7}$$

The equilibrium points occur when $\mathbf{x}_{\mathrm{k}+1} = \mathbf{x}_{\mathrm{k}} = \mathbf{x}^*$, $\mathrm{u}_{\mathrm{k}+1} = \mathrm{u}_{\mathrm{k}} = \mathrm{u}^*$.

For example, consider the autonomous discrete-time state-space system with two states. We are only interested in the state-equations as there is no forcing input or defined output other than the states themselves.

$$\begin{aligned} \mathrm{x}^1_{\mathrm{k}+1} &= 0.5\mathrm{x}^1_{\mathrm{k}} + \left(\mathrm{x}^2_{\mathrm{k}}\right)^2 \\ \mathrm{x}^2_{\mathrm{k}+1} &= \mathrm{x}^1_{\mathrm{k}} + 0.5\mathrm{x}^2_{\mathrm{k}} \end{aligned} \tag{15.8}$$

where the superscript denotes the number of the state and the second state in brackets $\left(\mathrm{x}^2_{\mathrm{k}}\right)^2$ indicates the state is squared. Now let

$$\begin{aligned} \mathrm{x}^1_{\mathrm{k}+1} &= \mathrm{x}^1_{\mathrm{k}} = \mathrm{x}^{1*} \\ \mathrm{x}^2_{\mathrm{k}+1} &= \mathrm{x}^2_{\mathrm{k}} = \mathrm{x}^{2*} \end{aligned} \tag{15.9}$$

The superscript * denotes an equilibrium value. From (15.8) we get two equations when using (15.9) and solving

$$\begin{aligned} \mathrm{x}^{1*} &= 2\left(\mathrm{x}^{2*}\right)^2 \\ \mathrm{x}^{2*} &= 2\mathrm{x}^{1*} \end{aligned} \tag{15.10}$$

Solving the above results in two possible solutions. We have the vector of equilibrium points at $\mathbf{x}^* = \begin{bmatrix} \mathrm{x}^{1*} & \mathrm{x}^{2*} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 \end{bmatrix}^{\mathrm{T}}$ or $\begin{bmatrix} 0.125 & 0.25 \end{bmatrix}^{\mathrm{T}}$.

It is interesting to compare the same methodology as used now on a *linear* discrete-time system

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}u_k \tag{15.11}$$

When $\mathbf{x}_{k+1} = \mathbf{x}_k = \mathbf{x}^*$, $u_k = u^*$ we have

$$\mathbf{x}^* = (\mathbf{I} - \mathbf{F})^{-1}\mathbf{G}u^* \tag{15.12}$$

This is the steady-state value of the states for a constant input $u_k = u^*$. This is easily verified using z-transforms and the discrete-time final-value theorem. Hence for a linear system the steady-state value of the states can only be one solution vector, but for our nonlinear system there is more than one solution or equilibrium state. Unlike a linear system which is either stable or unstable, the nonlinear system could have a stable equilibrium state and an unstable one too (or several of each in a higher-order system). We require a method to determine which equilibrium points are stable or otherwise. In (15.12), we can only make this assumption if the system is stable, otherwise the steady-state value of the state-vector will be infinity.

In the continuous-time case consider the autonomous example

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) = \begin{bmatrix} -x_1^2 - x_2^2 + 1 \\ -x_2 \end{bmatrix} \tag{15.13}$$

The equilibrium points are found when the rate of change of the state vector is zero. We have $\dot{\mathbf{x}}(t) = 0$ giving

$$\begin{bmatrix} -x_1^2 - x_2^2 + 1 \\ -x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{15.14}$$

From the above equations, $x_2 = 0, x_1 = \pm 1$. We therefore have the two equilibrium vectors

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

15.2 Linearizing Systems

To understand what is involved in the linearizing process, it is best to first consider a curve and to fit a straight-line tangent to it at an arbitrary point on the curve. In our case the points will be the equilibrium points and not just any randomly chosen point. For the time-being however observe Fig. 15.5, which is the graph of $y = x^2$.

Take an x value at $x = 3$ and its corresponding y value is $y = 9$. This is an arbitrary point we have chosen. Let us linearize around this point and fit a straight

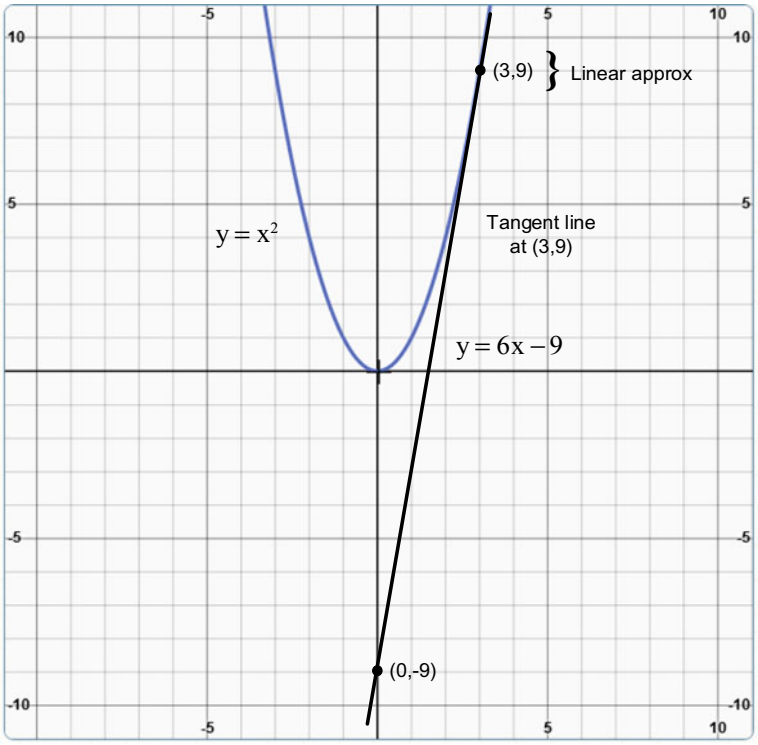


Fig. 15.5 Fitting a tangent to a curve

line of the form $y = mx + c$. The gradient at point (3,9) is found by differentiating $y = x^2$ to get $m = 2x = 6$. Substitute the values $x = 3$ and $y = 9$ and we get $9 = 6.3 + c$ giving $c = -9$. Our equation is now $y = 6x - 9$. However, this approximation is only good at the single point we chose, and even then, only for small perturbations from this point. This is the principle of linearizing a dynamic system. For a continuous-time system with equilibrium state-vector $\mathbf{x}^*$(there can of course be more than one equilibrium, so we apply the same technique to any such vector provided it is a stable equilibrium if at all possible). We use a Taylor-series [50] also known as *Lyapunov's first* or *indirect* method [51]. We must have that any equations are continuously differentiable thus

$$\begin{aligned}\dot{\mathbf{x}}(\mathrm{t}) &= \mathrm{f}(\mathbf{x}(\mathrm{t}), \mathrm{u}(\mathrm{t})) \\ &= \mathrm{f}(\mathbf{x}^*, \mathrm{u}(\mathrm{t})) + \left.\frac{\mathrm{df}(\mathbf{x}(\mathrm{t}), \mathrm{u}(\mathrm{t}))}{\mathrm{d}\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^*} (\mathbf{x}(\mathrm{t}) - \mathbf{x}^*) + \frac{1}{2}\left.\frac{\mathrm{d}^2\mathrm{f}(\mathbf{x}(\mathrm{t}), \mathrm{u}(\mathrm{t}))}{\mathrm{d}\mathbf{x}^2}\right|_{\mathbf{x}=\mathbf{x}^*} (\mathbf{x}(\mathrm{t}) - \mathbf{x}^*)^2 + \ldots\end{aligned} \tag{15.15}$$

We have assumed in the above that the input signal is held as a constant $\mathrm{u}(\mathrm{t}) = \mathrm{u}$ and is hence linear. (Differentiating a constant gives zero). If it had a square-term in

it or any form of nonlinearity, then it too would need to be differentiated. If we take a first-order approximation (ignoring the second and higher-order derivatives) then usually this is sufficient for small perturbations from the equilibrium vector. From (15.15) we now have a term $\frac{df(\mathbf{x}(t),u(t))}{d\mathbf{x}}$ which needs to be evaluated. This is differentiation of a vector wrt another vector. The result is not a vector but a matrix. This matrix is well known in vector calculus and is known as the *Jacobian* matrix. It is found as follows.

$$\frac{\mathrm{df}(\mathbf{x}(\mathrm{t}),\mathrm{u}(\mathrm{t}))}{\mathrm{d}\mathbf{x}} = \begin{bmatrix} \frac{\mathrm{df}_1}{\mathrm{dx}_1} & \cdot & \cdot & \cdot & \frac{\mathrm{df}_1}{\mathrm{dx}_n} \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ \frac{\mathrm{df}_n}{\mathrm{dx}_1} & \cdot & \cdot & \cdot & \frac{\mathrm{df}_n}{\mathrm{dx}_n} \end{bmatrix} \tag{15.16}$$

We usually label this matrix as $\boldsymbol{A}$ for continuous-time systems. We then can write our linearized system as an autonomous state-space system for small perturbations:

$$\Delta\,\dot{\mathbf{x}}(\mathrm{t}) = \mathbf{A}\Delta\mathbf{x}(\mathrm{t}) \tag{15.17}$$

The stability of the equilibrium vector follows in the usual state-space way by finding the eigenvalues of the $\boldsymbol{A}$ matrix and checking that they lie in the left half of the complex-plane.

In (15.17), this is only true when the control-signal itself is linear. In a more general case we require a second Jacobian and the state-space linear system becomes

$$\Delta\,\dot{x}(\mathrm{t}) = \mathbf{A}\Delta\mathrm{x}(\mathrm{t}) + \mathbf{B}\Delta\mathrm{u}(\mathrm{t}) \tag{15.18}$$

$$\Delta\,\mathrm{y}(\mathrm{t}) = \mathbf{C}\Delta\mathrm{x}(\mathrm{t}) \tag{15.19}$$

where

$$\mathbf{B} = \begin{bmatrix} \frac{\mathrm{df}_1}{\mathrm{du}_1} & \cdot & \cdot & \cdot & \frac{\mathrm{df}_1}{\mathrm{du}_n} \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ \frac{\mathrm{df}_n}{\mathrm{du}_1} & \cdot & \cdot & \cdot & \frac{\mathrm{df}_n}{\mathrm{du}_n} \end{bmatrix} \tag{15.20}$$

We have assumed a vector of inputs. In the scalar case the $\boldsymbol{B}$ Jacobian matrix is scalar. A further complexity is found if $y(t) = g(x(t))$ i.e. the output equation is also nonlinear. We then need to linearize it too:

$$\mathbf{C}=\begin{bmatrix} \frac{dg_1}{dx_1} & \cdot & \cdot & \cdot & \frac{dg_1}{dx_n} \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ \frac{dg_n}{dx_1} & \cdot & \cdot & \cdot & \frac{dg_n}{dx_n} \end{bmatrix} \tag{15.21}$$

In rarer cases we would also need a $\boldsymbol{D}$ matrix if $y(t) = g(\mathbf{x}(t), \mathbf{u}(t))$. Then

$$\mathbf{D}=\begin{bmatrix} \frac{dg_1}{du_1} & \cdot & \cdot & \cdot & \frac{dg_1}{du_n} \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ \frac{dg_n}{du_1} & \cdot & \cdot & \cdot & \frac{dg_n}{du_n} \end{bmatrix} \tag{15.22}$$

In the discrete-time case we have a very similar solution

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathrm{f}(\mathbf{x}_k, \mathrm{u}_k) \\ \mathrm{y}_k &= \mathrm{g}(\mathbf{x}_k, \mathrm{u}_k) \end{aligned} \tag{15.33}$$

The linearized system is

$$\begin{aligned} \Delta\mathbf{x}_{k+1} &= \mathbf{F}\Delta\mathbf{x}_k + \mathbf{G}\Delta\mathrm{u}_k \\ \Delta\mathrm{y}_k &= \mathbf{H}\Delta\mathbf{x}_k \end{aligned} \tag{15.34}$$

where

$$\mathbf{F} = \frac{\mathrm{df}(\mathbf{x}_k, \mathrm{u}_k)}{\mathrm{d}\mathbf{x}} = \begin{bmatrix} \frac{df_1}{dx_k^1} & \cdot & \cdot & \cdot & \frac{df_1}{dx_n^1} \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ \frac{df_n}{dx_k^n} & \cdot & \cdot & \cdot & \frac{df_n}{dx_k^n} \end{bmatrix} \tag{15.35}$$

$$\boldsymbol{G} = \frac{\mathrm{df}(\mathbf{x}_k, \mathrm{u}_k)}{\mathrm{d}\mathbf{u}_k} = \begin{bmatrix} \frac{df_1}{du_k^1} & \cdot & \cdot & \cdot & \frac{df_1}{du_n^1} \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ \frac{df_n}{du_k^n} & \cdot & \cdot & \cdot & \frac{df_n}{du_k^n} \end{bmatrix} \tag{15.36}$$

$$\mathbf{H} = \frac{\mathrm{dg}(\mathbf{x}_k, \mathrm{u}_k)}{\mathrm{d}\mathbf{x}_k} = \begin{bmatrix} \frac{\mathrm{dg}_1}{\mathrm{dx}_k^1} & \cdot & \cdot & \cdot & \frac{\mathrm{dg}_1}{\mathrm{dx}_n^1} \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ \cdot & & & \cdot & \cdot \\ \frac{\mathrm{dg}_n}{\mathrm{dx}_k^n} & \cdot & \cdot & \cdot & \frac{\mathrm{dg}_n}{\mathrm{dx}_k^n} \end{bmatrix} \tag{15.37}$$

The stability depends on the eigenvalues of $\boldsymbol{F}$ which must lie within the unit-circle of the z-plane.

Example 15.1. Continuous-Time Linearization

For our continuous-time autonomous example above:

$$\dot{\mathbf{x}}(\mathrm{t}) = \mathrm{f}(\mathbf{x}(\mathrm{t})) = \begin{bmatrix} -\mathrm{x}_1^2 - \mathrm{x}_2^2 + 1 \\ -\mathrm{x}_2 \end{bmatrix}$$

$$\frac{\mathrm{df}(\mathbf{x}(\mathrm{t}), \mathrm{u}(\mathrm{t}))}{\mathrm{d}\mathbf{x}} = \begin{bmatrix} -2\mathrm{x}_1 & -2\mathrm{x}_2 \\ 0 & -1 \end{bmatrix}$$

Substitute our two equilibrium state-vectors $\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}$ and we have two possible $\boldsymbol{A}$ matrices.

$\mathbf{A} = \begin{bmatrix} -2 & 0 \\ 0 & -1 \end{bmatrix}$ and $\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}$. These are diagonal matrices and the diagonal elements are thus their eigenvalues. The equilibrium state $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is therefore stable and $\begin{bmatrix} -1 \\ 0 \end{bmatrix}$ is unstable.

Example 15.2. Discrete-Time Linearization

Now consider a discrete-time example.

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathrm{f}(\mathbf{x}_k, \mathrm{u}_k) \\ &= \begin{bmatrix} 0.5\mathrm{x}_k^2 - \mathrm{x}_k^1\mathrm{x}_k^2 \\ -\mathrm{x}_k^1 + 0.5\mathrm{x}_k^2 + \mathrm{u}_k \end{bmatrix} \end{aligned}$$

Note that the superscript denotes the number of the state and not a raised power unless it is in brackets.

First we find the equilibrium states from solving the steady-state discrete-time simultaneous difference equations:

$$\begin{bmatrix} x^{1*} \\ x^{2*} \end{bmatrix} = \begin{bmatrix} 0.5x_k^{2*} - x^{1*}x^{2*} \\ -x^{1*} + 0.5x^{2*} + u^* \end{bmatrix}$$

We get two algebraic equations which we can solve for a particular input.

$$x^{1*}\left(1 + x^{2*}\right) = 0.5x^{2*} \tag{15.38}$$

and

$$x^{1*} = -0.5x^{2*} + u^* \tag{15.39}$$

Solving by substitution gives us

$$\left(x^{2*}\right)^2 + 2x^{2*}(1 - u^*) - 2u^* = 0$$

Let us take $u^* = 0.8$ and we get the quadratic

$$\left(x^{2*}\right)^2 + 0.4x^{2*} - 1.6 = 0 \tag{15.40}$$

This has solution $x^{2*} = 1.0806, -1.4806$ and by using (15.39) we find the corresponding values of x^{1*} =0.2596, 1.5403. The equilibrium states are therefore $(x^{1*}, x^{2*}) = (-0.1040,\ 1.0806)$ and $(1.5403,\ -1.4806)$.

Find the Jacobian matrix $\boldsymbol{F}$ by using (15.35)

$$\mathbf{F} = \begin{bmatrix} -x^{2*} & 0.5 - x^{1*} \\ -1 & 0.5 \end{bmatrix}$$

For the two equilibrium state-vectors we have two $\boldsymbol{F}$ matrices.

$$\mathbf{F} = \begin{bmatrix} -1.0806 & 0.5 - 0.257 \\ -1 & 0.5 \end{bmatrix} \text{ or } \mathbf{F} = \begin{bmatrix} 1.4806 & 0.5 - 1.5403 \\ -1 & 0.5 \end{bmatrix}$$

We also have from (15.36)

$$\mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The first of these Jacobian matrices has eigenvalues of magnitude −0.91, 0.3295 indicating stability at this equilibrium state. The second of these has eigenvalues of magnitude 2.12, −0.1414 indicating instability. MATLAB code is now shown to simulate the system with initial state conditions of [0.1,0.1] (Fig. 15.6).

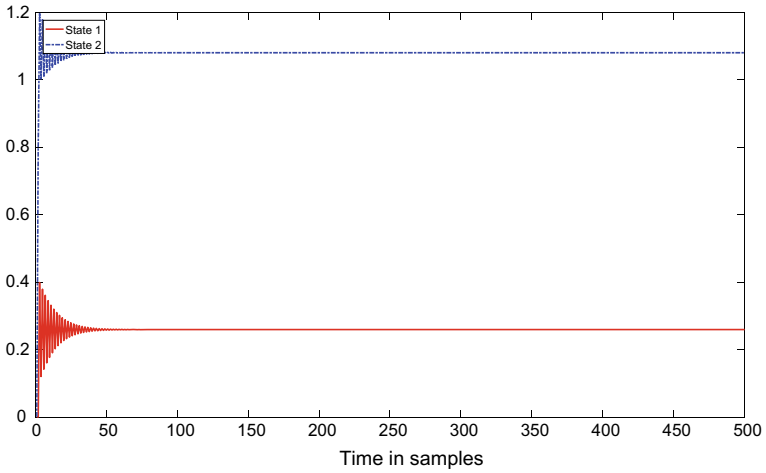


Fig. 15.6 Response of discrete-time nonlinear system to step input u = 0.8. Nonzero initial-conditions

MATLAB Code 15.1

```
%MATLAB Code 15.1
%Simulates nonlinear system
%No of points in simulation
N=500;
y=zeros(N,1);
x1=zeros(N,1);
x2=zeros(N,1);
t=[1:1:N];
  % Initial conditions on state vector
x1(1)=0.001;
x2(1)=0.001;
for   k = 1:size(x1)-1

x1(k+1)=0.5*x2(k)-x1(k).*x2(k);
x2(k+1)=-x1(k) +0.5*x2(k)+0.8;
end


%Now plot the states
plot(t,x1, '-r' ,t,x2, '-.b' )
legend( 'State 1' , 'State 2 ' , 'Location' , 'northwest'  )
xlabel({ 'Time in samples'  })
```

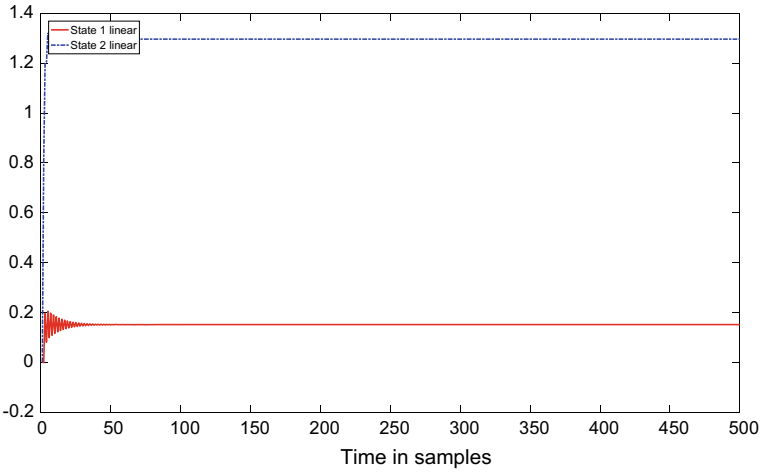


Fig. 15.7 Response of discrete-time linearized system

The states converge to (0.2596, 1.0806) as per the theory. By setting initial conditions further away from unity, we can get the unstable solution. In contrast, a stable *linear* system will *always* converge to a constant solution for the states no matter what the initial state values are. We also have that the stability of the system is governed by the magnitude of the input itself via Eq. (15.39). These two conditions are unheard of in a linear system.

We can now simulate the linear system $\Delta \mathbf{x}_{k+1} = \mathbf{F}\Delta \mathbf{x}_k + \mathbf{G}\Delta u_k$ and compare the responses of the states for the same initial conditions (Fig. 15.7).

The two system give fairly similar results.

MATLAB Code 15.2

```
%MATLAB Code 15.2
%Simulates nonlinear system and compares with linear
%No of points in simulation
N=500;
y=zeros(N,1);
x1=zeros(N,1);
x2=zeros(N,1);
t=[1:1:N];
  % Initial conditions on state vector
x1(1)=0.001;
x2(1)=0.001;
for   k = 1:size(x1)-1

x1(k+1)=0.5*x2(k)-x1(k).*x2(k);
x2(k+1)=-x1(k) +0.5*x2(k)+0.8;
end


%Now plot the states
plot(t,x1, '-r'  ,t,x2, '-.b' )
legend( 'State 1' , 'State 2 ' , 'Location' , 'northwest' )
xlabel({ 'Time in samples' })

% Now simulate the linearizsed system
x1(1)=0.001;
x2(1)=0.001;
for  k = 1:size(x1)-1

x1(k+1)=-1.0806*x1(k)+0.243*x2(k);
x2(k+1)=-x1(k) +0.5*x2(k)+0.8;
end

figure

%Now plot the states
plot(t,x1, '-r'  ,t,x2, '-.b' )
legend( 'State 1 linear' , 'State 2  linear'  , 'Location'  , 'northwest' )
xlabel({ 'Time in samples' })
```

We can also program the nonlinear system using LabView as shown in Fig. 15.8 (the graphical program) and Fig. 15.9 the front-panel showing the response.

