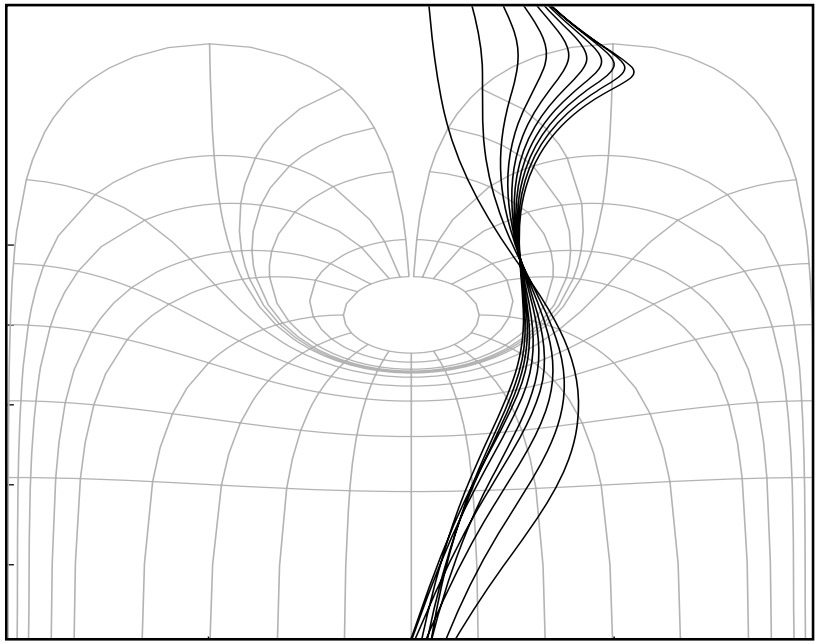
Linear Feedback Control

Analysis and Design with MATLAB



Linear Feedback Control

Analysis and Design with MATLAB

Dingyü Xue
Northeastern University
Shenyang, People's Republic of China

YangQuan Chen
Utah State University
Logan, Utah, USA

Derek P. Atherton
University of Sussex
Brighton, United Kingdom

siam

Contents

Preface

It is well known that the benefits from the wise use of control engineering are numerous and include improved product/life quality, minimized waste materials, reduced pollution, increased safety, reduced energy consumption etc. One can observe that the notions of feedback and control play important roles in most sociotechnological aspects. The phrase "control will be the physics of the 21st century"[1] implies that all engineering students should take an introductory course on systems control.

It is widely accepted that control is more "engineering" than "science," but it does require a firm theoretical underpinning for it to be successfully applied to ever more challenging projects. This attention to theory in academia has led to discussions through the years on the "theory/practice Gap" which culminated in a recent special issue of the *IEEE Control Systems Magazine* (Volume 19, Number 6, 1999).

The development of computer software for control has provided many benefits for teaching, research, and the development of control systems design in industry. MATLAB® and Simulink® are considered the dominant software platforms for control system analysis and design, with numerous off-the-shelf toolboxes dedicated to control systems and related topics. As Confucius said, "The craftsman who wishes to work well has first to sharpen his implements,"[2] and it is clear that MATLAB provides a suitable implement for control engineering. The major objective of this book is to provide information on how MATLAB can be used in control system design by covering many methods and presenting additional software routines. Many students today view control theory as difficult because of the mathematics involved in evaluating frequency responses, plotting root loci, and doing the many other calculations which can be easily accomplished in MATLAB, as shown in this book. It is therefore our opinion that the educational objective today should be to give students sufficient knowledge of these techniques to understand their relevance and teach how to use them correctly without the burden of the calculations which MATLAB can accomplish.

A distinguishing feature of the book is the organization and presentation of the material. Based on our teaching, research, and industrial experience, we have chosen to present the course materials in the following sequence: system models, time and frequency domain analysis, introduction to various model reduction techniques, model-based control design methods, PID techniques and robust control. In addition, a chapter is in-

[1] Doyle J. C. '*A new physics*?'. plenary talk presented at the 40th IEEE Conference on Decision and Control Orlando, FL, Dec. 2001.

[2] `http://www.confucius.org/lunyu/ed1509.htm`.

cluded on fractional-order control as an alternative for practical robustness trade-offs. MATLAB scripts and plots are extensively used in this textbook to illustrate basic concepts and examples. A dedicated toolbox called CtrlLAB developed by the authors can be used as an effective teaching and learning aid. CtrlLAB was developed to support our objective of enabling control studies to be done in MATLAB by students with no knowledge of MATLAB, thus avoiding the need to replace less mathematics with the requirement of learning a programming language (although this is not difficult). CtrlLAB is the most downloaded package in the Control Systems category in the File Exchange of MATLAB Central.[3]

We hope that readers will enjoy playing with and changing the scripts as they gain better understanding and accomplish deeper exploration with reduced effort. Additionally, each chapter comes with a set of problems to strengthen the readers' understanding of the chapter contents.

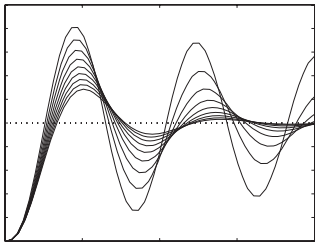
This book can be used as a reference text in the introductory control course for undergraduates in all engineering schools. The coverage of topics is broad, yet balanced, and should provide a solid foundation for the subsequent control engineering practice in both industry and research institutes. For graduates and researchers not majoring in control, this textbook is useful for knowledge enhancement. The authors also believe that this book will be a good desktop reference for control engineers.

The writing of this book started in the mid 1990s. In its evolving into the current form, many researchers, professors, and students have provided useful feedback, comments, and input. In particular, we thank the following professors: Xinhe Xu, Xingquan Ren, Yuanwei Jing, Taicheng Yang, Shuzhi Sam Ge, Igor Podlubny, Ivo Petras, István Kollár, Alain Oustaloup, Jocelyn Sabatier, Blas M. Vinagre, J. A. Tenreiro Machado, and Kevin L. Moore. Moreover, we are grateful to Elizabeth Greenspan, Acquisitions Editor of the Society for Industrial and Applied Mathematics (SIAM), for her professional help. The "Book Program" from The MathWorks Inc. is acknowledged for the latest MATLAB software.

Last, but not least, Dingyü Xue would like to thank his wife Jun Yang and his daughter Yang Xue; YangQuan Chen would like to thank his wife Huifang Dou and his sons Duyun, David, and Daniel, for their patience, understanding and complete support throughout this work. Derek Atherton wishes to thank his wife Constance for allowing him hours of overtime with many hardworking graduate students which included, in particular, many discussions with Dingyü when he was at Sussex and the email exchanges or with Dingyü and YangQuan, which led to this book.

Dingyü Xue, Northeastern University, Shenyang, China.
YangQuan Chen, Utah State University, Logan, UT, USA.
Derek P. Atherton, The University of Sussex, Brighton, UK.

[3] `http://www.mathworks.com/matlabcentral/index.shtml`



Chapter 1

Introduction to Feedback Control

1.1 Introduction

Feedback and control are almost everywhere. One can virtually link the powerful word "control" to almost anything, such as "diet control," "financial control," "pest control," "motor control," "robot control," etc. One can also say that "power is nothing without control," which is believed to be correct in both social and technological contexts. Feedback is an intuitive means for control. For example, when you *feel* cold (sensing), you add one more layer of cloth (decision and then control action) to keep yourself warm and comfortable (objective). This is biological feedback due to a change in the environment. In technological systems, the loop "sensing-feedback-decision-control" is implemented to change the system behavior into a desirable one. In most cases in this book, we shall focus on the "feedback control" for a given system described by ordinary differential equations (ODEs) with a single input–single output (SISO). More specifically, we will mainly concentrate on analytical and simulation methods for linear feedback control systems and a few aspects of simulation for nonlinear systems. For multiple input–multiple output (MIMO) linear systems, good references are [1–7].

Figure 1.1 shows a typical feedback control structure with three blocks, namely, the plant block, the controller block, and the feedback block. In this typical feedback control structure, the plant and the controller blocks form the forward path and the feedback path normally includes the sensor and, possibly, signal conditioning. This system structure is quite commonly seen in process control and other control applications.

For simplicity, throughout the book only the paths with negative actions will be labeled in the block diagram, and the ones with positive actions will have the plus sign omitted by default, as in Figure 1.1.

If all three blocks are linear, the feedback control structure can be redrawn, as shown in Figure 1.2. This model structure will be extensively used in the book.

In control systems, the concept of "feedback" is very important. If we assume that there is no feedback path, the system will be driven solely by the input signal, and after the effect of the control block, the output signal of the system will be generated. This kind of system structure is usually referred to as an open-loop control structure. Under ideal

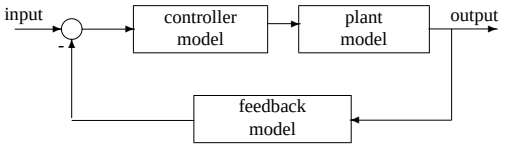


Figure 1.1. *Typical feedback structure.*

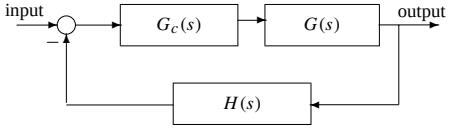


Figure 1.2. *Typical linear feedback structure.*

circumstances, an open-loop control strategy will work, but this is based on having an accurate plant model, which never exists in practice due to modeling errors and system disturbances. Thus, for accurate control a closed-loop system structure must be used instead. Closed-loop systems are often referred to as feedback control systems.

The objective of this book is to present methods for the analysis and design of feedback control systems using the interactive language MATLAB and its Control Systems Toolbox. Many methods are presented and details of the appropriate MATLAB routines given. For the routines, emphasis is placed on the effectiveness, relevance, and appropriateness of the different control design approaches covered. It is hoped that the reader will appreciate these aspects from the large number of examples included and will also recognize that practical specifications for a system's performance may include many factors. A design to meet these will invariably involve economic as well as technical considerations. This can result in systems operating in a nonlinear mode, so Simulink is introduced to show the value of simulation for these situations. Further, the technical specifications may require solutions which are not obtainable analytically, so Simulink is also used to show how numerical optimization solutions can be obtained. The appendix gives details of CtrlLAB, which provides a graphical user interface (GUI) for solving control problems which fit the structure of Figure 1.2. CtrlLAB is a flexible and powerful tool for self-learning, teaching, and engineering design and requires a minimum of user effort to obtain results. The features used in CtrlLAB are described in several of the book's chapters, but a reader with a basic control background may wish to read the appendix early on and start to use CtrlLAB for its ease in obtaining solutions to many control problems.

In practical control system design, the more general feedback control structure shown in Figure 1.3 is sometimes used with the feedback block simplified to 1. In Figure 1.3(a), the two submodels, the prefilter and controller, can be adjusted independently in control system design. This is often referred to as two-degrees-of-freedom control. In this book, we will focused on one-degree-of-freedom control problems.

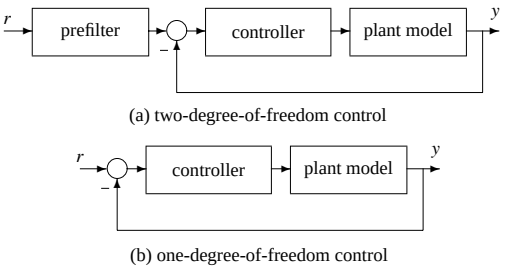


Figure 1.3. *Feedback control structures.*

1.2 Historical Background

The early development of automatic control devices can be traced back to the ancient water clock in Alexandria, Egypt, or to the ancient compass vehicle developed about 2,000 years ago during the Han Dynasty in China.

According to [8], the fly ball governor invented by James Watt in 1788 is regarded as the first widely used automatic feedback control system. Theoretical research on control systems was initiated by the study of stability problems involving differential equations pioneered by the work of Maxwell in 1868, Routh in 1874, and Hurwitz in 1895. Control strategy design problems were first proposed by Minorsky in 1922 in [9], where the three-term controller, or the PID (proportional integral derivative) controller, was first formulated. Practical algorithms for PID controller adjustment presented by Ziegler and Nichols in 1942 [10] still have an influence today in the practice of control engineering.

The framework of frequency domain analysis of linear feedback control systems was established in 1932 by the work of Nyquist [11], which was extended by Bode in 1945 [12] and Nichols in 1947 [13]. The root locus analysis proposed by Evans in 1948 [14] was another milestone in the study of linear feedback systems.

The introduction of the maximum principle proposed by Pontryagin in 1956 [15], dynamic programming by Bellman in 1957 [16], and state space representation by Kalman in 1959 [17] opened a new era of systems control which later became known as "modern control theory." Some of its significant achievements include the linear quadratic optimal regulator by Kalman in 1959, optimal state observers by Kalman in 1960, and the linear quadratic Gaussian (LQG) optimal controller also developed by Kalman [18, 19]. It was later found by Doyle in 1979 [20] that the LQG controller may reduce the stability margins of the system, which initiated interest in loop transfer recovery (LTR) design; see, e.g., [21].

Robust control is a very attractive new area in control systems design. Modern robust control investigations were started by Zames in 1981 in [22], where optimal control problems were formulated as the minimization of norms in Hardy spaces. The state space solution to such problems by Doyle et al. in 1989 is a significant computational contribution [23].

As will be demonstrated in this book, most feedback system analysis and design tasks can be solved easily using a computer. Therefore, suitable computer software is essential for control system investigations. The first generation of computer-aided analysis and design software includes the programs developed by Melsa and Jones [24] in 1970, in which a

significant amount of Fortran subroutines were provided. However, this required that the main program for any specific problem must be prepared by the user, making the solution procedure tedious and complicated.

The main feature of the second generation of the CACSD (computer-aided control system design) software was the provision of a man–machine interactive environment. Good examples of these software platforms are MATLAB, developed by Moler in 1980 [25], and INTRAC, developed by Åström in 1985 [26]. Currently, after generations of evolution among various CACSD software packages, MATLAB is the most dominant and widely used environment in engineering and nonengineering applications. In particular, for systems control, MATLAB is the most popular tool in research, development, and education. Furthermore, object-oriented programming techniques have been satisfactorily implemented in both the MATLAB and Simulink environments on which this book is based.

1.3 Structure of the Book

Broadly speaking, for systems control there are three major steps, i.e., **m**odeling, **a**nalyis and **d**esign, also known as the "**mad**" process. If one is given a system to control, one probably has to go through this "**mad**" process or loop to achieve a satisfactory control performance. The structure of this book follows a similar "**mad**" process.

For a systematic analysis and design of a control system, mathematical models of the components are usually required. For linear continuous-time system models, which will be the central theme of this book, there are usually four kinds of mathematical models, namely, the transfer function model, the state space model, the zero-pole-gain model, and more generally, the block diagram model.

The transfer function model is based on the theoretical results of Laplace transformation, a clever way to map linear system models described by ordinary differential equations (ODES) into corresponding algebraic equations. Many useful analysis and design tools based on this type of model are available.

The state space model, on the other hand, describes the internal characteristics of the system. When performing the analysis and design of control systems described by state space models, matrix algebra is extensively used.

The other two types of models can be used to either describe some of the characteristics of the system or describe more complicated systems. All the model types, although different in appearance, can be converted into each other. The details of models and their conversions are covered in Chapter 2.

In Chapter 2, we focus more on various model forms and their conversions rather than on how to build a model from experimental results, which is a large subject area known as "system identification," that we will briefly introduce.

Mathematical models of physical processes may be of relatively high order. For control system design, low-order models are often used, primarily because before the existence of modern computer software, calculations took a significant amount of time. This meant that expertise had been gained in both the understanding and designing of controllers for low-order models. It can therefore be useful to perform some form of model reduction in various phases of system analysis and controller design. The topic of model reduction is briefly introduced in the last section of Chapter 3.

System analysis methods for linear time-invariant (LTI) feedback systems covered in this book are briefly listed as follows:

- *Parametric analysis*: The characteristics of the system can be described by some parameters. For instance, the robustness can normally be measured by certain norm parameters of the system.
- *Time domain analysis*: Typically, the system response to a step input signal is often of direct interest and its properties may be a system specification. System responses to other signals are also useful in system analysis tasks. Analytical and numerical solutions to transient responses of linear control systems are covered.
- *Frequency domain analysis*: Frequency domain response tools are very useful in feedback control systems analysis and design. The form of a frequency response may also be a design specification. Based on the LTI model, the frequency responses can be easily evaluated with different graphical representations of the behavior of the system available in MATLAB. The dynamic performance of the system can be examined based on the graphical interpretations. There are basically two approaches to control system analysis, namely, time domain analysis and frequency domain analysis, and both are fully studied in Chapter 3.
- *Simulation analysis*: Simulation analysis of some nonlinear systems is covered in Chapter 4.

Most controller design methods utilize a mathematical model. We will refer to these as model-based design algorithms, and they provide the major content, which is summarized below, of Chapters 5–7.

- *Model-based approaches*: Model-based controller design approaches, including classical lead/lag cascaded compensators, the linear quadratic optimal controller, the pole placement controller, and decoupling controller, are presented in Chapter 5.
- *PID controllers*: PID controllers with different structures and parameter evaluation algorithms are studied in Chapter 6. Comparisons between various algorithms are presented. PID controllers are very widely used in industry, and we discuss some of their aspects such as consideration of integrator windup and relay automatic tuning. Optimal PID controller design using numerical techniques within Simulink is also discussed, as this approach allows consideration of multiple practical objectives.
- *Robust controllers*: Robust controller design techniques, starting from the LQG/LTR controller, are covered in Chapter 7 with a focus on Hardy space–based control, such as $\mathcal{H}_2$ and $\mathcal{H}_\infty$ controller design methods.
- *Fractional-order controllers*: Fractional-order controllers are covered in Chapter 8 for the first time in a textbook. These controllers are receiving increasing attention because of some of their more powerful properties. Again, MATLAB routines are given for studying their performance.

The appendix, as already mentioned, gives details of CtrlLAB and includes many examples of its use in feedback control system analysis and design.

Since this book is tightly coupled with MATLAB, a widely used computational software platform, we provide the following MATLAB survival guide that will be useful for beginners.

1.4 A Survival Guide to MATLAB

1.4.1 A Brief Overview of MATLAB

The MATLAB environment, also known as the MATLAB "language," was pioneered by Cleve Moler of the University of New Mexico in the early 1980s. A commercial version of MATLAB was first released in 1984 by The MathWorks Inc. This language is very easy to use and is a powerful tool for dealing with matrices. The graphical visualization utilities are impressive and flexible. Compared with other software packages, MATLAB has received outstanding merits in scientific computation and graphical visualization. MATLAB has now become the most widely used software in the field of control systems analysis and design, among other engineering and nonengineering areas. Numerous toolboxes have been written by well-known professionals. The Control Systems Toolbox and the Simulink program developed by The MathWorks Inc. will be extensively used in this book. All the examples used in this book are compatible with MATLAB version 7.5 (Release 18 or 2007b). It should also be noted that the material presented in the book does not rely too much on specific versions of MATLAB. Almost all the materials can be executed on earlier versions such as MATLAB 6.* or even MATLAB 5.*.

CtrlLAB, developed by the authors, is a GUI which can be used to solve typical problems in feedback control systems modeling, analysis, and design. CtrlLAB can be used as a companion for this book.

For a detailed description of MATLAB, please refer to [27, 28]. More comprehensive coverage of Simulink-related topics is presented in the recent textbook [29].

1.4.2 Standard MATLAB Statements and Functions

Unlike many other programming languages, the basic element in MATLAB is a complex-valued matrix, and powerful facilities have been provided for matrix manipulation and graphical visualization. To enter a matrix

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix},$$

one can simply issue the statement

```
>> A=[1,2,3; 4 5,6; 7,8 0];
```

where ≫ is the MATLAB prompt automatically given by the MATLAB program, the semicolons within the square brackets are used to separate the matrix rows, and a comma or a space is used to separate the elements in the same row. Vectors and scalars can also be accepted by MATLAB with even simpler statements.

More data structures, such as multidimensional arrays, structural data, object classes, and cell structures, are supported in MATLAB which makes the application of MATLAB easy and convenient.

Compared to other programming languages, MATLAB functions can be called in a special way. The syntax for a typical function call is

```
[list_of_return_variables]=func_name(input_list)
```


where the left-hand-side arguments in square brackets are the list of returned variables and the right-hand-side arguments are the input list used in the function. For instance, the function `bode()` can be called with the syntax

```
[mag,phase]=bode(G,w)
```

where the function `bode()` is used to draw the Bode diagram of the system given in variable G, and the input variable $\boldsymbol{w}$ is used to pass the frequency vector to the function. The [`mag`, `phase`] variables, which are the magnitude and phase vectors of the frequency response data, are then returned after the function call. One special feature of the MATLAB function is that different syntax definitions can be used in the same function to perform different manipulations. For instance, the `bode()` function can be called in the following formats:

```
bode(G,w)          % draw Bode diagram over frequency range w
bode(G)            % draw Bode diagram over default frequency range
bode(G1,G2,G3)     % draw Bode diagrams for several systems together
```

where the state space model and transfer function model can both be used and the MATLAB function can automatically detect which kind of input is provided. The advantage of using the G object is that the same function syntax can be used to handle continuous- and discrete-time systems, state space and transfer function models, single input–single output and multiple input–multiple output models, and so on. A unified framework of functions can be established, which greatly simplifies the task of system analysis and design.

1.4.3 Graphics Facilities in MATLAB

Two-dimensional curves can easily be drawn by calling the function `plot()` with the syntax

```
plot(x1,y1,x2,y2,x3,y3,···)
```

where $(\boldsymbol{x}_1, \boldsymbol{y}_1)$ is a pair of vectors (or matrices) holding the x- and y-axis data for the plots, $(\boldsymbol{x}_2, \boldsymbol{y}_2)$ is another pair, and so on. One may call other functions to enhance the plot, such as

(1) `grid` to add or remove grids on the plot;

(2) `xlabel()` and `ylabel()` to add labels for the axes;

(3) `title()` to add a title to the plot;

(4) `legend`, `text()` and `gtext()` to add one or more legends to plots.

One can also enhance the graphs in a visual way using the graphics processor in CtrlLAB, which will be explained in the appendix.

Three-dimensional plots can also be obtained by calling `mesh()` and `surf()` functions. Once correct variables are provided, the three-dimensional plot will be generated directly.

1.4.4 On-Line Help Facilities in MATLAB

In this book, MATLAB and its Control Systems Toolbox will be extensively used, and it will not be possible or suitable to have all the functions fully described. This is also

the case in other MATLAB related books such as [28]. Readers are advised to make full use of the MATLAB on-line help facilities for all the functions relevant to their specific work. For instance, a user can start the help process by issuing the `help` command in the MATLAB environment or by clicking the Help menu in the MATLAB interface, whence all the contents in the related directories will be displayed. A typical help message provided by the on-line help system is given below:

```
>> help lyap
```

The following message will be displayed:

```
LYAP   Lyapunov equation.
 X = LYAP(A,C) solves the special form of the Lyapunov
 matrix equation:
    A*X + X*A' = -C
 X = LYAP(A,B,C) solves the general form of the Lyapunov
 matrix equation:
    A*X + X*B = -C
 See also DLYAP.
```

When the `help` utility is used, an explanation of its calling syntax will be displayed for the function `lyap`. Alternatively, the command `doc` will display the on-line help information in HTML format.

In addition, the `lookfor` command can be used to search for a key word in the functions. For instance, if one wants to find a function which can be used to perform `'Hankel'` related manipulations, one can try

```
>> lookfor hankel
```

and the following information can be obtained:

```
HANKEL Hankel matrix.
BESSELH Bessel function of the third kind (Hankel function).
HANK2SYS Convert a Hankel matrix to a linear system model.
HSVOPTIONS  Creates option list for Hankel singular value plot.
BHRDEMO Demo of model reduction techniques (Hankel, Balanced, BST).
HKSV Hankel singular values and grammians P, Q.
OHKAPP Optimal Hankel norm approximation (stable plant).
OHKDEMO Demo of optimal Hankel model reduction technique.
OHKLMR Optimal Hankel norm approximation (unstable plant).
```

From the above displayed results, one can decide which function may be suitable for the intended task.

1.4.5 MATLAB Toolboxes

The Control Systems Toolbox is extensively used in this book to deal with the problems in the area of feedback control system analysis and design. Most parts of the Control Systems Toolbox are covered in this textbook together with CtrlLAB (see the appendix).

There are many toolboxes applicable to problems in control. Some of them are listed (in alphabetical order) below:

- Chemometrics Toolbox, by Richard Kramer;
- Control Systems Toolbox, by Jack Little et al.;

- CtrlLAB ToolKit, by Dingyü Xue (see the appendix);
- Frequency Domain Identification Toolbox, by I. Kollár and J. Schoukens;
- Fuzzy Logic Toolbox, by Ned Gulley et al.;
- LMI Control Toolbox, by Pascal Gahinet and Arkadi Nemirovski;
- Model Predictive Control Toolbox, by Manfred Morari and L. Ricker;
- Modified Maximum Likelihood Estimator Toolbox, by Wes Wang;
- μ-Analysis and Synthesis Toolbox, by G. Balas, A. Packard, and J. Doyle;
- Multivariable Frequency Domain Toolbox by Jan Meciejowski et al.;
- Neural Network Based Control Toolkit, by Magnus Nøgaard;
- Neural Network Based Identification Toolkit, by Magnus Nøgaard;
- Neural Network Toolbox, by Howard Demuth and Mark Beale;
- Nonlinear Control Design Blockset, by M. Yeddanapudi and A. Potvin;
- Polynomial Toolbox, by D. Henrion, F. Kraffer and H. Kwakernaak;
- QFT Control Design Toolbox, by Craig Borghesani, Yossi Chait, *et al.*;
- RIOTS_95, by Adam L. Schwartz, YangQuan Chen and Elya Polak;
- Robotics Toolbox, by Peter Corke;
- Robust Control Toolbox, by Richard Chiang and Michael Sofanov;
- Signal Processing Toolbox, Jack Little and Loren Shure;
- System Identification Toolbox, by Lennart Ljung.

Moreover, there are other toolboxes which may be useful for mathematical solutions to some problems in control systems, such as the Communications Toolbox, the Genetic Algorithm Optimization Toolbox, the Image Processing Toolbox, the Optimization Toolbox, the Partial Differential Equation Toolbox, the NAG Foundation Toolbox, the Spline Toolbox, the Statistics Toolbox, the Symbolic Toolbox, the Wavelet Toolbox, etc. Detailed information on the above mentioned MATLAB toolboxes can be found online; readers can consult the following two Web sites to explore further and find more information:

`http://www.mathworks.com/matlabcentral/index.shtml`
`http://www.mathtools.net/index.html`

CtrlLAB and all the code used in this book can also be downloaded from

`www.siam.org/books/dc14`

Problems

1. Run the MATLAB demonstration program by typing the `demo` command under the MATLAB prompt and see the attractive aspects provided by MATLAB and its toolboxes.
2. Find out what MATLAB toolboxes have been installed on the computer system you are using by typing the `help` or `ver` command. Try to install other toolboxes such as CtrlLAB and modify your own MATLAB search path.

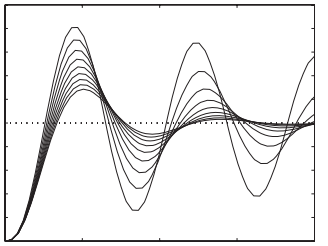
3. Prepare a MATLAB script to rotate the following matrices 90° and then compute their norms, traces, eigenvalues, pseudo-inverses, and characteristic polynomials:

$$\text{(a) } \boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 & 3 \\ 2 & 3 & 5 & 7 \\ 1 & 3 & 5 & 7 \\ 3 & 2 & 3 & 9 \\ 1 & 8 & 9 & 4 \end{bmatrix}, \quad \text{(b) } \boldsymbol{B} = \begin{bmatrix} 1 & 4 & 3 & 6 & 7 & 8 \\ 2 & 3 & 3 & 5 & 5 & 4 \\ 2 & 6 & 5 & 3 & 4 & 2 \\ 1 & 8 & 9 & 5 & 4 & 3 \end{bmatrix}.$$

4. Solve for the matrix $\boldsymbol{X}$ in the Lyapunov equation $\boldsymbol{AX} + \boldsymbol{XA}^{\mathrm{T}} = \boldsymbol{C}$ with

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} 1 & 5 & 4 \\ 5 & 6 & 7 \\ 4 & 7 & 9 \end{bmatrix}.$$

5. Draw the function $\mathrm{e}^{-t^2/2}\sin(5t)$ for $t \in (0, 2\pi)$ using different functions such as `plot()`, `stairs()`, `ezplot()`, and `stem()`.
6. Call the demonstration function `peaks()` by the command `[x,y,z]=peaks;` use the resulting `(x,y,z)` to show different three-dimensional graphs through the functions `mesh()`, `surf()`, and `waterfall()`.



Chapter 2

Mathematical Models of Feedback Control Systems

Most, but not all, of the existing design procedures for a control system make use of mathematical models. It is therefore important to try to obtain accurate mathematical models for the system components. The system can then be analyzed and designed in a systematic way and its properties assessed using the mathematical models as approximations of its true behavior.

If the system model is not known, two methods can be used to build a model of the system for the analysis and design tasks. The first method is to derive the system model using existing physical laws or principles. The second method, more often used, is to find an approximate mathematical model based on the observed response data of the system. The former method is referred to as the physical modeling and the latter the system identification. How to obtain a model of the system to be controlled is a large subject area and will not be fully pursued in this book. Instead, we will focus on how to manipulate the models.

In this chapter, the physical modeling problem is illustrated through an example in Sec. 2.1. In Sec. 2.2, the concept of the Laplace transformation is given with MATLAB-based solutions. The transfer function representation of linear systems is described in Sec. 2.3. Various descriptions of the standard transfer functions within MATLAB are presented. Other commonly used system descriptions, such as the state space representation and the zero-pole-gain representation, are presented in Sec. 2.4. The modeling principles for finding an overall system model from a given complicated interconnected submodel are presented in Sec. 2.5. In Sec. 2.6, the equivalent conversion among different model types for a given system is described. For instance, a given transfer function model can be converted into an equivalent state space model, or a given state space model can be converted into the transfer function form or the zero-pole-gain form. A comprehensive introduction to the system identification problem will be given briefly in Sec. 2.7, with an emphasis on discrete-time model identification and identification input signal selections.

2.1 A Physical Modeling Example

Consider the electric circuit shown in Figure 2.1, where a resistor R, an inductor L, and a capacitor C are connected in series. For this dynamic system, the input signal is $u(t)$ and the output signal is $u_c(t)$.

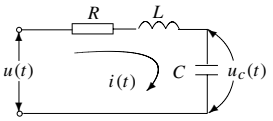


Figure 2.1. *An* RLC *series circuit.*

The current $i(t)$ satisfies

$$i(t) = C\frac{\mathrm{d}u_c(t)}{\mathrm{d}t} \tag{2.1}$$

and the voltage equation can be written as

$$u(t) = Ri(t) + L\frac{\mathrm{d}i(t)}{\mathrm{d}t} + u_c(t). \tag{2.2}$$

Substituting (2.1) into (2.2), one has

$$LC\frac{\mathrm{d}^2u_c(t)}{\mathrm{d}t^2} + RC\frac{\mathrm{d}u_c(t)}{\mathrm{d}t} + u_c(t) = u(t). \tag{2.3}$$

The second-order ordinary differential equation (ODE) given in (2.3) is called the mathematical model of the electric circuit.

In general, the mathematical model of a continuous-time, lumped parameter dynamic system can be represented by an ODE.

2.2 The Laplace Transformation

From the voltage equation (2.3), the voltage $u_c(t)$ across the capacitor C can be represented by a second-order linear ODE. A method used by engineers to solve linear differential equations is the Laplace transformation method, which is reviewed, below.

Definition 2.1. The Laplace transformation of a time function $f(t)$ is defined by

$$\mathscr{L}[f(t)] = \int_0^\infty f(t)\mathrm{e}^{-st}\mathrm{d}t = F(s), \tag{2.4}$$

where $\mathscr{L}[f(t)]$ is shorthand notation for the Laplace integral transformation.

The result of the Laplace transformation is a function of s, a complex variable, often denoted by $F(s)$. It should be noted that s has a unit of second^{-1}.

For a given function $f(t)$, it is usually possible to find its Laplace transformation via a Laplace transformation table, or by the direct use of the relevant MATLAB functions.

Theorem 2.1. Some of the important properties of the Laplace transformation are listed below without proofs.

1. *Linearity*: If a and b are scalars, then

$$\mathscr{L}[af(t) \pm bg(t)] = a\mathscr{L}[f(t)] \pm b\mathscr{L}[g(t)].$$

2. *Translation in time:* $\mathscr{L}[f(t-a)] = \mathrm{e}^{-as}F(s)$.
3. *Translation in s:* $\mathscr{L}[\mathrm{e}^{-at}f(t)] = F(s+a)$.
4. *Differentiation:* $\mathscr{L}[\mathrm{d}f(t)/\mathrm{d}t] = sF(s) - f(0^+)$. The nth order derivative can be evaluated from

$$\mathscr{L}\left[\frac{\mathrm{d}^n}{\mathrm{d}t^n}f(t)\right] = s^nF(s) - s^{n-1}f(0^+) - s^{n-2}\frac{\mathrm{d}f(0^+)}{\mathrm{d}t} - \cdots - \frac{\mathrm{d}^{n-1}f(0^+)}{\mathrm{d}t^{n-1}}. \tag{2.5}$$

   When all the initial values of $f(t)$ and its derivatives are zero, equation (2.5) can be further simplified to $\mathscr{L}[\,\mathrm{d}^nf(t)/\mathrm{d}t^n] = s^nF(s)$.
5. *Integration*: If zero initial conditions are assumed, $\mathscr{L}[\int_0^t f(\tau)\mathrm{d}\tau] = F(s)/s$. For the n-th order integration of a given function $f(t)$,

$$\mathscr{L}\left[\int_0^t \cdots \int_0^t f(\tau)(\mathrm{d}\tau)^n\right] = \frac{F(s)}{s^n}. \tag{2.6}$$

6. *Initial time and final time:*

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

7. *Convolution*: $\mathscr{L}[f(t) * g(t)] = \mathscr{L}[f(t)]\mathscr{L}[g(t)]$, where the convolution operator $*$ is defined as

$$f(t) * g(t) = \int_0^t f(\tau)g(t-\tau)\mathrm{d}\tau = \int_0^t f(t-\tau)g(\tau)\mathrm{d}\tau. \tag{2.7}$$

8. *Others:*

$$\mathscr{L}[t^nf(t)] = (-1)^n\frac{\mathrm{d}^nF(s)}{\mathrm{d}s^n}, \qquad \mathscr{L}\left[\frac{f(t)}{t^n}\right] = \int_s^\infty \cdots \int_s^\infty F(s)\mathrm{d}s^n. \tag{2.8}$$

A MATLAB function `laplace()`, provided in the Symbolic Toolbox, can be used to evaluate the Laplace transform from a given function $f(t)$. The syntax of the function is `F=laplace(f)`. Note that only a limited class of signals $f(t)$ can be used with `laplace(f)`.

Example 2.1. If one wants to perform the Laplace transformation for the function $\mathrm{e}^{bt}\cos(at+c)$, the following MATLAB statements can be used:

```
>> syms s t a b c; F=laplace(exp(b*t)*cos(a*t+c))
```

and the Laplace form of the function is

$$F(s) = \frac{\cos(c)(s-b)}{(s-b)^2+a^2} - \frac{\sin(c)a}{(s-b)^2+a^2}.$$

Definition 2.2. The inverse Laplace transformation of a given function $F(s)$ is defined by

$$f(t) = \mathscr{L}^{-1}[F(s)] = \frac{1}{2\pi\,\mathrm{j}}\int_{\sigma-\mathrm{j}\infty}^{\sigma+\mathrm{j}\infty} F(s)\mathrm{e}^{st}\mathrm{d}s, \tag{2.9}$$

where σ is greater than the real part of singularities of $F(s)$.

Given $F(s)$, its inverse Laplace transformation can be performed using a table or other relevant tools. With the use of the Symbolic Toolbox, the inverse Laplace transform can be evaluated from `f=ilaplace(F)`. Note again that only a limited class of $F(s)$ can be used with `ilaplace(F)`.

2.3 Transfer Function Models

2.3.1 Transfer Functions of Control Systems

It is obvious that, applying the differentiation law of Laplace transformation to the voltage equation, the differential equation (2.3) can be transformed into an "algebraic" equation as follows:

$$LCU_c(s)s^2 + RCU_c(s)s + U_c(s) = U(s), \tag{2.10}$$

where $U_c(s) = \mathscr{L}[u_c(t)]$, $U(s) = \mathscr{L}[u(t)]$, if zero initial conditions for $u_c(t)$ and its derivatives are assumed. Dividing both sides by $U_c(s)$ and taking the reciprocal yields

$$\frac{U_c(s)}{U(s)} = \frac{1}{LCs^2 + RCs + 1} \tag{2.11}$$

and $U_c(s)/U(s)$ is referred to as the transfer function from the input signal $u(t)$ to the output signal $u_c(t)$.

The transfer function of a linear continuous system can be generally defined by a rational function of the variable s in the form

$$G(s) = \frac{b_1 s^m + b_2 s^{m-1} + \cdots + b_m s + b_{m+1}}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_{n-1} s + a_n}. \tag{2.12}$$

If the coefficients b_i, $(i = 1, \ldots, m+1)$ and a_i, $(i = 1, \ldots, n)$ are constants, the system is referred to as a linear time invariant (LTI) system. The denominator is referred to as the characteristic polynomial of the system. The highest order n of the denominator is referred to as the order of the system. For a physically realizable system, it is often true that $m \leq n$. In this case, the system is called proper. If $m < n$, the system is called strictly proper. The value $n - m$ is sometimes called the relative degree/order or pole-zero excess of the system.

2.3.2 MATLAB Representations of Transfer Functions

A transfer function model can be easily entered into the MATLAB environment using the following MATLAB statements:

```
num=[b1,b2,...,bm,bm+1]; den=[1,a1,a2,...,an-1, an];
G=tf(num,den)
```

i.e., it is required to enter the numerator and denominator polynomial coefficients separately into two vector variables `num` and `den` in the descending order of s. The variable `G` returned is the transfer function object.

Example 2.2. The simple transfer function

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

can be represented in MATLAB as

```
>> num=[1,5]; den=[1,2,3,4,5]; G=tf(num,den)
```

and the system object G can then be used to uniquely describe the given transfer function.

Example 2.3. An even more complicated transfer function model

$$G(s) = \frac{6(s+5)}{(s^2+3s+1)^2(s+6)(s^3+6s^2+5s+3)}$$

can be entered into MATLAB using the statements

```
>> den=conv(conv(conv([1,3,1],[1,3,1]),[1,6]),[1,6,5,3]);
   num=6*[1,5]; G=tf(num,den)
```

and

$$G(s) = \frac{6s+30}{s^8 + 18s^7 + 124s^6 + 417s^5 + 740s^4 + 729s^3 + 437s^2 + 141s + 18},$$

where `conv()` is a standard MATLAB function used to evaluate the convolution of two vectors. Note that the multiplication of polynomials can be equivalently performed by calling `conv()`. The `conv()` function can be nested arbitrarily. However, one should make sure that the brackets are matched properly to avoid any possible error message.

Alternatively, a transfer function given in factorized form can be entered into MATLAB by declaring, s, the Laplace complex variable, with `s=tf('s')`, and the transfer function can then be specified in a mathematical way:

```
>> s=tf('s');
   G=6*(s+5)/(s^2+3*s+1)^2/(s+6)/(s^3+6*s^2+5*s+3)
```

Apart from the essential numerator and denominator variables, other fields are also defined in the transfer function object. One can list all the possible fields using the `set(tf)` command. The other useful fields in the transfer function object include, for instance, `ioDelay` and `Ts`, which correspond, respectively, to the input-output delay and the sampling interval; the latter is applicable only to discrete-time systems. The field `Variable` is defined as the operator symbol used in the transfer function, with s and p for continuous-time systems, and z, z^{-1} and q for discrete-time systems, where q is shorthand for z^{-1}.

If one wants to change the operator symbol in the transfer function representation to p, and assign a transport delay of 0.5 seconds, either of the following two sets of MATLAB commands can be used:

```
G.Variable='p'; G.ioDelay=0.5;
set(G,'Variable','p','ioDelay',0.5);
```

The model G is then displayed as

$$\mathrm{e}^{-0.5p}\frac{6p+30}{p^8 + 18p^7 + 124p^6 + 417p^5 + 740p^4 + 729p^3 + 437p^2 + 141p + 18}.$$

2.3.3 Transfer Function Matrices for Multivariable Systems

Systems with one input and one output are referred to as single input–single output (SISO) systems, while systems with more than one input and more than one output are referred to as multiple input–multiple output (MIMO) systems. For an MIMO system, the transfer function representation is in fact denoted by a matrix of transfer functions which is called the transfer function matrix.

The transfer function object `tf` can also be used to represent MIMO transfer function matrices. Note that, in this book, many MATLAB functions work for both SISO and MIMO systems. However, our default is SISO if not otherwise stated.

Example 2.4. Assume that the transfer function matrix of an MIMO system is given by

$$\boldsymbol{G}(s)=\begin{bmatrix}\dfrac{0.1134\mathrm{e}^{-0.72s}}{1.78s^2+4.48s+1} & \dfrac{0.924}{2.07s+1}\\ \dfrac{0.3378\mathrm{e}^{-0.3s}}{0.361s^2+1.09s+1} & \dfrac{-0.318\mathrm{e}^{-1.29s}}{2.93s+1}\end{bmatrix}.$$

This model can be entered into the MATLAB workspace using the following commands:

```
>> g11=tf(0.1134,[1.78 4.48 1],'ioDelay',0.72);
   g12=tf(0.924,[2.07 1]);
   g21=tf(0.3378,[0.361 1.09 1],'ioDelay',0.3);
   g22=tf(-0.318,[2.93 1],'ioDelay',1.29);
   G=[g11, g12; g21, g22];
```

In the above example, the individual transfer functions of the transfer function matrix are entered first, and then these elements are grouped together to establish the whole transfer function matrix object for the MIMO system.

The numerators and denominators of the system can be retrieved with the function call

```
[num,den]=tfdata(G,'v')
```

2.3.4 Transfer Functions of Discrete-Time Systems

The discrete-time transfer function

$$H(z)=\frac{b_0z^m+b_1z^{m-1}+\cdots+b_{m-1}z+b_m}{a_1z^n+a_2z^{n-1}+\cdots+a_nz+a_{n+1}}z^{-d} \tag{2.13}$$

which is obtained via Z transforms from difference equations, can also be entered into MATLAB with the similar statements

```
num=[b0,b1,⋯,bm-1,bm];    den=[a1,a2,⋯,an,an+1];
H=tf(num,den,'Ts',T,'ioDelay',d);
```


where T is the sampling interval and m is the transport delay. Alternatively, the z variable can be declared with `z=tf('z',T)` before specifying the transfer function in a mathematical way.

Example 2.5. Assume that a discrete-time system model is given by

$$H(z)=\frac{6z^2-0.6z-0.12}{z^4-z^3+0.25z^2+0.25z-0.125}z^{-5},$$

where $T=0.1$ second, the following statement can be used:

```
>> num=[6 -0.6 -0.12]; den=[1 -1 0.25 0.25 -0.125];
   H=tf(num,den,'Ts',0.1,'ioDelay',5)
```

Alternatively, one may specify the system by

```
>> z=tf('z',0.1);
   H=(6*z^2-0.6*z-0.12)/(z^4-z^3+0.25*z^2+0.25*z-0.125);
   H.ioDelay=5;
```

2.4 Other Mathematical Model Representations

2.4.1 State Space Modeling

State space representations of control system models have been widely used in control theory since the 1960s, which was when the well-established, so-called "modern control theory" was introduced. The state space is another way of describing a dynamic model of the system, and it can be used to represent not only linear systems but also nonlinear systems. The state space representation of a system is always referred to as the internal model description because the internal variables, such as the states, are fully described in such a model representation. In contrast, the transfer function representation is often called the external model, or the input-output model, since only the input-output relationship of the system is described.

Consider again the RLC circuit model given in (2.3). If one assumes that $x_1=u_c$ and $x_2=\mathrm{d}u_c/\mathrm{d}t$, a second-order ODE can be rewritten into the following form:

$$\begin{cases}\dfrac{\mathrm{d}x_1}{\mathrm{d}t}=x_2,\\ \dfrac{\mathrm{d}x_2}{\mathrm{d}t}=-\dfrac{1}{LC}x_1-\dfrac{R}{L}x_2+\dfrac{1}{LC}u.\end{cases} \tag{2.14}$$

In control theory, $\mathrm{d}x_i/\mathrm{d}t$ is often denoted by $\dot{x}_i$ and the matrix form of the above equations is written as

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix}=\begin{bmatrix}0 & 1\\ -1/(LC) & -R/L\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}0\\ 1/(LC)\end{bmatrix}u, \tag{2.15}$$

where x_1 and x_2 are referred to as the state variables, u is referred to as the input signal, and (2.15) is called the state equation of the system. Note that the state variable selection

is not unique. Thus, the state equation is also not unique. For instance, if one selects the voltage u_c and the current i as the state variables, denoted by x_1 and x_2, respectively, the state equation can then be written as

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix}=\begin{bmatrix}0 & 1/C\\ -1/L & -R/L\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}0\\ 1/L\end{bmatrix}u. \tag{2.16}$$

It is readily seen that there are differences in the above two state equations.

Suppose that there are p inputs $u_i(t), (i = 1, \ldots, p)$ and q outputs $y_i(t), (i = 1, \ldots, q)$, and there are n states which make up a state variable vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^{\mathbf{T}}$. The state space expression of the general dynamic system can be written as

$$\begin{cases}\dot{x}_i = f_i(x_1, x_2, \ldots, x_n, u_1, \ldots, u_p), & i = 1, \ldots, n,\\ y_i = g_i(x_1, x_2, \ldots, x_n, u_1, \ldots, u_p), & i = 1, \ldots, q,\end{cases} \tag{2.17}$$

where $f_i(\cdot)$ and $g_i(\cdot)$ can be any nonlinear functions. For LTI systems, the state space expression of the system can be simplified as

$$\begin{cases}\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t),\\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t),\end{cases} \tag{2.18}$$

where $\boldsymbol{u} = [u_1, \ldots, u_p]^{\mathbf{T}}$ and $\boldsymbol{y} = [y_1, \ldots, y_q]^{\mathbf{T}}$ are the input and output vectors, respectively. The matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are compatible matrices. The term "compatible" means that the related matrices have the correct dimensions. To be more specific, we say that $\boldsymbol{A}$ is an $n \times n$ matrix, $\boldsymbol{B}$ is an $n \times p$ matrix, $\boldsymbol{C}$ is a $q \times n$ matrix, and $\boldsymbol{D}$ is a $q \times p$ matrix. The dimensions under such conditions are called compatible dimensions.

As a side note, we mention that in robust control theory, the state space expression is often denoted by

$$\boldsymbol{G}(s) = \left[\begin{array}{c|c}\boldsymbol{A} & \boldsymbol{B}\\ \hline \boldsymbol{C} & \boldsymbol{D}\end{array}\right] \tag{2.19}$$

as shorthand notation.

The representation of a state space expression in MATLAB is simple and straightforward. One can simply enter the coefficient matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ into the MATLAB environment, and the state space object can be entered as `G=ss(A,B,C,D)`.

Example 2.6. A two input–two output system in state space form given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix}2.25 & -5 & -1.25 & -0.5\\ 2.25 & -4.25 & -1.25 & -0.25\\ 0.25 & -0.5 & -1.25 & -1\\ 1.25 & -1.75 & -0.25 & -0.75\end{bmatrix}\boldsymbol{x} + \begin{bmatrix}4 & 6\\ 2 & 4\\ 2 & 2\\ 0 & 2\end{bmatrix}\boldsymbol{u}, \quad \boldsymbol{y} = \begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 2 & 0 & 2\end{bmatrix}\boldsymbol{x}$$

can be entered into the MATLAB workspace using the following MATLAB statements:

```
>> A=[2.25, -5, -1.25, -0.5; 2.25, -4.25, -1.25, -0.25;
       0.25, -0.5, -1.25,-1; 1.25, -1.75, -0.25, -0.75];
   B=[4, 6; 2, 4; 2, 2; 0, 2];
   C=[0, 0, 0, 1; 0, 2, 0, 2];
   D=zeros(2,2); G=ss(A,B,C,D)
```


If the system matrices entered are not compatible, the error messages will be given automatically by the `ss()` function.

For discrete-time state space models

$$\begin{cases}\boldsymbol{x}[(k+1)T] = \boldsymbol{F}\boldsymbol{x}(kT) + \boldsymbol{G}\boldsymbol{u}(kT),\\ \boldsymbol{y}(kT) = \boldsymbol{C}\boldsymbol{x}(kT) + \boldsymbol{D}\boldsymbol{u}(kT)\end{cases} \tag{2.20}$$

with sampling interval T, the statement `G=ss(A,B,C,D,'Ts',T)` can be used directly.

2.4.2 Zero-Pole-Gain Description

The zero-pole-gain representation is another way to describe the transfer function of an SISO LTI system. The zero-pole-gain model of a given transfer function is usually represented as

$$G(s) = K\frac{(s+z_1)(s+z_2)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)}, \tag{2.21}$$

where K is referred to as the gain of the system. Note that K is not the zero frequency or DC (direct current) gain $G(0)$. In (2.21), $-z_i$ $(i = 1, \ldots, m)$ are called the zeros and $-p_i$ $(i = 1, \ldots, n)$ the poles of the system. It is noted that for real-coefficient transfer function models, the poles and zeros are either real or in complex conjugate pairs. The zero-pole-gain representation is immediately obtainable from the transfer function representation of a given system.

To enter the zero-pole-gain model representation into the MATLAB workspace, issue the following MATLAB statements:

```
z=-[z1; z2; ⋯; zm];   p=-[p1; p2; ⋯; pn];
G=zpk(z,p,K)
```

Alternatively, the pole-zero-gain model can also be established by declaring first

```
s=zpk('s')
```

.

Example 2.7. The zero-pole-gain model

$$G(s) = 6\frac{(s+1.9294)(s+0.0353\pm 0.9287\,\mathrm{i})}{(s+0.9567\pm 1.2272\,\mathrm{i})(s-0.0433\pm 0.6412\,\mathrm{i})}$$

can easily be entered into the MATLAB workspace by the following:

```
>> z=[-1.9294; -0.0353+0.9287j; -0.0353-0.9287j];
   p=[-0.9567+1.2272j; -0.9567-1.2272j;
       +0.0433+0.6412j; +0.0433-0.6412j];
   G=zpk(z,p,6)
```

and it can be displayed that

$$G(s) = \frac{6(s+1.929)(s^2+0.0706s+0.8637)}{(s^2-0.0866s+0.413)(s^2+1.913s+2.421)}.$$

2.5 Modeling of Interconnected Block Diagrams

The model formats described above are usually directly obtainable for single block models. In practical situations, a system model may be given by interconnected blocks, and the overall system model for the interconnected system structures can be obtained using the methods given in this section.

2.5.1 Series Connection

Consider the series connection of the two blocks shown in Figure 2.2(a). It can be seen that the input signal $\boldsymbol{u}(t)$ travels through the first block $\boldsymbol{G}_1(s)$, and the output of $\boldsymbol{G}_1(s)$ is the input to the second block $\boldsymbol{G}_2(s)$, which generates the output $\boldsymbol{y}(t)$ of the overall system. This kind of connection is referred to as a series, or cascade, connection and it is assumed that, in connecting a block, it does not "load" the previous block.

In the series connection, the overall transfer function of the whole system is given by $\boldsymbol{G}(s) = \boldsymbol{G}_2(s)\boldsymbol{G}_1(s)$. For SISO systems, the two blocks $G_1(s)$ and $G_2(s)$ are interchangeable, i.e., $G_1G_2 = G_2G_1$. For MIMO systems, however, the two blocks are generally not interchangeable.

Assume that the MATLAB description of the model $\boldsymbol{G}_1(s)$ is represented in an LTI object $\boldsymbol{G}_1$, which is either `tf`, `ss`, or `zpk`, and that $\boldsymbol{G}_2(s)$ is represented by $\boldsymbol{G}_2$. The overall system in a series connection can be simply obtained using the MATLAB statement $\boldsymbol{G}$=$\boldsymbol{G}_2$*$\boldsymbol{G}_1$.

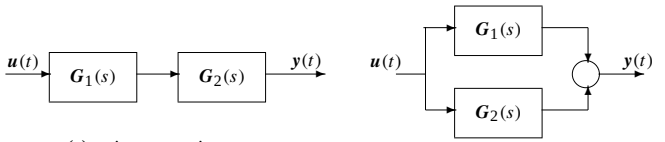
If the models G_1 and G_2 are given by symbolic variables, the above operation is also valid.

2.5.2 Parallel Connection

A typical parallel connection of two blocks $\boldsymbol{G}_1(s)$ and $\boldsymbol{G}_2(s)$ is shown in Figure 2.2(b), where the two blocks are subjected to the same input signal $\boldsymbol{u}(t)$. The outputs of the two blocks are summed up to form the output $\boldsymbol{y}(t)$ of the overall system. The overall transfer function of the parallel connection is then $\boldsymbol{G}(s) = \boldsymbol{G}_1(s) + \boldsymbol{G}_2(s)$.

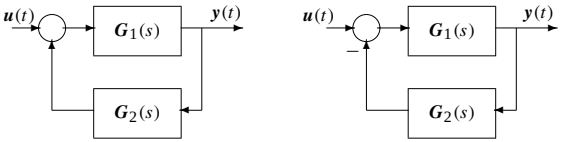
The LTI representation of the parallel connection can be obtained using the MATLAB statement $\boldsymbol{G}$=$\boldsymbol{G}_1$+$\boldsymbol{G}_2$, where $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ are LTI objects (`tf`, `ss`, or `zpk`) of $\boldsymbol{G}_1(s)$ and $\boldsymbol{G}_2(s)$, respectively. They can also be symbolic variables.

Example 2.8. It should be noted that, if $G_1(s)$ and $G_2(s)$ contain the same pole, then the result of the parallel manipulation may be simplified further in this case. Consider the two



(a) series connection (b) parallel connection

Figure 2.2. *Interconnections of blocks.*



(a) positive feedback (b) negative feedback

Figure 2.3. *Feedback connections.*

blocks $G_1(s) = 1/(s+1)^2$ and $G_2(s) = 1/(s+1)$. The result of calling the appropriate MATLAB functions is given by the following MATLAB statements:

```
>> G1=zpk([],[-1,-1],1); G2=zpk([],[-1],1); G=G1+G2
```

As a result, the overall system obtained is $G = (s+2)(s+1)/(s+1)^3$. In fact, the overall transfer function can be simplified to $G(s) = (s+2)/(s+1)^2$, since there is a common factor $(s+1)$ in both the denominators of $G_1(s)$ and $G_2(s)$.

The minimum realization technique can be used to obtain the simplified model, and details of the technique will be given later in this chapter.

2.5.3 Feedback Connection

The simple feedback connection of two blocks $\boldsymbol{G}_1(s)$ and $\boldsymbol{G}_2(s)$ is shown in Figures 2.3(a) and (b), respectively. The two feedback connections in Figure 2.3 are different; the left one is called a system with positive feedback and the right a system with negative feedback. The overall transfer function of the positive feedback is $\boldsymbol{G}(s) = \boldsymbol{G}_1(s)[\boldsymbol{I} - \boldsymbol{G}_2(s)\boldsymbol{G}_1(s)]^{-1}$, and for negative feedback it is $\boldsymbol{G}(s) = \boldsymbol{G}_1(s)[\boldsymbol{I} + \boldsymbol{G}_2(s)\boldsymbol{G}_1(s)]^{-1}$.

A MATLAB function `feedback()` is provided in the Control Systems Toolbox to get the overall system model from the feedback connection with the syntax

$\boldsymbol{G}$=`feedback(`$\boldsymbol{G}_1$`,`$\boldsymbol{G}_2$`,Sign)`,

where `Sign` is used to identify the positive or negative feedback connection. If `Sign=-1`, the negative feedback structure is indicated. The `Sign` variable can be omitted in the function call in a negative feedback connection. The LTI objects in the forward path and feedback path are given by $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$, respectively.

A MATLAB function, `feedback()`, is also written for the models represented by symbolic variables:

```
function H=feedback(G1,G2,key)
if nargin==2; key=-1; end,
H=G1/(sym(1)-key*G1*G2); H=simple(H);
```

This function should be placed under the `@sym` directory under MATLAB path. This function is useful in theoretically deriving the overall model from more complicated subsystem configuration.

Example 2.9. Consider again the models in Example 2.8. If a negative feedback connection is assumed, one can find the overall transfer function by using the following MATLAB statements:

```
>> G1=tf(1,[1 2 1]); G2=tf(1,[1 1]); G=feedback(G1,G2)
```

and it can be found that

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

For a positive feedback connection, the overall system model can be obtained from

```
>> G=feedback(G1,G2,+1)
```

where

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

2.5.4 More Complicated Connections

In the real world, a system structure can be very complex. In this section, we illustrate how to handle more complicated interconnections.

Consider the typical feedback control system structure shown in Figure 1.2. The overall system can be evaluated by assuming first that $G_c(s)$ and $G(s)$ are in series and then that they are connected to a negative feedback block $H(s)$. The overall closed-loop transfer function of the typical feedback control system is

$$G_{\rm cl}(s) = \frac{G(s)G_c(s)}{1+H(s)G(s)G_c(s)}. \tag{2.22}$$

The overall system object can be obtained using the MATLAB statement

`G_cl=feedback(G*Gc,H)`.

Example 2.10. If the three blocks in the above typical feedback structure are given by

$$G(s) = \frac{s^3+7s^2+24s+24}{s^4+10s^3+35s^2+50s+24}, \quad G_c(s) = \frac{10s+5}{s}, \quad H(s) = \frac{1}{0.01s+1},$$

the overall transfer function of the closed-loop system can be obtained by the following MATLAB statements:

```
>> G=tf([1 7 24 24],[1,10,35,50,24]); Gc=tf([10,5],[1,0]);
   H=tf([1],[0.01,1]); G_cl=feedback(G*Gc,H)
```

and it can be found that

$$G_{\rm cl}(s) = \frac{0.1s^5+10.75s^4+77.75s^3+278.6s^2+361.2s+120}{0.01s^6+1.1s^5+20.35s^4+110.5s^3+325.2s^2+384s+120}.$$

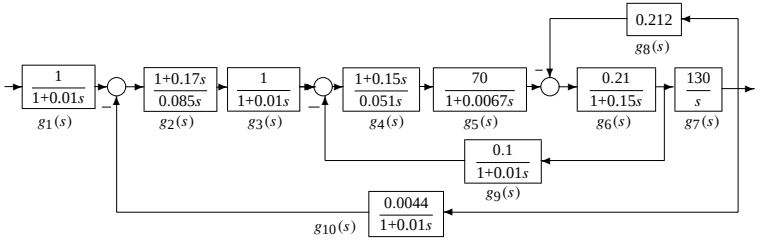


Figure 2.4. *An example of DC electric drive system.*

Unfortunately, the overall system evaluation is not always as simple as in this case, where one can perform the calculation by hand. For more complicated structures, evaluations by hand is laborious and computer aids are very useful.

Example 2.11. Consider the structure of the DC motor drive system, shown in Figure 2.4. It can be seen that the overall system model is not easily evaluated. From the block diagram, it can be seen that the difficulty lies in the interconnections among paths 6, 7, 8, and 9. Rearranging path 9 so that it starts from the output signal, we see that the equivalent transfer function then becomes $g_{91}(s) = g_9/g_7$. The overall system can then be constructed using the following MATLAB statements:

```
>> g1=tf(1,[0.01,1]); g2=tf([0.17,1],[0.085,0]); g3=g1;
   g4=tf([0.15,1],[0.051,0]); g5=tf(70,[0.0067,1]); g7=tf(130,[1,0]);
   g6=tf(0.21,[0.15,1]); g8=0.212; g9=tf(0.1,[0.01,1]); g91=g9/g7;
   g10=0.0044*g1; gg1=feedback(g7*g6,g8); %paths 6-8
   gg2=feedback(gg1*g5*g4,g91); %paths 4-9
   G=feedback(gg2*g3*g2,g10)*g1; minreal(zpk(G)), %overall system
```

where the overall model can be simply obtained as

$$G(s) = \frac{111852502194.908(s+6.667)(s+5.882)}{(s+180.9)(s+84.1)(s+48.2)(s^2+15.2s+74.3)(s^2+27.57s+354)}.$$

One can also perform symbolic manipulations to the models. When the following are given, the results follow immediately.

```
>> syms g1 g2 g3 g4 g5 g6 g7 g8 g9 g10
   g91=g9/g7; gg1=feedback(g7*g6,g8); gg2=feedback(gg1*g5*g4,g91);
   G=feedback(gg2*g3*g2,g10)*g1
```

thus the overall model can then be obtained as

$$G(s) = \frac{g_2g_3g_6g_7g_5g_4g_1}{1+g_7g_6g_8+g_6g_5g_4g_9+g_4g_5g_7g_6g_3g_2g_{10}}.$$

Example 2.12. Consider the motor drive system shown in Figure 2.5, where there are two inputs $r(t)$, $M(t)$, and one output $n(t)$.

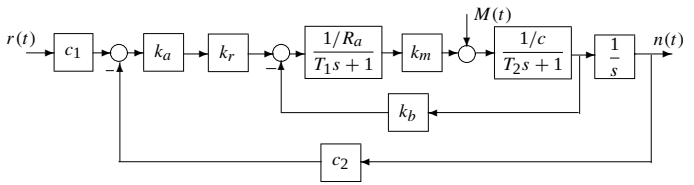


Figure 2.5. *Block diagram of a motor control system.*

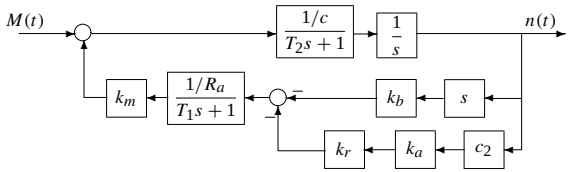


Figure 2.6. *Equivalent block diagram when $M(t)$ is applied alone.*

Let us first consider the modeling problem with set-point input $r(t)$ only. The overall system model can be obtained easily with the following statements:

```
>> syms Ka Kr c1 c2 c Ra T1 T2 Km Kb s  % symbolic declaration
   Ga=feedback(1/Ra/(T1*s+1)*Km*1/c/(T2*s+1),Kb); % inner loop
   g1=c1*feedback(Ka*Kr*Ga/s,c2); g1=collect(g1,s)
```

The transfer function is then derived from

$$g_1(s) = \frac{c_1 k_m k_a k_r}{R_a c T_1 T_2 s^3 + (R_a c T_1 + R_a c T_2)s^2 + (k_m k_b + R_a c)s + k_a k_r k_m c_2}.$$

If the load disturbance $M(t)$ is used alone, the original structure of the system can be rearranged as shown in Figure 2.6, and the following statements can be used to find the overall model

```
>> g2=feedback(1/c/(T2*s+1)/s, Km/Ra/(T1*s+1)*(Kb*s+c2*Ka*Kr));
   g2=collect(simplify(g2),s)
```

and it can be found that

$$g_2(s) = \frac{(T_1 s + 1)R_a}{c R_a T_2 T_1 s^3 + (c R_a T_1 + c R_a T_2)s^2 + (k_b k_m + c R_a)s + k_m c_2 k_a k_r}.$$

The transfer function matrix of the system is $\boldsymbol{G}(s) = [g_1(s), g_2(s)]$.

2.6 Conversion Between Different Model Objects

In the previous sections, three LTI model objects have been discussed. From the numerical point of view, the state space object is the most suitable one, especially for high-order

systems. In fact, each of the model formats can be converted into another, since all of them are equivalent. In this section, some of the typical model format conversions are discussed.

2.6.1 Conversion to Transfer Functions

Given a state space model $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \\ \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \end{cases} \tag{2.23}$$

with zero initial conditions, one can take the Laplace transform to give

$$\begin{cases} s\boldsymbol{I}\boldsymbol{X}(s) = \boldsymbol{A}\boldsymbol{X}(s) + \boldsymbol{B}\boldsymbol{U}(s), \\ \boldsymbol{Y}(s) = \boldsymbol{C}\boldsymbol{X}(s) + \boldsymbol{D}\boldsymbol{U}(s), \end{cases} \tag{2.24}$$

where $\boldsymbol{I}$ is the identity matrix, which has the same dimension as matrix $\boldsymbol{A}$. Thus, from the first formula of the above equation, one has

$$\boldsymbol{X}(s) = (s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}\boldsymbol{U}(s). \tag{2.25}$$

The equivalent transfer function model can then be obtained as

$$\boldsymbol{G}(s) = \boldsymbol{Y}(s)\boldsymbol{U}^{-1}(s) = \boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}. \tag{2.26}$$

In general, for MIMO systems, the transfer function matrix $\boldsymbol{G}(s)$ can also be evaluated from (2.26).

If the zero-pole-gain model of the system is given, one can expand the numerator and denominator polynomials expressed in a factorized form and then multiply the numerator by the gain to obtain the transfer function model.

In the Control Systems Toolbox, if an LTI object is given by $\boldsymbol{G}$, one can use the following command to get the equivalent transfer function object $\boldsymbol{G}_1$ by `G1=tf(G)`.

Example 2.13. Suppose that a system model is described by a state space model

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0 \\ 1 \\ 0 \\ -2 \end{bmatrix} u(t), \quad y(t) = [1, 0, 0, 0]\boldsymbol{x}(t).$$

Using the MATLAB statements

```
>> A=[0,1,0,0; 0,0,-1,0; 0,0,0,1; 0,0,5,0];
   B=[0;1;0;-2]; C=[1,0,0,0]; D=0; G=ss(A,B,C,D); G1=tf(G)
```

the transfer function model can be obtained as

$$G_1(s) = \frac{s^2 + 1.11 \times 10^{-15} s - 3}{s^4 - 5s^2}.$$

Note that the transfer function (matrix) transformed from a given state space model is unique.

Example 2.14. For a system given in zero-pole-gain form

$$G(s) = 6.8\frac{(s+3)(s+7)}{s(s+1.8\pm \text{j}1.63)(s+1)^2},$$

one can use the following MATLAB statements to get the transfer function model;

```
>> z=[-3; -7]; p=[0; -1.8+1.63j; -1.8-1.63j; -1; -1];
   K=6.8; G=zpk(z,p,K); G1=tf(G)
```

It follows that

$$G(s) = \frac{6.8s^2 + 68s + 142.8}{s^5 + 5.6s^4 + 14.1s^3 + 15.39s^2 + 5.897s},$$

and one can verify the results by hand.

2.6.2 Conversion to Zero-Pole-Gain Models

Having obtained the transfer function model, it is not a difficult task to get the equivalent zero-pole-gain model. One can easily solve this problem if one represents the numerator and denominator by their factorized forms. In the Control Systems Toolbox, one can use the function `G1=zpk(G)` to convert an LTI object $\boldsymbol{G}$ into its equivalent zero-pole-gain object $\boldsymbol{G}_1$.

Example 2.15. The state space model given in Example 2.13 can be converted into a zero-pole-gain model using the following MATLAB statements:

```
>> A=[0,1,0,0; 0,0,-1,0; 0,0,0,1; 0,0,5,0];
   B=[0;1;0;-2]; C=[1,0,0,0]; D=0; G=ss(A,B,C,D); G1=zpk(G)
```

where

$$G(s) = \frac{(s-1.732)(s+1.732)}{s^2(s-2.236)(s+2.236)}.$$

Example 2.16. If one has obtained the transfer function model, it is easy to find the equivalent zero-pole-gain model using MATLAB. For instance, the zero-pole-gain model of the transfer function of Example 2.14 can be found using the following MATLAB statements:

```
>> Z=[-3; -7]; P=[0; -1.8+1.63j; -1.8-1.63j; -1; -1];
   K=6.8; G=zpk(Z,P,K); G1=tf(G); G2=zpk(G1)
```

where

$$G_2(s) = \frac{6.8(s+7)(s+3)}{s(s+1)^2(s^2+3.6s+5.897)}.$$

If there are complex poles or zeros, a second-order polynomial will be used to represent the complex conjugates.

For an MIMO state space model, the zeros of the system cannot be easily obtained. However, one can rely on the MATLAB function `z=tzero(G)` provided in the Control Systems Toolbox to find the transmission zeros $\boldsymbol{z}$ of the system $\boldsymbol{G}$; see [5].

Example 2.17. Consider the state space expression for a two input–two output system given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 2.25 & -5 & -1.25 & -0.5 \\ 2.25 & -4.25 & -1.25 & -0.25 \\ 0.25 & -0.5 & -1.25 & -1 \\ 1.25 & -1.75 & -0.25 & -0.75 \end{bmatrix}\boldsymbol{x} + \begin{bmatrix} 4 & 6 \\ 2 & 4 \\ 2 & 2 \\ 0 & 2 \end{bmatrix}\boldsymbol{u}, \quad \boldsymbol{y} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \end{bmatrix}\boldsymbol{x}.$$

The transmission zeros can be easily obtained using

```
>> A=[2.25, -5, -1.25, -0.5; 2.25, -4.25, -1.25, -0.25;
      0.25, -0.5, -1.25,-1; 1.25, -1.75, -0.25, -0.75];
   B=[4, 6; 2, 4; 2, 2; 0, 2]; C=[0, 0, 0, 1; 0, 2, 0, 2];
   D=zeros(2,2); G=ss(A,B,C,D); Z=tzero(G)
```

and the transmission zeros are $z_{1,2} = -0.6250 \pm \text{j}\ 0.7806$.

2.6.3 State Space Realizations

Although, from a given state space model, the unique transfer function (matrix) can be obtained, the inverse transformation, i.e., finding a state space expression or realization from the given transfer function, is not unique. It has been shown through the RLC example in Sec. 2.1 that the state space expression can be different if the state variables are selected differently. The transformation process from a given transfer function to a state space expression is referred to as a state space realization of the transfer function. It is equivalent to saying that a given transfer function model may have an infinite number of different state space realizations. The commonly used state space realization of an LTI model $\boldsymbol{G}$ can be obtained from `G1 = ss(G)`.

Example 2.18. Consider an SISO transfer function

$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24}.$$

Using the MATLAB statements

```
>> num=[1,7,24,24]; den=[1,10,35,50,24]; G=tf(num,den); G1=ss(G)
```

the state space model of the system can be obtained:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} -10 & -4.375 & -3.125 & -1.5 \\ 8 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\boldsymbol{x}(t) + \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}u(t), \\ y(t) = [0.5,\ 0.4375,\ 0.75,\ 0.75]\boldsymbol{x}(t). \end{cases}$$

Example 2.19. An MIMO transfer function matrix can also be converted into a state space model using the same `ss()` function. Consider the MIMO transfer function matrix

$$\boldsymbol{G}(s)=\begin{bmatrix} 1/(s+1) & 0 & (s-1)/[(s+1)(s+2)] \\ -1/(s-1) & 1/(s+2) & 1/(s+2) \end{bmatrix}.$$

Using the MATLAB statements

```
>> s=tf('s'); h11=tf(1,[1,1]); h12=0; h13=(s-1)/(s+1)/(s+2);
   h21=tf(-1,[1,-1]); h22=tf(1,[1,2]); h23=tf(1,[1,2]);
   H=[h11,h12,h13; h21,h22,h23]; G=ss(H)
```

one can get the state space model

$$\begin{cases} \dot{\boldsymbol{x}}(t)=\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3 & -2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 \end{bmatrix}\boldsymbol{x}(t)+\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\boldsymbol{u}(t), \\ \boldsymbol{y}(t)=\begin{bmatrix} 1 & 0 & 0 & 0.5 & -0.5 & 0 \\ 0 & -1 & 1 & 0 & 0 & 1 \end{bmatrix}\boldsymbol{x}(t). \end{cases}$$

Similarity transformation of state space models

Since the selection of state variables can be different, the realization of a given transfer function model can also be different.