Example 15.3. Continuous-Time Pendulum

The pendulum is a well worked example that appears in most textbooks on automatic control. Often it appears with its twin brother, the *inverted* pendulum. Consider the pendulum of mass m, length ℓ, swinging at some angle θ and with gravitational constant g. Figure 15.10 shows the pendulum.

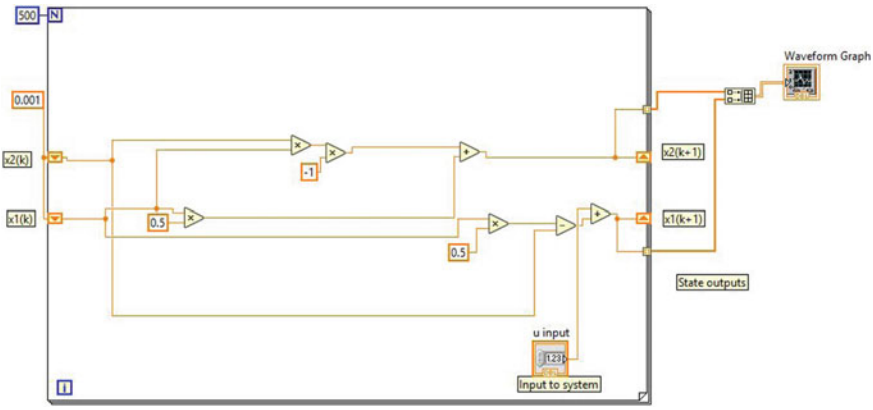


Fig. 15.8 Graphical program in LabView for nonlinear system

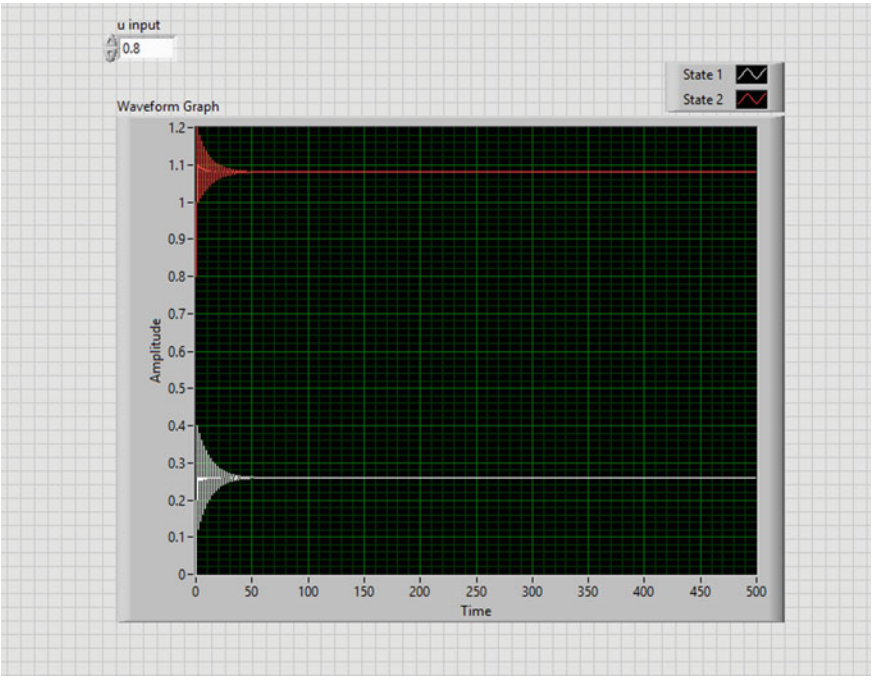


Fig. 15.9 Front panel of LabView program

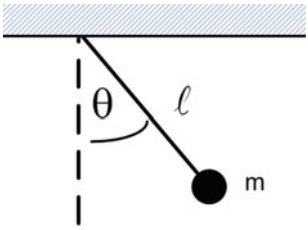


Fig. 15.10 Pendulum swinging

The equation is found by drawing a free-body diagram and using Newton's law for Torque.

$$\frac{\mathrm{d}^2\theta}{\mathrm{dt}^2} + \frac{\mathrm{K_d}}{\mathrm{m}}\frac{d\theta}{\mathrm{dt}} + \frac{\mathrm{g}}{\ell}\sin(\theta) = 0 \tag{15.41}$$

K_d represents the coefficient of viscous friction. Define as states $x_1 = \theta, x_2 = \frac{d\theta}{dt}$. The vector of states becomes

$$\mathrm{f}(\mathrm{x}) = \begin{bmatrix} \mathrm{f}_1(\mathrm{x}) \\ \mathrm{f}_2(\mathrm{x}) \end{bmatrix}$$

Where

$$\mathrm{f}_1(\mathrm{x}) = \dot{\mathrm{x}}_1 = \mathrm{x}_2 \tag{15.42}$$

$$\mathrm{f}_2(\mathrm{x}) = \dot{x}_2 = -\frac{\mathrm{g}}{\ell}\sin(\mathrm{x}_1) - \frac{\mathrm{K_d}}{\mathrm{m}}\mathrm{x}_2 \tag{15.43}$$

The two state derivatives are equated to zero for the equilibrium points giving two equations:

$\mathrm{x}_2 = 0$ and $-\frac{\mathrm{g}}{\ell}\sin(\mathrm{x}_1) - \frac{\mathrm{K_d}}{\mathrm{m}}\mathrm{x}_2 = 0$

We consider when $x_1 = \theta = 0, x_2 = \frac{d\theta}{dt} = 0$ i.e. $\left(\mathrm{x}_1^*, \mathrm{x}_2^*\right)$ = (0,0). Of course another equilibrium point exists when $\left(\mathrm{x}_1^*, \mathrm{x}_2^*\right)$ =(180,0), but this is when the pendulum is upside down. The Jacobian becomes

$$\boldsymbol{A} = \begin{bmatrix} \frac{\mathrm{df_1}}{\mathrm{dx_1}} & \frac{\mathrm{df_1}}{\mathrm{dx_2}} \\ \frac{\mathrm{df_2}}{\mathrm{dx_1}} & \frac{\mathrm{df_2}}{\mathrm{dx_2}} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{\mathrm{g}}{\ell}\cos\left(\mathrm{x}_1^*\right) & -\frac{\mathrm{K_d}}{\mathrm{m}} \end{bmatrix} \tag{15.44}$$

Now substitute the equilibrium state values when the angle is zero and we get $\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -\frac{\mathrm{g}}{\ell} & -\frac{\mathrm{K_d}}{\mathrm{m}} \end{bmatrix}$. Let us put some numbers into this problem. In SI units let g = 9.81 m/s^2 m = 0.1 kg, $\ell = 1$ m and $\mathrm{K_d} = 0.1$ N/m/s. Our Jacobian matrix becomes

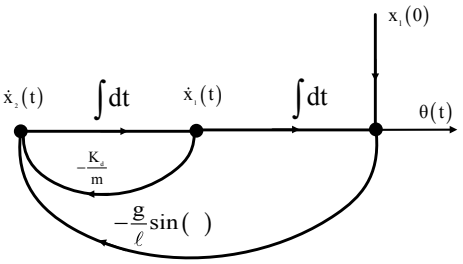


Fig. 15.11 Signal-flow graph for nonlinear pendulum

$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -9.81 & -1 \end{bmatrix}$ which has stable left-half plane eigenvalues of $-0.5 \pm$ j3.09. There are a number of ways to simulate the pendulum. The use of Simulink would be one method. Here we shall use LabView. In order to simulate a continuous-time system we need a signal-flow-graph, a bit like an old-fashioned analogue computer. In fact this is easier than an analogue computer since integrators in analogue computers needed to be scaled, and here we can use a Trapezoidal integrator as a pure 1/s term and put feedback around it. The signal-flow graph is shown in Fig. 15.11.

Each integrator is implemented directly using the Trapezoidal rule for integration as shown in Fig. 15.12. There is really no such thing as an analogue simulation on a digital computer. We can only approximate the analogue integrator with a digital one. The only true analogue simulation would be on an analogue computer using operational-amplifiers as integrators.

This is implemented as a sub vi in LabView. When combined with the rest of the signal-flow graph code we get the graphical representation shown in Fig. 15.13.

We set the initial-condition of the first state to be 60° and get the response as shown in Fig. 15.14.

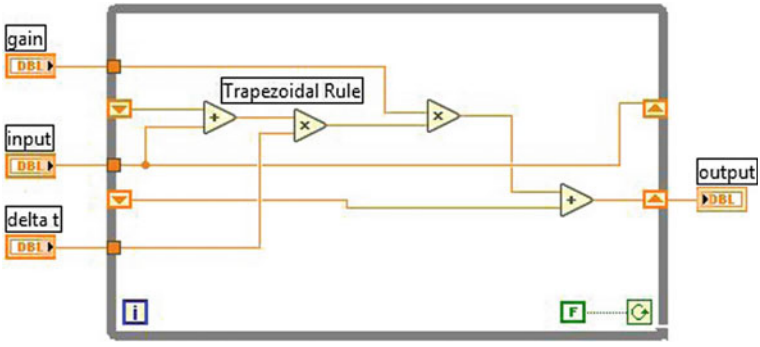


Fig. 15.12 Trapezoidal rule to implement an integrator

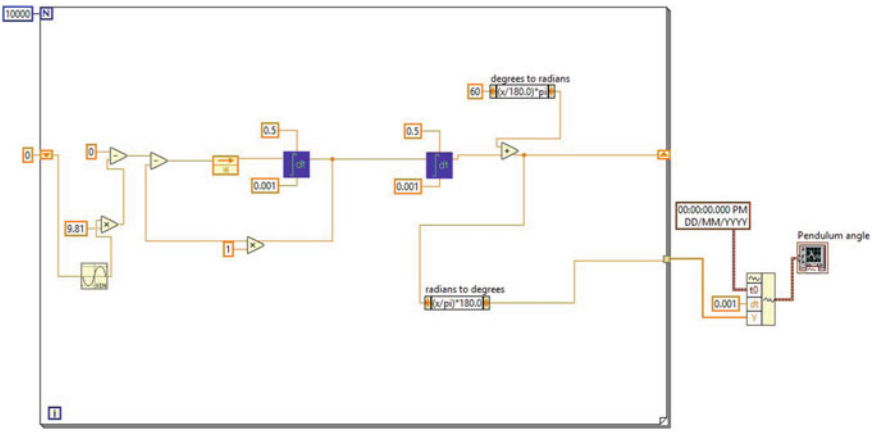


Fig. 15.13 Graphical LabView code to simulate nonlinear pendulum in continuous-time. Basic concept

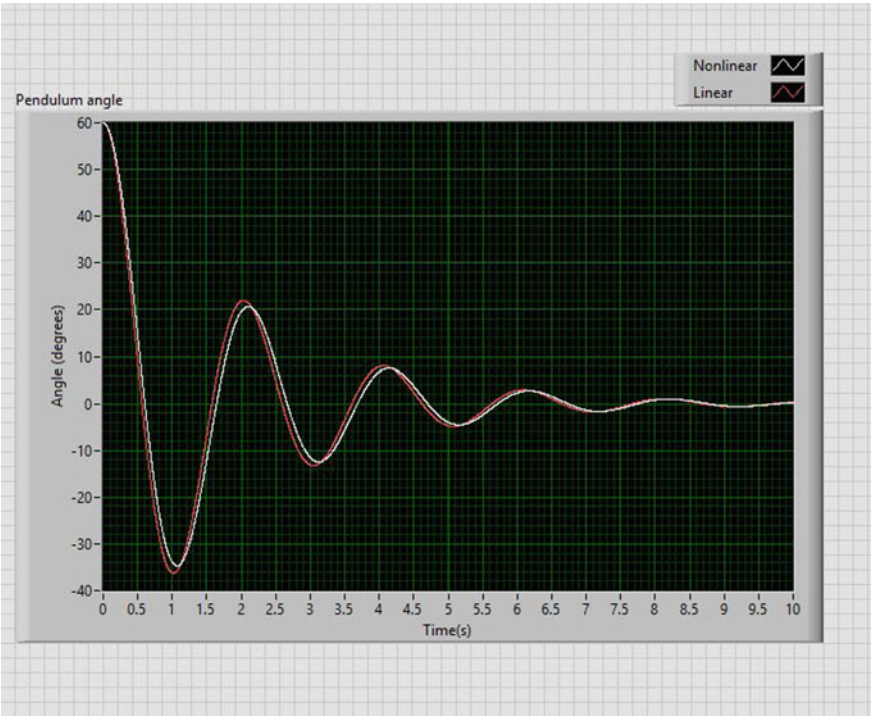


Fig. 15.14 Response of pendulum to 60-degrees initial-condition. Compares linear with nonlinear model

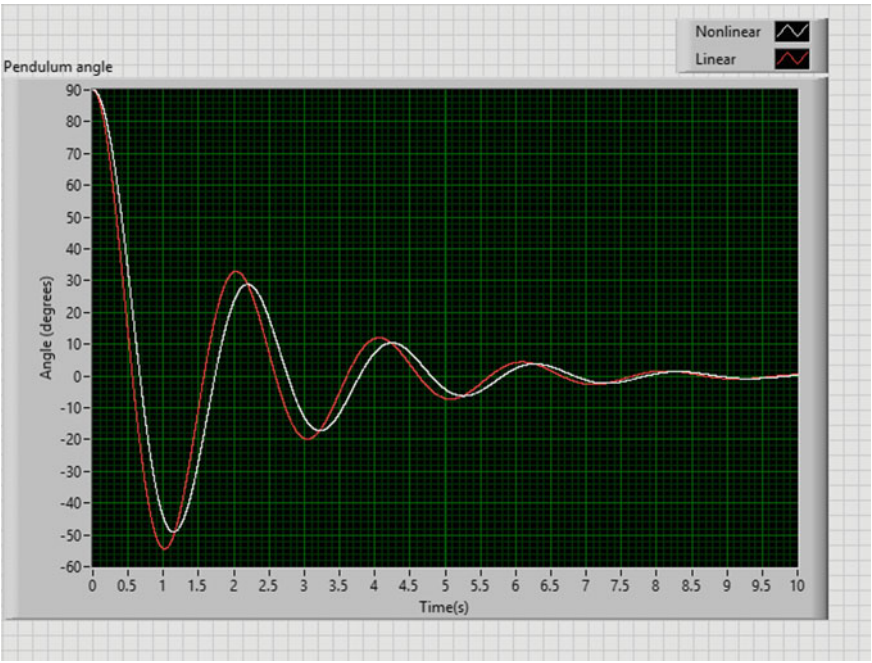


Fig. 15.15 Response of pendulum to 90-degrees initial-condition. Compares linear with nonlinear model. Shows greater error in liner model

The pendulum comes to rest after about 10 s. When the initial angle is increased to 90° the difference between the two models becomes more apparent as can be seen in Fig. 15.15.

Example 15.4. Upside Down Pendulum or Pole-Balancing Although at first sight this example looks somewhat frivolous, with no engineering application, it is in fact an important piece of theory and well worth investigating. The reason why, is that there are many similar types of control-systems in existence. For example, a rocket needs to be guided to maintain stability. Most rockets use steerable engines to achieve this and a closed-loop system. Then there is the Segway-type electric vehicles which are two-wheeled transportation that self-balance. Finally, robots that are humanoid in appearance need to be able to walk and an upside-down pendulum is a very simple approximation to a human balancing on their feet. For an upside down ordinary pendulum, we can ignore the effect of viscous friction because the force of gravity pulling it downwards will far outweigh it. Therefore the basic equation is:

$$\frac{d^2\theta}{dt^2} - \frac{g}{\ell}\sin(\theta) = 0 \tag{15.45}$$

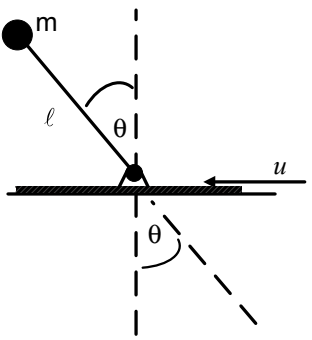


Fig. 15.16 Inverted pendulum and positioning horizontal displacement

If we mount the pendulum in such a way that a horizontal motion can be exerted to maintain vertical stability, we now have the diagram of Fig. 15.16.

The angle needs to be measured of course if a control-system is used and this can be done in a number of ways, the easiest of which is to put either an analogue potentiometer or a digital encoder on the turning hinge of the pendulum. This setup is most suited to an experiment with an x-y plotter and the pendulum is mounted near the pen-holder. The problem should not be confused with a similar yet subtly different problem of a pendulum of mass m on a cart with wheel of mass M which is covered in many textbooks.

With this addition, we get a new nonlinear differential equation caused by the angular acceleration introduced by the applied force.

$$\ell \frac{\mathrm{d}^2\theta}{\mathrm{dt}^2} - \mathrm{g}\sin(\theta) = -\frac{\mathrm{d}^2\mathrm{u}}{\mathrm{dt}^2}\cos(\theta) \tag{15.46}$$

We can learn from the ordinary pendulum that we can proceed far faster just by approximating $sin(\theta) \approx \theta$ and $\cos(\theta) \approx 1$ for small angles close to zero (The cosine term lies in the first quadrant so is positive). This will give us similar results to differentiating and finding equilibrium states. We get a linear approximation

$$\ell \frac{\mathrm{d}^2\theta}{\mathrm{dt}^2} - \mathrm{g}\theta = -\frac{\mathrm{d}^2\mathrm{u}}{\mathrm{dt}^2} \tag{15.47}$$

Taking Laplace-Transforms we get the interesting transfer-function

$$\frac{\theta}{\mathrm{u}}(\mathrm{s}) = -\frac{\mathrm{s}^2/\ell}{\mathrm{s}^2 - \frac{\mathrm{g}}{\ell}} \tag{15.48}$$

The system has two zeros at the origin and one stable and one unstable pole. For a one metre pendulum this transfer-function becomes

$$\frac{\theta}{u}(s) = -\frac{s^2}{s^2 - g} \tag{15.49}$$

To control this specific type of pendulum we need to cancel out the double zeros with a double integrator. What is left is an unstable system that can be controlled using lag-lead type control or PID. With long pendulums, although they are easier to control, they suffer from structural resonance of the shaft of the pendulum, and a lower bandwidth is often needed than is desirable. Other kinds of inverted pendulums (Segway or pendulum on two wheels) can be controlled either by state-space methods if an accurate model is known, or by simple tuning of PID. The pendulum on a cart is often shown with the added state of horizontal position so that not only is the pendulum balanced, it is also moved to a desired position. In the case above the pendulum is cruder in that there is no control in the horizontal plane.

15.3 Phase-Locked Loop (A Servo for Frequency-Demodulation)

The phase-locked loop (PLL) is a great example of a nonlinear system that for some reason is rarely if ever covered in control-system textbooks. It was discovered in 1932 by Frenchman Henri de Bellescize. In the early 1970s an integrated-circuit version appeared and the technique gathered much momentum. Primarily the PLL can be used to demodulate frequency-modulated (FM) signals in such applications as broadcast radio. It has been used in analogue and digital form and has been used to demodulated both analogue and digital signals. The block-diagram of a PLL is shown in Fig. 15.17.

In this application, FM is applied at the input and the demodulated output comes out of the filter. Before PLLs, FM was demodulated using more direct approaches of differentiation and filtering. The earliest approach was the Foster-Seeley discriminator. It is known that the PLL can give superior performance that these older

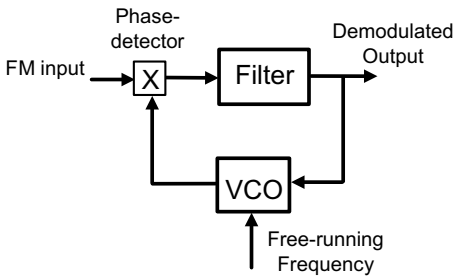


Fig. 15.17 PLL block-diagram

conversion methods and it found application in many hi-fidelity radio receivers of its day. With the advent of digital radio, it is less commonly found in this application, but is used in many varieties of digital communication applications. The phase-detector can be a linear multiplier in the case of analogue signals, though it has other forms when a logic signal is applied. If we take two waveforms with differing phase shifts, say $\cos(\omega t + \varphi_1)$ and $\cos(\omega t + \varphi_2)$, then when we multiply these together we get

$$\cos(\omega t + \varphi_1)\cos(\omega t + \varphi_2) = 0.5\cos(\varphi_1 - \varphi_2) + 0.5\cos(2\omega t + \varphi_1 + \varphi_2) \tag{15.50}$$

which is the classic sum and difference. Likewise, if we have a cos and sin term we can show

$$\sin(\omega t + \varphi_1)\cos(\omega t + \varphi_2) = 0.5\ \sin(\varphi_1 - \varphi_2) + 0.5\sin(2\omega t + \varphi_1 + \varphi_2) \tag{15.51}$$

In other words, if the two signals are in quadrature to one another then we get a similar formula. However, in this case we know that for small angles the *sin* term can be removed which leaves a phase-difference term and a term at twice the incoming frequency. If we filter out this 2ω term then we are left with just a classic summing junction to subtract two phase terms. Note that the waveforms *must be in quadrature* for this to happen and this is indeed in the case when "lock" happens. If there is no phase-shift then the phase-difference must be zero and nothing comes out of the filter in steady-state. Any dc that does come out of the filter drives a voltage-controlled oscillator (VCO) which moves the phase-shift of the feedback signal back into lock. Our filter is usually a low-pass filter but in terms of control-theory we can see it as a controller within a negative-feedback loop. The interesting thing about the VCO is that behaves like an integrator. In practice it is a square-wave oscillator (since sinewave oscillators are hard to make with controllable frequency) that is driven from a dc voltage. The input to the VCO is frequency and the output is phase. The transfer-function of the VCO is then a dynamic system of the form $\frac{\omega}{\varphi} = \frac{K_v}{s}$. This is rather unusual for a control-system in that we have an integrator in the feedback path. Usually we would never design such a system, but in this case we can see that for a closed-loop system with high-gain and an integrator in the feedback path, that its closed-loop transfer-function is $\frac{K}{1+\frac{K_v}{s}} = \frac{sK}{s+K_v}$. This is a high-pass filter or differentiator approximation at low-frequencies. We assume the filter above is just a simple gain, when in practice it must perform several functions. It must stabilise the loop and it must filter out the term at 2ω. There will always be some residual 2ω, but we do not want this to be too large as it in turn will multiply back into the input waveform at the phase-detector and cause

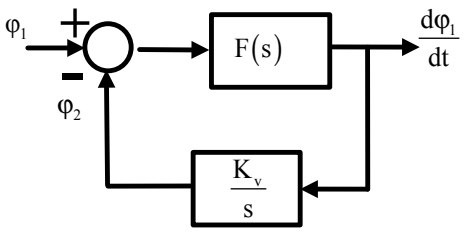


Fig. 15.18 Linearized model of the PLL

extra dc terms. The filter is therefore at its most primitive a low-pass filter and at its best some form of lag-lead compensator. In the block-diagram, the free-running frequency of the VCO is the frequency it runs at with no input voltage. It is important that this frequency is set up to be the same as the incoming frequency. Figure 15.18 shows the linearized model of the PLL.

15.3.1 Demodulation of Frequency-Modulation (FM)

The FM signal has the analogue form f(t) where

$$f(t) = \cos\left[\omega_c(t) + \Delta\omega \int \cos(\omega_m t)dt\right] \tag{15.52}$$

where ω_c and ω_m are respectively the carrier and base-band frequencies in rad/s and $\Delta\omega$ is the depth of modulation measured in rad/s. When Eq. (15.52) is integrated it has the more familiar form

$$f(t) = \cos[\omega_c(t) + \beta\sin(\omega_m t)] \tag{15.53}$$

Here $\beta = \Delta\omega/\omega_m$ is defined to be the FM modulation index.

The filter must be designed so as to attenuate the frequency generated by the loop at $2\omega_c$ rad/s. The simplest way to do this is to design the unity-gain crossover frequency (bandwidth of the loop) to be at $\omega_\phi = 2\omega_c/10$ rad/s. If we design the loop as the *ideal* Bode-plot of two integrators and a phase-advance, then the PLL is known as a type II PLL. One integrator is already present in the form of the VCO, so we need only add a second one. Overall we must have a filter of the form of an integrator cascaded with a phase-advance.

$$F(s) = \frac{K_1}{s}\left(\frac{1 + sT_1}{1 + sT_2}\right) \tag{15.54}$$

The gain of the VCO is assumed to be known, as information on it is usually given in the data-sheet of the PLL integrated circuit. If the VCO gain is unknown, it can be estimated by adding known time-constants to a filter in the loop to make it oscillate and measuring the resulting frequency of oscillation. We can then calculate the VCO gain mathematically. If we absorb the integrator gain and the filter gain into one gain, we have

$$\left|\frac{K}{s^2}\left(\frac{1 + sT_1}{1 + sT_2}\right)\right|_{\omega=\omega_\phi} = 1$$

where

$$K = K_1 K_v \tag{15.55}$$

The phase-advance must be designed such that it has a maximum phase-shift at $\omega = \omega_\phi$. The gain of a phase-advance at this frequency is

$$\left|\left(\frac{1 + sT_1}{1 + sT_2}\right)\right|_{\omega=\omega_\phi} = \sqrt{p} \tag{15.56}$$

where p is the span-ratio and is usually around 10 for a 55° phase-shift. Therefore from (15.55) we require a gain of

$$K = \frac{\omega_\phi^2}{\sqrt{p}} \tag{15.57}$$

The carrier frequency is not presented to a PLL since this is usually much too high to be practical. Instead, the intermediate-frequency (IF) is used as a standard. Depending on the receiver, this frequency is usually 10.7 MHz or 455 kHz. Good IF filtering usually happens before the PLL. We can simulate an equivalent PLL using the following MATLAB code by using a scaled model. The code is a digital representation of an analogue PLL in that digital integrators and phase-advance are used.

MATLAB Code 15.3

```
%Phase Locked Loop
%T.J.Moir 2019
%This simulates a PLL
%The VCO is sinusoidal but can be made a squarewave
%Suggested parameters: Sample at 10000Hz,Carrier freq 1000Hz = VCO
free running freq
% Make baseband freq 8Hz and frequency deviation 100Hz.
% Make run time 0.2 second
%When you run this you will see the demodulated sinewave (8Hz)
% It will have 2fcarrier (2fc) superimposed on top of it (as in a real PLL)

%First Generate Test Signal
%Frequency Modulation of a sine wave is simplest!
fs=input('Input Sampling Frequency (Hz)');
fb=input('Baseband Frequency (FM)');
fc=input('Carrier Frequency (FM)');
%fset=input('Free running frequency for VCO(make this normally = carrier freq)');
fset=fc;
delf=input('Frequency Deviation (Hz)');
%Compute FM Modulation Index
beta=delf/fb;
tmax=input('Run Time (Secs)');
npoints=tmax*fs;
t=[1:1:npoints]'/fs;
%%%%%%%%%%%%%%%%%%%%%%%%%
%Some arrays inialised
pout=zeros(npoints,1);
fm=zeros(npoints,1);
vco=zeros(npoints,1);
%%%%%%%%%%%%%%%%%%%%%%%%%

%Generate FM
in0=0;in1=0;
for i=1:1:npoints
 fm(i)=cos(2*pi*i*(fc/fs) + beta* sin(2*pi*i*(fb/fs))  );
end

%Unity Gain Crossover Frequency (Hz) for Bode Plot
fcp=(2*fc)/10;

%Make span ratio 10:1 gives a phase margin of around 57 degrees
%Change this and you change stability phase margin is arcsin[(span-1)/(span+1)]
spanp=10;
%This function computes phase advance (lead) parameters
[ap,bp,delp]=plead(spanp,fs,fcp);
```

```
%Compute Overall Gain

gainp=4*(1-cos (2*pi*(fcp/fs) ) )/sqrt(spanp);
%Divide up between Integrators
gainip=sqrt(gainp);

%Initialise just in case..
p0=0.0;
p1=0.0;
q0=0.0;
q1=0.0;
r0=0.0;
r1=0.0;
fu0=0.0;

%Main Loop for Phase Locked Loop

for i=1:1:npoints

%Controller (Filter)
%First Integrator
p1=p0;
p0=p1+fu0*gainip;

%fu0 is Phase Detector Output

%Second Integrator
q1=q0;
q0=q1+gainip*p0;

%Phase Advance
r1=r0;
r0=-r1*bp + delp*(q0+ap*q1);

%VCO Output
vco(i)=cos( (2*pi*i*(fset/fs) ) + r0 ) ;

%Phase Detector (A simple multiplier)
fu0=vco(i)*fm(i);

%PLL Output is first Integrator Output
pout(i)=p0;

end

figure(1)
plot(t,pout)
xlabel({'time(s)'})
ylabel({'Demodulated baseband'})
```

We also need a routine which calculates the digital phase-advance values.

```
function [a,b,del]=plead(span,fs,fc)
%Phase Lead
% Returns zero coefficient a
% Pole coefficient b
% and gain del
%span ratio is span
%fs is samplingfrequency
%fc is frenquency for maximum phase advance
thetac=2*pi*(fc/fs);
sqs=sqrt(span);
x=cos(thetac);
y=sin(thetac);
num1=sqs*(1+span)*y-2*span;
den1=sqs*(1-span)*y + 2*span*x;
a=num1/den1;
beta=(1-a)/(1+a);
b=(span-beta)/(span+beta);
del=(1+b)/(1+a);
```

In Fig. 15.19 we can just make out the superimposed 2 kHz feedthrough terms (generated from the 1 kHz carrier), but they are highly attenuated as in a real PLL. We can investigate the transient response and stability by demodulating a square-wave. This is shown in Fig. 15.20 and indicates a well damped response.

The PLL has a gain that is dependent on the amplitude of the incoming FM modulated carrier signal. This is because the phase-detector is a multiplier. In all FM demodulators a *hard limiter* is used to keep the amplitude at a constant value to avoid any problems with the gain changing with signal amplitude.

15.4 Amplitude-Locked Loop (A Servo Used to Remove Amplitude Variations)

The amplitude-locked loop (ALL) was patented by Archie Pettigrew in Scotland in 1991 [52]. It was designed to be the mathematical dual of the phase-locked loop (PLL). A block diagram of a typical ALL is shown in Fig. 15.21.

In comparison to a PLL, it has an AM (or any waveform with amplitude variation which is modulating a carrier waveform) waveform as its input. The purpose

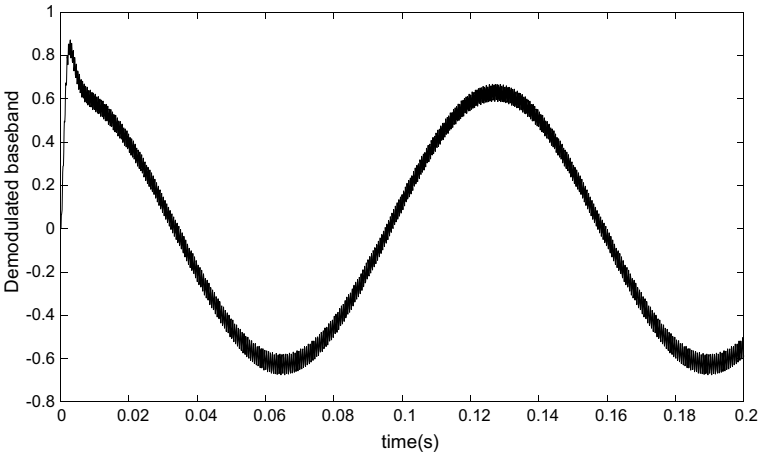


Fig. 15.19 PLL demodulated output of FM modulated signal. Sampling frequency 10 kHz, Baseband frequency 8 Hz, Carrier frequency and free running frequency 1 kHz. Frequency-deviation 100 Hz, runtime = 0.2 s

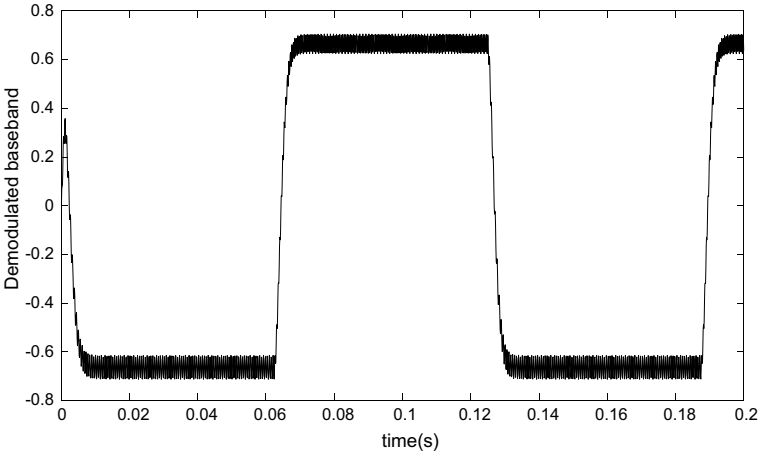


Fig. 15.20 Demodulated squarewave. Sampling frequency 10 kHz, Baseband frequency 8 Hz, Carrier frequency and free running frequency 1 kHz. Frequency-deviation 100 Hz, runtime = 0.2 s

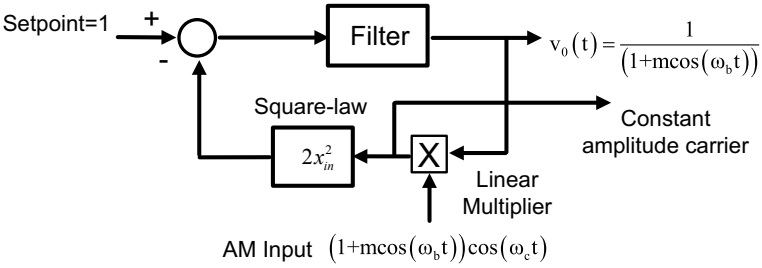


Fig. 15.21 Amplitude-locked loop block-diagram

of this control-system is to remove amplitude variations by servo action alone and not by the usual method of a hard-limiter. Hard limiters (followed by a bandpass filter) work well when the SNR is large, but with low SNRs the uncertainty in zero-crossings does not contribute any filtering action, even though the resulting waveform may well look as if all amplitude variations have been removed. This is because amplitude noise has been converted into phase-noise. The effect of this is to give inferior performance on FM receivers when it is presented to a PLL.

Instead of a phase-detector the ALL has an amplitude detector in the form of a square-law. The original ALL used a precision rectifier but the square-law gives us better performance due to the fact that there are less harmonics. The square-law produces a dc waveform plus a waveform at twice the carrier frequency in the same way as a PLL. It is scaled such that it gives out unity dc for a carrier of magnitude unity. Instead of a voltage-controlled oscillator in a PLL we have instead a voltage-controlled amplifier in the form of a linear multiplier. The filter is like the PLL filter, just a controller to stabilise the loop and a means of attenuating the twice carrier frequency term. Therefore, we select the bandwidth of the loop in the same way as a PLL. For a type II loop we need two integrators and a phase-advance, though we can operate with a type I loop (i.e. one integrator only). A type 0 loop with just a lowpass filter will not be good enough as a steady-state error will always exist, though for some applications it may be good enough.

The interesting signal that emerges from an ALL, apart from the carrier frequency devoid of amplitude variations, is the reciprocal modulation signal as shown in Fig. 15.21. This is the output of the filter. If the denominator goes to zero then

we have a division by zero, so in theory we should restrict ourselves to modulation indices of less than unity. In practice the ALL goes out of lock when this happens and jumps back into lock. A PLL does a similar thing for short periods of time when the FM signal disappears. This is a kind of free-wheeling effect like a rolling wheel going downhill over a rough surface. Its momentum is enough to keep it going.

The setpoint defines the amplitude of the recovered carrier and here it is set to unity. We should take care when comparing the ALL with a similar circuit, the automatic-gain control or AGC. An AGC has a much wider dynamic range (maybe over 100 dB) as compared to an ALL which only has around 20 dB dynamic range. The ALL has a high bandwidth and the AGC has a low bandwidth. More than often the VCA (voltage-controlled amplifier) in an AGC is nonlinear whereas it is linear in an ALL by virtue of a linear multiplier. In real applications an AGC should precede an ALL and then a PLL should come after the ALL. The reciprocal envelope signal in Fig. 15.21 shown as

$$v_0(t) = \frac{1}{(1 + m\cos(\omega_b t))} \tag{15.58}$$

This signal more than just a by-product of the loop and can be used as a special signal in other applications. For example, the term automatic-variance control was coined by Moir [53]. The ALL is used as a means of either controlling the envelope of a random signal or estimating the standard-deviation of a random signal. In [54], it is used to generate a signal to supress multipath interference in FM receivers. Multipath FM signals have spikes (often known as Rician spikes) which cause much distortion in an FM receiver. A signal similar to (15.58) is used to multiply the received signal by which in turn nulls these spikes.

MATLAB can be used to simulate a basic ALL of the form shown in Fig. 15.21.

MATLAB Code 15.4

```
%Amplitude Locked Loop
%T.J.Moir 2019
%This simulates an analogue ALL
%Suggested parameters: Sample at 10000Hz,Carrier freq 1000Hz
% Make baseband freq 2Hz
% Make run time 2 second
%First Generate Test Signal
 %Amplitude Modulation of a sine wave is simplest!
fs=input('Input Sampling Frequency (Hz)');
fb=input('Baseband Frequency (AM)');
fc=input('Carrier Frequency (AM)');

tmax=input('Run Time (Secs)');
npoints=tmax*fs;
t=[1:1:npoints]'/fs;
%%%%%%%%%%%%%%%%%%%%%%%%%%
%Some arrays inialised
pout=zeros(npoints,1);
am=zeros(npoints,1);
r0a=zeros(npoints,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate AM, modulation index m=0.9
m=0.9;
for i=1:1:npoints
am(i)=(1+m*cos(2*pi*i*(fb/fs)))*cos(2*pi*i*(fc/fs));
end
%Unity Gain Crossover Frequency (Hz) for Bode Plot
fcp=(2*fc)/10;
%Make span ratio 10:1 gives a phase margin of around 55 degrees
%Change this and you change stability phase margin is arcsin[(span-1)/(span+1)]
spanp=10;
%This function computes phase advance (lead) parameters
[ap,bp,delp]=plead(spanp,fs,fcp);
%Compute Overall Gain
gainp=(1-cos (2*pi*(fcp/fs) ) )/sqrt(spanp);
%Divide up between Integrators
gainip=sqrt(gainp);
%Initialise just in case..
p0=0.0;
p1=0.0;
q0=0.0;
q1=0.0;
r0=0.0;
r1=0.0;
```

```
fu0=0.0;
%setpoint = 1
setpoint=1;
%Main Loop for Amplitude - Locked Loop
for i=1:1:npoints
%Controller (Filter)
%First Integrator
p1=p0;
p0=p1+fu0*gainip;
%fu0 is rectifier (square-law output)
%Second Integrator
q1=q0;
q0=q1+gainip*p0;
%Phase Advance
r1=r0;
r0=-r1*bp + delp*(q0+ap*q1);
%r0 is reciprocal of modulation
r0a(i)=r0;
%A simple multiplier - recovered carrier is the output
rcar=r0*am(i);
%square-law for rectification
fu0=setpoint-2*rcar*rcar;
%ALL reciprocal of AM output is r0
pout(i)=rcar;

end
figure(1)
subplot(2,1,1)
plot(t,pout)
ylabel({'Recovered carrier'})
xlabel({'time(s)'})
subplot(2,1,2)
plot(t,r0a)
ylabel({'Reciprocal of envelope'})
xlabel({'time(s)'})
figure(2)
subplot(2,1,1)
plot(t,am)
ylabel({'AM Input'});
xlabel({'time(s)'})
subplot(2,1,2)
plot(t,r0a)
ylabel({'Reciprocal of envelope'})
xlabel({'time(s)'})
```

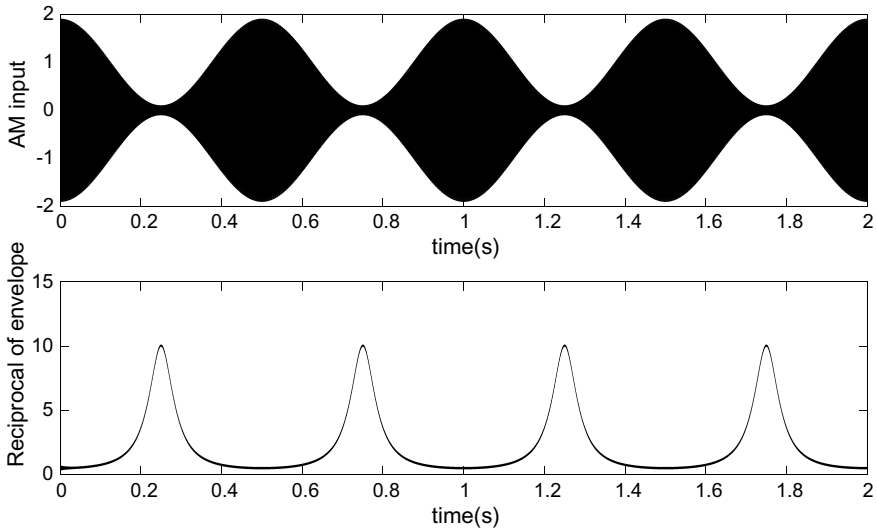


Fig. 15.22 AM input and reciprocal servo-action of envelope

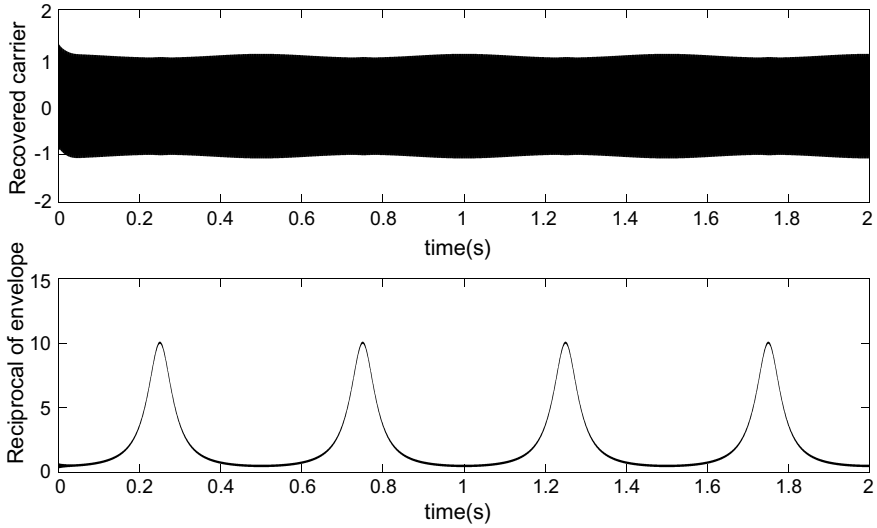


Fig. 15.23 Recovered carrier and reciprocal servo-action of envelope

Typical waveforms are shown in Figs. 15.22 and 15.23.

In a real type II ALL (with two integrators) some finesse needs to be taken with the integrators to avoid the loop from jumping into positive feedback when the modulation index is too large. A large transient could make this happen for instance if the transient manages to go negative. The ALL should be designed to lose lock and not jump into an unstable state. This is done by limiting the integrator travel with zener-diodes. In some of the later ALL applications a type I ALL is preferred for this reason.

15.5 Nonlinearity in the Feedback Path

Feedback has a wide variety of uses apart from the basic servo. In the analogue era, sometimes it became necessary to take the square-root of a signal. This was accomplished by applying negative feedback around an operational-amplifier and placing the inverse of the required nonlinearity in the feedback path. Provided the loop can be maintained in a stable state, the closed-loop system will always give the inverse of the nonlinearity in the feedback path. As an example consider Fig. 15.24.

The equations of this system are as follows. The system has an input $x(t)$, an output $w(t)$ and a square-law in the feedback path. It has an integrator in the forward path with a gain K. The error-signal is given by

$$e(t) = x(t) - w^2(t) \tag{15.59}$$

We also have that

$$\frac{dw(t)}{dt} = K\left(x(t) - w^2(t)\right) \tag{15.60}$$

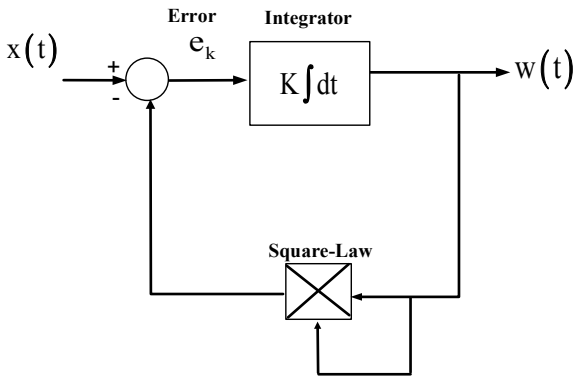


Fig. 15.24 Analogue square-root control-system. Multiplier in feedback-path

In steady state when $\frac{dw(t)}{dt} = 0$, we have the two equilibrium solutions

$$\begin{aligned} &\mathrm{K(x(t) - w^2(t)) = 0} \\ &\mathrm{w^* = \pm\sqrt{x}} \end{aligned} \tag{15.61}$$

We must ensure that the input $x(t)$ is always positive. The circuit converges to either positive or negative the square-root of the input x.

To find the gain K for stability we write (15.60) more generally

$$\frac{\mathrm{dw(t)}}{\mathrm{dt}} = \mathrm{g(w(t))} \tag{15.62}$$

where $g(w(t)) = K(x(t) - w^2(t))$.