Definition 2.3. Suppose that there exists a nonsingular matrix $\boldsymbol{T}$. Define a new state variable vector z such that $\boldsymbol{z}=\boldsymbol{T}\boldsymbol{x}$. The new state space expression in vector z can be written as

$$\begin{cases} \dot{z}=\boldsymbol{A}_t z+\boldsymbol{B}_t\boldsymbol{u}, \\ \boldsymbol{y}=\boldsymbol{C}_t z+\boldsymbol{D}\boldsymbol{u}, \quad z(0)=\boldsymbol{T}\boldsymbol{x}(0), \end{cases} \tag{2.27}$$

where $\boldsymbol{A}_t=\boldsymbol{T}\boldsymbol{A}\boldsymbol{T}^{-1}$, $\boldsymbol{B}_t=\boldsymbol{T}\boldsymbol{B}$, $\boldsymbol{C}_t=\boldsymbol{C}\boldsymbol{T}^{-1}$.

The transformation under matrix $\boldsymbol{T}$ is referred to as a similarity transformation.

The MATLAB function `ss2ss()` is provided in the Control Systems Toolbox to perform a similarity transformation of state space models. The syntax of the function is `G1 = ss2ss(G,T)`, where $\boldsymbol{G}$ is the original state space object, and $\boldsymbol{T}$ is the similarity transformation matrix. The transformed state space object under $\boldsymbol{T}$ is returned in $\boldsymbol{G}_1$.

Controllable canonical form

Suppose that the transfer function model is given as in (2.12). The controllable canonical form can then be written as

$$\begin{cases} \dot{\boldsymbol{x}}=\boldsymbol{A}_c\boldsymbol{x}+\boldsymbol{B}_c\boldsymbol{u} \\ \boldsymbol{y}=\boldsymbol{C}_c\boldsymbol{x}+\boldsymbol{D}\boldsymbol{u} \end{cases} \Longrightarrow \begin{cases} \dot{\boldsymbol{x}}=\begin{bmatrix} 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -a_1 & -a_2 & \cdots & -a_n \end{bmatrix}\boldsymbol{x}+\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}\boldsymbol{u}, \\ \boldsymbol{y}=[b_1,\ b_2,\ \ldots,\ b_n]\boldsymbol{x}. \end{cases} \tag{2.28}$$

Observable canonical form

The observable canonical form of (2.12) is

$$\begin{cases} \dot{\boldsymbol{x}}=\boldsymbol{A}_o\boldsymbol{x}+\boldsymbol{B}_o\boldsymbol{u} \\ \boldsymbol{y}=\boldsymbol{C}_o\boldsymbol{x}+\boldsymbol{D}\boldsymbol{u} \end{cases} \Longrightarrow \begin{cases} \dot{\boldsymbol{x}}=\begin{bmatrix} 0 & 0 & \cdots & 0 & -a_1 \\ 1 & 0 & \cdots & 0 & -a_2 \\ 0 & 1 & \cdots & 0 & -a_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_n \end{bmatrix}\boldsymbol{x}+\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}\boldsymbol{u}, \\ \boldsymbol{y}=[0,\ 0,\ \ldots,\ 1]\boldsymbol{x}. \end{cases} \tag{2.29}$$

It can be seen that the controllable and observable canonical forms are dual. That is,

$$\boldsymbol{A}_c=\boldsymbol{A}_o^{\mathrm{T}}, \quad \boldsymbol{B}_c=\boldsymbol{C}_o^{\mathrm{T}}, \quad \boldsymbol{C}_c=\boldsymbol{B}_o^{\mathrm{T}}, \tag{2.30}$$

where $(\boldsymbol{A}_c, \boldsymbol{B}_c, \boldsymbol{C}_c, \boldsymbol{D})$ denotes the state space model of the controllable canonical realization, and $(\boldsymbol{A}_o, \boldsymbol{B}_o, \boldsymbol{C}_o, \boldsymbol{D})$ denotes the realization of the observable canonical form.

Jordanian canonical form

Assume that the eigenvalues of the matrix $\boldsymbol{A}$ are $\lambda_1, \lambda_2, \ldots, \lambda_n$ and its ith eigenvector corresponding to the ith eigenvalue λ_i is denoted by $\boldsymbol{v}_i$ such that

$$\boldsymbol{A}\boldsymbol{v}_i=\lambda_i\boldsymbol{v}_i, \quad i=1,2,\ldots,n. \tag{2.31}$$

The modal matrix $\boldsymbol{\Lambda}$ of $\boldsymbol{A}$ is defined as

$$\boldsymbol{\Lambda}=\boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T}=\begin{bmatrix} \boldsymbol{J}_1 & & & \\ & \boldsymbol{J}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{J}_k \end{bmatrix}, \tag{2.32}$$

where the $\boldsymbol{J}_i$'s are referred to as the Jordanian matrices. Suppose that there exists a transformation matrix $\boldsymbol{T}_c$ such that the given state space model can be transformed into a controllable canonical form; then a transformation matrix $\boldsymbol{T}$ can be constructed as $\boldsymbol{T}=\boldsymbol{U}\boldsymbol{T}_c$ such that a modal realization can be obtained, where $\boldsymbol{U}=[\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_k]$. The following two cases

are considered for a Jordanian canonical form:

1. If $\lambda_{i,i+1}$ is a complex conjugate pair, such that $\lambda_{i,i+1} = -\sigma_i \pm \mathrm{j}\omega_i$, the Jordanian block takes the form

$$\boldsymbol{J}_i = \begin{bmatrix} \sigma_i & \omega_i \\ -\omega_i & \sigma_i \end{bmatrix}, \quad \boldsymbol{U}_i = \begin{bmatrix} 1 & 0 \\ \sigma_i & \omega_i \\ \vdots & \vdots \\ \mathrm{Re}[\lambda_i^{n-1}] & \mathrm{Im}[\lambda_i^{n-1}] \end{bmatrix}. \tag{2.33}$$

2. If λ_i is a real eigenvalue with multiplicity of m_i, the Jordanian block $\boldsymbol{J}_i$ is

$$\boldsymbol{J}_i = \begin{bmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_i \end{bmatrix} \tag{2.34}$$

and the transformation matrix block $\boldsymbol{U}_i$ is

$$\boldsymbol{U}_i = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \lambda_i & 1 & 0 & \cdots & 0 \\ \lambda_i^2 & 2\lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda_i^{n-1} & \dfrac{\mathrm{d}}{\mathrm{d}\lambda_i}(\lambda_i^{n-1}) & \dfrac{1}{2!}\dfrac{\mathrm{d}^2}{\mathrm{d}\lambda_i^2}(\lambda_i^{n-1}) & \cdots & \dfrac{1}{(m_i-1)!}\dfrac{\mathrm{d}^{m_i-1}}{\mathrm{d}\lambda_i^{m_i-1}}(\lambda_i^{n-1}) \end{bmatrix}. \tag{2.35}$$

A MATLAB function `canon()` is provided in the Control Systems Toolbox with syntax `[G1,T]=canon(G,type)`, where $\boldsymbol{G}$ is the state space object of the original system, and $\boldsymbol{G}_1$ is the state space object obtained after conversion. The argument `type` can be either `'companion'` (for the companion form of realization) or `'modal'` (for the modal form, i.e., the Jordanian realization). The transformation matrix is returned in $\boldsymbol{T}$.

Example 2.20. Consider a system model given by

$$G(s) = \frac{3s^2 + 21s + 36}{s^4 + 5s^3 + 10s^2 + 10s + 4}.$$

Using the MATLAB statements

```
>> G=tf([3 21 36],[1 5 10 10 4]); G1=canon(G,'modal')
```

the Jordanian realization of the system can be obtained:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} -2 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} -40 \\ 0 \\ 40.988 \\ 46 \end{bmatrix} u(t), \\ y(t) = [0.075\ 0\ -0.366\ 0.3913]\boldsymbol{x}(t). \end{cases}$$

Example 2.21. Consider the transfer function model given by

$$G(s) = \frac{6s^2 + 30s + 36}{s^5 + 12s^4 + 51s^3 + 98s^2 + 98s + 40}.$$

The model can be entered into the MATLAB environment and the required realizations of the system can also be obtained:

```
>> G=tf([6,30,36],[1,12,51,98,98,40]); G=ss(G);
   [G1,T1]=canon(G,'modal'), [G2,T2]=canon(G,'companion')
```

It can be found that the Jordanian form can be written as

$$\boldsymbol{A}_1 = \begin{bmatrix} -5 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}, \ \boldsymbol{B}_1 = \begin{bmatrix} -23.049 \\ -31.062 \\ 3.8582 \\ 17.362 \\ -15.334 \end{bmatrix}, \ \boldsymbol{C}_1^{\mathrm{T}} = \begin{bmatrix} -0.022969 \\ 0.012878 \\ 0.036591 \\ -0.073182 \\ -0.065216 \end{bmatrix}$$

with the conversion matrix $\boldsymbol{T}_1$,

$$\boldsymbol{T}_1 = \begin{bmatrix} -23.049 & -20.168 & -11.525 & -3.2413 & -1.4406 \\ -31.062 & -31.062 & -18.443 & -5.3387 & -2.4267 \\ 3.8582 & 7.4754 & 10.128 & 5.4106 & 3.3157 \\ 17.362 & 23.391 & 19.954 & 5.7723 & 2.11 \\ -15.334 & -21.084 & -19.167 & -6.948 & -4.7917 \end{bmatrix}.$$

The companion canonical form of the system is

$$\boldsymbol{A}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & -40 \\ 1 & 0 & 0 & 0 & -98 \\ 0 & 1 & 0 & 0 & -98 \\ 0 & 0 & 1 & 0 & -51 \\ 0 & 0 & 0 & 1 & -12 \end{bmatrix}, \ \boldsymbol{B}_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \ \boldsymbol{C}_2^{\mathrm{T}} = \begin{bmatrix} 0 \\ 0 \\ 6 \\ -42 \\ 234 \end{bmatrix}$$

with the conversion matrix $\boldsymbol{T}_2$,

$$\boldsymbol{T}_2 = \begin{bmatrix} 1 & 1.5 & 1.5938 & 0.76563 & 0.76563 \\ 0 & 0.125 & 0.375 & 0.39844 & 0.76563 \\ 0 & 0 & 0.03125 & 0.09375 & 0.39844 \\ 0 & 0 & 0 & 0.0078125 & 0.09375 \\ 0 & 0 & 0 & 0 & 0.0078125 \end{bmatrix}.$$

Balanced realization

Before discussing the balanced realization problem, we consider the following illustrative system [30]:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 10^{-6} \\ 10^{6} \end{bmatrix} u, \quad y(t) = [10^6\ \ 10^{-6}] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

It can be seen that the two elements in the $\boldsymbol{B}$ vector and those of the corresponding values in the $\boldsymbol{C}$ vector are significantly different. If a new pair of state variables $z_1 = 10^6 x_1$ and $z_2 = 10^{-6} x_2$ are selected, the system can be transformed into

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u, \quad y(t) = [1 \;\; 1] \begin{bmatrix} z_1 \\ z_2 \end{bmatrix},$$

where the elements in the new $\boldsymbol{B}$ and $\boldsymbol{C}$ vectors are of the same numeric order. One can observe here that the introduced transformation matrix rescales the coordinates of the system to form a set of new coordinates which look more balanced.

A MATLAB function `balreal()` is provided in the Control Systems Toolbox and can be used to perform the balanced realization of a given stable state space model. The actual algorithm for doing this transformation is given in the next chapter. The function `[G1,g,T] = balreal(G)` can be used to find the balanced realized state space object $\boldsymbol{G}_1$, where the existing state space object is given by $\boldsymbol{G}$. The transformation matrix is returned in $\boldsymbol{T}$ and the diagonals of the Gramians, which will be defined in the next chapter, of the new system will be returned in vector $\boldsymbol{g}$.

Example 2.22. Consider again the system model given in Example 2.18, using the following MATLAB statements:

```
>> num=[1,7,24,24]; den=[1,10,35,50,24]; G=tf(num,den);
   [G1,Sig,T]=balreal(ss(G));
```

the balanced realization of the system can be obtained as

$$\begin{cases} \dot{z}(t) = \begin{bmatrix} -0.81996 & -0.31463 & 0.73015 & 0.07656 \\ 0.31463 & -0.44795 & 3.7879 & 0.23645 \\ 0.73015 & -3.7879 & -7.109 & -1.3934 \\ 0.07656 & -0.23645 & -1.3934 & -1.6231 \end{bmatrix} z(t) + \begin{bmatrix} 0.92156 \\ -0.16627 \\ -0.42015 \\ -0.04307 \end{bmatrix} u(t), \\ y(t) = [0.9216, \; 0.1663, \; -0.4201, \; -0.0431] z(t). \end{cases}$$

It can be seen that in the balanced realization of an SISO system, the absolute values of the corresponding elements in the $\boldsymbol{B}$ and $\boldsymbol{C}$ vectors are the same. But for MIMO cases the above argument may not be true [5].

Minimum realization

It has been shown in Example 2.8 that the order of the overall model obtained by a parallel connection of blocks may be higher than the actual order of the system. In the real world, the state space model established using other methods may also produce models with an order higher than necessary or higher than the minimum.

This leads to the following question: What is the lowest possible order for a given system? This is the problem of finding the minimum realization.

For an SISO transfer function or a zero-pole-gain representation, the minimum realization solution is very simple and straightforward. If the poles and zeros at the same locations can be cancelled out (also called a pole-zero cancellation), the minimum realized model can be obtained immediately.

The situation with the state space expression is not so straightforward. Fortunately, a MATLAB function `minreal()` provided in the Control Systems Toolbox can be used directly for solving the minimum realization problem. The syntax of the function is

`G1=minreal(G)`,

where the original LTI object is given by $\boldsymbol{G}$ and the minimum realized one is given in the object $\boldsymbol{G}_1$.

Example 2.23. Consider a fourth-order state space model given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} -5 & 8 & 0 & 0 \\ -4 & 7 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & -2 & 6 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 4 \\ -2 \\ 2 \\ 1 \end{bmatrix} u, \quad y = [2 \;\; -2 \;\; -2 \;\; 2] \boldsymbol{x}.$$

Using the MATLAB statements

```
>> A=[-5,8,0,0; -4,7,0,0; 0,0,0,4; 0,0,-2,6]; B=[4; -2; 2; 1];
   C=[2,-2,-2,2]; D=0; G=ss(A,B,C,D); Gm=minreal(G)
```

we are prompted that "2 states removed," and the minimum realized model can easily be obtained as

$$\begin{cases} \dot{z}(t) = \begin{bmatrix} -1 & -4.4409 \times 10^{-16} \\ 5.3291 \times 10^{-15} & 2 \end{bmatrix} z(t) + \begin{bmatrix} 4.2426 \\ 2.2361 \end{bmatrix} u(t), \\ y(t) = [2.8284, -0.89443] z(t). \end{cases}$$

Note that MATLAB may return very small numbers instead of the actual value of zero after numerical operations. In fact, the zero-pole-gain model of the given fourth-order state space model can be obtained with the `zpk(G)` command as

$$G(s) = \frac{10(s-2.6)(s-3)(s-4)}{(s+1)(s-2)(s-3)(s-4)}.$$

It can be seen that there are common pole-zero pairs at $s = 3$ and $s = 4$. Canceling out these two pairs yields a transfer function of

$$\widehat{G}(s) = \frac{10(s-2.6)}{(s+1)(s-2)},$$

which is the minimum realization of the system.

2.6.4 Conversion Between Continuous and Discrete-Time Models

If a model is described with a continuous LTI object $\boldsymbol{G}$, its discretized version under the sampling interval T can be easily obtained with the function call `Gd=c2d(G,T)`. The default discretization method used is the zero-order-hold (ZOH) method. If one wants to use Tustin's method, the function can be called with `Gd=c2d(G,T,'Tustin')`. Of course one can even use more conversion algorithms and details prompted by the `help c2d` command.

If, on the other hand, a discrete-time object $\boldsymbol{G}_d$ is known, the continuous version of it can be obtained with `G=d2c(Gd)`, where no sampling interval T is necessary, since the information is already contained in the object $\boldsymbol{G}_d$.

Example 2.24. Consider again the multivariable system shown in Example 2.17. If one chooses the sampling interval of $T = 0.1$ second, the equivalent discrete-time version of the system can be found with the statements

```
>> A=[2.25, -5, -1.25, -0.5; 2.25, -4.25, -1.25, -0.25;
      0.25, -0.5, -1.25,-1; 1.25, -1.75, -0.25, -0.75];
   B=[4, 6; 2, 4; 2, 2; 0, 2]; C=[0, 0, 0, 1; 0, 2, 0, 2];
   D=zeros(2,2); G=ss(A,B,C,D); Gd=c2d(G,0.1)
```

from which it is found that the matrices in the discrete-time system are

$$\boldsymbol{F}=\begin{bmatrix} 1.1915 & -0.4455 & -0.1013 & -0.04215 \\ 0.2008 & 0.6124 & -0.1058 & -0.01884 \\ 0.01526 & -0.03499 & 0.8849 & -0.09054 \\ 0.1147 & -0.1622 & -0.01973 & 0.9279 \end{bmatrix}, \quad \boldsymbol{G}=\begin{bmatrix} 0.383253 & 0.5527 \\ 0.1906 & 0.3694 \\ 0.1879 & 0.1764 \\ 0.004833 & 0.1927 \end{bmatrix}.$$

Example 2.25. If the continuous model is given by

$$G(s)=\frac{1}{(s+2)^3}\mathrm{e}^{-2s},$$

and the sampling interval is $T = 0.1$ second, the following statements can be used to discretize the system:

```
>> s=tf('s'); G=1/(s+2)^3; G.ioDelay=2
```

If the ZOH and Tustin algorithms are used, one may use the statements

```
>> G1=c2d(G,0.1)  % ZOH method
   G2=c2d(G,0.1,'Tustin')  % Tustin algorithm
```

and it can be found that

$$G_{ZOH}(z)=\frac{0.0001436z^2+0.0004946z+0.0001064}{z^3-2.456z^2+2.011z-0.5488}z^{-20},$$

$$G_{Tustin}(z)=\frac{9.391\times10^{-5}z^3+0.0002817z^2+0.0002817z+9.391\times10^{-5}}{z^3-2.455z^2+2.008z-0.5477}z^{-20}.$$

If inverse conversions are used, i.e.,

```
>> G1c=d2c(G1), G2c=d2c(G2)
```

it can be found that

$$G_{1c}(s)=\frac{1.736\times10^{-17}s^2-5.561\times10^{-17}s+1}{s^3+6s^2+12s+8}\mathrm{e}^{-2s},$$

$$G_{2c}(s)=\frac{9.391\times10^{-5}s^3+0.003096s^2+0.04542s+1.01}{s^3+6.02s^2+12.08s+8.081}\mathrm{e}^{-2s}.$$

2.7 An Introduction to System Identification

So far, the previous descriptions regarding linear control systems all assume that the system model has been given. In real applications, not all the plant models can be derived with existing physical laws. The internal structure of the plant may not even be known at all. Thus, reconstructing the system model from the measured data, is referred to as system identification.

System identification is a general term used to describe mathematical tools and algorithms that build dynamical models from measured data. A dynamical model in this context is a mathematical description of the dynamic behavior of a system or process.

In real applications, many directly measured data are useful in identifying the model of the system, for instance, frequency response data, and input and output signals. In this section, we focus on the identification of discrete-time transfer functions from the measured input and output signals.

2.7.1 Identification of Discrete-Time Systems

A typical discrete-time transfer function is usually given by

$$G\left(z^{-1}\right)=\frac{b_1+b_2z^{-1}+\cdots+b_mz^{-m+1}}{1+a_1z^{-1}+a_2z^{-2}+\cdots+a_nz^{-n}}z^{-d} \tag{2.36}$$

and it corresponds to the difference equation

$$\begin{aligned} &y(t)+a_1y(t-1)+a_2y(t-2)+\cdots+a_ny(t-n) \\ &\quad=b_1u(t-d)+b_2u(t-d-1)+\cdots+b_mu(t-d-m+1)+\varepsilon(t) \end{aligned} \tag{2.37}$$

where $\varepsilon(t)$ can be regarded as the identification residuals. Here the shorthand notation $y(t)$ is used for the output signal $y(kT)$, and $y(t-1)$ can then be used to describe the output at the previous sample, i.e., $y[(k-1)T]$. Suppose that a set of input and output signals has been measured and written as $\boldsymbol{u}=[u(1),u(2),\ldots,u(M)]^{\mathbf{T}}$, $\boldsymbol{y}=[y(1),y(2),\ldots,y(M)]^{\mathbf{T}}$. From (2.37), it can be found that

$$\begin{aligned} y(1)&=-a_1y(0)-\cdots-a_ny(1-n)+b_1u(1-d)+\cdots+b_mu(2-m-d)+\varepsilon(1) \\ y(2)&=-a_1y(1)-\cdots-a_ny(2-n)+b_1u(2-d)+\cdots+b_mu(3-m-d)+\varepsilon(2) \\ &\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ y(M)&=-a_1y(M-1)-\cdots-a_ny(M-n)+b_1u(M-d) \\ &\qquad\qquad +\cdots+b_mu(M+1-m-d)+\varepsilon(M) \end{aligned}$$

where $y(t)$ and $u(t)$ are assumed to be zero when $t \leq 0$. The matrix form of the above equations can be written as

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{\theta} + \boldsymbol{\varepsilon}, \tag{2.38}$$

where

$$\boldsymbol{\Phi} = \begin{bmatrix} y(0) & \cdots & y(1-n) & u(1-d) & \cdots & u(2-m-d) \\ y(1) & \cdots & y(2-n) & u(2-d) & \cdots & u(3-m-d) \\ \vdots & & \vdots & \vdots & & \vdots \\ y(M-1) & \cdots & y(M-n) & u(M-d) & \cdots & u(M+1-m-d) \end{bmatrix} \tag{2.39}$$

$$\boldsymbol{\theta}^{\mathbf{T}} = [-a_1, -a_2, \ldots, -a_n, b_1, \ldots, b_m], \quad \boldsymbol{\varepsilon}^{\mathbf{T}} = [\varepsilon(1), \ldots, \varepsilon(M)]. \tag{2.40}$$

To minimize the sum of squared residuals, i.e.,

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{M} \varepsilon^2(i),$$

the optimum estimation to the undetermined elements in $\boldsymbol{\theta}$ can be written as

$$\boldsymbol{\theta} = [\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}]^{-1}\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{y}. \tag{2.41}$$

Since the sum of squared residuals is minimized, the method is also known as the least squares algorithm [31]. Note that $\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}$ might be ill-conditioned if the input excitation signal $u(t)$ is not properly designed for the identification experiments. This input signal design issue will be discussed and illustrated in Sec. 2.7.3.

A function `arx()` is provided in the System Identification Toolbox to identify the discrete-time model from measured input and output data. If the measured input and output signals are expressed by column vectors $\boldsymbol{u}$ and $\boldsymbol{y}$, and the orders of the numerator and denominator are assumed to be $m-1$ and n, respectively, and the delay term is d, the following statement can be used: `H=arx([y,u], [n,m,d])`.

The returned variable H is an `idpoly` object, where $H.\boldsymbol{A}$ and $H.\boldsymbol{B}$ represent the numerator and denominator polynomials of the identified system, respectively.

Example 2.26. Assume that the measured input and output data are given as in Table 2.1. One may assume that the order of the numerator and denominator is selected as 4, with a delay of 1; then the following statements can be used to identify the system model:

```
>> u=[1.4601,0.8849,1.1854,1.0887,1.413,1.3096,1.0651,0.7148,...
      1.3571,1.0557,1.1923,1.3335,1.4374,1.2905,0.841,1.0245,...
      1.4483,1.4335,1.0282,1.4149,0.7463,0.9822,1.3505,0.7078,...
      0.8111,0.8622,0.8589,1.183,0.9177,0.859,0.7122,1.2974,...
      1.056,1.4454,1.0727,1.0349,1.3769,1.1201,0.8621,1.2377,...
      1.3704,0.7157,1.245,1.0035,1.3654,1.1022,1.2675,1.0431]';
```


Table 2.1. *Measured input and output data.*

t	$u(t)$	$y(t)$	t	$u(t)$	$y(t)$	t	$u(t)$	$y(t)$
0	1.4601	0	1.6	1.4483	16.411	3.2	1.056	11.871
0.1	0.8849	0	1.7	1.4335	14.336	3.3	1.4454	13.857
0.2	1.1854	8.7606	1.8	1.0282	15.746	3.4	1.0727	14.694
0.3	1.0887	13.194	1.9	1.4149	18.118	3.5	1.0349	17.866
0.4	1.413	17.41	2	0.7463	17.784	3.6	1.3769	17.654
0.5	1.3096	17.636	2.1	0.9822	18.81	3.7	1.1201	16.639
0.6	1.0651	18.763	2.2	1.3505	15.309	3.8	0.8621	17.107
0.7	0.7148	18.53	2.3	0.7078	13.7	3.9	1.2377	16.537
0.8	1.3571	17.041	2.4	0.8111	14.818	4	1.3704	14.643
0.9	1.0557	13.415	2.5	0.8622	13.235	4.1	0.7157	15.086
1	1.1923	14.454	2.6	0.8589	12.299	4.2	1.245	16.806
1.1	1.3335	14.59	2.7	1.183	11.6	4.3	1.0035	14.764
1.2	1.4374	16.11	2.8	0.9177	11.607	4.4	1.3654	15.498
1.3	1.2905	17.685	2.9	0.859	13.766	4.5	1.1022	14.679
1.4	0.841	19.498	3	0.7122	14.195	4.6	1.2675	16.655
1.5	1.0245	19.593	3.1	1.2974	13.763	4.7	1.0431	16.63

```
y=[0,0,8.7606,13.1939,17.41,17.6361,18.7627,18.5296,17.0414,...
   13.4154,14.4539,14.59,16.1104,17.6853,19.4981,19.5935,...
   16.4106,14.3359,15.7463,18.1179,17.784,18.8104,15.3086,...
   13.7004,14.8178,13.2354,12.2993,11.6001,11.6074,13.7662,...
   14.195,13.763,11.8713,13.8566,14.6944,17.8659,17.6543,...
   16.6386,17.1071,16.5373,14.643,15.0862,16.8058,14.7641,...
   15.4976,14.679,16.6552,16.6301]';
t1=arx([y,u],[4,4,1])
```

The following results are obtained and displayed:

```
Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + e(t)
A(q) = 1 - q^-1 + 0.25 q^-2 + 0.25 q^-3 - 0.125 q^-4
B(q) = 4.83e-008 q^-1 + 6 q^-2 - 0.5999 q^-3 - 0.1196 q^-4
Estimated using ARX
Loss function 7.09262e-010 and FPE 9.92966e-010
Sampling interval: 1
```

From the displayed information, the identified model can be written as

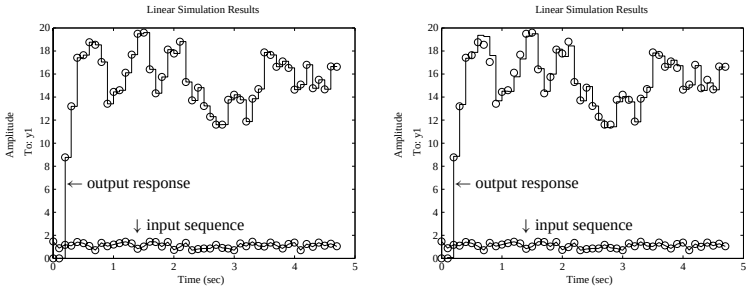
$$G\left(z^{-1}\right) = \frac{4.83 \times 10^{-8} z^{-1} + 6z^{-2} - 0.5999z^{-3} - 0.1196z^{-4}}{1 - z^{-1} + 0.25z^{-2} + 0.25z^{-3} - 0.125z^{-4}},$$

i.e.,

$$G(z) = \frac{4.83 \times 10^{-8} z^3 + 6z^2 - 0.5999z - 0.1196}{z^4 - z^3 + 0.25z^2 + 0.25z - 0.125}.$$

In fact, the data were generated from the system in Example 2.5. It can be seen that the model identified is rather close to the original model. Also, the sampling interval can be found from Table 2.1, where $T = 0.1$ second. A formal identification method is to establish the data object U with `U=iddata(y,u,T)`. Then the following statements can be used to identify the system model:

```
>> U=iddata(y,u,0.1); T=arx(U,[4,4,1]); H=tf(T); G=H(1)
```



(a) with $m = 4, n = 4, d = 1$ (b) $m = 3, n = 3, d = 1$

Figure 2.7. *Comparisons of identification results.*

It can then be found that

$$G(z) = \frac{4.83 \times 10^{-8} z^3 + 6z^2 - 0.5999z - 0.1196}{z^4 - z^3 + 0.25z^2 + 0.25z - 0.125}.$$

The transfer function model converted from the `tf()` function is in fact a double input transfer function matrix. The first one is the expected transfer function model, and the second is the transfer function from error signal $\varepsilon(k)$ to the output signal. This model is discarded in the example.

To verify the identified model, the MATLAB function `lsim()` can be used to simulate the system driven by the given $\boldsymbol{u}$ sequence. Details of the function will be given later in Sec. 3.3.3. The response is shown in Figure 2.7(a), superimposed as open circles by the measured output sequence $\boldsymbol{y}$. It can be seen that the identified model is very accurate:

```
>> t=0:0.1:4.7; lsim(G,u,t); hold on, plot(t,y,'o',t,u,'o')
```

If the orders are improperly selected as $m = 3, n = 3, d = 1$, the identified model is then obtained as

$$G_1(z) = \frac{0.04886z^2 + 6.017z + 2.806}{z^3 - 0.4362z^2 - 0.214z + 0.2828},$$

and the verification shown in Figure 2.7(b) illustrates that the fitting of the model is not so good. Thus, the selection of the orders is also very crucial in the identification process:

```
>> T=arx(U,[3,3,1]); H=tf(T); G1=H(1)
   lsim(G1,u,t); hold on, plot(t,y,'o',t,u,'o')
```

The identification can be completed from (2.39) and (2.41) without the use of the `arx()` function. The following statements can be used to solve the same problem:

```
>> Phi=[[0;y(1:end-1)] [0;0;y(1:end-2)],...
        [0;0;0; y(1:end-3)] [0;0;0;0;y(1:end-4)],...
        [0;u(1:end-1)] [0;0;u(1:end-2)],...
        [0;0;0; u(1:end-3)] [0;0;0;0;u(1:end-4)]]
   T=Phi\y; Gd=tf(ans(5:8),[1,-ans(1:4)],'Ts',0.1)
```

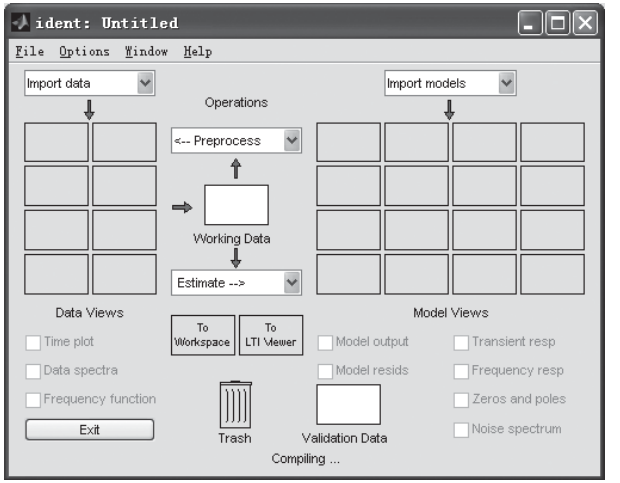



Figure 2.8. *GUI for system identification.*

The identified model is

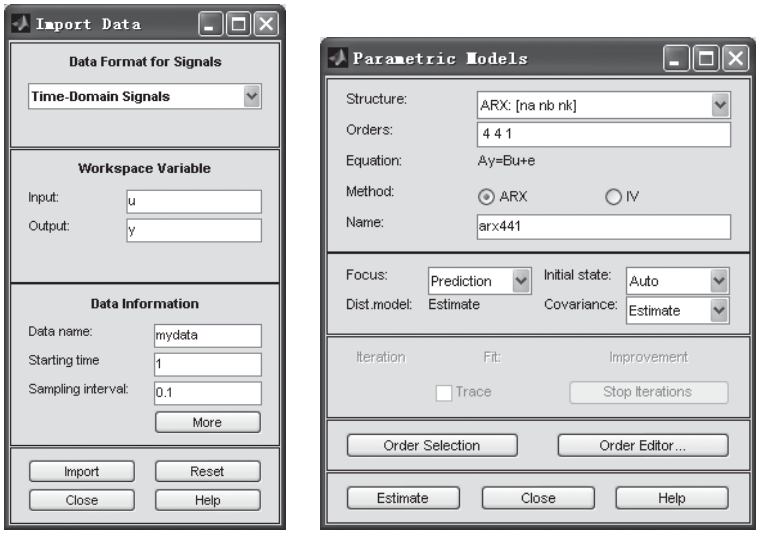
$$G(z) = \frac{-5.824 \times 10^{-7} z^3 + 6z^2 - 0.5999z - 0.1196}{z^4 - z^3 + 0.25z^2 + 0.25z - 0.125}.$$

A GUI `ident` is provided in the System Identification Toolbox, which can be used to identify discrete-time models in a visual way. If one types `ident` command, an interface, as shown in Figure 2.8, can be displayed.

To identify a system model, one should first provide the relevant data to the interface. This can be done by clicking the upper left Import Data list box. Select menu item Time-Domain Data. Then, a dialog box pops up, as shown in Figure 2.9(a), and the input and output data can be entered into the interface by filling them into the Input and Output columns, respectively. The sampling interval should also be filled in. Click the Import button to complete data input.

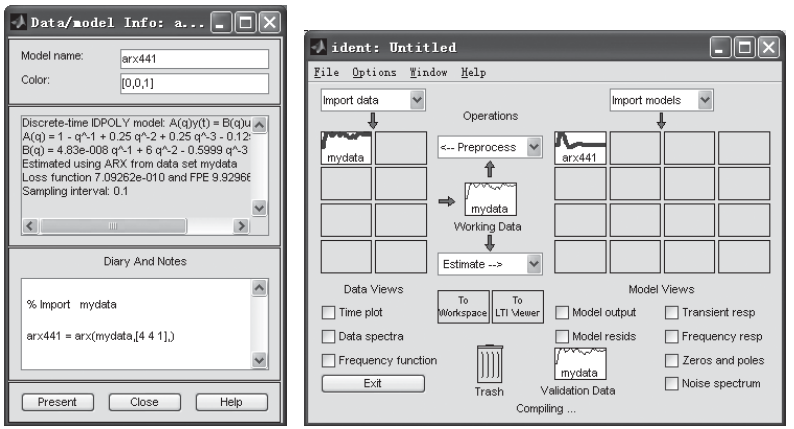
If one wants to identify the autoregressive exogenous (ARX) model, the Parametric Models item in the Estimate list box should be selected, and the dialog box shown in Figure 2.9(b) will be displayed. The expected orders of the system can be specified. Then, click the Estimate button to initiate the identification process. When the identification process is completed, the dialog box shown in Figure 2.10(a) will show the identification results. It can be seen that the identification results obtained in the interface are exactly the same as the result obtained using `arx()` function.

The final interface is shown in Figure 2.10(b). The user may further select other tasks for the analysis of the identified model.



(a) data input dialog box (b) order selection dialog box

Figure 2.9. *Dialog boxes for system identification.*



(a) identification results (b) identification solutions

Figure 2.10. *Dialog boxes for system identification.*

2.7.2 Order Selection

The Akaike information criterion (AIC) is a statistical model fit measure defined by [31, 32]

$$\text{AIC} = \lg\left\{\det\left[\frac{1}{M}\sum_{i=1}^{M}\boldsymbol{\epsilon}(i,\boldsymbol{\theta})\boldsymbol{\epsilon}^{\mathrm{T}}(i,\boldsymbol{\theta})\right]\right\} + \frac{k}{M}. \tag{2.42}$$

Table 2.2. *AIC for different order combinations.*

	The delay is $d = 1$						
n	$m = 1$	2	3	4	5	6	7
1	1.484	−0.25541	−0.66303	−1.0494	−1.57	−2.6414	−3.4085
2	1.346	−2.1263	−2.3685	−4.9326	−5.2359	−7.4658	−7.6678
3	1.0658	−2.8886	−3.4758	−5.4795	−5.6407	−7.7744	−7.9316
4	1.0329	−7.8839	−10.53	−20.733	−20.973	−20.984	−20.9737
5	1.0043	−10.034	−13.406	−20.971	−21.002	−21.037	−21.0356
6	1.023	−13.694	−18.965	−20.982	−21.037	−21.148	−21.1105
7	0.9909	−16.6423	−20.7387	−21.0160	−21.0324	−21.1105	−21.1115
	The delay is $d = 2$						
1	−0.29215	−0.70464	−1.0849	−1.6057	−2.6827	−3.415	−3.5863
2	−2.1672	−2.4101	−4.9737	−5.2763	−7.477	−7.7083	−10.2034
3	−2.929	−3.5109	−5.5163	−5.6663	−7.8124	−7.9722	−10.5894
4	−7.9075	−10.57	−20.775	−21.013	−21.026	−21.015	−20.9850
5	−10.07	−13.438	−21.011	−21.036	−21.079	−21.077	−21.0617
6	−13.71	−18.991	−21.023	−21.078	−21.184	−21.149	−21.1646
7	−16.6792	−20.7794	−21.0574	−21.0736	−21.1488	−21.1444	−21.1393

where M is the number of measurement points, the $\boldsymbol{\theta}$ vector contains the identified parameters, and k is the number of parameters to be identified. The function `v=aic(H)`, where H is an `idpoly` object calculated by the `arx()` function, can be used to evaluate the AIC value v. If the AIC value is very small, for instance, smaller than -20, which is equivalent to a loss function of 10^{-10}, the n, m, d values can be used as the orders of the identified system.

Example 2.27. Consider again the identification problem in Example 2.26. For different order combinations, the AIC values can be obtained as shown in Table 2.2. It can be seen that the shaded items are acceptable, and thus the orders of these combinations can be used. It can also be seen that even though the order can still be increased, it may not make much of a contribution to the improvement of fitting quality. Thus, the lowest possible orders in the shaded items, i.e., the (4,4,1) and (3,4,2) combinations in the example, are desirable for the system:

```
>> for n=1:7, for m=1:7
      T=arx(U,[n,m,1]); TAic1(n,m)=aic(T);
      T=arx(U,[n,m,2]); TAic2(n,m)=aic(T);
   end, end
```

2.7.3 Generation of Identification Signals

In the previous example, it can be seen that a 48-point input sequence is generated, and the original system can be excited by the sequence to generate the output signal. Based on these signals, the discrete-time model can be identified. However, there may exist some error in the identification results. This error could be contributed by the inadequately chosen input signal. In principle, the input signal has to be “rich” enough (in spectrum, or in a Fourier series expansion sense) to excite the system so that the output signal can reveal the

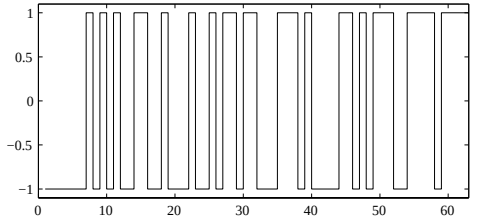


Figure 2.11. *PRBS sequence.*

unknown dynamics. A simple example, is when a constant DC signal is used to excite a system $G(s)$. In the steady state, a constant output signal can be measured. Clearly, only one point in the Nyquist plot $G(0)$, known as the DC gain, can be identified. Therefore, a DC signal fails to excite the system dynamics and it cannot be used to excite the system for system identification purposes. It is desirable to have an input signal that is "persistently exciting."

A pseudorandom binary sequence (PRBS) signal is a class of useful, "persistently exciting" signals suitable for identification purposes. The signals can be generated with the function `u=idinput(k,'prbs')`, where k is the length of the sequence and $k = 2^n - 1$ with n an integer.

Example 2.28. To generate a PRBS sequence of length 63, the following statements can be used:

```
>> u=idinput(63,'PRBS'); t=[0:.1:6.2]';
   stairs(u), set(gca,'XLim',[0,63],'YLim',[-1.1 1.1])
```

The PRBS generated is shown in Figure 2.11.

With the PRBS signal of length 31, the input and output data can be calculated, from which the discrete-time transfer function model can be identified as

```
>> num=[6 -0.6 -0.12]; den=[1 -1 0.25 0.25 -0.125];
   G=tf(num,den,'Ts',0.1);
   y=lsim(G,u,t); T1=arx([y,u],[4 4 1])
```

with the identification results

```
Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + e(t)
 A(q) = 1 - q^-1 + 0.25 q^-2 + 0.25 q^-3 - 0.125 q^-4
 B(q) = -2.141e-015 q^-1 + 6 q^-2 - 0.6 q^-3 - 0.12 q^-4
Estimated using ARX
Loss function 7.46734e-030 and FPE 1.2662e-029
```

Then the identified model is

$$G(z) = \frac{-2.141 \times 10^{-15} z^3 + 6z^2 - 0.6z - 0.12}{z^4 - z^3 + 0.25z^2 + 0.25z - 0.125}.$$

It can be seen that the model identified is almost the same as the original model. It is obvious that although much less input-output data are used than used in Example 2.26, the accuracy of the identification is much better. This is why the PRBS signal is a good input signal to use for identification problems.

There exist many identification algorithms for continuous systems. For instance, Levy's method can be used in frequency response fitting to the original model [33]. However, since the frequency response fitting may not be unique, the identification results may sometimes not be usable. Indirect identification methods can be used instead. One may identify the discrete-time transfer function, then convert the results into continuous systems with the `d2c()` function.

Example 2.29. Consider the continuous model

$$G(s) = \frac{s^3 + 7s^2 + 11s + 5}{s^4 + 7s^3 + 21s^2 + 37s + 30}.$$

Let the sampling interval be $T = 0.1$ second. The following statements can be used to excite the system with PRBS signals, and the input and output data can be generated:

```
>> G=tf([1,7,11,5],[1,7,21,37,30]);
   t=[0:.2:6]'; u=idinput(31,'PRBS');
   y=lsim(G,u,t); U=arx([y u],[4 4 1]);
   G1=tf(U); G1=G1(1); G1.Ts=0.2; G2=d2c(G1)
```

The identified model is then

$$G(s) = \frac{s^3 + 7s^2 + 11s + 5}{s^4 + 7s^3 + 21s^2 + 37s + 30}.$$

It can be seen that the identified model is very accurate.

If sinusoidal signals with 81 samples are used instead to excite the original system model, the following statements can be used to identify the system:

```
>> t=[0:.1:8]'; u=sin(t); y=lsim(G,u,t); u1=iddata(y,u,0.1);
   U=arx(u1,[4 4 1]); G1=tf(U); G1=G1(1); G2=d2c(G1)
```

Thus the identified model is

$$G(s) = \frac{0.01361s^3 - 0.06793s^2 + 9.897s - 2.564}{s^4 + 7s^3 + 21s^2 + 37s + 30}.$$

It can be seen that although more data are used, the identified model is not satisfactory. In fact, the results may be misleading. It should be noted that the frequency spectrum of the input data is very narrow. Thus it is not surprising that the sinusoidal signal gives erroneous results. From this example, one can better appreciate the role of "persistent excitation." In fact, $\mathbf{\Phi}^{\mathbf{T}}\mathbf{\Phi}$ in (2.41) might be ill-conditioned if the input excitation signal $u(t)$ is not properly designed for the identification experiments.

2.7.4 Identification of Multivariable Systems

The `arx()` function can also be used in the identification of multivariable systems. In the system, suppose that there are p inputs and q outputs. The difference equation for the multivariable system can be written as

$$\boldsymbol{A}(z^{-1})\boldsymbol{y}(t) = \boldsymbol{B}(z^{-1})\boldsymbol{u}(t-\boldsymbol{d}) + \boldsymbol{\varepsilon}(t), \tag{2.43}$$

where $\boldsymbol{d}$ is the delay matrix, $\boldsymbol{A}(z^{-1})$ and $\boldsymbol{B}(z^{-1})$ are both $p \times q$ polynomial matrices, and

$$\begin{cases} \boldsymbol{A}(z^{-1}) = \boldsymbol{I}_{p\times q} + \boldsymbol{A}_1 z^{-1} + \cdots + \boldsymbol{A}_{n_a} z^{-n_a}, \\ \boldsymbol{B}(z^{-1}) = \boldsymbol{I}_{p\times q} + \boldsymbol{B}_1 z^{-1} + \cdots + \boldsymbol{B}_{n_b} z^{-n_b}. \end{cases} \tag{2.44}$$

With the use of the `arx()` function, the matrices $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ can be obtained, and the transfer function matrix can be extracted with the `tf()` function.

Example 2.30. Assume that the transfer function matrix is given by

$$\boldsymbol{G}(z) = \begin{bmatrix} \dfrac{0.5234z - 0.1235}{z^2 + 0.8864z + 0.4352} & \dfrac{3z + 0.69}{z^2 + 1.084z + 0.3974} \\ \dfrac{1.2z - 0.54}{z^2 + 1.764z + 0.9804} & \dfrac{3.4z - 1.469}{z^2 + 0.24z + 0.2848} \end{bmatrix}.$$

The two input signals can be individually set to PRBS sequences. To cancel out the correlations of the two sets of signals, the two sequences u_1 and u_2 are arranged in reverse order. The following statements can be used to identify the system model:

```
>> u1=idinput(31,'PRBS'); t=0:.1:3; u2=u1(end:-1:1);
   g11=tf([0.5234, -0.1235],[1, 0.8864, 0.4352],'Ts',0.1);
   g12=tf([3, 0.69],[1, 1.084, 0.3974],'Ts',0.1);
   g21=tf([1.2, -0.54],[1, 1.764, 0.9804],'Ts',0.1);
   g22=tf([3.4, 1.469],[1, 0.24, 0.2848],'Ts',0.1);
   G=[g11, g12; g21, g22]; y=lsim(G,[u1 u2],t);
   na=4*ones(2); nb=na; nc=ones(2);
   U=iddata(y,[u1,u2],0.1); T=arx(U,[na nb nc])
```

The difference equation identified is a multivariable equation, and it can be converted to the required multivariable transfer function matrix. For instance, taking into consideration the subtransfer function item, with the first input versus the first output, the subtransfer function $g_{11}(z)$ can be extracted from

```
>> H=tf(T); g11=H(1,1)
```

and one finds that

$$g_{11}(z) = \frac{\begin{array}{l} 0.5234z^{11} + 1.493z^{10} + 1.847z^9 + 1.235z^8 + 0.5004z^7 + 0.09574z^6 - 0.01551z^5 \\ -0.0137z^4 - 1.683\times10^{-16}z^3 - 3.582\times10^{-17}z^2 - 4\times10^{-18}z + 5.362\times10^{-19} \end{array}}{z^{12} + 3.974z^{11} + 7.431z^{10} + 8.483z^9 + 6.585z^8 + 3.611z^7 + 1.401z^6}.$$

The order of the model is very high, and thus the minimum realization method to the model should be used, with relatively large error tolerance of $\epsilon = 10^{-4}$, to find a closer transfer function to the original one,


```
>> G11=minreal(g11,1e-4)
```

and the subtransfer function

$$g_{11}(z) = \frac{0.5234z - 0.1235}{z^2 + 0.8864z + 0.4352}$$

can be identified. Using similar methods, the other subtransfer functions can be extracted from the identified model. The transfer function matrix can also be obtained with

```
>> H=minreal(H(1:2,1:2),1e-3)
```

Since the state space equations are not unique, sometimes it is not a good choice to identify the state space model of the system from measured input and output data, since there are too many redundant parameters to be identified.

Problems

1. Enter the following system models into the MATLAB environment:

   (a) $G(s) = \dfrac{s^3 + 4s^2 + 3s + 2}{s^2(s+1)[(s+4)^2 + 4]}$,

   (b) $\dot{\boldsymbol{x}}(t) = \begin{bmatrix} -0.3 & 0.1 & -0.05 \\ 1 & 0.1 & 0 \\ -1.5 & -8.9 & -0.05 \end{bmatrix}\boldsymbol{x}(t) + \begin{bmatrix} 2 \\ 0 \\ 4 \end{bmatrix}u(t), \quad y = [1, 2, 3]\boldsymbol{x}(t).$

2. Suppose that the models in Problem 1 are all open-loop models. Using MATLAB, evaluate the closed-loop models if unity negative feedback is assumed. Find all the open-loop and closed-loop poles and zeros of the above models.

3. Assume that the linear ODEs describing a system are given by

   $$\begin{cases} \dot{x}_1(t) = -x_1(t) + x_2(t), \\ \dot{x}_2(t) = -x_2(t) - 3x_3(t) + u_1(t) \\ \dot{x}_3(t) = -x_1(t) - 5x_2(t) - 3x_3(t) + u_2(t), \end{cases}$$
   and $y = -x_2(t) + u_1(t) - 5u_2(t)$,

   where there are two input signals $u_1(t)$ and $u_2(t)$. Model the two-input single-output (TISO) system in the MATLAB workspace.

4. An ODE is given by

   $$y^{(3)}(t) + 13\ddot{y}(t) + 6\dot{y}(t) + 5y(t) = 2u(t).$$

   Select a set of states and represent the equation in the MATLAB workspace.

5. Find the equivalent transfer function for the state space model

   $$\dot{\boldsymbol{x}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}\boldsymbol{x} + \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix}u, \quad y = [1, 2, 3]\boldsymbol{x}$$

   and also find the poles and zeros of the model.

6. Assume that in the typical feedback control structure, the blocks are given by

(a) $G(s) = \dfrac{211.87s + 317.64}{(s+20)(s+94.34)(s+0.1684)}$,
$G_c(s) = \dfrac{169.6s + 400}{s(s+4)}$, $H(s) = \dfrac{1}{0.01s+1}$;

(b) $G(s) = \dfrac{35786.7s + 108444}{(s+4)(s+20)(s+74.04)}$, $G_c(s) = \dfrac{1}{s}$, $H(s) = \dfrac{1}{0.01s+1}$;

Find state space models and transfer functions of the overall systems. Get the zero-pole-gain representations of the systems.

7. Suppose that a typical feedback system is given such that

$$G(s) = \frac{K_m J}{Js^2 + Bs + K_r}, \quad G_c(s) = \frac{L_q}{L_q s + R_q}, \quad H(s) = sK_v.$$

Find the model.

8. Enter the following plant model into MATLAB:

$$G(s) = \frac{1}{s^5 + 8s^4 + 19.5s^3 + 19s^2 + 7.5s + 1}$$

and evaluate the closed-loop model if unity positive or negative feedback is assumed. Find and make comments on the closed-loop poles and zeros.

9. Find a state space realization of the plant model given by $G(s) = 1/(s+1)^3$. Comment on what may affect the Jordanian canonical form. Compare the computer results with those obtained by direct manual calculations.

10. Consider the system models

(a) $\dot{\boldsymbol{x}} = \begin{bmatrix} -9 & -26 & -24 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1800000 \\ 0 \\ 1234 \end{bmatrix} u, \quad y = [0, 1, 1.5 \times 10^{-5}]\boldsymbol{x}$

(b) $G(s) = \dfrac{1.25 \times 10^8 s^2 + 50s + 1.33 \times 10^{-4}}{s^4 + 10s^3 + 35s^2 + 50s + 24}$.

Perform balanced realizations for the systems.

11. Assume that the models of the systems are given by

(a) $\dot{\boldsymbol{x}} = \begin{bmatrix} -9 & -26 & -24 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & -1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} u, \quad y = [0, 1, 1, 2]\boldsymbol{x}$

(b) $G(s) = \dfrac{2s^2 + 18s + 16}{s^4 + 10s^3 + 35s^2 + 50s + 24}$.

Try to check whether these models are minimally realized. If not, find the minimally realized models and give an explanation from the transfer function point of view.

12. Suppose that an overall system is composed from the series connection of two blocks $G_1(s)$ and $G_2(s)$ given, respectively, by

$$G_1(s) = \frac{s+1}{s^2+3s+4} \quad \text{and} \quad G_2(s) = \frac{s^2+3s+5}{s^4+4s^3+3s^2+2s+1}.$$

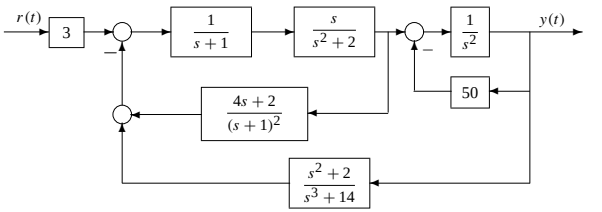
If the state space representation for the overall system is required, compare the difference in the results using the following two approaches in MATLAB:
(a) Perform the series connection of the two transfer functions, and then find the state space expression for the overall system model.
(b) Find the state space expressions of the two blocks, and then find the overall system model.

13. Assume that the multivariable plant $\boldsymbol{G}(s)$ and its precontroller $\boldsymbol{G}_c(s)$ are given by

$$\boldsymbol{G}(s) = \begin{bmatrix} \dfrac{-0.252}{(1+3.3s)^3(1+1800s)} & \dfrac{0.43}{(1+12s)(1+1800s)} \\ \dfrac{-0.0435}{(1+25.3s)^3(1+360s)} & \dfrac{0.097}{(1+12s)(1+360s)} \end{bmatrix}, \quad \boldsymbol{G}_c(s) = \begin{bmatrix} -10 & 77.5 \\ 0 & 50 \end{bmatrix}.$$

Evaluate the closed-loop transfer function matrix under unity negative feedback, and then find the state space realization.

14. Derive the overall system model from $r(t)$ to $y(t)$ as shown in the following block diagram:



15. Assume that the plant model is given by

$$G(s) = \frac{12}{s(s+1)^3} e^{-2s},$$

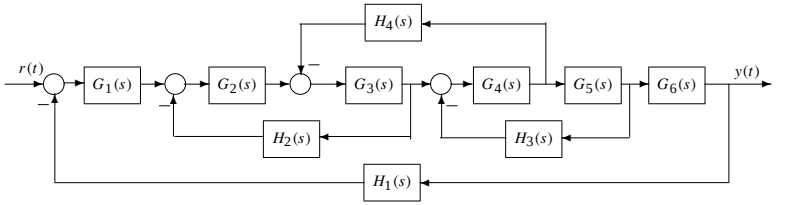
and the controller is

$$G_c(s) = \frac{2s+3}{s}.$$

For a unity negative feedback system, check whether it is possible to express the closed-loop system by the MATLAB `tf` object. Please give reasons why.

16. Draw the PRBS sequence for 127 points and draw the autocorrelation function of the sequence with the `xcorr()` function.

17. If the block diagram of a linear system is shown as below, derive the total system model from the input $r(t)$ to the output $y(t)$:



18. Suppose that the measured input/output data of a discrete-time model is given in the table below. Identify the transfer function model, based on the suitable order selection with AIC values:

i	u_i	y_i	i	u_i	y_i	i	u_i	y_i
1	0.9103	0	9	0.9910	54.5252	17	0.6316	62.1589
2	0.7622	18.4984	10	0.3653	65.9972	18	0.8847	63.0000
3	0.2625	31.4285	11	0.2470	62.9181	19	0.2727	68.6356
4	0.0475	32.3228	12	0.9826	57.5592	20	0.4364	60.8267
5	0.7361	28.5690	13	0.7227	67.6080	21	0.7665	57.1745
6	0.3282	39.1704	14	0.7534	70.7397	22	0.4777	60.5321
7	0.6326	39.8825	15	0.6515	73.7718	23	0.2378	57.3803
8	0.7564	46.4963	16	0.0727	74.0165	24	0.2749	49.6011

19. For a system model

$$G(s) = \frac{4s^2 - 4}{s^4 + 7s^3 + 18s^2 + 22s + 12},$$

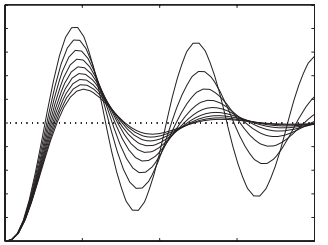
excite the system by different signals, for instance, step signal, sinusoidal signal, and PRBS signal. Check how many samples are necessary to accurately identify the system model.

20. Based on the AIC criterion, suitable orders can be found and the discrete-time model can be identified. In control systems analysis and design, however, sometimes a low-order approximate model may be needed. This is the topic of model reduction and will be explored in Chapter 3. Try to find a good low-order approximation for the data given in Problem 18 and test how good the reduced-order models are.

21. Suppose that the measured step response data of a continuous system are as shown in the table below. Identify the transfer function model and, with the help of the AIC

values, determine a suitable order combination for the system:

t	$y(t)$	t	$y(t)$	t	$y(t)$	t	$y(t)$	t	$y(t)$	t	$y(t)$
0	0	1.6	0.2822	3.2	0.3024	4.8	0.3145	6.4	0.3218	8	0.3263
0.1	0.08324	1.7	0.2839	3.3	0.3034	4.9	0.315	6.5	0.3222	8.1	0.3265
0.2	0.1404	1.8	0.2855	3.4	0.3043	5	0.3156	6.6	0.3225	8.2	0.3267
0.3	0.1798	1.9	0.287	3.5	0.3051	5.1	0.3161	6.7	0.3228	8.3	0.3269
0.4	0.2072	2	0.2885	3.6	0.306	5.2	0.3166	6.8	0.3231	8.4	0.3271
0.5	0.2265	2.1	0.2899	3.7	0.3068	5.3	0.3172	6.9	0.3235	8.5	0.3273
0.6	0.2402	2.2	0.2912	3.8	0.3076	5.4	0.3176	7	0.3238	8.6	0.3275
0.7	0.2501	2.3	0.2925	3.9	0.3084	5.5	0.3181	7.1	0.324	8.7	0.3277
0.8	0.2574	2.4	0.2937	4	0.3092	5.6	0.3186	7.2	0.3243	8.8	0.3278
0.9	0.2629	2.5	0.2949	4.1	0.3099	5.7	0.319	7.3	0.3246	8.9	0.328
1	0.2673	2.6	0.2961	4.2	0.3106	5.8	0.3195	7.4	0.3249	9	0.3282
1.1	0.2708	2.7	0.2973	4.3	0.3113	5.9	0.3199	7.5	0.3251	9.1	0.3283
1.2	0.2737	2.8	0.2983	4.4	0.312	6	0.3203	7.6	0.3254	9.2	0.3285
1.3	0.2762	2.9	0.2994	4.5	0.3126	6.1	0.3207	7.7	0.3256	9.3	0.3286
1.4	0.2784	3	0.3004	4.6	0.3133	6.2	0.3211	7.8	0.3258	9.4	0.3288
1.5	0.2804	3.1	0.3014	4.7	0.3139	6.3	0.3214	7.9	0.3261	9.5	0.3289



Chapter 3

Analysis of Linear Control Systems

The most important property of a linear system is the well-known superposition principle. Assume that the response of a system to a signal $\boldsymbol{u}_1(t)$ is $\boldsymbol{y}_1(t)$ and the response to a signal $\boldsymbol{u}_2(t)$ is $\boldsymbol{y}_2(t)$. Then, the system is linear if for any constants a and b, the response for the signal $a\boldsymbol{u}_1(t) + b\boldsymbol{u}_2(t)$ can be represented by $a\,\boldsymbol{y}_1(t) + b\,\boldsymbol{y}_2(t)$.

The models discussed in the previous chapter, such as the transfer function model, the zero-pole-gain model, the state space model (2.18), etc., are all linear time-invariant (LTI) models. Given a mathematical model describing a linear system, we will discuss in this chapter other properties which can be obtained about the linear system. First, the stability and direct stability assessment of LTI systems are discussed in Sec. 3.1. The internal stability property of feedback control systems will also be discussed. Properties of controllability, observability, Gramians, Kalman decomposition, and norm evaluations of systems are also covered in Sec. 3.1. The canonical structural forms of linear control systems and the Kalman decomposition are presented, and the definitions and evaluations of time moments and Markov parameters of a linear system are also given.

Equipped with the properties in Sec. 3.1, we will present time domain analysis of linear systems in Sec. 3.2 and numerical simulation in Sec. 3.3. This enables us to obtain and sketch the step response, impulse response, and time transient response to any input signal. This is proved to be an effective and straightforward way to describe the behavior of the systems. In Sec. 3.4, the root locus of the system is studied which illustrates the behavior of the system that might be expected from its poles in the complex s-plane. In Sec. 3.5, the frequency domain analysis of a linear system is performed, with different graphical analysis tools presented, such as the Bode diagrams, Nyquist plots, and Nichols charts. Finally, Sec. 3.6 is an introduction to various model reduction techniques for linear systems.

3.1 Properties of Linear Control Systems

3.1.1 Stability Analysis

Direct stability assessment

When feedback is used, a system could be either stable or unstable. That is, an open-loop stable system could become unstable or destabilized after feedback control, which is undesirable. Conversely, an open-loop unstable system could be stable or stabilized, which is desirable. Therefore, stability is a fundamental requirement for any controlled system.

There are many different stability notions. Here, we first focus on the bounded input–bounded output (BIBO) stability notion. Then, we discuss the internal stability. BIBO stability of a system ensures that the output of the system remains bounded over any amount of time if the input signal is bounded. In other words, if a dynamic system is BIBO stable, the output signal cannot "blow up" if the input remains finite.

The easiest way to check the stability of a linear continuous system is to check the pole locations in the complex plane. If there is a pole with a positive real part, the system is said to be unstable and this pole is referred to as an unstable mode of the system. In other words, if there is one or more poles on the right half plane (RHP), the system is unstable. The system is stable if all poles have negative real parts, that is, all poles lie in the left half plane (LHP).

For poles on the imaginary axis, there are two cases. If a specific pole on the imaginary axis is simple, that is, multiplicity is only one or it is not repeating, this pole is a critically stable pole. If a specific pole on the imaginary axis has multiplicity more than one, it is an unstable pole. Note that BIBO is a stronger stability notion. For example, a pure integrator is stable but not BIBO stable since, given a bounded input such as a unit step signal, the output is t, which is unbounded.

The poles of a given LTI model G can be obtained directly with `eig(G)` or `pole(G)`. If there is no right-hand-side s-plane pole for a continuous system G, then it is stable. If G is a discrete model, however, the magnitudes of the poles, which can be evaluated by `abs(eig(G))` or `abs(pole(G))`, are all smaller than 1, and the system G is stable. To keep the magnitudes of the poles smaller than 1 means that all the poles are located within a unit circle.

The zeros of the system G can be obtained with the function `zero(G)`, and the poles and zeros of G can be sketched with the function `pzmap(G)`.

Example 3.1. Suppose the transfer function model of a system is given by

$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24}.$$

As illustrated below, the poles can be evaluated and sketched with the statements

```
>> G=tf([1,7,24,24],[1,10,35,50,24]); eig(G), pzmap(G)
```

and it can be found that the poles are located at $-1, -2, -3, -4$, all on the left-hand side of the s-plane, which means that G is stable. This can also be verified by the pole positions shown in Figure 3.1.

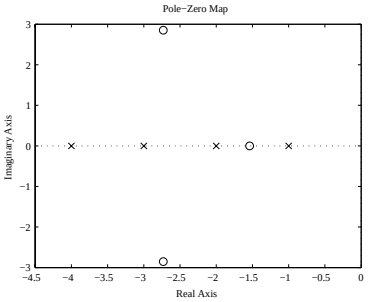


Figure 3.1. *Example* 3.1.

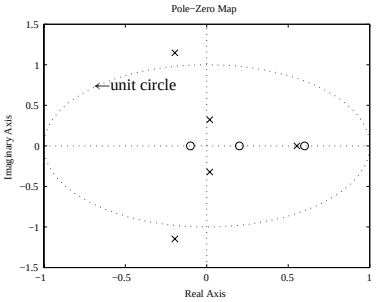


Figure 3.2. *Example* 3.2.

Example 3.2. Suppose that in a discrete-time unity negative feedback system, the plant model is given by

$$H(z) = \frac{6z^2 - 0.6z - 0.12}{z^4 - z^3 + 0.25z^2 + 0.25z - 0.125},$$

with sampling interval $T = 0.1$, and the controller is

$$G_c(z) = 0.3\frac{z - 0.6}{z + 0.8}.$$

The stability of the closed-loop system can be assessed with the following statements:

```
>> num=[6 -0.6 -0.12]; den=[1 -1 0.25 0.25 -0.125];
   H=tf(num,den,'Ts',0.1);   % plant model
   z=tf('z','Ts',0.1); Gc=0.3*(z-0.6)/(z+0.8);  % controller model
   GG=feedback(H*Gc,1); abs(eig(GG)), pzmap(GG)
```

It can be seen that the magnitudes of the poles are, respectively, 1.1644, 1.1644, 0.5536, 0.3232, 0.3232. Since the magnitudes of the first two poles are greater than 1, which means that they are outside the unit circle, then the closed-loop system is unstable. The pole-zero positions in Figure 3.2 also verify the result.

Well-posedness and internal stability

In terms of good control behavior in a feedback control system, the stability criteria given in the previous sections are not sufficient since only the input-output stability of the system is considered. In other words, the stability criteria ensures that if an input-output stable system is driven by a bounded signal, the output is also bounded. However, it does not guarantee that the other signals inside the system are bounded. If the signals inside the system are not bounded, the internal physical structures of the system may be damaged.

Consider the general feedback system structure shown in Figure 3.3 which is an immediate extension of the typical feedback system structure. In Figure 3.3, the signal d is often called the external disturbance and the signal n the measurement noise.

Before introducing the concept of internal stability, the definition of well-posedness will be presented first.

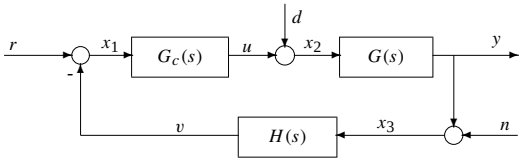


Figure 3.3. *Linear feedback control structure with disturbances.*

Definition 3.1. The feedback system shown in Figure 3.3 is said to be well-posed if all nine closed-loop transfer functions from the input signals (r, d, n) to the output signals (u, y, v) exist.

Well-posedness can be determined by the following theorem.

Theorem 3.1. The system is well-posed if and only if the 3×3 matrix

$$\begin{bmatrix} 1 & 0 & H(s) \\ -G_c(s) & 1 & 0 \\ 0 & -G(s) & 1 \end{bmatrix} \tag{3.1}$$

is nonsingular, i.e., the determinant of the matrix $1 + G(s)G_c(s)H(s)$ does not equal zero.

Definition 3.2. The system shown in Figure 3.3 is said to be internally stable if all nine closed-loop transfer functions from the inputs (r, d, n) to the internal signals (x_1, x_2, x_3) are stable.

The nine transfer functions can be described by

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{1}{1+G(s)G_c(s)H(s)} \begin{bmatrix} 1 & -G(s)H(s) & -H(s) \\ G_c(s) & 1 & -G_c(s)H(s) \\ G(s)G_c(s) & G(s) & 1 \end{bmatrix} \begin{bmatrix} r \\ d \\ n \end{bmatrix}. \tag{3.2}$$

Theorem 3.2. The system is internally stable if and only if the following two conditions are satisfied:

(1) The transfer function $1 + H(s)G(s)G_c(s)$ has no zeros on $\text{Re}[s] \geq 0$.

(2) $H(s)G(s)G_c(s)$ has no pole-zero cancellation on $\text{Re}[s] \geq 0$.

The first condition is equivalent to saying that the closed-loop system is input-output stable. So, one can concentrate on the second condition, which is also very easy to check by using the MATLAB function `intstable()`:

```
function key=intstable(G,Gc,H)
GG=minreal(feedback(G*Gc,H)); Go=H*G*Gc; Go1=minreal(Go); p=eig(GG);
z0=eig(Go); z1=eig(Go1); zz=setdiff(z0,z1); % find the cancellations
if (G.Ts>1), % discrete-time system
   key=any(abs(p)>1); if key==0, key=2*any(abs(zz)>1); end
else, % continuous system
   key=any(real(p)>0); if key==0, key=2*any(real(zz)>0); end
end
```


Its syntax is `[V,c]=intstable(G,Gc,H)`. In the function call, the returned variables are defined as follows:

(1) If the system is internally stable, $V = 0$ is returned and $\boldsymbol{c}$ is empty.

(2) If $V = 1$, the system is input-output unstable and the unstable closed-loop poles are returned in the vector $\boldsymbol{c}$.

(3) If $V = 2$, the system is input-output stable, but not internally stable, and the internally canceled unstable poles are returned in $\boldsymbol{c}$.

Example 3.3. Consider the typical feedback system with

$$G(s) = \frac{5(s-1)(s+2)}{s^3+4s^2+3s+4}, \quad G_c(s) = \frac{s^2+3s+4}{(s-1)(s^2+3s+2)}, \quad H(s) = 1.$$

The stability of the system can be tested by the following MATLAB statements:

```
>> s=tf('s'); G=5*(s-1)*(s+2)/(s^3+4*s^2+3*s+4);
   Gc=(s^2+3*s+4)/((s-1)*(s^2+3*s+2)); H=1;
   G_a=minreal(ss(feedback(Gc*G,H))); eig(G_a)
```

The closed-loop poles of the system, after the two pairs of pole-zero cancellation, are located at $-0.2328 \pm \text{j}2.0546$, $-2.2672 \pm \text{j}0.6879$, which are all on the left-hand side of the s-plane. It can be seen that the closed-loop system is stable. However, by checking the internal stability from the statement

```
>> [V,cc]=intstable(G,Gc,H)
```

we conclude that the system is not internally stable with the canceled pole-zero having positive real parts $z_i = p_j = 1$ returned in `cc`.

3.1.2 Controllability and Observability Analysis

Controllability of linear systems

Controllability is an important property of a control system and plays a crucial role in many control problems, such as stabilization of unstable systems by feedback control.

Definition 3.3. The state $x_i(t)$ is said to be controllable if there is an input that in finite time drives it to any specified $x_i(t_f)$ from initial state $x_i(0)$. The system is said to be fully controllable if all its states are controllable.

Since the full controllability of the system depends only upon the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices of the state space model, it is simply said that $(\boldsymbol{A}, \boldsymbol{B})$ is controllable.

Construct a transformation matrix $\boldsymbol{T}_c$ in the form

$$\boldsymbol{T}_c = [\boldsymbol{B}, \boldsymbol{A}\boldsymbol{B}, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{B}], \tag{3.3}$$

where n is the order of the system or the number of states. The matrix $\boldsymbol{T}_c$ is referred to as the controllability matrix and it can be generated using the MATLAB function `ctrb()`,

provided in the Control Systems Toolbox, with the syntax `Tc=ctrb(A,B)`. The rank of $\boldsymbol{T}_c$, i.e., rank($\boldsymbol{T}_c$), is called the controllability index of the system and equals the number of controllable states in the system. If rank($\boldsymbol{T}_c$) $= n$, the system is fully controllable.

Under a suitably chosen transformation matrix $\widehat{\boldsymbol{T}}_c$, the state space model can be transformed into the following canonical form, through the staircase transformation;

$$\boldsymbol{A}_c = \begin{bmatrix} \widehat{\boldsymbol{A}}_{\bar{c}} & \boldsymbol{0} \\ \widehat{\boldsymbol{A}}_{21} & \widehat{\boldsymbol{A}}_c \end{bmatrix}, \quad \boldsymbol{B}_c = \begin{bmatrix} \boldsymbol{0} \\ \widehat{\boldsymbol{B}}_c \end{bmatrix}, \quad \boldsymbol{C}_c = [\widehat{\boldsymbol{C}}_{\bar{c}}, \widehat{\boldsymbol{C}}_c]. \tag{3.4}$$

The above transformed state space representation is known as the controllability staircase form. The eigenvalues of $\widehat{\boldsymbol{A}}_{\bar{c}}$ are called the uncontrollable modes. If the system is controllable, the uncontrollable subspace $\widehat{\boldsymbol{A}}_{\bar{c}}$ will be empty. The simplified transfer function of the system can be obtained from the controllable subspace

$$\boldsymbol{G}(s) = \widehat{\boldsymbol{C}}_c(s\boldsymbol{I} - \widehat{\boldsymbol{A}}_c)^{-1}\widehat{\boldsymbol{B}}_c + \boldsymbol{D}. \tag{3.5}$$

A MATLAB function `ctrbf()`, provided in the Control Systems Toolbox, can be used to perform the controllable staircase form transformation. The syntax of this function is `[Ac,Bc,Cc,Tc]=ctrbf(A,B,C)`, where ($\boldsymbol{A}$,$\boldsymbol{B}$,$\boldsymbol{C}$) is the given state space model and the returned state space model ($\boldsymbol{A}_c$,$\boldsymbol{B}_c$,$\boldsymbol{C}_c$) has a staircase format which separates the controllable and uncontrollable subspaces. The returned matrix $\boldsymbol{T}_c$ is the transformation matrix.