For stability, by Lyapunov's indirect method we require

$$\left.\frac{\mathrm{dg(w(t))}}{\mathrm{dw}}\right|_{\mathrm{w=w^*}} < 0 \tag{15.63}$$

This results in the expression

$$-2\mathrm{w^*(t)K} < 0 \tag{15.64}$$

Substituting the two equilibrium values gives us $K > 0$ when $w^* = \sqrt{x}$ and $K < 0$ for $w^* = -\sqrt{x}$. Hence if the gain of the loop is positive, we get a positive square-root and a negative square-root if the gain is negative. The sign of the gain K determines the sign of the output square-root.

Now consider the discrete-time equivalent as shown in Fig. 15.25.

Consider Fig. 15.25 when $x_k > 0$ and η is a small real gain or step-size. The recursive algorithm becomes the nonlinear *difference* equation

$$\mathrm{w_{k+1} = w_k - \eta w_k^2 + \eta x_k} \tag{15.65}$$

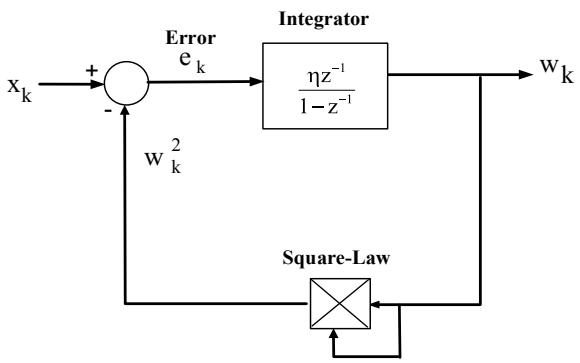


Fig. 15.25 Discrete-time square-root control-system

Consider the case when the input is held at a constant $x_k = x$, the equilibrium points when $w_{k+1} = w_k$ are

$$\mathrm{w}^* = \pm\sqrt{\mathrm{x}} \tag{15.66}$$

Initially we consider the positive root only since the square-root has two possible solutions. Write

$$\begin{aligned} \mathrm{w}_{\mathrm{k}+1} &= \mathrm{g}(\mathrm{w}_{\mathrm{k}}) \\ &= \mathrm{w}_{\mathrm{k}} - \eta\,\mathrm{w}_{\mathrm{k}}^2 + \eta\,\mathrm{x} \end{aligned} \tag{15.67}$$

Using Lyapunov's first or indirect method as applied to discrete-time systems, the condition for asymptotic stability is that provided g is continuously differentiable at w^* then $\left|\frac{\mathrm{d}}{\mathrm{dw}^*}\mathrm{g}(\mathrm{w}^*)\right|_{\mathrm{w}^*=\sqrt{\mathrm{x}}} < 1$

Giving

$$\left|1 - 2\eta\sqrt{\mathrm{x}}\right| < 1 \tag{15.68}$$

So that $0 < \eta < \frac{1}{\sqrt{x}}$. For positive and negative roots this gives the upper limits on the step-size as:

$$-\frac{1}{\sqrt{\mathrm{x}}} < \eta < \frac{1}{\sqrt{\mathrm{x}}} \tag{15.69}$$

We can easily simulate this system and show that the upper limit on the step-size is correct.

We choose $x_k = 2$ and the upper limit om step-size is by (15.69) that $\eta < \frac{1}{\sqrt{\mathrm{x}}} < 0.707$. If we choose 0.7 so it is close to instability then we can simulate this problem using the MATLAB code below.

MATLAB Code 15.5

```
%Square-root discrete-time system

%Number we need to take square-root of
x=2;
%Define gain, or step-size - must be less than 1/(sqrt(x)
gain=0.7;
% number of points in simulation
npoints=500;

%define array
wout=zeros(npoints,1);
%Main Loop for square-root algorithm
w=0;
for i=1:1:npoints
e=x-w*w;
w=w+gain*e;
wout(i)=w;
end

plot(wout)
xlabel({'time(samples)'})
ylabel({'Square-root versus time'})
```

Figure 15.26 shows the convergence to the square root of 2.

We can slightly modify the code to show the effect of a time-varying signal instead of a constant dc at the input. Let $x_k = 1 + \cos(2\pi 0.02k)$, which never goes negative. Figure 15.27 shows the true square-root and the square-root as obtained using nonlinear feedback.

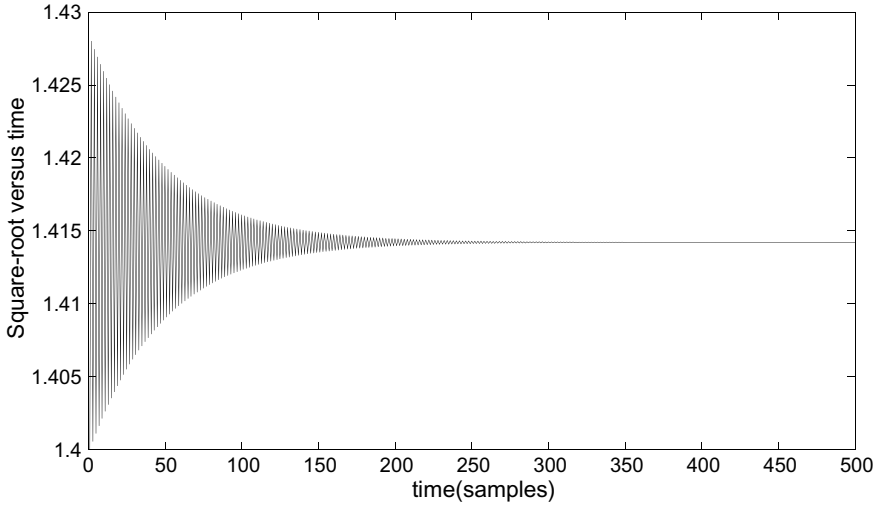


Fig. 15.26 Square-root nonlinear control-system. Deliberate choice of gain near instability

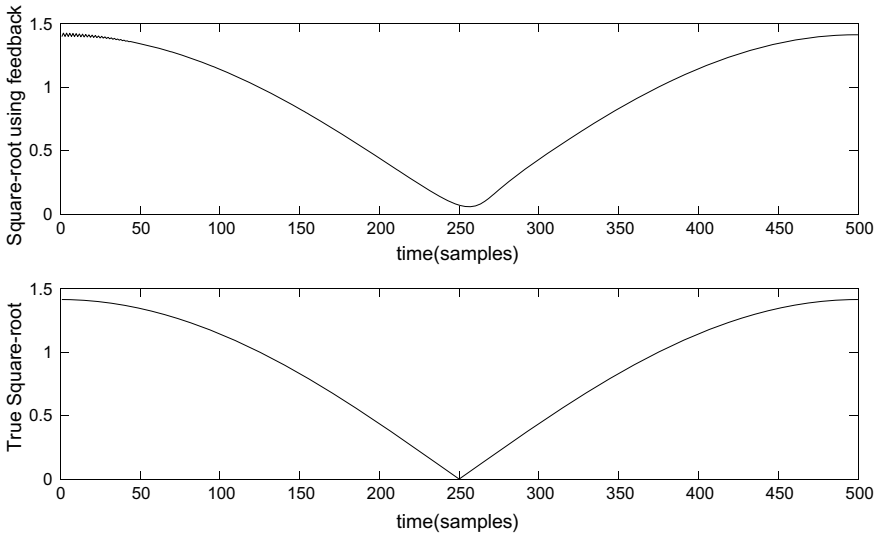


Fig. 15.27 Comparison of square-root (bottom) and true square-root (top) for ac signal

MATLAB Code 15.6

```
%Square-root discrete-time system
%Define gain, or step-size  - must be less than 1/(sqrt(x)
gain=0.7;
% number of points in simulation
npoints=500;
%define array
wout=zeros(npoints,1);
%Main Loop for square-root algorithm
w=0;
for   i=1:1:npoints
   x=1+cos(2*pi*0.002*i);
e=x-w*w;
w=w+gain*e;
wout(i)=w;
end
%For true waveform we calculated directly
t=[1:1:npoints];
wtrue=sqrt(1+cos(2*pi*0.002*t));
subplot(2,1,1)
plot(wout)
xlabel({ 'time(samples)' })
ylabel({ 'Square-root using feedback' })
subplot(2,1,2)
plot(wtrue)
xlabel({ 'time(samples)' })
ylabel({ 'True Square-root' })
```

Like any control-system the estimate (tracking) will get worse as we go up in frequency and the loop loses bandwidth. We could of course improve the loop gain by adding a second integrator with phase-lead, and so on.

Chapter 16
Feedback in Mathematical Algorithms

We have seen how feedback was invented by engineers to control physical systems. A whole design science has evolved around it. In the previous chapter we have touched upon nonlinear systems theory and showed how the square-root of a number can be found using feedback. This is the starting point where we can begin to see that in the field of mathematical algorithms, the practitioners have been using feedback for centuries with even being aware of it. We begin with a classical mathematical algorithm.

16.1 Gradient Descent Method as a Control-System

The gradient descent method should not be confused with the method of steepest descent in calculus. Gradient descent is an algorithm used in many applications for optimization in order to find the minimum or best fit in the sense of least-squares. Cost-functions other than least-squares can also be considered but the quadratic cost function is most popular due to simplicity and uniqueness of solution. A quadratic surface has a unique minimum like an upside-down hill. In an attempt to find the minimum of some kind of quadratic surface, we find that the best direction is to travel in the direction of negative gradient. This was discovered by the famous French mathematician Cauchy in 1847 [55]. For a single variable problem (we will use the square-root problem as before), we create a cost-function from an error:

$$e_k = \left(x_k - w_k^2\right) \tag{16.1}$$

The cost is the squared-error (scaling by the fraction does not affect the result).

$$J = \frac{1}{4}(e_k)^2 \tag{16.2}$$

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_16

We take its derivative or gradient

$$\frac{\partial \mathrm{J}}{\partial \mathrm{w_k}} = -\frac{1}{2} 2\mathrm{w}_k \left(\mathrm{x} - \mathrm{w_k^2}\right) \tag{16.3}$$

$$= -\mathrm{w}_k \mathrm{e_k} \tag{16.4}$$

Gradient descent gives us the following update in the negative direction of this slope.

$$\mathrm{w_{k+1}} = \mathrm{w_k} + \eta \mathrm{w_k e_k} \tag{16.5}$$

We can find the limits on stability for a given step-size by writing (16.5) as the nonlinear difference equation

$$\begin{aligned} \mathrm{w_{k+1}} &= \mathrm{g(w_k)} \\ &= \mathrm{w_k} + \eta \mathrm{w_k}\left(\mathrm{x} - \mathrm{w_k^2}\right) \end{aligned} \tag{16.6}$$

In steady state we must have that the *three* equilibrium points are

$$\mathrm{w}^* = \pm\sqrt{\mathrm{x}},\ \mathrm{w}^* = 0 \tag{16.7}$$

The condition for asymptotic stability is that provided $\mathrm{g_k}$ is continuously differentiable at w^* then $\left|\frac{d}{dw^*} g(w^*)\right|_{w^*=\sqrt{x}} < 1$

Giving

$$\left|1 + \eta \mathrm{x} - 3\eta (\mathrm{w}^*)^2\right| < 1 \tag{16.8}$$

For positive or negative step-size we then have

$$-\frac{1}{\mathrm{x}} < \eta < \frac{1}{\mathrm{x}} \tag{16.9}$$

Which differs slightly from the basic control case of the previous chapter. The block-diagram of the gradient descent method can be seen to be a nonlinear feedback system as shown in Fig. 16.1.

This is an interesting method not seen in ordinary linear control-systems. The output multiplies the error. The third equilibrium point is zero and can be avoided by choosing the initial conditions to be non-zero.

The performance of this method is not so good at high frequencies than the ordinary control method. The frequency had to be reduced to get the results and they lack perfect tracking as can be seen from Fig. 16.2. The MATLAB code is shown below. If the initial conditions are chosen to be negative rather than positive,

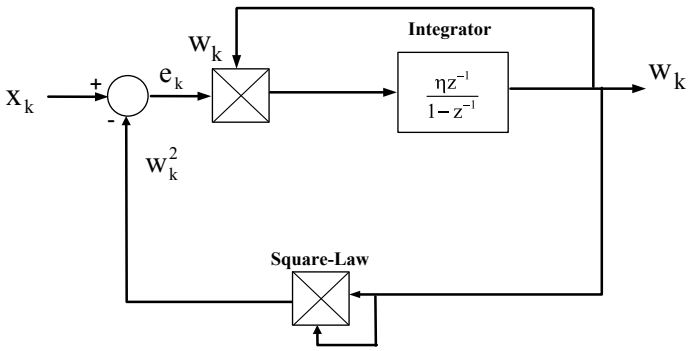


Fig. 16.1 Gradient method for square-root of a signal

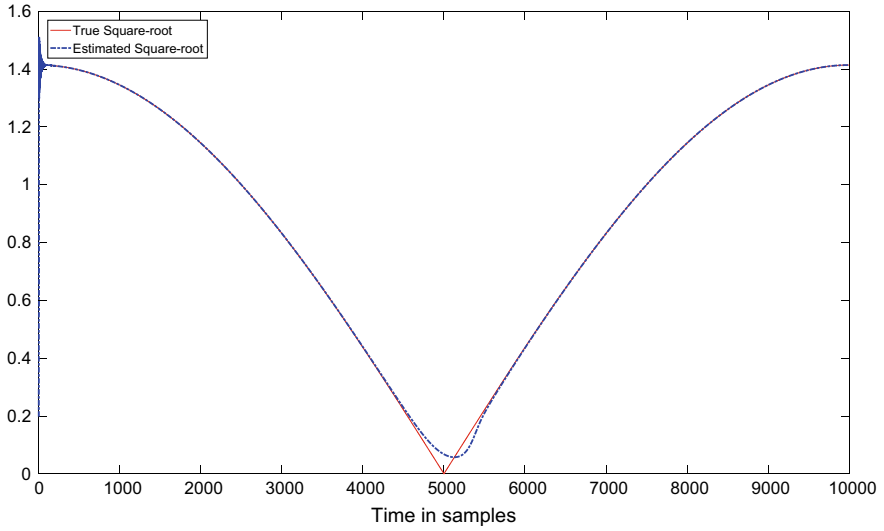


Fig. 16.2 Comparison of true square-root and gradient descent method. Loop begins to lose tracking ability at a normalised frequency of 0.0001

then the negative square-root equilibrium is the result. The gradient descent method is known to have slow convergence, nevertheless it is interesting to think that such a nonlinear control-system existed in the mid 19th century, at least on paper.

MATLAB Code 16.1

```
%Square-root discrete-time system using Gradient descent
%Define gain, or step-size  - must be less than 1/x
gain=0.49;
% number of points in simulation
npoints=10000;
%define array
wout=zeros(npoints,1);
%Main Loop for gradient descent algorithm
w=0.1;
for i=1:1:npoints
   x=1+cos(2*pi*0.0001*i);
  e=x-w*w;
w=w+gain*w*e;
wout(i)=w;
end
%For true waveform we calculated directly
t=[1:1:npoints];
wtrue=sqrt(1+cos(2*pi*0.0001*t));
%Now plot true and estimated square-roots
plot(t,wtrue,'-r',t,wout,'-.b')
legend('True Square-root','Estimated Square-
root','Location','northwest')
xlabel({'Time in samples'})
```

16.2 Newton's Method as a Control-System

Newton's method (or Newton-Raphson) for finding the square-root is a method which has quadratic convergence and is by far the fastest method we have of these examples. The reason it is fast is because the gain is not constant but is time-varying. The Newton method to find the square-root of x uses a tangent line to a curve $\mathrm{f}(\mathrm{w_k}) = \mathrm{w_k^2} - \mathrm{x}$ of the form

$$\mathrm{y} = \mathrm{f}'(\mathrm{w_k})(\mathrm{w} - \mathrm{w_k}) + \mathrm{f}(\mathrm{w_k}) \tag{16.10}$$

where w is the square-root of x at time-index k and $f'(w_k)$ is the derivative. The root happens at the intersection of the horizontal axis which is when $y = 0$. Now we replace w with the new estimate by relabelling $\mathrm{w} = \mathrm{w_{k+1}}$. Then re-arranging (16.10) we get

$$0 = (\mathrm{w_{k+1}} - \mathrm{w_k}) + \frac{\mathrm{f}(\mathrm{w_k})}{\mathrm{f}'(\mathrm{w_k})} \tag{16.11}$$

and hence the commonly known relationship

$$w_{k+1} = w_k - \frac{f(w_k)}{f'(w_k)} \tag{16.12}$$

Now $f(w_k) = w_k^2 - x$, so its derivative is $f'(w_k) = 2w_k$. Now (16.2) becomes

$$w_{k+1} = w_k - \frac{\left(w_k^2 - x\right)}{2w_k} \tag{16.13}$$

Now the interesting part. We again relabel, by noticing that $e_k = \left(x - w_k^2\right)$ is in fact an error term. Equation (16.13) is now

$$w_{k+1} = w_k + g_k e_k \tag{16.14}$$

where $g_k = \frac{1}{2w_k}$ is a time-varying gain. A closer examination of (16.14) will reveal that it is in fact a negative-feedback system with an input x, a time-varying gain and an output w. It has an integrator within the loop which does the updates. Figure 16.3 shows the block-diagram of the Newton method. There is no constant gain to find the limits on for stability. In fact the Newton method is not guaranteed to converge, but here it will always converge for the right initial condition. We cannot select zero as an initial-condition as the gain will be infinite. A MATLAB example of the working of this method is given.

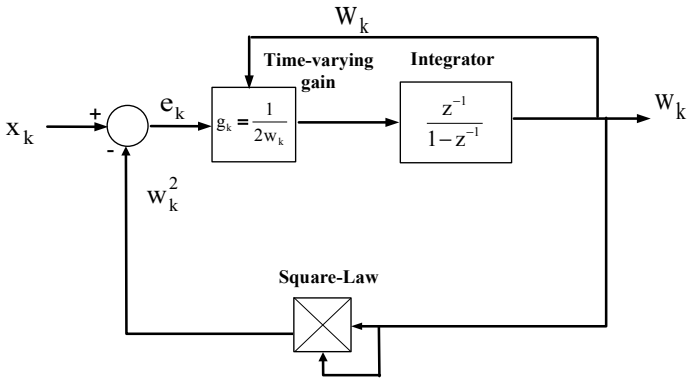


Fig. 16.3 Newton-Raphson method for finding a square-root

MATLAB Code 16.2

```
%Square-root discrete-time system using Newton-Raphson method
% number of points in simulation
npoints=1000;
%define array
wout=zeros(npoints,1);
%Main Loop
w=0.1;
for i=1:1:npoints
    gain=1/(2*w);
   x=1+cos(2*pi*0.001*i);
  e=x-w*w;
w=w+gain*w*e;
wout(i)=w;
end
%For true waveform we calculated directly
t=[1:1:npoints];
wtrue=sqrt(1+cos(2*pi*0.001*t));
%Now plot true and estimated square-roots
plot(t,wtrue,'-r',t,wout,'-.b')
legend('True Square-root','Estimated Square-
root','Location','northwest')
xlabel({'Time in samples'})
```

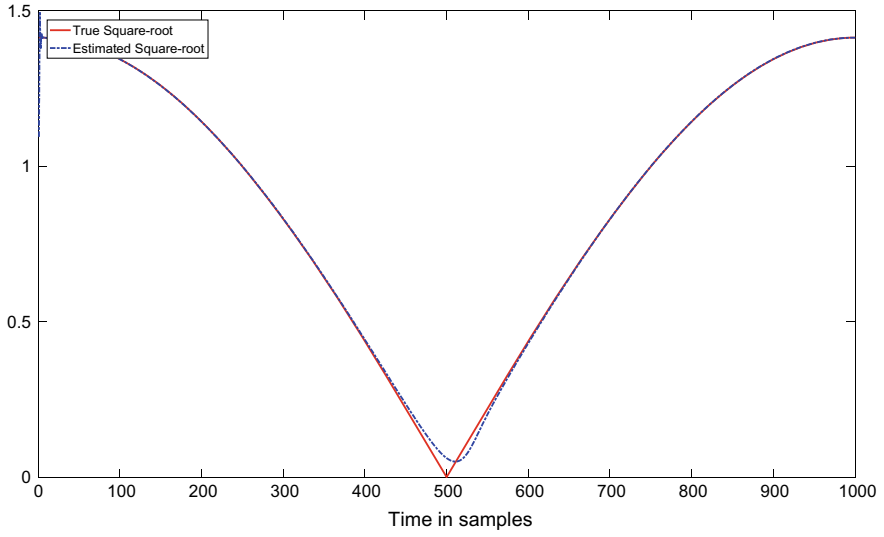


Fig. 16.4 Tracking of the Newton-Raphson method. Loop begins to lose tracking ability at a normalised frequency of 0.001

Figure 16.4 shows the performance of the algorithm for a similar time-varying signal as before. It has a far higher bandwidth than the gradient method because it has a time-varying gain which gives fast convergence.

It is at least ten times faster than the gradient method before it begins to lose tracking ability.

16.3 Division by Using Feedback

Now we consider the problem of dividing two real numbers *b/a* using negative feedback. This is once again a nonlinear problem, since to perform division we require multiplication in the feedback path. Although we consider real numbers for initial analysis, the methods will work with signals as with the square-root problem already discussed. For all cases we require $a \neq 0$ to avoid division by zero. The case when $b = 0$ is rather trivial since the answer is always zero.

16.3.1 Continuous-Time Cases

Consider the diagram of Fig. 16.5.

This loop is in fact linear but time-varying if *a* varies with time. We can still apply the same theory as before however as the linear case is a special case of the nonlinear theory. We can write the differential equation of this loop as follows:

$$\frac{\mathrm{dw(t)}}{\mathrm{dt}} = \mathrm{K(b - w(t)a)} \tag{16.15}$$

In steady state, when $\frac{dw(t)}{dt} = 0$, assuming the closed-loop system is stable, we have the equilibrium solution when $\mathrm{b - w(t)a} = 0$

$$\mathrm{w^* = \frac{b}{a}} \tag{16.16}$$

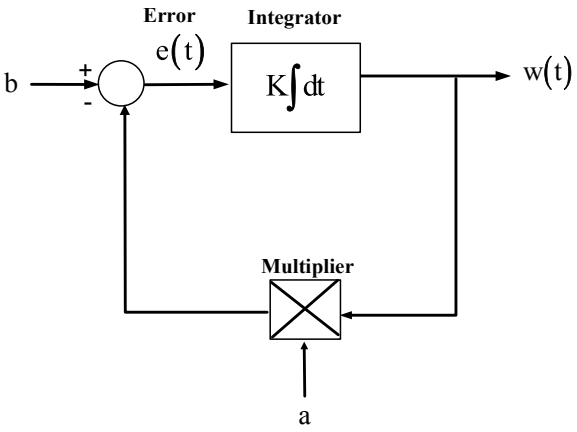


Fig. 16.5 Division using feedback

So the circuit gives an output which is the division we require. To find the gain K for stability we write (16.15) more generally

$$\frac{\mathrm{dw(t)}}{\mathrm{dt}} = \mathrm{g(w(t))} \tag{16.17}$$

where $g(w(t)) = K(b - w(t)a)$. Then using Lyapunov's first or indirect method

$$\left.\frac{\mathrm{dg(w(t))}}{\mathrm{dw}}\right|_{\mathrm{w=w^*}} < 0 \tag{16.18}$$

giving

$$-\mathrm{aK} < 0 \tag{16.19}$$

So for stability of the closed-loop system we require sgn(a) = $sgn(K)$. This avoids positive-feedback and instability. In theory the system is always stable if we get the right sign for negative feedback. We can modify this approach by using the gradient method already discussed. The gradient is found by differentiating the error-squared cost-function which is

$$\mathrm{J} = \frac{1}{2}(\mathrm{b} - \mathrm{w(t)a})^2 \tag{16.20}$$

The half only makes the calculation less messy and does not affect the overall result. This gives us a gradient

$$\frac{\partial \mathrm{J}}{\partial \mathrm{w(t)}} = -\mathrm{a(b} - \mathrm{w(t)a}) \tag{16.21}$$

The differential equation is then

$$\frac{\mathrm{dw(t)}}{\mathrm{dt}} = \mathrm{Ka(b} - \mathrm{w(t)a}) \tag{16.22}$$

The gradient method is rarely expressed in continuous-time, but the reality is that it can be discrete or continuous as with any differential-equation. Any difference equation has an equivalent differential equation in the limit when the sampling rate

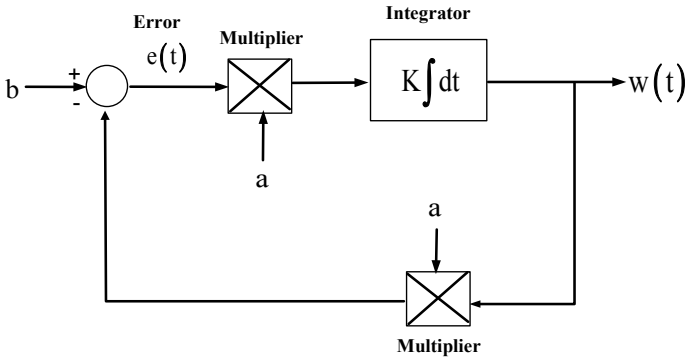


Fig. 16.6 Continuous-time gradient-descent method for division

is fast-enough. We get a slightly different block-diagram here as shown in Fig. 16.6.

An analysis for stability yields

$$-a^2K<0 \tag{16.23}$$

In this case the sign of a makes no difference and the loop is always stable for *positive* gain K.

Finally for the continuous-time case we can consider the Newton-Raphson method. The error is $e(t) = (b - w(t)a)$ and its derivative is

$$e'(t) = -a \tag{16.24}$$

The continuous-time Newton method is mentioned in the literature as a means of solving sets of nonlinear equations [56, 57].

The Newton update is therefore in continuous-time $\frac{dw(t)}{dt} = -\frac{e(t)}{e'(t)}$ (see (16.12))

$$\frac{dw(t)}{dt} = \frac{1}{a}(b - w(t)a) \tag{16.25}$$

However, this result does not progress the theory in any way since we still require a division by a. We can progress further by defining a different error $\varepsilon(t) = \left(\frac{1}{w(t)} - a\right)$ for $b = 1$ only. Then $\varepsilon'(t) = -\frac{1}{w^2(t)}$ and we have

$$\frac{dw(t)}{dt} = w(t)(1 - w(t)a) \tag{16.26}$$

We now can redefine the error when we draw the block-diagram as $e(t) = (1 - w(t)a)$. We can solve the nonlinear differential equation (16.26) directly using *Mathematica* to show that it does in fact give the desired result.

The code is only one line long

$$\text{DSolve}[\{w'[t] == w[t](1 - a * w[t]), w[0] == \text{w0}\}, w[t], t]$$

and the result is

$$\left\{\left\{w[t] \to \frac{e^t \text{w0}}{1 - a\text{w0} + ae^t \text{w0}}\right\}\right\}$$

In more familiar form this becomes

$$\text{w(t)} = \frac{\text{w}_0}{\text{e}^{-\text{t}}(1 - \text{aw(t)}) + \text{aw}_0} \tag{16.27}$$

This becomes $\frac{w_0}{aw_0}$ in the limit as $t \to \infty$.

Therefore, provided the integrator is not initialised at zero the loop provides the required division solution. Note that this is true for $b = 1$ only, so we can only find the reciprocal of a. Figure 16.7 shows the new block-diagram. Examining (16.26) we can see that $\text{w} = 0$ as an equilibrium point for the nonlinear system and this is why we cannot initialise at zero as the output of the loop will be zero.

16.3.2 Discrete-Time Cases

The discrete-time case has the same basic structure as continuous-time though the equations are of course slightly different and so are the stability margins.

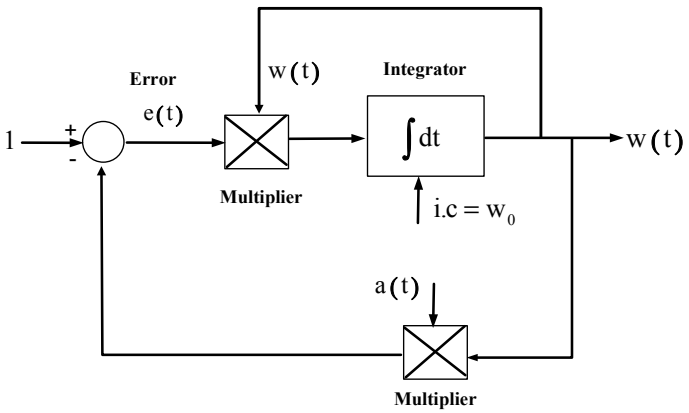


Fig. 16.7 Continuous-time Newton method of division (reciprocal)

The recursive algorithm becomes the *linear difference* equation

$$w_{k+1} = w_k + \eta(b - aw_k) \tag{16.28}$$

The equilibrium points occur when $w_{k+1} = w_k$ giving

$$w^* = \frac{b}{a} \tag{16.29}$$

Then write

$$\begin{aligned} w_{k+1} &= g_k \\ &= w_k + \eta(b - aw_k) \end{aligned} \tag{16.30}$$

and the condition for asymptotic stability is that provided g_k is continuously differentiable at w^* then $\left|\frac{d}{dw^*} g(w^*)\right|_{w^*=\frac{b}{a}} < 1$

Giving

$$|1 - a\eta| < 1 \tag{16.31}$$

Depending on the sign of a, the stability margins are:

$$-\frac{2}{a} < \eta < \frac{2}{a} \tag{16.32}$$

Define $\lambda = 1 - a\eta$. We can then substitute k = 0, 1, 2 …in (16.30) and obtain by repeated substitution the formula

$$w_n = \lambda^n w_0 + b\sum_{i=0}^{n-1} \lambda^i \eta \tag{16.33}$$

Since from (16.31), $|\lambda| < 1$, the summation part of (16.33) is a geometric series with growth-factor λ. It is then clear that the first part of (16.33) dies out, and the remaining sum of (16.33) becomes in the limit

$$\lim_{n \to \infty} w_n = 0 + \frac{b\eta}{1 - \lambda} \to \frac{b}{a} \tag{16.34}$$

The discrete-time gradient-descent case has also has a linear difference equation given by

$$w_{k+1} = w_k + \eta a(b - aw_k) \tag{16.35}$$

The limit on stability is

$$0 < \eta < \frac{2}{a^2} \tag{16.36}$$

Which is stable for positive or negative values of a. This is because a second multiplier changes the sign (i.e. negative feedback) back if it does change in the feedback path by changing it in the forward path to compensate. As with the ordinary feedback case, if a is not a constant the loop is linear but with time-varying overall gain.

Newton's method can be found when $b = 1$ only, otherwise we still need a divide in the solution which defeats the whole purpose. We could proceed with a quasi-Newton type approach, but we can show that we are only guaranteed quadratic convergence when $b = 1$. Therefore we have

$$w_{k+1} = w_k + w_k(1 - aw_k) \tag{16.37}$$

This method has a problem in that the stability is not guaranteed and very much dependent on the initial-conditions. It does have the correct equilibrium points however if it does converge, provided it does not converge to zero. Write (16.37) as

$$w_{k+1} = w_k(1 + e_k) \tag{16.38}$$

where

$$e_k = (1 - aw_k) \tag{16.39}$$

The initial error is written at $k = 0$ as

$$e_0 = (1 - aw_0) \tag{16.40}$$

Let the initial guess of the Newton method be

$$w_0 = \alpha a \tag{16.41}$$

Then

$$e_0 = \left(1 - a^2\alpha\right) \tag{16.42}$$

We require $|e_0| < 1$ and this gives us

$$0 < \alpha < \frac{2}{a^2} \tag{16.43}$$

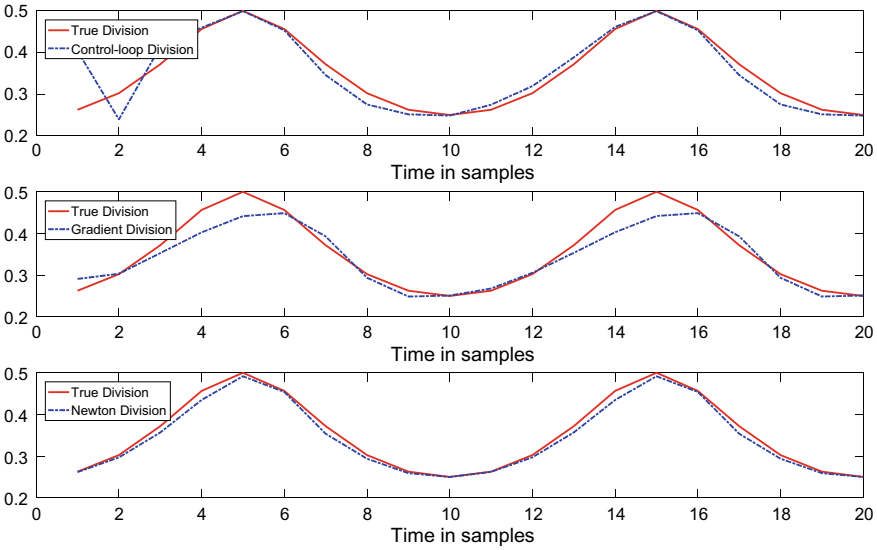


Fig. 16.8 Comparison of Feedback, Gradient and Newton's Methods. Calculating $1/(3 + \cos(2\pi * 0.9 * \mathrm{k})))$

Now the error is given by (16.39). Using (16.38) we can write

$$\begin{aligned} \mathrm{e_k} &= (1 - \mathrm{aw_k}) \\ &= 1 - \mathrm{aw_{k-1}}(1 + \mathrm{e_{k-1}}) \\ &= 1 - \mathrm{aw_{k-1}} - \mathrm{aw_{k-1}e_{k-1}} \\ &= \mathrm{e_{k-1}} - \mathrm{aw_{k-1}e_{k-1}} \\ &= \mathrm{e_{k-1}^2} \end{aligned}$$

This shows that provided the initial condition is chosen such that it satisfies (16.43), that we are guaranteed quadratic convergence. If a is time-varying however we can get away with using its maximum value for the initial guess (Fig. 16.8).

The code is shown below.

MATLAB Code 16.3

```
%Division using ordinary feedback
%Compare with other methods
% number of points in simulation
npoints=20;
%define arrays
wout_o=zeros(npoints,1);
wout_g=zeros(npoints,1);
wout_n=zeros(npoints,1);
%Main Loop
w=0.1;
b=1;
a=4;
%Ordinary linear loop first (time-varying)
% Max gain is 2/a
gain=0.49;
for  i=1:1:npoints
    a=(3+cos(2*pi*0.9*i));
 e=b-a*w;
w=w+gain*e;
wout_o(i)=w;
end
%Now gradient descent method
% Max gain is 2/a^2
gaing=0.1;
%Initial condition
w=0.2;
for  i=1:1:npoints
  a=(3+cos(2*pi*0.9*i));
  e=b-a*w;
w=w+gaing*a*e;
wout_g(i)=w;
end
%Now Newton method
% Note - only guaranteed convergence when b=1.
%initial Guess for Newton Method is 0.98*( (1/a^2) ) x a
alpha=1/(a^2);
```

```
w0=a*alpha*0.98;
%0.98 makes it just less than the max allowed value for safety.
w=w0;
for  i=1:1:npoints
   a=(3+cos(2*pi*0.9*i));
    e=b-a*w;
w=w+w*e;
wout_n(i)=w;
end
%For true waveform we calculated directly
t=[1:1:npoints];
wtrue=1./(3+cos(2*pi*0.9*t));
subplot(3,1,1)
%Now plot true and estimated division - ordinary
plot(t,wtrue,'-r',t,wout_o,'-.b')
legend('True Division','Control-loop Division','Location','northwest')
xlabel({'Time in samples'})
subplot(3,1,2)
%Now plot true and estimated division - gradient method
plot(t,wtrue,'-r',t,wout_g,'-.b')
legend('True Division','Gradient Division','Location','northwest')
xlabel({'Time in samples'})
subplot(3,1,3)
%Now plot true and estimated division - Newton method
plot(t,wtrue,'-r',t,wout_n,'-.b')
legend('True Division','Newton Division','Location','northwest')
xlabel({'Time in samples'})
```

From (16.37)

$$w_{k+1} = w_k + w_k(1 - aw_k) \tag{16.44}$$

We can find the solution to this difference-equation using *Mathematica*:

$$\text{RSolve}[\{w[n] == w[n-1] + w[n-1] * (1 - w[n-1] * a), w[0] == \text{w0}\}, w[n], n]$$

For initial condition w_0 we get a solution to this nonlinear difference equation of

$$\left\{\left\{w[n] \to -\frac{-1+(1-a\mathrm{w0})^{2^n}}{a}\right\}\right\}$$

This we write using our notation as

$$\mathrm{w_k} = \frac{1-(1-\mathrm{aw_0})^{2^k}}{\mathrm{a}} \tag{16.45}$$

and we can see that we require $|1-aw_0|<1$ for convergence (stability) to occur. If $w_0 = \alpha a$ then we have $\left|1-\alpha a^2\right|<1$ or $0<\alpha<\frac{2}{a^2}$. We can of course add a non-unity gain such as $w_{k+1} = w_k + \eta w_k(1-aw_k)$, but quadratic convergence is no longer guaranteed. We also note from (16.45) that if $w_0 = 0$, then $w_k \to 0$. We see that (16.45) has a power of 2^k when decaying which is is an indication of quadratic convergence.

16.4 Vector Based Loops

The previous examples are all scalar systems with one input and one output (SISO). We can extend the principles quiet easily to multivariable systems by using vectors as inputs and outputs. One such example is the problem of spectral factorization. Recall from a previous chapter that for a discrete polynomial $a(z^{-1}) = a_0 + a_1 z^{-1} + \ldots + a_n z^{-n}$ that has all its roots *inside* the unit-circle in the z-plane, that we can write for positive q

$$\mathrm{a}\left(\mathrm{z}^{-1}\right)\mathrm{a}(\mathrm{z}) + \mathrm{q} = \mathrm{e}\left(\mathrm{z}^{-1}\right)\mathrm{e}(\mathrm{z}) \tag{16.46}$$

where $a(z)$ has all its roots *outside* of the unit-circle. We can also think of any polynomial that has all of its roots outside as the noncausal twin of the one with the poles inside the unit-circle. Hence $a(z)$ has a region of convergence inside the unit-circle and $a(z^{-1})$ has its region of convergence outside the unit-circle. The same applies to the polynomial e also of degree n, which is known as the polynomial spectral-factor. Usually in such problems the q constant in (16.46) represents the variance of a noise term that appears in the theory of optimal-filtering. Other nomenclature is often used for brevity such as:

$$\mathrm{aa}^* + \mathrm{q} = \mathrm{ee}^* \tag{16.47}$$

and without losing generality we can consider problems of the form

$$\mathrm{ara}^* + \mathrm{cqc}^* = \mathrm{ee}^* \tag{16.48}$$

where the superscript asterisk * represents the conjugate polynomial. We can also write (16.48) in the form

$$\mathrm{r}|\mathrm{a}|^2 + \mathrm{q}|\mathrm{c}|^2 = |\mathrm{e}|^2 \tag{16.49}$$

The squared term on the RHS of (16.49) and previous equations of this type will be a polynomial which has both negative and positive powers of z. This is called a Laurent polynomial. For example, for $n = 2$ and $a(z^{-1}) = 1 - z^{-1} + 0.5z^{-2} = a$, then we find its Laurent polynomial as

$$\mathrm{L}(\mathrm{z}) = \mathrm{aa}^* = |\mathrm{a}|^2 = 0.5\mathrm{z}^2 - 1.5\mathrm{z} + 2.25 - 1.5\mathrm{z}^{-1} + 0.5\mathrm{z}^{-2} \tag{16.50}$$

The spectral factorization problem is as follows.

Given a Laurent polynomial $L(z)$, can we find the polynomial which generated it? This is working backwards from (16.50) and a bit like finding the square-root. The Laurent polynomial is always symmetric, so we need only consider half of it in any algorithm. We will use feedback to accomplish the task. For convenience here we use the discrete-time notation with brackets denoting the integer time-index k. Vectors or matrices are usually always defined in bold notation. Define an error vector:

$$\mathbf{E}(\mathrm{k}) = \mathbf{L}(\mathrm{k}) - \mathbf{W}(\mathrm{k}) * \mathbf{W}(-\mathrm{k}) \tag{16.51}$$

Here we have defined the elements of a Laurent vector $\mathbf{L}(\mathrm{k})$ as the elements of the Laurent polynomial (half of it only). The RHS of (16.51) is the time-domain equivalent of multiplying a polynomial by its conjugate. This is the same as convolving a causal signal by its mirror-image noncausal counterpart (i.e. time-reversed version of itself). The convolution can be carried out either in the time-domain by simply reversing the coefficients of the polynomial and convolving with the original, or in the frequency-domain using FFTs. This latter approach is preferred for higher-order polynomials. Multiplication of two polynomials is known to be equivalent to convolution, so multiplication of a causal by a noncausal polynomial is the same as their convolution. Having defined the error, all we need is a vector of integrators in the forward path of the loop.

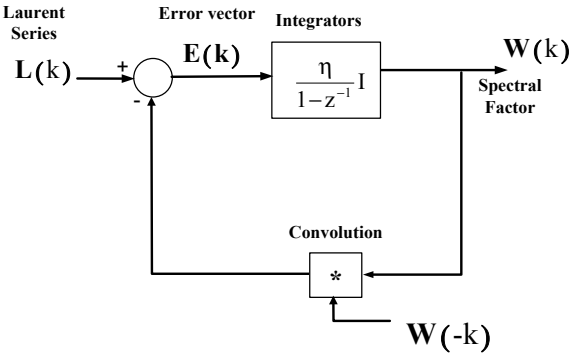


Fig. 16.9 Block-diagram of spectral factorization control-system

$$\mathbf{W}(k+1) = \mathbf{W}(k+1) + \eta \mathbf{E}(k) \tag{16.52}$$

In the above equation, η is a scalar stepsize or gain term which is to be determined. Figure 16.9 shows a block-diagram of the complete control-system.

This is a closed-loop system with a type of squared polynomial in the feedback-path. Provided the gain is large enough, the loop gives the square-root of the square, or in this case the spectral-factor, by forcing the error-vector to zero. To determine the limits on the gain value we need to linearize the system. First write (16.52) as

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \eta[\mathbf{L}(k) - \mathbf{W}(k) * \mathbf{W}(-k)] \tag{16.53}$$

To make better sense we replace the convolution term with a vector of dimension $n + 1$. (for an nth order polynomial there are $n + 1$ coefficients including the zeroth one). Let

$$\mathbf{V}(k) = [\, v_0 \quad v_1 \quad . \quad . \quad v_n \,]^T \tag{16.54}$$

and

$$\mathbf{W}(k) = [\, w_0 \quad w_1 \quad . \quad . \quad w_n \,]^T \tag{16.55}$$

This last vector will be the coefficients of the spectral-factor polynomial as it converges to the true values.

Then write (16.53) as

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \eta[\mathbf{L}(k) - \mathbf{V}(k)] \tag{16.56}$$

where the elements of our $\boldsymbol{V}$ vector are given by the convolution

$$v_i = \sum_{j=0}^{n-i} w_j w_{i+j}, i = 0, 1 \ldots n \tag{16.57}$$

Now write (16.56) in nonlinear notation

$$\mathbf{W}(k+1) = \mathbf{f}(\mathbf{W}(k), \mathbf{L}(k), \mathbf{V}(k))$$

The equilibrium points are by *design* found from when $\boldsymbol{L}(k) - \boldsymbol{V}(k) = 0$. Because the spectral factor is unique, the equilibrium polynomial must be the true spectral-factor if this error is to be minimized using feedback.

To linearize (16.56) we must differentiate this equation wrt the vector $\boldsymbol{W}(k)$ the expression

$$\mathbf{f}(\mathbf{W}(k), \mathbf{L}(k), \mathbf{V}(k)) = \mathbf{W}(k) + \eta[\mathbf{L}(k) - \mathbf{V}(k)] \tag{16.58}$$

We get

$$\mathbf{F} = \left. \frac{\partial \mathbf{f}(\mathbf{W}(k), \mathbf{L}(k), \mathbf{V}(k))}{\partial \mathbf{W}(k)} \right|_{\mathbf{w}=\mathbf{w}^*} \tag{16.59}$$

Now for the RHS of (16.58)

$$\frac{\partial \mathbf{W}(k)}{\partial \mathbf{W}(k)} = \mathbf{I}_{n+1} \tag{16.60}$$

$$\frac{\partial \eta[\mathbf{L}(k) - \mathbf{V}(k)]}{\partial \mathbf{W}(k)} = -\eta \mathbf{J} \tag{16.61}$$

In the above, only $\boldsymbol{V}(k)$ has elements that are dependent on the elements of $\boldsymbol{W}(k)$, hence $\frac{\partial \eta \boldsymbol{L}(k)}{\partial \boldsymbol{W}(k)} = 0$.