Example 3.4. Consider a state space model given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ -2 \end{bmatrix} u, \quad y = [1, 0, 0, 0]\boldsymbol{x}.$$

One can use the following statements to check the controllability of the system:

```
>> A=[0,1,0,0; 0,0,-1,0; 0,0,0,1; 0,0,5,0]; B=[0; 1; 0; -2];
   C=[1,0,0,0]; D=0; Tc=[B, A*B, A^2*B, A^3*B]; rank(Tc)
```

Since the rank of $\boldsymbol{T}_c$ is 4, which equals the order of the system, the system is fully controllable.

Example 3.5. Let us consider another system model given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 3 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & -2 & 0 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} u, \quad y = [1, 0, 0, 0]\boldsymbol{x}.$$

The controllability can be analyzed using the following MATLAB statements:

```
>> A=[0,1,0,0; 3,0,0,2; 0,0,0,1; 0,-2,0,0]; B=[0;1;0;0];
   C=[1,0,0,0]; Tc=[B, A*B, A^2*B, A^3*B]; rank(Tc)
   [Ac,Bc,Cc,T]=ctrbf(A,B,C)
```

The controllable index of the system is 3 since the rank of $\boldsymbol{T}_c$ is 3. The controllable staircase form can be written, in partitioned form, as

$$\boldsymbol{A}_c = \left[\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \hline -0.4472 & 0 & -0.8944 & 0 \\ 0 & 0 & 0 & -2.236 \\ 3.5777 & 0 & 0.4472 & 0 \end{array}\right], \; \boldsymbol{B}_c = \begin{bmatrix} 0 \\ \hline 0 \\ 0 \\ -1 \end{bmatrix}, \; \boldsymbol{C}_c = \left[\begin{array}{c|ccc} -0.8944 & 0 & 0.4472 & 0 \end{array}\right]$$

and it can be observed that the uncontrollable mode is at $s = 0$, which is the eigenvalue of the $\widehat{\boldsymbol{A}}_{\bar{c}}$ matrix.

Observability of linear systems

Observability is a measure of how well internal states of a system can be inferred from knowledge of its external inputs and outputs. The observability and controllability of a system are mathematical duals.

Definition 3.4. A state $x_i(t)$ is said to be observable if for any $t_f > 0$, the initial state $x_i(0)$ can be determined from the time history of the input $\boldsymbol{u}(t)$ and the output $\boldsymbol{y}(t)$ in the interval of $[0, t_f]$. The system is said to be fully observable if all the states in the system are observable.

Since the observability of the system depends only upon the $\boldsymbol{A}$ and $\boldsymbol{C}$ matrices of the state space model, one can simply say that ($\boldsymbol{A}$, $\boldsymbol{C}$) is observable.

Construct a transformation matrix $\boldsymbol{T}_o$ in the following form:

$$\boldsymbol{T}_o = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{C}\boldsymbol{A} \\ \vdots \\ \boldsymbol{C}\boldsymbol{A}^{n-1} \end{bmatrix}, \tag{3.6}$$

where n is the order of the system. $\boldsymbol{T}_o$ is referred to as the observability matrix, which can be generated using the MATLAB function `obsv()`, provided in the Control Systems Toolbox by `To=obsv(A,C)`. rank($\boldsymbol{T}_o$) is called the observability index of the system or the number of observable states. If rank($\boldsymbol{T}_o$) $= n$, the system is then fully observable.

With a suitable transformation matrix $\widehat{\boldsymbol{T}}_o$, the state space model can be transformed into the following canonical form:

$$\boldsymbol{A}_o = \begin{bmatrix} \widehat{\boldsymbol{A}}_{\bar{o}} & \widehat{\boldsymbol{A}}_{12} \\ \boldsymbol{0} & \widehat{\boldsymbol{A}}_o \end{bmatrix}, \quad \boldsymbol{B}_o = \begin{bmatrix} \widehat{\boldsymbol{B}}_{\bar{o}} \\ \widehat{\boldsymbol{B}}_o \end{bmatrix}, \quad \boldsymbol{C}_o = [\boldsymbol{0}, \widehat{\boldsymbol{C}}_o], \tag{3.7}$$

known as the observability staircase form. The eigenvalues of $\widehat{A}_{\bar{o}}$ are called the unobservable modes. If the system is fully observable, the unobservable subspace $\widehat{A}_{\bar{o}}$ will be empty. The transfer function of the system can be simply expressed by

$$G(s) = \widehat{C}_o(sI - \widehat{A}_o)^{-1}\widehat{B}_o + D. \tag{3.8}$$

By comparing the controllability problem to the observability problem, it is not difficult to see that these problems are dual. That is, the observability problem of the system $(A^{\mathrm{T}}, C^{\mathrm{T}}, B^{\mathrm{T}}, D^{\mathrm{T}})$ is exactly the same as the controllability problem of the system (A, B, C, D).

A MATLAB function `obsvf()` in the Control Systems Toolbox can be used to perform the staircase form transformation. The syntax is

`[Ao,Bo,Co,To]=obsvf(A,B,C)`

and the arguments are similar to those of the `ctrbf()` function.

Example 3.6. Consider again the uncontrollable system shown in Example 3.5. The observability index and the staircase form can be obtained by

```
>> rank([C; C*A; C*A^2; C*A^3]) % or rank(obsv(A,C))
   [Ao,Bo,Co,T,K]=obsvf(A,B,C); Ao,Bo,Co
```

In the partitioned matrix form, the staircase form can be written as

$$A_o = \left[\begin{array}{c|ccc} 0 & 1 & 0 & 0 \\ \hline 0 & 0 & -2 & 0 \\ 0 & 2 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{array}\right], \quad B_o = \left[\begin{array}{c} 0 \\ \hline 0 \\ 1 \\ 0 \end{array}\right], \quad C_o = \left[\begin{array}{c|ccc} 0 & 0 & 0 & 1 \end{array}\right].$$

The unobservable mode is at $s = 0$. In fact, it is not hard to verify that there are two eigenvalues at $s = 0$, with one uncontrollable and the other unobservable.

Controllability and observability Gramians

The controllability and observability Gramians are very important in the balanced realization of a transfer function model and the related model reduction method [30]. In terms of controllability and observability, one may ask how controllable and observable the system is. The controllability and observability Gramians can be used to address this concern.

Definition 3.5. The controllability and observability Gramians of the system (A, B, C, D) are defined, respectively, as

$$W_c = \int_0^\infty \mathrm{e}^{At} B B^{\mathrm{T}} \mathrm{e}^{A^{\mathrm{T}}t}\mathrm{d}t, \quad W_o = \int_0^\infty \mathrm{e}^{A^{\mathrm{T}}t} C^{\mathrm{T}} C \mathrm{e}^{At}\mathrm{d}t. \tag{3.9}$$

From the above definition, W_c and W_o are symmetric positive semidefinite matrices satisfying, respectively, the Lyapunov equations

$$AW_c + W_cA^{\mathrm{T}} = -BB^{\mathrm{T}}, \quad A^{\mathrm{T}}W_o + W_oA = -C^{\mathrm{T}}C, \tag{3.10}$$

where W_c and W_o can be easily solved by calling the corresponding Lyapunov equation solvers `Wc=lyap(A,B*B')` and `Wo=lyap(A',C'*C)`.

Moreover, the following properties for the Gramians hold:

(1) W_c is positive definite if and only if (A, B) is controllable.

(2) W_o is positive definite if and only if (A, C) is observable.

The singular values of W_c, which can be obtained by the standard MATLAB built-in function `svd()`, characterize the contribution of the input signal to each of the states. The larger the ith singular value of W_c, the more the input contributes to the ith state. The singular values of W_o, on the other hand, corresponds to the contribution of each state to the output of the system.

The realization of a transfer function model in state space form is not unique, as discussed in the previous chapter. A specific realization may be more controllable but less observable, or more observable but less controllable. A unique realization $W_c = W_o = W$, known as the balanced realization, is clearly more desirable in some applications.

In the balanced realization, by dropping off the smaller singular values of the common W, a reduced-order model can be obtained. This idea is exactly the balanced realization model reduction technique which will be discussed in more details in Sec. 3.6.

The controllability and observability Gramians can also be computed using the `gram()` function provided in the Control Systems Toolbox. The syntax of the function is

`W=gram(G,type)`,

where G is the state space model object. When the variable `type` equals `'c'`, the controllability Gramian is returned in W. If `type` is `'o'`, the observability Gramian will be returned.

3.1.3 Kalman Decomposition of Linear Systems

The two properties, controllability and observability, discussed previously imply that there are four possible modes of a linear system: controllable and observable, uncontrollable and observable, controllable and unobservable, and uncontrollable and unobservable. Given a linear system, how to decompose it, via a similarity transformation, into these four modes is the major topic of this section. This decomposition is called the Kalman decomposition. It is useful in understanding the inherent inner structure of a linear system.

Kalman decomposition

Kalman pointed out that any state space model can be decomposed into the canonical form

$$\left\{\begin{array}{l} \dot{z}(t) = \left[\begin{array}{c|c|c|c} \widehat{A}_{\bar{c},\bar{o}} & \widehat{A}_{1,2} & 0 & 0 \\ \hline 0 & \widehat{A}_{\bar{c},o} & 0 & 0 \\ \hline \widehat{A}_{3,1} & \widehat{A}_{3,2} & \widehat{A}_{c,\bar{o}} & \widehat{A}_{3,4} \\ \hline 0 & \widehat{A}_{4,2} & 0 & \widehat{A}_{c,o} \end{array}\right] z(t) + \left[\begin{array}{c} 0 \\ \hline 0 \\ \hline \widehat{B}_{c,\bar{o}} \\ \hline \widehat{B}_{c,o} \end{array}\right] u(t), \\ y(t) = \left[\begin{array}{c|c|c|c} 0 & \widehat{C}_{\bar{c},o} & 0 & \widehat{C}_{c,o} \end{array}\right] z(t), \end{array}\right. \tag{3.11}$$

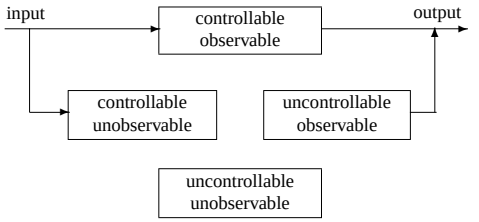


Figure 3.4. *Illustration of Kalman decomposition.*

where the subspace $(\widehat{\boldsymbol{A}}_{\bar{c},\bar{o}}, \mathbf{0}, \mathbf{0})$ is uncontrollable/unobservable, $(\widehat{\boldsymbol{A}}_{\bar{c},o}, \mathbf{0}, \widehat{\boldsymbol{C}}_{c,\bar{o}})$ is controllable/unobservable, $(\widehat{\boldsymbol{A}}_{c,\bar{o}}, \widehat{\boldsymbol{B}}_{c,\bar{o}}, \mathbf{0})$ is observable/uncontrollable, and $(\widehat{\boldsymbol{A}}_{c,o}, \widehat{\boldsymbol{B}}_{c,o}, \widehat{\boldsymbol{C}}_{c,o})$ is controllable/observable. This is the so-called Kalman decomposition form. It can be illustrated by the block diagram shown in Figure 3.4.

Theorem 3.3. The properties such as controllability and observability cannot be changed through any similarity transformation.

A MATLAB function `kalmdec()` is written which can be used to perform the Kalman decomposition of a given system:

```
function [Gk,T,K]=kalmdec(G)
G=ss(G); A=G.a; B=G.b; C=G.c; [Ac,Bc,Cc,Tc,Kc]=ctrbf(A,B,C);
nc=rank(ctrb(A,B),eps*100); n=length(A); ic=n-nc+1:n;
[Ao1,Bo1,Co1,To1,Ko1]=obsvf(Ac(ic,ic),Bc(ic),Cc(ic));
if nc<n, inc=1:n-nc;
   [Ao2,Bo2,Co2,To2,Ko2]=obsvf(Ac(inc,inc),Bc(inc),Cc(inc));
end
[m1,n1]=size(To1); [m2,n2]=size(To2); To=blkdiag(To2,To1);
T=To*Tc; e0=eps*100; n1=rank(obsv(Ac(ic,ic),Cc(ic)),e0);
n2=rank(obsv(Ac(inc,inc),Cc(inc)),e0);
K=[zeros(1,n-nc-n2),ones(1,n2), 2*ones(1,nc-n1), 3*ones(1,n1)];
Ak=T*A*inv(T); Bk=T*B; Ck=C*inv(T); Gk=ss(Ak,Bk,Ck,G.d);
```

The syntax of the function is `[Gk,Tk,k]=kalmdec(G)`. The returned variable $\boldsymbol{G}_k$ is the Kalman decomposition of the system $\boldsymbol{G}$. The variable $\boldsymbol{T}_k$ is the transformation matrix, while the vector $\boldsymbol{k}$ returns a vector which holds the flags for the modes of each state. If the flag is zero, the corresponding state is uncontrollable/unobservable. The flag values of 1, 2, 3 correspond to the uncontrollable/observable, controllable/unobservable, and controllable/observable modes, respectively.

Example 3.7. Consider a linear system model

$$\dot{\boldsymbol{x}} = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & -3 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \\ 3 \\ 0 \end{bmatrix} u, \quad y = [4, 5, 0, 0, 0, 6]\, \boldsymbol{x}.$$

Its Kalman decomposition can be performed using the following MATLAB scripts:

```
>> A=[-1,1,0,0,0,0; 0,-1,0,0,0,0; 0,0,-2,1,0,0;
       0,0,0,-2,0,0; 0,0,0,0,-3,1; 0,0,0,0,0,-3];
   B=[1; 2; 3; 0; 4; 0]; C=[4,5,0,6,0,0]; G=ss(A,B,C,0);
   [Gk,Tk,K]=kalmdec(G),
```

which yield the following familiar mathematical format:

$$\begin{cases} \dot{\boldsymbol{x}} = \left[\begin{array}{cc:cc:cc} -3 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & -2 & 0 & 0 & 0 & 0 \\ \hdashline -0.0832 & -0.9965 & -2.007 & -0.08288 & 0 & 0 \\ -0.9965 & 0.0832 & -0.08288 & -2.993 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & -1.488 & -0.6098 \\ 0 & 0 & 0 & 0 & 0.3902 & -0.5122 \end{array}\right] \boldsymbol{x} + \left[\begin{array}{c} 0 \\ \hdashline 0 \\ \hdashline -3.3223 \\ -3.7366 \\ \hdashline 0.4685 \\ 2.1864 \end{array}\right] u \\ y = \left[\begin{array}{cc:cc:cc} 0 & 6 & 0 & 0 & 0 & 6.4031 \end{array}\right] \boldsymbol{x}. \end{cases}$$

Contrary to (3.11), the partitioned state vector here is defined as $\boldsymbol{x}^{\mathbf{T}} = [\boldsymbol{x}_{\bar{c},\bar{o}}^{\mathbf{T}}, \boldsymbol{x}_{\bar{c},o}^{\mathbf{T}}, \boldsymbol{x}_{c,\bar{o}}^{\mathbf{T}}, \boldsymbol{x}_{c,o}^{\mathbf{T}}]^{\mathbf{T}}$.

Minimum realization problems revisited

If all the initial states are zero, the output signal $\boldsymbol{y}(t)$ of a linear continuous state space model can be simplified to

$$\boldsymbol{y}(t) = \widehat{\boldsymbol{C}}_{c,o} \int_0^t \mathrm{e}^{\widehat{\boldsymbol{A}}_{c,o}(t-\tau)} \widehat{\boldsymbol{B}}_{c,o} u(\tau) \mathrm{d}\tau, \tag{3.12}$$

which is exactly the solution of the controllable/observable subspace. Therefore, in the Kalman decomposition form, the subspace $(\widehat{\boldsymbol{A}}_{c,o}, \widehat{\boldsymbol{B}}_{c,o}, \widehat{\boldsymbol{C}}_{c,o})$ is referred to as the minimum realization of the original system. That is, the minimum realized model is always fully controllable and observable. For transfer function models, the "minimum realized model" is the one in which all the pole-zero pairs have the same value canceled out.

The procedures to obtain the minimum realization of a linear system are summarized in the following:

1. Find a similarity transformation matrix $\boldsymbol{T}_c^{-1}$ to separate the controllable and uncontrollable parts:

$$\begin{aligned} \boldsymbol{A}_c = \boldsymbol{T}_c^{-1} \boldsymbol{A} \boldsymbol{T}_c &= \begin{bmatrix} \widehat{\boldsymbol{A}}_{\bar{c}} & \mathbf{0} \\ \widehat{\boldsymbol{A}}_{21} & \widehat{\boldsymbol{A}}_c \end{bmatrix}, \\ \boldsymbol{B}_c = \boldsymbol{T}_c^{-1} \boldsymbol{B} = \begin{bmatrix} \mathbf{0} \\ \widehat{\boldsymbol{B}}_c \end{bmatrix}, \quad \boldsymbol{C}_c = \boldsymbol{C} \boldsymbol{T}_c &= \begin{bmatrix} \widehat{\boldsymbol{C}}_{\bar{c}} & \widehat{\boldsymbol{C}}_c \end{bmatrix}; \end{aligned} \tag{3.13}$$

2. Find a transformation matrix $\widehat{\boldsymbol{T}}_o$ such that the controllable subsystem $(\widehat{\boldsymbol{A}}_c, \widehat{\boldsymbol{B}}_c, \widehat{\boldsymbol{C}}_c)$ can be further decomposed to find the observable part:

$$\begin{aligned} \widehat{\boldsymbol{A}}_o = \widehat{\boldsymbol{T}}_o^{-1} \widehat{\boldsymbol{A}}_c \widehat{\boldsymbol{T}}_o &= \begin{bmatrix} \widehat{\boldsymbol{A}}_{c,\bar{o}} & \widehat{\boldsymbol{A}}_{c,12} \\ \mathbf{0} & \widehat{\boldsymbol{A}}_{c,o} \end{bmatrix}, \\ \widehat{\boldsymbol{B}}_o = \widehat{\boldsymbol{T}}_o^{-1} \widehat{\boldsymbol{B}}_c = \begin{bmatrix} \widehat{\boldsymbol{B}}_{c,\bar{o}} \\ \widehat{\boldsymbol{B}}_{c,o} \end{bmatrix}, \quad \widehat{\boldsymbol{C}}_o = \widehat{\boldsymbol{C}}_c \widehat{\boldsymbol{T}}_o &= \begin{bmatrix} \mathbf{0} & \widehat{\boldsymbol{C}}_{c,o} \end{bmatrix}. \end{aligned} \tag{3.14}$$

3. Construct a matrix

$$\widetilde{\boldsymbol{T}}_o^{-1} = \begin{bmatrix} \boldsymbol{I}_{n-\,\mathrm{rank}\{\widehat{\boldsymbol{A}}_c\}} & \boldsymbol{0} \\ \boldsymbol{0} & \widehat{\boldsymbol{T}}_o^{-1} \end{bmatrix}.$$

Then, define the similarity transformation matrix $\boldsymbol{T}^{-1} = \widetilde{\boldsymbol{T}}_o^{-1}\boldsymbol{T}_c^{-1}$ to transform the original system into the minimum realized form $(\widehat{\boldsymbol{A}}_{c,o}, \widehat{\boldsymbol{B}}_{c,o}, \widehat{\boldsymbol{C}}_{c,o})$. Under the similarity transformation matrix $\boldsymbol{T}$, the whole system can be transformed into the canonical form as

$$\dot{\boldsymbol{z}} = \begin{bmatrix} \widehat{\boldsymbol{A}}_{\bar{c}} & & \boldsymbol{0} \\ \widehat{\boldsymbol{A}}_{21} & \widehat{\boldsymbol{A}}_{c,\bar{o}} & \widehat{\boldsymbol{A}}_{c,12} \\ \boldsymbol{0} & & \widehat{\boldsymbol{A}}_{c,o} \end{bmatrix} \boldsymbol{z} + \begin{bmatrix} \boldsymbol{0} \\ \widehat{\boldsymbol{B}}_{c,\bar{o}} \\ \widehat{\boldsymbol{B}}_{c,o} \end{bmatrix} \boldsymbol{u}, \qquad \boldsymbol{y} = [\boldsymbol{C}_{\bar{c}}\ \boldsymbol{0}\ \widehat{\boldsymbol{C}}_{c,o}]\boldsymbol{z} + \boldsymbol{D}\boldsymbol{u}. \quad (3.15)$$

Example 3.8. Revisit the state space model in Example 2.23. The above three steps can be implemented using the following MATLAB scripts to find the minimum realized model

```
>> A=[-5,8,0,0; -4,7,0,0; 0,0,0,4; 0,0,-2,6]; B=[4; -2; 2; 1];
   C=[2,-2,-2,2]; D=0; [Ac,Bc,Cc,Tc]=ctrbf(A,B,C);
   [Ao,Bo,Co,To]=obsvf(Ac,Bc,Cc); A_r=Ao(3:4,3:4); B_r=Bo(3:4);
   C_r=Co(3:4); G_r=zpk(ss(A_r,B_r,C_r,D))
```

The minimum realized model is obtained as $G_m(s) = 10(s-2.6)/[(s-2)(s+1)]$. It can be seen from the above that we did not find the canonical form as in (3.15), since the realized model has already been obtained and is the same as that given in Example 2.23.

3.1.4 Time Moments and Markov Parameters

Assume that the original transfer function $G(s)$ is described by

$$G(s) = \frac{b_1 s^k + b_2 s^{k-1} + \cdots + b_k s + b_{k+1}}{a_1 s^n + a_2 s^{n-1} + \cdots + a_n s + a_{n+1}}, \quad k \le n, \quad (3.16)$$

where for simplicity, it is assumed that $a_{n+1} = 1$.

Consider the initial and final value properties of the Laplace transformation. It is seen that the Taylor series, in particular the expansion around $s = 0$ and $s = \infty$, are useful in describing the steady-state response and initial transient of the system behavior.

Expansion around $s = 0$: the time moments

The Taylor series expansion of $G(s)$ around $s = 0$, or the Maclaurin series, can be written as

$$G(s) = \sum_{i=0}^{\infty} c_i s^i = c_0 + c_1 s + c_2 s^2 + \cdots. \quad (3.17)$$

If e^{-st} is expanded around $s = 0$ in the Laplace transformation of the impulse response function $g(t)$, $G(s)$ can be written as

$$G(s) = \int_0^\infty g(t)\mathrm{e}^{-st}\mathrm{d}t = \int_0^\infty g(t)\sum_{i=0}^{\infty}\frac{(-1)^i}{i!}(st)^i\mathrm{d}t = \sum_{i=0}^{\infty}\frac{(-1)^i}{i!}M_i s^i, \quad (3.18)$$

where $M_i = \int_0^\infty t^i g(t)\mathrm{d}t$ is referred to as the ith time moment of the impulse response function $g(t)$. From (3.17),

$$c_i = \frac{(-1)^i}{i!}M_i.$$

Assume that the state space model is given by $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$. The transfer function of the system can be equivalently obtained from

$$G(s) = \boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}. \quad (3.19)$$

The time moments c_i of the system can then be evaluated from

$$c_i = \frac{1}{i!}\frac{\mathrm{d}^i G(s)}{\mathrm{d}s^i}\bigg|_{s=0} = -\boldsymbol{C}\boldsymbol{A}^{-(i+1)}\boldsymbol{B}, \quad i = 0, 1, \ldots. \quad (3.20)$$

A MATLAB function `timmomt()` can be used to compute the time moments of a given LTI model object G:

```
function c=timmomt(G,k)
G=ss(G); C=G.c; B=G.b; iA=inv(G.a); iA1=iA;
c=zeros(1,k); for i=1:k, c(i)=-C*iA1*B; iA1=iA*iA1; end
```

The syntax of the function is `c=timmomt(G,k)`, where G is the LTI object model and k is the number of time moments to be evaluated, and the returned variable $\boldsymbol{c}$ is a vector containing the first k time moments.

Example 3.9. Consider a fourth-order model

$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24}.$$

The first seven time moments of the system can be obtained from the following MATLAB scripts:

```
>> G=tf([1,7,24,24],[1,10,35,50,24]);
   c=timmomt(G,7); [n,d]=rat(c)
```

which indicates that $G(s)$ can be approximated by the Taylor series expansion

$$G(s) = 1 - \frac{13}{12}s + \frac{157}{144}s^2 - \frac{609}{571}s^3 + \frac{899}{863}s^4 - \frac{128}{125}s^5 + \frac{386}{381}s^6 + o[s^7].$$

Expansion around $s = \infty$: the Markov parameters

The $G(s)$ given in (3.16) can be expanded as a power series of $1/s$, i.e.,

$$G(s) = \sum_{i=n-k}^{\infty}\delta_i\left(\frac{1}{s}\right)^i, \quad (3.21)$$

where the coefficients δ_i are referred to as the Markov parameters. Alternatively, it is equivalent to performing the Taylor series expansion of $G(s)$ around $s = \infty$. For state space model $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$, the Markov parameters δ_i can be evaluated from

$$\delta_0 = \boldsymbol{C}\boldsymbol{B} + \boldsymbol{D}, \quad \text{and } \delta_i = \boldsymbol{C}\boldsymbol{A}^i\boldsymbol{B}, \ \ i = 1, 2, \ldots. \tag{3.22}$$

Recall the properties of the Laplace transformation. It is easily seen that the time moments determine the steady-state time response of the system, while the Markov parameters determine the transient responses. In frequency response terms, the time moments determine the response over the low- and mid-frequency ranges, while Markov parameters determine the response over the mid- and high-frequency ranges.

A MATLAB function `markovp()` can be used to evaluate the Markov parameters of a given transfer function $G(s)$:

```
function m=markovp(G,k)
G=ss(G); A=G.a; B=G.b; C=G.c; D=G.d; m=[C*B+D,zeros(1,k-1)];
A1=A; for i=1:k-1, m(i+1)=C*A1*B; A1=A*A1; end
```

The syntax of the function is `m=markovp(G,k)`, where G is an LTI object and k is the number of Markov parameters to be evaluated. The returned variable $\boldsymbol{m}$ is a vector containing the first k Markov parameters.

Example 3.10. For the system in Example 3.9, the first seven Markov parameters can be evaluated using the following MATLAB statement:

```
>> M=markovp(G,7)
```

such that the Taylor series expansion about $s = \infty$ can be obtained as

$$G(s) = 1 - \frac{3}{s} + \frac{19}{s^2} - \frac{111}{s^3} + \frac{571}{s^4} - \frac{2703}{s^5} + \frac{12139}{s^6} + o\left[\frac{1}{s^7}\right].$$

3.1.5 Norm Measures of Signals and Systems

Robustness of a feedback control system is very important in control engineering practice. In actual control problems, there are always disturbances due to the environment and uncertainties due to the imperfect model used in the controller design. Clearly, it is desirable for the controlled system to have certain robustness against these disturbances and uncertainties. To assess the robustness, first of all, a proper measure is needed. Norm measures to signals and systems are introduced, which can be regarded as the basis of robust control.

Norm measures of signals

The size of a signal $u(t)$ is usually measured in its $\mathcal{L}_p$-norm defined as

$$\|u(t)\|_p = \left(\int_{-\infty}^{\infty} |u(t)|^p \mathrm{d}t\right)^{1/p}, \tag{3.23}$$

where p is a positive integer. The following norms are commonly used:

1. *The $\mathcal{L}_1$-norm:* $\|u(t)\|_1 = \int_{-\infty}^{\infty} |u(t)|\mathrm{d}t$.
2. *The $\mathcal{L}_2$-norm, the measure of signal power:* $\|u(t)\|_2 = \sqrt{\int_{-\infty}^{\infty} u^2(t)\mathrm{d}t}$.
3. *The $\mathcal{L}_\infty$-norm, the least upper bound of $|u(t)|$:* $\|u(t)\|_\infty = \sup_t |u(t)|$.

Norm measures of systems

The size of a system in a transfer function form is usually measured by its $\mathcal{H}_2$- and $\mathcal{H}_\infty$-norms.

1. The $\mathcal{H}_2$-norm is defined by

$$\|G(s)\|_2 = \sqrt{\frac{1}{2\pi\mathrm{j}} \int_{-\mathrm{j}\infty}^{\mathrm{j}\infty} |G(\mathrm{j}\omega)|^2 \mathrm{d}\omega}. \tag{3.24}$$

   The $\mathcal{H}_2$-norm is in fact a measure of the square root of the integral squared value of the output when the input is an impulse signal. In stochastic system terminology, the $\mathcal{H}_2$-norm is the root mean square value of the output signal when the input is white noise.

2. The $\mathcal{H}_\infty$-norm is defined by

$$\|G(s)\|_\infty = \sup_{u(t)\neq 0} \frac{\|y(t)\|_2}{\|u(t)\|_2}, \tag{3.25}$$

   where $u(t)$ and $y(t)$ are the input and output of the system, respectively. For stable systems, the $\mathcal{H}_\infty$-norm of the system can be computed from

$$\|G(s)\|_\infty = \sup_{\omega} |G(\mathrm{j}\omega)|. \tag{3.26}$$

   It is readily seen that the $\mathcal{H}_\infty$-norm is in fact the peak value of the magnitude of the frequency response.

   The symbols $\mathcal{L}$ and $\mathcal{H}$ are due to Lebesgue and Hardy, respectively.

Properties of $\mathcal{L}$- and $\mathcal{H}$-norms

Theorem 3.4. The following properties of norms are given without proofs:

1. $\|y(t)\|_2 \leq \|G(s)\|_\infty \|u(t)\|_2$.
2. $\|y(t)\|_\infty \leq \|G(s)\|_2 \|u(t)\|_\infty$.
3. $\|G_1(s)G_2(s)\|_\infty \leq \|G_1(s)\|_\infty \|G_2(s)\|_\infty$.

$\mathcal{H}_2$-norm and $\mathcal{H}_\infty$-norm evaluations

If the system model is given by an LTI object $\boldsymbol{G}$, the $\|\boldsymbol{G}(s)\|_2$ and $\|\boldsymbol{G}(s)\|_\infty$ norms of the system can be evaluated, respectively, from the MATLAB function calls `norm(G)` and `norm(G,inf)`. The norms of discrete-time systems can also be obtained with the same functions.

Example 3.11. Consider the discrete-time system

$$\begin{cases} \boldsymbol{x}[(k+1)T] = \begin{bmatrix} -2.2 & -0.7 & 1.5 & -1 \\ 0.2 & -6.3 & 6 & -1.5 \\ 0.6 & -0.9 & -2 & -0.5 \\ 1.4 & -0.1 & -1 & -3.5 \end{bmatrix} \boldsymbol{x}(kT) + \begin{bmatrix} 6 & 9 \\ 4 & 6 \\ 4 & 4 \\ 8 & 4 \end{bmatrix} \boldsymbol{u}(kT), \\ y(kT) = [1 \;\; 2 \;\; 3 \;\; 4]\boldsymbol{x}(kT). \end{cases}$$

The $\mathcal{H}_2$- and $\mathcal{H}_\infty$- norms of the discrete-time system can be evaluated directly with the following statements:

```
>> A=[-2.2,-0.7,1.5,-1; 0.2,-6.3,6,-1.5;
      0.6,-0.9,-2,-0.5; 1.4,-0.1,-1,-3.5];
   B=[6,9; 4,6; 4,4; 8,4]; C=[1 2 3 4]; G=ss(A,B,C,[0 0]);
   norm(G,2), norm(G,inf)
```

and it can be found that $||\boldsymbol{G}(z)||_2 = 32.8586$ and $||\boldsymbol{G}(z)||_\infty = 28.4423$.

3.2 Time Domain Analysis of Linear Systems

We remark that the time domain analytical solution to a linear system is always possible given a typical input signal. For general input signals, however, the time domain analysis has to be performed numerically.

3.2.1 Analytical Solutions to Continuous Time Responses

State space method

Consider an LTI system with its n-dimensional state space model

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) \end{cases} \quad \text{with initial condition } \boldsymbol{x}(0) = \boldsymbol{x}_0. \tag{3.27}$$

It has been stated in Chapter 2 that the time domain solution of (3.27) is

$$\boldsymbol{x}(t) = \mathrm{e}^{\boldsymbol{A}t}\boldsymbol{x}_0 + \int_0^t \mathrm{e}^{\boldsymbol{A}(t-\tau)}\boldsymbol{B}u(\tau)\mathrm{d}\tau, \tag{3.28}$$

where $\mathrm{e}^{\boldsymbol{A}t}$ is called the state transition matrix.

It can be seen from (3.28) that the most difficult part in the solution is the evaluation of the integral. If a certain transformation is introduced to remove the $\boldsymbol{B}$ term, the solution to the original problem can be significantly simplified.

Assume that the input signal is a unit step signal; then an extra state $x_{n+1}(t) = u(t)$ can be introduced. Clearly, $\dot{x}_{n+1}(t) = 0$. Thus, the state space equation can be rewritten as

$$\begin{bmatrix} \dot{\boldsymbol{x}}(t) \\ \dot{x}_{n+1}(t) \end{bmatrix} = \left[\begin{array}{c:c} \boldsymbol{A} & \boldsymbol{B} \\ \hdashline \boldsymbol{0} & \boldsymbol{0} \end{array}\right] \begin{bmatrix} \boldsymbol{x}(t) \\ x_{n+1}(t) \end{bmatrix}. \tag{3.29}$$

So, the original state space equation can be converted into an autonomous system

$$\begin{cases} \dot{\widetilde{\boldsymbol{x}}}(t) = \widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{x}}(t), \\ \widetilde{\boldsymbol{y}}(t) = \widetilde{\boldsymbol{C}}\widetilde{\boldsymbol{x}}(t), \end{cases} \tag{3.30}$$

where $\widetilde{\boldsymbol{x}}^{\mathrm{T}}(t) = [\boldsymbol{x}^{\mathrm{T}}(t), x_{n+1}(t)]$ and $\widetilde{\boldsymbol{x}}^{\mathrm{T}}(0) = [\boldsymbol{x}^{\mathrm{T}}(0), 1]$. The analytical solution can be easily found as

$$\widetilde{\boldsymbol{x}}(t) = \mathrm{e}^{\widetilde{\boldsymbol{A}}t}\widetilde{\boldsymbol{x}}(0). \tag{3.31}$$

A class of commonly used input signals, which can be converted into an autonomous system, is defined as

$$u(t) = u_1(t) + u_2(t) = \sum_{i=0}^{m} c_i t^i + \mathrm{e}^{d_1 t}\Big[d_2\cos(d_4 t) + d_3\sin(d_4 t)\Big]. \tag{3.32}$$

One may introduce some extra states, called augmented states, such that $x_{n+1} = \mathrm{e}^{d_1 t}\cos(d_4 t)$, $x_{n+2} = \mathrm{e}^{d_1 t}\sin(d_4 t)$, $x_{n+3} = u_1(t), \ldots, x_{n+m+3} = u_1^{(m-1)}(t)$. It can be shown that the augmented state space equations under such an input signal can be written as

$$\widetilde{\boldsymbol{A}} = \left[\begin{array}{c:cc:ccc} \boldsymbol{A} & d_2\boldsymbol{B} & d_3\boldsymbol{B} & \boldsymbol{B} & \boldsymbol{0} & \cdots \; \boldsymbol{0} \\ \hdashline \boldsymbol{0} & d_1 & -d_4 & & \boldsymbol{0} & \\ & d_4 & d_1 & & & \\ \hdashline & & & 0 & 1 & \cdots \; 0 \\ & & & 0 & 0 & \cdots \; 0 \\ \boldsymbol{0} & \multicolumn{2}{c:}{\boldsymbol{0}} & \vdots & \vdots & \ddots \; \vdots \\ & & & 0 & 0 & \cdots \; 0 \end{array}\right], \; \widetilde{\boldsymbol{x}}(t) = \begin{bmatrix} \boldsymbol{x}(t) \\ \hdashline x_{n+1}(t) \\ x_{n+2}(t) \\ \hdashline x_{n+3}(t) \\ x_{n+4}(t) \\ \vdots \\ x_{n+m+3}(t) \end{bmatrix}, \; \widetilde{\boldsymbol{x}}(0) = \begin{bmatrix} \boldsymbol{x}(0) \\ \hdashline 1 \\ 0 \\ \hdashline c_0 \\ c_1 \\ \vdots \\ c_m m! \end{bmatrix}, \tag{3.33}$$

whose analytical solution is

$$\widetilde{\boldsymbol{x}}(t) = \mathrm{e}^{\widetilde{\boldsymbol{A}}t}\widetilde{\boldsymbol{x}}(0). \tag{3.34}$$

A MATLAB function `ss_augment()` is written to establish the augmented state space model for the typical input signal:

```
function [Ga,Xa]=ss_augment(G,cc,dd,X)
G=ss(G); Aa=G.a; Ca=G.c; Xa=X; Ba=G.b; D=G.d;
if (length(dd)>0 & sum(abs(dd))>1e-5),
   if (abs(dd(4))>1e-5),
      Aa=[Aa dd(2)*Ba, dd(3)*Ba; ...
          zeros(2,length(Aa)), [dd(1),-dd(4); dd(4),dd(1)]];
      Ca=[Ca dd(2)*D dd(3)*D]; Xa=[Xa; 1; 0]; Ba=[Ba; 0; 0];
   else,
      Aa=[Aa dd(2)*B; zeros(1,length(Aa)) dd(1)];
      Ca=[Ca dd(2)*D]; Xa=[Xa; 1]; Ba=[B;0];
end, end
if (length(cc)>0 & sum(abs(cc))>1e-5), M=length(cc);
   Aa=[Aa Ba zeros(length(Aa),M-1); zeros(M-1,length(Aa)+1) ...
       eye(M-1); zeros(1,length(Aa)+M)];
   Ca=[Ca D zeros(1,M-1)]; Xa=[Xa; cc(1)]; ii=1;
   for i=2:M, ii=ii*i; Xa(length(Aa)+i)=cc(i)*ii;
end, end
Ga=ss(Aa,zeros(size(Ca')),Ca,D);
```

The syntax of the function is `[Ga,xx0]=ss_augment(G,c,d,x0)`, where the vectors $\boldsymbol{c} = [c_0, c_1, \ldots, c_m]$ and $\boldsymbol{d} = [d_1, d_2, d_3, d_4]$ are used to describe the input function $u(t)$ in (3.32). The arguments G and $\boldsymbol{x}_0$ are the model object and initial state vector, respectively, while the returned variables $\widehat{G}$ and $\widehat{\boldsymbol{x}}_0$ are, respectively, the augmented state-space model and its initial vector. Once the augmented system is established, the analytical solutions to the system can be easily obtained using the Symbolic Toolbox.

Example 3.12. Assume that a state space model is given by

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} -19 & -16 & -16 & -19 \\ 21 & 16 & 17 & 19 \\ 20 & 17 & 16 & 20 \\ -20 & -16 & -16 & -19 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \end{bmatrix} u(t), \\ y(t) = [2, 1, 0, 0]\, \boldsymbol{x}(t) \end{cases}$$

with initial states $\boldsymbol{x}^{\mathrm{T}}(0) = [0, 1, 1, 2]$. If the input signal is defined as $u(t) = 2 + 2\mathrm{e}^{-3t}\sin(2t)$, the function `ss_augment()` can be used to construct the augmented state space model:

```
>> cc=[2]; dd=[-3,0,2,2]; x0=[0; 1; 1; 2];
   A=[-19,-16,-16,-19; 21,16,17,19; 20,17,16,20;
      -20,-16,-16,-19];
   B=[1; 0; 1; 2]; C=[2 1 0 0]; D=0; G=ss(A,B,C,D);
   [Ga,xx0]=ss_augment(G,cc,dd,x0); Ga.a, xx0'
```

and the augmented model is

$$\dot{\widetilde{\boldsymbol{x}}}(t) = \left[\begin{array}{cccc:cc:c} -19 & -16 & -16 & -19 & 0 & 2 & 1 \\ 21 & 16 & 17 & 19 & 0 & 0 & 0 \\ 20 & 17 & 16 & 20 & 0 & 2 & 1 \\ -20 & -16 & -16 & -19 & 0 & 4 & 2 \\ \hdashline 0 & 0 & 0 & 0 & -3 & -2 & 0 \\ 0 & 0 & 0 & 0 & 2 & -3 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \widetilde{\boldsymbol{x}}(t), \quad \widetilde{\boldsymbol{x}}(0) = \left[\begin{array}{c} 0 \\ 1 \\ 1 \\ 2 \\ \hdashline 1 \\ 0 \\ \hdashline 2 \end{array}\right].$$

The following statements can be used to find the analytical solution of the system:

```
>> syms t; y=Ga.c*expm(Ga.a*t)*xx0;
```

The output signal of the system is obtained as

$$y(t) = -54 + \frac{127}{4}t\mathrm{e}^{-t} + 57\mathrm{e}^{-3t} + \frac{119}{8}\mathrm{e}^{-t} + 4t^2\mathrm{e}^{-t} - \frac{135}{8}\mathrm{e}^{-3t}\cos 2t + \frac{77}{4}\mathrm{e}^{-3t}\sin 2t.$$

Laplace transform method

Let us consider the equivalent transfer function model:

$$G(s) = \frac{b_1 s^m + b_2 s^{m-1} + \cdots + b_m s + b_{m+1}}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_{n-1} s + a_n}. \tag{3.35}$$

For any input signal $u(t)$ with $U(s)$ as its Laplace transform, the output signal can be obtained from $Y(s) = G(s)U(s)$. Thus, in order to find $y(t)$, an inverse Laplace transformation is needed such that $y(t) = \mathscr{L}^{-1}[Y(s)]$. The Symbolic Toolbox of MATLAB can be used to evaluate the Laplace transform of given input signals, and the inverse Laplace transform function can be used to evaluate the analytical solution of the system.

Example 3.13. Assume that

$$G(s) = \frac{s^3 + 7s^2 + 3s + 4}{s^4 + 7s^3 + 17s^2 + 17s + 6}$$

is the transfer function to be analyzed, and the input signal is given by $u(t) = 2 + 2\mathrm{e}^{-3t}\sin 2t$. The analytical solution to the output signal can be evaluated using the statements

```
>> syms s t;
   G=(s^3+7*s^2+3*s+4)/(s^4+7*s^3+17*s^2+17*s+6);
   u=2+2*exp(-3*t)*sin(2*t); U=laplace(u);
   y=ilaplace(G*U)
```

and the analytical solution can be written as

$$y(t) = \frac{4}{3} - \frac{31}{12}\mathrm{e}^{-3t} - \frac{23}{20}\mathrm{e}^{-3t}\cos 2t + \left(6 - \frac{21}{4}t\right)\mathrm{e}^{-t} - \frac{18}{5}\mathrm{e}^{-2t} - \frac{103}{40}\mathrm{e}^{-3t}\sin 2t.$$

3.2.2 Analytical Solutions to Discrete-Time Responses

Similar to the s-domain approach to the analytical solution for continuous systems, the Z transform can be used for discrete systems to evaluate the response to an input signal $U(z)$. Then, the analytical solution of the system $H(z)$ can be obtained by solving the inverse Z transform such that $y(n) = \mathscr{Z}^{-1}[H(z)U(z)]$.

Example 3.14. Assume that

$$G(z) = \frac{(z - 1/3)}{(z - 1/2)(z - 1/4)(z + 1/5)}$$

is a discrete-time transfer function of the system. Also assume that the input signal is a unit step signal. The analytical solution can be obtained using the statements

```
>> syms z; u=sym(1); U=ztrans(sym(u));
   H=(z-1/3)/(z-1/2)/(z-1/4)/(z+1/5);
   y=iztrans(H*U)
```

and the analytical solution can be written as

$$y(n) = \frac{40}{27} - \frac{80}{81}\left(\frac{1}{4}\right)^n + \frac{800}{567}\left(-\frac{1}{5}\right)^n - \frac{40}{21}\left(\frac{1}{2}\right)^n.$$

If the sampling interval T is given, the analytical solution can be rewritten as

$$y(kT) = \frac{40}{27} - \frac{80}{81}\left(\frac{1}{4}\right)^{kT} + \frac{800}{567}\left(-\frac{1}{5}\right)^{kT} - \frac{40}{21}\left(\frac{1}{2}\right)^{kT}.$$

3.3 Numerical Simulation of Linear Systems

The analytical solutions to linear systems were studied in the previous section. In real applications, one may prefer to have numerical solutions, and based on the results, the time domain responses can be plotted. Graphical visualization of system responses is usually more straightforward and informative for control engineers.

In this section, the numerical solution techniques to linear systems are presented, with a focus on some common responses such as step responses, impulse responses, and more generally, the time domain responses to arbitrary input signals.

3.3.1 Step Responses of Linear Systems

Step input signals and their responses are commonly used in control systems analysis and design. The typical step response of a second-order system is studied and specifications are given. Then, MATLAB-based evaluation of step responses are given.

Second-order system analysis

In classical control courses, second-order systems are often used as an example, where many properties of the linear control systems are illustrated.

Theorem 3.5. The closed-loop unit step response of a second order system

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

can be obtained easily by considering the following four cases:

1. When $\zeta = 0$, the step response is $y(t) = 1 - \cos(\omega_n t)$.
2. When $0 < \zeta < 1$, the step response is

$$y(t) = 1 - \mathrm{e}^{-\zeta\omega_n t}\frac{1}{\sqrt{1-\zeta^2}}\sin(\omega_d t + \theta),$$

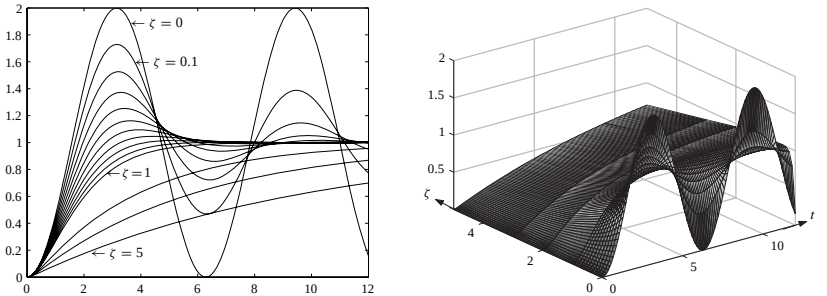
   where $\theta = \tan^{-1}\sqrt{1-\zeta^2}/\zeta$ and $\omega_d = \omega_n\sqrt{1-\zeta^2}$.
3. When $\zeta = 1$, the step response is $y(t) = 1 - (1+\omega_n t)\mathrm{e}^{-\omega_n t}$.
4. When $\zeta > 1$, the step response is

$$y(t) = 1 - \frac{\omega_n}{2\sqrt{\zeta^2-1}}\left(\frac{\mathrm{e}^{\lambda_1 t}}{\lambda_1} - \frac{\mathrm{e}^{\lambda_2 t}}{\lambda_2}\right),$$

   where $\lambda_1 = -\zeta - \sqrt{\zeta^2-1}$, $\lambda_2 = -\zeta + \sqrt{\zeta^2-1}$.

Example 3.15. With the use of the powerful Symbolic Toolbox, the analytical solutions to a second-order system can be easily derived:

```
>> syms z s, syms wn positive
   y=ilaplace(wn^2/(s*(s^2+2*z*wn*s+wn^2)))
```

(a) step response (b) three-dimensional representation

Figure 3.5. *Step responses of second-order systems.*

and it follows immediately from the results, with obvious simplifications, that

$$y(t) = 1 - \omega_n\mathrm{e}^{-\zeta\omega_n t}\left[\frac{\cosh(\omega_d t)}{\omega_n} + \frac{\zeta\sinh(\omega_d t)}{\omega_d}\right], \tag{3.36}$$

where $\omega_d = \sqrt{\zeta^2-1}\omega_n$. It can be seen that the format of the new results is much more concise than that given in Theorem 3.5 if complex variables are allowed. The only restriction in (3.36) is that $\zeta \neq 1$; otherwise zero may be used in the denominator. One may avoid this particular case by defining $\zeta = 1+\epsilon$, where ϵ is a very small number, and the problem can be solved successfully.

The step response of the system can be evaluated easily as shown in Figure 3.5(a), and the three-dimensional version is shown in Figure 3.5(b).

The following MATLAB commands show the step responses:

```
>> wn=1; yy=[]; t=0:.1:12; zet=[0:0.1:0.9, 1+eps,2,3,5];
   for z=zet
      wd=sqrt(z^2-1)*wn;
      y=1-wn*exp(-z*wn*t).*[cosh(wd*t)/wn+z*sinh(wd*t)/wd];
      yy=[yy; y];
   end
   plot(t,yy), figure, surf(t,zet,yy)
```

It can be seen that when $\zeta = 0$, the output is an undamped oscillation. When ζ is smaller than 1, there exists a damped oscillation and, with an increase in the value of ζ, the oscillation tends to be less and the overshoot becomes smaller. When the value of ζ is equal to or greater than 1, there exists no oscillation in the output signal. With the increment of the value ζ, it may take longer to approach, but will never exactly reach the desired steady-state value. The behavior of the output signal versus the changes of value ζ can be better observed on the three-dimensional surface in Figure 3.5(b).

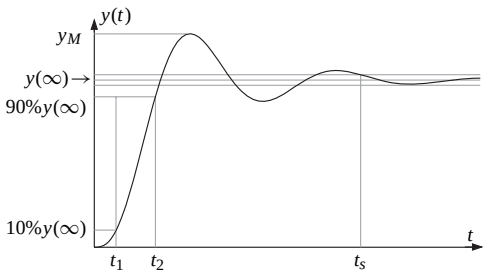


Figure 3.6. *Typical step response specifications.*

Quantitative specifications in step responses

From the typical step response curves, several useful quantitative specifications are defined and shown in Figure 3.6. Details of these commonly used specifications are summarized below:

1. *The steady-state value* $y(\infty)$: The steady-state value of the system under the step response is the output when $t \to \infty$. For a transfer function model, using the final value property of the Laplace transformation, the steady-state value of the system can be easily obtained from

$$y(\infty) = \lim_{s\to 0} sG(s)\frac{1}{s} = G(0) = \frac{b_m}{a_n}. \tag{3.37}$$

   If the system is given with state space model $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$, the steady-state value of the system can be obtained from

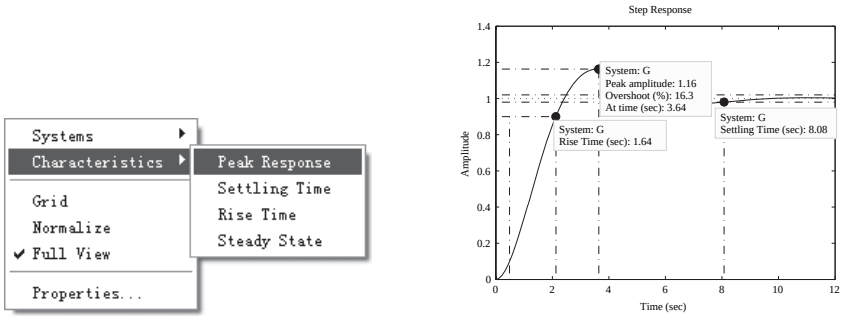
$$y(\infty) = \lim_{s\to 0} sG(s)\frac{1}{s} = -\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B} + \boldsymbol{D}. \tag{3.38}$$

   The steady-state value of the system G can be evaluated using the function

```
K=dcgain(G)
```

   provided in the Control Systems Toolbox.
2. *The rise time* t_r: The rise time is defined as $t_r = t_2 - t_1$ with t_2 and t_1, respectively, the time when $y(t)$ reaches 90% and 10% of its steady-state value.
3. *The settling time* t_s: When the output signal $y(t)$ enters and is kept within the range of $[y(\infty) - \Delta y, y(\infty) + \Delta y]$, the moment $y(t)$ enters the range is referred to as the settling time. According to different definitions, Δy can be defined as either 2% or 5% of the steady-state value $y(\infty)$.
4. *The overshoot* M_p *and the peak value* y_M: M_p is also known as the percent overshoot which is defined as

$$M_p = \frac{y_M - y(\infty)}{y(\infty)} \times 100\%. \tag{3.39}$$



(a) pop up menu (b) with specifications

Figure 3.7. *Step response with specifications.*

In control systems design, one often expects to design a system which has short rise time and settling time, with a small percentage of overshoot or no overshoot.

Example 3.16. Consider the second-order system with $\zeta = 0.5$ and $\omega_n = 1$ rad./sec. The `step()` function, which will be described later, provided in the Control Systems Toolbox can be used directly to draw the step response curve

```
>> z=0.5; wn=1; G=tf(wn^2,[1,2*z*wn,wn^2]); step(G).
```

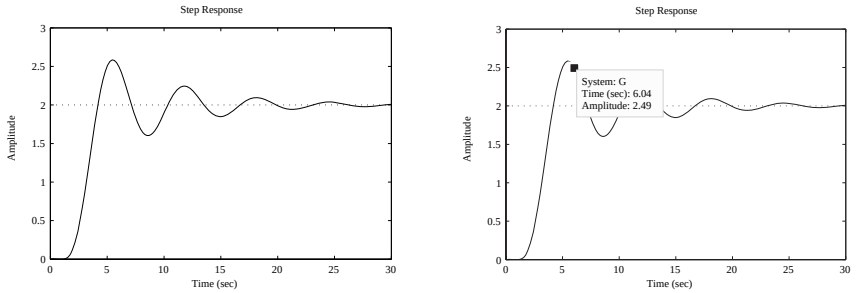
Right click on the window that appears to see the pop-up menu shown in Figure 3.7(a). One may select different specifications from the menu, and the corresponding specification will be superimposed on the step response curve, as shown in Figure 3.7(b).

Step response evaluations with MATLAB

The step response of linear systems can be evaluated and drawn using the function `step()`, and the function can be called with a variety of syntaxes:

```
step(G)                % automatic draw of step response curves
[y,t]=step(G)          % evaluate the responses, but not drawn
[y,t]=step(G,tf)       % final simulation time tf setting
y=step(G,t)            % simulation on user defined time vector t
[y,t,x]=step(G)        % state x is returned, if G is state space
```

In the function call, G is an LTI model object, which can be either a transfer function, state space, or pole-zero-gain model. This function applies to continuous systems and discrete-time systems. It can also be used for SISO and MIMO systems and systems with or without time delays. Thus the function provides a unified way of finding the step response of linear systems. If no argument is returned in the function call, the step response will be drawn automatically. If the response data are returned as output arguments, there will be no response drawn. The data can be drawn later with the `plot()` function. However, the plain curves drawn by `plot()` may lose many useful properties, such as the pop-up menu shown in Figure 3.7(a).



(a) automatically drawn step response (b) get the response information

Figure 3.8. *Step response of a delayed continuous system.*

The step response of more than one system model, for instance, G_1, G_2, and G_3, can be drawn under the same coordinate if the function is called as follows:

`step(`G_1`,'-',`G_2`,'-.b',`G_3`,':r')`,

where the options are the same as the conventional `plot()` function options. In the curves, the step response G_1 is shown by the solid line, for G_2 by the dashed-dotted blue lines, and for G_3 by the red dotted lines.

Example 3.17. If a continuous model

$$G(s)=\frac{10s+20}{10s^4+23s^3+26s^2+23s+10}e^{-s}$$

is a transfer function which contains a time delay, the following statements can be used to enter the system model and draw the step response as shown in Figure 3.8(a).

```
>> G=tf([10 20],[10 23 26 23 10],'ioDelay',1); % model input
   step(G,30); % step response with terminate time of 30 sec.
```

On the step response drawn, if one left clicks a point on the curve, the magnitude and time will be displayed on the response curve, as shown in Figure 3.8(b). The overshoot, settling time, and other specifications can be easily displayed on the curve, using the method shown previously. One may easily investigate the response of the system with these flexible auxiliary facilities.

Example 3.18. If a system is given by

$$G(s)=\frac{1}{s^2+0.2s+1}e^{-s},$$

with the sampling interval of $T = 0.01, 0.1, 0.5, 1.2$ seconds, respectively, the following statements can be used to obtain the discrete-time models for different sampling intervals, and the step responses are obtained as shown in Figure 3.9. It can be seen that the original system information may be lost if the sampling interval is selected to be too large.

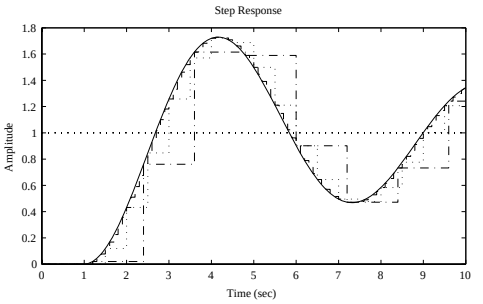


Figure 3.9. *Step response comparisons of discretized systems.*

```
>> G=tf(1,[1 0.2 1],'ioDelay',1); G1=c2d(G,0.01,'zoh');
   G2=c2d(G,0.1); G3=c2d(G,0.5); G4=c2d(G,1.2);
   step(G,'-',G2,'--',G3,':',G4,'-.',10)
```

The discrete-time models thus obtained are, respectively,

$$G_1(z)=\frac{4.997\times10^{-5}z+4.993\times10^{-5}}{z^2-1.998z+0.998}z^{-100},\qquad G_2(z)=\frac{0.004963z+0.00493}{z^2-1.97z+0.9802}z^{-10}$$

$$G_3(z)=\frac{0.1185z+0.1145}{z^2-1.672z+0.9048}z^{-2},\qquad G_4(z)=\frac{0.01967z^2+0.7277z+0.3865}{z^3-0.6527z^2+0.7866z}.$$

It should be noted that the step response curve of the discrete-time system is automatically drawn in the stairs format. One can still read the response data and specifications by clicking the points on the curves.

Example 3.19. Consider the system given in Example 2.4, which has two inputs and two outputs. The following statements can be used and the step response of the multivariable system obtained as shown in Figure 3.10.

```
>> g11=tf(0.1134,[1.78 4.48 1],'ioDelay',0.72);
   g12=tf(0.924,[2.07 1]);
   g21=tf(0.3378,[0.361 1.09 1],'ioDelay',0.3);
   g22=tf(-0.318,[2.93 1],'ioDelay',1.29);
   G=[g11, g12; g21, g22]; step(G)
```

The first column of the curves contain the outputs of the system when the first channel of inputs acts alone. The curves in the second column are the step response of the system, if the second channel of input acts alone. From the step response curves, the interactions between the input output pairs can easily be found.

3.3.2 Impulse Responses of Linear Systems

The impulse responses of the system can be drawn easily with the `impulse()` function provided in the Control Systems Toolbox, and the syntaxes of the function are exactly the same as the `step()` function given earlier.

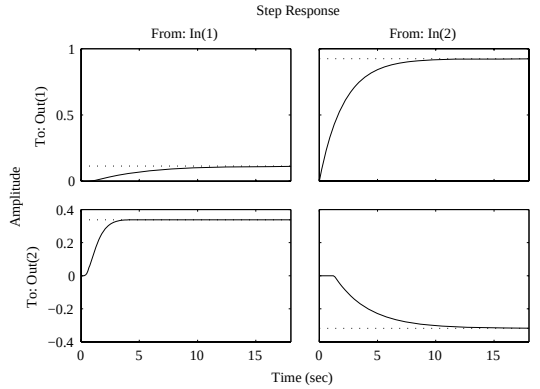


Figure 3.10. *Step response of a multivariable system.*

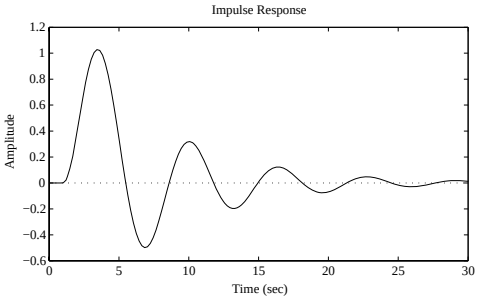


Figure 3.11. *Impulse response of the system.*

Example 3.20. Consider again the system model studied in Example 3.17. The impulse response of the system can be obtained as shown in Figure 3.11:

```
>> G=tf([10 20],[10 23 26 23 10],'ioDelay',1); impulse(G, 30);
```

3.3.3 Time Responses to Arbitrary Inputs

In the previous discussion, two types of input signals were studied. Here, two other types of signals will be studied.

If the Laplace transform $R(s)$ of the input signal can be written as a rational function, the output of the system can be expressed as $Y(s) = G(s)R(s)$, which is also rational. Thus, the time response under $R(s)$ can be equivalently evaluated with the `impulse()` function if $Y(s)$ is assumed to be the transfer function of a system.

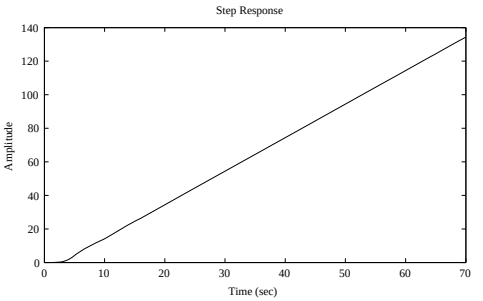


Figure 3.12. *Ramp response of the system in Example* 3.21.

Example 3.21. Consider again

$$G(s) = \frac{10s + 20}{10s^4 + 23s^3 + 26s^2 + 23s + 10} e^{-s}.$$

The ramp response of the system can be obtained with the help of the `impulse()` function.

It is known that the Laplace transform of a ramp function is $1/s^2$; then the ramp response of the system can be evaluated as either the step response of system $G(s)/s$ or the impulse response of the system $G(s)/s^2$. The following statements can then be used to evaluate the ramp response of the system, as shown in Figure 3.12.

```
>> G=tf([10 20],[10 23 26 23 10],'ioDelay',1);
   s=tf('s'); step(G/s);  % or use impulse(G/s^2)
```

If the input signal cannot be expressed by mathematical equations, or the Laplace transform cannot be a rational function, the above methods cannot be used. In this case, the function `lsim()` can be used to evaluate the time domain response of the system. The syntax of the function `lsim()` is similar to the `step()` function, and the difference is that the input vector $\boldsymbol{u}$ should be used such that `lsim(`$\boldsymbol{G}$`,`$\boldsymbol{u}$`,`$\boldsymbol{t}$`)`.

Example 3.22. Consider the multivariable system given in Example 2.4, where the two inputs are defined as $u_1(t) = 1 - e^{-t}\sin(3t+1)$ and $u_2(t) = \sin(t)\cos(t+2)$. The system response can then be evaluated with the following statements, and the system responses are shown in Figure 3.13, where the dotted curves represent the two input signals.

```
>> g11=tf(0.1134,[1.78 4.48 1],'ioDelay',0.72);
   g12=tf(0.924,[2.07 1]);
   g21=tf(0.3378,[0.361 1.09 1],'ioDelay',0.3);
   g22=tf(-0.318,[2.93 1],'ioDelay',1.29);
   G=[g11, g12; g21, g22]; t=[0:.1:15]';
   u=[1-exp(-t).*sin(3*t+1),sin(t).*cos(t+2)]; lsim(G,u,t);
```

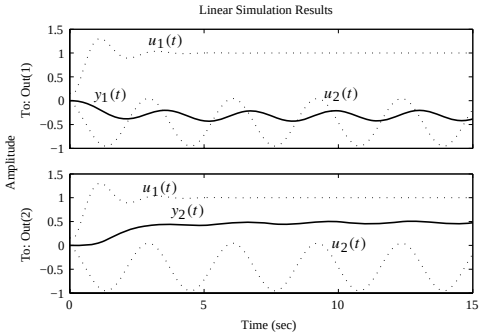


Figure 3.13. *Time domain response of a multivariable system.*

3.4 Root Locus of Linear Systems

Assume that the feedback control system is established by a unity negative feedback system whose forward path is defined as a static gain K, followed by an open-loop model $G(s)$. For each value of K, a set of closed-loop poles can be found by solving the characteristic equation $1+KG(s)=0$. With continuous change in the gain K, the trajectories of closed-loop pole positions can be constructed. The trajectories of the poles of the closed-loop system can be obtained and are referred to as a root locus of the system. It should be noted that the open-loop model $G(s)$ should be used to draw the root locus, and the root locus can be used to describe the pole positions of the closed-loop system.

A MATLAB function `rlocus()` is provided in the Control Systems Toolbox to draw the root locus of a given system. The function can be called in one of the following ways:

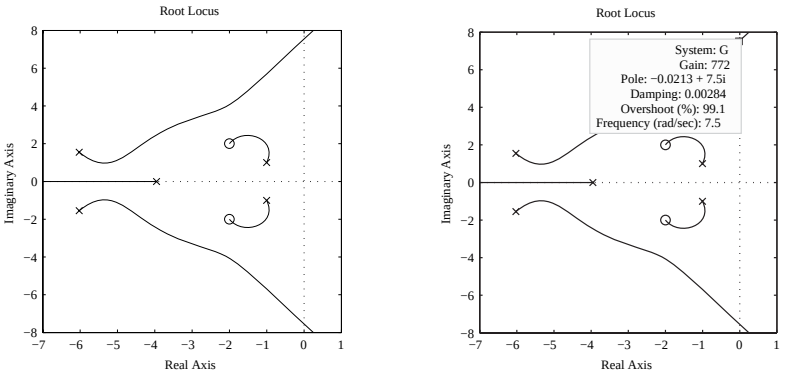
```
rlocus(G)          % automatic draw of the root locus
rlocus(G,[kmin,kmax])          % root locus over the gain range
rlocus(G,K)          % root locus for a given gain vector K
[R,K]=rlocus(G)          % evaluate the closed-loop pole positions R
rlocus(G1,'-',G2,'-.b',G3,':r')      % root locus for several models
```

It should be noted that this function applies to both continuous- and discrete-time systems. Only SISO LTI models can be processed in the function. It can also be used in drawing the root locus for discrete-time transfer functions with pure time delays.

On the root locus of the system, one may left click any point on the locus to show the gain, pole position, damping ratio, and overshoot of the system at the same gain K. One can easily find the values of the open-loop gain K for which the closed-loop system is stable. The command `grid` can be used to superimpose the isodamping and isofrequency lines of the system. These lines may provide useful information in control systems design.

Example 3.23. Let

$$G(s)=\frac{s^2+4s+8}{s^5+18s^4+120.3s^3+357.5s^2+478.5s+306}$$



(a) root locus (b) find the critical point

Figure 3.14. *Root locus analysis of the system and its inverse.*

be an open-loop model of the system under investigation. Using the following MATLAB scripts, the root locus of the system can be easily and accurately drawn, as shown in Figure 3.14(a).

```
>> num=[1 4 8]; den=[1,18,120.3,357.5,478.5,306];
   G=tf(num,den); rlocus(G)
```

If one left clicks at the point on the intersection with the imaginary axis, the information about the critical point is shown as in Figure 3.14(b), from which it is immediately seen that the critical gain is 772. It can be concluded that when the gain $K>772$, the closed-loop system is unstable.

Example 3.24. If a discrete-time open-loop model is given by

$$G(z)=\frac{0.52(z-0.49)(z^2+1.28z+0.4385)}{(z-0.78)(z+0.29)(z^2+0.7z+0.1586)}$$

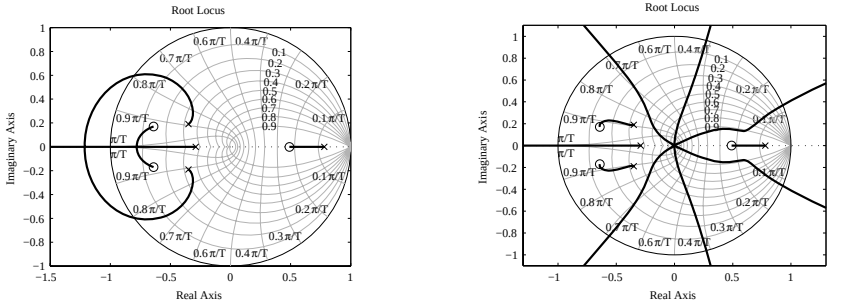
with a sampling interval of $T=0.1$ seconds, the following statements can be used to input the open-loop system model and draw the root locus of the system, as shown in Figure 3.15(a). It can be seen by clicking the relevant points that the critical gain is $K=2.83$.

```
>> z=tf('z','Ts',0.1);
   G=0.52*(z-0.49)*(z^2+1.28*z+0.4385)/(z+0.29)/(z^2+0.7*z+0.1586);
   rlocus(G), grid
```

If there exists a pure delay term z^{-6} in the original system, the root locus of the delayed system can be redrawn, as shown in Figure 3.15(b):

```
>> G.ioDelay=6; rlocus(G), grid
```

It can be found that the critical gain is reduced to 1.16. It can be seen from the example that the delay term in the discrete-time model reduces the critical gain of the system.



(a) root locus (b) root locus of the delayed system

Figure 3.15. *Root locus of a discrete-time system.*

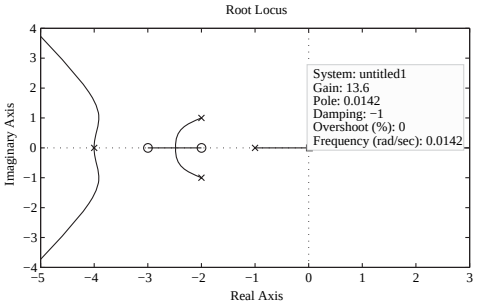


Figure 3.16. *Root locus for positive feedback systems.*

Example 3.25. If

$$G(s) = \frac{s^2 + 5s + 6}{s^5 + 13s^4 + 65s^3 + 157s^2 + 184s + 80}$$

is an open-loop model, the following statements can be used to draw the root locus for the system with unity positive feedback, as shown in Figure 3.16. It can be seen that when $0 \leq K \leq 13.6$, the closed-loop system is stable.

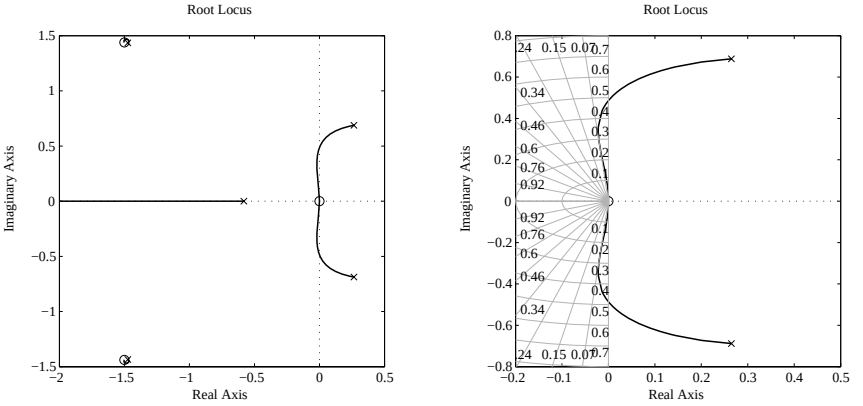
```
>> G=tf([1 5 6],[1 13 65 157 184 80]); rlocus(-G)
```

Example 3.26. For the open-loop model

$$G(s) = \frac{0.3(s+2)(s^2+2.1s+2.23)}{s^2(s^2+3s+4.32)(s+a)},$$

if one wants to draw the root locus according to variable a, the characteristic equation $1 + G(s) = 0$ can be rewritten as

$$a(s^4 + 3s^3 + 4.32s^2) + (s^5 + 3s^4 + 4.62s^3 + 1.23s^2 + 1.929s + 1.338) = 0$$



(a) root locus (b) zoomed root locus

Figure 3.17. *Root locus according to variable a.*

from which it can be seen that

$$1 + a\frac{s^4 + 3s^3 + 4.32s^2}{s^5 + 3s^4 + 4.62s^3 + 1.23s^2 + 1.929s + 1.338} = 0.$$

Let

$$\widehat{G}(s) = \frac{s^4 + 3s^3 + 4.32s^2}{s^5 + 3s^4 + 4.62s^3 + 1.23s^2 + 1.929s + 1.338}.$$

The characteristic equation can be written as $1 + a\widehat{G}(s) = 0$. The root locus according to variable a can be drawn for the $\widehat{G}(s)$ model. The following statements can be given, and the root locus obtained is shown in Figure 3.17(a), and Figure 3.17(b) is the zoomed version:

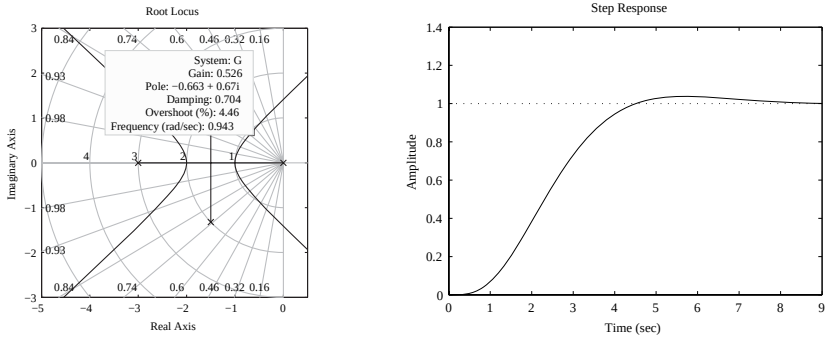
```
>> G1=tf([1,3,4.32,0,0],[1,3,4.62,1.23,1.929,1.338]); rlocus(G1)
```

The root locus can be used in controller design to select an appropriate value for the gain K. If there exists a pair of dominant complex poles on the root locus, which have a relatively low damping ratio for a specific value of K, then selecting this value of K may be appropriate. It is assumed that if the complex poles are dominant and the effects of any zeros can be ignored, then the resulting closed-loop response will approximate that of a second-order system with these complex poles.