The Jacobian $\boldsymbol{J}$ is found from

$$\mathbf{J} = \begin{bmatrix} \frac{\partial V_0}{\partial W_0} & \cdot & \cdot & \cdot & \frac{\partial V_0}{\partial W_n} \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ \frac{\partial V_n}{\partial W_0} & \cdot & \cdot & \cdot & \frac{\partial V_n}{\partial W_n} \end{bmatrix} \tag{16.62}$$

The Jacobian matrix is found by straight forward differentiation of $v_i = \sum_{j=0}^{n-i} w_j w_{i+j}, i = 0, 1 \ldots n$ To differentiate $v_i = \sum_{j=0}^{n-i} w_j w_{i+j}, i = 0, 1 \ldots n$ we use the

product rule and the Kronecker delta to remove any w terms which are out of bounds. Thus for any combination i, k = 0, 1 … n we have

$$\frac{dv_i}{dw_k} = \sum_{j=0}^{n-i} w_{j+i}\delta_{j,k} + w_j\delta_{j+i,k} \tag{16.63}$$

It is straight forward to find any term in the Jacobian from the above differential by putting j = k in the kronecker delta and hence removing the summation. This leaves four possibilities for any positive integers i k from 0, 1, 2 ... n. For simple notation, the values of w in what follows are assumed to be equilibrium values.

$$\frac{\partial v_i}{\partial w_k} = \begin{cases} w_{k+i} + w_{k-i} & k+i \le n \quad k-i \ge 0 \\ w_{k+i} & k < i \\ w_{k-i} & k+i > n \\ 0 & k+i > n \quad k-i < 0 \end{cases} \tag{16.64}$$

For example when n = 2 the Jacobian becomes

$$\mathbf{J} = \begin{bmatrix} 2w_0 & 2w_1 & 2w_2 \\ w_1 & w_0 + w_2 & w_1 \\ w_2 & 0 & w_0 \end{bmatrix} \tag{16.65}$$

$$\mathbf{F} = \left.\frac{\partial \mathbf{f}(\mathbf{W}(k), \mathbf{L}(k), \mathbf{V}(k))}{\partial \mathbf{W}(k)}\right|_{\mathbf{w}=\mathbf{w}^*} = \mathbf{I}_{n+1} - \eta\mathbf{J} \tag{16.66}$$

The eigenvalues of $\boldsymbol{F}$ must all lie within the unit circle. However, since the Jacobian has values which are determined by the solution, we do not know these values from the outset. Therefore it is only possible to find bounds on the stepsize for maximum coefficient values, or an algebraic expression. The algebra quickly becomes unmanageable beyond only a few unknowns. Fortunately, the loop will converge with any stepsize provided (16.66) has stable eigenvalues. We can therefore use an arbitrary value of stepsize instead. Too small and the convergence will be slow, too large and we get instability. A little experimentation is therefore needed. We usually use a positive stepsize. If the stepsize is negative the loop will give a spectral factor which is the negative of the true values.

Consider an example when $a(z^{-1}) = 1 - z^{-1} + 0.5z^{-2}, q = 1$ in (16.47). We can write a MATLAB program to find the spectral factor. We use the convolution command to perform the convolution in the feedback-path using $\eta = 0.4$.

MATLAB Code 16.4

```
%Spectral factorization using feedback
%Define a polynomial a and find the Laurent series
%aa*+q = dr'd* where d is the spectral factor (normalised)
%r' is the innovations variance. We can extend this to two polynomials or
%more eg aa*+cqc*=dr'd*
% the star represents conjugate polynomials in z
% make this any length you like but adjust gain accordingly.
a=[1 -1 0.5];
n=length(a);
% add a q value - say r=q. represents additive variance of noise;
q=1;
%find aa* using convolution of the polynomial with its uncausal counterpart
L=conv(a,fliplr(a))';
m=length(L);
%Truncate so that only one side is used of the Laurent series. Start from
%an and go to a1 - in reverse order for convenience.
L=L(1:n)';
%and add the additive noise variance term nth term since coefficients are
%in reverse order

L(n)=L(n)+q;
%L is the Laurent series we feed into the loop - the setpoint and the
%output is the spectral factor
 %define gain of loop (stepsize)
 gain=0.4;
 %Initialise integrator output vector
 w=zeros(n,1)';
 g=zeros(n,1)';
 %initialise error vector
 e=zeros(n,1)';
 %Main while  loop
 go=true;
 k=0;
 while   go
     %increment counter
    k=k+1;
    %Update integrators
```

```
w=w+gain*e;
%error first
g=conv(w,fliplr(w));
%truncate in half
F=g(1:n);
%error
e=L-F;
%check norm of error
delt=norm(e);
if delt<1e-6 go=false
end

end

%Reverse coefficients
w=fliplr(w);
%remove the first term
co=w(1);
w=w/co;
%ri is innovations variance
ri=co^2
w
%No of iterations to converge
k
```

We use an infinite while-loop which stops when we get convergence. This is checked by ensuring the Euclidian norm of the error in the loop satisfies $\|\mathrm{E}(k)\| < \delta$ for some small δ which we take here to be 1e−6.

The spectral factor converges in 28 steps to $d(z^{-1}) = 1 - 0.4948z^{-1} + 0.1975z^{-2}$. The zeroth term in the spectral factor has divided the whole polynomial to give unity as the first coefficient. This gives us an extra constant term $\mathrm{aa}^* + \mathrm{q} = \mathrm{ee}^* = \mathrm{rdd}^*$ with constant $\mathrm{r} = 2.5314$. This normalization procedure is usual when designing optimal filters. The constant term is usually an innovations variance term. In the MATLAB Code, the spectral factor coefficients are found in reverse order, so we have to flip the result to get the polynomial coefficients the right way around.

16.5 Matrix Inversion Using Negative Feedback

Fast algorithms have been used since the early days of computing in order to invert matrices. For example, the method of inverting a Matrix using Newton's quadratic convergent method was discovered by Ben-Israel [58] for generalized inverses and even earlier by Schultz [59]. For a square-matrix $\boldsymbol{A}$ the iterative solution is found from

$$\mathbf{W}_{k+1} = \mathbf{W}_k(2\mathbf{I} - \mathbf{A}\mathbf{W}_k) \tag{16.67}$$

It requires the right initial-conditions to converge. If we choose $\boldsymbol{W}_0 = \alpha \boldsymbol{A}^T$ where $0 < \alpha < \frac{2}{\sigma^2_{max}}$. Then $\boldsymbol{W}_k \to \boldsymbol{A}^{-1}$ as $k \to \infty$. Here σ_{max} represents the square-root of the maximum eigenvalue of the matrix product $\boldsymbol{A}\boldsymbol{A}^T$, also known as the maximum *singular-value*. If we do not select the initial-conditions in this way then (16.67) will not converge. What has not been noticed however, is that (16.67) can also be re-written in an alternative form

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mathbf{W}_k(\mathbf{I} - \mathbf{A}\mathbf{W}_k) \tag{16.68}$$

Now we can see that there is an error matrix

$$\mathbf{E}_k = (\mathbf{I} - \mathbf{A}\mathbf{W}_k) \tag{16.69}$$

and a set of matrix integrators. This is a classic time-varying nonlinear-system with feedback. The gain is a time-varying matrix gain of value $\boldsymbol{W}_k$, which is the output fed back to the integrator inputs (Fig. 16.10).

This is a kind of multivariable control-system, but different in that usually vectors are the input and output and not matrices. Consider the following MATLAB code for inverting the 3 × 3 matrix

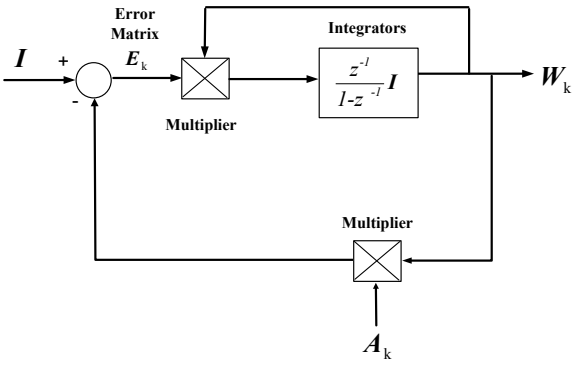


Fig. 16.10 Matrix Inversion using negative-feedback. Newton's method

$$A = \begin{bmatrix} -0.5 & 0 & 0 \\ 2 & -2 & 0 \\ 1 & 2 & -3 \end{bmatrix}$$

Its inverse is found in 11 iterations. Like the scalar case it has quadratic convergence.

MATLAB Code 16.5

```
%Solves inverse of A matrix  using Newton's Method
% works on pos def or neg def matrices ok but needs good starting value as
% below W=alpha A' alpha = 2/max eig(A*A')  ..less than this actually.
% Use trace instead of eigenvalues - simpler in real-time
clear

A=[-0.5 0 0
   2 -2 0
   1 2 -3];

%Z=eig(A*A');
z=trace(A*A');
alpha=0.9*2/max(z)

B=eye(3);

[m,n]=size(A);


I=eye(n);

% Init condition on W
W=alpha*A';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Loop
i=0;
k=1;
while k>0
   i=i+1;
  E=(I-A*W);
   W=W+W*(I-A*W);
```

```
L(n)=L(n)+q;
%L is the Laurent series we feed into the loop - the setpoint and the
%output is the spectral factor
 %define gain of loop (stepsize)
 gain=0.4;
 %Initialise integrator output vector
 w=zeros(n,1)';
 g=zeros(n,1)';
 %initialise error vector
 e=zeros(n,1)';
 %Main while  loop
 go=true;
 k=0;
 while   go
    %increment counter
   k=k+1;
   %Update integrators
```

We use a stopping method similar to the scalar division case. Clearly this is just a generalization of the simple divider case.

Chapter 17
Introduction to Optimal Control

17.1 A Brief Overview

The previous chapters in this book have covered classical control design and state-variable design. In terms of applications it is best to consider classical control as suitable for systems with a single-input and output (SISO). The system dynamic model may be unknown, but nevertheless classical methods such as PID control and to a certain extent lag-lead compensation can be used despite this. The latter of these methods is harder to implement when no model of the system is given and relies on experienced engineering insight. When a feedback system oscillates we use a phase-lead compensator to stabilise it. This also tells us the unity-gain crossover frequency (the frequency of oscillation). The unity-gain crossover frequency (bandwidth) is usually the only frequency of oscillation if the gain is kept within sensible bounds. Of course, higher frequency parasitic oscillations may also be present and it is the job of the experienced design engineer to separate these out. The bandwidth in Hz must make sense from the components of the system. For an electro-mechanical system the bandwidth will not be in MHz but usually Hz and rarely kHz. For electronic systems such as a PLL we can have bandwidths well into the MHz range. Low frequency compensators such as phase-lag or PI can then be added relative to the bandwidth (say a decade down or more in frequency). This is "seat of the pants" engineering and requires some educated guesses and skilful design.

Likewise for PID control a mathematical model is not a requirement and tuning the controller is the main design issue. Due to the simplicity of this approach, by far the most common controllers in industry are PID based. They are often implemented on a programmable-logic system (PLC). The move to state-variables

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7_17

occurred perhaps first in aerospace and defence to control systems with multiple inputs and outputs (multivariable). Such multivariable systems had the added problem of interaction between desired inputs and outputs of the system. Either state-variable and matrix methods had to be used, or polynomial matrix (or matrix transfer-function) approaches. A scalar controller in each loop may well disturb other outputs and the cross-coupling problem needs to be considered. An attempt at classical approaches was tried in the 1970s. For example Rosenbrock's Inverse-Nyquist array method (INA) [60]. A few years-later a move was made from transfer-function matrices to polynomial or polynomial matrix methods by Kucera [61]. Many other authors contributed in this area in attempts to turn optimal control into adaptive optimal control. However, despite extensive work in transfer-function and polynomial based methods the state-variable approach does appear to have survived the test of time. For SISO systems it is hardly worth the effort for most problems to go to state-space, but for multivariable systems the effort has many rewards. The effort required is getting an accurate state-space dynamic model of the system which can be time-consuming as compared with PID. Optimal control involves the minimisation of a quadratic performance index (often referred to as a criterion or cost-function). The approach is a bit like methods of least-squares where minimisation is achieved using fundamental calculus. The reason a quadratic cost is used is because a convex-shaped minimisation is formed where only one solution lies at the optimum. The optimal control problem has both continuous-time and discrete-time solutions. Of course the discrete-time solution is often favoured because we can implement a digital controller.

17.2 Discrete-Time Linear-Quadratic (LQ) Control-Problem or Linear Regulator

Consider a discrete-time noise-free system given by

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k, \mathbf{y}_k = \mathbf{H}\mathrm{x}_k \tag{17.1}$$

The open-loop system need not be open-loop stable, but the state-vector must be completely controllable. Since state-feedback is used, the states must either be measureable and/or completely observable. The bold letters indicate that the inputs and outputs of the system are both vectors of appropriate dimension. Here we assume n states, m inputs and p outputs. Non-square multivariable systems are systems with the number of inputs and outputs which are unequal but usually the square-case applies where $m = p$. The scalar cost function is defined as a summation

$$\mathrm{J} = \sum_{i=0}^{\infty} \left[\mathbf{x}_i^{\mathrm{T}}\mathbf{Q}\mathbf{x}_i + \mathbf{u}_i^{\mathrm{T}}\mathbf{R}\mathbf{u}_i\right] \tag{17.2}$$

The summation to infinity denotes the infinite-horizon problem. The weighting matrices on state and control are defined to be $\boldsymbol{Q}$ and $\boldsymbol{R}$ respectively. Here $\boldsymbol{Q}$ is an n-square positive semi-definite matrix and $\boldsymbol{R}$ is m-square positive-definite. If the order of the input is scalar ($m = 1$) then the $\boldsymbol{R}$ matrix is replaced with a scalar weighting. The latter statements are akin to saying that if the problem were scalar, that the weighting on control can be zero but not that of the state. In physical terms the weights denote how much energy we need to expend to restrict motion in say the control-signal versus that of the states. A heavy penalty on the control signal means that the variations will not be as strong as that of a small weighting. This could in some electromechanical systems reduce the wear on actuators. The summation in (17.2) can be made over a finite-time with an extra term added. We prefer the solution to the infinite-time problem for greater simplicity. Authors prior to 1960 had solved this problem but the solution only applied to certain low-order models. Kalman [62] solved the continuous-time version of the problem using a rigourous Hamiltonian-Jacobi approach (also known as a Bellman equation). For this problem we assume that the setpoint or setpoints (vector case) are zero and that the outputs must track the inputs to zero. This is a little restrictive and also results in a solution with no integral action. However, integral action can be added by augmenting the system with an extra state.

The solution to the discrete-time steady-state LQ problem can be similarly found by the same approach as the continuous-time case, namely that the optimal state-feedback

$$\mathbf{u}_{\mathrm{k}}^{\mathrm{o}} = -\mathbf{K}\mathbf{x}_{\mathrm{k}} \tag{17.3}$$

where $\boldsymbol{K}$ is optimal state-feedback matrix of dimension $m \times n$ matrix given by

$$\mathbf{K} = \left(\mathbf{R} + \mathbf{G}^{\mathbf{T}}\mathbf{P}\mathbf{G}\right)^{-1}\mathbf{G}^{\mathrm{T}}\mathbf{P}\mathbf{F} \tag{17.4}$$

and $\boldsymbol{P}$ is the symmetric positive-definite solution of the algebraic Riccati equation

$$\mathbf{P} = \mathbf{Q} + \mathbf{F}^{\mathbf{T}}\mathbf{P}\mathbf{F} - \mathbf{F}^{\mathbf{T}}\mathbf{P}\mathbf{G}\left(\mathbf{R} + \mathbf{G}^{\mathbf{T}}\mathbf{P}\mathbf{G}\right)^{-1}\mathbf{G}^{\mathbf{T}}\mathbf{P}\mathbf{F} \tag{17.5}$$

We can see the similarity with the Kalman filter of Chap. 13. In fact the optimal-control and optimal state-estimation problems are said to be the *duals* of one another. The solution to Eq. (17.5) can be found algebraically for low-order problems and numerically using the Riccati-equation

$$\mathbf{P}_{\mathrm{k}-1} = \mathbf{Q} + \mathbf{F}^{\mathbf{T}}\mathbf{P}_{\mathrm{k}}\mathbf{F} - \mathbf{F}^{\mathbf{T}}\mathbf{P}_{\mathrm{k}}\mathbf{G}\left(\mathbf{R} + \mathbf{G}^{\mathbf{T}}\mathbf{P}_{\mathrm{k}}\mathbf{G}\right)^{-1}\mathbf{G}^{\mathbf{T}}\mathbf{P}_{\mathrm{k}}\mathbf{F} \tag{17.6}$$

This equation will converge to the solution of the algebraic case for a long-enough time-interval. However, for the control case, unlike Kalman filtering, the iteration must start at some time in the future and work *backwards* in time to zero. $k = N, N - 1 \ldots 1$.

Example 17.1 Mass with Force State-Regulator.
Consider a previously used continuous-time system of Example 11.1.1

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \end{bmatrix} f(t)$$

This was shown to have a discrete-time state-variable description according to

$$\mathbf{F} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} \frac{T_s^2}{2} \\ T_s \end{bmatrix} \frac{1}{M}$$

With no **H** matrix being specified since we require to regulate both states. Let us define a sampling frequency of 1 kHz giving a sampling interval of $T_s = 1$ ms and a mass M = 1 kg. The discrete-time state-matrices now become

$$\mathbf{F} = \begin{bmatrix} 1 & 10^{-3} \\ 0 & 1 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0.5 \times 10^{-6} \\ 10^{-3} \end{bmatrix}$$

The MATLAB command to solve the LQ control problem as found by the solution to (17.6) in steady-state is given by `[Kd,P,e] = lqrd(A,B,Q,R,Ts)`. Within this command Kd represents the feedback matrix $\boldsymbol{K}$. We note that the discrete-time matrices are not even needed as MATLAB calculates these matrices based on the specified sampling-interval. We select an initial condition in the state-vector as $\mathbf{x}_0 = [\,1 \quad 1\,]^T$ with $\boldsymbol{Q} = \boldsymbol{I}$ and the scalar $R = 2$ where $\boldsymbol{I}$ is the identity matrix.

MATLAB Code 17.1

```
%MATLAB Code 17.1
%clear
% LQ State-feedback control of open-loop
unstable system
% Define continuous-time system
A=[0 1;0 0];
B=[0 1]';
% define any C vector for the ss() command to work.
Will be the same in discrete-time
C=[1 0];
% Weighting matrices are diagonal.
Q=eye(2);
R=2;
% Sampling interval in seconds
Ts=1e-3;
%Find the discrete-time gain vector Kd
[Kd,P,e] = lqrd(A,B,Q,R,Ts)
%--------------------------------
% Then find discrete-time state-space description
in terms of the continuous-time matrices
%D=0
sys=ss(A,B,C,0);
% Convert to discrete-time  is one line command
sysd=c2d(sys,Ts)
%access the discrete-time matrices from sysd - D'
will be zero
[F,G,H,D]=ssdata(sysd)


% time vector length N
N=20000;
% length of run in seconds
Tmax=N*Ts
t=[1:1:N];
% input for the reference, zero magnitude
r=0;

x0=[1;1] % Initial conditions on discrete
state vector
x1=zeros(N,1);
x2=zeros(N,1);
```

```
% storeage arrays for plotting states
xs1=zeros(N,1);
xs2=zeros(N,1);
us=zeros(N,1);

% Loop through N times
x=x0;
uo=0;
for k = 1:N
% State outputs of system

%Optimal control signal uo
uo=(r-Kd*x);
%Store the control-signal for later plotting
us(k)=uo;
%new state
x=F*x+G*uo;
% Store states for plotting later
xs1(k)=x(1);
xs2(k)=x(2);
end

%Now plot the states
figure('color','white')
plot(t,xs1,'r',t,xs2,'-.b')
legend('State 1','State 2 ','Location','northeast')
title('State 1 and 2  outputs')
xlabel({'Time in samples'})

% Plot optimal control signal
figure('color','white')
plot(t,us,'r')
title('Optimal control signal')
xlabel({'Time in samples'})
```

Such a system could be some dynamic platform where the angle needs to return to zero or be maintained at zero after a disturbance affects the states to throw-them off position.

Here both states start at unity and find their way to zero. The MATLAB response for the states is shown in Fig. 17.1 and the optimal control signal in Fig. 17.2.

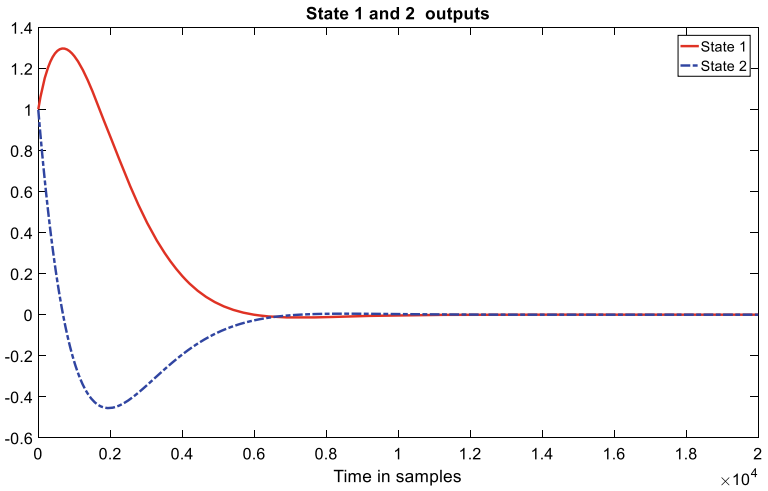


Fig. 17.1 States as controlled by LQ controller

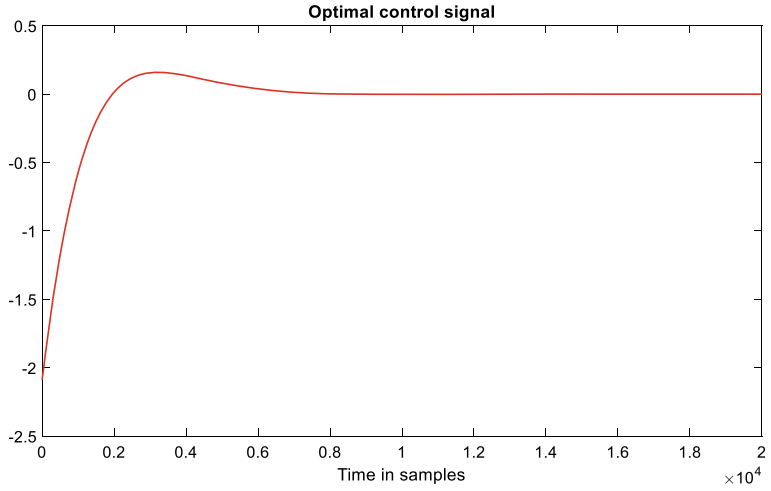


Fig. 17.2 Optimal control signal

17.3 Optimal Output Tracking Problem

We can modify (17.2) to weight the output of the system rather than the states. This is done by a simple substitution $\boldsymbol{Q} = \boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}$ and substituting back into (17.2) giving in the case of SISO systems

$$\mathrm{J} = \sum_{\mathrm{i}=0}^{\infty} \left[\|\mathbf{y}_\mathrm{i}\|^2 + \rho \|\mathbf{u}_\mathrm{i}\|^2 \right] \tag{17.7}$$

We can still have weighting on the output but we have normalised it to unity making the scalar control weighting ρ the only weighting in the criterion (17.7). For $\rho = 0$ we have a don't - care situation about fluctuations in the control signal but the output is fast. When ρ is large we have small variations in output and the control is restricted or damped. We cannot penalise both heavily or have no penalty on either.