Example 3.27. Consider the open-loop model

$$G(s) = \frac{10}{s(s+3)(s^2+2s+4)}.$$

The following statements can be entered into MATLAB, and the root locus of the system can be drawn as shown in Figure 3.18(a). The isodamping lines are also superimposed on the curves.



(a) root locus (b) closed-loop step response

Figure 3.18. *Root locus and step response of the system in Example* 3.27.

```
>> s=tf('s'); G=10/(s*(s+3)*(s^2+3*s+4));
   rlocus(G), grid
```

From the isodamping lines, it is easily found by clicking the pole position located at the $\zeta = 0.707$ line that the gain is about $K = 1.68$, as shown in Figure 3.18(a). It is also seen that this pair of poles is dominant, so selecting the gain $K = 1.68$ gives the closed-loop step response shown in Figure 3.18(b). This, as expected, is very similar to that of the second-order system with these poles.

```
>> K=1.68; step(feedback(G*K,1))
```

Example 3.28. Consider a simple plant model $G(s) = 1/(s+1)^3$. The root locus of the system can immediately be drawn as shown in Figure 3.19(a) with the following statements:

```
>> s=tf('s'); G=1/(s+1)^3; rlocus(G)
```

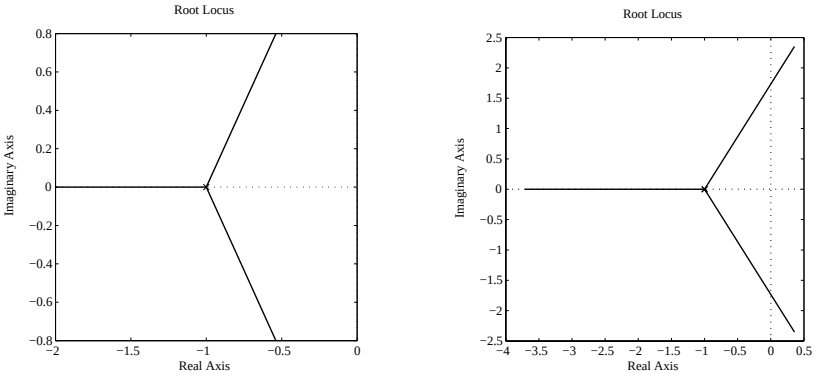
However, the root locus drawn with default settings is not complete, since the intersection of the root locus with the imaginary axis is not shown, due to the improper selection of the gain range, by default. One should then enlarge the gain range, for instance, select a range of $(0, 20)$, to modify the root locus drawn. The modified root locus can be obtained as shown in Figure 3.19(b).

```
>> rlocus(G,[0,20])
```

Example 3.29. Assume that

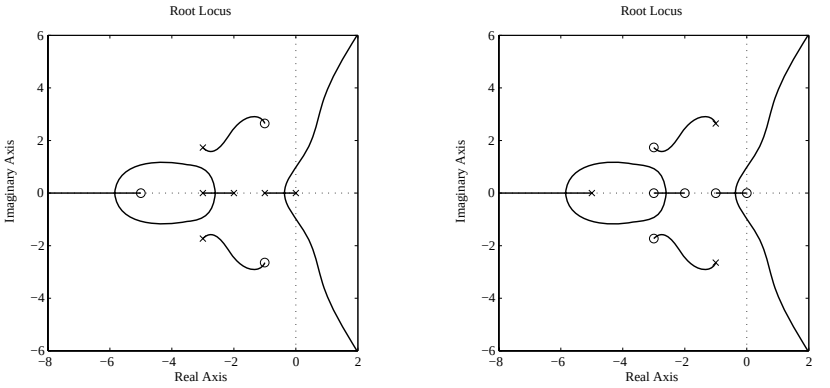
$$G(s) = \frac{(s+5)(s^2+2s+8)}{s(s+1)(s+2)(s+3)(s^2+6s+12)}$$

is an open-loop model. The root locus of the system is immediately obtained, as shown in Figure 3.20(a).



(a) default root locus (b) modified root locus

Figure 3.19. *Problems in automatic root locus drawing.*



(a) root locus (b) root locus of inverse system

Figure 3.20. *Root locus of a system and its inverse system.*

```
>> s=tf('s');
   G=(s+5)*(s^2+2*s+8)/s/(s+1)/(s+2)/(s+3)/(s^2+6*s+12)
```

It is interesting to note that the root locus of its inverse system $1/G(s)$ has exactly the same shape as the original $G(s)$ if it is drawn with the command `rlocus(1/G)`, shown in Figure 3.20(b). In the inverse system, it is not surprising to note that the poles and zeros are interchanged, and thus the directions of the root loci are all reversed.

Besides, if one reads the gain at a certain point on Figure 3.20(a), one will get the reciprocal of the gain by clicking the same point on Figure 3.20(b).

3.5 Frequency Domain Analysis of Linear Systems

Frequency domain analysis methods make up a class of very important methods in control systems analysis and design. In 1932, Nyquist presented a graphical method which can be used to assess the stability of a control system. Within a few years a frequency domain analysis and design framework had been set up. It was found to be a very useful approach because component models were often available as frequency response data.

3.5.1 Frequency Domain Graphs with MATLAB

For a linear transfer function $G(s)$, if the frequency domain variable $\mathrm{j}\omega$ is used to substitute for the complex variable s, then $G(\mathrm{j}\omega)$ can be regarded as the "complex gain" of the system, which is complex and a function of the frequency ω. There are many different ways of describing the complex quantity $G(\mathrm{j}\omega)$. Based on these descriptions, different frequency domain methods can be established as follows.

1. *Real and imaginary part representation*: The complex gain can be represented as the real part and imaginary part, such that

$$G(\mathrm{j}\omega) = P(\omega) + \mathrm{j}Q(\omega), \tag{3.40}$$

and it can be seen that $P(\omega)$ and $Q(\omega)$ are functions of frequency ω. If the horizontal axis is used to represent the real part and the vertical axis the imaginary part, the trajectory of the complex gain $G(\mathrm{j}\omega)$ is referred to as a Nyquist plot. A drawback of the traditional Nyquist plot is that the frequency dependence of the locus can only be found if a limited number of points on the locus have their frequency marked.

A MATLAB function `nyquist()` provided in the Control Systems Toolbox can be used to draw the Nyquist plot of LTI model G. The syntaxes of the function are

```
nyquist(G)          % automatic drawing of Nyquist plot
nyquist(G,{ωm,ωM})  % draw Nyquist plot over range (ωm,ωM)
nyquist(G, ω)       % draw Nyquist plot over frequency vector ω
[R, I, ω]=nyquist(G)        % Nyquist response data evaluation
nyquist(G1,'-',G2,'-.b',G3,':r')        % several systems
```

If one left clicks a point on the Nyquist plot drawn, the frequency information can be displayed, together with the values of the complex gain. This facility provides an extremely useful tool in the analysis and design of linear systems. The overloaded command `grid` can be used to superimpose iso-M circles on top of the Nyquist curve.

2. *Magnitude and phase representation*: A complex quantity $G(\mathrm{j}\omega)$ can be expressed in magnitude and phase form, that is,

$$G(\mathrm{j}\omega) = A(\omega)\mathrm{e}^{-\mathrm{j}\phi(\omega)}. \tag{3.41}$$

Thus, the frequency ω can be used as a horizontal axis, and the magnitude $A(\omega)$ and phase $\phi(\omega)$ can be used separately as vertical axes. A new set of diagrams can be constructed. If frequency is plotted on a logarithmic scale, the magnitude M plotted in decibels (dB), that is, $M(\omega) = 20\lg[A(\omega)]$, and the phase is in degrees, the diagram is referred to as a Bode diagram.

The function `bode()` is provided in the Control Systems Toolbox, and it can used for drawing the Bode diagram of the linear system G. The syntaxes of the function are

```
bode(G)          % automatic draw of the Bode diagram
bode(G,{ωm,ωM})  % draw Bode diagram over range (ωm,ωM)
bode(G,ω)        % draw Bode diagram over frequency vector ω
[A,φ,ω]=bode(G)  % Bode diagram data evaluation
bode(G1,'-',G2,'-.b',G3,':r')        % several systems
```

3. *Magnitude and phase representation in a single plot*: The magnitude and phase representation to the complex gains is again used. Selecting magnitude and phase as vertical and horizontal axes, respectively, we will get the well-known Nichols chart.

The function `nichols()` provided in the Control Systems Toolbox can be used to draw the Nichols chart for the given system G, and the `grid` command can be used to superimpose the constant M and N contours on the chart.

For discrete-time systems $H(z)$, one may substitute $z = \mathrm{e}^{\mathrm{j}\omega T}$ into the transfer function model such that the relationship between the complex magnitude $\widehat{H}(\mathrm{j}\omega)$ and the frequency can be established. The above-mentioned `nyquist()`, `bode()`, and `nichols()` can be used in discrete-time models directly.

Example 3.30. Consider the continuous model

$$G(s) = \frac{s+8}{s(s^2+0.2s+4)(s+1)(s+3)}.$$

The Nyquist plot can be easily drawn with the following statements, with superimposed contours:

```
>> s=tf('s'); G=(s+8)/(s*(s^2+0.2*s+4)*(s+1)*(s+3));
   nyquist(G), grid, set(gca,'Ylim',[-1.5 1.5])
```

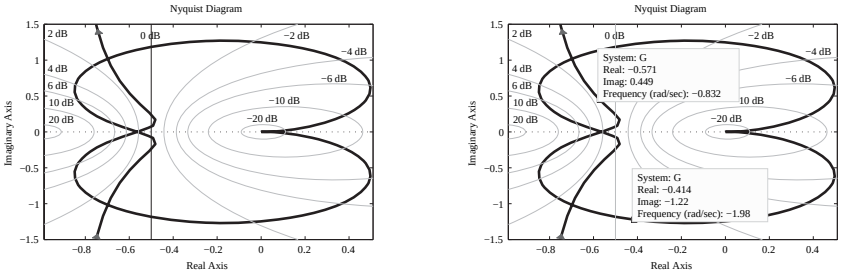
Since one of the open-loop poles is located at $s = 0$, the magnitude may be very large at low frequencies ω, and thus sometimes manual selection of the magnified range of the plots should be used. For instance, the Nyquist plot of the above system under manual zooming is shown in Figure 3.21(a).

With the Nyquist plot drawn by MATLAB, one may left click a point on the Nyquist plot and the frequency information is displayed, together with accurate values of the axis coordinates, as shown in Figure 3.21(b). This new facility is very useful in the analysis and design of control systems.

If one wants to have the Bode diagram and Nichols chart, the following statements can be used, and the Bode diagram and Nichols chart can be displayed as shown in Figure 3.22.

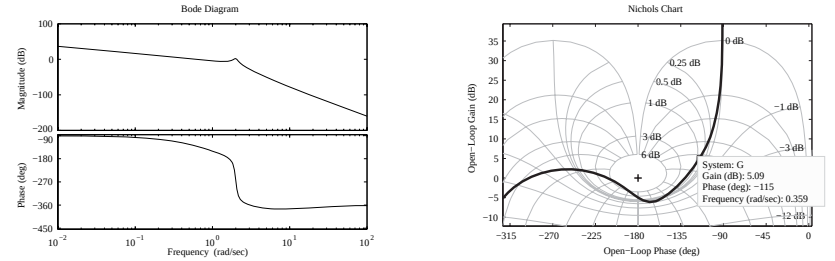
```
>> bode(G); figure; nichols(G), grid
```

One may right click on the curves to get further facilities to analyze the curves. For instance, if one right clicks the curves on the Bode diagram, a shortcut menu will appear, and the Characteristics item is displayed as shown in Figure 3.23(a). One may further



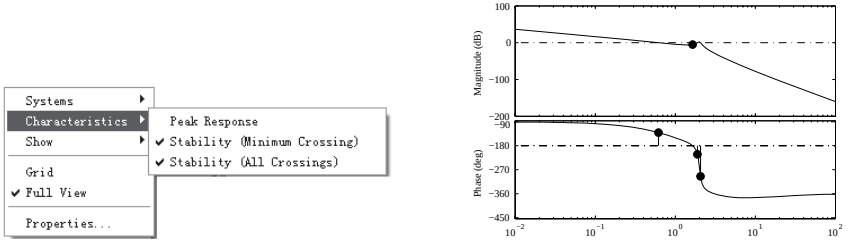
(a) Nyquist plot (b) Extra data from Nyquist plot

Figure 3.21. *Nyquist plot analysis of the system in Example* 3.30.



(a) Bode diagram (b) Nichols chart

Figure 3.22. *Frequency domain analysis of the system in Example* 3.30.

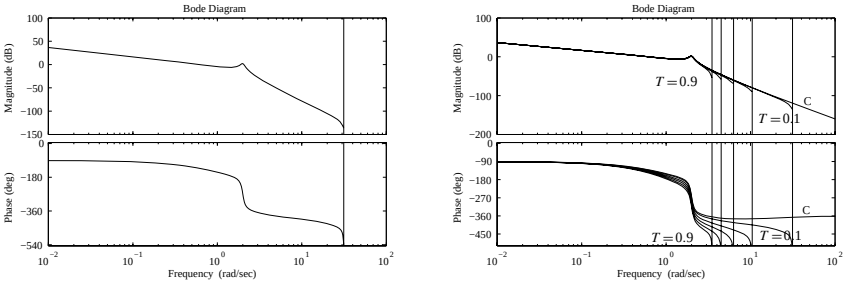


(a) Shortcut menu (b) Bode diagram with key points

Figure 3.23. *Frequency domain analysis results for Example* 3.30.

select the Stability (All Crossings) item from it. The Bode diagram with all the stability key points is shown in Figure 3.23(b). The Characteristics shortcut menus are supported for all the frequency domain curves.

Example 3.31. Consider the continuous system studied in the previous example. One may select the sampling interval of $T = 0.1$ second to find the discrete-time model. The Bode diagram of the discrete-time model can be obtained as shown in Figure 3.24(a).



(a) Bode diagram of discrete-time model (b) Bode diagrams for different T's

Figure 3.24. *Bode diagrams of the discrete-time models in Example* 3.31.

```
>> s=tf('s'); G=(s+8)/(s*(s^2+0.2*s+4)*(s+1)*(s+3));
   G1=c2d(G,0.1); bode(G1)
```

Selecting different sampling intervals, we see that the Bode diagrams are obtained as shown in Figure 3.24(b), together with that of the original continuous model. It can be seen that, if the sampling interval is large, the high-frequency part may not be satisfactory.

```
>> bode(G), hold on; for T=[0.1:0.2:1], bode(c2d(G,T)); end
```

3.5.2 Stability Analysis Using Frequency Domain Methods

The Nyquist plot, often drawn for the open-loop models, can be used to infer the closed-loop behavior in terms of stability and even the time domain response. The stability can be concluded using the well-known Nyquist Theorem.

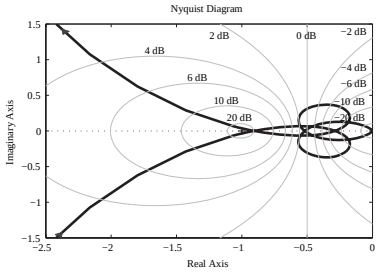
Theorem 3.6. For the closed-loop system with an open-loop transfer function $G(s)$ to be stable, the Nyquist plot of $G(s)$ must encircle in a counterclockwise direction the point $(-1, \text{j}0)$ as many times as the number of poles of $G(s)$ located in the right half of the s-plane.

The Nyquist Theorem can further be interpreted for the following two cases:

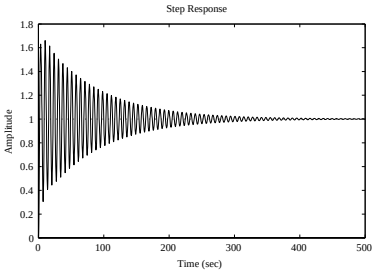
1. For a stable open-loop model $G(s)$, the closed-loop system (with unity negative feedback) is stable if and only if the Nyquist plot of $G(s)$ does not encircle the point $(-1, \text{j}0)$. If the full Nyquist plot encircles the point $(-1, \text{j}0)$ p times in a clockwise direction, then there will be p unstable closed-loop poles.
2. For an unstable open-loop model of $G(s)$ with p unstable modes, the closed-loop system is stable if and only if the Nyquist plot of $G(s)$ encircles the point $(-1, \text{j}0)$ p times in a counterclockwise direction. If there are q counterclockwise encirclements of the point $(-1, \text{j}0)$, then there will be $q - p$ unstable closed-loop poles.

Example 3.32. Consider the open-loop model

$$G(s) = \frac{2.7778(s^2+0.192s+1.92)}{s(s+1)^2(s^2+0.384s+2.56)}.$$



(a) Nyquist plot



(b) step response of closed-loop system

Figure 3.25. *Analysis of the system in Example* 3.32.

The model can be easily entered into MATLAB with the following statements, and the Nyquist plot of the system can be drawn as shown in Figure 3.25(a):

```
>> s=tf('s');
   G=2.7778*(s^2+0.192*s+1.92)/(s*(s+1)^2*(s^2+0.384*s+2.56));
   nyquist(G); axis([-2.5,0,-1.5,1.5]); grid
   figure; step(feedback(G,1))
```

It can be seen that although the trajectory is rather complicated, there is no encirclement around the $(-1, \text{j}0)$ point. Since there are no unstable poles in the open-loop model, it can be shown by the use of the Nyquist Theorem that the closed-loop system under unity negative feedback is stable. The step response of the closed-loop system is obtained as shown in Figure 3.25(b).

Although the closed-loop system is stable, the step response has a strong oscillation, so the system performance will probably not be satisfactory. In this case, a controller will need to be designed to improve its performance.

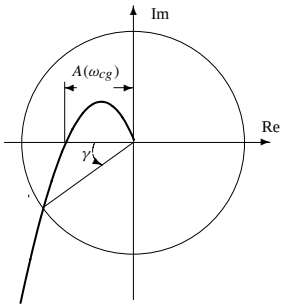
3.5.3 Gain and Phase Margins of a System

It can be seen from the previous examples that, although the stability of the systems is extremely important, it is not the only important factor in describing the behavior of the systems. Frequency response margins are effective indicators in addressing relative stability and performance problems.

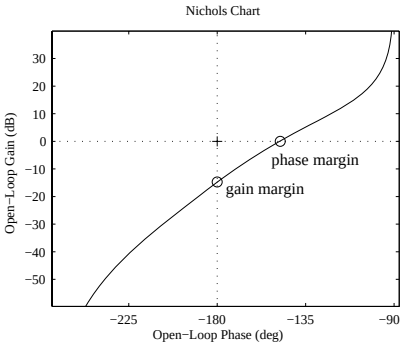
In Figures 3.26(a) and (b), the sketches of gain and phase margins are illustrated, respectively, on the Nyquist plot and Nichols chart. The gain and phase margins can also be illustrated on the Bode diagram.

If for a system with a stable open-loop transfer function the Nyquist plot intersects the negative real axis at frequency ω_{cg}, the gain margin is defined as the reciprocal of the gain, i.e., $G_m = 1/A(\omega_{cg})$, and is often expressed in dBs. If the Nyquist plot intersects the unit circle at frequency ω_{cp}, the phase margin is defined as $\gamma = \phi(\omega_{cp}) - 180°$.

It can be seen that, normally, the larger the value of gain margin G_m, the faster the closed-loop step response. If $G_m < 1$, the closed-loop system is unstable. Similarly, if



(a) Nyquist curve



(b) Nichols chart

Figure 3.26. *Graphical representations of gain and phase margins.*

the open-loop frequency response is relatively smooth in the gain/phase margin region, the larger the phase margin, the less the overshoot in the closed-loop step response. However, if $\gamma < 0$, the closed-loop system is unstable. The following special cases should also be considered.

1. If there is no intersection between the Nyquist plot and the negative real axis, the gain margin is infinite.
2. If the Nyquist plot intersects many times the negative real axis between $(-1, \text{j}0)$ and $(0, \text{j}0)$, the one nearest to the point $(-1, \text{j}0)$ can be regarded as the gain margin point.
3. If there is no intersection between the Nyquist plot with the unit circle, the phase margin is infinite.
4. If the Nyquist plot intersects many times the unit circle in the third quadrant, the one which is nearest to the negative real axis can be regarded as the phase margin point.

A function `margin()` is provided in the Control Systems Toolbox to evaluate the gain and phase margins. The syntax of the function is

```
[Gm,γ,ωcg,ωcp]=margin(G);
```

As a result, if a margin is infinite, the returned value will be `Inf`, while the corresponding frequency will be `NaN`.

Example 3.33. Consider again the open-loop model in Example 3.32. The following statements can be used to analyze the gain and phase margins of the system:

```
>> s=tf('s');
   G=2.7778*(s^2+0.192*s+1.92)/(s*(s+1)^2*(s^2+0.384*s+2.56));
   [gm,pm,wg,wp]=margin(G)
```

The gain margin is 1.105 at frequency 0.9621 rad/sec, and the phase margin is 2.0985°, at frequency 0.9261 rad/sec. Since they are both very small, the closed-loop system will exhibit very strong oscillation, although it is stable.

3.5.4 Variations of Conventional Nyquist Plots

It can be seen from the examples in the previous section that if the frequency range is not selected properly, or if the gain of the system is too high, one cannot obtain any adequate information about the actual and detailed behavior around the point $(-1, \mathrm{j}0)$, which is crucial in assessing the stability of the closed-loop system. In this case, two nonlinear transformation techniques can be applied to provide more information from the Nyquist plot of the system, namely the arc tangent (atan) Nyquist plot and the logarithmic Nyquist plot.

Arc tangent transformations

From the ordinary Nyquist plots, if the phase angles are kept unchanged, one can perform an arc tangent transformation on the magnitude such that

$$|G_{\text{new}}(\mathrm{j}\omega)| = \frac{2}{\pi}\text{atan}|G(\mathrm{j}\omega)|. \tag{3.42}$$

Clearly, the nonlinearly transformed Nyquist plot is kept within a unit circle for all ω. In this case, the readability of the new Nyquist plot improves significantly. With the above nonlinear transformation, the critical point in the Nyquist stability criterion is changed from $(-1, \mathrm{j}0)$ to $(-1/2, \mathrm{j}0)$, and it is also very easy to check.

A MATLAB function `atannyq()` is prepared in the following to draw the transformed Nyquist plot of the system:

```
function atannyq(G,w)
if nargin==1, [x,y,w]=nyquist(G);
elseif nargin==2, [x,y]=nyquist(G,w); end
pp=atan2(y,x); H=2/pi*atan(x.^2+y.^2).*exp(sqrt(-1)*pp);
nyquist(frd(H,w))
```

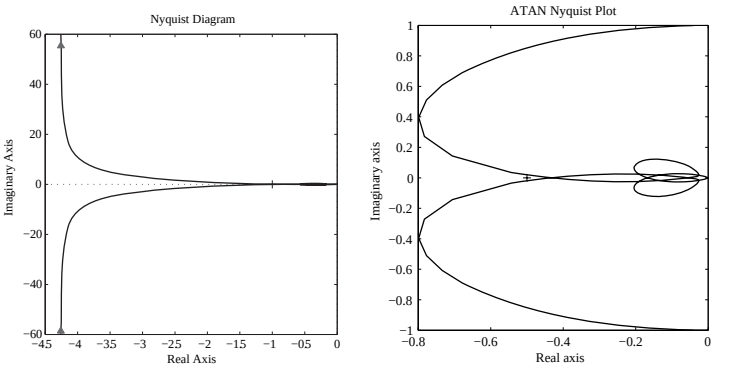
The syntax of the function is `atannyq(G,w)`, where G is the open-loop model and $\boldsymbol{w}$ is the specified frequency vector, which is optional.

Example 3.34. Consider again the system in Examples 3.32. The ordinary Nyquist plot and the atan Nyquist plot are obtained as shown, respectively, in Figures 3.27(a) and (b). It is obvious that information around the $(-1, \mathrm{j}0)$ point in the ordinary Nyquist plot is not well provided. However, the information around the $(-0.5, \mathrm{j}0)$ point is clearly given.

```
>> s=tf('s');
   G=2.7778*(s^2+0.192*s+1.92)/(s*(s+1)^2*(s^2+0.384*s+2.56));
   nyquist(G), figure, atannyq(G)
```

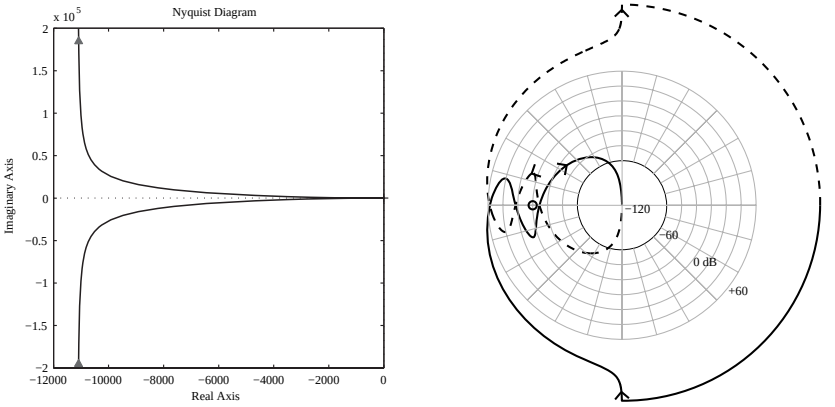
Logarithmic Nyquist plots

If the ordinary Nyquist plot can be regarded as the polar plot, where the complex gain $G(\mathrm{j}\omega)$ can be represented as $A(\omega)\mathrm{e}^{\mathrm{j}\phi(\omega)}$, the logarithmic Nyquist plot can be drawn as the new representation $20\lg[A(\omega)]\mathrm{e}^{\mathrm{j}\phi(\omega)}$. The improved Nyquist plot is referred to as the logarithmic-amplitude polar diagram [34]. The function `nyqlog()` can be freely downloaded from `http://www.mathworks.com/matlabcentral/files/7444/nyqlog.zip`.



(a) Ordinary Nyquist plot (b) Atan Nyquist plot

Figure 3.27. *Transformed Nyquist plots (arc tangent).*



(a) Ordinary Nyquist plot (b) Logarithmic Nyquist plot

Figure 3.28. *Logarithmic transformed Nyquist plots.*

Example 3.35. Consider a more challenging open-loop model

$$G(s) = \frac{200(1+3s)(1+2s)}{s(1+50s)(1+10s)(1+0.5s)(1+0.1s)},$$

where, with the following statements, the ordinary Nyquist plot and the logarithmic Nyquist plot are both obtained, as shown in Figures 3.28(a) and (b), respectively.

```
>> s=tf('s');
   G=200*(1+3*s)*(1+2*s)/(s*(1+50*s)*(1+10*s)*(1+0.5*s)*(1+0.1*s));
   nyquist(G), figure, nyqlog(G)
```

It can be seen that the ordinary Nyquist plot does not provide any information around the critical point. Even if the original Nyquist plot is magnified, it may still lead to misleading conclusions [34]. In this case, the new transformed Nyquist plot is more informative around the critical point.

3.6 Introduction to Model Reduction Techniques

A model reduction technique was first introduced by Davison in 1966 [35]. The method introduced was to reduce the dimension of the coefficient matrix of the system while preserving some of the dominant eigenvalues or more influential states of the original system. Transfer function model reduction techniques are sometimes referred to as "model simplification" [36]. Here, the term "model reduction" will be used throughout the book since this terminology appears more often in the literature; see, e.g., [37, 38]. In this section, model reduction techniques for both the transfer function models and the state space models will be introduced.

In what follows, the reduced-order model is denoted by

$$G_{r/k}(s) = \frac{\beta_1 s^r + \beta_2 s^{r-1} + \cdots + \beta_{r+1}}{\alpha_1 s^k + \alpha_2 s^{k-1} + \cdots + \alpha_k s + \alpha_{k+1}}, \tag{3.43}$$

where $k < n$ with n the order of the original system. Again, for simplicity, it is assumed that $\alpha_{k+1} = 1$.

3.6.1 Padé Approximations and Routh Approximations

Suppose that the Taylor series of the original model $G(s)$ can be written as

$$G(s) = c_0 + c_1 s + c_2 s^2 + \cdots, \tag{3.44}$$

where the time moments c_i can be computed from (3.20). A low-order transfer function can be constructed to approximate the original model. If one wants to retain the first $r+m+1$ time moments c_i ($i = 0, \ldots, r+k$) of the original model, the following formulae can be established:

$$\begin{cases} \beta_{r+1} = c_0, \\ \beta_r = c_1 + \alpha_k c_0, \\ \qquad \vdots \\ \beta_1 = c_r + \alpha_k c_{r-1} + \cdots + \alpha_{k-r+1} c_0, \\ 0 = c_{r+1} + \alpha_k c_r + \cdots + \alpha_{k-r} c_0, \\ 0 = c_{r+2} + \alpha_k c_{r+1} + \cdots + \alpha_{k-r-1} c_0, \\ \qquad \vdots \\ 0 = c_{k+r} + \alpha_k c_{k+r-1} + \cdots + \alpha_2 c_{r+1} + \alpha_1 c_r. \end{cases} \tag{3.45}$$

From the last m formulae in (3.45), the following algebraic equations can be obtained:

$$\begin{bmatrix} c_r & c_{r-1} & \cdots & . \\ c_{r+1} & c_r & \cdots & . \\ \vdots & \vdots & \ddots & \vdots \\ c_{k+r-1} & c_{k+r-2} & \cdots & c_r \end{bmatrix} \begin{bmatrix} \alpha_k \\ \alpha_{k-1} \\ \vdots \\ \alpha_1 \end{bmatrix} = - \begin{bmatrix} c_{r+1} \\ c_{r+2} \\ \vdots \\ c_{k+r} \end{bmatrix} \tag{3.46}$$

from which the coefficients α_i of the denominator can be evaluated easily by solving the linear algebraic equations. From the first $r+1$ formulae of (3.45), the coefficients β_i of the numerator can be evaluated. The matrix form is given by

$$\begin{bmatrix} \beta_{r+1} \\ \beta_r \\ \vdots \\ \beta_1 \end{bmatrix} = \begin{bmatrix} c_0 & 0 & \cdots & 0 \\ c_1 & c_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ c_r & c_{r-1} & \cdots & c_0 \end{bmatrix} \begin{bmatrix} 1 \\ \alpha_k \\ \vdots \\ \alpha_{k-r+1} \end{bmatrix}. \tag{3.47}$$

The above algorithm is referred to as the Padé approximation technique. A MATLAB function `pademod()` is written based on this model reduction technique:

```
function G_red=pademod(G_Sys,r,k)
c=timmomt(G_Sys,r+k+1); G_red=pade_app(c,r,k);
```

The function also calls a low-level function `pade_app()`, where

```
function Gr=pade_app(c,r,k)
w=-c(r+2:r+k+1)'; vv=[c(r+1:-1:1)'; zeros(k-1-r,1)];
W=rot90(hankel(c(r+k:-1:r+1),vv)); V=rot90(hankel(c(r:-1:1)));
x=[1 (W\w)']; dred=x(k+1:-1:1)/x(k+1);
y=[c(1) x(2:r+1)*V'+c(2:r+1)]; nred=y(r+1:-1:1)/x(k+1);
Gr=tf(nred,dred);
```

The syntax of the `pademod()` function is `Gr=pademod(G,r,k)`, where G is the transfer function object of the original plant model, and r and k are the desired orders of the numerator and denominator, respectively. The returned variables G_r is the reduced-order model.

Example 3.36. Consider a fourth-order transfer function given by

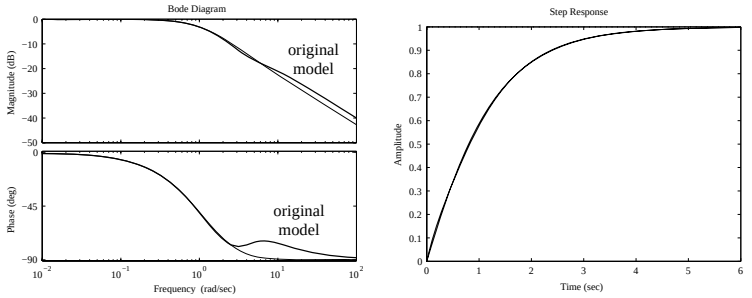
$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24}.$$

The time moments c_i and the second-order reduced model can be obtained using the following MATLAB commands:

```
>> G=tf([1 7 24 24],[1 10 35 50 24]); Gr=pademod(G,1,2)
   bode(G,Gr), figure, step(G,Gr)
```

and the second-order approximate model is

$$G_r(s) = \frac{0.8544s + 0.6957}{s^2 + 1.091s + 4.174}.$$



(a) Bode diagrams (b) Step responses

Figure 3.29. *Bode diagram and step response comparisons.*

The Bode diagram and step response comparisons are obtained as shown in Figures 3.29(a) and (b), respectively. It can be seen that the reduced model may satisfactorily approximate the original fourth-order system.

Example 3.37. Assume that the original model is given by

$$G(s) = \frac{0.067s^5 + 0.6s^4 + 1.5s^3 + 2.016s^2 + 1.55s + 0.6}{0.067s^6 + 0.7s^5 + 3s^4 + 6.67s^3 + 7.93s^2 + 4.63s + 1}.$$

The poles of $G(s)$ are found as follows:

```
>> num=[0.067,0.6,1.5,2.016,1.66,0.6];
   den=[0.067 0.7 3 6.67 7.93 4.63 1]; G=tf(num,den); zpk(G)
```

The zero-pole-gain format of the original model is

$$G(s) = \frac{(s+5.92)(s+1.221)(s+0.897)(s^2+0.9171s+1.381)}{(s+2.805)(s+1.856)(s+1.025)(s+0.501)(s^2+4.261s+5.582)}.$$

It can be seen that the original model is stable. The third-order Padé approximation of the original model can be obtained by

```
>> Gr=zpk(pademod(G,1,3))
```

The reduced model obtained is

$$G_r(s) = \frac{-0.6328(s+0.7695)}{(s-2.598)(s^2+1.108s+0.3123)},$$

and it is obvious that the reduced-order model is unstable. This means that the Padé approximation method may not preserve the stability of the original system.

Since the Padé approach may fail to preserve the stability of the original system, Routh approximation techniques are sometimes used. The Routh approximation method proposed by Hutton [39] was to find a stable reduced-order model for the original asymptotically stable model.

A MATLAB function `routhmod()` is written according to the Routh approximation algorithm [39]:

```
function Gr=routhmod(G,nr)
num=G.num{1}; den=G.den{1}; n0=length(den); n1=length(num);
a1=den(end:-1:1); b1=[num(end:-1:1) zeros(1,n0-n1-1)];
for k=1:n0-1,
   k1=k+2; alpha(k)=a1(k)/a1(k+1); beta(k)=b1(k)/a1(k+1);
   for i=k1:2:n0-1,
      a1(i)=a1(i)-alpha(k)*a1(i+1); b1(i)=b1(i)-beta(k)*a1(i+1);
end, end
nn=[]; dd=[1]; nn1=beta(1); dd1=[alpha(1),1]; nred=nn1; dred=dd1;
for i=2:nr,
   nred=[alpha(i)*nn1, beta(i)]; dred=[alpha(i)*dd1, 0];
   n0=length(dd); n1=length(dred); nred=nred+[zeros(1,n1-n0),nn];
   dred=dred+[zeros(1,n1-n0),dd]; nn=nn1; dd=dd1; nn1=nred; dd1=dred;
end
Gr=tf(nred(nr:-1:1),dred(end:-1:1));
```

and it can be used to find the Routh approximant of a given plant model. The syntax of this function is `Gr=routhmod(G,k)`, where G and G_r are the original and reduced models, respectively, and k is the expected order of the Routh approximation. Note that in the reduced-order model, the order of the numerator is one less than that of the denominator.

Example 3.38. Consider the model in Example 3.37. The third-order stable Routh approximation can be obtained using the following MATLAB statements:

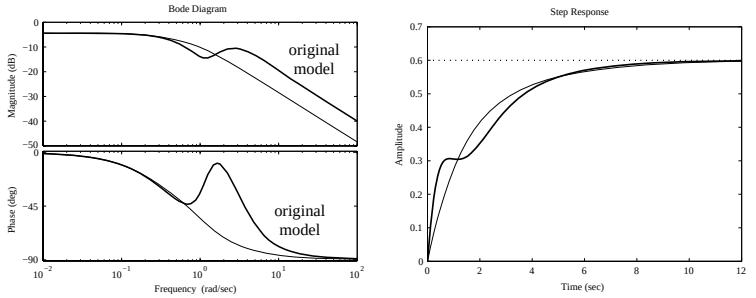
```
>> num=[0.067,0.6,1.5,2.016,1.66,0.6];
   den=[0.067 0.7 3 6.67 7.93 4.63 1]; G=tf(num,den);
   Gr=routhmod(G,3); zpk(Gr)
   bode(G,Gr), figure, step(G,Gr)
```

where the reduced-order model is

$$G_r(s) = \frac{0.37792(s^2+0.9472s+0.3423)}{(s+0.4658)(s^2+1.15s+0.463)},$$

and it can be seen that the model is stable. The Bode diagrams and step responses are compared in Figure 3.30. Although the stability in the reduced-order system is retained, the fitting results are not satisfactory.

However, it is usually claimed that although the Routh approximation method may preserve the stability of the original system, the dynamic fitting of time domain and frequency domain responses to those of the original systems may not be satisfactory. Thus, other approaches, such as the dominant mode methods [40], impulse energy approximation method and its variations [41], the stability equation–based methods [42], the multifrequency fitting Padé approximation method [43, 44], and the optimal model reduction methods [45], can be used to get a better fitting while preserving the stability of the original system. A relatively complete summary on frequency domain model reduction techniques can be found in [46].



(a) Bode diagrams (b) Step responses

Figure 3.30. *Bode diagram and step response comparisons.*

3.6.2 Padé Approximations to Delay Terms

Similar to the case of Padé approximation to transfer function models, the Padé technique can also be used in the approximation of pure delay terms. An nth-order Padé approximation to the delay term $e^{-\tau s}$ can be written as

$$P_{n,\tau}(s) = \frac{1 - \tau s/2 + p_2(\tau s)^2 - p_3(\tau s)^3 + \cdots + (-1)^{n+1} p_n(\tau s)^n}{1 + \tau s/2 + p_2(\tau s)^2 + p_3(\tau s)^3 + \cdots + p_n(\tau s)^n}. \tag{3.48}$$

A MATLAB function `pade()` is provided in the Control Systems Toolbox with the syntax `[`$\boldsymbol{n}_P$`,`$\boldsymbol{d}_P$`]= pade(`τ`,`n`)`, where n is the order of Padé approximation. The numerator and denominator of the rational Padé approximation are returned in $(\boldsymbol{n}_P, \boldsymbol{d}_P)$ for $P_{n,\tau}(s)$.

Now assume that the order of the numerator can be chosen arbitrarily. The Maclaurin series expansion for the pure delay term can be written as

$$e^{-\tau s} = 1 - \frac{1}{1!}\tau s + \frac{1}{2!}\tau^2 s^2 - \frac{1}{3!}\tau^3 s^3 + \cdots \tag{3.49}$$

which is similar to the time moment expansion of (3.44). Therefore, the same algorithm can be used to find a Padé approximation of the delay term. A MATLAB function `paderm()` is written

```
function [n,d]=paderm(tau,r,k)
c(1)=1; for i=2:r+k+1, c(i)=-c(i-1)*tau/(i-1); end
Gr=pade_app(c,r,k); n=Gr.num{1}(k-r+1:end); d=Gr.den{1};
```

which can be used to find a Padé approximation of a delay term. The syntax of this function is `[`$\boldsymbol{n}$`,`$\boldsymbol{d}$`]= paderm(`τ`,`r`,`k`)`, where r and k are the orders of the numerator and denominator, respectively. The numerator and denominator coefficients are returned in the vectors $(\boldsymbol{n},\boldsymbol{d})$.

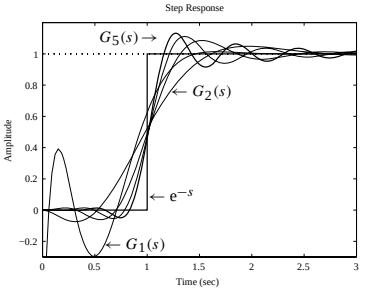


Figure 3.31. *Example* 3.39.

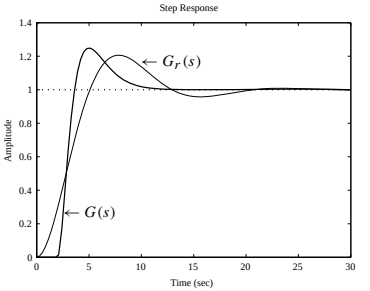


Figure 3.32. *Example* 3.40.

Example 3.39. Consider a pure delay term $G(s) = e^{-s}$. The following MATLAB statements can be used to find its Padé approximation:

```
>> tau=1; [n1,d1]=pade(tau,3); G1=tf(n1,d1)
   [n2,d2]=paderm(tau,1,3); G2=tf(n2,d2)
   [n3,d3]=paderm(tau,2,5); G3=tf(n3,d3);
   [n4,d4]=paderm(tau,3,7); G4=tf(n4,d4);
   [n5,d5]=paderm(tau,4,9); G5=tf(n5,d5);
   step(G1,G2,G3,G4,G5), line([0 1 1+eps 3],[0,0,1 1])
```

The approximate models using the two functions are, respectively,

$$G_1(s) = \frac{-s^3 + 12s^2 - 60s + 120}{s^3 + 12s^2 + 60s + 120}, \quad G_2(s) = \frac{-6s + 24}{s^3 + 6s^2 + 18s + 24}$$

and the step response comparisons are shown in Figure 3.31. It can be seen that the fitting to the pure delay term can be well approximated by rational terms if the order selected is suitable.

Example 3.40. Consider a transfer function with a delay

$$G(s) = \frac{3s + 1}{(s + 1)^3} e^{-2s}.$$

The Maclaurin series expansion of the pure delay term can be evaluated as

```
>> cd=[1]; tau=2; for i=1:5, cd(i+1)=-tau*cd(i)/i; end
```

One can then obtain the time moments of the whole system by multiplying the two series to find a Padé approximation model of the original system, which is illustrated in the following:

```
>> G=tf([3,1],[1,3,3,1],'ioDelay',2); c=timmomt(G,5);
   c_hat=conv(c,cd); Gr=pade_app(c_hat,1,3), step(G,Gr)
```

and it can be found that

$$G_r(s) = \frac{0.2012s + 0.009146}{s^3 + 0.4482s^2 + 0.2195s + 0.009146}.$$

The step response fitting is shown in Figure 3.32, from which it can be seen that the approximation is not satisfactory.

3.6.3 Suboptimal Reduction Techniques for Systems with Delays

Fitness measures on reduced-order models

There could be many measures on the quality of reduced-order models. Here, a simple and commonly used measure is introduced. Consider the block diagram shown in Figure 3.33, where the two blocks, namely the original model and the reduced model, are subject to the same input signal $r(t)$, and the error between the output of the original model and that of the reduced-order model is denoted by $e(t)$.

Based on the error signal, many measures can be used, such as

$$I_{\text{ISE}} = \int_0^\infty e^2(t)\mathrm{d}t, \quad I_{\text{ITAE}} = \int_0^\infty t|e(t)|\mathrm{d}t, \quad I_{\text{ISTE}} = \int_0^\infty t^2 e^2(t)\mathrm{d}t. \tag{3.50}$$

where ISE stands for integral of squared error, ITAE for integral of time-multiplied absolute value of error, and ISTE for integral of time-multiplied squared error. The commonly used ISE criterion will be used in the following discussions.

Suppose the original model is given by

$$G(s)\mathrm{e}^{-Ts} = \frac{b_1 s^{n-1} + \cdots + b_{n-1}s + b_n}{s^n + a_1 s^{n-1} + \cdots + a_{n-1}s + a_n}\mathrm{e}^{-Ts} \tag{3.51}$$

and the reduced-order model is given by

$$G_{r/k}(s)\mathrm{e}^{-\tau s} = \frac{\beta_1 s^r + \cdots + \beta_r s + \beta_{r+1}}{s^k + \alpha_1 s^{k-1} + \cdots + \alpha_{k-1}s + \alpha_k}\mathrm{e}^{-\tau s}. \tag{3.52}$$

The Laplace transform of the error signal can be written as

$$E(s) = \left[G(s)\mathrm{e}^{-Ts} - G_{r/k}(s)\mathrm{e}^{-\tau s}\right]R(s), \tag{3.53}$$

where $R(s)$ is the Laplace transform of the input signal $r(t)$. Therefore, if one considers the input signal as an impulse function, I_{ISE} equals the squared $\mathcal{H}_2$-norm of the difference transfer function in the above equation. Some MATLAB functions from Sec. 3.1.5 can be used to compute the $\mathcal{H}_2$-norm.

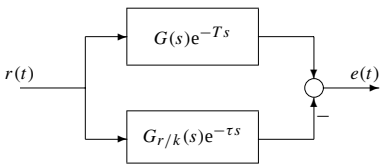


Figure 3.33. *Error for model reduction.*

Introduction to suboptimal order reduction

The idea of optimization-based model reduction algorithms is straightforward. It is simply to transform the model reduction problem into a parameter optimization problem by minimizing $e(t)$ via the ISE criterion.

It has been shown that the integral squared value of the signal $h(t) = w(t)e(t)$ ($w(t)$ is the weighting function) is in fact the square of the $\mathcal{H}_2$-norm of $H(s)$, the Laplace transform of signal $h(t)$, i.e.,

$$\sigma_h^2 = \int_0^\infty h^2(t)\mathrm{d}t = \int_0^\infty w^2(t)e^2(t)\mathrm{d}t = \|H(s)\|_2^2. \tag{3.54}$$

If $H(s)$ is a rational function of s with no pole having any nonnegative real part, the integral squared value σ_h^2 can also be obtained from `norm(H)`. However, in our case, due to the delay term, the above method cannot be used directly. Padé approximation for the time delay can be used to make $H(s)$ rational. In this sense, the optimization-based method should be referred to as the suboptimal model reduction technique since the time delay term is replaced by its Padé approximation.

Define a parameter vector $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta} = [\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_{r+1}, \tau]^{\mathbf{T}}. \tag{3.55}$$

The model approximate error can be written explicitly as $\widehat{e}(t, \boldsymbol{\theta})$ for a given original model and the input signal. An objective function for suboptimal model reduction can be defined as

$$J = \min_{\boldsymbol{\theta}} \left[\int_0^\infty w^2(t)\widehat{e}^2(t, \boldsymbol{\theta})\mathrm{d}t\right]. \tag{3.56}$$

A MATLAB function `opt_app()` is written based on the numerical optimization algorithms for the suboptimal model reduction:

```
function G_r=opt_app(G_Sys,r,k,key,G0)
GS=tf(G_Sys); num=GS.num{1}; den=GS.den{1};
Td=totaldelay(GS); GS.ioDelay=0; GS.InputDelay=0;GS.OutputDelay=0;
if nargin<5,
   n0=[1,1]; for i=1:k-2, n0=conv(n0,[1,1]); end
   G0=tf(n0,conv([1,1],n0));
end
beta=G0.num{1}(k+1-r:k+1); alph=G0.den{1}; Tau=1.5*Td;
x=[beta(1:r),alph(2:k+1)]; if abs(Tau)<1e-5, Tau=0.5; end
dc=dcgain(GS); if key==1, x=[x,Tau]; end
y=opt_fun(x,GS,key,r,k,dc); x=fminsearch('opt_fun',x,[],GS,key,r,k,dc);
alph=[1,x(r+1:r+k)]; beta=x(1:r+1); if key==0, Td=0; end
beta(r+1)=alph(end)*dc; if key==1, Tau=x(end)+Td; else, Tau=0; end
G_r=tf(beta,alph,'ioDelay',Tau);
```

A subfunction `opt_fun()`, which is used to evaluate the objective function in the model reduction process, is also written as

```
function y=opt_fun(x,G,key,r,k,dc)
ff0=1e10; a=[1,x(r+1:r+k)]; b=x(1:r+1); b(end)=a(end)*dc; g=tf(b,a);
if key==1,
   tau=x(end); if tau<=0, tau=eps; end, [n,d]=pade(tau,3); gP=tf(n,d);
```

```
else, gP=1; end
G_e=G-g*gP; G_e.num{1}=[0,G_e.num{1}(1:end-1)];
[y,ierr]=geth2(G_e); if ierr==1, y=10*ff0; else, ff0=y; end
function [v,ierr]=geth2(G)
G=tf(G); num=G.num{1}; den=G.den{1}; ierr=0; v=0; n=length(den);
if abs(num(1))>eps
   disp('System not strictly proper');
   ierr=1; return
else, a1=den; b1=num(2:length(num)); end
for k=1:n-1
  if (a1(k+1)<=eps), ierr=1; return
  else,
     aa=a1(k)/a1(k+1); bb=b1(k)/a1(k+1); v=v+bb*bb/aa; k1=k+2;
     for i=k1:2:n-1
        a1(i)=a1(i)-aa*a1(i+1); b1(i)=b1(i)-bb*a1(i+1);
end, end, end
v=sqrt(0.5*v);
```

The syntax of the function is `Gr=opt_app(G,nr,nd,key,G0)`, where G and G_r are the original and the reduced-order model objects; n_r, n_d are the expected orders of the numerator and denominator, respectively. The argument `key` indicates whether a delay term is expected in the reduced model. G_0 is the optional initial reduced-order model. This function considers only the case of the ISE criterion with a step input, which is a simplified version of [47].

Example 3.41. Consider the following transfer function studied in [43]:

$$G(s) = \frac{1 + 8.8818s + 29.9339s^2 + 67.087s^3 + 80.3787s^4 + 68.6131s^5}{1 + 7.6194s + 21.7611s^2 + 28.4472s^3 + 16.5609s^4 + 3.5338s^5 + 0.0462s^6}.$$

Using the following MATLAB statements

```
>> num=[68.6131,80.3787,67.087,29.9339,8.8818,1];
   den=[0.0462,3.5338,16.5609,28.4472,21.7611,7.6194,1];
   G=tf(num,den); Gr=zpk(opt_app(G,2,3,0)), step(G,Gr,8)
```

the optimal reduced model is obtained as

$$G_r(s) = \frac{1523.6536(s^2 + 0.3492s + 0.2482)}{(s + 74.85)(s^2 + 3.871s + 5.052)}.$$

The step response comparisons are shown in Figure 3.34, and it can be seen that the fitting results are satisfactory.

Example 3.42. Consider the plant model given by [48],

$$G(s) = \frac{432}{(5s + 1)(2s + 1)(0.7s + 1)(s + 1)(0.4s + 1)}.$$

A suboptimal reduced model with a delay can be obtained using the following MATLAB statements:

```
>> s=tf('s'); G=432/((5*s+1)*(2*s+1)*(0.7*s+1)*(s+1)*(0.4*s+1));
   Gr=zpk(opt_app(G,0,2,1)), step(G,Gr)
```

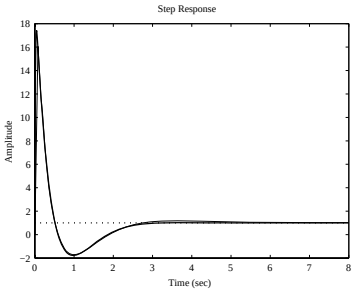



Figure 3.34. *Example* 3.41.

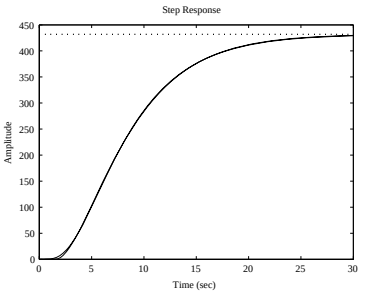


Figure 3.35. *Example* 3.42.

and the reduced model is

$$G_r(s) = \frac{31.4907}{(s + 0.3283)(s + 0.222)} e^{-1.5s}.$$

The step response comparisons are shown in Figure 3.35, and it can be seen that the original high-order system can satisfactorily be approximated by the low-order one with a delay.

3.6.4 State Space Model Reduction

Balanced realization method

Suppose that the balanced realization of the original model can be partitioned as

$$\begin{bmatrix} \dot{\boldsymbol{x}}_1 \\ \dot{\boldsymbol{x}}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{B}_2 \end{bmatrix} \boldsymbol{u}, \ \boldsymbol{y} = [\boldsymbol{C}_1 \ \ \boldsymbol{C}_2] \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} + \boldsymbol{D}\boldsymbol{u} \tag{3.57}$$

and assume that the states in subvector $\boldsymbol{x}_2$ are to be chopped off. Then, the reduced model is written as the following:

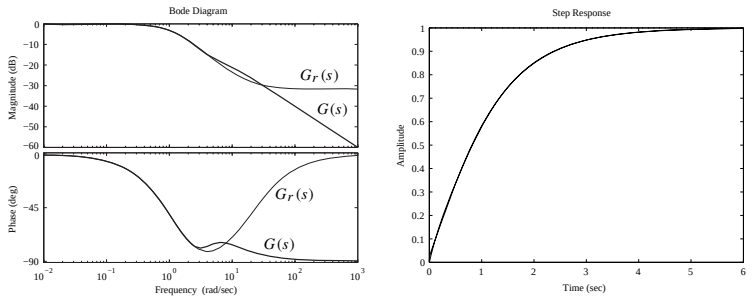
$$\begin{aligned} \dot{\boldsymbol{x}}_1 &= [\boldsymbol{A}_{11} - \boldsymbol{A}_{12}\boldsymbol{A}_{22}^{-1}\boldsymbol{A}_{21}]\boldsymbol{x}_1 + [\boldsymbol{B}_1 - \boldsymbol{A}_{12}\boldsymbol{A}_{22}^{-1}\boldsymbol{B}_2]\boldsymbol{u}, \\ \boldsymbol{y} &= [\boldsymbol{C}_1 - \boldsymbol{C}_2\boldsymbol{A}_{22}^{-1}\boldsymbol{A}_{21}]\boldsymbol{x}_1 + [\boldsymbol{D} - \boldsymbol{C}_2\boldsymbol{A}_{22}^{-1}\boldsymbol{B}_2]\boldsymbol{u}. \end{aligned} \tag{3.58}$$

A function `modred()` implementing the above algorithm is provided in the Control Systems Toolbox with syntax `Gr=modred(G,elim)`, where $\boldsymbol{G}$ is the balanced realized state space object and `elim` contains the states to be dropped off. The reduced model $\boldsymbol{G}_r$ is then returned.

Example 3.43. Consider again the system model in Example 2.22. To get a second-order reduced model, the following MATLAB statements can be used to obtain the Gramian of the balanced realized system model:

```
>> G=tf([1,7,24,24],[1,10,35,50,24]); [Gb,g]=balreal(ss(G))
```

where the Gramians $\boldsymbol{g} = [0.5179, 0.0309, 0.0124, 0.0006]^{\mathbf{T}}$. Clearly, the contribution to the input-output relationship from the third and fourth states is not very important. Thus, it is safe to eliminate them to get a second-order reduced model using the following



(a) Bode diagrams (b) Step responses

Figure 3.36. *Bode diagram and step response comparisons.*

MATLAB statements:

```
>> Gr=modred(Gb,[3,4]); zpk(Gr), bode(G,Gr), figure, step(G,Gr)
```

The reduced model is then

$$G_r(s) = \frac{0.025974(s+4.307)(s+22.36)}{(s+1.078)(s+2.319)}.$$

The Bode diagram and step response comparisons are shown in Figure 3.36 and it can be seen that the fitting is satisfactory. It should also be noted that the reduced-order system is not strictly proper, and hence we have the small initial jump in the step response.

Schur's balanced realization truncation method

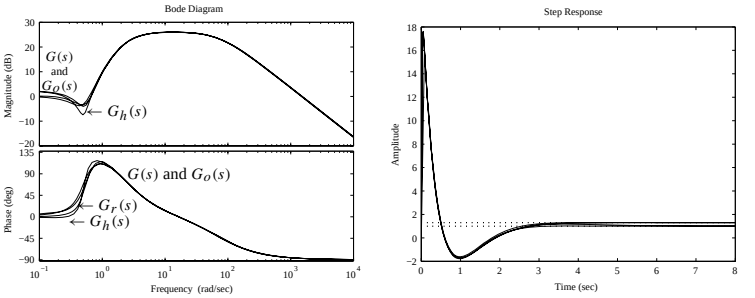
Schur's balanced realization truncation function `schmr()` provided in the Robust Control Toolbox can perform a model reduction task similar to `modred()`. The difference between the two techniques is that an unstable system can be handled in `schmr()`. The syntax of `schmr()` is `Gr=schmr(G,1,nr)`, where $\boldsymbol{G}$ is the original model object in state space format, n_r is the expected order of the reduced model, and $\boldsymbol{G}_r$ returns the reduced-order model object also in the state space form.

Example 3.44. Consider again the plant model in Example 3.41. To apply the Schur model reduction algorithm, the state space model of the system should be obtained first. This can be done using the following MATLAB statements:

```
>> num=[68.6131,80.3787,67.087,29.9339,8.8818,1];
   den=[0.0462,3.5338,16.5609,28.4472,21.7611,7.6194,1];
   G=ss(tf(num,den)); Gr=zpk(schmr(G,1,3))
```

It is indicated that three states are removed, and the third-order reduced model using Schur's method can be written as

$$G_r(s) = \frac{1485.3076(s^2+0.1789s+0.2601)}{(s+71.64)(s^2+3.881s+4.188)}.$$



(a) Bode diagrams (b) Step responses

Figure 3.37. *Bode diagram and step response comparisons.*

Optimal Hankel norm approximation

Glover presented an algorithm to find the optimal Hankel approximation to a given state space model [49]. The reduced-order model using the Hankel norm approximation algorithm can be obtained with the MATLAB function `ohklmr()` provided in the Robust Control Toolbox. The syntax of the function is `Gr=ohklmr(G,1,k)`, where $\boldsymbol{G}$ is the original model object in state space format, k is the expected order of the reduced-order model, and $\boldsymbol{G}_r$ returns the reduced-order model object in state space.

Example 3.45. Consider again the plant model in Example 3.41. The third-order reduced model using the optimal Hankel norm approximation method can be obtained as follows:

```
>> num=[68.6131,80.3787,67.087,29.9339,8.8818,1];
   den=[0.0462,3.5338,16.5609,28.4472,21.7611,7.6194,1];
   G=ss(tf(num,den)); Gh=zpk(ohklmr(G,1,3))
```

and it is indicated that three states are removed, and a reduced-order model is then returned as

$$G_h(s) = \frac{1527.8048(s^2+0.2764s+0.2892)}{(s+73.93)(s^2+3.855s+4.585)}.$$

For the same original model, the optimal approximation can also be obtained by

```
>> Go=zpk(opt_app(G,2,3,0)),
   bode(G,Go,Gr,Gh,{0.1,10000}), figure, step(G,Go,Gr,Gh,8)
```

where the reduced-order model is

$$G_o(s) = \frac{1523.6536(s^2+0.3492s+0.2482)}{(s+74.85)(s^2+3.871s+5.052)}.$$

The Bode diagram and step response comparisons are shown in Figure 3.37. It can be seen that they all fit satisfactorily into the original model. Among all three models, the optimal reduced-order model is significantly better than the other two reduced models.

Problems

1. Check the stability for the following systems:

   (a) $G(s) = \dfrac{1}{s^3 + 2s^2 + s + 2}$,

   (b) $G(s) = \dfrac{1}{6s^4 + 3s^3 + 2s^2 + s + 1}$,

   (c) $G(s) = \dfrac{1}{s^4 + s^3 - 3s^2 - s + 2}$,

   (d) $G(s) = \dfrac{3s + 1}{s^2(300s^2 + 600s + 50) + 3s + 1}$,

   (e) $G(s) = \dfrac{0.2(s + 2)}{s(s + 0.5)(s + 0.8)(s + 3) + 0.2(s + 2)}$,

   (f) $H(z) = \dfrac{-3z + 2}{z^3 - 0.2z^2 - 0.25z + 0.05}$,

   (g) $H(z) = \dfrac{3z^2 - 0.39z - 0.09}{z^4 - 1.7z^3 + 1.04z^2 + 0.268z + 0.024}$,

   (h) $H(z) = \dfrac{z^2 + 3z - 0.13}{z^5 + 1.352z^4 + 0.4481z^3 + 0.0153z^2 - 0.01109z - 0.001043}$,

   (i) $H(z^{-1}) = \dfrac{2.12z^{-2} + 11.76z^{-1} + 15.91}{z^{-5} - 7.368z^{-4} - 20.15z^{-3} + 102.4z^{-2} + 80.39z^{-1} - 340}$,

   (j)
   $$\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} -0.2 & 0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 1.6 & 0 & 0 \\ 0 & 0 & -14.3 & 85.8 & 0 \\ 0 & 0 & 0 & -33.3 & 100 \\ 0 & 0 & 0 & 0 & -10 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 30 \end{bmatrix} u(t), \\ y = [1, 0, 0, 0, 0]\boldsymbol{x}, \end{cases}$$

   (k)
   $$\begin{cases} \dot{\boldsymbol{x}} = \begin{bmatrix} 17 & 24.54 & 1 & 8 & 15 \\ 23.54 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13.75 & 20 & 22.5889 \\ 10.8689 & 1.2900 & 19.099 & 21.896 & 3 \\ 11 & 18.089799 & 25 & 2.356 & 9 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} u, \\ y = [5, 4, 3, 2, 1]\boldsymbol{x}. \end{cases}$$

2. Find the poles and zeros of the multivariable system and check the stability of the system. If unity negative feedback is assumed, check the stability of the closed-loop system
$$\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} -3 & 1 & 2 & 1 \\ 0 & -4 & -2 & -1 \\ 1 & 2 & -1 & 1 \\ -1 & -1 & 1 & -2 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 0 & 3 \\ 1 & 1 \end{bmatrix} \boldsymbol{u}(t), \\ \boldsymbol{y}(t) = \begin{bmatrix} 1 & 2 & 2 & -1 \\ 2 & 1 & -1 & 2 \end{bmatrix} \boldsymbol{x}(t). \end{cases}$$

3. Find the controllability index and observability index of the state space models in the previous problem. Obtain the controllable and observable staircase forms.

4. Find the controllable and observable decompositions of the systems given by

   (a) $\dot{\boldsymbol{x}} = \begin{bmatrix} 1 & -3 & 3 & 3 \\ -5 & -1 & -5 & 5 \\ -2 & 0 & -4 & 0 \\ -2 & 0 & -2 & 4 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ -1 \\ -1 \end{bmatrix} u, \quad y = [1, 2, 1, -2]\boldsymbol{x},$

   (b) $\dot{\boldsymbol{x}} = \begin{bmatrix} 1 & -2 & -1 \\ 1 & -2 & -2 \\ -1 & 1 & 2 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} u, \quad y = [1, -1, 0]\boldsymbol{x}.$

5. Perform the Kalman decomposition of the system model given by
$$\dot{\boldsymbol{x}} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & -3 & 0 \\ 0 & -2 & 0 & -4 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} u, \quad y = [0, 1, 1, 1]\boldsymbol{x}$$
   and write down the transformation matrix. From the Kalman decomposition of the system, obtain the minimum realization in the state space model. Give an explanation of the minimum realization from the transfer function point of view.

6. Compute the first three time moments and Markov parameters for the models given in Problems 1 and 2.

7. Determine the $\mathcal{H}_2$- and $\mathcal{H}_\infty$-norms of the following systems:

   (a) $G_1(s) = \dfrac{3s + 5}{(s + 1)(s + 2)(s + 3)(s + 4)}$,

   (b) $G_2(s) = \dfrac{3s^3 + 4s^2 - 3s + 5}{(s + 1)(s^2 + 3s + 8)(s + 3)^2(s + 4)}$,

   (c) $\dot{\boldsymbol{x}} = \begin{bmatrix} -3 & -4 & -2/3 & -1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & -4/3 & -2/3 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} u, \quad y = [0, 0, 2/3, 1]\boldsymbol{x}.$

8. Find the analytical solution to the autonomous system

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} -5 & 2 & 0 & 0 \\ 0 & -4 & 0 & 0 \\ -3 & 2 & -4 & -1 \\ -3 & 2 & 0 & -4 \end{bmatrix} \boldsymbol{x}(t), \quad \boldsymbol{x}(0) = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 1 \end{bmatrix}.$$

Compare the results with numerical results.

9. An eighth-order model $G(s)$ is given by

$$\frac{18s^7+514s^6+5982s^5+36380s^4+122664s^3+222088s^2+185760s+40320}{s^8+36s^7+546s^6+4536s^5+22449s^4+67284s^3+118124s^2+109584s+40320}.$$

Assume that the system has zero initial conditions. Find the analytical and numerical solutions of the system under step and impulse inputs. Also assume that the input signal is sinusoidal $u(t) = \sin(3t + 5)$. Assume again the system has zero initial conditions. Find the analytical solutions to the system response and verify the results by graphical comparison.

10. Draw the step response of the system

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} -0.2 & 0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 1.6 & 0 & 0 \\ 0 & 0 & -14.3 & 85.8 & 0 \\ 0 & 0 & 0 & -33.3 & 100 \\ 0 & 0 & 0 & 0 & -10 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 30 \end{bmatrix} u(t), \end{cases}$$

and $y(t) = [1, 0, 0, 0, 0]\boldsymbol{x}(t)$. Draw also the step response of all the states. For different sampling intervals of T, find the equivalent discrete-time system and compare the overshoot and settling time.

11. Draw the root locus diagrams of the following systems and determine the range of K which stabilizes the open-loop system with unity negative feedback:

(a) $G(s) = \dfrac{K(s+6)(s-6)}{s(s+3)(s+4-4\mathrm{j})(s+4-4\mathrm{j})}$,

(b) $G(s) = K\dfrac{s^2+2s+2}{s^4+s^3+14s^2+8s}$,

(c) $G(s) = \dfrac{K}{s(s^2/2600+s/26+1)}$,

(d) $G(s) = \dfrac{800K(s+1)}{s^2(s+10)(s^2+10s+50)}$,

(e) $\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} -1.5 & -13.5 & -13 & 0 \\ 10 & 0 & 0 & \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{x}(t) + K \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} u(t), \\ y(t) = [0, 0, 0, 1]\boldsymbol{x}(t). \end{cases}$

(f) $H(z) = K\dfrac{1}{(z+0.8)(z-0.8)(z-0.99)(z-0.368)}$,

(g) $\widehat{H}(z) = H(z)z^{-8}$.

12. Assume the plant model

$$G(s) = \frac{K(s-1)\mathrm{e}^{-2s}}{(s+1)^5}.$$

Find the approximate range of K which stabilizes the closed-loop system with unity negative feedback.

13. The open-loop transfer function is given by

$$G(s) = \frac{K}{(s+2)(s+4)(s^2+6s+25)}.$$

Find the range of K to make the closed-loop system with unity negative feedback stable. Also find the value of K which gives the closed-loop system a dominant damping ratio of 0.707.

14. Draw the Bode diagrams, Nyquist plots, and Nichols charts for the following systems, and check the stability of the systems under unity negative feedback control from the plots obtained. Mark the gain and phase margins on the plots obtained. Verify the results through closed-loop step responses.

(a) $G(s) = \dfrac{8(s+1)}{s^2(s+15)(s^2+6s+10)}$,

(b) $G(s) = \dfrac{4(s/3+1)}{s(0.02s+1)(0.05s+1)(0.1s+1)}$,

(c) $\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} 0 & 2 & 1 \\ -3 & -2 & 0 \\ 1 & 3 & 4 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix} u(t), \\ y(t) = [1, 2, 3]\boldsymbol{x}(t) \end{cases}$

(d) $H(z) = 0.45\dfrac{(z+1.31)(z+0.054)(z-0.957)}{z(z-1)(z-0.368)(z-0.99)}$,

(e) $G(s) = \dfrac{6(-s+4)}{s^2(0.5s+1)(0.1s+1)}$,

(f) $G(s) = \dfrac{10s^3 - 60s^2 + 110s + 60}{s^4+17s^3+82s^2+130s+100}$.

15. Draw nonlinearly transformed Nyquist plots for the systems containing integrators in the previous problems and see whether the same conclusion can be obtained.

16. Assume that a plant model is given by $G(s) = 1/s^2$, and an optimal controller can be expressed as

$$G_c(s) = \frac{5620.82s^3 + 199320.76s^2 + 76856.97s + 7253.94}{s^4 + 77.40s^3 + 2887.90s^2 + 28463.88s + 2817.59}.$$

Also assume unity negative feedback. Draw the Nyquist and Nichols plots and superimpose the M and N circles on the diagrams. Plot the closed-loop frequency response of the system and confirm that the magnitude of the peak and its corresponding phase are in agreement with the deductions from the Nyquist and Nichols plots.

17. Assume that the plant model is

$$G(s) = \frac{100(1 + s/2.5)}{s(1 + s/0.5)(1 + s/50)},$$

and a cascade controller is given by

$$G_c(s) = \frac{1000(s + 1)(s + 2.5)}{(s + 0.5)(s + 50)}.$$

Assess the closed-loop behavior of the system under unity negative feedback control. Verify the assessment by time domain analysis.

18. For the feedback system structures with

(a) $G(s) = \dfrac{3.5(s + 6)}{s(s + 1)(s + 3)(s + 8)}$, $G_c(s) = (5s + 4)/s$, $H(s) = \dfrac{0.01s + 6}{2s + 4}$,

(b) $G(s) = \dfrac{3.5(s + 6)^2}{(s + 1)(s + 3)(s + 8)(s^2 + 3s + 6)}$, $G_c(s) = \dfrac{5s + 4}{6s + 2}$, $H(s) = 1$.

By definition, the sensitivity of the feedback system can be defined as $S(s) = 1/[1 + H(s)G(s)G_c(s)]$, and the complimentary sensitivity can be defined as $T(s) = 1-S(s)$. Find the sensitivity and complementary sensitivity transfer functions.

19. Find reduced-order models for the following original models using different algorithms presented in this chapter:

(a) $G(s) = \dfrac{10 + 3s + 13s^2 + 3s^2}{1 + s + 2s^2 + 1.5s^3 + 0.5s^4}$,

(b) $G(s) = \dfrac{500 + 9984.3234s + 50664.9675s^2 + 8169.1337s^3}{500 + 10105s + 52101s^2 + 10520s^3 + 100s^4}$,

(c) $G(s) = \dfrac{1 + 0.4s}{1 + 2.283s + 1.875s^2 + 0.7803s^3 + 0.125s^4 + 0.0083s^5}$,

(d) $G(z) = \dfrac{24.1467z^3 - 67.7944z^2 + 63.4768z - 19.8209}{z^4 - 3.6193z^3 + 4.9124z^2 - 2.9633z + 0.6703}$.

20. Consider a high-order model given by

$$G(s) = \frac{(1 + 2.0587s)(1 + 2.5529s + 5.4342s^2)(1 + 3.2648s + 2.1476s^2)}{(1+3.0092s+.7970s^2)(1+6.8538s+0.6965s^2)(1+.1394s+0.6861s^2)}.$$

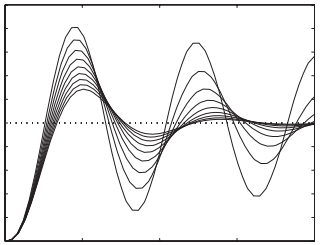
The (2/3)-order FF-Padé reduced model given in [43] is

$$G^{FF}_{2/3}(s) = \frac{1 - 1.4257s + 4.3109s^2}{1 + 0.7003s + 0.8613s^2 + 0.0837s^3}.$$

Meanwhile, a correspondence to the paper given by Stahl and Hippe [50] suggested a (3/4)-order model

$$G_{3/4}(s) = \frac{62.85(s + 2.64)(s + 0.192 + 0.608j)(s + 0.192 - 0.608j)}{(s + 8.09)(s + 5.75)(s + 0.1 + \mathrm{j}1.2)(s + 0.1 - \mathrm{j}1.2)}.$$

Using the optimum reduced-order model given in the chapter, compare the reduced-order model with the above existing reduced-order models.



Chapter 4

Simulation Analysis of Nonlinear Systems

In the previous chapters we have addressed modeling and analysis methods for linear systems. In the real world, however, control systems always contain nonlinear effects, which may be inherent and unavoidable such as friction, or may be introduced intentionally to provide better performance either technically or economically. A good example is the use of a relay for on-off temperature control. Indeed one could argue that a control system which does not operate under actuator saturation at some time is a bad design from an economic perspective. MATLAB includes the simulation language Simulink, and although the analysis of nonlinear systems is more difficult than that of linear systems, their simulation is straightforward. It is not the intent of this book to present theoretical methods for studying nonlinear systems, but in introducing Simulink it was felt appropriate to show some of its facilities for simulating nonlinear feedback systems and to give the reader a small appreciation of the effects of nonlinearity on system behavior. This is done through the discussion of a few examples. Further, since initial designs for many nonlinear systems involve consideration of linearized models, this important topic is also covered. A reader who wishes to know more about nonlinear systems or indeed clearly understand all the unique effects that might occur in these systems is referred to appropriate reference [51].

In Sec. 4.1, a brief overview of Simulink, and in particular, the model library of Simulink, are presented. The procedures for Simulink-based modeling and simulation are also given. Nonlinear system modeling problems are presented through examples in Sec. 4.2, where Simulink modeling of nonlinear differential equations, multivariable systems, computer control systems, and time varying systems is illustrated. In Sec. 4.3, a systematic way of modeling piecewise linear single-valued and double-valued nonlinearities is given, and limit cycle problems are explored through simulation approaches. In Sec. 4.4, the linearization of nonlinear systems is presented.

4.1 An Introduction to Simulink

Simulink was developed by the MathWorks in 1990. Its original name was SimuLAB, and the name was changed to the current name in 1992. Two meanings are implied in its name, “simu” and “link.” The word “link” means that the system block diagram can be established

with building blocks and links between the blocks. The word "simu" means its simulation facilities. With the use of the powerful facilities provided in the Simulink program, different systems can be simulated easily and straightforwardly.

4.1.1 Commonly Used Simulink Blocks

The modeling algorithms given in Chapters 2 and 3 cannot be directly applied for nonlinear systems. In this case, the sophisticated Simulink environment can be used to represent such nonlinear system models. Details of Simulink modeling are not given in this book but the interested reader may refer to [28, 29, 52].

To model a nonlinear system, the block library of Simulink should be opened first. It can be opened in one of the following two ways. We shall use the first way throughout the book.

1. Type the `open_system('simulink')` command under the MATLAB prompt. Then, the main window of Simulink will be shown (or brought to the front, when it is already started) as shown in Figure 4.1.
2. Initiate a block library by clicking the Simulink icon in the toolbar in the MATLAB window, as shown in Figure 4.2.

It can be seen that a large number of model blocks are provided in Simulink. Here, the commonly used blocks in control systems are summarized:

1. *Linear system blocks*: The continuous transfer function model, state space model, and zero-pole-gain model are provided in the Continuous group as shown in Figure 4.3. The integrator, differentiator, and time delay blocks are also provided in the group. Moreover, in the Discrete group, the discrete versions of the Continuous block set are provided.
2. *Nonlinear blocks*: The commonly used nonlinearities are provided in the Discontinuity group as shown in Figure 4.4, where nonlinearities such as saturation, dead-zone, and others can be used directly.

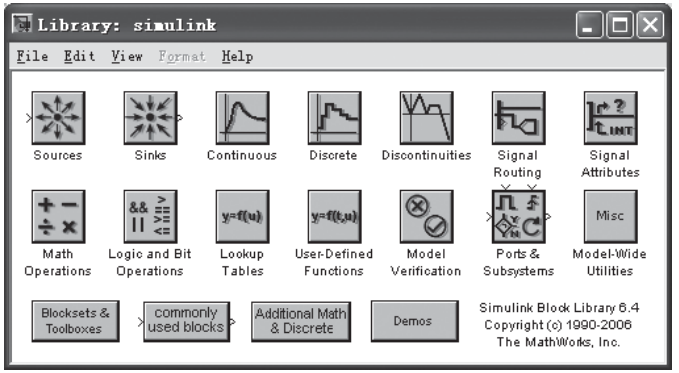


Figure 4.1. *Simulink block library.*

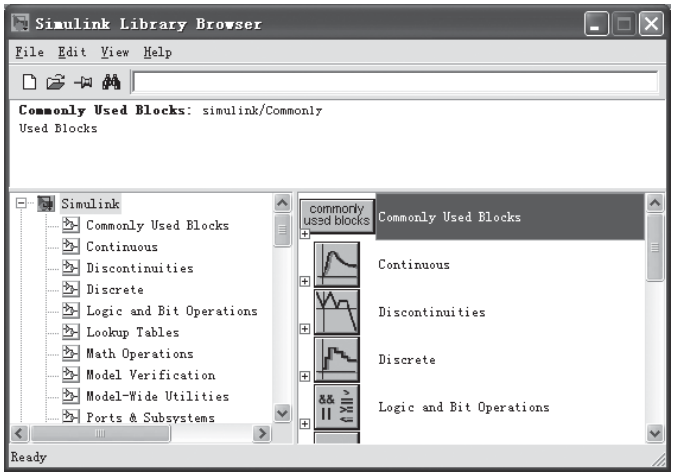


Figure 4.2. *Simulink block library browser.*

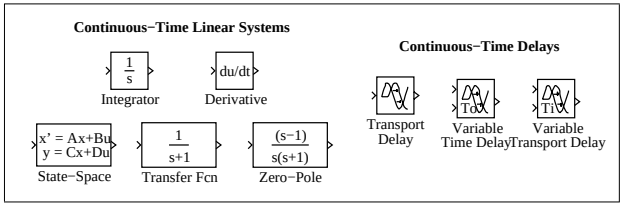


Figure 4.3. *Linear continuous blocks.*

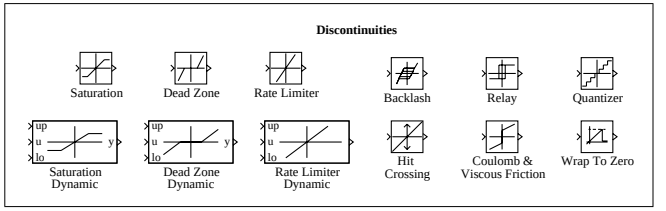


Figure 4.4. *Nonlinear blocks.*

3. *Input and output blocks*: The input signals can be modeled using the blocks in the Sources group, shown in Figure 4.5. The step input, pulse input, and other input signals can be represented by the blocks in the group. In particular, the Inport block can be used to model the input port of the system.

   The output of the system can be displayed with the blocks in the Sinks group, shown in Figure 4.6. One may use the Scope block to show the curves of the selected signals during simulation. The Outport block in the group may be used to indicate the output port of the system.

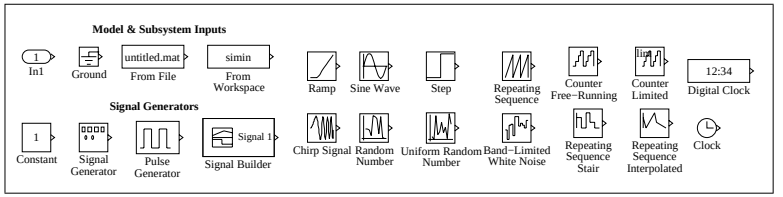


Figure 4.5. *Input (source) blocks.*

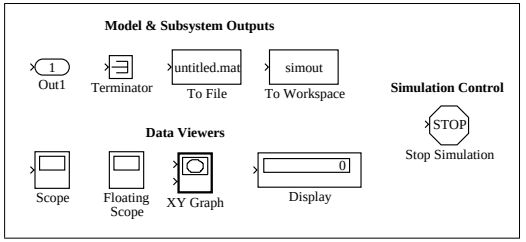


Figure 4.6. *Output (sink) blocks.*

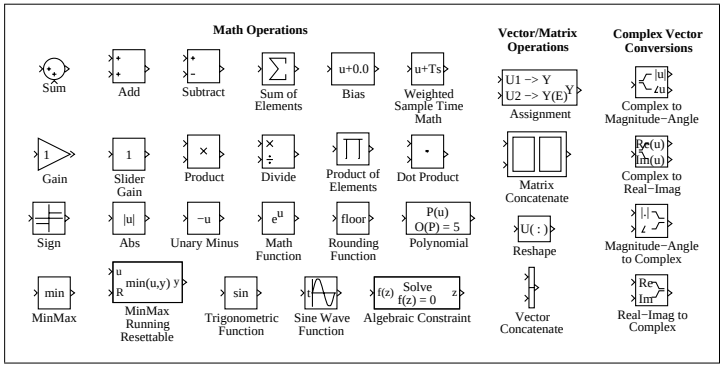


Figure 4.7. *Math blocks.*

4. *Easy mathematical blocks*: The signals in the system should be computed using $+$, $-$, $\times$, $\div$, and other mathematical computations. The blocks in the Math group shown in Figure 4.7 can be used to model these operations.

To easily build up a simulation model for a dynamic system, the user should be familiar with the blocks provided in the Simulink environment. In the following sections, the use of Simulink modeling will be illustrated through examples.

4.1.2 Simulink Modeling

Here we give brief, step-by-step guidelines to Simulink modeling as follows:

1. *Startup and initial preparation*: To enter a model in Simulink format, one should first start up the Simulink environment. One should also open a blank window for the new system model by clicking the File | New menu item.
2. *Draw blocks of the system*: Open the relevant model block library group so that the components of the system can be copied from them. For instance, the icon labeled Continuous in Figure 4.1 contains the blocks shown in Figure 4.3, while the Discontinuities icon contains those shown in Figure 4.4. One can select the blocks in these groups and then drag them into the new system model window.
3. *Specify parameters*: It should be noted that the libraries as shown in Figure 4.3 contain only default models of certain types. For instance, the linear transfer function icon is contained in Figure 4.3, but only with a default $1/(s+1)$ model. To specify the parameters of such a model, one should double click it to get the dialog box, as shown in Figure 4.8, and then fill in the dialog box with the required parameters. It should be noted that the numerators and denominators requested in the dialog box are with the coefficients in the descending order of s.
4. *Draw links*: Once all the blocks needed are copied into the model window, one can draw the links between the blocks to make the system complete. The links between the blocks can be drawn by first clicking the output port of the starting block, and then dragging the mouse to the input port of the ending block. A linkage between the two blocks will then be internally established by Simulink.
5. *Input and output specifications*: One should use the Inport icon in the Sources group to get an input signal for the system and use the Outport icon in the Sinks group to connect to the output of the system.

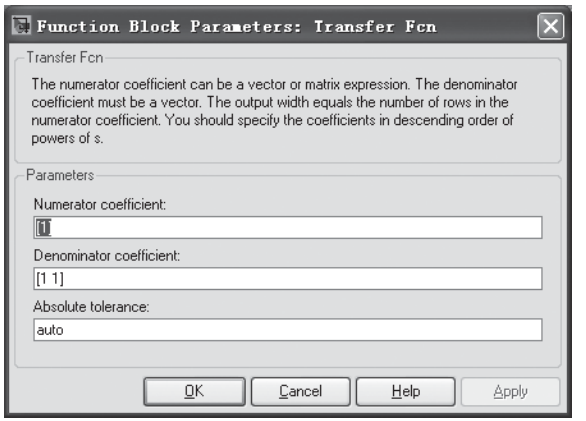


Figure 4.8. *Transfer function parameters dialog box.*

Example 4.1. Consider the nonlinear system model shown in Figure 4.9. It can be seen that there are two nonlinear elements in the system. Using the Simulink program, one can easily draw the block diagram of the nonlinear system, as shown in Figure 4.10.

Using the sophisticated Simulink program, the user can, in theory, draw the block diagram of a control system of any complexity. The Simulink program also allows the user to perform simulation analysis by its menu items or by relevant function calls. The simulation results can be shown on the scopes provided within Simulink, or returned to the MATLAB workspace so that they can be shown by the relevant MATLAB plotting facilities.

4.1.3 Simulation Algorithms and Control Parameters

When the Simulation menu in the Simulink model window is selected, it appears as in Figure 4.11, where the Configuration Parameters menu item can be selected to further

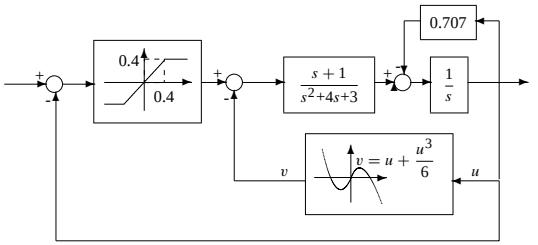


Figure 4.9. *An example of a nonlinear system.*

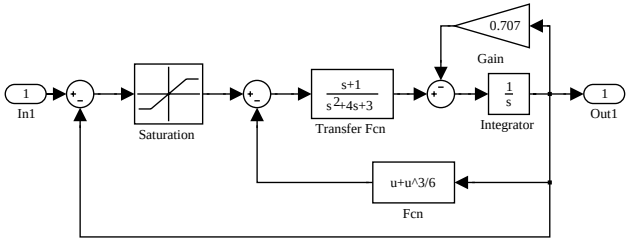


Figure 4.10. *The Simulink model (file: c4mnl.mdl).*

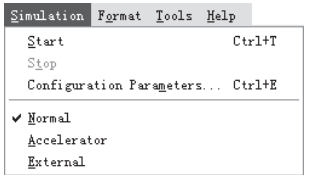


Figure 4.11. *Simulation menu.*

specify the simulation algorithms and control parameters. The dialog box is shown in Figure 4.12. The following parameters can be set from the dialog box.

1. Start time and Stop time edit boxes can specify the start and stop times of the simulation process.
2. The Type list box in the Solver options column has two options, where "variable step" and "fixed step" algorithms can be selected. In order to keep high simulation accuracy, it is suggested to select variable step algorithms. Often the ode45 (Dormand–Prince) and the ode15s (stiff/NDF) algorithms are suitable for simulating control problems.
3. The accuracy of the simulation results can be controlled by the Relative Tolerance option and the Absolute Tolerance option. The default error tolerance is `1e-3`, which means that the relative error may reach to one thousandth. The default value is usually too large and it is suggested to reduce it to `1e-6` or `1e-7`. It should be noted that, although the relative error tolerance is reduced significantly, the computation effort required will not increase much, due to the use of the variable step algorithms.
4. The minimum and maximum allowed computation step sizes can be set by filling in the Min step size and Max step size edit boxes. If the actual step size goes beyond the specified step size range in variable step simulation, an error message box will be displayed.
5. The warning and error messages can be set by the Diagnostics column.

After completing the specification of the control parameters, the item Simulation/Start can be selected to initiate the simulation process. A variable `tout` will be returned automatically, and the matrix `yout` can also be generated when the output port is used in the Simulink model. The function `plot(tout,yout)` can be used to show the simulation results.

The `sim()` function can also be used to initiate the simulation process. The syntax of the function is

`[t,x,y]=sim(model_name, tf, options)`

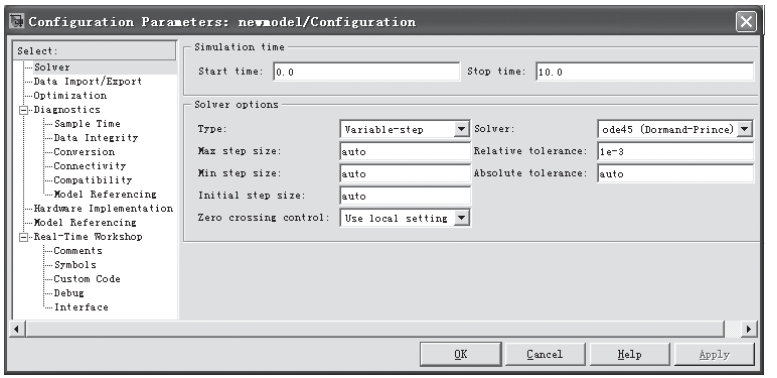


Figure 4.12. *Dialog box of control parameters.*

where, the "model_name" is the file name of the Simulink model, with the suffix `.mdl` omitted. The argument t_f is the stopping time of simulation. The returned arguments $\boldsymbol{t}$, $\boldsymbol{x}$, and $\boldsymbol{y}$ are, respectively, the time vector, state matrix, and output matrix of the system.

The control parameters `options` can be specified by the `simset()` function, whose syntax is

```
options=simset(property, parameter 1, property 2, parameter 2, ···)
```

where the "property" is the name of the parameter, while the "parameter" is the value associated with it. The relevant properties can be listed with the command `help simset`. For instance, one may change the relative error tolerance by setting its `RelTol` property to 10^{-7} by the statement `options=simset('RelTol',1e-7)` or simply by `options.RelTol=1e-7`.

When the `options` variable is modified, it can be used in the simulation process by filling it to the `sim()` function.

4.2 Modeling of Nonlinear Systems by Examples

A series of examples related to control systems will be used to illustrate the use of the Simulink program. A nonlinear ordinary differential equation, (ODE) example is used first, followed by a multivariable system, a computer controlled system, and a time varying system. It will be appreciated from these examples that systems with significant complexity can be simulated using Simulink.

Example 4.2 (block diagram solution of nonlinear differential equations). Consider the well-known Rössler chaotic equation, whose mathematical form is

$$\begin{cases} \dot{x}(t) = -y(t) - z(t), \\ \dot{y}(t) = x(t) + ay(t), \\ \dot{z}(t) = b + [x(t) - c]z(t). \end{cases}$$

Selecting $a = b = 0.2$, $c = 5.7$, and $x(0) = y(0) = z(0) = 0$, the Simulink model can be constructed and simulation analysis can be applied to the given system.

A simple trick for simulating ODEs is by assigning each integrator to a state variable to represent the output as the state $x_i(t)$; then the input to the integrator is $\dot{x}_i(t)$. The Simulink model in Figure 4.13 can be easily established. The control parameters for simulation can then be set accordingly. If one starts the simulation process, two variables `tout` and `yout` will be returned. The time response of the three states are obtained with the `plot(tout,yout)` command as shown in Figure 4.14(a).

If the states $x_1(t)$, $x_2(t)$, and $x_3(t)$ are used as the three axes, the phase space trajectories can be drawn as shown in Figure 4.14(b). The function `comet3()` can further be used to draw dynamically the trajectories of the phase space curve:

```
>> plot3(yout(:,1),yout(:,2),yout(:,3))  % or further using comet3
   comet3(yout(:,1),yout(:,2),yout(:,3))
```

Many blocks in Simulink support vector operations, i.e., the block can easily process the case when several inputs are placed into a vector signal by using the Mux block. If such

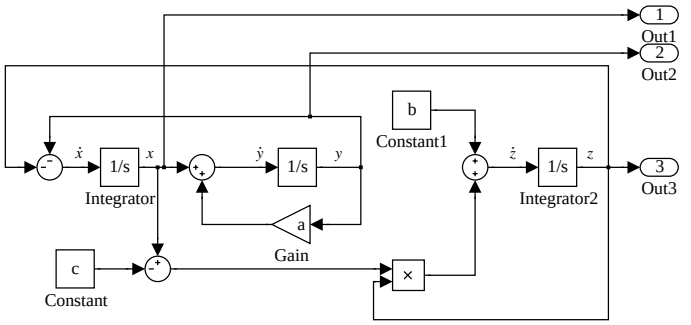
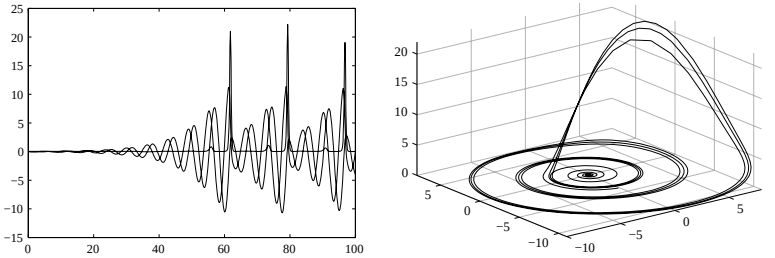
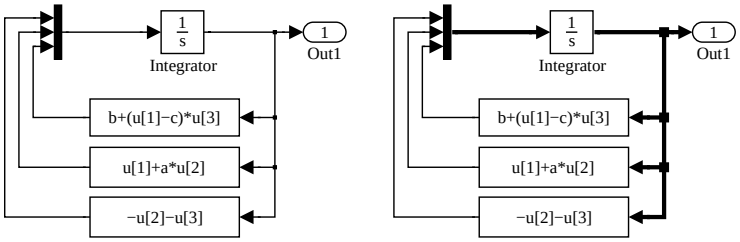


Figure 4.13. *Simulink model of Rössler equations (file: c4mrossler.mdl).*



(a) time response of states (b) phase space curves

Figure 4.14. *Simulation results of the Rössler equations.*



(a) improved (file: c4mross1a.mdl) (b) thickened vectors (file: c4mross1b.mdl)

Figure 4.15. *Another simulink description of the Rössler equations.*

a signal is put through an integrator block, the output signal is also a vector, whose channels are the integrals of the input channels. Thus, the blocks in Figure 4.15(a) can be used to rewrite the blocks in the Simulink model.

In the model, the block Fcn is used to define the mathematical operation on the input signals. The input to the block is the state of the system, and the input to the Fcn block is

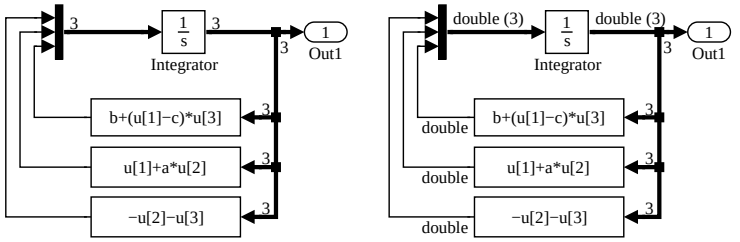


Figure 4.16. *Enhancements of vector signal blocks.*

denoted by `u`. If `u` is a vector, `u[i]` can be used to denote the components of its ith channel. Thus the model constructed is much easier than the one established in Figure 4.13, and it is more easily maintained.

The vector signals can be visually enhanced in Simulink. For instance, the Format/Wide nonscalar lines menu in the model window allows the thickened line expression for vector signals, as shown in Figure 4.15(b). If the user selects the Format/Signal dimensions menu item, the dimensions of the signal will be displayed over the vector signals. For instance, since the state variable is three-dimensional a "3" is displayed, as shown in Figure 4.16(a). The Format menu further provides the item Format/Port data types, which allows the display of data types in the signals, as shown in Figure 4.16(b). These facilities make the physical meanings more informative.

Example 4.3 (simulation of multivariable delayed systems). Consider the multivariable system in Example 3.19. Since there exists time delays in the transfer function matrix of the open-loop system, the closed-loop function matrix cannot be expressed with the `feedback()` function. In Example 3.19, Padé approximations are used to approximate the delay terms. There was no other way to verify the accuracy of the simulation results. With the use of Simulink, the accurate model can be established as shown in Figure 4.17. In the simulation model, the two input signals are assigned as the variables `u1` and `u2`.

Recalling the Padé approximation used in Example 3.19, we can use the `step()` function to find the approximate simulation results for the multivariable system above when each input acts individually:

```
>> g11=tf(0.1134,[1.78 4.48 1],'ioDelay',0.72);
   g21=tf(0.3378,[0.361 1.09 1],'ioDelay',0.3);
   g12=tf(0.924,[2.07 1]); g22=tf(-0.318,[2.93 1],'ioDelay',1.29);
   G=[g11, g12; g21, g22];
   [n1,d1]=paderm(0.72,0,2); g11.ioDelay=0; g11=tf(n1,d1)*g11;
   [n1,d1]=paderm(0.30,0,2); g21.ioDelay=0; g21=tf(n1,d1)*g21;
   [n1,d1]=paderm(1.29,0,2); g22.ioDelay=0; g22=tf(n1,d1)*g22;
   G1=[g11, g12; g21, g22];
   Kp=[0.1134,0.924; 0.3378,-0.318]; G2=ss(G1*Kp);
   [y1,x1,t1]=step(G2.a,G2.b,G2.c,G2.d,1,15);
   [y2,x2,t2]=step(G2.a,G2.b,G2.c,G2.d,2,15);
```

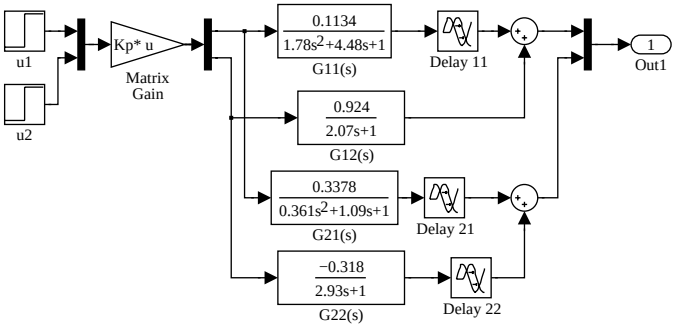



Figure 4.17. *Simulink model of the multivariable system (c4mmimo.mdl).*

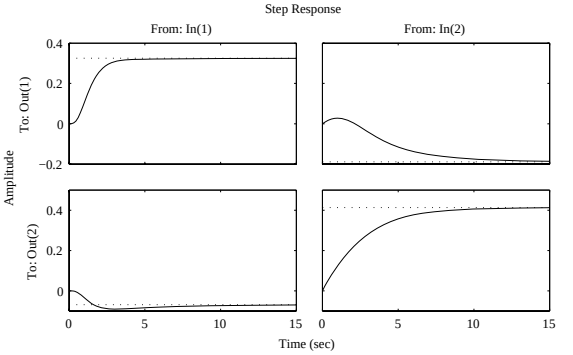


Figure 4.18. *Comparisons of the multivariable system simulation results.*

By simulating the system with Simulink, we can obtain the step response of the system driven by the two signals individually. They can be drawn together with the approximation results, as shown in Figure 4.18:

```
>> u1=1; u2=0; [tt1,x1,yy1]=sim('c4mmimo',15);
   u1=0; u2=1; [tt2,x2,yy2]=sim('c4mmimo',15);
   subplot(221), plot(t1,y1(:,1),':',tt1,yy1(:,1))
   subplot(222), plot(t1,y1(:,2),':',tt1,yy1(:,2))
   subplot(223), plot(t2,y2(:,1),':',tt2,yy2(:,1))
   subplot(224), plot(t2,y2(:,2),':',tt2,yy2(:,2))
```

It can be seen from Figure 4.18 that the approximate simulation results are quite close to the exact results.

Example 4.4 (computer control system simulation). Consider the classical computer controlled system [53] shown in Figure 4.19, where the controller is a discrete controller, with sampling interval of T seconds. The ZOH is the zero-order-hold, and the plant is given by

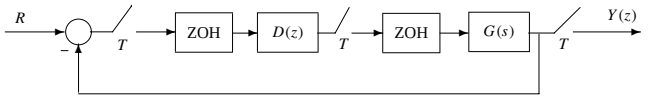


Figure 4.19. *Block diagram of a computer controlled system.*

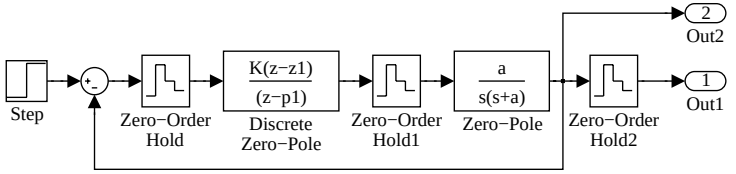


Figure 4.20. *Simulink model for a computer controlled system (c4mcompc.mdl).*

a continuous model. Assume that the plant and controller are given, respectively, as

$$G(s) = \frac{a}{s(s+1)}, \quad D(z) = \frac{1-\mathrm{e}^{-T}}{1-\mathrm{e}^{-0.1T}} \frac{z-\mathrm{e}^{-0.1T}}{z-\mathrm{e}^{-T}},$$

where $a = 0.1$. It is not possible to write out the corresponding differential equation for this system, since both continuous and discrete elements exist in the system.

Simulink has the advantage that it can solve this type of hybrid problem. From the given block diagram of the system, the Simulink model can be easily established, as shown in Figure 4.20. In the system model, a few variables a, T, z_1, p_1, K are used, where the former two should be specified by the user, while the latter three should be calculated. In the first ZOH block, the sampling interval is set to T, and for simplicity, the rest of the blocks are assigned to -1, indicating that they inherit the sampling intervals of their input signals. It is not necessary to put the value of T in each discrete block.

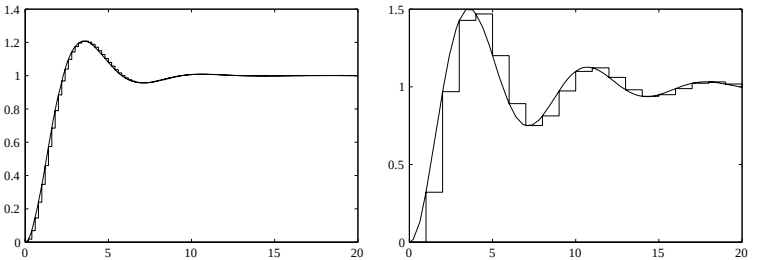
If $a = 0.1$, and the sampling interval is given by $T = 0.2$ seconds, the step response of the closed-loop system can be obtained as shown in Figure 4.21(a):

```
>> T=0.2; a=0.1; z1=exp(-0.1*T); p1=exp(-T); K=(1-p1)/(1-z1);
   [t,x,y]=sim('c4mcompc',20);
   plot(t,y(:,2)); hold on; stairs(t,y(:,1))
```

If the sampling interval is further increased to $T = 1$ second, the step response of the closed-loop system can be obtained as shown in Figure 4.21(b). It can be seen that when the sampling interval increases, the difference between the continuous and discrete signals increases:

```
>> T=1; z1=exp(-0.1*T); p1=exp(-T); K=(1-p1)/(1-z1);
   [t,x,y]=sim('c4mcompc',20);
   plot(t,y(:,2)); hold on; stairs(t,y(:,1))
```

In fact, with the conversion algorithm given in Chapter 2, the discrete version of the plant model can be found under sampling interval T. Thus the closed-loop discrete



(a) $T = 0.2$ seconds (b) $T = 1$ second

Figure 4.21. *Step responses for different sampling intervals.*

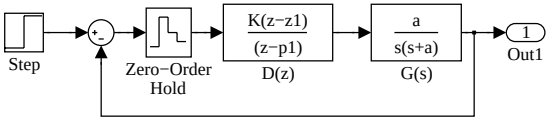


Figure 4.22. *Simplified computer control system (file: c4mcomc1.mdl).*

system can be obtained. The step response of the system can be obtained with the following statements:

```
>> T=0.2; z1=exp(-0.1*T); p1=exp(-T); K=(1-p1)/(1-z1);
   Dz=zpk(z1,p1,K,'Ts',T); G=zpk([],[0;-a],a);  Gz=c2d(G,T);
   GG=zpk(feedback(Gz*Dz,1)), step(GG)
```

The controller can be obtained as

$$G_c(z) = \frac{0.018187(z+0.9934)(z-0.9802)}{(z-0.9802)(z^2-1.801z+0.8368)}.$$

The statements can be used to get the same results with the Simulink model, and they are much simpler. However, this method has its own limitations.

Further investigation of the Simulink model shows that the ZOH after the controller $D(z)$ is redundant, since the output of $D(z)$ is already a discrete signal and remains the same within a sampling interval. Thus it can be removed. The ZOH on the output signal can also be removed. The simulation model can finally be reduced to the one shown in Figure 4.22, without causing any problems.

Of course, the system can be further simplified in the Simulink model, since all the ZOHs can be removed, as shown in Figure 4.23. Although this is not an official method from a theoretical aspect, the approximation is correct under Simulink.

Example 4.5 (simulation of time varying systems). Assume that the plant model is given by $\ddot{y}(t) + \mathrm{e}^{-0.2t}\dot{y}(t) + \mathrm{e}^{-5t}\sin(2t+6)y(t) = u(t)$. Now consider a PI control system, shown in Figure 4.24, where $K_p = 200$ and $K_i = 10$. The width of the saturation element is $\delta = 2$.

It can be seen that, apart from the time varying block, the modeling of the rest of system is very straightforward. In the time varying part, assume that the first-order explicit

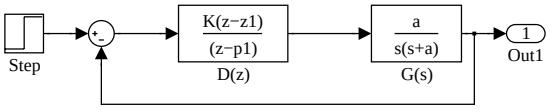


Figure 4.23. *Further simplified Simulink model (file: c4mcomc2.mdl).*

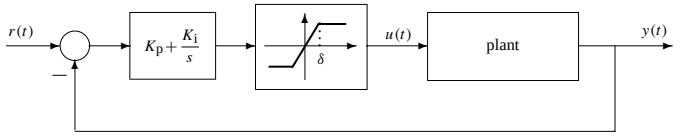


Figure 4.24. *Block diagram of the time varying system.*

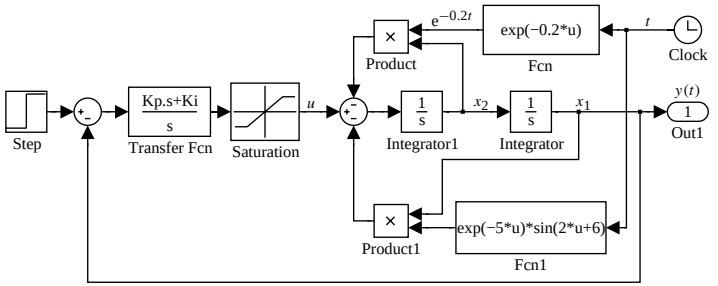


Figure 4.25. *Simulink model (file: c4mtimv.mdl).*

differential equations $x_1(t) = y(t)$, $x_2(t) = \dot{y}(t)$ can be established as

$$\begin{cases} \dot{x}_1(t) = x_2(t), \\ \dot{x}_2(t) = -\mathrm{e}^{-0.2t}x_2(t) - \mathrm{e}^{-5t}\sin(2t+6)x_1(t) + u(t). \end{cases}$$

Similar to the method in Example 4.2, an integrator should be assigned to each state variable. The Simulink model in Figure 4.25 can be established, where the time varying function can be set up with Simulink blocks.

Once the simulation model is established, the following MATLAB statements can be issued to simulate the system. The step response of the time varying system can be obtained as shown in Figure 4.26.

```
>> opt=simset('RelTol',1e-8); Kp=200; Ki=10;
   [t,x,y]=sim('c4mtimv',10,opt); plot(t,y)
```

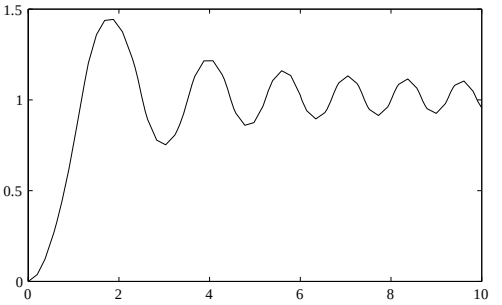



Figure 4.26. *Step response of the time varying system.*

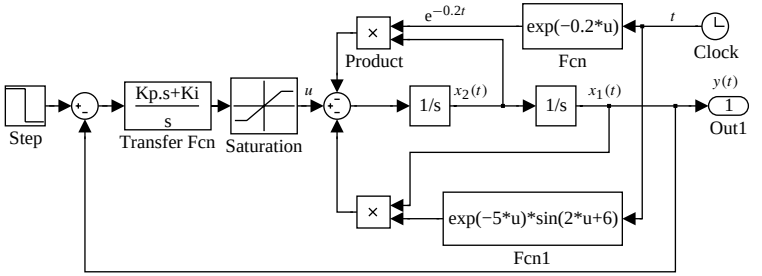


Figure 4.27. *Impulse response of a time varying system (c4mtimva.mdl).*

Example 4.6 (impulse response). Consider again the time varying model of Example 4.5. Assume that the input signal is a unit impulse signal. Here Simulink is used to find the impulse response of the time varying system.

Since there is no unit impulse block provided in Simulink, the step input block can be used instead to approximate it. If the step time is a, where a is an extremely small value, the initial value of the step block can be set to $1/a$, and the final time can be set to 0. The simulation model in Figure 4.27 can be used to model the whole system.

Theoretically, when $a \to 0$, the impulse signal can be approximated. In real simulation, a can be set to relatively large values, for instance, $a = 0.001$. The impulse response of the system can be obtained with the following MATLAB statements, as shown in Figure 4.28:

```
>> opt=simset('RelTol',1e-8); Kp=200; Ki=10; a=0.001;
   [t,x,y]=sim('c4mtimva',10,opt); plot(t,y)
```

In fact, even though a is selected as a large value, for instance $a = 0.1$, a very good approximation to the results can still be obtained.

In real applications, the inputs with arbitrary periodic signals can be established as well with the use of the Repeating Sequence block. Even more complicated signals or system behaviors can be modeled with the use of the S-functions.

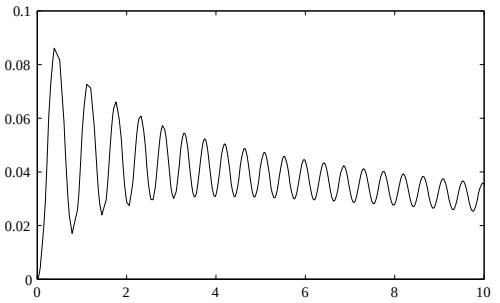


Figure 4.28. *Impulse response of the time varying system.*

4.3 Nonlinear Elements Modeling

A well-known technique for trying to predict limit cycles in nonlinear systems is the describing function method [51]. Since this is an approximate method, it is very useful for comparative purposes when determining solutions by simulation. In this section nonlinearity modeling is discussed in more detail and then a simple system possessing a limit cycle is simulated.

4.3.1 Modeling of Piecewise Linear Nonlinearities

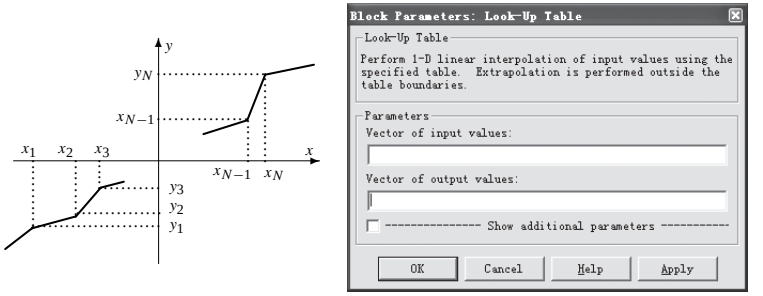
Static nonlinearities of any complexity can be constructed using existing Simulink blocks. In this section, Simulink modeling of single-valued and double-valued nonlinearities is presented.

The single-valued static nonlinearity can be easily established with the one-dimensional look-up table block. For instance, consider the piecewise nonlinearity shown in Figure 4.29(a), for known turning points (x_1, y_1), (x_2, y_2), ..., (x_{N-1}, y_{N-1}), (x_N, y_N). One may select a point x_0 such that $x_0 < x_1$. The value y_0 can be easily computed from the nonlinear behavior. Also for another point x_{N+1} such that $x_{N+1} > x_N$, the value of y_{N+1} can be obtained. Thus the two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ can be established such that

$$\boldsymbol{x} = [x_0, x_1, x_2, \ldots, x_N, x_{N+1}]\ ; \quad \boldsymbol{y} = [y_0, y_1, y_2, \ldots, y_N, y_{N+1}]\ ;$$

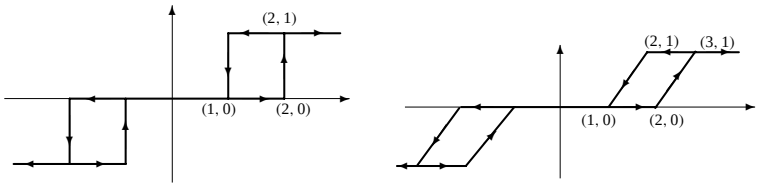
Double click the one-dimensional Look-Up Table block, to display the dialog box as shown in Figure 4.29(b). One should specify in the Vector of input values and Vector of output values edit boxes, respectively, the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and then a single-valued nonlinearity can be set up successfully.

The construction of a general doubled-valued nonlinearity is not as simple as constructing the single-valued case. For specific input signals it is possible to define these nonlinearities in terms of the input and its derivative, since one path around the nonlinear element will be taken when the input is increasing and another when it is decreasing. Thus, the approach will be valid for a sinusoidal input but not for a random input, or indeed for an input whose derivative changes within the double-valued region. An approach which can be use for these restricted situations is given below.



(a) single-valued nonlinearity (b) Dialog box of parameters

Figure 4.29. *Construction of single-valued nonlinearities.*



(a) relay loop (b) saturated relay loop

Figure 4.30. *Loop function expression.*

Example 4.7. When there exist loops in the nonlinearity, apart from a few existing blocks in the Simulink block library, a general nonlinearity cannot be easily established. The Switch block can be used to tackle the problem.

Now consider the two double-valued loop nonlinearities shown in Figures 4.30(a) and (b). First, consider the loop nonlinearity shown in Figure 4.30(a). It can be seen that the loop function may be expressed by a single-valued nonlinearity when the input signal is increasing and by another single-valued nonlinearity when decreasing. This means that the single-valued nonlinearity is conditional. For instance, the two single-valued nonlinearities in Figure 4.31 can be used to express the loop nonlinearity in Figure 4.29(a).

The Simulink block Memory can be used to extract the input signal at the previous time instance. Thus, the Simulink model shown in Figure 4.32 can be used to express the double-valued loop nonlinearity. A comparative block is used to check whether the input signal is increasing or not, i.e., by checking whether the current value is greater than its previous value or not. A switch block can be used to control the single-valued block selection, with the Threshold set to 0.5. If it is increasing, the switch block is set to the increasing block single-valued nonlinearity; otherwise the decreasing one is chosen. In this way, the double-valued loop nonlinearity can be established.

The two single-valued nonlinearities can be expressed individually by two table-look-up blocks given by

$$\boldsymbol{x}_1 = [-3, -1, -1+\epsilon, 2, 2+\epsilon, 3], \quad \boldsymbol{y}_1 = [-1, -1, 0, 0, 1, 1],$$
$$\boldsymbol{x}_2 = [-3, -2, -2+\epsilon, 1, 1+\epsilon, 3], \quad \boldsymbol{y}_2 = [-1, -1, 0, 0, 1, 1],$$

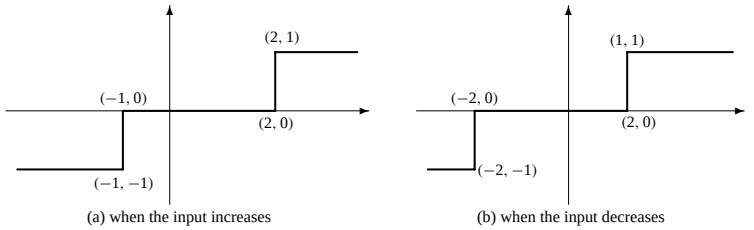


Figure 4.31. *Loop function can be expressed as single-valued functions.*

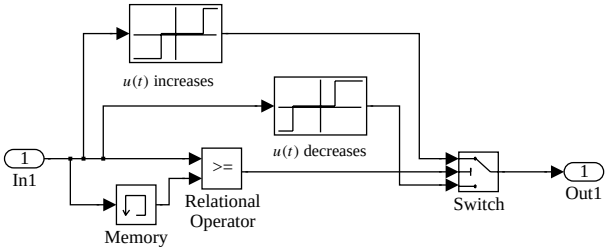


Figure 4.32. *Double-valued relay nonlinearity (file: c4mloop.mdl).*

where ϵ can be set to a very small number. For instance, it can be set to the MATLAB reserved constant `eps`.

Consider now the nonlinearity shown in Figure 4.30(b). The previously established Simulink model can still be used to model the nonlinearity. The new sets of data should be used to modify the table-look-up blocks

$$\boldsymbol{x}_1 = [-3, -2, -1, 2, 3, 4], \quad \boldsymbol{y}_1 = [-1, -1, 0, 0, 1, 1],$$
$$\boldsymbol{x}_2 = [-3, -2, -1, 1, 2, 3], \quad \boldsymbol{y}_2 = [-1, -1, 0, 0, 1, 1].$$

The Simulink model for the new loop nonlinearity can be expressed as shown in Figure 4.33.

It can be seen therefore that single-valued or double-valued static nonlinearities of any complexity can easily be modeled by Simulink blocks using similar methods. The nonlinearity can be used directly in simulation.

Example 4.8. Consider the double-valued nonlinearity shown in Figure 4.30(b). One may observe the distortion in the sinusoidal signals through it. To observe this distortion, the Simulink model can be established as shown in Figure 4.34.

If the magnitude A of the sinusoidal signal is set to 2, 4, and 8, the output of the nonlinearity can be obtained as shown in Figure 4.35.

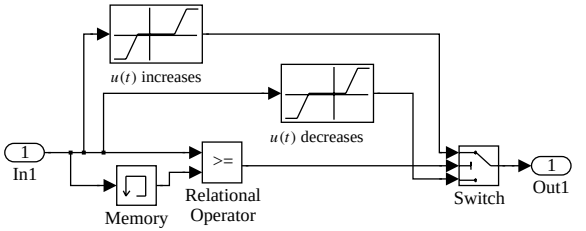


Figure 4.33. *Double-valued nonlinearity (file: c4mloopa.mdl).*

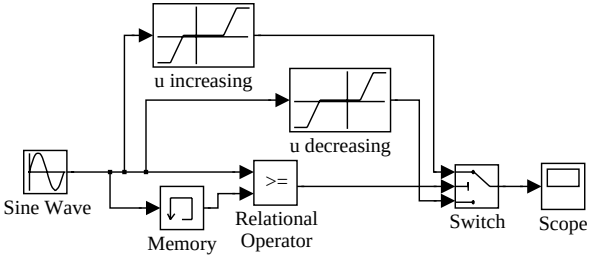


Figure 4.34. *Simulink model for sinusoidal distortions (file: c4msin.mdl).*

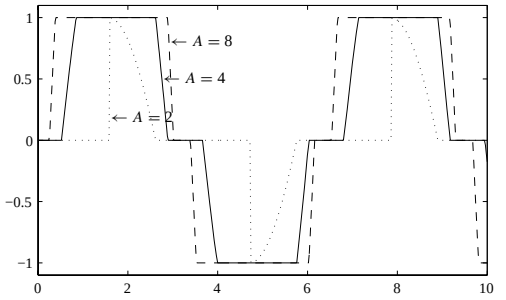


Figure 4.35. *Nonlinear distortion of sinusoidal inputs.*

4.3.2 Limit Cycles of Nonlinear Systems

Nonlinear systems can have behaviors which are not present in linear systems. One such situation is the existence of a limit cycle, or self-oscillation, which can be attained when the system is released from different initial conditions.

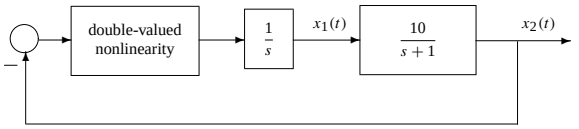


Figure 4.36. *Block diagram of a nonlinear feedback system in Example* 4.9.

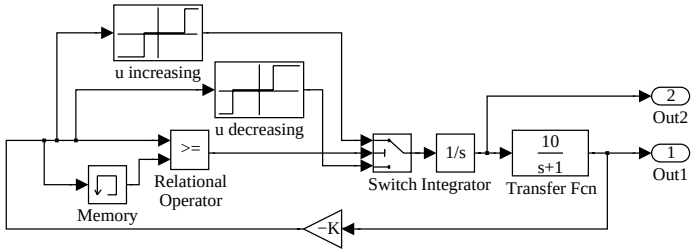
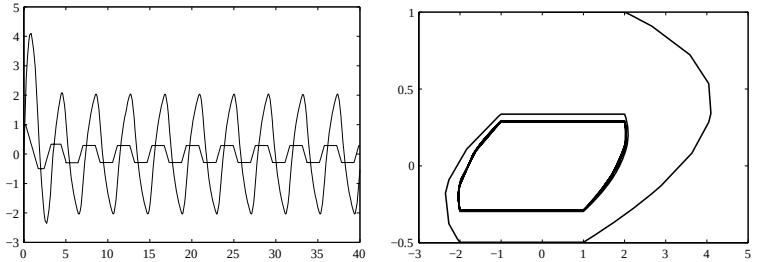


Figure 4.37. *Simulink model for Example* 4.9 *(file:c4mlimcy.mdl).*



(a) time domain simulation (b) phase plane trajectory

Figure 4.38. *Simulation result of nonlinear feedback system.*

Example 4.9. Consider the typical nonlinear feedback system shown in Figure 4.36, with the nonlinear element shown in Figure 4.30(a), whose Simulink model is established in Figure 4.32. For such a feedback system, the Simulink model can be built up as shown in Figure 4.37. In the simulation model, the initial value of the integrator is set to one.

Let the simulation terminate time be 40 seconds, and to keep a high accuracy set the relative error tolerance (Relative tolerance) to 10^{-8}, or to an even smaller value. With the following MATLAB code, the time response of the system, when there is no external input signal, can be obtained as shown in Figure 4.38(a).

```
>> [t,x,y]=sim('c4mlimcy',40); plot(t,y)
```

It can be seen that the signals $x_1(t)$ and $x_2(t)$ reach steady-state oscillations after the initial transients have died away. With the graphics functions `plot(y(:,1),y(:,2))`

provided in MATLAB, the phase plane trajectory appears as in Figure 4.38(b). It can be seen that the phase plane trajectory settles down at a closed curve, which is referred to as a limit cycle. A limit cycle is an interesting feature that may occur in a nonlinear system.

4.4 Linearization of Nonlinear Models

Linear systems are far easier to analyze and design than nonlinear ones. Unfortunately, system models which must be dealt with in practice are rarely linear. In this case a linear approximation of the system is often required to simplify the analysis and design procedures. One procedure for doing this is the linearization process.

System linearization extracts an approximate linear model, i.e., a linear model in a neighborhood of the operating point.

Consider the nonlinear dynamic system model

$$\dot{x}_i(t) = f_i(x_1, x_2, \ldots, x_n, \boldsymbol{u}, t), \quad i = 1, 2, \ldots, n. \tag{4.1}$$

An operating point is defined as the values of the state and input variables when the derivatives of the state variables approach zero. It can be obtained by solving the nonlinear equations defined in (4.1) such that

$$f_i(x_1, x_2, \ldots, x_n, \boldsymbol{u}, t) = 0, \quad i = 1, 2, \ldots, n, \tag{4.2}$$

which can be solved numerically. Denote by $\boldsymbol{x}_0$ the operating point with an input signal $\boldsymbol{u}_0$. The nonlinear system can be approximated by

$$\Delta\dot{x}_i = \sum_{j=1}^{n} \left.\frac{\partial f_i(\boldsymbol{x}, \boldsymbol{u})}{\partial x_j}\right|_{\boldsymbol{x}_0, \boldsymbol{u}_0} \Delta x_j + \sum_{j=1}^{p} \left.\frac{\partial f_i(\boldsymbol{x}, \boldsymbol{u})}{\partial u_j}\right|_{\boldsymbol{x}_0, \boldsymbol{u}_0} \Delta u_j. \tag{4.3}$$

Using the new state variables $\boldsymbol{z}(t) = \Delta\boldsymbol{x}(t)$ for the system model, the linearized model can be obtained as follows:

$$\dot{\boldsymbol{z}}(t) = \boldsymbol{A}_l|_{\boldsymbol{x}_0, u_0}\, \boldsymbol{z}(t) + \boldsymbol{B}_l|_{\boldsymbol{x}_0, u_0}\, \boldsymbol{v}(t), \tag{4.4}$$

where $\boldsymbol{v}(t) = \Delta\boldsymbol{u}(t)$, and

$$\boldsymbol{A}_l = \begin{bmatrix} \partial f_1/\partial x_1 & \cdots & \partial f_1/\partial x_n \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1 & \cdots & \partial f_n/\partial x_n \end{bmatrix}, \quad \boldsymbol{B}_l = \begin{bmatrix} \partial f_1/\partial u_1 & \cdots & \partial f_1/\partial u_p \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial u_1 & \cdots & \partial f_n/\partial u_p \end{bmatrix}. \tag{4.5}$$

Useful functions for performing the linearization of nonlinear systems are provided in Simulink. The user can use the `trim()` function to find the operating point. The syntax of the `trim()` function is

```
[x,u,y,z]=trim(model_name,x0,u0)
```

where `model_name` is the Simulink model name. The variables $\boldsymbol{x}_0$, $\boldsymbol{u}_0$ are the initial guess for the states and input at the operating point. A constrained optimization technique is used

to obtain the operating point. For systems without nonlinear elements, the initial guess of $\boldsymbol{x}_0, \boldsymbol{u}_0$ can be omitted. The actual operating point is returned in $\boldsymbol{x}$, $\boldsymbol{u}$, $\boldsymbol{y}$, and the values of the derivatives of the state variables are returned in $\boldsymbol{z}$. Theoretically speaking, the derivatives of the state variables at the operating point should be equal to 0.

Example 4.10. Assume that the Simulink model of the nonlinear system is as given in Figure 4.10 and the name of the Simulink model file is `c4mnl.mdl`. Using the MATLAB statements

```
>> [x0,u0,y,dx]=trim('c4mnl',[],1)
```

the operating point of the system can be obtained as $\boldsymbol{x}_0 = [0.1281, 0, 0.0905]^{\mathbf{T}}$, $u_0 = 1$. Having obtained the operating point, we can obtain the linearized state space model of the nonlinear system by the corresponding MATLAB function provided in Simulink. The syntaxes of these function are

```
[A,B,C,D]=linmod2(model_name,x0,u0)
[A,B,C,D]=linmod(model_name,x0,u0),     % delayed systems
[A,B,C,D]=dlinmod(model_name,x0,u0),    % discrete-time systems
```

where $\boldsymbol{x}_0$, $\boldsymbol{u}_0$ denote the operating point. The linearized state space expression is returned in ($\boldsymbol{A}$,$\boldsymbol{B}$,$\boldsymbol{C}$,$\boldsymbol{D}$). If one omits the $\boldsymbol{x}_0$, $\boldsymbol{u}_0$ variables (operating point), the default linearized model can be obtained. It should be noted that different types of systems can be linearized with the different functions listed above.

Consider again the nonlinear system model analyzed above. Under the above operating point and, with the MATLAB statements

```
>> [x0,u0,y,dx]=trim('c4mnl',[],1);
   [A,B,C,D]=linmod2('c4mnl',x0,u0); G1=ss(A,B,C,D),
```

the state space expression of the linearized model of the original nonlinear system can be obtained as

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} -0.707 & 1 & 1 \\ -2 & -4 & -3 \\ 0 & 1 & 0 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} u(t), \quad y(t) = [1, 0, 0]\boldsymbol{x}(t).$$

Example 4.11. Consider again the linear DC (direct current) motor problem given in Example 2.11. The Simulink description is shown in Figure 4.39, and by using the MATLAB commands

```
>> [a,b,c,d]=linmod2('c4dcmot');
   G=ss(a,b,c,d); minreal(zpk(G))
```

the pole-zero-gain model can easily be obtained as

$$G(s) = \frac{1118525021.9491(s+5.882)(s+6.667)}{(s+179.6)(s+98.9)(s+8.3)(s^2+0.89s+5.8)(s^2+68.26s+2248)}$$

which is the same as the one obtained earlier in Example 2.11.

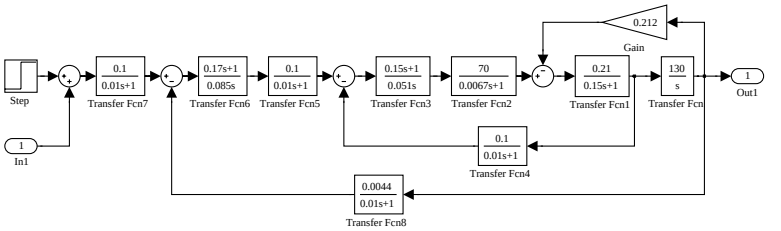


Figure 4.39. *Simulink model for the DC motor system (file: c4dcmot.mdl).*

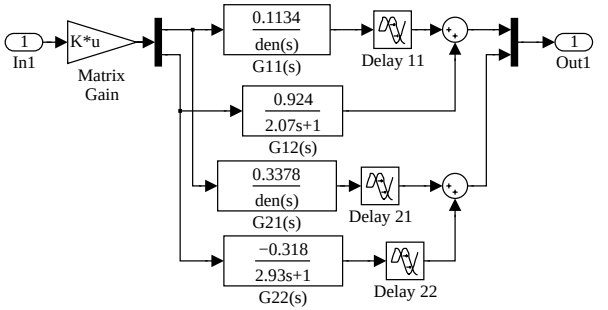


Figure 4.40. *MIMO Simulink model (file: c4mmdly1.mdl).*

Example 4.12. Consider the multivariable system in Example 4.3. If a linearized model is expected, the Padé approximation should be used in linearizing the delay terms. One may set the edit box of Pade order (for linearization) to 2 in the delay blocks, where second-order Padé approximation can be used. The Simulink model for the multivariable system can then be constructed as shown in Figure 4.40.

When the Simulink model is established, the following statements can be used in the linearization process, and one can obtain the linear state space model. The exact simulation results are obtained, together with the linearized model, as shown in Figure 4.41. It can be seen that the simulation results of the linearized model are very accurate.

```
>> Kp=[0.1134,0.924; 0.3378,-0.318];
   [A,B,C,D]=linmod('c4mmdly1'), step(ss(A,B,C,D))
```

The state space model can be extracted from the Simulink system

$$\boldsymbol{A} = \begin{bmatrix} -8.33 & -23.15 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0637 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.4831 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -20 & -133.33 & 0 & 0 & 0 & 0 & 0 & 0.9357 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -4.6512 & -7.2111 & 0 & 0 & 0 & 0 & -0.1085 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2.5169 & -0.5618 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3.0194 & -2.7701 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.3413 \end{bmatrix},$$

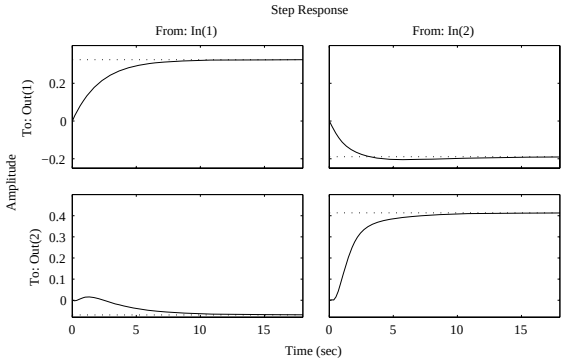


Figure 4.41. *Comparisons of exact and approximate results.*

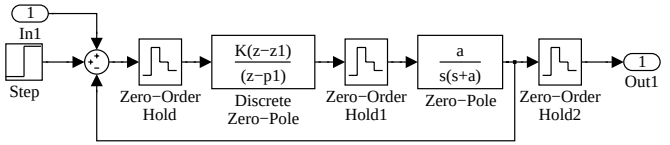


Figure 4.42. *Another Simulink model (file: c4mcomp2.mdl).*

$$\boldsymbol{B}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0.3378 & 0 & 0 & 0 & 0 & 0.1134 & 0 & 0.1134 & 0 & 0.3378 \\ 0 & 0 & -0.318 & 0 & 0 & 0 & 0 & 0.924 & 0 & 0.924 & 0 & -0.318 \end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix} -16.667 & 0 & 0.44638 & 0 & 0 & 0 & 0 & 0 & 0.063708 & 0 & 0 & 0 \\ 0 & 0 & 0 & -40 & 0 & -9.3023 & 0 & 0 & 0 & 0 & 0.93573 & -0.10853 \end{bmatrix}.$$

It should be noted that the `linmod2()` function cannot be used in the linearization process, since the delay term cannot be handled correctly.

Example 4.13. Consider the computer controlled system studied in Example 4.4. For simulation analysis and linearization, the Simulink model can be constructed with the `inport` and `outport` used in the system model. The final Simulink model is shown in Figure 4.42.

The following statements can be used in the linearization problem, where the discrete-time transfer function can finally be obtained:

```
>> T=0.2; a=0.1; z1=exp(-0.1*T); p1=exp(-T); K=(1-p1)/(1-z1);
   [A,B,C,D]=dlinmod('c4mcomp2'); zpk(ss(A,B,C,D,'Ts',0.2))
```

The linearized model can be written as

$$G(z) = \frac{0.018187(z+0.9934)(z-0.9802)}{(z-0.9802)(z^2-1.801z+0.8368)}.$$

It can be seen that the results are exactly the same as those of Example 4.4. It should be noted that `dlinmod()` should be used rather than `linmod2()`.

Problems

1. Become familiar with the Simulink library groups and observe the use of the commonly used blocks so that you can easily use them in solving simulation problems. A handy new group which contains the frequently used blocks can be set up for later use.

2. Consider the linear differential equation

$$y^{(4)} + 5y^{(3)} + 63\ddot{y} + 4\dot{y} + 2y = \mathrm{e}^{-3t} + \mathrm{e}^{-5t}\sin(4t + \pi/3).$$

   If the initial conditions are given by $y(0) = 1$, $\dot{y}(0) = \ddot{y}(0) = 1/2$, $y^{(3)}(0) = 0.2$, establish the Simulink model and plot the simulation results. The analytical solutions to linear differential equations can be evaluated with the `dsolve()` function. Try to find the analytical results with this function.

3. For the time varying differential equation

$$y^{(4)} + 5ty^{(3)} + 6t^2\ddot{y} + 4\dot{y} + 2\mathrm{e}^{-2t}y = \mathrm{e}^{-3t} + \mathrm{e}^{-5t}\sin(4t + \pi/3),$$

   assume that $y(0) = 1$, $\dot{y}(0) = \ddot{y}(0) = 1/2$, $y^{(3)}(0) = 0.2$. Draw the Simulink model to study the time varying system.

4. The Apollo trajectory (x, y) can be described by the equations

$$\ddot{x} = 2\dot{y} + x - \frac{\mu^*(x+\mu)}{r_1^3} - \frac{\mu(x-\mu^*)}{r_2^3}, \quad \ddot{y} = -2\dot{x} + y - \frac{\mu^* y}{r_1^3} - \frac{\mu y}{r_2^3},$$

   where $\mu = 1/82.45$, $\mu^* = 1-\mu$, $r_1 = \sqrt{(x+\mu)^2+y^2}$, $r_2 = \sqrt{(x-\mu^*)^2+y^2}$. Assume that $x(0) = 1.2$, $\dot{x}(0) = 0$, $y(0) = 0$, $\dot{y}(0) = -1.04935751$. Try to establish a Simulink model and draw the trajectory of Apollo.

5. For the well-known Van der Pol nonlinear differential equation described by $\ddot{y}+\mu(y^2-1)\dot{y} + y = 0$, draw the phase plane trajectory and study its limit cycles for different initial conditions.

6. For the famous chaotic Lorenz system

$$\begin{cases} \dot{x}_1(t) = -\beta x_1(t) + x_2(t)x_3(t), \\ \dot{x}_2(t) = -\rho x_2(t) + \rho x_3(t), \\ \dot{x}_3(t) = -x_1(t)x_2(t) + \sigma x_2(t) - x_3(t) \end{cases}$$

   with $\beta = 18/3$, $\sigma = \rho = 10$ and $x_1(0) = x_2(0) = 0$, $x_3(0) = 10^{-10}$, try to establish a Simulink model and draw the three-dimensional phase space trajectory of the results.

7. Consider the two input–two output system described by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 2.25 & -5 & -1.25 & -0.5 \\ 2.25 & -4.25 & -1.25 & -0.25 \\ 0.25 & -0.5 & -1.25 & -1 \\ 1.25 & -1.75 & -0.25 & -0.75 \end{bmatrix}\boldsymbol{x} + \begin{bmatrix} 4 & 6 \\ 2 & 4 \\ 2 & 2 \\ 0 & 2 \end{bmatrix}\boldsymbol{u}, \quad \boldsymbol{y} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \end{bmatrix}\boldsymbol{x}.$$

If the inputs are given by $\sin t$ and $\cos t$, construct the Simulink model and draw the simulation results.

8. For a 4×4 multivariable system given by

$$\boldsymbol{G}(s) = \begin{bmatrix} 1/(1+4s) & 0.7/(1+5s) & 0.3/(1+5s) & 0.2/(1+5s) \\ 0.6/(1+5s) & 1/(1+4s) & 0.4/(1+5s) & 0.35/(1+5s) \\ 0.35/(1+5s) & 0.4/(1+5s) & 1/(1+4s) & 0.6/(1+5s) \\ 0.2/(1+5s) & 0.3/(1+5s) & 0.7/(1+5s) & 1/(1+4s) \end{bmatrix},$$

construct its Simulink model and draw the simulation results when a unit step input signal is applied to each input channel individually. Compare the results with the one obtained with the `step()` function.

9. For a given implicit differential equation

$$\begin{cases} \sin x_1 \dot{x}_1 + \cos x_2 \dot{x}_2 + x_1 = 1, \\ -\cos x_2 \dot{x}_1 + \sin x_1 \dot{x}_2 + x_2 = 0, \end{cases}$$

if $x_1(0) = x_2(0) = 0$, numerically solve the differential equation using simulation method.

10. Establish a Simulink model for the block diagram of a nonlinear system shown in Figure 4.43. Observe the output signal for the unit step input.

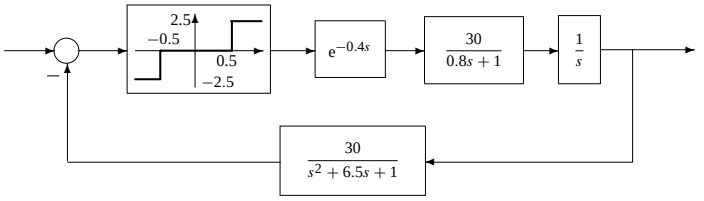


Figure 4.43. *Block diagram in Problem* 10.

11. Construct a Simulink model for the nonlinear block diagram shown in Figure 4.44. If the amplitude of the input step signal is 1.1, observe the output signal.

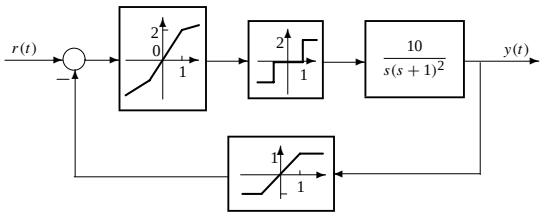


Figure 4.44. *Nonlinear system block diagram in Problem* 11.

12. If the Simulink model of a nonlinear system is given in Figure 4.45, write down the mathematical expression of the system from the Simulink model.

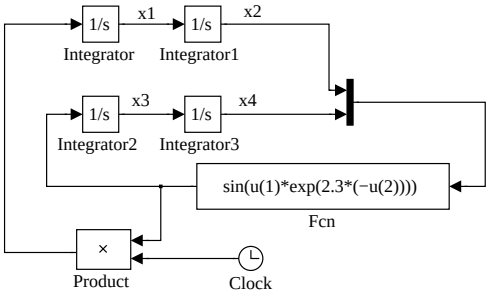


Figure 4.45. *Simulink model for Problem* 12.

13. Find the overall system model using MATLAB for the feedback systems shown in Figure 4.46. Try to use Simulink to get the closed-loop system model. Is there any problem in using Simulink, and if so, why? Obtain the overall system model by hand calculation or by direct call of MATLAB functions such as `series()` and `feedback()`.

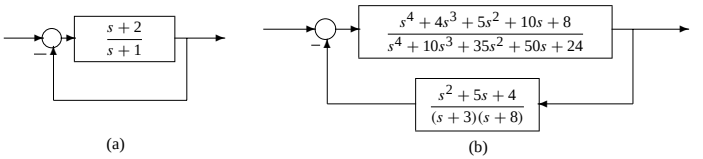


Figure 4.46. *The block diagrams for Problem* 13.

14. For a delayed differential equation

$$\mathrm{d}y(t)/\mathrm{d}t = \frac{0.2y(t-30)}{1+y^{10}(t-30)} - 0.1y(t),$$

if $y(0) = 0.1$, model the equations with Simulink, and simulate the system to draw the $y(t)$ curve.

15. For the neutral-type delayed differential equation

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}_1 \boldsymbol{x}(t-\tau_1) + \boldsymbol{A}_2 \dot{\boldsymbol{x}}(t-\tau_2) + \boldsymbol{B}u(t),$$

where $\tau_1 = 0.15$, $\tau_2 = 0.5$ and

$$\boldsymbol{A}_1 = \begin{bmatrix} -13 & 3 & -3 \\ 106 & -116 & 62 \\ 207 & -207 & 113 \end{bmatrix}, \quad \boldsymbol{A}_2 = \begin{bmatrix} 0.02 & 0 & 0 \\ 0 & 0.03 & 0 \\ 0 & 0 & 0.04 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix},$$

try to establish a Simulink model to find the solutions of the system.

16. Represent the block diagram shown in Figure 4.47 in Simulink and then perform a linearization to find the closed-loop transfer function and a state space model.

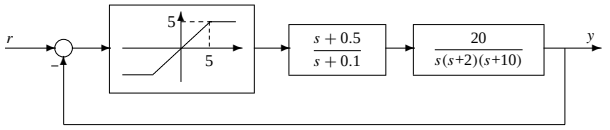


Figure 4.47. *The block diagram for Problem* 16.

17. Consider the well-known benchmark problem for testing a computer aided design environment (the F-14 airplane problem [54]). The linear model is shown in Figure 4.48. The parameters in the diagram are given by

$$\tau_a = 0.05,\ \sigma_{wG} = 3.0,\ a = 2.5348,\ b = 64.13,$$
$$V_{\tau_0} = 690.4,\ \sigma_\alpha = 5.236 \times 10^{-3},\ Z_b = -63.9979,\ M_b = -6.8847,$$
$$U_0 = 689.4,\ Z_w = -0.6385,\ M_q = -0.6571,\ M_w = -5.92 \times 10^{-3},$$
$$\omega_1 = 2.971,\ \omega_2 = 4.144,\ \tau_s = 0.10,\ \tau_\alpha = 0.3959,$$
$$K_Q = 0.8156,\ K_\alpha = 0.6770,\ K_f = -3.864,\ K_F = -1.745.$$

Select the input and output as $u = n(t)$ and $y(t) = N_{Z_p}(t)$ with

$$N_{Z_p}(t) = \frac{1}{32.2}[-\dot{w}(t) + U_0 q(t) + 22.8\dot{q}(t)].$$

Try to show the state space expression and find all the poles and zeros of the system.