Example 17.2 Mass with Force Output Regulator

We consider the same example as 17.1 with the only change that $\boldsymbol{Q} = \boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}$ and we use two control-weightings, $\rho = 0.1$ and $\rho = 10$. The MATLAB code is not shown because it only requires a change in $\boldsymbol{Q}$ and the plotting of the system output instead of the states (Figs. 17.3 and 17.4).

We can see that with a heavier weighting on the control signal the control-signal and output slow down considerably. The time to reach steady-state is about four times slower when $\rho = 10$. Also at high weighting the variation or swing of the control is much reduced.

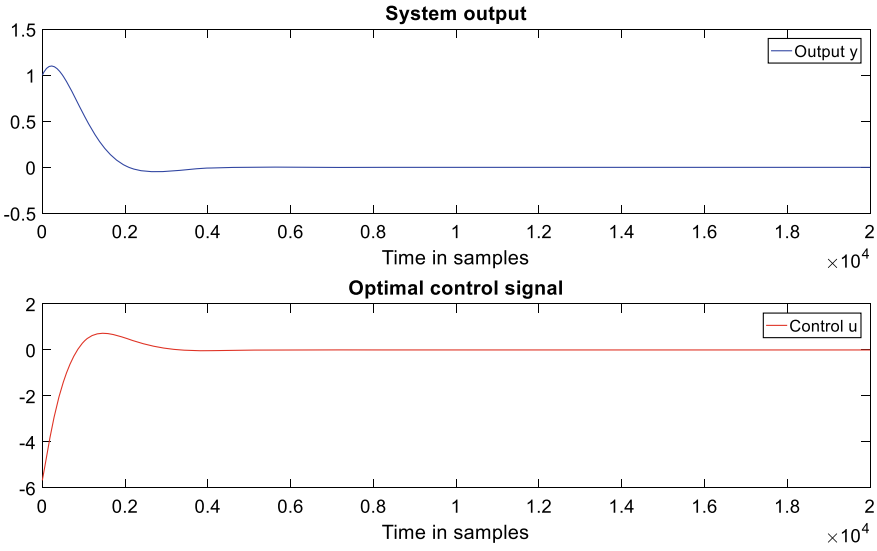


Fig. 17.3 System output and optimal control signal for $\rho = 0.1$

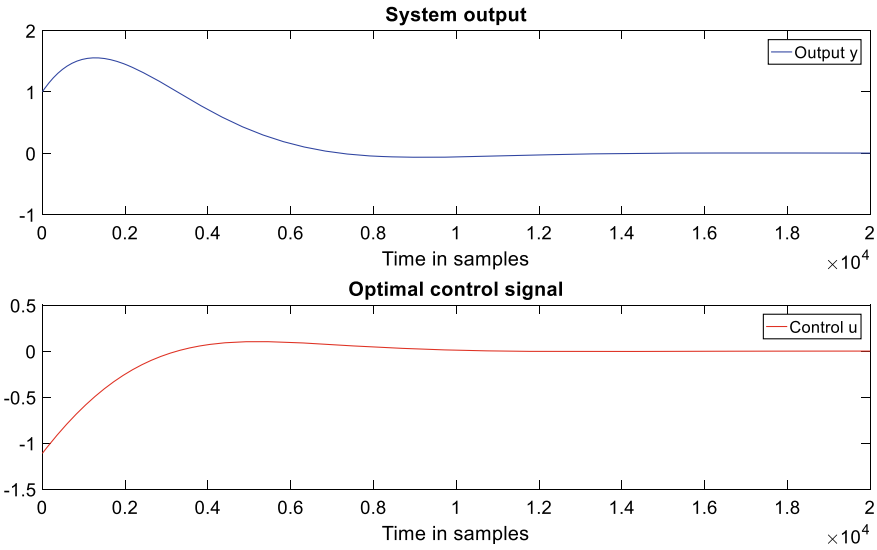


Fig. 17.4 System output and optimal control signal for $\rho = 10$

17.4 Optimal Tracking

The problem with the basic LQ regulator is that it has no integral action in the loop. Although scaling to the input can be used to get a particular steady-state output, this is really not a solution because without at least one integrator there cannot be good disturbance rejection. Therefore we must introduce an integrator into the optimal problem itself. This we can achieve in a similar manner as we already did in Chap. 11, Example 11.5. Consider only the SISO case. We augment the system by adding an extra state x_k^I with corresponding feedback gain k_I. Thus the open-loop discrete-time system becomes

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ x_{k+1}^I \end{bmatrix} = \begin{bmatrix} \mathbf{F} & \mathbf{0} \\ -\mathbf{H} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ x_k^I \end{bmatrix} + \begin{bmatrix} \mathbf{G} \\ 0 \end{bmatrix} u_k \tag{17.8}$$

$$y_k = [\mathbf{H} \quad 0] \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_k^I \end{bmatrix} \tag{17.9}$$

where r_k is a possibly time-varying setpoint and no-longer held constant unlike the regulator case. The bold case zero $\mathbf{0}$ represents a matrix or vector of zeros of appropriate dimension. We assume a control-law of the form

$$u_k = r_k - \begin{bmatrix} \mathbf{K} & k_I \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ x_k^I \end{bmatrix} \tag{17.10}$$

This control law is better shown on the block-diagram shown in Fig. 17.5 below. Essentially it is the same as ordinary state-feedback with an extra state added exactly as was performed for the pole-placement case in Chap. 11.

The output should now track the non-zero setpoint because the integrator forces the error to zero. Substitute (17.9) into (17.8) and we get the closed-loop optimal system

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ x_{k+1}^I \end{bmatrix} = \begin{bmatrix} \mathbf{F} - \mathbf{GK} & -\mathbf{G}k_I \\ -\mathbf{H} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ x_k^I \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} r_k \tag{17.11}$$

$$y_k = \begin{bmatrix} \mathbf{H} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ x_k^I \end{bmatrix} \tag{17.12}$$

We can now use our new augmented open-loop matrices (17.8), (17.9) to find the optimal state-feedback gains. This must be a completely controllable augmented system. If we define $\mathbf{F}_a = \begin{bmatrix} \mathbf{F} & 0 \\ -\mathbf{H} & 1 \end{bmatrix}, \mathbf{G}_a = \begin{bmatrix} \mathbf{G} \\ 0 \end{bmatrix}$

Where $\boldsymbol{F}_a$ is a square matrix of dimension $n + 1$ and $\boldsymbol{G}_a$ is a column vector of length $n + 1$.

Then

$$\mathcal{C} = \text{Rank}\begin{bmatrix} \mathbf{G}_a & \mathbf{F}_a\mathbf{G}_a & \mathbf{F}_a^2\mathbf{G}_a & . & . & \mathbf{F}_a^n\mathbf{G}_a \end{bmatrix}$$

Must be full rank $n + 1$ where n here denotes the original number of states.

We can simulate the closed-loop system using (17.11) and (17.12).

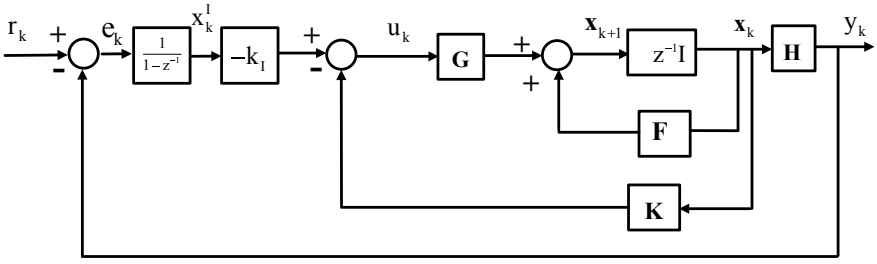


Fig. 17.5 The optimal discrete-time tracking problem

Example 17.3. LQ Tracking Control

A system has z-transfer function $\mathrm{E}(\mathrm{z}) = \frac{\mathrm{Z}-2}{\mathrm{Z}^2-2\mathrm{Z}+2}$. A quick check indicates this system is open-loop unstable with poles outside the unit-circle at $\mathrm{z} = 1 \pm \mathrm{j}$. A zero is outside the unit-circle making the system also nonminimum phase. Forming a controllable-canonical form state-space realisation:

$$\mathbf{F} = \begin{bmatrix} 0 & 1 \\ -2 & 2 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{H} = \begin{bmatrix} 1 & -2 \end{bmatrix}$$

Augmenting

$$\begin{bmatrix} \mathbf{F} & 0 \\ -\mathbf{H} & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -2 & 2 & 0 \\ -1 & 2 & 1 \end{bmatrix}, \begin{bmatrix} \mathbf{G} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Check for controllability. $\mathcal{C} = \mathrm{Rank}\begin{bmatrix} \mathbf{G}_\mathrm{a} & \mathbf{F}_\mathrm{a}\mathbf{G}_\mathrm{a} & \mathbf{F}_\mathrm{a}^2\mathbf{G}_\mathrm{a} \end{bmatrix} = \mathrm{rank}\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 2 \\ 0 & 2 & 5 \end{bmatrix} = 3.$

Therefore the augmented system is fully controllable.

We find by using `[Kda,P,e] = dlqr(Fa,Ga,Q,Rho)` (see MATLAB code below) that the optimal augmented gain vector is

$$\mathbf{K}_\mathrm{a} = \begin{bmatrix} \mathbf{K} & \mathrm{k_i} \end{bmatrix} = \begin{bmatrix} -1.91 & 1.51 & -0.16 \end{bmatrix}$$

The dimension is increased from 2 to 3 since the system is of order 2 and one more state has been added for the integrator. A MATLAB simulation follows with a setpoint square wave of amplitude 1. The output must track this input.

MATLAB Code 17.2

```
%MATLAB Code 17.2
%optimal tracking problem in discrete-time
clear

% Define discrete-time system

F=[0 1
    -2 2];
G=[0 1]';
H=[1 -2];


% Augment discrete-time system
% This is necessary to find optimal feedback gains
%Creat null vector 2x1
vnull=[0 0]';

Fa=[F vnull;-H 1];
Ga=[G' 0]';
Ha=[H 0];
%Form controllability matrix
Co=[Ga Fa*Ga Fa*Fa*Ga]
%check rank
rank(Co)
% Weighting matrix Q
Q=eye(3);

%Scalar weighting on control
Rho=5;

%Find the discrete-time gain vector Kda for 3 states
[Kda,P,e] = dlqr(Fa,Ga,Q,Rho)
%---------------------------------
ega=abs(e)%just check the closed-loop system is stable
%Note - Kda is dimension 3
% Split the gain vector
ki=Kda(3);
K=[Kda(1) Kda(2)];

% time vector length N
N=1000;
%time vector
t=[1:1:N];
%Closed-loop Augmented system
Fc=Fa-Ga*Kda;
Gc=[0 0 1]';
```

```
%storeage for control signal
    us=zeros(N,1);
  %Storeage for system output
   ys=zeros(N,1);
   %Storeage for states 1 to 3
   xs1=zeros(N,1);
   xs2=zeros(N,1);
   xs3=zeros(N,1);
```

For $\rho = 5, \boldsymbol{Q} = \boldsymbol{I}$ we obtain the following closed-loop state-response and output response to a unit-step input (Figs. 17.6 and 17.7).

The system output rises up to the magnitude of unity and follows the setpoint and so does the first two states. The final state which is the integrator state rises up to around 5.5 in the positive direction. Note the well damped response. No overshoot at all results using this method with these ratio of weightings on the cost function. However, if the control weighting is lowered to $\rho = 0.2$ the beginnings of instability is now seen on the system output. This is illustrated in Fig. 17.8.

The open-loop unstable and nonminimum phase system chosen is particularly difficult to control, especially with zero control-weighting. As the weighting on control approaches zero the system goes unstable. Such a system cannot be controlled by minimisation of just the output or states. A penalty on the control signal must also exist.

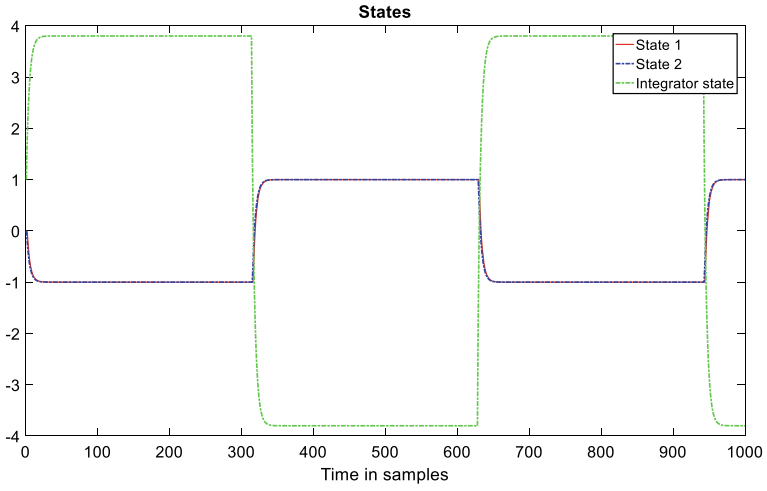


Fig. 17.6 State response

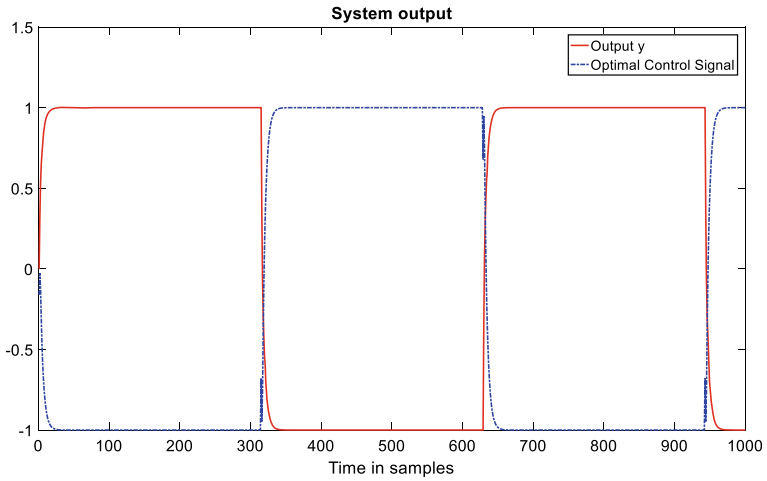


Fig. 17.7 Output of system and optimal control signal $\rho = 5, \boldsymbol{Q} = \boldsymbol{I}$

17.5 Discrete-Time Stochastic Optimal Control

We have already discussed additive noise in dynamical systems (Chap. 12). We can write a SISO discrete-time noisy system as

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}u_k + \mathbf{E}\xi_k \tag{17.13}$$

$$y_k = \mathbf{H}\mathbf{x}_k \tag{17.14}$$

$$s_k = y_k + v_k$$

The noise terms v_k, ξ_k are zero-mean with variances σ_v^2 and σ_ξ^2 respectively. We require the system to be completely controllable and observable. By taking z-transforms we can also express the above-state-space equations in transfer-function form

$$s_k = W(z)u_k + W_o(z)\boldsymbol{\xi}_k + v_k \tag{17.15}$$

A block-diagram of the stochastic system is better viewed in transfer-function form (Fig. 17.9).

We define the scalar LQG cost function for the more general MIMO problem as

$$J = E\left[\mathbf{x}_i^T\mathbf{Q}\mathbf{x}_i + \mathbf{u}_i^T\mathbf{R}\mathbf{u}_i\right] \tag{17.16}$$

where E[.] is the expectation operator. The reason we no longer require a summation is that we assume an *ergodic* stochastic process where the *ensemble-average*

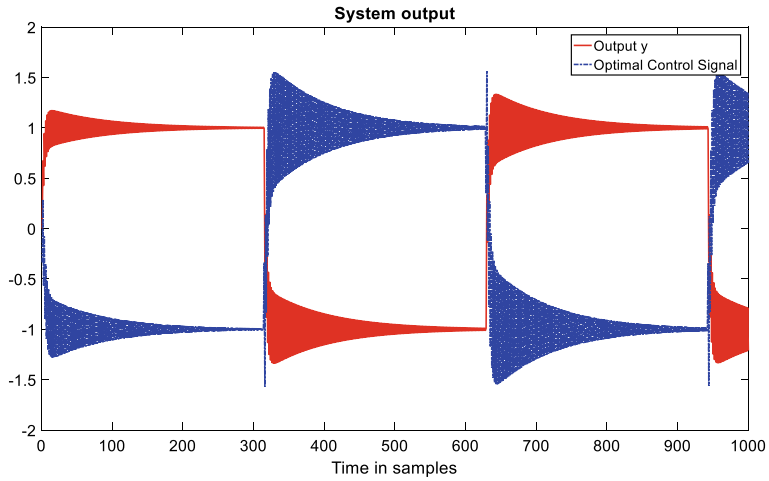


Fig. 17.8 Output of system and optimal control signal $\rho = 0.2, \boldsymbol{Q} = \boldsymbol{I}$

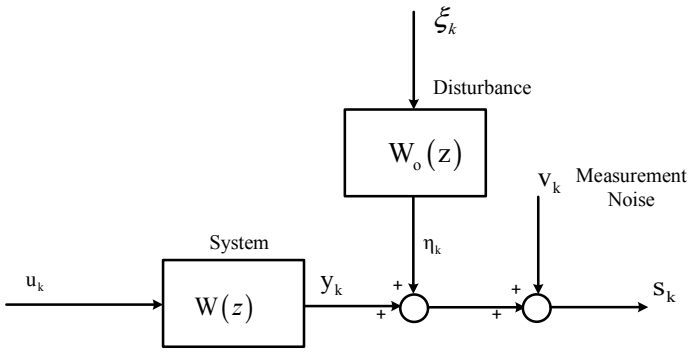


Fig. 17.9 Stochastic system showing random disturbance and additive measurement noise (SISO case)

is the same as the *time-average*. This assumption will be true for stationary noise provided we have enough samples. We can also minimise the output instead of the states as we did before. In the scalar case the cost function reduces to the simplified

$$\mathrm{J} = \mathrm{E}\left[\mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{Q}\mathbf{x}_\mathrm{i} + \rho\,\mathrm{u}_\mathrm{k}^2\right] \tag{17.17}$$

Or for output SISO LQG regulation

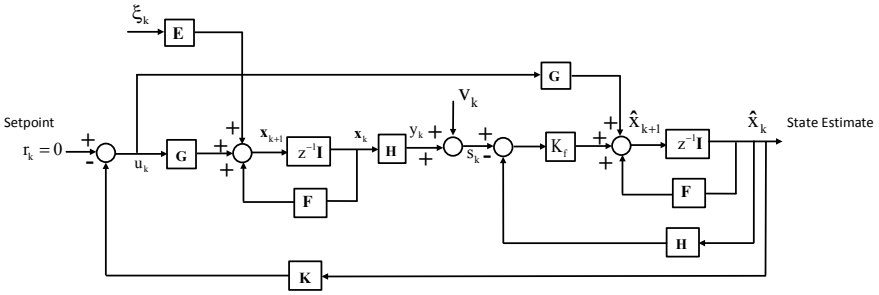


Fig. 17.10 Optimal LQG regulator, showing Kalman filter and state-feedback

$$J = E\left[qy_k^2 + ru_k^2\right] \tag{17.18}$$

In (17.18) we can simplify further by normalising $q = 1$ and writing

$$J = E\left[y_k^2 + \rho u_k^2\right] \tag{17.19}$$

We only have then one design parameter to choose, the control-weighting.

The state-variable solution was found by Kalman [35] to satisfy the *separation principle*. Stated briefly this means that the problem can be separated in two parts. The optimal control is the same as the previously discussed LQ case which has zero noise (the deterministic problem Sect. 17.2). The states are estimated using a discrete-time Kalman filter as previously discussed in Sect. 13.2. Therefore we already have all the tools to solve this problem. Solve the deterministic problem and solve for the Kalman filter. The solution is the combination of the two as shown in Fig. 17.10.

17.6 Polynomial or Transfer-Function Methods in LQG Control

As discussed in Sect. 12.6, for the SISO problem we can turn each transfer-function of (17.15) into a ratio of two polynomials. We arrive at an ARMAX stochastic model that becomes

$$s_k = \frac{z^{-d}b(z^{-1})}{a(z^{-1})}u_k + \frac{c(z^{-1})}{a(z^{-1})}\xi_k + v_k \tag{17.20}$$

where the polynomials are defined as

$$c\left(z^{-1}\right) = c_1 z^{-1} + c_2 z^{-2} + \ldots + c_n z^{-n} \tag{17.21}$$

$$a\left(z^{-1}\right) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n} \tag{17.22}$$

$$b\left(z^{-1}\right) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-m} \tag{17.23}$$

Generally all polynomials do not begin at unity for the zeroth coefficient with the exception that $a(0) = 1$. The integer d is a time-delay which for many process-control systems is an important facet of the model. Even in the simplest of models the delay should be at least one-step. Usually the degree of *b* is less than or equal to the degree of *a,* so that $m < n$. The *b* polynomial can also be nonminimum phase. The *c* polynomial is usually minimum phase for many control-problems but for certain equalisation and filtering problems could also be nonminimum-phase. It is usual to combine the additive noise with the disturbance term and treat them as a combined system. This involves a spectral factorization.

The reason (17.20) is used is because we can use recursive-least squares (RLS) to estimate the polynomials (Sect. 13.4) and then calculate the control law. We can at each iteration estimate the parameters of the polynomials (the unknown polynomial coefficients) from direct measurements of the input and output of the system. We then calculate the optimal controller based on these estimates. The concept of state-space is completely avoided along with the solution of Riccati equations. We would normally need to solve two Riccati equations for the Kalman solution, one for the Kalman filter and one for the controller state-feedback gains. For the polynomial approach we replace Riccati equations with spectral factorizations instead. We have already briefly studied the spectral factorization problem in Chap. 12.