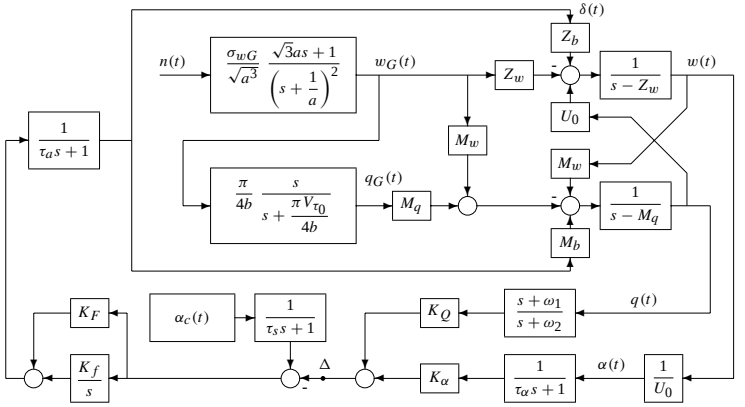
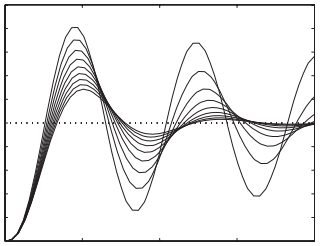


Figure 4.48. *The F-*14 *benchmark problem.*



Chapter 5

Model-Based Controller Design

For systems control, broadly speaking, there are three major steps, i.e., **m**odeling, **a**nalyis and **d**esign, also known as the "**mad**" process. If you are given a system to control, you probably have to go through this "mad" process, or loop, to achieve a satisfactory control performance.

For modeling, in Chapter 2 we discussed mathematical models of linear feedback control systems, where we focused more on various model forms and their conversions rather than on how to build a model from experimental results, which is a large subject area known as "system identification" only very lightly covered in Sec. 2.7.

For analysis, the available methods may typically be classified into time domain or frequency domain analysis. Equipped with the techniques from Chapter 3, given models of the plant and a controller, we can find in detail the time domain and frequency domain properties.

Now, we are ready to discuss the design of controllers. Given a plant model, how does one design or synthesize a controller so that the resulting system meets certain desired specifications? Since the controller design is dependent on the given model, we call the controller design "model-based."

In this chapter, frequency domain model-based controller design methods will be introduced for the cascade lead-lag compensator design (Sec. 5.1), followed by three time domain model-based controller design methods. They are the popular linear quadratic (LQ) optimal control method (Sec. 5.2), pole placement techniques (Sec. 5.3) and decoupling methods (Sec. 5.4). We will also discuss state observers and observer-based control. In Sec. 5.5, the SISOTool, an interactive controller design tool in the Control Systems Toolbox used mainly for single input–and single output (SISO) systems, is briefly demonstrated.

It should be noted that the "mad" process, in practice, may be iterative; that is, achieving a successful control system design may need several rounds of modeling, analysis, and design.

5.1 Cascade Lead-Lag Compensator Design

5.1.1 Introduction to Lead-Lag Synthesis

In the early days of control system design, controllers were usually implemented in analog form. Due to its simplicity, the phase lead-lag compensator was a popular form of controller since it can be easily implemented using a passive RC (resistor and capacitor) network or an RC network with an operational amplifier.

Basically, there are three commonly used compensators, namely, the phase lead compensator, phase lag compensator, and phase lead-lag compensator. Note that the compensator, or the controller, $G_c(s)$, is usually applied in cascade (series) connection to the plant model $G(s)$.

Phase lead compensator

The equivalent RC network to realize a phase lead compensator is shown in Figure 5.1(a). We denote the impedances by $Z_1 = R_1/(1 + R_1Cs)$ and $Z_2 = R_2$. The transfer function of the phase lead network can be written as

$$G_c(s) = \frac{U_o(s)}{U_i(s)} = \frac{Z_2}{Z_1 + Z_2} = \frac{1}{\alpha}\frac{1+\alpha T s}{1+T s}, \tag{5.1}$$

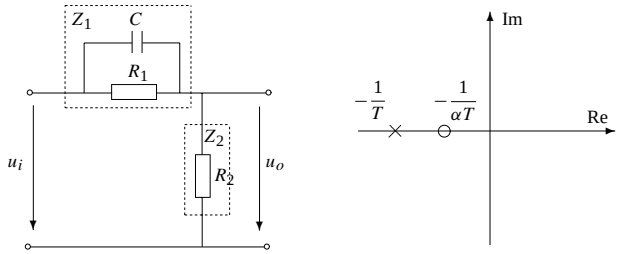
where

$$T = \frac{R_1R_2}{R_1 + R_2}C, \alpha = \frac{R_1 + R_2}{R_2}. \tag{5.2}$$

Obviously, $\alpha > 1$. In general, the phase lead compensator can be written as

$$G_c(s) = K_c\frac{1+\alpha T s}{1+T s}. \tag{5.3}$$

The pole-zero location of the compensator is sketched in Figure 5.1(b). Since $\alpha > 1$, the pole is always located on the left-hand side of the zero. For some different α's, the Bode and Nyquist diagrams of the lead compensator with $T = 1$ are shown in Figures 5.2(a) and (b), respectively, through the following MATLAB statements:



(a) lead network (b) pole-zero positions

Figure 5.1. *Lead compensator.*


```
>> f1=figure; f2=figure; T=1;
   for alpha0=1.5:0.5:5
      G1=tf([alpha0*T,1]/alpha0,[T,1]);
      figure(f1), nyquist(G1);hold on;figure(f2), bode(G1); hold on
   end
```

It can be observed that when α is large, the gain compensation is small but the phase compensation is large.

Example 5.1. Consider a plant model given by

$$G(s) = \frac{100}{s(0.04s + 1)}.$$

The behavior of a feedback system with a lead compensator for the above plant model is illustrated in the frequency domain through this example.

The gain and phase margins of the system with the plant alone in the loop are obtained using the following MATLAB statements:

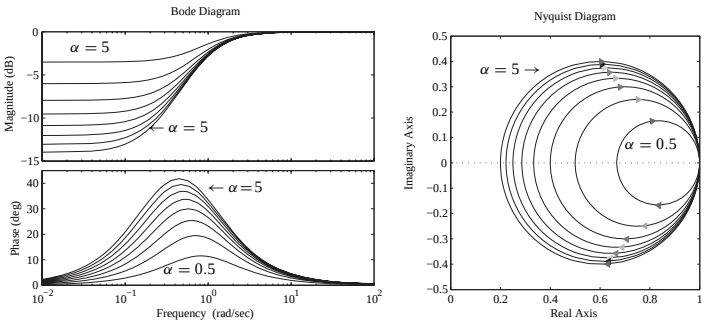
```
>> G=tf(100,[0.04,1,0]); [Gm,Pm,Wcg,Wcp]=margin(G), bode(G)
```

It is found that the phase margin is 28.0243° at a frequency of 46.9701 rad/sec, with an infinite gain margin. The Bode diagram of the open-loop model is shown in Figure 5.3(a), where the phase margin is marked.

The phase margin can be increased by introducing a phase lead compensator given by $G_c(s) = (0.0262s + 1)/(0.0106s + 1)$. The Bode diagram of the compensator is shown in Figure 5.3(b). In this case, the gain and phase margins of the compensated system can be obtained using the following MATLAB statements:

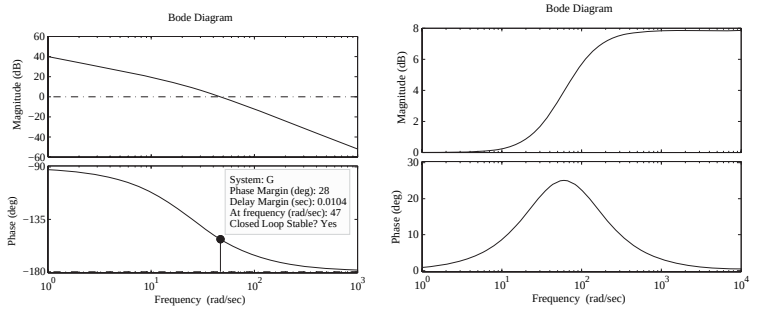
```
>> Gc1=tf([0.0262 1],[0.0106,1]);  bode(Gc1)
   [Gm,Pm,Wcg,Wcp]=margin(G*Gc1)
```

It is found that the phase margin of the compensated system is 47.6° at a frequency of 60.3 rad/sec, again with an infinite gain margin. The magnitude and phase crossover frequencies



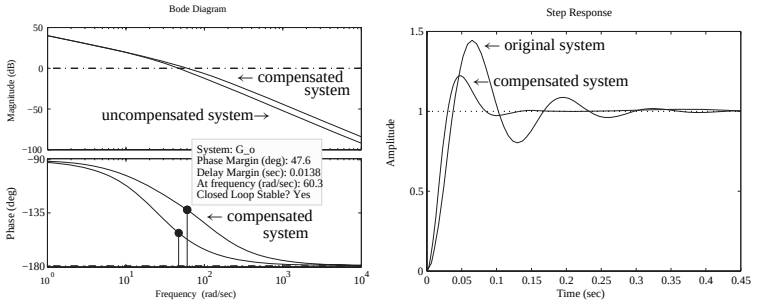
(a) Bode diagrams (b) Nyquist plots

Figure 5.2. *Frequency domain representation of a lead compensator.*



(a) Plant Bode diagrams (b) Compensator Bode diagrams

Figure 5.3. *Bode diagrams of the plant and the lead compensator.*



(a) Bode diagrams comparison (b) step response comparison

Figure 5.4. *Comparison of system responses.*

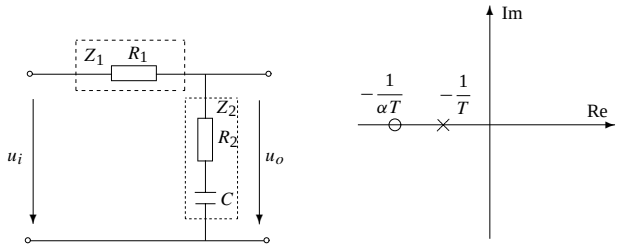
in the compensated system, as expected, are both increased. The open-loop Bode diagrams of the compensated system and the original system are compared in Figure 5.4(a) using the following MATLAB statements:

```
>> G_o=Gc1*G; bode(G,G_o); figure
   G_c1=feedback(G,1); G_c2=feedback(G_o,1); step(G_c1,G_c2)
```

The closed-loop step responses of the systems before and after phase lead compensation are compared in Figure 5.4(b). The step response of the compensated system is significantly improved, since the overshoot is reduced due to the increased phase margin, and the speed of response is also increased, due to the increased crossover frequency.

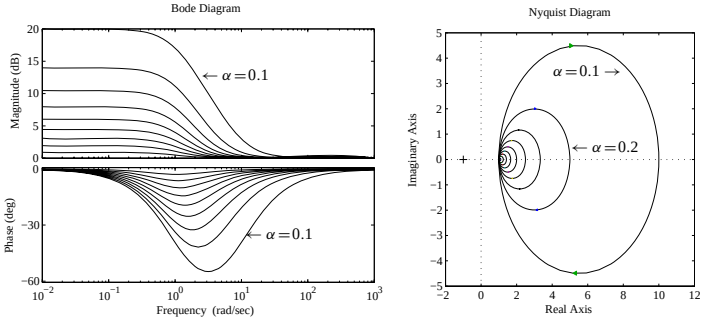
Phase lag compensator

The equivalent RC network for a phase lag compensator is shown in Figure 5.5(a), with the pole-zero positions sketched in Figure 5.5(b). Let $Z_1 = R_1$ and $Z_2 = R_2 + 1/(Cs)$. The



(a) lag network (b) pole-zero positions

Figure 5.5. *Lag compensator.*



(a) Bode diagrams (b) Nyquist plots

Figure 5.6. *Frequency responses of lag compensators.*

transfer function of the phase lag network can be written as

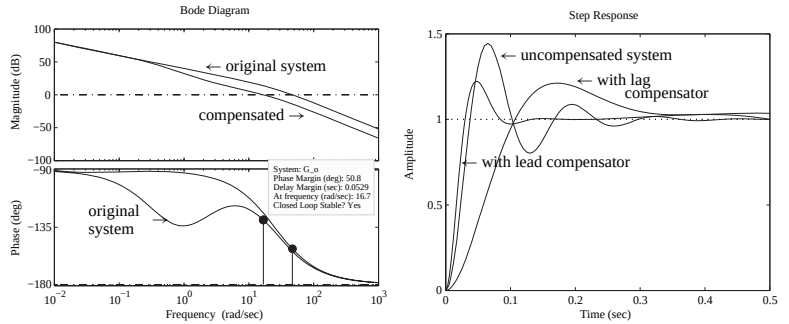
$$G_c(s) = \frac{U_o(s)}{U_i(s)} = \frac{Z_2}{Z_1 + Z_2} = \frac{1 + \alpha T s}{1 + T s}, \tag{5.4}$$

where $R_2C = \alpha T$, $\alpha = R_2/(R_1 + R_2) < 1$. In a more general form, the phase lag compensator can be written as

$$G_c(s) = K_c \frac{1 + \alpha T s}{1 + T s}. \tag{5.5}$$

The Bode diagrams and the Nyquist plots for $K_c = 1$ and $T = 1$ are shown in Figures 5.6(a) and (b), respectively, for different values of α. These diagrams are obtained using the following MATLAB statements:

```
>> f1=figure; f2=figure; T=1;
   for alpha0=0.9:-0.1:0.1
      G1=tf([alpha0*T,1]/alpha0,[T,1]);
      figure(f1), nyquist(G1), hold on;figure(f2), bode(G1), hold on
   end
```



(a) Bode diagrams comparison (b) step response comparison

Figure 5.7. *Comparison of system responses.*

Example 5.2. Consider again the plant model in Example 5.1. If a phase lag compensator $G_c(s) = (0.5s+1)/(2.5s+1)$ is now used, the gain and phase margins of the compensated system can be obtained using the following MATLAB statements:

```
>> Gc2=tf([0.5 1],[2.5,1]); G=tf(100,[0.04,1,0]); G_o=Gc2*G;
   [Gm,Pm,Wcg,Wcp]=margin(G_o); bode(G_o,G)
   figure;step(feedback(G,1),feedback(G_o,1),feedback(Gc1*G,1),0.5)
```

The phase margin is 50.7572° at a frequency of 16.7339 rad/sec, with an infinite gain margin. The Bode diagram of the compensated system can be obtained as shown in Figure 5.7(a).

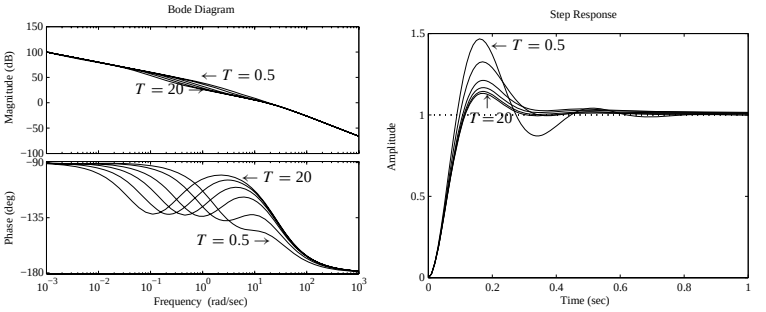
The basic idea of a lag compensator is to decrease the crossover frequency so as to increase the phase margin of the system. However, since this technique reduces the open-loop bandwidth, it also reduces the response speed of the system. However, it does have the advantage, unlike lead compensation, that a solution can always normally be found.

The step responses of the phase lag compensated system, the original system, and the phase lead compensated system, are all shown in Figure 5.7(b). As with the lead compensator, the increased phase margin given by the lag compensator has reduced the overshoot in the step response.

Now, let us fix $\alpha = 0.2$ and change T, i.e., the lag compensator is $G_c(s) = (1 + 0.2Ts)/(1 + Ts)$. Let us see how T affects the performance of the compensated system. Using the MATLAB statements

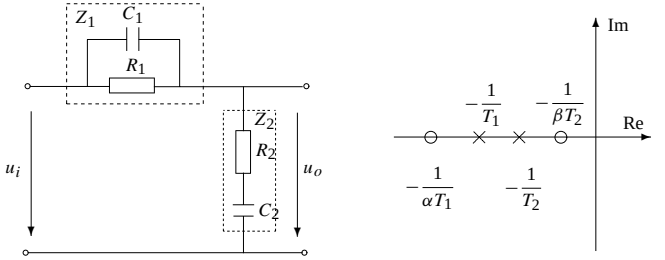
```
>> G=tf(100,[0.04,1,0]); f1=figure; f2=figure;
   for T=[0.5,1,2.5,5,10,20]
      Gc2=tf([0.2*T 1],[T,1]); G_o=G*Gc2; G_c=feedback(G_o,1);
      figure(f1),bode(G_o), hold on;figure(f2),step(G_c,1), hold on
   end
```

the superimposed Bode diagrams and step response are shown in Figures 5.8(a) and (b), respectively. Among the phase lag compensators, it can be seen that the larger the value of T, the better the performance of the compensated system.



(a) Bode diagrams comparison (b) step response comparison

Figure 5.8. *The effect of changing T.*



(a) lead-lag network (b) pole-zero positions

Figure 5.9. *Lead-lag compensator.*

Phase lead-lag compensator

For a phase lead-lag compensator, its equivalent RC network is shown in Figure 5.9(a), and its pole-zero map is shown in Figure 5.9(b). Denote $Z_1 = R_1/(1 + R_1C_1s)$ and $Z_2 = R_2 + 1/(C_2s)$. The transfer function of the phase lead-lag compensator can be written as

$$G_c(s) = \frac{U_o(s)}{U_i(s)} = \frac{Z_2}{Z_1 + Z_2} = \frac{(1+\alpha T_1 s)(1+\beta T_2 s)}{(1+T_1 s)(1+T_2 s)}, \tag{5.6}$$

where $\alpha T_1 = R_1C_1$, $\beta T_2 = R_2C_2$, $\alpha\beta = 1$, and clearly, $R_1C_1 + R_2C_2 + R_1C_2 = T_1 + T_2$, $T_1T_2 = R_1C_1R_2C_2$. When $\alpha > 1$ and $\beta < 1$, the first term in (5.6) exhibits the phase lead property, while the second term has the phase lag characteristics.

As shown in Figure 5.10, with $T_1 = 0.5$, $T_2 = 0.005$, and $\alpha = 3$, $\beta = 1/3$, and $T_2 = 0.5$, $T_1 = 0.005$, the Bode diagrams of the lead-lag compensator and the lag-lead compensators are obtained using the following MATLAB statements, respectively:

```
>> T1=0.5; T2=0.005; alpha=3; beta=1/3; s=zpk('s');
   G1=(alpha*T1*s+1)*(beta*T2*s+1)/(T1*s+1)/(T2*s+1);
   T2=0.5; T1=0.005; alpha=3; beta=1/3;
   G2=(alpha*T1*s+1)*(beta*T2*s+1)/(T1*s+1)/(T2*s+1); bode(G1,G2)
```

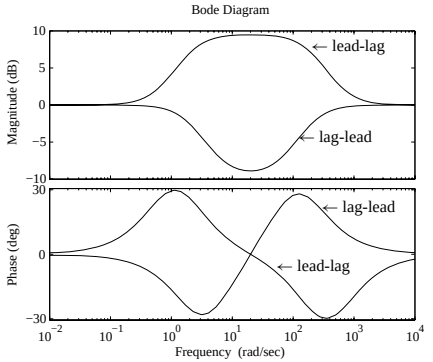


Figure 5.10. *The Bode diagrams of the lead-lag and lag-lead compensators.*

It can be observed that in the phase lead-lag compensator, the phase is positive (lead) before it becomes negative (lag). However, the phase lag action is taken first, followed by the lead action for lag-lead compensator.

In practical applications, the lead-lag compensator is usually used. In what follows, we will show a method for designing a phase lead-lag compensator.

5.1.2 Lead-Lag Synthesis by Phase Margin Assignment

The transfer function of a lead-lag compensator can be written as

$$G_c(s) = \frac{K_c(s+z_{c_1})(s+z_{c_2})}{(s+p_{c_1})(s+p_{c_2})}, \quad (z_{c_1} \le p_{c_1},\ z_{c_2} \ge p_{c_2}). \tag{5.7}$$

Denote by K_p the desired static position error constant which is defined as

$$K_p = \lim_{s\to 0} G_c(s)G(s)$$

with $G(s)$ the plant model.

If the phase angle of the plant model is $\phi_1(\omega_c)$ at the expected crossover frequency $\omega = \omega_c$, the phase angle of the compensator at ω_c should be $\phi_c(\omega_c) = \gamma - 180^\circ - \phi_1(\omega_c)$, where γ is the expected phase margin of the compensated system. The magnitude of the plant model at ω_c is denoted by $A(\omega_c)$. The following synthesis procedure can be used.

Case 1. When $\phi_c(\omega_c) > 0$, a lead compensation is required which can be designed as

$$\alpha = \frac{z_{c_1}}{p_{c_1}} = \frac{1-\sin\phi_c}{1+\sin\phi_c} \tag{5.8}$$

and

$$z_{c_1} = \sqrt{\alpha}\omega_c, \quad p_{c_1} = \frac{z_{c_1}}{p_{c_1}} = \frac{\omega_c}{\sqrt{\alpha}}, \quad K_c = \frac{\sqrt{\omega_c^2 + p_{c_1}^2}}{\sqrt{\omega_c^2 + z_{c_1}^2}A(\omega_c)}. \tag{5.9}$$

The static position error constant of the system can be obtained as

$$K_1 = \lim_{s\to 0} s^v G_o(s) = \frac{b_m}{a_{n-v}}\frac{K_c z_{c_1}}{p_{c_1}}, \tag{5.10}$$

where v is the multiplicity of the pole $s = 0$ of the plant model $G(s)$ with

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

and $G_o(s)$ is the open-loop transfer function with the compensator, i.e., $G_o(s) = G_c(s)G(s)$.

If $K_1 \ge K_p$, the designed phase lead compensator is adequate according to the phase margin assignment. Otherwise, a phase lead-lag compensation is required.

Case 2. For phase lead-lag compensation, it is required to further specify

$$z_{c_2} = \frac{\omega_c}{10}, \quad p_{c_2} = \frac{K_1 z_{c_2}}{K_p}. \tag{5.11}$$

Case 3. If $\phi_c(\omega_c) < 0$, the phase lag compensation is expected, and

$$K_c = \frac{1}{A(\omega_c)}, \quad z_{c_2} = \omega_c/10, \quad p_{c_2} = K_1 z_{c_2}/K_p, \tag{5.12}$$

where $K_1 = b_m K_c/a_{n-v}$.

A MATLAB function `leadlagc()` has been written to implement the three cases in the above algorithm:

```
function Gc=leadlagc(G,Wc,Gam_c,Kv,key)
G=tf(G); [Gai,Pha]=bode(G,Wc);
Phi_c=sin((Gam_c-Pha-180)*pi/180);
den=G.den{1}; a=den(length(den):-1:1);
ii=find(abs(a)<=0); num=G.num{1}; G_n=num(end);
if length(ii)>0, a=a(ii(1)+1); else, a=a(1); end;
alpha=sqrt((1-Phi_c)/(1+Phi_c)); Zc=alpha*Wc; Pc=Wc/alpha;
Kc=sqrt((Wc*Wc+Pc*Pc)/(Wc*Wc+Zc*Zc))/Gai; K1=G_n*Kc*alpha/a;
if nargin==4, key=1;
   if Phi_c<0, key=2; else, if K1<Kv, key=3; end, end
end
switch key
case 1, Gc=tf([1 Zc]*Kc,[1 Pc]);
case 2, Kc=1/Gai; K1=G_n*Kc/a; Gc=tf([1 0.1*Wc],[1 0.1*K1*Wc/Kv]);
case 3
   Zc2=Wc*0.1; Pc2=K1*Zc2/Kv; Gcn=Kc*conv([1 Zc],[1,Zc2]);
   Gcd=conv([1 Pc],[1,Pc2]); Gc=tf(Gcn,Gcd);
end
```

The syntax of the function is `Gc=leadlagc(G,ωc,γ,Kp,key)`, where G is the LTI (linear time-invariant) object of the plant model, ω_c is the expected crossover frequency, γ is the expected phase margin of the compensated system, and K_p is the static position error constant. If `key` is provided, the controller will be designed according to the type specified in `key` with `key` $= 1, 3, 2$ for Cases 1, 2, 3, respectively, discussed in the above. If `key` is not provided, the controller structure will be selected automatically. The returned G_c is the transfer function object of the compensator.

Example 5.3. Consider the plant model in Example 5.1. Different phase margins are assigned with the crossover frequency at $\omega_c = 100$ rad/sec. The corresponding compensators can be designed using the following MATLAB statements:

```
>> G=tf(100,[0.04,1,0]); wc=100; f1=figure; f2=figure;
   for gamma=[30,40,50,60,70,80,90]
      Gc=leadlagc(G,wc,gamma,10); G_o=Gc*G; G_c=feedback(G_o,1);
      figure(f1), bode(G_o), hold on;
      figure(f2), step(G_c,0.1), hold on
   end
```

The Bode diagrams of the systems under some different γ's can be obtained as shown in Figure 5.11(a), and it can be seen that the phase margins of the compensated systems are consistent with the predefined values. The closed-loop step responses of the systems are shown in Figure 5.11(b), where it can be seen that when the specified phase margin increases, the step response is improved in terms of a smaller overshoot. However, for this example, if the phase margin is specified too large, for instance, $80^\circ \sim 90^\circ$, the system responses are not satisfactory. Because the integrator gives a 90° phase lag, the bandwidth has to be reduced considerably to obtain the high phase margin, and the result is a slow response. This illustrates that the program needs to be used with a good physical understanding and will not give results for poor assumptions. In this example, a good compensator can be designed, when setting $\gamma = 70^\circ$, as

$$G_c(s) = 13.4708\frac{s + 30.61}{s + 326.7}.$$

Now, let us fix the specified phase margin to $\gamma = 70^\circ$ and change the values of the crossover frequencies ω_c. The Bode diagrams and closed-loop step responses of the compensated systems are compared in Figures 5.12(a) and (b), respectively, by using the following MATLAB statements:

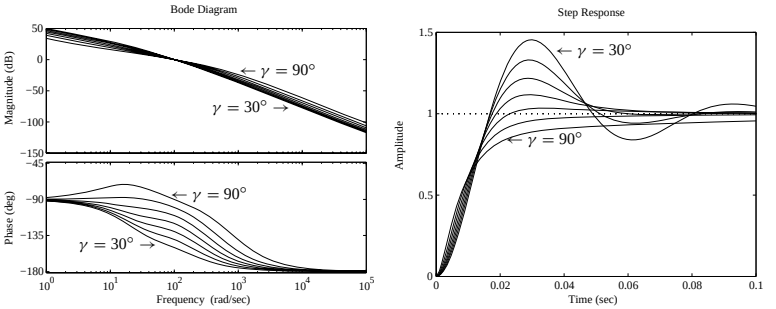
```
>> G=tf(100,[0.04,1,0]); gamma=70; f1=figure; f2=figure;
   for wc=[50,100,200,300,500]
      Gc=leadlagc(G,wc,gamma,10); G_o=Gc*G; G_c=feedback(G_o,1);
      figure(f1), bode(G_o), hold on;
      figure(f2), step(G_c,0.1), hold on
   end
```

Clearly, the overshoots under different ω_c's are almost the same. However, when ω_c increases, the step response becomes faster.

Example 5.4. Consider the transfer function of the plant model given by

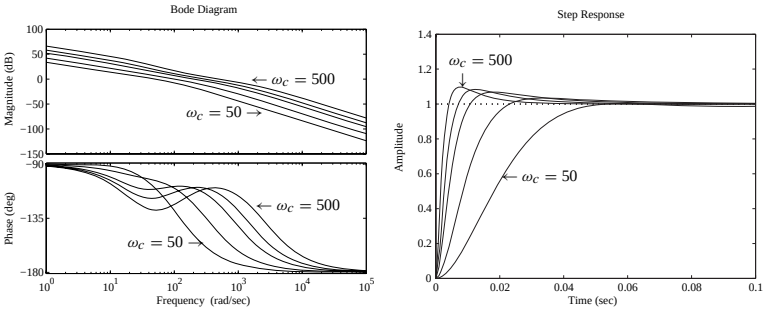
$$G(s) = \frac{100}{s(s + 1)(0.0125s + 1)}.$$

Set the crossover frequency at $\omega_c = 50$ rad/sec and assign the expected phase margin of the compensated system to $\gamma = 50^\circ$. The compensator G_c can be designed by calling in the `leadlagc()` function. The Nichols charts of the systems before and after compensation



(a) Bode diagrams comparison (b) step response comparison

Figure 5.11. *The effect of the desired phase margin γ.*



(a) Bode diagrams comparison (b) step response comparison

Figure 5.12. *The effect of the desired crossover frequency ω_c.*

are compared in Figure 5.13(a) using the following MATLAB statements:

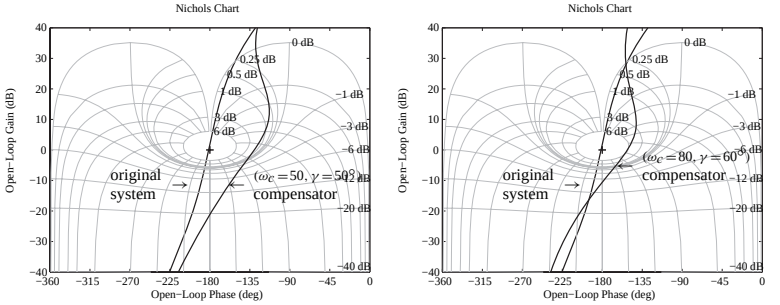
```
>> s=zpk('s'); G=100/(s*(s+1)*(0.0125*s+1));
   Gc=leadlagc(G,50,50,100); zpk(Gc), G1=Gc*G;
   nichols(G,G1); grid; axis([-360,0,-40,40])
```

The controller designed is

$$G_c(s) = \frac{368.8908(s + 3.997)}{s + 625.5}.$$

From the Nichols chart, the closed-loop uncompensated system is unstable. By introducing the lead compensator, at high frequencies, the Nichols chart is kept away from the $M = 1$dB contour, which not only makes the closed-loop compensated system stable, but also ensures a rather good time domain response of the compensated system.

To get an even larger phase margin by using the lead compensator, for instance, $\gamma = 60^\circ$ at $\omega_c = 80$ rad/sec, a new design result can be obtained by using the following



(a) expected $\omega_c = 50, \gamma = 50°$ (b) expected $\omega_c = 50, \gamma = 100°$

Figure 5.13. *Nichols charts for different controllers.*

MATLAB statements:

```
>> Gc=leadlagc(G,80,60,100); zpk(Gc)
   G2=G*Gc; nichols(G,G2), grid; axis([-360,0,-40,40])
   [G1,P1,w1,w2]=margin(G2); [G1,P1,w1,w2]
```

The controller is now

$$G_c(s) = \frac{722.4022(s + 10.02)}{s + 638.5}.$$

It is found that the gain and phase margins are 5.5430, 31.4323° at frequencies 211.1766, and 80 rad/sec, respectively. The Nichols charts of the systems are shown in Figure 5.13(b). The actual phase margin under this newly designed controller is $\gamma = 28.28°$, which is, surprisingly, well below the expected $\gamma = 90°$. It can be seen that although the system is stabilized, the closed-loop behavior of the compensated system may not be very good, due to the poor Nichols charts. Therefore, the prespecified phase margin of 90° cannot be met and the requirements are too demanding for the implementation under a lead-lag compensation.

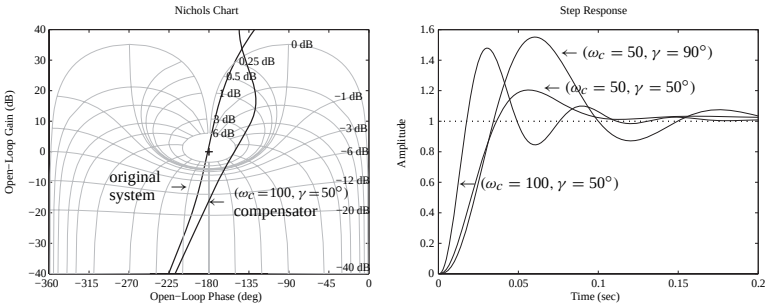
Similarly, if one wishes to achieve a phase margin of $\gamma = 50°$ and, at the same time, to have a crossover frequency at $\omega_c = 100$ rad/sec, the following MATLAB statements can be used:

```
>> Gc=leadlagc(G,100,50,100); zpk(Gc)
   G3=G*Gc; nichols(G,G3), grid; axis([-360,0,-40,40])
   [G1,P1,w1,w2]=margin(G3); [G1,P1,w1,w2]
```

The controller is designed as

$$DG_c(s) = \frac{1698.7153(s + 9.424)}{s + 1061},$$

and the actual gain and phase margins are 6.3204, 28.4655° at frequencies 274.3693 and 100 rad/sec, respectively. The Nichols charts are shown in Figure 5.14(a). Once more, it can be seen that the prespecified properties of the desired system cannot be achieved. Therefore, the



(a) expected $\omega_c = 100, \gamma = 50°$ (b) step response comparison

Figure 5.14. *System responses comparisons.*

controller design objective is overspecified. The closed-loop step responses of the systems under the three compensators designed in the above are compared in Figure 5.14(b), with the following MATLAB statements:

```
>> step(feedback(G1,1),feedback(G2,1),feedback(G3,1),0.3);
```

It can be seen that the closed-loop response comparison agrees with the analysis performed earlier from the Nichols charts.

In fact, even if the expected crossover frequency ω_c and the phase margin γ are both assigned, the compensator designed using the approach given above may not guarantee a compensated system satisfying all the specifications. Moreover, the closed-loop system may not be even stable with the designed compensator. The closed-loop behavior of the system should be examined before the controller can be used in practice.

Other lead-lag compensator design approaches, such as the root locus method, will not be discussed in this book.

5.2 Linear Quadratic Optimal Control

5.2.1 Linear Quadratic Optimal Control Strategies

Consider an LTI system given by its state space model

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t), \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t). \end{cases} \tag{5.13}$$

Introduce the following performance index for the optimal controller design:

$$J = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}(t_f)\boldsymbol{S}\boldsymbol{x}(t_f) + \frac{1}{2}\int_{t_0}^{t_f}\Big[\boldsymbol{x}^{\mathrm{T}}(t)\,\boldsymbol{Q}(t)\boldsymbol{x}(t) + \boldsymbol{u}^{\mathrm{T}}(t)\boldsymbol{R}(t)\boldsymbol{u}(t)\Big]\mathrm{d}t, \tag{5.14}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are weighting matrices for the state variables and the input variables, respectively, and t_f is the terminal time for control action, which means that the control

action is in a finite time interval. $\boldsymbol{S} \geq 0$ is the weighting matrix for the terminal states. This optimal control problem is referred to as the linear quadratic (LQ) optimal control problem.

To solve this LQ optimal control problem, let us first construct a Hamiltonian function

$$\boldsymbol{H} = -\frac{1}{2}\Big[\boldsymbol{x}^{\mathbf{T}}(t)\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}^{\mathbf{T}}(t)\boldsymbol{R}\boldsymbol{u}(t)\Big] + \boldsymbol{\lambda}^{\mathbf{T}}(t)\Big[\boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)\Big]. \tag{5.15}$$

When there is no constraint on the input signal, the optimal (in this case, the minimum) value can be solved by taking the derivative of $\boldsymbol{H}$ with respect to $\boldsymbol{u}$ and then solving the following equation:

$$\frac{\partial \boldsymbol{H}}{\partial \boldsymbol{u}} = -\boldsymbol{R}\boldsymbol{u}(t) + \boldsymbol{B}^{\mathbf{T}}\boldsymbol{\lambda}(t) = 0. \tag{5.16}$$

Denote by $\boldsymbol{u}^*(t)$ the optimal control signal $\boldsymbol{u}(t)$. Then, $\boldsymbol{u}^*(t)$ can be explicitly written in the following form:

$$\boldsymbol{u}^*(t) = \boldsymbol{R}^{-1}\boldsymbol{B}^{\mathbf{T}}\boldsymbol{\lambda}(t). \tag{5.17}$$

On the other hand, it can be shown that the Lagrangian multiplier $\boldsymbol{\lambda}(t)$ can be written as $\boldsymbol{\lambda}(t) = -\boldsymbol{P}(t)\boldsymbol{x}(t)$, where $\boldsymbol{P}(t)$ is the symmetrical solution matrix of the well-known differential Riccati equation (DRE)

$$\dot{\boldsymbol{P}}(t) = -\boldsymbol{P}(t)\boldsymbol{A} - \boldsymbol{A}^{\mathbf{T}}\boldsymbol{P}(t) + \boldsymbol{P}(t)\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathbf{T}}\boldsymbol{P}(t) - \boldsymbol{Q} \tag{5.18}$$

with its final value $\boldsymbol{P}(t_f) = \boldsymbol{S}$. So, the optimal control signal can also be written as

$$\boldsymbol{u}^*(t) = -\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathbf{T}}\boldsymbol{P}(t)\boldsymbol{x}(t). \tag{5.19}$$

It is interesting to note that the solution of the finite time LQ optimal control problem turns out to be a linear state feedback with a time varying gain matrix, which is equal to $-\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathbf{T}}\boldsymbol{P}(t)$.

5.2.2 Linear Quadratic Regulator Problems

When t_f is finite, solving the LQ optimal control problem amounts to solving the DRE (5.18), which is very difficult to solve. In many applications, one is more concerned with the regulation performance, which implies that $t_f \to \infty$, as in many process control systems. If we consider this steady-state performance, the LQ optimal control problem is referred to as an LQR (linear quadratic regulator) problem.

In the LQR problem, $t_f = \infty$ and the closed-loop system will be asymptotically stabilized. The solution matrix $\boldsymbol{P}(t)$ to the DRE will tend to a constant matrix, i.e., $\dot{\boldsymbol{P}}(t) = 0$. In this case, the DRE reduces to the so-called algebraic Riccati equation (ARE) as follows:

$$\boldsymbol{P}\boldsymbol{A} + \boldsymbol{A}^{\mathbf{T}}\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathbf{T}}\boldsymbol{P} + \boldsymbol{Q} = \boldsymbol{0}. \tag{5.20}$$

The above ARE can be easily solved by the MATLAB function `are()` in the Control Systems Toolbox, where `P=are(A',B*inv(R)*B',Q)`. Then, the LQR problem can be solved using a linear state feedback with a constant gain matrix, i.e.,

$$\boldsymbol{u}(t) = -\boldsymbol{K}\boldsymbol{x}(t), \quad \boldsymbol{K} = -\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathbf{T}}\boldsymbol{P}.$$

Clearly, the closed-loop system is simply $[(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}), \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}]$.

A MATLAB function `lqr()` provided in the Control Systems Toolbox can be used to design an LQR for a given system with given weighting matrices. The syntax of the function is `[K,P]=lqr(A,B,Q,R)`, where $(\boldsymbol{A},\boldsymbol{B})$ is the given state space model, and $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the weighting matrices. $\boldsymbol{K}$ is the state feedback gain matrix, and $\boldsymbol{P}$ is the solution matrix for the ARE.

Example 5.5. Consider the following plant in state space form:

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} -0.2 & 0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 1.6 & 0 & 0 \\ 0 & 0 & -14.3 & 85.8 & 0 \\ 0 & 0 & 0 & -33.3 & 100 \\ 0 & 0 & 0 & 0 & -10 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 30 \end{bmatrix} u(t), \quad y = [1,0,0,0,0]\boldsymbol{x}.$$

Select the weighting matrices as $\boldsymbol{Q} = \text{diag}\{\rho, 0, 0, 0, 0\}$ and $R = 1$. When $\rho = 1$, the LQR problem can be easily solved using the following MATLAB statements:

```
>> A=[-0.2,0.5,0,0,0;0,-0.5,1.6,0,0;0,0,-14.3,85.8,0;
       0,0,0,-33.3,100;0,0,0,0,-10];
   B=[0; 0; 0; 0; 30]; Q=diag([1,0,0,0,0]); R=1;
   C=[1,0,0,0,0]; D=0; [K,P]=lqr(A,B,Q,R)
```

and it can be found that

$$\boldsymbol{K}^{\mathbf{T}} = \begin{bmatrix} 0.926 \\ 0.1678 \\ 0.0157 \\ 0.0371 \\ 0.2653 \end{bmatrix}, \quad \boldsymbol{P} = \begin{bmatrix} 0.3563 & 0.0326 & 0.0026 & 0.0056 & 0.0309 \\ 0.0326 & 0.0044 & 0.00039 & 0.00088 & 0.0056 \\ 0.00259 & 0.00039 & 3.5\times10^{-5} & 7.95\times10^{-5} & 0.00052 \\ 0.0056 & 0.00088 & 7.95\times10^{-5} & 0.00018 & 0.0012 \\ 0.0309 & 0.0056 & 0.00052 & 0.0012 & 0.0088 \end{bmatrix}.$$

The closed-loop state matrix $\boldsymbol{A}_c$ under the LQR can be obtained with the following MATLAB statements:

```
>> Ac=A-B*K; step(ss(Ac,B,C,D))
```

The step response of the closed-loop system is shown in Figure 5.15(a).

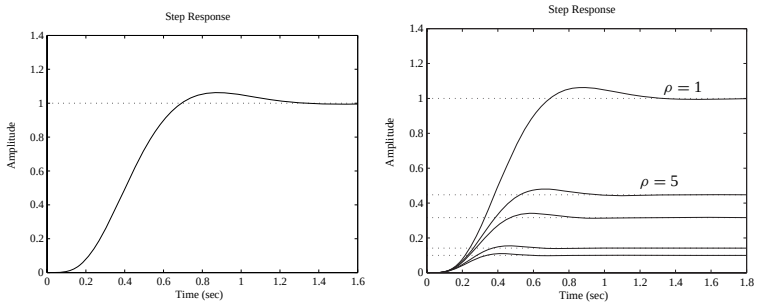
For different values of ρ, for example, $\rho = 5, 10, 50, 100$, the step responses of closed-loop systems under the optimal LQR can be obtained using the following MATLAB statements:

```
>> for rho=[1,5,10,50,100]
     Q(1,1)=rho;[K,P]=lqr(A,B,Q,R); step(ss(A-B*K,B,C,D)); hold on
   end
```

The results are compared in Figure 5.15(b). Clearly, when ρ increases, the magnitude of $y(t) = x_1(t)$ becomes smaller since the penalty on $x_1(t)$ is heavier.

To see the step responses of the other states under the LQR, let us fix $\rho = 100$. The step response of $x_2(t)$ to $x_5(t)$, obtained using the MATLAB statements

```
>> Q(1,1)=100; [K,P]=lqr(A,B,Q,R); Ac=A-B*K;
   [y,t,x]=step(ss(Ac,B,C,D)); plot(t,x(:,2:5))
```



(a) LQR design with $\rho = 1$ (b) LQR with different ρ

Figure 5.15. *Step responses of closed-loop systems.*

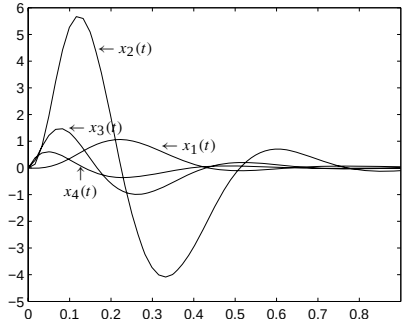


Figure 5.16. *Step responses of the states of closed-loop systems.*

are shown in Figure 5.16, where it can be observed that since there is no penalty on other state variables due to the special structure of the $\boldsymbol{Q}$ matrix, all the other state variables except x_1 may become very large.

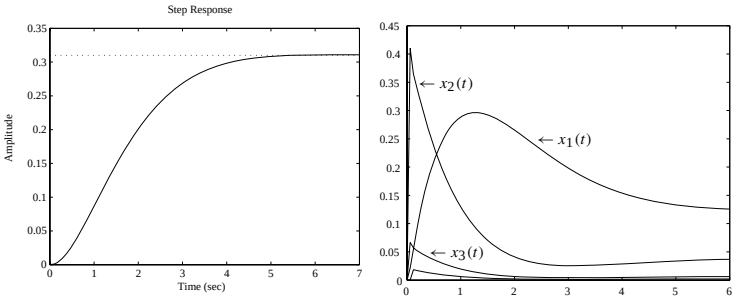
Now, let us put some penalties on the other states by redefining the weighting matrix $\boldsymbol{Q}$ such that $\boldsymbol{Q} = \text{diag}(10, 2, 6, 2, 1)$. By the following MATLAB statements:

```
>> Q=diag([10,2,6,2,1]); [K,P]=lqr(A,B,Q,R); Ac=A-B*K;
   step(ss(Ac,B,C,D)), figure, [y,t,x]=step(ss(Ac,B,C,D));
   plot(t,x(:,2:5))
```

the step response of the system under the newly designed LQR is obtained as shown in Figure 5.17(a), where it can be seen that the output response speed is significantly reduced. However, from the step responses of the other states, obtained using the MATLAB statements

```
>> [y,t,x]=step(ss(Ac,B,C,D)); plot(t,x(:,2:5))
```

as shown in Figure 5.17(b), the amplitudes of all the other states except x_1 are significantly reduced due to the new weighting matrix $\boldsymbol{Q}$.



(a) with a new $\boldsymbol{Q}$ (b) other states

Figure 5.17. *Step responses of closed-loop systems.*

Example 5.6. Consider a multivariable state space equation given by

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} -8 & -6 & 0 & -2 \\ -6 & -14 & -1 & -6 \\ 0 & -1 & -26 & -2 \\ -2 & -6 & -2 & -20 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ -1 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{u}(t).$$

Selecting a diagonal matrix $\boldsymbol{Q} = \text{diag}(10, 8, 2, 0)$ and an identity matrix $\boldsymbol{R} = \boldsymbol{I}$, we can obtain the state feedback matrix with the following statements

```
>> A=[-8,-6,0,-2; -6,-14,-1,-6; 0,-1,-26,-2; -2,-6,-2,-20];
   B=[1,-1; 0,1; -1,1; 0,0]; C=[1 0 0 0; 0 0 1 0];
   Q=diag([10,8,2,0]); R=eye(2); [K,P]=lqr(A,B,Q,R), eig(A-B*K)
```

The state feedback matrix $\boldsymbol{K}$ and the Riccati equation solution $\boldsymbol{P}$ can be obtained with the previous statements

$$\boldsymbol{K}^{\mathbf{T}} = \begin{bmatrix} 0.7208 & -0.9919 \\ -0.2609 & 0.6705 \\ -0.0287 & 0.0185 \\ 0.0140 & -0.0735 \end{bmatrix}, \quad \boldsymbol{P} = \begin{bmatrix} 0.7309 & -0.2711 & 0.0101 & 0.0138 \\ -0.2711 & 0.4096 & -0.0102 & -0.0595 \\ 0.0101 & -0.0102 & 0.0388 & -0.00017 \\ 0.0138 & -0.0595 & -0.00017 & 0.0163 \end{bmatrix},$$

and the closed-loop poles are -6.3396, -12.7427, -23.5384, -27.8096.

5.2.3 Linear Quadratic Control for Discrete-Time Systems

For discrete-time systems, the performance index in the quadratic form can be written as

$$J = \frac{1}{2} \sum_{k=0}^{N} \Big[\boldsymbol{x}^{\mathbf{T}}(k)\, \boldsymbol{Q} \boldsymbol{x}(k) + \boldsymbol{u}^{\mathbf{T}}(k) \boldsymbol{R} \boldsymbol{u}(k) \Big] \tag{5.21}$$

and its dynamic Riccati equation can be written as [53]

$$\boldsymbol{S}(k) = \boldsymbol{F}^{\mathbf{T}} \Big[\boldsymbol{S}(k+1) - \boldsymbol{S}(k+1) \boldsymbol{G} \boldsymbol{R}^{-1} \boldsymbol{G}^{\mathbf{T}} \boldsymbol{S}(k+1) \Big] \boldsymbol{F} + \boldsymbol{Q}\,, \tag{5.22}$$

where $\boldsymbol{S}(N) = \boldsymbol{Q}$, N is the termination instance, and $(\boldsymbol{F}, \boldsymbol{G})$ is the state space model of a discrete-time system. For the quadratic regulation problem, $\boldsymbol{S}$ is a constant matrix. The corresponding discrete-time ARE can be written as

$$\boldsymbol{S} = \boldsymbol{F}^{\mathrm{T}}\Big[\boldsymbol{S} - \boldsymbol{S}\boldsymbol{G}\boldsymbol{R}^{-1}\boldsymbol{G}^{\mathrm{T}}\boldsymbol{S}\Big]\boldsymbol{F} + \boldsymbol{Q}\,, \tag{5.23}$$

and the state feedback matrix is

$$\boldsymbol{K} = \Big[\boldsymbol{R} + \boldsymbol{G}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{G}\Big]^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{F}. \tag{5.24}$$

The discrete-time ARE can be solved by the `dare()` function, and the state feedback matrix $\boldsymbol{K}$ can be evaluated by the function `dlqr()`, with the syntax `[K,S]=dlqr (F,G,Q,R)`.

Example 5.7. Consider a discrete-time state space equation

$$\boldsymbol{x}(k+1) = \begin{bmatrix} 0.4725 & 0.2376 & 0.0589 & 0.1971 \\ 0.1451 & 0.5669 & 0.2311 & 0.0439 \\ 0.0932 & 0.119 & 0.5752 & 0.2319 \\ 0.2628 & 0.0757 & 0.1406 & 0.4465 \end{bmatrix}\boldsymbol{x}(k) + \begin{bmatrix} 0.5711 \\ -0.3999 \\ 0.6899 \\ 0.8156 \end{bmatrix} u(k).$$

Selecting the weighting matrices $\boldsymbol{Q} = \text{ diag}([0.5, 0.8, 2, 4])$ and $R = 1$, we can design the optimal controller with the following statements:

```
>> A=[0.4725,0.2376,0.0589,0.1971; 0.1451,0.5669,0.2311,0.0439;
      0.0932,0.119,0.5752,0.2319; 0.2628,0.0757,0.1406,0.4465];
   B=[0.5711; -0.3999; 0.6899; 0.8156]; Q=diag([0.5 0.8 2 4]); R=1;
   [K,P]=dlqr(A,B,Q,R), eig(A-B*K)
```

The state feedback vector $\boldsymbol{K}$ and the discrete-time ARE solution $\boldsymbol{P}$ are obtained as

$$\boldsymbol{K}^{\mathrm{T}} = \begin{bmatrix} 0.2582 \\ 0.13 \\ 0.3017 \\ 0.3932 \end{bmatrix},\ \boldsymbol{P} = \begin{bmatrix} 0.8206 & 0.3661 & 0.1423 & 0.2302 \\ 0.3661 & 1.5025 & 0.4916 & 0.233 \\ 0.1423 & 0.4916 & 2.7702 & 0.1581 \\ 0.2302 & 0.233 & 0.1581 & 4.2852 \end{bmatrix},$$

and with the state feedback vector $\boldsymbol{K}$, the closed-loop poles of the system are 0.7157, 0.4603, $0.1304 \pm \mathrm{j}\,0.0483$.

5.2.4 Selection of Weighting Matrices

It can be seen from the previous subsection that the performance of the LQR system is heavily dependent upon the selection of the weighting matrices. So, more precisely, we should say that the LQR is optimal with respect to the chosen $\boldsymbol{Q}$ and $\boldsymbol{R}$ weighting matrices. Thus an LQR solution which is optimal with one choice of $\boldsymbol{Q}$ and $\boldsymbol{R}$ will not normally be optimal for other choices of the $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices. The problem is that the specification for the performance of a practical control system will not be in terms of $\boldsymbol{Q}$ and $\boldsymbol{R}$, so the designer is faced with the problem of trying to find values of $\boldsymbol{Q}$ and $\boldsymbol{R}$ which will meet the specifications.

In SISO cases, since the matrix $\boldsymbol{R}$ is a nonzero scalar, one can fix it to unity and adjust only the matrix $\boldsymbol{Q}$. In this case, the original optimal control performance index in (5.14) can be equivalently represented as

$$\begin{aligned} \frac{J}{R} &= \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}(t_f)\frac{\boldsymbol{S}}{R}\boldsymbol{x}(t_f) + \frac{1}{2}\int_{t_0}^{t_f}\Big[\boldsymbol{x}^{\mathrm{T}}(t)\frac{\boldsymbol{Q}}{R}\boldsymbol{x}(t) + \boldsymbol{u}^{\mathrm{T}}(t)\boldsymbol{u}(t)\Big]\mathrm{d}t \\ &= \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}(t_f)\boldsymbol{S}_1\boldsymbol{x}(t_f) + \frac{1}{2}\int_{t_0}^{t_f}\Big[\boldsymbol{x}^{\mathrm{T}}(t)\,\boldsymbol{Q}_1\boldsymbol{x}(t) + u^2(t)\Big]\mathrm{d}t, \end{aligned} \tag{5.25}$$

where $\boldsymbol{S}_1 = \boldsymbol{S}\boldsymbol{R}^{-1}$ and $\boldsymbol{Q}_1 = \boldsymbol{Q}\boldsymbol{R}^{-1}$. In what follows, for simplicity we will replace $\boldsymbol{Q}_1$ and $\boldsymbol{S}_1$ with $\boldsymbol{Q}$ and $\boldsymbol{S}$, respectively. Several commonly used strategies for weighting matrix selection are summarized [55] below.

Cheap control

The term "cheap control" here means that control effort is inexpensive and one can use any large control signals to ensure the dynamic behavior of the system. In this case, the weight on $u(t)$ can be made very small, i.e., R is very small. Equivalently, under $R = 1$, the weight on $\boldsymbol{x}(t)$, i.e., $\boldsymbol{Q}$, should be very large. Usually, we use a single tuning knob ρ such that $\widehat{\boldsymbol{Q}} = \rho\,\boldsymbol{Q}$. The effect of a cheap control LQR design is demonstrated through an example below.

Example 5.8. Consider the plant model in Example 5.5. Let $\boldsymbol{Q} = \rho\ \text{diag}(10, 20, 6, 2, 5)$. For different values of ρ, the LQ optimal controllers can be designed using the following MATLAB statements:

```
>> A=[-0.2,0.5,0,0,0;0,-0.5,1.6,0,0;0,0,-14.3,85.8,0;
       0,0,0,-33.3,100;0,0,0,0,-10]; B=[0; 0; 0; 0; 30];
   Q=diag([10,20,6,2,5]); R=1; C=[1,0,0,0,0]; D=0;
   for rho=[1,10,100,100,10000]
      [K,P]=lqr(A,B,rho*Q,R); Ac=A-B*K; step(ss(Ac,B,C,D)); hold on
   end
```

The closed-loop step responses are shown in Figure 5.18.

It can be seen that when ρ increases, which means that the penalty on the states is increased, the magnitudes of the state responses are significantly reduced, which in turn make the output signal smaller.

Expensive control

In contrast to the cheap control strategy, with an expensive control strategy, the control cost is assumed to be quite large. Therefore, the control signal $u(t)$ should be made as small as possible. In this case, a large R should be used. With $R = 1$, equivalently, $\boldsymbol{Q}$ should be very small. Similarly, $\boldsymbol{Q}$ should be replaced by $\rho\,\boldsymbol{Q}$ with ρ very small.

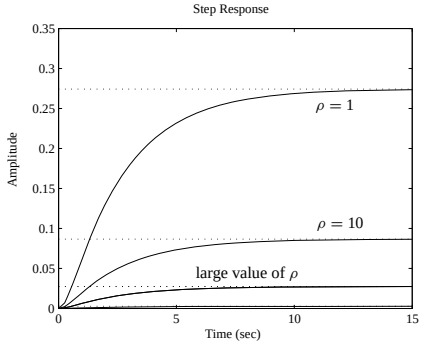


Figure 5.18. *Step responses of closed-loop systems.*

Terminal control

"Terminal control" means that in the dynamic optimal control problem, the weighting matrix $\boldsymbol{S}$ tends to infinity, which forces the terminal states to zero. A simple solution is to replace $\boldsymbol{S}$ by $\rho\boldsymbol{S}$ and send ρ to ∞ or a very large number. In this case, one can set $\boldsymbol{Q} = \boldsymbol{0}$ to reduce the DRE to the following form:

$$-\dot{\boldsymbol{P}} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{P}, \quad \text{where} \quad \boldsymbol{P}(t_f) = \rho\boldsymbol{S}. \tag{5.26}$$

Degree-of-stability design

If in the controller design all the closed-loop poles are located on the left-hand side of $s = -\alpha$ on the s-plane, where $\alpha > 0$, this is commonly referred to as the "degree-of-stability of $-\alpha$." To achieve this, a new performance index for LQR problems can be defined as

$$J = \int_0^\infty \mathrm{e}^{2\alpha t}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{x} + u^2)\mathrm{d}t, \tag{5.27}$$

where $\boldsymbol{Q}$ is a constant matrix. By introducing a new state variable vector $\boldsymbol{\xi}(t)$ such that $\boldsymbol{\xi}(t) = \mathrm{e}^{\alpha t}\boldsymbol{x}(t)$, and a new control $v(t) = \mathrm{e}^{\alpha t}u(t)$, the original state space equation can be written as $\dot{\boldsymbol{\xi}} = (\boldsymbol{A} + \alpha\boldsymbol{I})\boldsymbol{\xi} + \boldsymbol{B}v$. Then, (5.27) becomes

$$J = \int_0^\infty [\boldsymbol{\xi}^{\mathrm{T}}(t)\,\boldsymbol{Q}\boldsymbol{\xi}(t) + v^2(t)]\mathrm{d}t. \tag{5.28}$$

The modified ARE becomes

$$(\boldsymbol{A} + \alpha\boldsymbol{I})^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{P}(\boldsymbol{A} + \alpha\boldsymbol{I}) + \boldsymbol{Q} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{P} = \boldsymbol{0} \tag{5.29}$$

with the optimal control law $u^*(t) = -\boldsymbol{B}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x}(t)$.

Example 5.9. Consider again the plant model in Example 5.5. Set $\boldsymbol{Q} = \mathrm{diag}\,(10, 20, 6, 2, 5)$. The closed-loop poles achieved under the normal LQR control strategy can be obtained using the following MATLAB statements:

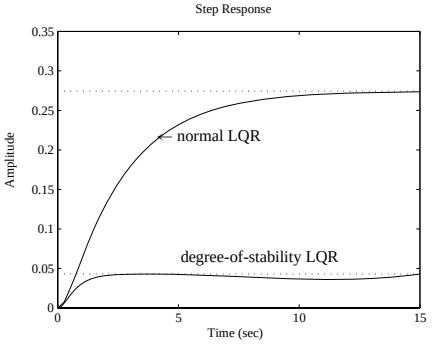


Figure 5.19. *Step responses of the closed-loop systems.*

```
>> A=[-0.2,0.5,0,0,0;0,-0.5,1.6,0,0;0,0,-14.3,85.8,0;
       0,0,0,-33.3,100;0,0,0,0,-10];
   B=[0; 0; 0; 0; 30]; Q=diag([10,20,6,2,5]); R=1;
   C=[1,0,0,0,0]; D=0; [K,P]=lqr(A,B,Q,R); eig(A-B*K)
```

where the closed-loop poles are -92.5275, $-52.5576 \pm \mathrm{j}63.9570$, -2.9264, -0.4046. It can be seen that there is one pole at $s = -0.4$ which is quite close to the imaginary axis. If one wants all the closed-loop poles of the system to be on the left-hand side of the line $s = -1$, the following MATLAB commands can be used to achieve this:

```
>> [K2,P2]=lqr(A+eye(size(A)),B,Q,R); eig(A-B*K2)
```

The closed-loop poles of the system are then -93.3510, $-53.4023 \pm \mathrm{j}64.1404$, -3.9272, -1.8793. Clearly, by the "degree-of-stability design" of the LQR, all the closed-loop poles have been moved to the left-hand side of $s = -1$. To show this benefit, the step responses of the closed-loop system before and after using the degree-of-stability design are compared in Figure 5.19 by the following MATLAB statements:

```
>> step(ss(A-B*K,B,C,D),ss(A-B*K2,B,C,D));
```

5.2.5 Observer and Observer Design

In LQR design, we have explicitly assumed that

1. the plant model is perfectly known, and
2. all the states are directly measurable.

If assumption 1 is not true, that is, the model may contain uncertainty, we shall use the robust control design framework, which is the subject of Chapter 7. In this section, we focus on the case when assumption 2 is not true. Actually, in practice, the state variables are usually not all measurable. If only the output signals rather than the states are measurable, which is often the case in many applications, can we still use an LQR? The answer is "yes" if we can design an observer to observe the states from the input and output information. Of course, the system has to be observable.

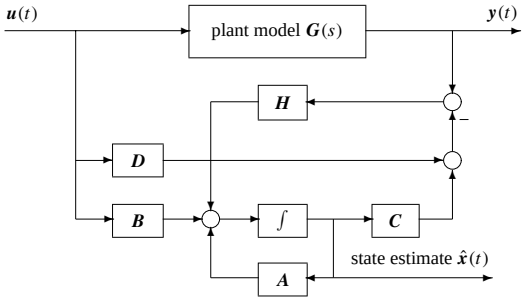


Figure 5.20. *A typical structure of state observers.*

One obvious method is to create an extra copy of the state space model of the original plant model ($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$). When these two plants are subjected to the same input signal, the state variables can be created or "observed" from the input signal alone. These states should be exactly the same as the states of the original model. This intuitively simple "open-loop" method for state observation is feasible only when the model is exact. However, when there exist some disturbances or the parameters of the original model are not exactly known, the created or "copied" system will fail to give a correct state observation for the original system. Therefore, in practical applications, the output signal should also be used, together with the input signal. This then allows a feedback loop to be implemented to correct any errors.

The typical control structure of a state observer is shown in Figure 5.20. If ($\boldsymbol{A}$, $\boldsymbol{C}$) is fully observable, the mathematical description of the state observer can be expressed by the following state space model:

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}u - \boldsymbol{H}(\boldsymbol{C}\hat{\boldsymbol{x}} + \boldsymbol{D}\boldsymbol{u} - \boldsymbol{y}) = (\boldsymbol{A} - \boldsymbol{H}\boldsymbol{C})\hat{\boldsymbol{x}} + (\boldsymbol{B} - \boldsymbol{H}\boldsymbol{D})\boldsymbol{u} + \boldsymbol{H}\boldsymbol{y}, \qquad (5.30)$$

where $\hat{\boldsymbol{x}}(t)$ is the observation or estimation of the true state vector $\boldsymbol{x}(t)$, and $\boldsymbol{H}$ is a matrix called the observer gain matrix, which is selected to make $(\boldsymbol{A} - \boldsymbol{H}\boldsymbol{C})$ stable. From (5.30), it can be further derived that

$$\dot{\hat{\boldsymbol{x}}} - \dot{\boldsymbol{x}} = (\boldsymbol{A} - \boldsymbol{H}\boldsymbol{C})\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{H}\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{u} = (\boldsymbol{A} - \boldsymbol{H}\boldsymbol{C})(\hat{\boldsymbol{x}} - \boldsymbol{x}), \qquad (5.31)$$

which has an analytical solution

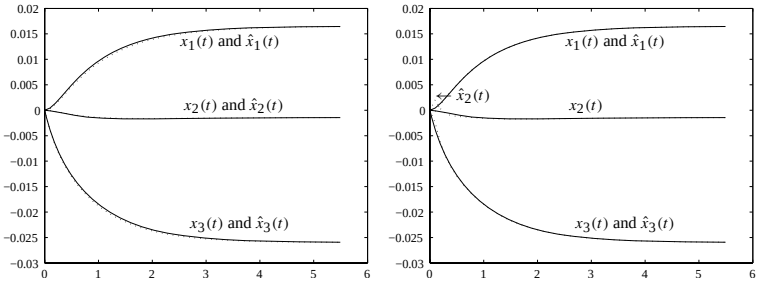
$$\hat{\boldsymbol{x}}(t) - \boldsymbol{x}(t) = \mathrm{e}^{(\boldsymbol{A}-\boldsymbol{H}\boldsymbol{C})(t-t_0)}[\hat{\boldsymbol{x}}(t_0) - \boldsymbol{x}(t_0)]. \qquad (5.32)$$

Since $(\boldsymbol{A} - \boldsymbol{H}\boldsymbol{C})$ is stable, $\lim_{t\to\infty}[\hat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)] = \boldsymbol{0}$, which means that the observed state $\hat{\boldsymbol{x}}(t)$ asymptotically converges to the true state $\boldsymbol{x}(t)$.

A MATLAB function `simobsv()` is written

```
function [xh,x,t]=simobsv(G,L)
[y,t,x]=step(G); G=ss(G); A=G.a; B=G.b; C=G.c; D=G.d;
[y1,xh1]=step((A-L*C),(B-L*D),C,D,1,t);
[y2,xh2]=lsim((A-L*C),L,C,D,y,t); xh=xh1+xh2;
```

which can be used to obtain the observed states of the system. The syntax of the function is `[`$\hat{\boldsymbol{x}}$`,`$\boldsymbol{x}$`,`t`]=simobsv(`$\boldsymbol{G}$`,`$\boldsymbol{H}$`)`, where $\boldsymbol{G}$ is an LTI object of the system model, and



(a) with gain $\boldsymbol{H}_1$ (b) with gain $\boldsymbol{H}_2$

Figure 5.21. *Step responses of state observers.*

$\boldsymbol{H}$ is the observer gain vector. The step responses of the observed or reconstructed states are returned in matrix $\hat{\boldsymbol{x}}$, and the original state vector is returned in $\boldsymbol{x}$. The time vector is returned in $\boldsymbol{t}$, which is automatically determined by the function.

Example 5.10. Consider the following state space model:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} -3.6994 & 0.6627 & -2.3879 \\ 0.6627 & -1.4220 & 0.4994 \\ -2.3879 & 0.4994 & -3.2736 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ -0.0449 \end{bmatrix} u,$$

$$y = [-0.7989, -0.7652, 0.8617]\,\boldsymbol{x}.$$

Design an observer with the observer gain $\boldsymbol{H}_1^{\mathrm{T}} = [-12.6270, -1.0468, -4.0212]$. The eigenvalues of the observer are obtained using the following MATLAB statements:

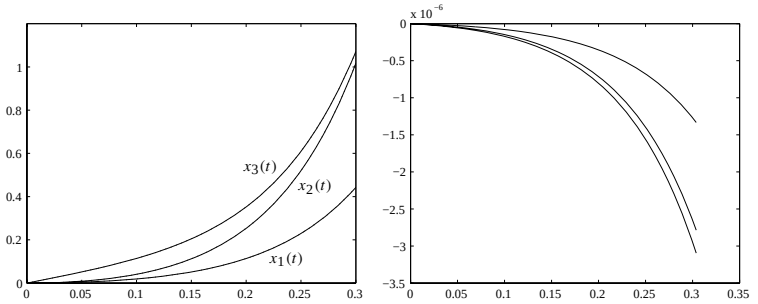
```
>> A=[-3.6994,0.6627,-2.3879;0.6627,-1.4220,0.4994;
      -2.3879,0.4994,-3.2736];
   B=[0; 0; -0.0449]; C=[ -0.7989,-0.7652,0.8617]; D=0;
   H1=[-29.9482; 7.2402; -27.2489];  eig(A-H1*C)
   [xh,x,t]=simobsv(ss(A,B,C,D),H1); plot(t,x,'-',t,xh,':')
```

The closed-loop poles are -0.9819, -1.1305, -1.1877, and the step responses of the true states and their observations are compared in Figure 5.21(a). If the observer gain is changed to $\boldsymbol{H}_2^{\mathrm{T}} = [-650.74, 2173.52, 1355.35]$, a different set of observer poles are shown as follows:

```
>> H2=[-650.74; 2173.52; 1355.35]; eig(A-H2*C)
   [xh,x,t]=simobsv(ss(A,B,C,D),H2); plot(t,x,'-',t,xh,':')
```

The closed-loop poles are -11.9613, -11.0561, -9.9814. The corresponding step responses are compared in Figure 5.21(b).

Clearly, a different choice of the observer gain will lead to a different transient behavior of the observer dynamics. It is generally true that the further away the closed-loop poles are from the imaginary axis, the quicker the transient responses vanish with time.



(a) states and reconstructed states (b) state errors with observers

Figure 5.22. *State observers of unstable models.*

Example 5.11. Consider an unstable state space model

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u, \quad y = [1, 2, 3]\boldsymbol{x}.$$

An observer with the gain matrix $\boldsymbol{H}^{\mathrm{T}} = [1.2222; 2.4786; 1.9402]$, which makes $\boldsymbol{A} - \boldsymbol{HC}$ stable, can be designed by the following MATLAB scripts:

```
>> A=[1,2,3; 4,5,6; 7,8,0]; B=[0; 0; 1]; C=[1,2,3]; D=0;
   H=[1.2222; 2.4786; 1.9402]; [xh,x,t]=simobsv(ss(A,B,C,D),H);
   plot(t,x,'-',t,xh,':'); figure; plot(t,x-xh)
   plot(t,e(:,1),'-',t,e(:,2),'--',t,e(:,3),':')
```

The step responses of the true states and the observed ones are compared in Figure 5.22(a), with the observer error signals shown in Figure 5.22(b).

It is concluded that even if the original model is unstable, the designed state observer can still reconstruct the states of the system satisfactorily, provided that a suitable observer gain vector is designed. Of course, the original system, although unstable, has to be fully observable.

5.2.6 State Feedback and Observer-Based Controllers

Having designed a suitable state observer, the state feedback control strategy with observers can be implemented as shown in Figure 5.23.

Consider the feedback structure shown in Figure 5.20(a). The feedback signal $\boldsymbol{K}\hat{\boldsymbol{x}}(t)$ can be rewritten, using (5.30), as two subsystems $\boldsymbol{G}_1(s)$ and $\boldsymbol{G}_2(s)$, driven only by $\boldsymbol{u}(t)$ and $\boldsymbol{y}(t)$, respectively. Therefore, $\boldsymbol{G}_1(s)$ can be described by

$$\dot{\hat{\boldsymbol{x}}}_1(t) = (\boldsymbol{A} - \boldsymbol{HC})\hat{\boldsymbol{x}}_1(t) + (\boldsymbol{B} - \boldsymbol{HD})\boldsymbol{u}(t), \quad \boldsymbol{y}_1 = \boldsymbol{K}\hat{\boldsymbol{x}}_1(t), \tag{5.33}$$

and $\boldsymbol{G}_2(s)$ by

$$\dot{\hat{\boldsymbol{x}}}_2(t) = (\boldsymbol{A} - \boldsymbol{HC})\hat{\boldsymbol{x}}_2(t) + \boldsymbol{H}\boldsymbol{y}(t), \quad \boldsymbol{y}_2 = \boldsymbol{K}\hat{\boldsymbol{x}}_2(t). \tag{5.34}$$

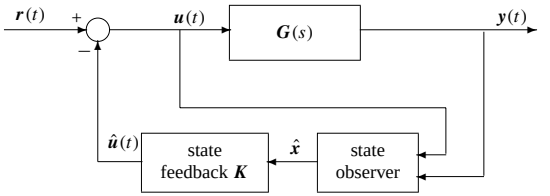
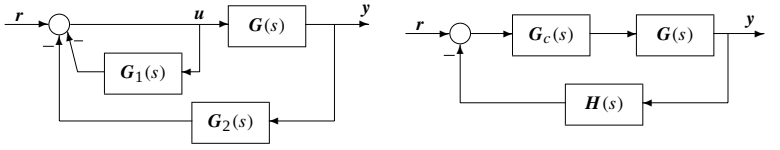


Figure 5.23. *Feedback control with a state observer.*



(a) (b)

Figure 5.24. *Observer-based state feedback control.*

The block diagram for the closed-loop system is represented in Figure 5.24(a), where with some block manipulations, the closed-loop system can be equivalently expressed as shown in Figure 5.24(b) with $\boldsymbol{G}_c(s) = [\boldsymbol{I} + \boldsymbol{G}_1(s)]^{-1}$ and $\boldsymbol{H}(s) = \boldsymbol{G}_2(s)$, which is identical to the typical feedback control system structure. The controller $\boldsymbol{G}_c(s)$ can be further derived in the following form:

$$\boldsymbol{G}_c(s) = \boldsymbol{I} - \boldsymbol{K}(s\boldsymbol{I} - \boldsymbol{A} + \boldsymbol{BK} + \boldsymbol{HC})^{-1}\boldsymbol{B} \tag{5.35}$$

with its state space realization

$$\boldsymbol{G}_c(s) = \left[\begin{array}{c:c} \boldsymbol{A} - \boldsymbol{BK} - \boldsymbol{HC} & \boldsymbol{B} \\ \hdashline -\boldsymbol{K} & \boldsymbol{I} \end{array}\right]. \tag{5.36}$$

If the reference input $\boldsymbol{r}(t) = \boldsymbol{0}$, $\boldsymbol{G}_c(s)$ can be further simplified into the following state space representation:

$$\boldsymbol{G}_c(s) = \left[\begin{array}{c:c} \boldsymbol{A} - \boldsymbol{BK} - \boldsymbol{HC} + \boldsymbol{HDK} & \boldsymbol{H} \\ \hdashline \boldsymbol{K} & \boldsymbol{0} \end{array}\right]. \tag{5.37}$$

The above simplified form, with $\boldsymbol{r}(t) = \boldsymbol{0}$ and a unity negative feedback, will be used throughout the book unless otherwise stated. The lumped controller $G_c(s)$ in (5.37) is often referred to as the observer-based controller, since the structural information of the observer is implicitly reflected within the controller.

The observer-based controller can also be obtained using the MATLAB function `reg()` provided in the Control Systems Toolbox with the following syntax:

```
[Ac,Bc,Cc,Dc]=reg(A,B,C,D,K,H)
Gc=-reg(G,K,H)
```

where the state space model of the plant, ($\boldsymbol{A}$,$\boldsymbol{B}$,$\boldsymbol{C}$,$\boldsymbol{D}$), the state feedback gain vector $\boldsymbol{K}$, and the observer gain vector $\boldsymbol{H}$ are then returned, respectively. The returned ($\boldsymbol{A}_c$,$\boldsymbol{B}_c$,$\boldsymbol{C}_c$,$\boldsymbol{D}_c$) is

the state space model of the observer-based controller $\boldsymbol{G}_c(s)$. In the second statement of syntax above, LTI models can be used directly. In fact, `reg()` is an immediate implementation of (5.37).

Example 5.12. Consider the LQ optimal control problem in Example 5.5. If the weighting matrix $\boldsymbol{Q}$ is selected as $\boldsymbol{Q} = \text{diag}([1, 0, 0, 0, 0])$ with $R = 1$, and if an observer gain vector is selected as $\boldsymbol{H} = [-8.3, 979.24, -19367.61, 4293.85, 0]^{\mathbf{T}}$, with the following MATLAB statements:

```
>> A=[-0.2,0.5,0,0,0;0,-0.5,1.6,0,0;0,0,-14.3,85.8,0;
       0,0,0,-33.3,100;0,0,0,0,-10];
   B=[0; 0; 0; 0; 30]; Q=diag([1,0,0,0,0]); R=1;
   C=[1,0,0,0,0]; D=0; G=ss(A,B,C,D); [K,P]=lqr(A,B,Q,R);
   H=[-8.3, 979.24, -19367.61, 4293.85, 0]';
   Gc=-reg(ss(A,B,C,D),K,H); zpk(Gc)
```

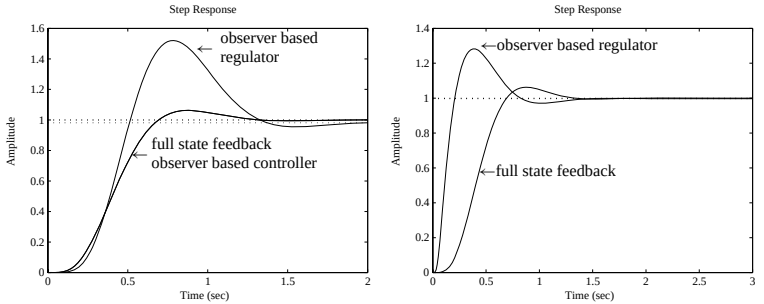
then the resulting controller model can be obtained as

$$G_c(s) = \frac{11.4839(s + 33.34)(s + 14.3)(s + 10)(s + 1.792)}{(s + 20.92)(s^2 + 30.19s + 328.1)(s^2 + 6.845s + 120)}.$$

The controller $G_c(s)$ can be designed and it is a stable minimum phase model. To check the time domain performance of the observer-based control, the following MATLAB statements can be used:

```
>> GG1=feedback(G*Gc,1); GG=ss(A-B*K,B,C,D);
   G1=ss(A-B*K-H*C,B,-K,1); G2=ss(A-H*C,H,K,0);
   GG2=feedback(G*G1,G2); step(GG,GG1,GG2,2)
```

to get the closed-loop step response of the system under the above designed controller as shown in Figure 5.25(a), where for comparison, the step response under the full state feedback, i.e., all states are measurable, is also plotted. It can be observed that the response under the observer-based controller is not exactly the same as the one under the direct full



(a) LQ controller comparison (b) with another $\boldsymbol{H}$

Figure 5.25. *Step responses of LQ optimal control.*

state feedback control. It is worth mentioning that it is not fair to compare it with the direct full state feedback, since in deriving the regulator, it has been assumed that the external input signal $r(t) = 0$.

Let us continue the example with an adjusted observer gain vector

$$\boldsymbol{H} = [441.7, 146689.24, 6670765.515, 2866690.238, 860270.98]^{\mathbf{T}}.$$

To design the new observer-based controller, the following MATLAB statements can be used:

```
>> H=[441.7,146689.24,6670765.515,2866690.238,860270.98]';
   G=ss(A,B,C,D); Gc=-reg(G,K,H); zpk(Gc)
   G_c=feedback(G*Gc,1); step(ss(A-B*K,B,C,D),G_c,3)
```

The new regulator is

$$G_c(s) = \frac{464353.6388(s + 33.3)(s + 14.48)(s + 8.953)(s + 4.973)}{(s + 157.4)(s^2 + 104.3s + 5008)(s^2 + 246.2s + 1.814)}.$$

The step response comparison is shown in Figure 5.25(b). It can be seen that, as expected, the performance of the observer-based regulator depends on the observer gain vector $\boldsymbol{H}$ and, of course, the state feedback gain vector $\boldsymbol{K}$.

5.3 Pole Placement Design

It has been demonstrated that the dynamic behavior of a controlled system is influenced by its closed-loop poles. If the system is fully controllable, the poles of the closed-loop system can be arbitrarily shifted or placed in any prespecified positions. Given the desired pole positions and the system model, in this section we show how to design the controller to shift the original system poles to the desired positions. This is referred to as the pole placement controller design method.

Consider the state space model of a plant given by

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \ \ \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}, \tag{5.38}$$

where the $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ matrices have compatible dimensions. State feedback is to be used with a constant gain matrix $\boldsymbol{K}$, and the external reference input to the system is denoted by $\boldsymbol{r}$. Then, the actual control signal applied to the plant is $\boldsymbol{u} = \boldsymbol{r} - \boldsymbol{K}\boldsymbol{x}$. The closed-loop state space model can be written as

$$\dot{\boldsymbol{x}} = (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})\boldsymbol{x} + \boldsymbol{B}\boldsymbol{r}, \ \ \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}. \tag{5.39}$$

An important theorem regarding the state feedback and its use in pole placement is given below.

Theorem 5.1. If the system $(\boldsymbol{A}, \boldsymbol{B})$ is fully controllable, the eigenvalues of $\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}$ can be freely assigned (with the restriction that complex eigenvalues are in conjugate pairs) by a suitable matrix $\boldsymbol{K}$.

From Theorem 5.1, if the given system is fully controllable, the closed-loop poles of the system can be assigned arbitrarily through a static state feedback. In what follows, some of the commonly used pole placement algorithms are introduced.

5.3.1 The Bass–Gura Algorithm

Assume that the desired closed-loop poles of the system are $\mu_i, i = 1, \ldots, n$. Clearly, the closed-loop characteristic equation $\alpha(s)$ is simply

$$\alpha(s) = \prod_{i=1}^{n}(s - \mu_i) = s^n + \alpha_1 s^{n-1} + \alpha_2 s^{n-2} + \cdots + \alpha_{n-1}s + \alpha_n. \tag{5.40}$$

Denote by $a(s)$ the open-loop characteristic equation of the original plant model, which is written as

$$a(s) = \det(s\boldsymbol{I} - \boldsymbol{A}) = s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_{n-1}s + a_n. \tag{5.41}$$

If the original plant model is fully controllable, the state feedback gain vector $\boldsymbol{K}$ can be obtained from [56] as

$$\boldsymbol{K} = [\boldsymbol{a} - \boldsymbol{\alpha}]^{\mathrm{T}} \boldsymbol{L}^{-1} \widehat{\boldsymbol{C}}^{-1}, \tag{5.42}$$

where

$$[\boldsymbol{\alpha} - \boldsymbol{a}]^{\mathrm{T}} = [(\alpha_1 - a_1), \ldots, (\alpha_n - a_n)], \quad \widehat{\boldsymbol{C}} = [\boldsymbol{B}, \boldsymbol{AB}, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{B}],$$

and

$$\boldsymbol{L} = \begin{bmatrix} a_{n-1} & a_{n-2} & \cdots & a_1 & 1 \\ a_{n-2} & a_{n-3} & \cdots & 1 & \\ \vdots & \vdots & \ddots & & \\ a_1 & 1 & & & \\ 1 & & & & \end{bmatrix}. \tag{5.43}$$

It can be seen that $\boldsymbol{L}$ is a nonsingular Hankel matrix.

A MATLAB implementation of the above pole placement algorithm is given in `bass_pp()`, where

```
function K=bass_pp(A,B,p)
n=length(B); a1=poly(p); alpha=[a1(n:-1:2),1];
a=poly(A); aa=[a(n:-1:2),1]; L=hankel(aa); C=ctrb(A,B);
K=(a1(n+1:-1:2)-a(n+1:-1:2))*inv(L)*inv(C);
```

The syntax of `bass_pp()` is `K=bass_pp(A,B,p)`, where (**A**,**B**) is the state space model and **p** is a vector containing the expected pole positions. The returned variable **K** is the state feedback gain vector.

5.3.2 Ackermann's Algorithm

The pole placement problem can alternatively be solved in a slightly different way using Ackermann's algorithm. The state feedback gain vector $\boldsymbol{K}$ is given by the following formula:

$$\boldsymbol{K} = -[0, 0, \ldots, 0, 1]\widehat{\boldsymbol{C}}^{-1}\boldsymbol{\alpha}, \tag{5.44}$$

where $\boldsymbol{\alpha}^{\mathrm{T}} = [\alpha_0, \alpha_1, \ldots, \alpha_{n-1}]$.

A MATLAB function `acker()`, provided in the Control Systems Toolbox, implements the above algorithm with its syntax the same as that of `bass_pp()`.

5.3.3 Numerically Robust Pole Placement Algorithm

It has been found that the above two pole placement algorithms may not be numerically robust. A MATLAB function `place()` provided in the Control Systems Toolbox can be used to find the feedback matrix **K** using a numerically robust pole placement algorithm [57]. The syntax is

`K=place(A,B,p)`

where (**A**,**B**) is the state space model, and **p** is a vector containing the expected pole positions. The returned variable **K** is the state feedback gain matrix.

It should be pointed out that the `place()` function can be used to deal with MIMO (multiple input–multiple output) pole placement problems. However, in `place()`, the expected pole positions have to be distinct, that is, the multiplicity of any desired pole cannot be greater than 1. On the other hand, `acker()`, although it cannot handle the MIMO pole placement problem, can be used for pole placement with desired poles of any multiplicity.

Example 5.13. Given the plant model

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 11 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix} u, \quad y = [1, 2, 3, 4]\boldsymbol{x},$$

design a state feedback controller to place the closed-loop poles at $s_{1,2,3,4} = -1, -2, -1 \pm \mathrm{j}1$. To design the **K** for pole placement, the following MATLAB statements can be used:

```
>> A=[0,1,0,0; 0,0,-1,0; 0,0,0,1; 0,0,11,0]; B=[0;1;0;-1];
   eig(A)', P=[-1; -2; -1+sqrt(-1); -1-sqrt(-1)];
   K=place(A,B,P), eig(A-B*K)'
```

It is found that the open-loop poles are $0, 0, \pm 3.3166$, indicating that the original system is unstable. The state feedback vector for placing the poles at the expected location is $\boldsymbol{K} = [-0.4, -1, -21.4, -6]$. Since the original system is fully controllable, via the pole placement technique, one is able to place the poles of the closed-loop system at the desired positions.

Example 5.14. Consider the multivariable model

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} 0 & 2 & 0 & 0 & -2 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 1 & 2 \\ 0 & 0 \\ 0 & 1 \\ 0 & -1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{u}(t).$$

If one wants to place the closed-loop poles at $-1, -2, -3, -4, -1 \pm \mathrm{j}$, the following statements can be used:

```
>> A=[0,2,0,0,-2,0; 1,0,0,0,0,-1; 0,1,0,0,0,0;
      0,0,0,3,0,0;  2,0,0,1,0,0; 0,0,-1,0,1,0];
   B=[1,2; 0,0; 0,1; 0,-1; 0,1; 0,0];
   p=[-1 -2 -3 -4 -1+1i -1-1i]; K=place(A,B,p), eig(A-B*K)'
```

The feedback control gain matrix can be obtained as

$$\boldsymbol{K} = \begin{bmatrix} 7.9333 & -18.553 & -19.134 & 20.65 & 18.698 & 22.126 \\ -0.36944 & -2.0412 & -2.3166 & -9.5475 & 0.57469 & 1.5013 \end{bmatrix}$$

and it can be found that the poles are at the desired locations.

Example 5.15. Consider the fourth-order system

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} -5 & 8 & 0 & 0 \\ -4 & 7 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & -2 & 6 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 4 \\ -2 \\ 2 \\ 1 \end{bmatrix} u(t).$$

To assign the poles of the closed-loop system to $s_{1,2,3,4} = -1, -2, -1 \pm \mathrm{j}1$, the following MATLAB statements can be applied:

```
>> A=[-5,8,0,0;-4,7,0,0;0,0,0,4;0,0,-2,6]; B=[4;-2;2;1];
   C=[2,-2,-2,2]; D=0; P=[-1; -2;-1+sqrt(-1);-1-sqrt(-1)];
   K=place(A,B,P)
```

The user is then prompted that the system "can't place eigenvalues there," which means that the closed-loop poles cannot be assigned to the prespecified positions. Let us check the controllability of the system using

```
>> Tc=ctrb(A,B); rank(Tc)
```

The rank of $\boldsymbol{T}_c$ is found to be 3 rather than 4, which means that the original system is actually not fully controllable. Therefore, the poles of the closed-loop system cannot be freely assigned.

Example 5.16. Consider the discrete-time state space model given by [53]

$$\boldsymbol{x}(k+1) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -0.91 & -0.036 & 0.91 & 0.036 \\ 0 & 0 & 0 & 1 \\ 0.091 & 0.036 & -0.091 & -0.036 \end{bmatrix} \boldsymbol{x}(k) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u(k),\ y(k) = x_1(k)$$

with sampling interval $T = 0.1$ second. If one wants to place all the closed-loop poles at 0.9, the following statements can be used:

```
>> F=[0,1,0,0; -0.91 -0.036 0.91 0.036;
      0 0 0 1; 0.091 0.036 -0.091 -0.036];
   G=[0; 0; 0; 1]; C=[1 0 0 0]; W=ss(F,G,C,0,'Ts',0.1);
   p=0.9*ones(4,1); K=acker(F,G,p), eig(F-G*K)
```


The state feedback vector is $\boldsymbol{K} = [-3.2544, 0.4391, 3.9754, -3.6720]$. Under such a state feedback gain vector, it can be found that the poles of the closed-loop system are located at $0.9001, 0.9000 \pm \mathrm{j}0.0001, 0.8999$.

5.3.4 Observer Design Using the Pole Placement Technique

Using the pole placement technique discussed above, one can design a state observer by assigning the observer poles to prespecified positions. Since pole placement in observer design is simply a dual problem to the state feedback design, the problem can be solved using the MATLAB function `place()` or `acker()`, as illustrated in the following example.

Example 5.17. Consider the fourth-order system in Example 5.13. The desired poles of the observer are specified as $s_{1,2,3,4} = -1, -2, -1 \pm \mathrm{j}1$. To determine the observer gain matrix $\boldsymbol{H}$, the following MATLAB statements can be applied:

```
>> A=[0,1,0,0; 0,0,-1,0; 0,0,0,1; 0,0,11,0]; B=[0;1;0;-1];
   C=[1,2,3,4]; P=[-1; -2; -1+sqrt(-1); -1-sqrt(-1)];
   H=place(A',C',P)', eig(A-H*C)'
```

It can be found that the observer vector is $\boldsymbol{H}^{\mathrm{T}} = [-0.2203, -0.4750, 0.4238, 1.2247]$, and it can be seen that the poles of the observer can indeed be placed at the desired locations.

5.3.5 Observer-Based Controller Design Using the Pole Placement Technique

We have discussed controller and observer design using the pole placement technique. Clearly, the pole placement technique can be used to design the observer-based controller. This observer-based controller has been discussed in Sec. 5.2.6 for LQ optimal control problems. Under the typical feedback control system structure as shown in Figure 5.24(b), $\boldsymbol{G}_c(s)$ and $\boldsymbol{H}(s)$ can be represented by $[(\boldsymbol{A} - \boldsymbol{BK} - \boldsymbol{HC}), \boldsymbol{B}, -\boldsymbol{K}, \boldsymbol{I}]$ and $[(\boldsymbol{A} - \boldsymbol{HC}), \boldsymbol{H}, \boldsymbol{C}, \boldsymbol{0}]$, respectively. With the reference $\boldsymbol{r} = \boldsymbol{0}$, the observer-based regulator is then represented as

$$\boldsymbol{G}_c(s) = \left[\begin{array}{c|c} \boldsymbol{A} - \boldsymbol{BK} - \boldsymbol{HC} + \boldsymbol{HDK} & \boldsymbol{H} \\ \hline \boldsymbol{K} & \boldsymbol{0} \end{array}\right], \tag{5.45}$$

which can be obtained again using the MATLAB function `reg()` provided in the Control Systems Toolbox.

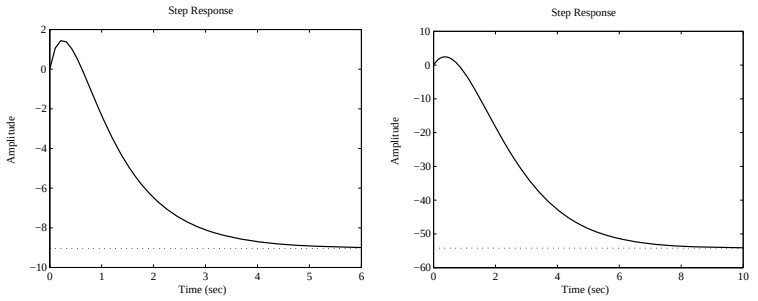
Example 5.18. Consider a plant model given by

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} -0.3 & 0.1 & -0.05 \\ 1 & 0.1 & 0 \\ -1.5 & -8.9 & -0.05 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 2 \\ 0 \\ 4 \end{bmatrix} u(t).$$

Assume that the desired pole positions of the closed-loop system are $-1, -2, -3$, respectively. From the following MATLAB statements:

```
>> A=[-0.3,0.1,-0.05; 1,0.1,0; -1.5,-8.9,-0.05]; B=[2; 0; 4];
   C=[1,2,3]; D=0; P1=[-1,-2,-3];  Kp1=place(A,B,P1);
   step(ss(A-B*Kp1,B,C,D));
```

the step response of the output signal is obtained as shown in Figure 5.26(a).



(a) system poles at $(-1, -2, -3)$ (b) system poles at $(-1, -1, -1)$

Figure 5.26. *Step responses of the compensated system with state feedback.*

If one tries to place all the poles at -1 with the following MATLAB statements:

```
>>  P2=[-1,-1,-1];  Kp2=place(A,B,P2);
```

then the prompt "can't place poles with multiplicity greater than rank($\boldsymbol{B}$)" is given. This means that for the desired poles with multiplicity greater than 1, `place()` fails to perform the pole placement. In this case, the function `acker()` should be used. Let us try the following MATLAB statements:

```
>> Kp2=acker(A,B,P2); step(ss(A-B*Kp2,B,C,D));
```

One can perform the pole placement successfully. The closed-loop response is shown in Figure 5.26(b), where it can be seen that, compared with Figure 5.26(a), the speed of response is reduced and the magnitude of the output is increased.

Now, let us assume that the states are not directly measurable. One has to use the observer-based controller. To design an observer with its desired poles located at $(-1, -2, -3)$, one can use the following MATLAB statements:

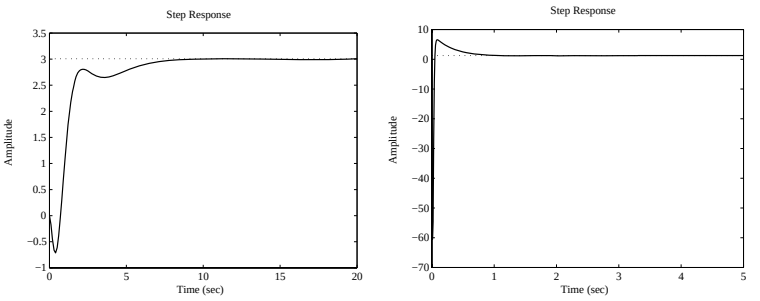
```
>> P4=[-1,-2,-3]; Ko1=place(A',C',P4)'; Ac=A-B*Kp1-Ko1*C;
   Ah=A-Ko1*C; t=0:.1:20; G=ss(A,B,C,D); Gc=ss(Ac,Ko1,Kp1,0);
   G_c=feedback(G*Gc,1); step(G_c,t)
```

with the closed-loop step response shown in Figure 5.27(a), which is completely different than the one using the direct state feedback shown in Figure 5.26(a). Due to the observer dynamics, the expected system behavior under the direct state feedback may change significantly. In Chapter 7 this issue will be discussed further.

Let us examine what will happen if the desired observer pole positions are chosen further left, e.g., $(-100, -200, -150)$. By the following MATLAB statements:

```
>> P5=[-100;-200;-150]; Ko2=place(A',C',P5)';
   Ac=A-B*Kp1-Ko2*C; Ah=A-Ko2*C; t=0:.01:5; G=ss(A,B,C,D);
   Gc=ss(Ac,Ko2,Kp1,0); G_c=feedback(G*Gc,1); step(G_c,t)
```

the step response of the closed-loop system is shown in Figure 5.27(b), which is very different to what is shown in Figure 5.27(a).



(a) observer poles at $(-1, -2, -3)$ (b) observer poles at $(-100, -200, -150)$

Figure 5.27. *Step responses of observed-based systems.*

5.4 Decoupling Control of Multivariable Systems

5.4.1 Decoupling Control with State Feedback

Consider a plant described by the state space model $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ with m inputs and m outputs. If the control signal $\boldsymbol{u}$ is constructed by state feedback such that $\boldsymbol{u} = \boldsymbol{\Gamma r} - \boldsymbol{Kx}$, the closed-loop transfer function matrix can be written as

$$\boldsymbol{G}(s) = \Big[(\boldsymbol{C} - \boldsymbol{DK})(s\boldsymbol{I} - \boldsymbol{A} + \boldsymbol{BK})^{-1}\boldsymbol{B} + \boldsymbol{D}\Big]\boldsymbol{\Gamma}. \tag{5.46}$$

Defining the order d_j for each j, $j = 1, \ldots, m$, such that it is the lowest order which makes $\boldsymbol{c}_j^{\mathrm{T}}\boldsymbol{A}^i\boldsymbol{B} \neq \boldsymbol{0}$, $i = 0, 1, \ldots, n-1$, and $\boldsymbol{c}_j^{\mathrm{T}}$ is the jth row of matrix $\boldsymbol{C}$.

Theorem 5.2. If the $m \times m$ matrix

$$\boldsymbol{B}_1 = \begin{bmatrix} \boldsymbol{c}_1^{\mathrm{T}}\boldsymbol{A}^{d_1}\boldsymbol{B} \\ \vdots \\ \boldsymbol{c}_m^{\mathrm{T}}\boldsymbol{A}^{d_m}\boldsymbol{B} \end{bmatrix} \tag{5.47}$$

is nonsingular, the closed-loop system defined in (5.46) can be dynamically decoupled if the state feedback matrix $\boldsymbol{K}$ can be established [56] as

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{c}_1^{\mathrm{T}}\boldsymbol{A}^{d_1+1} \\ \vdots \\ \boldsymbol{c}_m^{\mathrm{T}}\boldsymbol{A}^{d_m+1} \end{bmatrix} \boldsymbol{\Gamma}, \tag{5.48}$$

where $\boldsymbol{\Gamma} = \boldsymbol{B}_1^{-1}$.

A MATLAB function `decoupler()` is written using the above algorithm to design the decoupler:

```
function [G1,K,d,Gam]=decoupler(G)
A=G.a; B=G.b; C=G.c; [n,m]=size(G.b); B1=[]; K0=[];
for j=1:m,
```

```
    for k=0:n-1
       if norm(C(j,:)*A^k*B)>eps, d(j)=k; break; end
    end
    B1=[B1; C(j,:)*A^d(j)*B]; K0=[K0; C(j,:)*A^(d(j)+1)];
end
Gam=inv(B1); K=Gam*K0; G1=minreal(tf(ss(A-B*K,B,C,G.d))*inv(B1));
```

The function can be called `[G1,K,d,Gam]=decoupler(G)`, where $\boldsymbol{G}$ is the original multivariable state space model, $\boldsymbol{G}_1$ is the decoupled transfer function matrix, and $\boldsymbol{K}$ is the state feedback matrix as in (5.48). The vector $\boldsymbol{d}$ contains the values of d_j defined above. Matrix $\boldsymbol{\Gamma}$ is the precompensation matrix.

Example 5.19. Consider the two input–two output system discussed in Example 2.6, which is rewritten as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 2.25 & -5 & -1.25 & -0.5 \\ 2.25 & -4.25 & -1.25 & -0.25 \\ 0.25 & -0.5 & -1.25 & -1 \\ 1.25 & -1.75 & -0.25 & -0.75 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 4 & 6 \\ 2 & 4 \\ 2 & 2 \\ 0 & 2 \end{bmatrix} \boldsymbol{u}, \quad \boldsymbol{y} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \end{bmatrix} \boldsymbol{x}.$$

The state feedback gain matrix $\boldsymbol{K}$ can be designed and the system can be fully decoupled by using the following:

```
>> A=[2.25, -5, -1.25, -0.5;  2.25, -4.25, -1.25, -0.25;
      0.25, -0.5, -1.25,-1;  1.25, -1.75, -0.25, -0.75];
   B=[4, 6; 2, 4; 2, 2; 0, 2]; C=[0, 0, 0, 1; 0, 2, 0, 2];
   D=zeros(2,2); G=ss(A,B,C,D); [G1,K,d,Gam]=decoupler(G)
```

The state feedback matrix $\boldsymbol{K}$ and matrix $\boldsymbol{\Gamma}$ can be obtained, and it can be seen that the transfer function matrix $\boldsymbol{G}_1(s)$ is fully decoupled:

$$\boldsymbol{G}_1(s) = \begin{bmatrix} \frac{1}{s} & 0 \\ 0 & \frac{1}{s} \end{bmatrix}, \quad \boldsymbol{K} = \frac{1}{8}\begin{bmatrix} -1 & -3 & -3 & 5 \\ 5 & -7 & -1 & -3 \end{bmatrix}, \quad \boldsymbol{\Gamma} = \begin{bmatrix} -1.5 & 0.25 \\ 0.5 & 0 \end{bmatrix}.$$

By introducing the state feedback $\boldsymbol{K}$ and a precompensator $\boldsymbol{\Gamma}$, the square multivariable system can be fully decoupled. The decoupled transfer function matrix can usually be expressed by

$$\boldsymbol{G}_1 = \operatorname{diag}\left(\left[\frac{1}{s^{d_1+1}}, \ldots, \frac{1}{s^{d_m+1}}\right]\right).$$

By introducing the decoupling compensator $(\boldsymbol{K}, \boldsymbol{\Gamma})$, the full feedback control structure can be established as shown in Figure 5.28. Since the control system within the dashbox can be fully decoupled, the outer controller $\boldsymbol{G}_c(s)$ can be easily designed for the system, on an individual loop basis.

5.4.2 Pole Placement of Decoupling Systems with State Feedback

It is seen that the dynamic decoupling system presented previously can be used only to decouple the multivariable system into integrator types of diagonal system, which makes

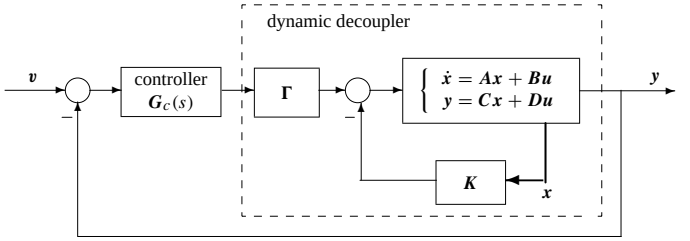


Figure 5.28. *Decoupling and controller structure.*

the closed-loop design difficult. If one can still assume that the decoupling law is state feedback which can be written as $\boldsymbol{u} = \boldsymbol{\Gamma r} - \boldsymbol{Kx}$, one may expect to have a decoupled system in the form

$$\boldsymbol{G}_{\boldsymbol{K},\boldsymbol{\Gamma}}(s) = \begin{bmatrix} \frac{1}{s^{d_1+1} + a_{1,1}s^{d_1} + \cdots + a_{1,d_1+1}} & & \\ & \ddots & \\ & & \frac{1}{s^{d_m+1} + a_{m,1}s^{d_m} + \cdots + a_{m,d_m+1}} \end{bmatrix}, \tag{5.49}$$

where d_i, $i = 1, \ldots, m$ are defined as previously. The parameters in each of the polynomials $s^{d_i+1} + a_{i,1}s^{d_i} + \cdots + a_{i,d_i+1}$ can be assigned by the pole placement method.

The expected polynomial can be defined as the standard transfer function. The standard transfer function of an nth order system, with the integral of time-multiplied absolute value of error (ITAE) optimal criterion, is defined in the form [58]

$$T(s) = \frac{1}{s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_{n-1}s + a_n}, \tag{5.50}$$

where the coefficients a_i minimizing the ITAE criterion of the system $T(s)$ have been precomputed, as summarized in Table 5.1.

Table 5.1. *Standard models under ITAE criterion (type I).*

n	Overshoot	$\omega_n t_s$	Denominator, with $a_{n+1} = \omega_n^n$
1			$s + \omega_n$
2	4.6%	6.0	$s^2 + 1.41\omega_n s + \omega_n^2$
3	2%	7.6	$s^3 + 1.75\omega_n s^2 + 2.15\omega_n^2 s + \omega_n^3$
4	1.9%	5.4	$s^4 + 2.1\omega_n s^3 + 2.4\omega_n^2 s^2 + 2.7\omega_n^3 s + \omega_n^4$
5	2.1%	6.6	$s^5 + 2.8\omega_n s^4 + 5.0\omega_n^2 s^3 + 5.5\omega_n^3 s^2 + 2.4\omega_n^4 s + \omega_n^5$
6	5%	7.8	$s^6 + 2.25\omega_n s^5 + 6.6\omega_n^2 s^4 + 8.6\omega_n^3 s^3 + 7.45\omega_n^4 s^2 + 2.95\omega_n^5 s + \omega_n^6$

The nth order standard transfer function model for the frequency ω_n can be created by the function `std_tf()`, written as

```
function G=std_tf(wn,n)
M=[1,1,0,0,0,0,0; 1,1.41,1,0,0,0,0;
   1,1.75,2.15,1,0,0,0; 1,2.1,3.4,2.7,1,0,0;
   1,2.8,5.0,5.5,3.4,1,0; 1,3.25,6.6,8.6,7.45,3.95,1];
G=tf(wn^n,M(n,1:n+1).*(wn*ones(1,n+1)).^[0:n]);
```

The function can be called by `G=std_tf(`ω_n`,`n`)`, where the value ω_n is the natural frequency, n is the order, and G is the standard transfer function obtained.

Define a matrix $\boldsymbol{E}$, where each row is $\boldsymbol{e}_i^{\mathrm{T}} = \boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{A}^{d_i}\boldsymbol{B}$, and each row $\boldsymbol{f}_i^{\mathrm{T}}$ in another matrix $\boldsymbol{F}$ can be defined as

$$\boldsymbol{f}_i^{\mathrm{T}} = \boldsymbol{c}_i^{\mathrm{T}}\Big(\boldsymbol{A}^{d_i+1} + a_{i,1}\boldsymbol{A}^{d_i} + \cdots + a_{i,d_i+1}\boldsymbol{I}\Big). \tag{5.51}$$

The state feedback matrix $\boldsymbol{K}$ and compensation matrix $\boldsymbol{\Gamma}$ can be defined as

$$\boldsymbol{K} = \boldsymbol{E}^{-1}\boldsymbol{F}, \quad \boldsymbol{\Gamma} = \boldsymbol{E}^{-1}. \tag{5.52}$$

Based on the algorithm, the new dynamic decoupling function can be written. In the function, the above-mentioned algorithm can be easily implemented.

```
function [G1,K,d,Gam]=decouple_pp(G,wn)
A=G.a; B=G.b; C=G.c; [n,m]=size(G.b); E=[]; F=[];
for j=1:m,
   for i=0:n-1
       if norm(C(j,:)*A^i*B)>eps, d(j)=i; break, end
   end,
   g1=std_tf(wn,d(j)+1); [n,cc]=tfdata(g1,'v');
   F=[F; C(j,:)*polyvalm(cc,A)]; E=[E; C(j,:)*A^d(j)*B];
end
Gam=inv(E); K=Gam*F; G1=tf(ss(A-B*K,B,C,G.d))*Gam;
```

The function can be called by `[`$\boldsymbol{G}_1$`,`$\boldsymbol{K}$`,`$\boldsymbol{d}$`,`$\boldsymbol{\Gamma}$`]=decouple_pp(`$\boldsymbol{G}$`,`ω_n`)`, where the arguments are the same as the ones in the `decouple()` function, and ω_n is the natural frequency of the standard transfer function.

Example 5.20. Consider again the multivariable model in Example 5.19. If one selects a natural frequency of $\omega_n = 5$ rad/sec, the following statements can be used:

```
>> A=[2.25, -5, -1.25, -0.5;  2.25, -4.25, -1.25, -0.25;
      0.25, -0.5, -1.25,-1;  1.25, -1.75, -0.25, -0.75];
   B=[4, 6; 2, 4; 2, 2; 0, 2]; C=[0, 0, 0, 1; 0, 2, 0, 2];
   D=zeros(2,2); G=ss(A,B,C,D); [G1,K,d,Gam]=decouple_pp(G,5)
```

The system can be decoupled satisfactorily with the decoupling method and the state feedback matrix $\boldsymbol{K}$ and precompensation matrix $\boldsymbol{\Gamma}$ can be evaluated as

$$\boldsymbol{G}_1(s) = \begin{bmatrix} \dfrac{1}{s+5} & \\ & \dfrac{1}{s+5} \end{bmatrix}, \quad \boldsymbol{K} = \frac{1}{8}\begin{bmatrix} -1 & 17 & -3 & -35 \\ 5 & -7 & -1 & 17 \end{bmatrix}, \quad \boldsymbol{\Gamma} = \begin{bmatrix} -1.5 & 0.25 \\ 0.5 & 0 \end{bmatrix}.$$

5.5 SISOTool: An Interactive Controller Design Tool

SISOTool is an interactive controller design tool provided in the Control Systems Toolbox mainly for SISO systems. SISOTool is particularly useful for classroom use due to its nice GUI (graphical user interface). In SISOTool, the root locus and Bode diagram methods are implemented for model-based controller synthesis. We shall demonstrate the use of SISOTool through an example.

Example 5.21. Consider the following open-loop plant model:

$$G(s) = \frac{1}{s(0.1s+1)(0.02s+1)(0.01s+1)(0.005s+1)}.$$

First, let us enter this model into the workspace of MATLAB and then invoke the SISOTool program using the following MATLAB statements:

```
>> s=zpk('s'); G=1/(s*(0.1*s+1)*(0.02*s+1)*(0.01*s+1)*(0.005*s+1));
   sisotool(G)
```

A user interface window appears as shown in Figure 5.29, where the root locus and the Bode diagram for the plant model are displayed.

The gain and phase margins are also displayed in the Bode diagram. The step response obtained is shown in Figure 5.31, which is fairly satisfactory with a small overshoot and a fast response speed.

Within the SISOTool interface, a toolbar is provided for the interactive controller design with several utility icons. The leftmost icon in the toolbar is the "edit" mode. After the edit icon, there are four icons for the user to add: a real pole, a real zero, a pair of

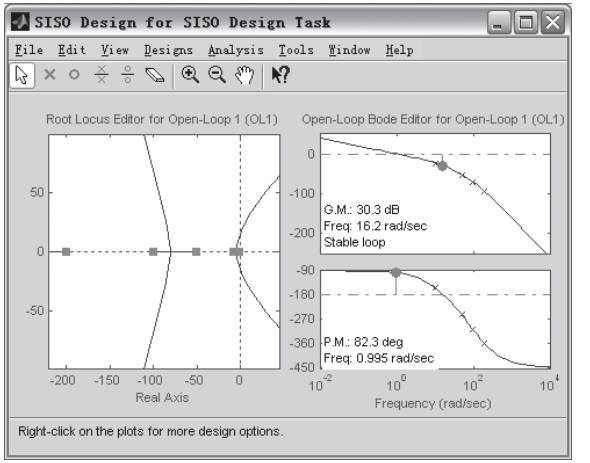


Figure 5.29. *The user interface of SISOTool.*

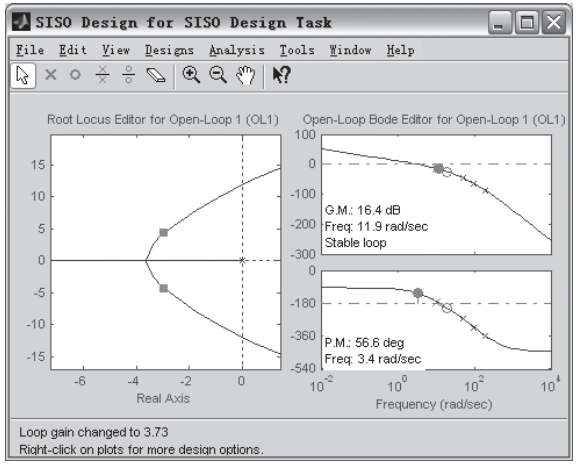


Figure 5.30. *A designed controller.*

complex poles, and a pair of complex zeros for the controller. An "eraser" icon is provided to remove the added zeros or poles. After the eraser icon, two other icons provide different zooming facilities, which are useful to the user in designing a controller.

In what follows, we use SISOTool, to present an interactive method for tuning a phase lead compensator. According to Figure 5.1(b), the phase lead controller should have a zero to the right of the pole. Then, one can use the corresponding icons to add a real zero and a real pole. It can be seen that the shape of the root locus and the Bode diagram will change automatically for the new system with a phase lead controller. Then, clicking the Analysis | Response to Step Command menu item displaying the step response of the closed-loop system in another window. For an interactive design, one should check the Real-Time Update box.

For an effective interactive design, one can place the step response window next to the main interface. Furthermore, the zooming facilities can be applied to select an area on the root locus axis to get more information around the imaginary axis. The controller design task is done simply by dragging the pole, zero, and gain into the root locus plot until a satisfactory step response is observed.

A sample design is shown in Figure 5.30. The gain and phase margins are also displayed in the Bode diagram. The step response obtained is shown in Figure 5.31, which is fairly satisfactory with a small overshoot and a fast response speed.

By interactively changing the positions of the zeros, poles, and the gain of the controller, one may have a much better understanding of the qualitative relationship between the pole and zero positions and the system step response.

Alternatively, Bode diagrams can also be used to design a cascade compensator. As an example, let us design the phase lead controller again. One can add a real zero and a real pole onto the Bode diagram. Zooming into the Bode diagram to focus on the area of interest, one can interactively tune the pole position, zero position, and the gain of the controller by

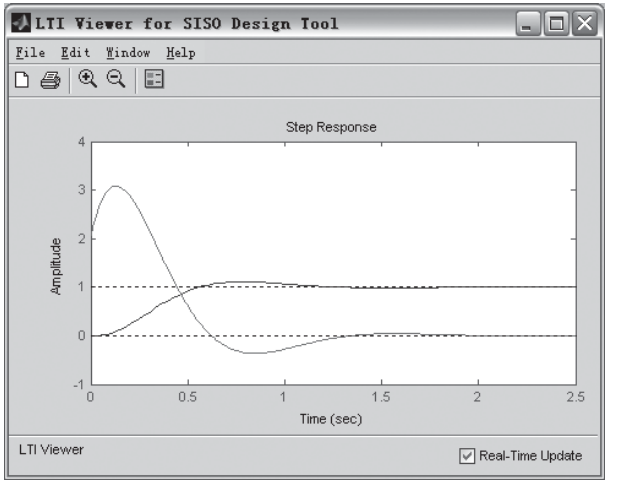


Figure 5.31. *Step response of the compensated system.*

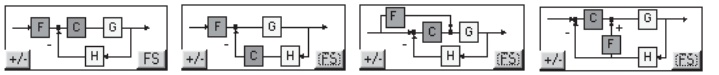


Figure 5.32. *Other supported feedback structures in SISOTool.*

dragging the magnitude response up and down. Finally, the controller can be designed with

$$C(s) = 3.728\frac{0.053s+1}{0.096s+1}.$$

Apart from the typical feedback structure, other system structures are also supported in SISOTool. Clicking the FS button on the system structure icon, one can pick up one of the feedback system structures, as shown in Figure 5.32. The rest of the controller design is the same as described in the above.

Problems

1. Consider a plant model

$$G(s) = \frac{210(s+1.5)}{(s+1.75)(s+16)(s+1.5\pm \mathrm{j}3)}$$

and its controller

$$G_c(s) = \frac{52.5(s+1.5)}{s+14.86}.$$

Investigate the closed-loop dynamic behavior. Compare the gain and phase margins of the original plant and the compensated plant. Suggest how to further improve the control performance.

2. Given the following two transfer function models:
   (a) $G(s) = \dfrac{16}{s(s+1)(s+2)(s+8)}$,
   (b) $G(s) = \dfrac{2(s+1)}{s(47.5s+1)(0.0625s+1)^2}$,
   design lead-lag compensators to ensure that the compensated systems have the expected phase margins and crossover frequencies. Try to change the expected margins to improve the performances of the closed-loop systems. Use both time and frequency domain analysis to characterize your closed-loop systems.

3. Suppose the plant model is given by

$$G(s) = \frac{1000}{s(0.26s+1)}.$$

   Design a phase lead and a phase lead-lag controller.

4. Given the state space matrices

$$\boldsymbol{A} = \begin{bmatrix} -0.2 & 0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 1.6 & 0 & 0 \\ 0 & 0 & -14.3 & 85.8 & 0 \\ 0 & 0 & 0 & -33.3 & 100 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 30 \end{bmatrix}$$

$$\boldsymbol{Q} = \text{diag}([\rho, 0, 0, 0, 0]), \quad R = 1,$$

   for different values of ρ, solve the corresponding ARE. For each ρ, design a state feedback controller and compare the closed-loop response. Comment on the comparison results with frequency domain justifications.

5. Design a state observer for the plant model given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & -1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 2 \\ 1 \\ 0 \\ 1 \end{bmatrix} u, \quad y = [0, 0, 0, 1, 1]\boldsymbol{x}.$$

   Perform a simulation analysis of the designed observer. Comment on whether the behavior of the observer is satisfactory. If not, redesign the observer until a satisfactory result is obtained.

6. For the state space model

$$\begin{cases} \dot{\boldsymbol{x}} = \begin{bmatrix} 17 & 24.54 & 1 & 8 & 15 \\ 23.54 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13.75 & 20 & 22.5889 \\ 10.8689 & 1.2900 & 19.099 & 21.896 & 3 \\ 11 & 18.089799 & 25 & 2.356 & 9 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} u \\ y = [5, 4, 3, 2, 1]\boldsymbol{x}, \end{cases},$$

   if a state feedback vector $\boldsymbol{k} = [0.0004, 0.0004, -0.0035, 0.3946, -1.4433]$ is used, find the closed-loop state space model for the system. Find all the closed-loop poles under this state feedback gain vector. Compute the controllability and observability Gramians of the open-loop and closed-loop system models.

7. Consider a state space model given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} -0.2 & 0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 1.6 & 0 & 0 \\ 0 & 0 & -14.3 & 85.8 & 0 \\ 0 & 0 & 0 & -33.3 & 100 \\ 0 & 0 & 0 & 0 & -10 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 30 \end{bmatrix} u, \quad y = [1, 0, 0, 0, 0]\boldsymbol{x}.$$

   Find the poles and zeros of the system. With the expected poles located at $\boldsymbol{P} = [-1, -2, -3, -4, -5]$, design a state feedback using the pole placement method. Try other expected pole locations to further improve the dynamic behavior of the closed-loop system. Design an observer-based controller for the original plant model and investigate the closed-loop behavior of the new closed-loop system.

8. For a given multivariable system

$$\boldsymbol{x}(t) = \begin{bmatrix} -12 & 3 & 1 & -8 & -1 \\ -1 & -12 & 0 & 7 & 4 \\ 2 & -4 & -12 & 6 & 0 \\ 5 & -2 & -6 & -16 & -12 \\ 5 & -7 & -6 & 7 & -15 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} -11 & 0 \\ -8 & -7 \\ 10 & 0 \\ 0 & 0 \\ 0 & -10 \end{bmatrix} \boldsymbol{u}(t),$$

   design a state feedback matrix $\boldsymbol{K}$ which can place the closed-loop poles at $-1 \pm \text{j}1$, -2, -3, -4.

9. For a discrete-time state space model

$$\boldsymbol{x}(k+1) = \begin{bmatrix} 0.13 & 0.154 & 0.328 & -0.046 \\ -0.126 & 0.085 & -0.232 & -0.319 \\ 0.128 & -0.376 & 0.783 & 0.134 \\ 0.318 & -0.081 & 0.117 & 0.044 \end{bmatrix} \boldsymbol{x}(k) + \begin{bmatrix} 0 \\ -1.292 \\ 0 \\ -0.331 \end{bmatrix} u(k),$$

   with $y(k) = [-0.844, 0.497, 1.488, -0.547]\boldsymbol{x}(k) - 0.8468u(k)$, design a state feedback vector $\boldsymbol{K}$ which places the closed-loop poles of the system at $-0.5 \pm \text{j}0.5$, ± 0.1. Observe the output signal in the closed-loop system.

10. Given a plant model

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} u,$$

find a state feedback gain vector $\boldsymbol{K}$ to place the closed-loop poles at $(-2, -2, -1, -1)$. Check whether it is possible to place all the poles at -2. If not, explain why.

11. For the plant models in the previous problems, design the observer-based controllers. Compare the results with the existing direct state feedback approaches and comment on the performance differences.

12. For the two input–two output system given by

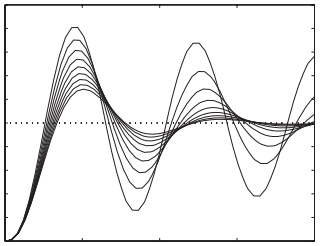
$$\begin{cases} \dot{\boldsymbol{x}}(t) \begin{bmatrix} 2.25 & -5 & -1.25 & -0.5 \\ 2.25 & -4.25 & -1.25 & -0.25 \\ 0.25 & -0.5 & -1.25 & -1 \\ 1.25 & -1.75 & -0.25 & -0.75 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 4 & 6 \\ 2 & 4 \\ 2 & 2 \\ 0 & 2 \end{bmatrix} \boldsymbol{u}(t) \\ \boldsymbol{y}(t) = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \end{bmatrix} \boldsymbol{x}(t), \end{cases}$$

assume that the weighting matrix is selected as $\boldsymbol{Q} = \text{diag}([1, 4, 3, 2])$, with $\boldsymbol{R} = \boldsymbol{I}_2$. Design an optimal LQR and observe the step response of the closed-loop system. If one wants to further improve the step response of the system, the matrix $\boldsymbol{Q}$ should be changed, for instance, with the trial-and-error method. Find a reasonable weighting matrix $\boldsymbol{Q}$ for the system.

13. Design a dynamic decoupling compensator for the multivariable model

$$\boldsymbol{G}(s) = \begin{bmatrix} \dfrac{0.806s + 0.264}{s^2 + 1.15s + 0.202} & \dfrac{-15s - 1.42}{s^3 + 12.8s^2 + 13.6s + 2.36} \\ \dfrac{1.95s^2 + 2.12s + 0.49}{s^3 + 9.15s^2 + 9.39s + 1.62} & \dfrac{7.15s^2 + 25.8s + 9.35}{s^4 + 20.8s^3 + 116.4s^2 + 111.6s + 18.8} \end{bmatrix}.$$

The integral-type decoupling and the pole placement type can be tested. In the latter type, different expected natural frequency can be tested. Find a good choice of the natural frequency.



Chapter 6

PID Controller Design

PID (proportional integral derivative) control is one of the earlier control strategies [59]. Its early implementation was in pneumatic devices, followed by vacuum and solid state analog electronics, before arriving at today's digital implementation of microprocessors. It has a simple control structure which was understood by plant operators and which they found relatively easy to tune. Since many control systems using PID control have proved satisfactory, it still has a wide range of applications in industrial control. According to a survey for process control systems conducted in 1989, more than 90 percent of the control loops were of the PID type [60]. PID control has been an active research topic for many years; see the monographs [60–64]. Since many process plants controlled by PID controllers have similar dynamics, it has been possible to set satisfactory controller parameters from less plant information than a complete mathematical model. These techniques came about because of the desire to adjust controller parameters in situ with a minimum of effort, and also because of the possible difficulty and poor cost benefit of obtaining mathematical models. The two most popular PID techniques were the step reaction curve experiment, and a closed-loop "cycling" experiment under proportional control around the nominal operating point.

In this chapter, several useful PID-type controller design techniques will be presented, and implementation issues for the algorithms will also be discussed. In Sec. 6.1, the proportional, integral, and derivative actions are explained in detail, and some variations of the typical PID structure are also introduced. In Sec. 6.2, the well-known empirical Ziegler–Nichols tuning formula and modified versions will be covered. Approaches for identifying the equivalent first-order plus dead time (FOPDT) model, which is essential in some of the PID controller design algorithms, will be presented. A modified Ziegler–Nichols algorithm is also given. Some other simple PID setting formulae, such as the Chien–Hrones–Reswick formula, Cohen–Coon formula, refined Ziegler–Nichols tuning, Wang–Juang–Chan formula and Zhuang–Atherton optimum PID controller will be presented in Sec. 6.3. In Sec. 6.4, the PID tuning formulae for FOIPDT (first-order lag and integrator plus dead time) and IPDT (integrator plus dead time) plant models, rather than the FOPDT model, will be given. A graphical user interface (GUI) implementing hundreds of PID controllers tuning formulae for FOPDT model will be given in Sec. 6.5. In Sec. 6.6, an optimization-based design algo-

rithm, together with a GUI for optimal controller design, is given. In Sec. 6.7, some of the advanced topics on PID control will be presented, such as integrator windup phenomenon and prevention, and automatic tuning techniques. Finally, some suggestions on controller structure selections for practical process control are provided.

6.1 Introduction

6.1.1 The PID Actions

A typical structure of a PID control system is shown in Figure 6.1, where it can be seen that in a PID controller, the error signal $e(t)$ is used to generate the proportional, integral, and derivative actions, with the resulting signals weighted and summed to form the control signal $u(t)$ applied to the plant model. A mathematical description of the PID controller is

$$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(\tau)\mathrm{d}\tau + T_d \frac{\mathrm{d}e(t)}{\mathrm{d}t} \right], \tag{6.1}$$

where $u(t)$ is the input signal to the plant model, the error signal $e(t)$ is defined as $e(t) = r(t) - y(t)$, and $r(t)$ is the reference input signal.

The behavior of the proportional, integral, and derivative actions will be demonstrated individually through the following example.

Example 6.1. Consider a third-order plant model given by $G(s) = 1/(s+1)^3$. If a proportional control strategy is selected, i.e., $T_i \to \infty$ and $T_d = 0$ in the PID control strategy, for different values of K_p, the closed-loop responses of the system can be obtained using the following MATLAB statements:

```
>> G=tf(1,[1,3,3,1]);
   for Kp=[0.1:0.1:1], H=feedback(Kp*G,1); step(H), hold on; end
   figure; rlocus(G,[0,15])
```

The closed-loop step responses are obtained as shown in Figure 6.2(a), and it can be seen that when K_p increases, the response speed of the system increases, the overshoot of the closed-loop system increases, and the steady-state error decreases. However, when K_p is large enough, the closed-loop system becomes unstable, which can be directly concluded from the root locus analysis in Sec. 3.4. The root locus of the example system is shown in Figure 6.2(b), where it is seen that when K_p is outside the range of $(0, 8)$, the closed-loop system becomes unstable.

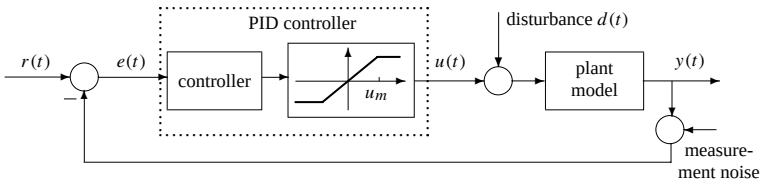
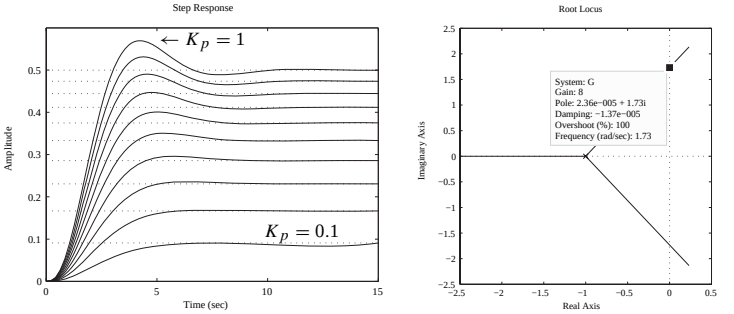
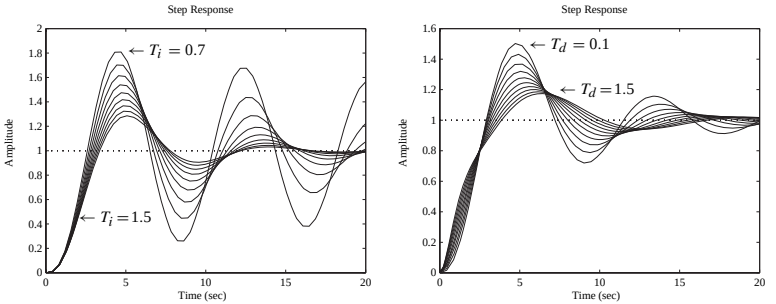


Figure 6.1. *A typical PID control structure.*



(a) closed-loop step response (b) root locus

Figure 6.2. *Closed-loop step responses.*



(a) PI control (b) PID control

Figure 6.3. *Closed-loop step responses.*

If we fix $K_p = 1$ and apply a PI (proportional plus integral) control strategy for different values of T_i, we can use the following MATLAB statements:

```
>> Kp=1; s=tf('s');
   for Ti=[0.7:0.1:1.5]
      Gc=Kp*(1+1/Ti/s); G_c=feedback(G*Gc,1); step(G_c), hold on
   end
```

to generate the closed-loop step responses of the example system shown in Figure 6.3(a). The most important feature of a PI controller is that there is no steady-state error in the step response if the closed-loop system is stable. Further examination shows that if T_i is smaller than 0.6, the closed-loop system will not be stable. It can be seen that when T_i increases, the overshoot tends to be smaller, but the speed of response tends to be slower.

Fixing both K_p and T_i at 1, i.e., $T_i = K_p = 1$, when the PID control strategy is used, with different T_d, we can use the MATLAB statements

```
>> Kp=1; Ti=1; s=tf('s');
   for Td=[0.1:0.2:2]
      Gc=Kp*(1+1/Ti/s+Td*s); step(feedback(G*Gc,1)); hold on
   end
```

to get the closed-loop step response shown in Figure 6.3(b). Clearly, when T_d increases the response has a smaller overshoot with a slightly slower rise time but similar settling time.

In practical applications, the pure derivative action is never used, due to the "derivative kick" produced in the control signal for a step input, and to the undesirable noise amplification. It is usually cascaded by a first-order low pass filter. Thus, the Laplace transformation representation of the approximate PID controller can be written as

$$U(s) = K_p \left(1 + \frac{1}{T_i s} + \frac{sT_d}{1 + s\frac{T_d}{N}}\right) E(s). \tag{6.2}$$

The effect of N is illustrated through the following example.

Example 6.2. Consider the plant model in Example 6.1. The PID controller parameters are $K_p = 1$, $T_i = 1$, and $T_d = 1$. With different selections of N, we can use the MATLAB commands

```
>> Td=1; Gc=Kp*(1+1/Ti/s+Td*s); step(feedback(G*Gc,1)), hold on
   for N=[100,1000,10000,1:10]
      Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N)); step(feedback(G*Gc,1));
   end
   figure; [y,t]=step(feedback(G*Gc,1)); err=1-y; plot(t,err)
```

to get the closed-loop step responses with the approximate derivative terms as shown in Figure 6.4(a). The error signal $e(t)$ when $N = 10$ is shown in Figure 6.4(b). It can be seen that with $N = 10$, the approximation is fairly satisfactory.

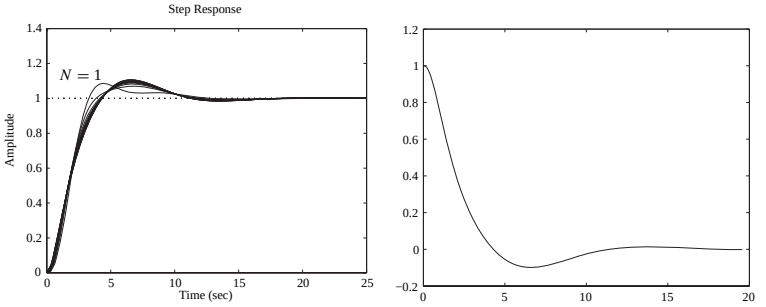
6.1.2 PID Control with Derivative in the Feedback Loop

From Figure 6.4(b), it can be seen that there exists a jump when $t = 0$ in the error signal of the step response. This means that the derivative action may not be desirable in such a control strategy.

Thus, in practice, the derivative term may be preferred in the feedback path. Since the output does not change instantaneously, for a step input, a smoother signal is produced by taking the derivative of the output. This PID control strategy, which will be denoted PI-D, is shown in Figure 6.5.

Recall the typical feedback control structure shown in Figure 1.2. The controller and feedback transfer functions can be equivalently written as

$$G_c(s) = K_p \left(1 + \frac{1}{T_i s}\right), \tag{6.3}$$



(a) output signal (b) error signal

Figure 6.4. *PID control with approximate derivatives.*

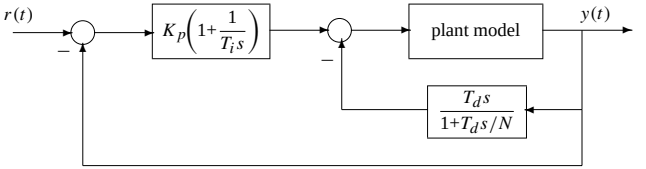


Figure 6.5. *PID control with derivative on output signal.*

$$H(s) = \frac{(1 + K_p/N)T_i T_d s^2 + K_p(T_i + T_d/N) + K_p}{K_p(T_i s + 1)(T_d s/N + 1)}. \tag{6.4}$$

The following example is designed to illustrate the consequence of using the derivative in the feedback path.

Example 6.3. For the plant model in Example 6.1, by the following MATLAB statements:

```
>> G=tf(1,[1,3,3,1]); Ti=1; Td=1; Kp=1; N=10; s=tf('s');
   Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N));
   G_c=feedback(G*Gc,1); Gc1=Kp*(1+1/s/Ti);
   H=((1+Kp/N)*Ti*Td*s^2+Kp*(Ti+Td/N)*s+Kp)/(Kp*(Ti*s+1)*(Td/N*s+1));
   G_c1=feedback(G*Gc1,H); step(G_c,G_c1)
```

the closed-loop step responses for the system with PID and PI-D are obtained and compared in Figure 6.6. By observation, the response with the PI-D controller is slower and the overshoot larger for this particular example.

6.2 Ziegler–Nichols Tuning Formula

6.2.1 Empirical Ziegler–Nichols Tuning Formula

A very useful empirical tuning formula was proposed by Ziegler and Nichols in early 1942 [10]. The tuning formula is obtained when the plant model is given by a first-order plus

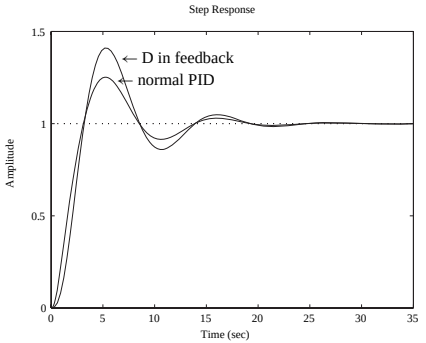


Figure 6.6. *The closed-loop step responses comparison.*

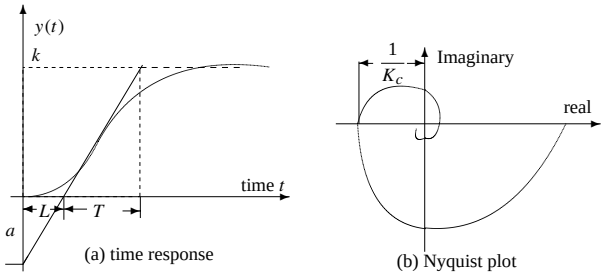


Figure 6.7. *Sketches of the responses of an FOPDT model.*

dead time (FOPDT) model expressed by

$$G(s) = \frac{k}{1+sT}\,\mathrm{e}^{-sL}. \tag{6.5}$$

In real-time process control systems, a large variety of plants can be approximately modeled by (6.5). If the system model cannot be physically derived, experiments can be performed to extract the parameters for the approximate model (6.5). For instance, if the step response of the plant model can be measured through an experiment, the output signal can be recorded as sketched in Figure 6.7(a), from which the parameters of k, L, and T (or a, where $a = kL/T$) can be extracted by the simple approach shown. More sophisticated curve fitting approaches can also be used. With L and a, the Ziegler–Nichols formula in Table 6.1 can be used to get the controller parameters.

If a frequency response experiment can be performed, the crossover frequency ω_c and the ultimate gain K_c can be obtained from the Nyquist plot as shown in Figure 6.7(b). Let $T_c = 2\pi/\omega_c$. The PID controller parameters can also be retrieved from Table 6.1. It should be noted that Table 6.1 applies for the design of P (proportional) and PI controllers in addition to the PID controller with the same set of experimental data from the plant. Since only the 180° point on the Nyquist locus is used in this approach, Ziegler and Nichols

suggested it can be found by putting the controller in the proportional mode and increasing the gain until an oscillation takes place. The point is then obtained from measurement of the gain and the oscillation frequency. This result, however, is based on linear theory, and although the technique has been used in practice, it does have major problems.

A MATLAB function `ziegler()` exists to design PI/PID controllers using the Ziegler–Nichols tuning formulas:

```
function [Gc,Kp,Ti,Td,H]=ziegler(key,vars)
Ti=[]; Td=[]; H=1;
if length(vars)==4,
   K=vars(1); L=vars(2); T=vars(3); N=vars(4); a=K*L/T;
   if key==1,  Kp=1/a;
   elseif key==2, Kp=0.9/a; Ti=3.33*L;
   elseif key==3 | key==4, Kp=1.2/a; Ti=2*L; Td=L/2; end
elseif length(vars)==3,
   K=vars(1); Tc=vars(2); N=vars(3);
   if key==1, Kp=0.5*K;
   elseif key==2, Kp=0.4*K; Ti=0.8*Tc;
   elseif key==3 | key==4, Kp=0.6*K; Ti=0.5*Tc; Td=0.12*Tc; end
elseif length(vars)==5,
   K=vars(1); Tc=vars(2); rb=vars(3); N=vars(5);
   pb=pi*vars(4)/180; Kp=K*rb*cos(pb);
   if key==2, Ti=-Tc/(2*pi*tan(pb));
   elseif key==3|key==4, Ti=Tc*(1+sin(pb))/(pi*cos(pb)); Td=Ti/4; end
end
[Gc,H]=writepid(Kp,Ti,Td,N,key);
```

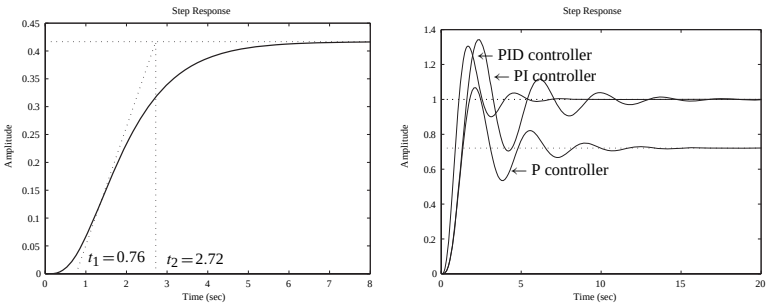
There is a low-level function `writepid()` which can be used in the design function; the content of the function is

```
function [Gc,H]=writepid(Kp,Ti,Td,N,key)
switch key
case 1, Gc=Kp;
case 2, Gc=tf(Kp*[Ti,1],[Ti,0]); H=1;
case 3, nn=[Kp*Ti*Td*(N+1)/N,Kp*(Ti+Td/N),Kp];
   dd=Ti*[Td/N,1,0]; Gc=tf(nn,dd); H=1;
case 4,  d0=sqrt(Ti*(Ti-4*Td)); Ti0=Ti; Kp=0.5*(Ti+d0)*Kp/Ti;
   Ti=0.5*(Ti+d0); Td=Ti0-Ti; Gc=tf(Kp*[Ti,1],[Ti,0]);
   nH=[(1+Kp/N)*Ti*Td, Kp*(Ti+Td/N), Kp];
   H=tf(nH,Kp*conv([Ti,1],[Td/N,1]));
case 5, Gc=tf(Kp*[Td*(N+1)/N,1],[Td/N,1]); H=1;
end
```

It seems that this function is quite lengthy for the simple Ziegler–Nichols formula given in Table 6.1. In fact, the MATLAB function also embeds a design formula discussed

Table 6.1. *Ziegler–Nichols tuning formulae.*

Controller	from step response			from frequency response		
type	K_p	T_i	T_d	K_p	T_i	T_d
P	$1/a$			$0.5K_c$		
PI	$0.9/a$	$3L$		$0.4K_c$	$0.8T_c$	
PID	$1.2/a$	$2L$	$L/2$	$0.6K_c$	$0.5T_c$	$0.12T_c$



(a) open-loop step response (b) closed-loop step response

Figure 6.8. *Controller design and responses with time domain parameters.*

later in this chapter. Here we shall consider only the syntax for the simple Ziegler–Nichols tuning rule

```
[Gc,Kp,Ti,Td]=ziegler(key,vars),
```

where `key` determines the controller type with `key` $= 1$ for the P controller, `key` $= 2$ for the PI controller, and `key` $= 3$ for the PID controller. When step response data are available, one should specify `vars` $= [K, L, T, N]$, while `vars` $= [K_c, T_c, N]$ are designed for the given frequency response data.

Example 6.4. Consider a fourth-order plant

$$G(s) = \frac{10}{(s+1)(s+2)(s+3)(s+4)}.$$

Enter the following MATLAB statements:

```
>> s=tf('s'); G=10/(s+1)/(s+2)/(s+3)/(s+4);
   step(G); k=dcgain(G)
```

The open-loop step response is shown in Figure 6.8(a), with a steady-state value of 0.4167. From the step response, the parameters of the approximate FOPDT model are $k = 0.2941$, $L = 0.76$, and $T = 2.72 - 0.76 = 1.96$, based on which the P, PI, and PID controllers can be designed using the following MATLAB statements:

```
>> L=0.76; T=2.72-L; [Gc1,Kp1]=ziegler(1,[k,L,T,10])
   [Gc2,Kp2,Ti2]=ziegler(2,[k,L,T,10])
   [Gc3,Kp3,Ti3,Td3]=ziegler(3,[k,L,T,10])
```

The P, PI, and PID controllers designed are, respectively,

$$G_{\rm p}(s) = 6.1895,\ G_{\rm PI}(s) = 5.57\left(1 + \frac{1}{2.5308s}\right),\ G_{\rm PID}(s) = 7.4274\left(1 + \frac{1}{1.52s} + 0.38s\right).$$

The closed-loop responses for these different controllers are obtained using the MATLAB statements

```
>> G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1);
   G_c3=feedback(G*Gc3,1); step(G_c1,G_c2,G_c3);
```

and they are shown in Figure 6.8(b). It can be observed that the steady-state error exists when the P controller is used, and the response of the PID controller is faster than that of the PI controller.

If the frequency response of the plant model can be measured, the ultimate gain K_c and the crossover frequency ω_c can be read from the Nyquist plot as shown in Figure 6.7(b). With K_c and ω_c, the parameters of different PID-type controllers can be obtained from Table 6.1. In this case, the MATLAB function `ziegler()` can still be used.

In fact, since the crossover frequency ω_c and the ultimate gain K_c are the gain margin of the open-loop plant model, one can directly obtain the parameters using the `margin()` function.

Example 6.5. Consider the plant model in Example 6.4. By the MATLAB statements

```
>> G=tf(10, [1,10,35,50,24]);
   nyquist(G); axis([-0.2,0.6,-0.4,0.4])
   [Kc,pp,wg,wp]=margin(G); [Kc,wg], Tc=2*pi/wg;
   [Gc1,Kp1]=ziegler(1,[Kc,Tc,10]); Kp1
   [Gc2,Kp2,Ti2]=ziegler(2,[Kc,Tc,10]); [Kp2,Ti2]
   [Gc3,Kp3,Ti3,Td3]=ziegler(3,[Kc,Tc,10]); [Kp3,Ti3,Td3]
```

the gain margin and its crossover frequency are found to be, respectively, 12.6, and 2.2361 rad/sec. The controllers are designed as

$$G_{\rm p}(s) = 6.3,\quad G_{\rm PI}(s) = 5.04\left(1 + \frac{1}{2.2479s}\right),\quad G_{\rm PID}(s) = 7.56\left(1 + \frac{1}{1.405s} + 0.3372s\right).$$

The Nyquist plot of the system can be obtained and is shown in Figure 6.9(a). With these different controllers, the closed-loop system responses can be obtained using the MATLAB statements

```
>> G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1);
   G_c3=feedback(G*Gc3,1); step(G_c1,G_c2,G_c3);
```

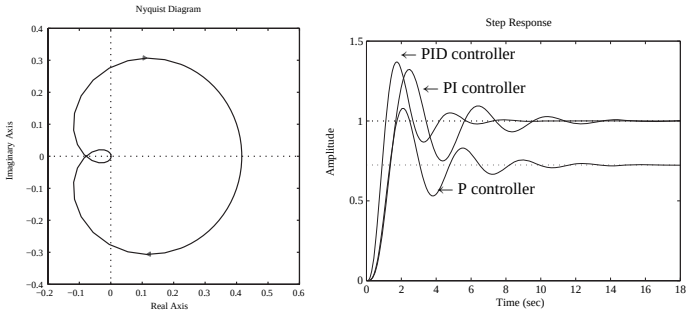
and the step responses of the closed-loop system are shown in Figure 6.9(b).

6.2.2 Derivative Action in the Feedback Path

Assume that the derivative action is placed in the feedback path; then the normal PID parameters (K_p, T_i, T_d) can be obtained from [65] as

$$K_p = K_p'\left(1 + \frac{T_d'}{T_i'}\right),\quad T_i = T_i' + T_d',\quad T_d = \frac{T_i' T_d'}{T_i' + T_d'}, \tag{6.6}$$

where (K_p', T_i', T_d') are the PID parameters with derivative in the feedback path.