Suppose we simplify the problem to the LQG SISO regulator case and define the cost-function as

$$J = E\left[y_k^2 + \rho u_k^2\right] \tag{17.24}$$

The optimal control is then given by [63]

$$u^0 = -\frac{g(z^{-1})}{h(z^{-1})} s_k \tag{17.25}$$

where the polynomials $g(z^{-1})$ and $h(z^{-1})$ are given by the minimum-degree solution of the Diophantine equation

$$a(z^{-1})h(z^{-1}) + z^{-d}b(z^{-1})g(z^{-1}) = d_c(z^{-1})d_f(z^{-1}) \tag{17.25}$$

The polynomials $h(z^{-1})$ and $g(z^{-1})$ have degree $nh = d + m - 1$ and $ng = n - 1$. For a non-singular solution we require any common factors of $a(z^{-1})$ and $b(z^{-1})$ to have been cancelled. We say that these two polynomials are relative-prime. The equivalence in state-space is that the system is completely controllable and observable. The polynomials $d_c(z^{-1})$ and $d_f(z^{-1})$ are the polynomials that describe the poles of the closed-loop system and that of the filter respectively. In the case of the filter, the stable polynomial $d_f(z^{-1})$ is found from a spectral factorization

$$a(z^{-1})\sigma_v^2 a(z) + c(z^{-1})\sigma_\xi^2 c(z) = d_f(z^{-1})d_f(z) \tag{17.26}$$

Another spectral factor is needed to find the poles of the closed-loop system.

$$a(z^{-1})\rho a(z) + b(z^{-1})b(z) = d_c(z^{-1})d_c(z) \tag{17.27}$$

The spectral factor polynomials will both have order n. It is seen that in (17.27), the pole polynomial $d_c(z^{-1})$ is governed from the control weighting term ρ as well as the system dynamics. On the other hand, the filter polynomial $d_f(z^{-1})$ is governed from the noise variance terms and the model of the disturbance transfer function $c(z^{-1})/a(z^{-1})$. We could skip (17.27) altogether and fix the poles somewhere in the z-plane within the unit-circle. This involves fixing $d_c(z^{-1})$ but the result would not be optimal and only pole-placement. The first spectral factor polynomial $d_f(z^{-1})$ cannot be skipped but can be estimated directly from observations of the input and output of the system using RLS. The Diophantine Eq. (17.25) is more of a problem for higher-order polynomials. There are a number of ways to solve this sort of equation. For low orders we can simply multiply out and solve directly using basic algebra. Alternatively we can solve in matrix form or finally the most elegant way is to use the Euclidean algorithm [61, 64, 65]. The Euclidean (or Euclid's) algorithm is used to find the greatest-common divisor (gcd) of two polynomials. This is the highest order polynomial that is a factor both polynomials. The method uses polynomial long division. In this introduction we will just look at a simple example which does not require any such complex numerical algorithms.

Example 17.4 LQG Polynomial Regulator
Consider the example of Astrom and Wittenmark [66]. This is an open-loop stable but nonminimum phase system.

$$s_k = \frac{z^{-2}(1-2z^{-1})}{1-0.95z^{-1}}u_k + \frac{1-0.7z^{-1}}{1-0.95z^{-1}}\xi_k + v_k \tag{17.28}$$

The white-noise variances are both unity and the control weighting is also defined as unity. The $c(z^{-1})$ polynomial is defined from the zeroth coefficient whereas our polynomial is defined from a delay as the first coefficient. However, adding a one-step delay to the disturbance term makes no difference to its spectrum and so we can continue with no changes. This is because the disturbance white-noise is coloured by $c(z^{-1})/a(z^{-1})$ and unmeasurable. So we have $\sigma_v^2 = 1$, $\sigma_\xi^2 = 1$ and $\rho = 1$. First we find the filter spectral factor polynomial from

$$a(z^{-1})\sigma_v^2 a(z) + c(z^{-1})\sigma_\xi^2 c(z) = d_f(z^{-1})d_f(z)$$

We already studied the spectral-factorization problem and there are many ways to solve for $d_f(z^{-1})$. Using Newton's method [67] we can find

$$d_f(z^{-1}) = 1.445 - 1.1414z^{-1}$$

Calculating the second spectral factor for the poles of the closed-loop system we have

$$a(z^{-1})\rho a(z) + b(z^{-1})b(z) = d_c(z^{-1})d_c(z)$$

and we obtain in a similar fashion

$$d_c(z^{-1}) = 2.289 - 1.288z^{-1}$$

Now we solve the Diophantine equation where the orders $nh = d + m - 1 = 2$ and $ng = n - 1 = 0$.

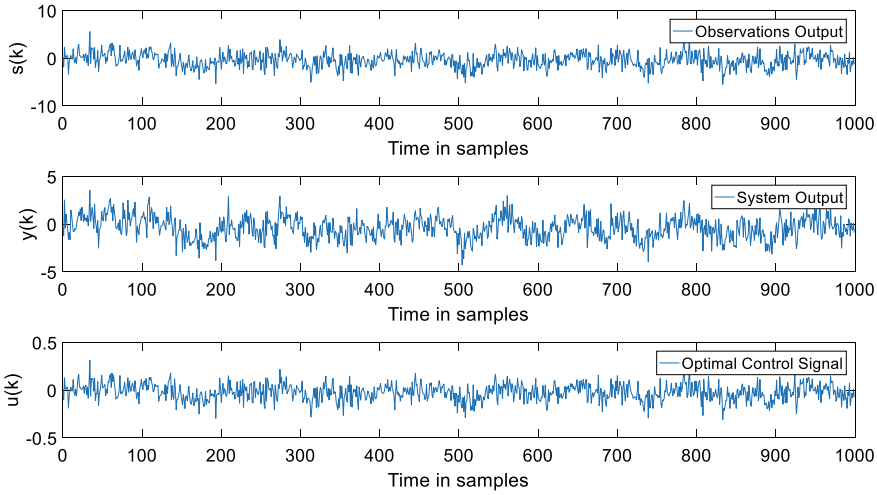


Fig. 17.11 LQG controller. Observations, system output and optimal control signal

$$a(z^{-1})h(z^{-1}) + b(z^{-1})g(z^{-1}) = d_c(z^{-1})d_f(z^{-1})$$
$$(1 + a_1 z^{-1})(h_o + h_1 z^{-1} + h_2 z^{-2}) + z^{-2}(1 + b_1 z^{-1})g_0 = (d_{co} + d_{c1} z^{-1})(d_{fo} + d_{f1} z^{-1})$$

Comparing coefficients of negative powers of z gives us

$$h_o = d_{co} d_{fo} = 3.3067$$
$$h_1 = d_{co} d_{f1} + d_{c1} d_{fo} - h_o a_1 = -1.3316$$
$$h_2 + g_o + h_1 a_1 = d_{c1} d_{f1}$$
$$h_2 a_1 + b_1 g_o = 0$$

The last two of these equations can be solved simultaneously to yield

$$g_o = -\frac{a_1 h_2}{b_1} = -0.1856,\ h_2 = \frac{d_{c1} d_{f1} - a_1 h_1}{1 - (a_1/b_1)} = 0.3907$$

The MATLAB program that generated Fig. 17.11 is shown.

MATLAB Code 17.3

```
%State vector is dimension 2
x0=[0 0]'; % Initial conditions on discrete state vector
%state-vector
x=x0;
%Control signal
u0=0;
%extra state for integrator
xi=0;
y=0;
%setpoint is an array ra
ra=square(0.01*t);

% Main Loop

for k=1:N


%Update states
x=F*x+G*u0;
y=H*x;
%Store first 2 states
xs1(k)=x(1);
xs2(k)=x(2);
%store output
ys(k)=y;
r=ra(k);
 %Update integrator - extra state 3
     xi=xi+(r-y);
   %store third state - integrator state
    xs3(k)=xi;
%Optimal control signal
u0=-K*x-ki*xi;
%Store control signal
us(k)=u0;
end

 %Now plot the output and states
figure('color','white')
plot(t,ys,'r',t,us,'-.b')
legend('Output y','Optimal Control Signal','Location',
'northeast')
title('System output')
xlabel({'Time in samples'})
figure('color','white')
plot(t,xs1,'r',t,xs2,'-.b',t,xs3,'-.g')
legend('State 1','State 2','Integrator state','Location',
'northeast')
title('States')
xlabel({'Time in samples'})
```

```
%Polynomial LQG controller
%Spectral fators computed elsewhere
a1=-0.95;
b1=-2;
b0=1;
c1=-0.7;
c0=1;
% Disturbance Model
c=[1 c1]; a=[1 a1];
sq=1;sv=1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%b polynomial - include time-delay
b=[0 0 1 -2];
% Use N points for simulation
N=1000;
t=[0:1:N-1]';

%Generate Random Noise Variance sq=1 driving c/a
rn=randn(N,1);
%Filter it

%Re-seed wnoise generator
rng(100);
sv=1;
% Additive uncorrelated white-noise variance sv=1
rv=sqrt(sv)*randn(N,1);

%Spectral factor coefficients of polynomials
df0=1.445;
df1=-1.1414;
dc0=2.289;
dc1=-1.288;
%find controller parameters from Diophantine equation
h0=dc0*df0
h1=dc0*df1+dc1*df0-h0*a1
h2=(dc1*df1-a1*h1)/(1-(a1/b1))
g0=-(a1*h2)/b1

%Some arrays
y=zeros(N,1);
u=zeros(N,1);
s=zeros(N,1);
us=zeros(N,1);
ss=zeros(N,1);
ys=zeros(N,1);
```

There is not much to glean from the plots other than to say that the closed-loop system is stable despite the open-loop system being nonminimum phase.

Optimal control is well suited to multivariable systems unlike classical control. The matrix concepts are easily extended to take account of multiple inputs and outputs. The polynomial methods can in turn be extended to the multivariable cases by using polynomial-matrices instead of polynomials. A huge literature exists in this area.

17.7 Conclusions

In this book I have introduced the mathematics and engineering concepts of feedback control to a wide variety of practical applications. The implementation part is of particular importance because it is rarely if ever covered in professional text-books of this kind which tend to stick with simulations. Many of the methods in other textbooks survive, whilst others are given less emphasis due to the change in available design software compared with 35 years ago when I first began teaching. Techniques involving structural resonance, faced in nearly all servo design are rarely if ever covered in text-books, and is covered routinely here. Similarly, operational-amplifier stability is usually covered in analogue electronics textbooks but is given a great deal of attention since it is an essential part of control-theory. Likewise the phase-locked loop is covered and its cousin the amplitude-locked loop appears for the first time in any textbook. I have attempted to gather the topics most relevant to electrical-electronics and mechatronics in the same place. The reader should use this book as a good introduction to the topic. Whilst process-control is not offered any space at all, it should be noted that such systems are controlled using the same methods as we have already covered. The only difference is that many of the systems are far slower in response-time and more prone to some additional problems such as dead-time.

Somewhat of a bonus is the Chap. 16 where I have shown that feedback was in existence in mathematics for centuries without the knowledge of the mathematicians who created many of the algorithms. Topics such as fast methods of inverting a matrix are seen as a type of multivariable system. In fact the idea of a multivariable system is usually thought of as a system with vector inputs and outputs and here we have taken the idea to the next level with a matrix input and output instead.

Kalman filters is a subject that whilst the theory is quiet old, has become more and more relevant for non-military applications than it has been in the past. The topic is covered in some depth herein and with a practical implementation example. Space does not permit the addition of many more topics such as robust control. Be aware however that by far the majority of feedback still uses the old tried and tested methods of PID and lag-lead as are covered frequently in this text. As such, the majority of this book is well suited to undergraduate courses in the field.

Conclusions

In this book I have introduced the mathematics and engineering concepts of feedback control to a wide variety of practical applications. The implementation part is of particular importance because it is rarely if ever covered in professional text-books of this kind which tend to stick with simulations. Many of the methods in other textbooks survive, whilst others are given less emphasis due to the change in available design software compared with 35 years ago when I first began teaching. Techniques involving structural resonance, faced in nearly all servo design are rarely if ever covered in text-books, and is covered routinely here. Similarly, operational-amplifier stability is usually covered in analogue electronics textbooks but is given a great deal of attention since it is an essential part of control-theory. Likewise the phase-locked loop is covered and its cousin the amplitude-locked loop appears for the first time in any textbook. I have attempted to gather the topics most relevant to electrical-electronics and mechatronics in the same place. The reader should use this book as a good introduction to the topic. Whilst process-control is not offered any space at all, it should be noted that such systems are controlled using the same methods as we have already covered. The only difference is that many of the systems are far slower in response-time and more prone to some additional problems such as dead-time.

Somewhat of a bonus is the Chap. 16 where I have shown that feedback was in existence in mathematics for centuries without the knowledge of the mathematicians who created many of the algorithms. Topics such as fast methods of inverting a matrix are seen as a type of multivariable system. In fact the idea of a multivariable system is usually thought of as a system with vector inputs and outputs and here we have taken the idea to the next level with a matrix input and output instead.

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7

Kalman filters is a subject that whilst the theory is quiet old, has become more and more relevant for non-military applications than it has been in the past. The topic is covered in some depth herein and with a practical implementation example. Space does not permit the addition of many more topics such as robust control. Be aware however that by far the majority of feedback still uses the old tried and tested methods of PID and lag-lead as are covered frequently in this text. As such, the majority of this book is well suited to undergraduate courses in the field.

References

1. S. Bennett, *A History of Control Engineering, 1800-1930*. (Institute of Engineering and Technology (IET), London, UK, 1986)
2. J.C. Maxwell, On Governors. Proc R. Soc. Lond. **16**, 270–283 (1867)
3. R.H. Thurston, *A History of the Growth of the Steam-Enginer* (D Appleton and Company (available through project Guttenberg), New York, 1878)
4. E.J. Routh, *Treatise on the Stability of a Given State of Motion*, (MacMillan and Co, (later reprint in Stability of Motion by Taylor and Francis London 1975), 1877)
5. H.S. Black, Stabilized feedback amplifiers. Bell Syst. Tech. J. **13**, 1–18 (1934)
6. H. Nyquist, Regeneration theory. Bell Syst. Tech. J. **11**, 126–147 (1932)
7. H.W. Bode, Relationship between attenuation and phase in feedback amplifiers. Bell Syst. Tech. J. **19**, 421–454, (1940)
8. L.V. Ahlfors, *Complex Analysis: An Introduction to the Theory of Analytic Functions of One Complex Variable*, (McGraw-Hill, 1966)
9. R.H. Goddard, *Rocket Development Liquid-Fuel Rocket Research 1929-1941*, (Prentice Hall, New York, 1960)
10. R.E. Kalman, A new approach to linear filtering and prediction problems. ASME Trans. J. Basic Eng. **82**, 35–45 (1960)
11. B.P. Gibbs, *Advanced Kalman Filtering, Least-Squares and Modeling: A Practical Handbook* (Wiley, New York, 2011)
12. A. Johnson, LQG applications in the process industries. Chem. Eng. Sci. **48**, 2829–2838 (1993)
13. M.J. Grimble, *Industrial Control System Design* (Wiley, New York, 2001)
14. B. Goodwine, *Engineering Differential Equations, Theory and Applications* (Springer, New York, London, 2010)
15. E.T. Whittaker, Works of Laplace. Math. Gaz. **33**, 1–12 (1949)
16. A. Ambardar, *Analog and Digital Signal Processing*, 2nd edn. (Brooks-Cole Publishing Company, USA, 1999)
17. P.J. Nahin, *Oliver Heaviside: The Life, Work and Times of an Electrical Genius of the Victorian Age* (The John Hopkins universoty Press, USA, 2002)
18. K. Ogata, *Modern Control Engineering*, 5th edn. (Pearson, USA, 2009)
19. R. Iserman, M. Munchhof, *Identification of Dynamic Systems, an Introduction with Applications* (Springer, London, New York, 2010)
20. W.R. Evans, Control System Synthesis by Root Locus Method. Trans Am. Inst. Electr. Eng. **69**, 66–69 (1950)

T. Moir, *Feedback*, https://doi.org/10.1007/978-3-030-34839-7

21. A. Hurwitz, On the conditions under which an equation has only roots with negative real parts (Original German version Ueber die Bedingungen, unter welchen eine Gleichung nur Wurzeln mit negativen reellen Theilen besitz), in *Selected Papers on Mathematical Trends in Control Theory (originally published in 1895 in German)*, ed. R. Bellman, R. Kalaba (Dover, New York, 1964), pp. 70–82
22. B. Carter, R. Mancini, *Op-Amps for Everyone* (Newness, MA, USA, 2009)
23. F. Golnaraghi, B. Kuo, *Automatic Control Systems* (John Wiley and Sons, USA, 2009)
24. E.I. Jury, *Sampled-Data Control-Systems*, (John Wiley and Sons, 1958)
25. E.T. Whittaker, On the functions which are represented by the expansions of the interpolation theory. Proc. Roy. Soc. Edinburgh **35**, 181–194 (1915)
26. V.A. Kotelnikov, On the carrying capacity of the "ether" and wire in telecommunications. *Material for the First All-Union Conference on Questions of Communication*, (Izd. Red. Upr. Svyazi RKKA, Moscow, 1933)
27. C.E. Shannon, A mathematical theory of communications. Bell Syst. Tech J. **27**, 379–423 (1948)
28. H. Nyquist, Certain topics in telegraph transmission theory. Trans. AIEE **47**, 617–644 (1928)
29. J.R. Ragazzini, L.A. Zadeh, The analysis of sampled-data systems. Trans. Am. Inst. Elec. Eng. **71**, 225–234 (1952)
30. A.V. Oppenheim, R.W. Schafer, *Digital Signal Processing*, (Prentice Hall International Ltd, 1975)
31. N. Wiener, *Extrapolation, Interpolation and Smoothing of Stationary Time Series, with Engineering Applications*, (Technology press and Wiley, (Originally issued in Feb. 1942 as a classified Nat. Defence Res. Council Rep.), New York, 1949)
32. A.N. Kolmogorov, Stationary sequences in Hilbert Space (English Trans, in ed. T. Kailath, *Linear Least Squares Estimation* (Dowden, Hutchinson & Ross, Pennsylvania 1977), pp. 66–89), *Bolletin Moskovskogo Gosudarstvennogo Universiteta. Matematika* **2**, 1–40 (1941)
33. G.E.P. Box, G.M. Jenkins, *Time series analysis: forecasting and control*, 5th edn. (John Wiley and Sons, 2015)
34. J.F. Barrett, T.J. Moir, A unified approach to multivariable discrete-time filtering based on the Wiener-theory. Kybernetika **23**, 177–197 (1987)
35. R.E. Kalman, A new approach to linear filtering and prediction problems. J. Basic Eng. 35–45 (1960)
36. R.E. Kalman, R.S. Bucy, New results in linear filtering and prediction theory. J. Basic Eng., 95–108 (1961)
37. R.L. Stratonovich, Optimum nonlinear systems which bring about a separation of a signal with constant parameters from noise. Radiofizika **2**, 892–901 (1959)
38. R.L. Plackett, Some theorems in least-squares. Biometrika **37**, 149–157 (1950)
39. L. Guo, Estimating time-varying parameters by the Kalman filter based algorithm: stability and convergence. IEEE Trans. Autom. Control. **35**, 141–147 (1990)
40. B. Widrow, J.R. Glover, J.M. McCool, J. Kaunitz, C.S. Williams, R.H. Hearn et al., Adaptive noise cancelling: principles and applications. Proc. IEEE **63**, 1692–1716 (1975)
41. B.W. Kim, B.S. Park, Robust control for the segway with unknown control coefficient and model uncertainties. *Sensors (Basel, Switzerland)* **16**, 1000 (2016)
42. J.M. Hilkert, Inertially stabilized platform technology Concepts and principles. IEEE Control. Syst. Mag. **28**, 26–46 (2008)
43. S.F. Ahmed, K. Kushsairy, M.I.A. Bakar, D. Hazry, M.K. Joyo, Attitude stabilization of Quad-rotor (UAV) system using Fuzzy PID controller (an experimental test), in *2015 Second International Conference on Computing Technology and Information Management (ICCTIM)* (2015) pp. 99–104
44. R. Jia, V.K. Nandikolla, G. Haggart, C. Volk, D. Tazartes, System performance of an inertially stabilized gimbal platform with friction, resonance, and vibration effects. J. Nonlinear Dyn. **2017**, 20 (2017)

45. N. Ghaeminezhad, W. Daobo, F. Farooq, Stabilizing a gimbal platform using self-tuning fuzzy PID controller. Int. J. Comput. Appl. **93**, 0975–8887 (2014)
46. A. Tayebi, S. McGilvray, Attitude stabilization of a VTOL quadrotor aircraft. IEEE Trans. Control. Syst. Technol. **14**, 562–571 (2006)
47. Z. Koruba, Dynamics and control of a gyroscope-stabilized platform in a ship anti-aircraft rocket missile launcher. Solid State Phenom. **196**, 124–139 (2013)
48. S. Li, M. Zhong, J. Qin, The internal model control design of three-axis inertially stabilized platform for airborne remote sensing, in *2012 8th IEEE International Symposium on Instrumentation and Control Technology (ISICT) Proceedings* (2012), pp. 5–10
49. S. Elaydi, *Introduction to Difference Equations* (Springer, USA, 2000)
50. D.G. Luenberger, *Introduction to Dynamic Systems, Theory, Models and Applications* (John Wiley and Sons, NY, USA, 1979)
51. H.K. Khalil, *Nonlinear Systems* (Prentice-Hall, NJ, USA, 1996)
52. A.M. Pettigrew, Amplitude-locked loop circuits, GB Patent (1991)
53. T.J. Moir, Automatic variance control and variance estimation loops. Circuits, Syst. Signal Process. **20**, 1–10 (2001)
54. T.J. Moir, A.M. Pettigrew, A multiplicative cancellation approach to multipath suppression in FM radio. Wirel. Pers. Commun. **75**, 799–819 (2014)
55. A. Cauchy, Methode generale pour la resolution des systemes d´equations simultanees. C.R. Acad.Sci. Paris **25**, 536–538 (1847)
56. J. Moser, A rapidly convergent iteration method and non-linear differential equations. Ann. Sc. Norm. Sup. Pisa **20**, 266–315 (1966)
57. J.W. Neuberger, The continuous Newton's method, inverse functions, and Nash-Moser. Am. Math. Mon. **114**, 432–437 (2007)
58. A. Ben-Israel, An iterative method for computing the generalized inverse of an arbitrary matrix. Math. Comput. **19**, 452–455 (1965)
59. G. Schulz, Iterative Berechung der reziproken Matrix. Zeitschrift fur Angewandte Mathematik und Mechanik **13**, 57–59 (1933)
60. H.H. Rosenbrock, *Computer-Aided Control System Design*, (Academic Press, 1974)
61. V. Kucera, *Discrete linear control the Polynomial equation approach*, (John Wiley, 1979)
62. R.E. Kalman, Contributions to the theory of optimal control. Bol. Soc. Mat. Mex. **5**, 102–119 (1960)
63. M.J. Grimble, Implicit and explicit LQG self-tuning controllers. Automatica **20**, 661–669 (1984)
64. J. Jezek, New algorithm for minimal solution of linear polynomial equations. Kybernetika **18**, 505–516 (1982)
65. M. Halwass, Self-tuning LQG control. MSR MESS Steuern Regeln **31**, 2–6 (1988)
66. K.J. Astrom, B. Wittenmark, Analysis of a self-tuning regulator for nonminimum phase systems, in *IFAC Symposium on Stochastic Control*, (Budapest Hungary, 1974)
67. T.J. Moir, Stabilising discrete polynomials using spectral factorisation. Electron. Lett. **27**, 581–583 (1991)