(a) Nyquist plots (b) closed-loop step response

Figure 6.9. *Controller design and responses.*

In other words, if a PID controller, with derivative action in a forward path, is designed, then an equivalent PID controller with the derivative action in the feedback path can be obtained by solving the following algebraic equation:

$$x^2 - T_i x + T_i T_d = 0, \quad \Rightarrow \quad x_{1,2} = \frac{T_i \pm \sqrt{T_i(T_i - 4T_d)}}{2}. \tag{6.7}$$

It is reasonable to assume in most PID controller designs that $T_i > 4T_d$. In this case, the above equation will have real roots $x_{1,2}$. Thus, from (K_p, T_i, T_d), the equivalent PID parameters for the new structure, i.e., with derivative in the feedback path, can be computed as follows:

$$\begin{aligned} T_i' &= \frac{T_i + \sqrt{T_i(T_i - 4T_d)}}{2}, \quad T_d' = \frac{T_i - \sqrt{T_i(T_i - 4T_d)}}{2}, \\ K_p' &= \frac{2T_i K_p}{T_i + \sqrt{T_i(T_i - 4T_d)}}. \end{aligned} \tag{6.8}$$

The MATLAB function `ziegler()` can still be used to design such a PID controller. The syntax of the function now becomes

```
[Gc,Kp,Ti,Td,H]=ziegler(key,vars)
```

with `key` $= 4$ and H is the equivalent feedback transfer function object.

Example 6.6. Consider the plant model in Example 6.4. The normal PID controller can be designed using the Ziegler–Nichols algorithm. An effective design of a PID controller with a derivative in the feedback path can also be obtained with the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); N=10; [Kc,Pm,wc,wp]=margin(G);
   Tc=2*pi/wc; [Gc1,Kp1,Ti1,Td1]=ziegler(3,[Kc,Tc,N]),
   [Gc2,Kp2,Ti2,Td2,H]=ziegler(4,[Kc,Tc,N]),
   G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,H);
   step(G_c1,G_c2)
```

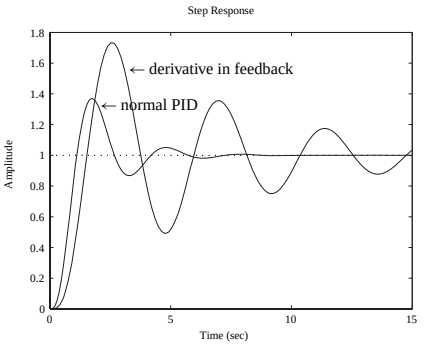



Figure 6.10. *PID controllers comparison.*

The controllers designed are $G_{\text{PID}}(s) = 7.5600\,(1 + 1/1.4050s + 0.3372s)$, with $K_p' = 4.5360$, $T_i' = 0.8430$, $T_d' = 0.5620$, and the step response comparison is shown in Figure 6.10(a).

It can be seen that although the PID controller with derivative in the feedback path might be easier and faster to be implemented compared to the normal PID controller, its performance may not be very satisfactory. Sometimes, such a PID controller should be designed using a dedicated algorithm to ensure a good control performance.

6.2.3 Methods for First-Order Plus Dead Time Model Fitting

It can be seen that the model (6.5) is useful for designing a PID controller because of the availability of a simple formula. The method in Sec. 6.2.1 for finding L and T of a given plant is simple to use with the graph of a plant step response. Although in modern computation it is not necessary to reduce a model to this form to find suitable PID controller parameters, which may be found by using the original model with one of many possible approaches, nevertheless it can be useful. Given the plant transfer function, we can use one of the model reduction methods described in Chapter 3. For example, the suboptimal reduction method [47] is very effective at the expense of a heavy, but acceptable, computational load. The optimal reduced-order model can be obtained with the function `opt_app()`, covered in Sec. 3.6. In this section, two other effective and frequently used algorithms will be introduced.

Frequency response method

Consider the frequency response of a first-order model

$$G(\mathrm{j}\omega) = \frac{k}{Ts+1}\mathrm{e}^{-Ls}\Big|_{s=\mathrm{j}\omega} = \frac{k}{T\mathrm{j}\omega+1}\mathrm{e}^{-\mathrm{j}\omega L}. \tag{6.9}$$

The ultimate gain K_c at the crossover frequency ω_c is actually the first intersection of a Nyquist plot with the negative part of the real axis, i.e.,

$$\begin{cases} \dfrac{k(\cos\omega_c L - \omega_c T \sin\omega_c L)}{1+\omega_c^2 T^2} = -\dfrac{1}{K_c}, \\ \sin\omega_c L + \omega_c T \cos\omega_c L = 0, \end{cases} \tag{6.10}$$

where k is the steady-state value or DC (direct current) gain of the system which can be directly evaluated from the given transfer function. Define two variables $x_1 = L$ and $x_2 = T$ satisfying

$$\begin{cases} f_1(x_1, x_2) = kK_c(\cos\omega_c x_1 - \omega_c x_2 \sin\omega_c x_1) + 1 + \omega_c^2 x_2^2 = 0, \\ f_2(x_1, x_2) = \sin\omega_c x_1 + \omega_c x_2 \cos\omega_c x_1 = 0. \end{cases} \tag{6.11}$$

The Jacobian matrix is that

$$\begin{aligned} \boldsymbol{J} &= \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \end{bmatrix} \\ &= \begin{bmatrix} -kK_c\omega_c \sin\omega_c x_1 - kK_c\omega_c^2 x_2 \cos\omega_c x_1 & 2\omega_c^2 x_2 - kK_c\omega_c \sin\omega_c x_1 \\ \omega_c \cos\omega_c x_1 - \omega_c^2 x_2 \sin\omega_c x_1 & \omega_c \cos\omega_c x_1 \end{bmatrix}. \end{aligned} \tag{6.12}$$

So, (x_1, x_2) can be solved using any quasi-Newton algorithm. The MATLAB function `[K,L,T]= getfod(G)` is written for solving x_1 and x_2 in order to find the parameters K, L, T of the system.

```
function [K,L,T]=getfod(G,method)
K=dcgain(G);
if nargin==1
   [Kc,Pm,wc,wcp]=margin(G); ikey=0; L=1.6*pi/(3*wc); T=0.5*Kc*K*L;
   if finite(Kc), x0=[L;T];
      while ikey==0, u=wc*x0(1); v=wc*x0(2);
         FF=[K*Kc*(cos(u)-v*sin(u))+1+v^2; sin(u)+v*cos(u)];
         J=[-K*Kc*wc*sin(u)-K*Kc*wc*v*cos(u), -K*Kc*wc*sin(u)+2*wc*v;
            wc*cos(u)-wc*v*sin(u), wc*cos(u)];
         x1=x0-inv(J)*FF;
         if norm(x1-x0)<1e-8, ikey=1; else, x0=x1;
      end, end
      L=x0(1); T=x0(2);
   end
elseif nargin==2 & method==1
   [n1,d1]=tfderv(G.num{1},G.den{1}); [n2,d2]=tfderv(n1,d1);
   K1=dcgain(n1,d1); K2=dcgain(n2,d2);
   Tar=-K1/K; T=sqrt(K2/K-Tar^2); L=Tar-T;
end
function [e,f]=tfderv(b,a)
f=conv(a,a); na=length(a); nb=length(b);
e1=conv((nb-1:-1:1).*b(1:end-1),a);
e2=conv((na-1:-1:1).* a(1:end-1),b); maxL=max(length(e1),length(e2));
e=[zeros(1,maxL-length(e1)) e1]-[zeros(1,maxL-length(e2)) e2];
```


Transfer function method

Consider the first-order model with delay given by

$$G_n(s) = \frac{k\mathrm{e}^{-Ls}}{Ts+1}.$$

Taking the first- and second-order derivatives of $G_n(s)$ with respect to s, one can immediate find that

$$\frac{G_n'(s)}{G_n(s)} = -L - \frac{T}{1+Ts}, \quad \frac{G_n''(s)}{G_n(s)} - \left(\frac{G_n'(s)}{G_n(s)}\right)^2 = \frac{T^2}{(1+Ts)^2}.$$

Evaluating the values at $s = 0$ yields

$$T_{\mathrm{ar}} = -\frac{G_n'(0)}{G_n(0)} = L + T, \quad T^2 = \frac{G_n''(0)}{G_n(0)} - T_{\mathrm{ar}}^2, \tag{6.13}$$

where T_{ar} is also referred to as the average residence time. From the former equation, one has $L = T_{\mathrm{ar}} - T$. Again, the DC gain k can be evaluated from $G_n(0)$.

The solution for the FOPDT model is thus obtained by using the derivatives of its transfer function $G(s)$ in the above formula.

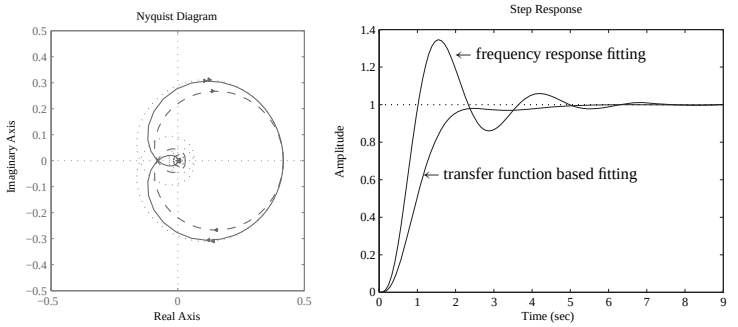
The MATLAB function `getfod()` listed earlier can be used with the syntax `[K,L,T]= getfod(G,1)` to find the parameters K, L, T of the system.

Example 6.7. Consider the fourth-order model used in Example 6.4. The parameters of its approximate FOPDT model can be obtained using the MATLAB statements

```
>> G=tf(10,[1,10,35,50,24]);
   [k,L,T]=getfod(G); G1=tf(k,[T 1]); G1.ioDelay=L;
   [Gc1,Kp3,Ti3,Td3]=ziegler(3,[k,L,T,10])
   [k,L,T]=getfod(G,1), G2=tf(k,[T 1]); G2.ioDelay=L;
   nyquist(G,'-',G1,'--',G2,':'); figure
   [Gc2,Kp4,Ti4,Td4]=ziegler(3,[k,L,T,10])
   G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1); step(G_c1,G_c2)
```

The Nyquist plot comparisons of the plant model and the two approximations are shown in Figure 6.11(a).

With the frequency response method, the K, L, T parameters are obtained as 0.4167, 0.7882, 2.3049. The PID controller designed with the Ziegler–Nichols formulas is $G_{c1}(s) = 8.4219\,(1 + 1/1.5764s + 0.3941s)$. While the parameters using the transfer function method are 0.4167, 0.8902, 1.1932, the PID controller is $G_{c2}(s) = 3.8602(1 + 1/1.7804s + 0.4451s)$. The closed-loop step responses with the above two PID controllers are shown in Figure 6.11(b).



(a) Nyquist plots (b) closed-loop step responses

Figure 6.11. *PID controller responses.*

It can be seen that although the PID controller designed with the transfer function identification algorithm looks better, it does not reflect the usual overshoot characteristics of Ziegler–Nichols tuning, presumably due to the inaccurately identified parameters of an FOPDT model.

With the use of the suboptimal model reduction technique presented in Sec. 3.6.3, the parameters can be extracted with the following statements and the controller can better be designed:

```
Gr=opt_app(G,0,1,1); [n,d]=tfdata(G,'v');
K=dcgain(G); T=d(1)/d(2); L=Gr.ioDelay;
```

6.2.4 A Modified Ziegler–Nichols Formula

Consider the Nyquist frequency response shown in Figure 6.12(a), where for a selected point A on the Nyquist plot, the control effects of the P, I, and D terms are shown in the appropriate directions. Thus, with properly chosen K_p, T_i, and T_d, it is possible to move the given point A on the Nyquist curve of the uncontrolled plant to an arbitrary position on the Nyquist plot of the controlled system. The typical Nyquist plot under PID control is shown in Figure 6.12(b), where A_1 corresponds to the point A in Figure 6.12(a).

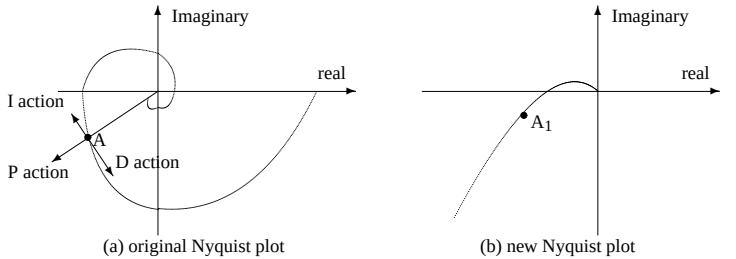
Denote point A in the complex plane as $G(j\omega_0) = r_a e^{j(\pi+\phi_a)}$. Suppose A is to be moved to A_1 which is represented by $G_1(j\omega_0) = r_b\, e^{j(\pi+\phi_b)}$. Assume that the PID controller at frequency ω_0 is $G_c(s) = r_c e^{j\phi_c}$. Then, obviously,

$$r_b e^{j(\pi+\phi_b)} = r_a r_c e^{j(\pi+\phi_a+\phi_c)}. \tag{6.14}$$

Therefore, $r_c = r_b/r_a$ and $\phi_c = \phi_b - \phi_a$. So, based on the above analysis, PI and PID controllers can be designed as follows:

- *PI control*: The controller can be designed such that

$$K_p = \frac{r_b\cos(\phi_b-\phi_a)}{r_a},\quad T_i = \frac{1}{\omega_0\tan(\phi_a-\phi_b)}, \tag{6.15}$$



(a) original Nyquist plot (b) new Nyquist plot

Figure 6.12. *Sketches of FOPDT model.*

which means that $\phi_a > \phi_b$ for a positive T_i.

As a special case, the Ziegler–Nichols algorithm design is by

$$K_p = K_c r_b\cos\phi_b,\quad T_i = -\frac{T_c}{2\pi\tan\phi_b}, \tag{6.16}$$

where $T_c = 2\pi/\omega_c$, $r_a = 1/K_c$, and $\phi_a = 0$.

- *PID control*: The controller can be designed such that

$$K_p = \frac{r_b\cos(\phi_b-\phi_a)}{r_a},\quad \omega_0 T_d - \frac{1}{\omega_0 T_i} = \tan(\phi_b-\phi_a). \tag{6.17}$$

Clearly, T_i and T_d are not unique according to (6.17). To get a unique PID design, it is a usual practice to set $T_d = \alpha T_i$, where α is a constant. Given an α, T_i, and T_d can be obtained uniquely from

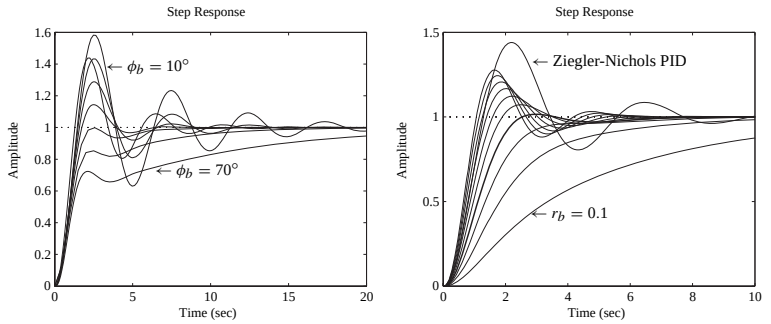
$$T_i = \frac{1}{2\alpha\omega_0}\left(\tan(\phi_b-\phi_a)+\sqrt{4\alpha+\tan^2(\phi_b-\phi_a)}\right),\ T_d = \alpha T_i. \tag{6.18}$$

By inspection, it is seen that the Ziegler–Nichols tuning formula is a special case when $\alpha = 1/4$. The Ziegler–Nichols tuning formula can be rewritten as follows:

$$K_p = K_c r_b\cos\phi_b,\ T_i = \frac{T_c}{\pi}\left(\frac{1+\sin\phi_b}{\cos\phi_b}\right),\ T_d = \frac{T_c}{4\pi}\left(\frac{1+\sin\phi_b}{\cos\phi_b}\right), \tag{6.19}$$

where $r_a = 1/K_c$, $\phi_a = 0$, and $\alpha = 1/4$.

It can be seen that the PI and PID controllers can be designed by a suitable choice of r_b and ϕ_b. The design problem is then one of selecting suitable values for these two parameters to give the appropriate performance. This is called a modified Ziegler–Nichols PI/PID tuning formula, which has been implemented in the MATLAB function `ziegler()`, too. The only difference is that `vars` $= [K_c, T_c, r_b, \phi_b, N]$.



(a) for different ϕ_b (b) for different r_b

Figure 6.13. *Closed-loop step responses.*

Example 6.8. Consider the plant model given by $G(s) = 1/(s+1)^3$. The PID controller by the original Ziegler–Nichols tuning method can be obtained as follows:

```
>> G=tf(1,[1,3,3,1]); [Kc,pp,wg,wp]=margin(G); Tc=2*pi/wg;
   [Gc1,Kp1,Ti1,Td1]=ziegler(3,[Kc,Tc,10])
```

and the controller $G(s) = 4.8007\,(1 + 1/1.8137s + 0.4353s)$ is obtained. Now, let us illustrate the flexibility of the modified Ziegler–Nichols PI/PID tuning formula. First, fix $r_b = 0.5$ and change ϕ_b. By the following MATLAB statements:

```
>> G_c=feedback(G*Gc1,1); step(G_c,20); rb=0.5; hold on
   for pb=[10:10:70]
      [Gc2,Kp2,Ti2,Td2]=ziegler(3,[Kc,Tc,rb,pb,10]);
      G_c2=feedback(G*Gc2,1); step(G_c2,20);
   end
```

the closed-loop step responses of the system for different values of ϕ_b are shown in Figure 6.13(a). Clearly, when ϕ_b increases, the overshoot and oscillation become smaller. When ϕ_b is larger than 60°, there is no overshoot, but the response becomes too sluggish. A good choice for the phase angle based on these responses is approximately 45°.

Now, fix ϕ_b at $\phi_b = 45°$ and change r_b. By the MATLAB statements

```
>> G_c=feedback(G*Gc1,1); step(G_c,10); pb=45; hold on;
   for rb=[0.1:0.1:1]
      [Gc2,Kp2,Ti2,Td2]=ziegler(3,[Kc,Tc,rb,pb,10]);
      G_c2=feedback(G*Gc2,1); step(G_c2,10);
   end
```

the closed-loop step responses of the system for different r_b are compared in Figure 6.13(b). It can be seen that the smaller the r_b, the smaller the overshoot and the slower the response. Clearly, $r_b = 0.45$, and $\phi_b = 45°$ can be considered as a good choice for this example with almost no overshoot and with a reasonably fast response.

It can be concluded that the modified tuning method is advantageous over the original Ziegler–Nichols PI/PID tuning technique.

6.3 Other PID Controller Tuning Formulae

Many variants of the traditional Ziegler–Nichols PID tuning methods have been proposed. Several of these are given in the following section.

6.3.1 Chien–Hrones–Reswick PID Tuning Algorithm

The Chien–Hrones–Reswick (CHR) method [66] emphasizes the set-point regulation or disturbance rejection. In addition one qualitative specifications on the response speed and overshoot can be accommodated. Compared with the traditional Ziegler–Nichols tuning formula, the CHR method uses the time constant T of the plant explicitly.

The CHR PID controller tuning formulas are summarized in Table 6.2 for set-point regulation. The more heavily damped closed-loop response, which ensures, for the ideal plant model, the "quickest response without overshoot" is labeled "with 0% overshoot," and the "quickest response with 20% overshoot" is labeled "with 20% overshoot."

Similarly, Table 6.3 is used to design controllers for disturbance rejection purposes.

A MATLAB function `chrPID()` is written which can be used to design different controllers using the CHR algorithms:

```
function [Gc,Kp,Ti,Td,H]=chrpid(key,tt,vars)
K=vars(1); L=vars(2); T=vars(3); N=vars(4); a=K*L/T; Ti=[]; Td=[];
ovshoot=vars(5); if tt==1, TT=T; else TT=L; tt=2;  end
if ovshoot==0,
   KK=[0.3,0.35,1.2,0.6,1,0.5; 0.3,0.6,4,0.95,2.4,0.42];
else,
   KK=[0.7,0.6,1,0.95,1.4,0.47; 0.7,0.7,2.3,1.2,2,0.42];
end
switch key
case 1, Kp=KK(tt,1)/a;
case 2, Kp=KK(tt,2)/a; Ti=KK(tt,3)*TT;
case {3,4}, Kp=KK(tt,4)/a; Ti=KK(tt,5)*TT; Td=KK(tt,6)*L;
end
[Gc,H]=writepid(Kp,Ti,Td,N,key);
```

Table 6.2. *CHR tuning formulae for set-point regulation.*

Controller	with 0% overshoot			with 20% overshoot		
type	K_p	T_i	T_d	K_p	T_i	T_d
P	$0.3/a$			$0.7/a$		
PI	$0.35/a$	$1.2T$		$0.6/a$	T	
PID	$0.6/a$	T	$0.5L$	$0.95/a$	$1.4T$	$0.47L$

Table 6.3. *CHR tuning formulae for disturbance rejection.*

Controller	with 0% overshoot			with 20% overshoot		
type	K_p	T_i	T_d	K_p	T_i	T_d
P	$0.3/a$			$0.7/a$		
PI	$0.6/a$	$4L$		$0.7/a$	$2.3L$	
PID	$0.95/a$	$2.4L$	$0.42L$	$1.2/a$	$2L$	$0.42L$

The syntax of the `chrpid()` function is

```
[Gc, Kp, Ti, Td]=chrPID(key,typ,vars)
```

where the returned variables are defined similar to those in `ziegler()`. `key` = 1, 2, 3 is for P, PI, and PID controllers, respectively. The variable `typ` denotes the type of criteria used with `typ` = 1 for set-point control and any other value for disturbance rejection. `vars` = [k, L, T, N, O_s] with $O_s = 0$ denotes no overshoot, and any other value denotes 20% overshoot.

Example 6.9. Consider the plant model in Example 6.4. The Ziegler–Nichols PID controller and the four CHR controllers for different controller types and specifications are obtained using the following statements:

```
>> s=tf('s'); G=10/((s+1)*(s+2)*(s+3)*(s+4)); N=10;
   [k,L,T]=getfod(G); [Gc1,Kp,Ti,Td]=ziegler(3,[k,L,T,N])
   [Gc2,Kp,Ti,Td]=chrpid(3,1,[k,L,T,N,0])
   [Gc3,Kp,Ti,Td]=chrpid(3,1,[k,L,T,N,20])
   [Gc4,Kp,Ti,Td]=chrpid(3,2,[k,L,T,N,0]);
```

The four PID controllers designed are, respectively,

$$G_1(s) = 8.4219\left(1+\frac{1}{1.5764s}+0.3941s\right),\ G_2(s) = 4.2110\left(1+\frac{1}{2.3049s}+0.3941s\right),$$

$$G_3(s) = 6.6674\left(1+\frac{1}{3.2268s}+0.3704s\right),\ G_4(s) = 6.6674\left(1+\frac{1}{1.8917s}+0.3310s\right).$$

For the different controllers designed in the above, the step responses of the closed-loop systems can be obtained using the following MATLAB statements:

```
>> step(feedback(G*Gc1,1),feedback(G*Gc2,1),
        feedback(G*Gc3,1),...
        feedback(G*Gc4,1),10)
```

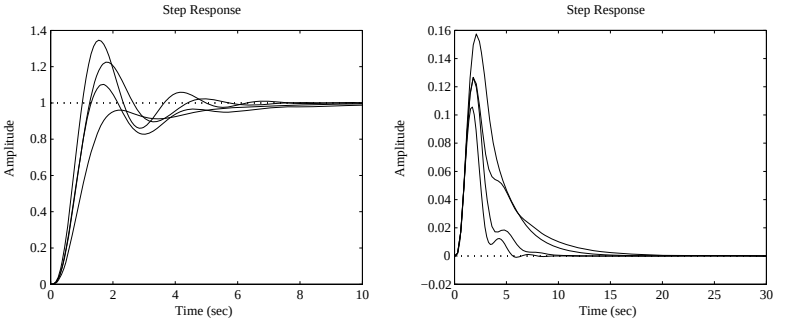
as summarized in Figure 6.14(a). It can be seen that the set-point regulation controller with 0% overshoot gives a satisfactory result. Similarly, with the following MATLAB statements:

```
>> step(feedback(G,Gc1),feedback(G,Gc2),feedback(G,Gc3),...
        feedback(G,Gc4),30)
```

the closed-loop responses to a step disturbance signal can be obtained as shown in Figure 6.14(b). Clearly, compared with the traditional Ziegler–Nichols controller, the effect of the disturbance signal can be significantly reduced by a CHR controller.

6.3.2 Cohen–Coon Tuning Algorithm

Another Ziegler–Nichols type tuning algorithm is the Cohen–Coon tuning formula [67]. Referring to the FOPDT model (6.5) approximately obtained from experiments, denote



(a) set-point step response (b) disturbance step response

Figure 6.14. *Closed-loop step responses of CHR controllers.*

$a = kL/T$ and $\tau = L/(L+T)$. The different controllers can be designed by the direct use of Table 6.4.

A MATLAB function `cohenpid()` is written which can be used to design a PID controller using the Cohen–Coon tuning formulas:

```
function [Gc,Kp,Ti,Td,H]=cohenpid(key,vars)
K=vars(1); L=vars(2); T=vars(3); N=vars(4);
a=K*L/T; tau=L/(L+T); Ti=[]; Td=[];
switch key
case 1,Kp=(1+0.35*tau/(1-tau))/a;
case 2,
   Kp=0.9*(1+0.92*tau/(1-tau))/a; Ti=(3.3-3*tau)*L/(1+1.2*tau);
case {3,4}, Kp=1.35*(1+0.18*tau/(1-tau))/a;
   Ti=(2.5-2*tau)*L/(1-0.39*tau); Td=0.37*(1-tau)*L/(1-0.81*tau);
case 5
   Kp=1.24*(1+0.13*tau/(1-tau))/a; Td=(0.27-0.36*tau)*L/(1-0.87*tau);
end
[Gc,H]=writepid(Kp,Ti,Td,N,key);
```

The syntax is `[Gc, Kp, Ti, Td, H]=cohenpid(key,vars)`, where the `vars` arguments should be written as `vars` = [k, L, T, N].

Table 6.4. *Controller parameters of Cohen–Coon method.*

Controller	K_p	T_i	T_d
P	$\frac{1}{a}\left(1+\frac{0.35\tau}{1-\tau}\right)$		
PI	$\frac{0.9}{a}\left(1+\frac{0.92\tau}{1-\tau}\right)$	$\frac{3.3-3\tau}{1+1.2\tau}L$	
PD	$\frac{1.24}{a}\left(1+\frac{0.13\tau}{1-\tau}\right)$		$\frac{0.27-0.36\tau}{1-0.87\tau}L$
PID	$\frac{1.35}{a}\left(1+\frac{0.18\tau}{1-\tau}\right)$	$\frac{2.5-2\tau}{1-0.39\tau}L$	$\frac{0.37-0.37\tau}{1-0.81\tau}L$

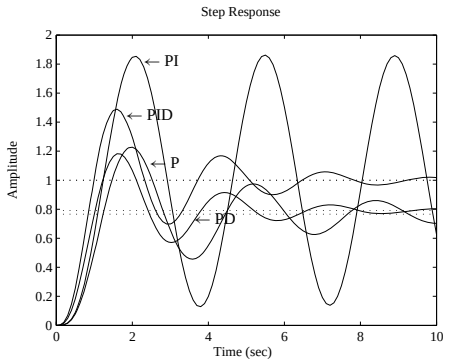


Figure 6.15. *Step responses under controllers of the Cohen–Coon method.*

Example 6.10. Consider the plant model given in Example 6.4 with its P, PI, PD, and PID controllers designed using the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); [k,L,T]=getfod(G);
   [Gc1,Kp1]=cohenpid(1,[k,L,T,10])
   [Gc2,Kp2,Ti2]=cohenpid(2,[k,L,T,10])
   [Gc3,Kp3,Ti3,Td3]=cohenpid(5,[k,L,T,10])
   [Gc4,Kp4,Ti4,Td4]=cohenpid(3,[k,L,T,10])
```

and the controllers are obtained as

$$G_1(s) = 7.8583,\ G_2(s) = 8.3036\,(1 + 1/1.5305s)\,,$$
$$G_3(s) = 9.0895(1 + 0.1805s),\ G_4(s) = 10.0579\,(1 + 1/1.7419s + 0.2738s)\,.$$

With the following MATLAB statements:

```
>> G_c1=feedback(G*Gc1,1); G_c2=feedback(G*Gc2,1);
   G_c3=feedback(G*Gc3,1); G_c4=feedback(G*Gc4,1);
   step(G_c1,G_c2,G_c3,G_c4,10)
```

the closed-loop step responses of the systems with the different controllers are shown in Figure 6.15.

6.3.3 Refined Ziegler–Nichols Tuning

Since the PID controller designed by the conventional Ziegler–Nichols tuning formulas often exhibits rather strong oscillation in the set-point response and a large overshoot, a refinement to such a PID controller tuning algorithm can be obtained with the use of set-point weighting [68]:

$$u(t) = K_p\left[(\beta u_c - y) + \frac{1}{T_i}\int e\mathrm{d}t - T_d\frac{\mathrm{d}y}{\mathrm{d}t}\right], \tag{6.20}$$

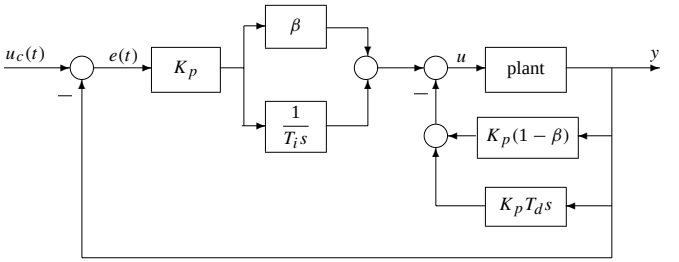


Figure 6.16. *Refined PID control structure.*

where the derivative action is performed on the output signal and a fraction of the input signal is added to the control signal. Usually, $\beta < 1$. The control law can be rewritten as

$$u(t) = K_p\left(\beta e + \frac{1}{T_i}\int e\mathrm{d}t\right) - K_p\left[(1-\beta)y + T_d\frac{\mathrm{d}y}{\mathrm{d}t}\right]. \tag{6.21}$$

The block diagram representation of the control system can be constructed as shown in Figure 6.16. Compared with the typical feedback control structure shown in Figure 1.2, after some transfer function block manipulations, the controller $G_c(s)$ and the feedback $H(s)$ can be easily obtained as follows:

$$G_c(s) = K_p\left(\beta + \frac{1}{T_i s}\right), \tag{6.22}$$

$$H(s) = \frac{T_i T_d \beta(N+2-\beta)s^2/N + (T_i + T_d/N)s + 1}{(T_i\beta s + 1)(T_d s/N + 1)}. \tag{6.23}$$

Define the normalized delay constant τ as $\tau = L/T$ and a constant κ by $\kappa = K_c k$. For different ranges of the variables τ and κ, PID controller parameters are suggested as follows:

- If $2.25 < \kappa < 15$ or $0.16 < \tau < 0.57$, use the original Ziegler–Nichols design parameters. To ensure that the overshoot is less than 10% or less than 20%, β should be evaluated, respectively, from

$$\beta = \frac{15-\kappa}{15+\kappa} \ \text{ or } \ \beta = \frac{36}{27+5\kappa}. \tag{6.24}$$

- If $1.5 < \kappa < 2.25$ or $0.57 < \tau < 0.96$, the integral parameter T_i in the Ziegler–Nichols controller should be changed to $T_i = 0.5\mu T_c$, where

$$\mu = \frac{4}{9}\kappa \ \text{ and } \ \beta = \frac{8}{17}(\mu - 1). \tag{6.25}$$

- If $1.2 < \kappa < 1.5$, in order to keep the overshoot less than 10%, the parameters of the PID should be refined as

$$K_p = \frac{5}{6}\left(\frac{12+\kappa}{15+14\kappa}\right), \quad T_i = \frac{1}{5}\left(\frac{4}{15}\kappa + 1\right). \tag{6.26}$$

A MATLAB function `rziegler()` is written which can be used to design a refined PID controller:

```
function [Gc,Kp,Ti,Td,beta,H]=rziegler(vars)
K=vars(1); L=vars(2); T=vars(3); N=vars(4); a=K*L/T; Kp=1.2/a;
Ti=2*L; Td=L/2; Kc=vars(5); Tc=vars(6); kappa=Kc*K; tau=L/T; H=[];
if (kappa > 2.25 & kappa<15) | (tau>0.16 & tau<0.57)
   beta=(15-kappa)/(15+kappa);
elseif (kappa<2.25 & kappa>1.5) | (tau<0.96 & tau>0.57)
   mu=4*jappa/9; beta=8*(mu-1)/17; Ti=0.5*mu*Tc;
elseif (kappa>1.2 & kappa<1.5),
   Kp=5*(12+kappa)/(6*(15+14*kappa)); Ti=0.2*(4*kappa/15+1); beta=1;
end
Gc=tf(Kp*[beta*Ti,1],[Ti,0]); nH=[Ti*Td*beta*(N+2-beta)/N,Ti+Td/N,1];
dH=conv([Ti*beta,1],[Td/N,1]); H=tf(nH,dH);
```

The syntax of the function is `[`G_c`,`K_p`,`T_i`,`T_d`,`β`,`H`]=rziegler(vars)`, where `vars` $= [k, L, T, N, K_c, T_c]$.

Example 6.11. Consider the plant model in Example 6.4. The refined PID controller can be designed using the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); [k,L,T]=getfod(G);
   [Kc,p,wc,m]=margin(G);  Tc=2*pi/wc;
   [Gc,Kp,Ti,Td,beta,H]=rziegler([k,L,T,10,Kc,Tc])
   [Gc1,Kp1,Ti1,Td1]=ziegler(3,[k,L,T,10]);
   G_c=feedback(G*Gc,H); G_c1=feedback(G*Gc1,1);
   step(G_c,G_c1);
```

The parameters of the refined PID controller should be taken as $K_p = 8.4219, T_i = 1.5764, T_d = 0.3941, \beta = 0.4815$. The closed-loop step responses under the refined Ziegler–Nichols PID controller are shown in Figure 6.17, with a comparison to the response from the conventional Ziegler–Nichols PID controller. The response is significantly improved but not as good as the responses using other tuning algorithms such as the modified Ziegler–Nichols method with $r_b = 0.45$, and $\phi_b = 45°$.

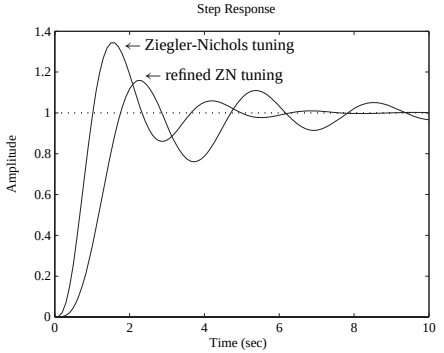


Figure 6.17. *Step responses under the original and the refined Ziegler–Nichols controllers.*

6.3.4 The Wang–Juang–Chan Tuning Formula

Based on the optimum ITAE criterion, the tuning algorithm proposed by Wang, Juang, and Chan [69] is a simple and efficient method for selecting the PID parameters. If the k, L, T parameters of the plant model are known, the controller parameters are given by

$$K_p = \frac{(0.7303 + 0.5307T/L)(T + 0.5L)}{K(T + L)}, \qquad T_i = T + 0.5L, \quad T_d = \frac{0.5LT}{T + 0.5L}. \tag{6.27}$$

A MATLAB function `wjcpid()` is written for the PID controller design, using the Wang–Juang–Chan tuning formula:

```
function [Gc,Kp,Ti,Td]=wjcpid(vars)
K=vars(1); L=vars(2); T=vars(3); N=vars(4); Td=0.5*L*T/(T+0.5*L);
Kp=(0.7303+0.5307*T/L)*(T+0.5*L)/(K*(T+L)); Ti=T+0.5*L;
s=tf('s'); Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N));
```

where `vars` $= [k, L, T, N]$.

6.3.5 Optimum PID Controller Design

Optimum setting algorithms for a PID controller were proposed by Zhuang and Atherton [70] for various criteria. Consider the general form of the optimum criterion

$$J_n(\theta) = \int_0^\infty [t^n e(\theta, t)]^2 \mathrm{d}t, \tag{6.28}$$

where $e(\theta, t)$ is the error signal which enters the PID controller, with θ the PID controller parameter vector. For the system structure shown in Figure 6.1, two parameter setting strategies are proposed: one for the set-point input and the other for the disturbance signal $d(t)$. In particular, three values of n are discussed, i.e., for $n = 0, 1, 2$. These three cases correspond, respectively, to three different optimum criteria: the integral squared error (ISE) criterion, integral squared time weighted error (ISTE) criterion, and the integral squared time-squared weighted error (IST^{2}E) criterion [65]. The expressions given were obtained by fitting curves to the optimum theoretical results.

Set-point optimum PID tuning

If the plant can be represented by the FOPDT model in (6.5), the typical PI controller can be empirically represented as

$$K_p = \frac{a_1}{k}\left(\frac{L}{T}\right)^{b_1}, \quad T_i = \frac{T}{a_2 + b_2(L/T)}, \tag{6.29}$$

where the (a, b) pairs can be obtained from Table 6.5. When the first-order approximation to the plant model can be obtained, the PI controller can be designed easily by the direct use of Table 6.5 and (6.29).

Table 6.5. *Set-point PI controller parameters.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	0.980	0.712	0.569	1.072	0.786	0.628
b_1	−0.892	−0.921	−0.951	−0.560	−0.559	−0.583
a_2	0.690	0.968	1.023	0.648	0.883	1.007
b_2	−0.155	−0.247	−0.179	−0.114	−0.158	−0.167

Table 6.6. *Set-point PID controller parameters.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1.048	1.042	0.968	1.154	1.142	1.061
b_1	−0.897	−0.897	−0.904	−0.567	−0.579	−0.583
a_2	1.195	0.987	0.977	1.047	0.919	0.892
b_2	−0.368	−0.238	−0.253	−0.220	−0.172	−0.165
a_3	0.489	0.385	0.316	0.490	0.384	0.315
b_3	0.888	0.906	0.892	0.708	0.839	0.832

Table 6.7. *Set-point PID controller parameters with D in feedback path.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1.260	1.053	0.942	1.295	1.120	1.001
b_1	−0.887	−0.930	−0.933	−0.619	−0.625	−0.624
a_2	0.701	0.736	0.770	0.661	0.720	0.754
b_2	−0.147	−0.126	−0.130	−0.110	−0.114	−0.116
a_3	0.375	0.349	0.308	0.378	0.350	0.308
b_3	0.886	0.907	0.897	0.756	0.811	0.813

For the PID controller, its gains can be set as follows:

$$K_p = \frac{a_1}{k}\left(\frac{L}{T}\right)^{b_1}, \quad T_i = \frac{T}{a_2 + b_2(L/T)}, \quad T_d = a_3 T\left(\frac{L}{T}\right)^{b_3}, \tag{6.30}$$

where for different ratios L/T, the coefficients (a, b) are defined in Table 6.6.

To include the derivative action in the output signal, the corresponding PI-D is given by

$$U(s) = K_p\left(1 + \frac{1}{T_i s}\right)E(s) - \frac{sT_d}{1 + sT_d/N}Y(s), \tag{6.31}$$

where the parameters (a, b) should be determined according to Table 6.7.

Disturbance rejection PID tuning

Sometimes one may want to design disturbance rejection PID controllers, i.e., to design a controller having a good rejection performance on the disturbance signal $d(t)$. The parameters of the PI controller should be set as

$$K_p = \frac{a_1}{T}\left(\frac{L}{T}\right)^{b_1}, \quad T_i = \frac{T}{a_2}\left(\frac{L}{T}\right)^{b_2}, \tag{6.32}$$

where the parameters (a, b) are obtained directly from Table 6.8.

Furthermore, for the PID controller,

$$K_p = \frac{a_1}{T}\left(\frac{L}{T}\right)^{b_1}, \quad T_i = \frac{T}{a_2}\left(\frac{L}{T}\right)^{b_2}, \quad T_d = a_3 T\left(\frac{L}{T}\right)^{b_3}, \tag{6.33}$$

and the (a, b) parameters are determined from Table 6.9.

A MATLAB function `optpid()` is written which can be used to get the parameters of the PID controller:

```
function [Gc,Kp,Ti,Td,H]=optPID(key,typ,vars)
k=vars(1); L=vars(2); T=vars(3); N=vars(4); Td=[];
if length(vars)==5, iC=vars(5);
   switch key
   case 2
A=[0.980,0.712,0.569,1.072,0.786,0.628; 0.892,0.921,0.951,0.560,0.559,0.583;
   0.690,0.968,1.023,0.648,0.883,1.007; 0.155,0.247,0.179,0.114,0.158,0.167];
   case 3
A=[1.048,1.042,0.968,1.154,1.142,1.061; 0.897,0.897,0.904,0.567,0.579,0.583;
```

Table 6.8. *Disturbance rejection PI controller parameters.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1.279	1.015	1.021	1.346	1.065	1.076
b_1	−0.945	−0.957	−0.953	−0.675	−0.673	−0.648
a_2	0.535	0.667	0.629	0.552	0.687	0.650
b_2	0.586	0.552	0.546	0.438	0.427	0.442

Table 6.9. *Disturbance rejection PID controller parameters.*

Range of L/T	$0.1-1$			$1.1-2$		
Criterion	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1.473	1.468	1.531	1.524	1.515	1.592
b_1	−0.970	−0.970	−0.960	−0.735	−0.730	−0.705
a_2	1.115	0.942	0.971	1.130	0.957	0.957
b_2	0.753	0.725	0.746	0.641	0.598	0.597
a_3	0.550	0.443	0.413	0.552	0.444	0.414
b_3	0.948	0.939	0.933	0.851	0.847	0.850

```
   1.195,0.987,0.977,1.047,0.919,0.892; 0.368,0.238,0.253,0.220,0.172,0.165;
   0.489,0.385,0.316,0.490,0.384,0.315; 0.888,0.906,0.892,0.708,0.839,0.832];
   case 4
A=[1.260,1.053,0.942,1.295,1.120,1.001; 0.887,0.930,0.933,0.619,0.625,0.624;
   0.701,0.736,0.770,0.661,0.720,0.754; 0.147,0.126,0.130,0.110,0.114,0.116;
   0.375,0.349,0.308,0.378,0.350,0.308; 0.886,0.907,0.897,0.756,0.811,0.813];
   end
   ii=0; if (L/T>1) ii=3; end; tt=L/T; a1=A(1,ii+iC); b1=-A(2,ii+iC);
   a2=A(3,ii+iC); b2=-A(4,ii+iC); Kp=a1/k*tt^b1; Ti=T/(a2+b2*tt);
   if key==3| key==4
      a3=A(5,ii+iC); b3=A(6,ii+iC); Td=a3*T*tt^b3;
   end
else,
   Kc=vars(5); Tc=vars(6); k=vars(7);
   switch key
   case 2, Kp=0.361*Kc;Ti=0.083*(1.935*k+1)*Tc;
   case 3, Kp=0.509*Kc; Td=0.125*Tc; Ti=0.051*(3.302*k+1)*Tc;
   case 4, Kp=(4.437*k-1.587)/(8.024*k-1.435)*Kc;
      Ti=0.037*(5.89*k+1)*Tc; Td=0.112*Tc;
   end
end
[Gc,H]=writepid(Kp,Ti,Td,N,key);
```

The syntax of the function is

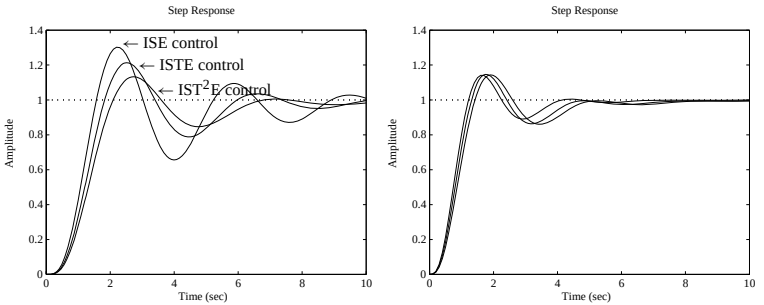
```
[Gc, Kp, Ti, Td, H]=optpid(key,typ,vars)
```

where `key` $= 2, 3, 4$ for PI, normal PID, and PID controllers with D in the feedback path, respectively, and `typ` $= 1, 2$ for set-point and disturbance rejection, respectively. The variable `vars` $= [k, L, T, N, C]$, where C is the criterion type with $C = 1, 2, 3$ for ISE, ISTE, and IST^{2}E criteria, respectively. The returned variables are G_c, the cascade controller object, and K_p,T_i,T_d are the PID controller parameters. H is returned, if `key` $= 4$, as the equivalent feedback transfer function for the structure with the derivative in the feedback path.

Example 6.12. Consider the plant model in Example 6.4. The optimal PI and PID controllers can be designed using the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); N=10; [k,L,T]=getfod(G);
   f1=figure; f2=figure;
   for iC=1:3
      [Gc,Kp,Ti,Td]=optpid(2,1,[k,L,T,N,iC]);
      figure(f1),G_c=feedback(G*Gc,1); step(G_c,10),hold on,
      [Gc,Kp,Ti,Td]=optpid(3,1,[k,L,T,N,iC]);
      figure(f2),G_c=feedback(G*Gc,1); step(G_c,10),hold on,
   end
```

The relevant closed-loop step responses are shown in Figures 6.18(a) and (b).



(a) PI control (b) PID control

Figure 6.18. *Closed-loop step responses of optimal controllers.*

PID controller design based on ultimate frequency and gain

When the crossover frequency ω_c and the ultimate gain K_c are known, with $T_c = 2\pi/\omega_c$, three types of PID controllers are summarized in Table 6.10, where $\kappa = kK_c$ is the normalized gain of the plant model [70]. The values given were deduced from the relationship between the FOPDT plant parameters and the ultimate gain and frequency.

The corresponding values for the PI controller are given in Table 6.11.

When the relay automatic tuning strategy is used, which will be discussed later in this chapter, the oscillation frequency ω_0 and the magnitude a_0 can be measured. Then, $T_0 = 2\pi/\omega_0$ and $K_0 = 4h/(a_0\pi)$. Assume that $\kappa_0 = kK_0$. ω_0 and K_0 are approximations to ω_c and K_c, but more accurate results can be obtained for the PID controller parameters from Table 6.12.

The PI controllers for disturbance rejection can also be obtained with the direct use of Table 6.13.

Table 6.10. *PID controller parameters for ISTE criterion.*

PID	Set-point	Disturbance rejection	D in feedback
K_p	$0.509K_c$	$\dfrac{4.434\kappa - 0.966}{5.12\kappa + 1.734}K_c$	$\dfrac{4.437\kappa - 1.587}{8.024\kappa - 1.435}K_c$
T_i	$0.051(3.302\kappa + 1)T_c$	$\dfrac{1.751\kappa - 0.612}{3.776\kappa + 1.388}T_c$	$0.037(5.89\kappa + 1)T_c$
T_d	$0.125T_c$	$0.144T_c$	$0.112T_c$

Table 6.11. *PI controller parameters for ISTE criterion.*

PI	Set-point	Disturbance rejection
K_p	$\dfrac{4.264 - 0.148\kappa}{12.119 - 0.432\kappa}K_c$	$\dfrac{1.892\kappa + 0.244}{3.249\kappa + 2.097}K_c$
T_i	$0.083(1.935\kappa + 1)T_c$	$\dfrac{0.706\kappa - 0.227}{0.7229\kappa + 1.2736}T_c$

Table 6.12. *PID controller parameters for ISTE criterion for autotuning.*

PID	Set-point	Disturbance rejection	D on output
K_p	$0.604K_0$	$\frac{6.068\kappa_0 - 4.273}{5.758\kappa_0 - 1.058}K_0$	$\frac{2.354\kappa_0 - 0.696}{3.363\kappa_0 + 0.517}K_0$
T_i	$0.04(4.972\kappa_0 + 1)T_0$	$\frac{1.1622\kappa_0 - 0.748}{2.516\kappa_0 - 0.505}T_0$	$0.271\kappa_0 T_0$
T_d	$0.130T_0$	$0.15T_0c$	$0.1162T_0c$

Table 6.13. *PI controller parameters for ISTE criterion for autotuning.*

PI	Set-point	Disturbance rejection
K_p	$\frac{1.506\kappa_0 - 0.177}{3.341\kappa_0 + 0.606}K_0$	$\frac{6.068\kappa_0 - 4.273}{5.758\kappa_0 - 1.058}K_0$
T_i	$0.055(3.616\kappa_0 + 1)T_0$	$\frac{5.352\kappa_0 - 2.926}{5.539\kappa_0 + 5.536}T_0$

Improved gain-phase approach

The gain-phase assignment algorithm can be used to design a PID controller

$$K_p = \frac{m\cos\phi}{|\,G(\mathrm{j}\omega_c)\,|} = mK_c\cos\phi,\ T_d = \frac{\tan\phi + \sqrt{4/\alpha + \tan^2\phi}}{2\omega_c},\ T_i = \alpha T_d, \tag{6.34}$$

where $\alpha = 0.413(3.302\kappa + 1)$ or $\alpha = 1.687\kappa_0$. The constants ϕ and m can be obtained from one of the following two cases:

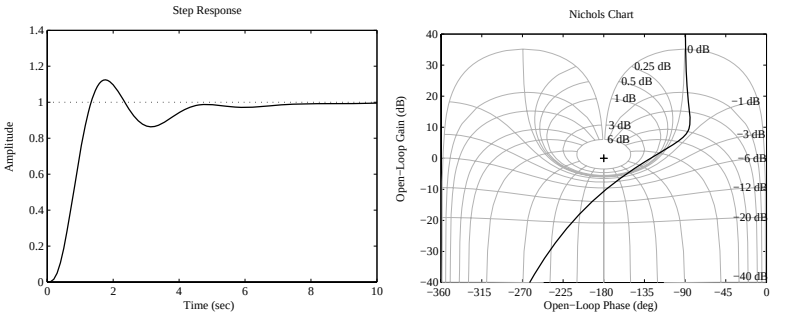
- For the normalized gain κ,

$$\phi = 33.8^\circ(1 - 0.97\mathrm{e}^{-0.45\kappa}),\ \ m = 0.614(1 - 0.233\mathrm{e}^{-0.347\kappa}). \tag{6.35}$$

- If the frequency and the gain under automatic tuning are measured, the following approach can be used:

$$\phi = 33.2^\circ(1 - 1.38\,\mathrm{e}^{-0.68\kappa_0}),\ \ m = 0.613(1 - 0.262\,\mathrm{e}^{-0.44\kappa_0}). \tag{6.36}$$

The MATLAB function `optpid()` can be used again to solve for the PID controller parameters with the improved gain-phase method. The syntax of the function, for the particular design tasks with this algorithm, is `[Gc,Kp,Ti,Td,H]=optpid(key,typ,vars)` where `vars` $= [k, L, T, N, K_c, T_c, \kappa]$ are the relevant parameters of the plant model. As before, if the value of `key` is selected as `key` $= 4$, the effective PID controller, with derivative action in the feedback path, can be designed.



(a) step responses (b) Nichols charts

Figure 6.19. *Responses for the optimal gain-phase margins design.*

Example 6.13. Consider again the plant model in Example 6.4. The PID controller can be designed using the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); [Kc,pm,wc,wm]=margin(G);
   Tc=2*pi/wc; kappa=dcgain(G)*Kc; [k,L,T]=getfod(G);
   N=10; vars=[k,L,T,N,Kc,Tc,kappa];
   [Gc,Kp,Ti,Td,H]=optpid(3,1,vars); step(feedback(G*Gc,1));
   figure,nichols(G*Gc);grid;axis([-360,0,-40,40])
```

the controller is

$$G_c(s) = 6.4134\left(1 + \frac{2.6276}{s} + 0.3512s\right).$$

The closed-loop step response and the Nichols chart of the system are obtained as shown in Figures 6.19(a) and (b), respectively. It can be seen that the responses are satisfactory, compared with the controllers designed using other approaches.

Example 6.14. Let us revisit the original Ziegler–Nichols tuning algorithm. We have seen in Sec. 6.2 that the original Ziegler–Nichols parameter setting formula does not achieve a very satisfactory PID control performance. In this example, we will show, via redesigning the PID controller for the plant model in Example 6.4, a new Ziegler–Nichols parameter setting procedure can give a much improved performance which is close to that achieved by the optimum PID parameter setting method.

Before applying the original Ziegler–Nichols parameter setting formula, the optimal reduced-order model is obtained first to extract the characteristics of the plant model. Then, with this optimally reduced FOPDT model, a PID controller can be designed using the Ziegler–Nichols algorithm. By the following MATLAB statements:

```
>> G=tf(10,[1,10,35,50,24]); R=opt_app(G,0,1,1); L=R.ioDelay;
   T=R.den{1}(1)/R.den{1}(2); K=R.num{1}(2)/R.den{1}(2);
   Gc=ziegler(3,[K,L,T,10]); Gc1=optpid(3,1,[K,L,T,10,2]);
   step(feedback(G*Gc,1),feedback(G*Gc1,1))
```

the new Ziegler–Nichols PID controller and the optimum PID controller can be designed. Their step responses are compared in Figure 6.20. We can see that the new Ziegler–Nichols

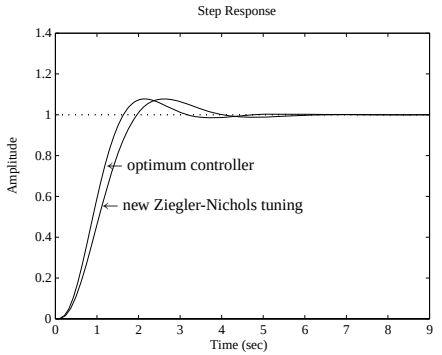


Figure 6.20. *Step responses comparison of two PID controllers.*

parameter setting procedure gives a much improved performance compared with that presented in Example 6.4. In fact, this new Ziegler–Nichols PID controller performs similarly to the optimum PID controller in terms of step response speed and overshoot.

6.4 PID Controller Tuning Algorithms for Other Types of Plants

All the PID tuning algorithms discussed in the previous sections are based on the FOPDT plant models; they cannot be used for many other plant models in practice. A great many PID tuning algorithms have been collected in the handbook [71], where apart from the FOPDT-based algorithms, tuning algorithms for other plant models are also given. Here only a few PID controller algorithms are summarized, with their MATLAB implementations.

6.4.1 PD and PID Parameter Setting for IPDT Models

A widely encountered plant model is described by a mathematical description $G(s) = K\mathrm{e}^{-Ls}/s$, which is referred to as the integrator plus dead time (IPDT) model. This kind of plant model cannot be controlled by the PD and PID controllers using the setting algorithms given in the previous sections.

Since there already exists an integrator in the plant model, an extra integrator in the controller is not required to remove a steady-state error to a step input, but it is needed to remove the output error caused by a steady disturbance at the plant input. PD controllers may also be used to avoid large overshoot. The mathematical models of PD and PID controllers are, respectively,

$$G_{\mathrm{PD}}(s) = K_p(1 + T_d s), \quad G_{\mathrm{PID}}(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right). \tag{6.37}$$

Table 6.14. *The coefficients of the controller for IPDT models.*

criterion	a_1	a_2	a_3	a_4	a_5
ISE	1.03	0.49	1.37	1.49	0.59
ITSE	0.96	0.45	1.36	1.66	0.53
ISTSE	0.9	0.45	1.34	1.83	0.49

PD and PID parameter setting algorithms were presented in [72], based on various performance indices, and given as

$$\begin{aligned} &\text{PD controller } \ K_p = \frac{a_1}{KL}, \ \ T_d = a_2 L, \\ &\text{PID controller } \ K_p = \frac{a_3}{KL}, \ \ T_i = a_4 L, \ \ T_d = a_5 L, \end{aligned} \tag{6.38}$$

where for different criteria, the coefficients a_i can be selected as shown in Table 6.14. The following MATLAB function can be written to implement the above algorithms:

```
function [Gc,Kp,Ti,Td]=ipdtctrl(key,key1,K,L,N)
a=[1.03,0.49,1.37,1.49,0.59; 0.96,0.45,1.36,1.66,0.53;
    0.9,0.45,1.34,1.83,0.49]; s=tf('s'); Ti=inf;
if key==1
   Kp=a(key1,1)/K/L; Td=a(key1,2)*L; Gc=Kp*(1+Td*s/(1+Td/N*s));
else
   Kp=a(key1,3)/K/L; Ti=a(key1,4)*L; Td=a(key1,5)*L;
   Gc=Kp*(1+1/Ti/s+Td*s/(1+Td/N*s));
end
```

In the function, `key` is the switch for PD and PID controller selections, with `key` $= 1$ for PD, 2 for PID. The argument for `key1` is set to 1, 2, 3 for ISE, ITAE, and ISTSE selections, respectively.

6.4.2 PD and PID Parameters for FOIPDT Models

Another category of plant model is defined by a first-order lag and integrator plus dead time (FOIPDT) whose mathematical model is

$$G(s) = \frac{K\mathrm{e}^{-Ls}}{s(Ts+1)}.$$

Since an integrator is contained in the model, an extra integrator is not necessary in the controller to remove the steady-state error to a set point change. Thus, a PD controller may be used if there is no steady-state disturbance at the plant. A PD controller setting algorithm is included in [71, 73]:

$$K_p = \frac{2}{3KL}, \quad T_d = T. \tag{6.39}$$

Also a PID setting algorithm is included in [71, 74] such that

$$K_p = \frac{1.111T}{KL^2}\frac{1}{\left[1+(T/L)^{0.65}\right]^2}, \ T_i = 2L\left[1+\left(\frac{T}{L}\right)^{0.65}\right], \ \ T_d = \frac{T_i}{4}. \tag{6.40}$$

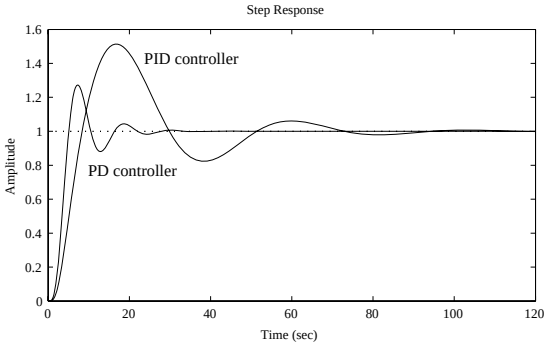


Figure 6.21. *Comparisons of the PID and PD controllers.*

A control design function `foipdt()` is written to implement the two algorithms, where `key` is used to select the structure of the controller, i.e., 1 for PD and 2 for PID. If the parameters K, L, T, N are known, the controller can immediately be designed.

```
function [Gc,Kp,Ti,Td]=foipdt(key,K,L,T,N)
s=tf('s');
if key==1
   Kp=2/3/K/L; Td=T; Ti=inf; Gc=Kp*(1+Td*s/(1+Td*s/N));
else
   a=(T/L)^0.65; Kp=1.111*T/(K*L^2)/(1+a)^2;
   Ti=2*L*(1+a); Td=Ti/4; Gc=Kp*(1+1/Ti/s+Td*s/(1+Td*s/N));
end
```

Example 6.15. Consider the plant model

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

where there exists an integrator and the rest of the model can be described by an FOPDT model. Thus, the original model can be approximated by an FOIPDT model. The following statements can be used to design PD and PID controllers. The step response of the closed-loop systems are obtained as shown in Figure 6.21.

```
>> s=tf('s'); G1=1/(s+1)^4; G=G1/s; R=opt_app(G1,0,1,1);
   K=R.num{1}(2)/R.den{1}(2); L=R.ioDelay; T=1/R.den{1}(2);
   [Gc1,Kp1,Ti1,Td1]=foipdt(1,K,L,T,10);
   [Gc2,Kp2,Ti2,Td2]=foipdt(2,K,L,T,10);
   step(feedback(G*Gc1,1),feedback(G*Gc2,1))
```

The controllers are

$$G_{\rm PD}(s) = 0.3631\left(1+\frac{2.3334s}{1+0.23334s}\right), \quad G_{\rm PID}(s) = 0.1635\left(1+\frac{1}{7.9638s}+\frac{1.9910s}{1+0.1991s}\right).$$

It can be seen from the control results that the PD controller is significantly better than the PID controller. This is because the 180° lag given by two integrators makes good control more difficult.

Table 6.15. *The coefficients of the controller for unstable FOPDT models.*

Criterion	a_1	b_1	a_2	b_2	a_3	b_3	γ
ISE	1.32	0.92	4	0.47	3.78	0.84	0.95
ITSE	1.38	0.9	4.12	0.9	3.62	0.85	0.93
ISTSE	1.35	0.95	4.52	1.13	3.7	0.86	0.97

6.4.3 PID Parameter Settings for Unstable FOPDT Models

In practical control systems, the plant model may approximate an unstable FOPDT model, i.e.,

$$G(s) = \frac{K\mathrm{e}^{-Ls}}{Ts-1}.$$

The following algorithms may be used to design the PID controller, [72].

$$K_p = \frac{a_1}{K}A^{b_1}, \quad T_i = a_2 T A^{b_2}, \quad T_d = a_3 T\left[1-b_3 A^{-0.02}\right]A^{\gamma}, \tag{6.41}$$

where $A = L/T$. For different criteria, the coefficients a_i, b_i, γ of the PID controller can be obtained in Table 6.15. Based on the algorithm, a PID controller design function for unstable FOPDT models can be written such that

```
function [Gc,Kp,Ti,Td]=ufopdt(key,K,L,T,N)
Tab=[1.32, 0.92, 4.00, 0.47, 3.78, 0.84, 0.95;
     1.38, 0.90, 4.12, 0.90, 3.62, 0.85, 0.93;
     1.35, 0.95, 4.52, 1.13, 3.70, 0.86, 0.97];
a1=Tab(key,1); b1=Tab(key,2); a2=Tab(key,3); b2=Tab(key,4);
a3=Tab(key,5); b3=Tab(key,6); gam=Tab(key,7); A=L/T;
Kp=a1*A^b1/K; Ti=a2*T*A^b2; Td=a3*T*(1-b3*A^(-0.02))*A^gam;
s=tf('s'); Gc=Kp*(1+1/Ti/s+Td*s/(1+Td/N*s));
```

6.5 PID_Tuner: A PID Controller Design Program for FOPDT Models

Hundreds of PID parameter tuning algorithms have been collected in the handbook [71]. Many of the methods are based on the FOPDT plant models. Thus, a GUI is designed, which can be used to design PID-type controllers, and also a closed-loop simulation for the designed controllers can be obtained. With the interface, the following procedures can be used to design PID controllers:

1. Enter `pid_tuner` under the MATLAB prompt. The interface in Figure 6.22 is given, which can be used to design PID-type controllers.
2. Click the Plant model button; a dialog box will be given to prompt you to enter the plant model. Any single input–single output (SISO) continuous model, with or without time delays, can be defined. The button Modify Plant Model can be used to modify the plant models.
3. Once the plant model is specified, the Get FOPDT parameters button can be clicked to extract the FOPDT parameters, i.e., to find the parameters K, L, T. Many different

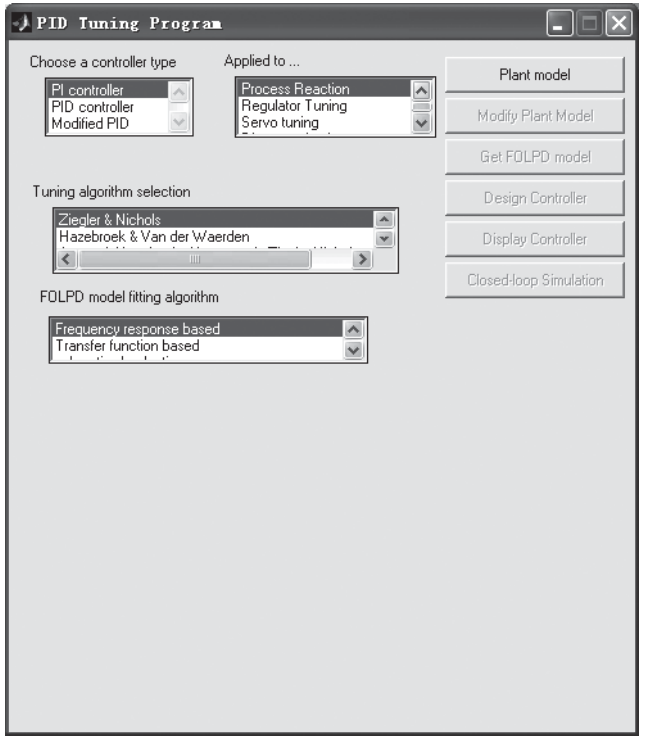


Figure 6.22. *PID controller design interface.*

methods can be used to extract the parameters, for instance, using the optimum fitting methods. The fitting algorithms can be selected via the FOPDT model parameters fitting list box.

4. With the K, L, T parameters, the controller can be designed. The controller type can be selected by the combinations of the list boxes Choose controller type, Apply to, and Tuning algorithm selection, which provides the algorithms in [75].
5. The Design Controller button can be used to design the relevant PID controller.
6. The Closed-loop Simulation button can be used to show the closed-loop step response of the system under the controllers designed.

Example 6.16. For the plant model

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

click Plant model to enter the model. The dialog box shown in Figure 6.23 is displayed, and the numerator, denominator, coefficient vectors, and delay constant can be entered. Then click the Apply button to model the input procedure.

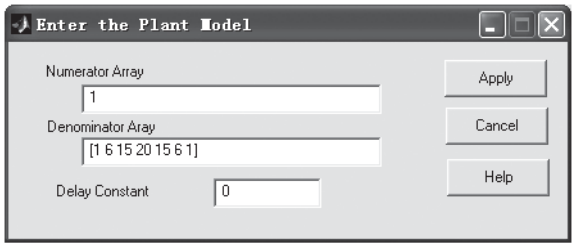


Figure 6.23. *Dialog box of plant model input.*

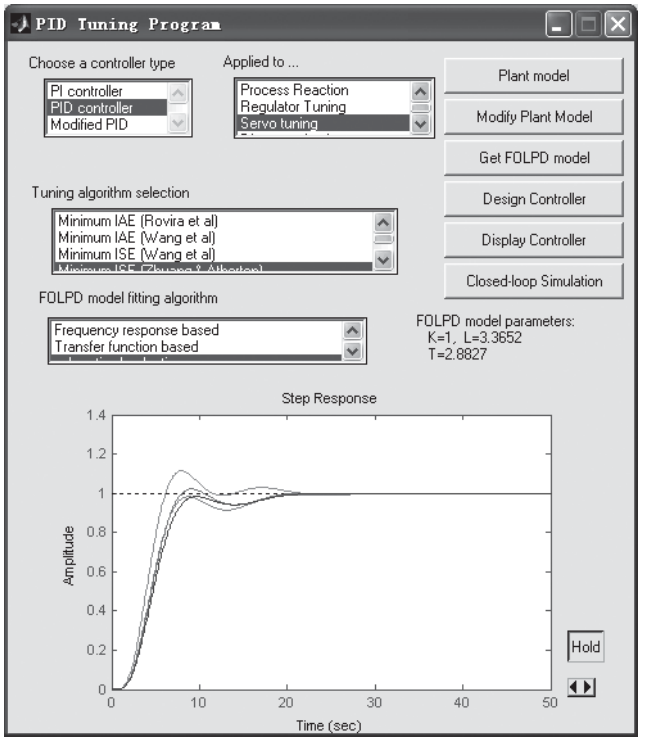


Figure 6.24. *PID controller design and display.*

To design a controller, the FOPDT parameters should be obtained first. The fitting algorithms can be selected as the suboptimal reduction item; the button Get FOPDT model can then be clicked to extract the model parameters, as shown in Figure 6.24.

The controller can be obtained by the Design Controller button. For instance, the Minimum IAE (Wang et al) item can be used to design the controller

$$G_c(s) = 0.936172\left(1 + \frac{1}{4.565340s} + 1.062467s\right).$$

Click the Closed-loop Simulation button to show the closed-loop step response. One may click the Hold button to hold the results. The step responses under different controllers can be displayed together. So, this feature can be used to compare different algorithms, as shown in Figure 6.24.

6.6 Optimal Controller Design

Optimal control is defined as the optimization of certain predefined performance indices. For instance, commonly used performance indices can be the ones in (3.50). Sometimes, parametric objective functions may be used, for example, the linear quadratic optimal regulator problem, where the two weighting matrices $\boldsymbol{Q}$, $\boldsymbol{R}$ need to be defined. There is as yet no universally accepted way to define these two matrices.

In this section, we first summarize and illustrate some solutions to unconstrained and constrained optimization problems using MATLAB. Then the method can be applied to optimal controller design problems. Finally, a MATLAB interface optimal controller designer (OCD) for optimal controller design is presented.

6.6.1 Solutions to Optimization Problems with MATLAB

Unconstrained optimization problems

The mathematical formulation of the unconstrained optimization problem is

$$\min_{\boldsymbol{x}} F(\boldsymbol{x}), \tag{6.42}$$

where $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^{\mathbf{T}}$. The interpretation of the formula is: find the vector $\boldsymbol{x}$ such that the objective function $F(\boldsymbol{x})$ is minimized. If a maximization problem is treated, the objective function can be changed to $-F(\boldsymbol{x})$ such that it can be converted to a minimization problem.

A MATLAB function `fminsearch()` is provided using the well-established simplex algorithm [76]. The syntax of the function is

```
[x, fopt, key, c]=fminsearch(Fun, x0, OPT)
```

where `Fun` is a MATLAB function, an inline function, or an anonymous function to describe the objective function. The variable $\boldsymbol{x}_0$ is the starting point for the search method. The argument `OPT` contains further control options for the optimization process.

Example 6.17. If a function with two variables is given by $z = f(x, y) = (x^2 - 2x)\mathrm{e}^{-x^2-y^2-xy}$ and the minimum point is required, one should first introduce a vector $\boldsymbol{x}$ for the unknown variables x and y. One may select $x_1 = x$ and $x_2 = y$. The objective function can be rewritten as $f(\boldsymbol{x}) = (x_1^2 - 2x_1)\mathrm{e}^{-x_1^2-x_2^2-x_1x_2}$. The objective function can be expressed as an anonymous function such that

```
>> f=@(x)[(x(1)^2-2*x(1))*exp(-x(1)^2-x(2)^2-x(1)*x(2))];
```


If one selects an initial search point at $(0, 0)$, the minimum point can be found with the statements

```
>> x0=[0; 0]; x=fminsearch(f,x0).
```

Then the solution obtained is $\boldsymbol{x} = [0.6110, -0.3055]^{\mathbf{T}}$.

Constrained optimization problems

The general form of the unconstrained optimization problem is

$$\min_{\boldsymbol{x}\ \text{s.t.}} \begin{cases} F(\boldsymbol{x}) \\ \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{B} \\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{B}_{eq} \\ \boldsymbol{x}_m \le \boldsymbol{x} \le \boldsymbol{x}_M \\ \boldsymbol{C}(\boldsymbol{x}) \le \boldsymbol{0} \\ \boldsymbol{C}_{eq}(\boldsymbol{x}) = \boldsymbol{0}, \end{cases} \tag{6.43}$$

where $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^{\mathbf{T}}$. The constraints are classified as linear equality constraints $\boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{B}_{eq}$, linear inequality constraints $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{B}$, and nonlinear constraints $\boldsymbol{C}_{eq}(\boldsymbol{x}) = \boldsymbol{0}$ and $\boldsymbol{C}(\boldsymbol{x}) \le \boldsymbol{0}$. The upper and lower bounds of the optimization variables can also be defined such that $\boldsymbol{x}_m \le \boldsymbol{x} \le \boldsymbol{x}_M$.

The interpretation of the optimization problem is: find the vector $\boldsymbol{x}$, which minimizes the objective function $F(\boldsymbol{x})$, while satisfying all the constraints.

A MATLAB function `fmincon()` can be used to solve constrained optimization problems. The syntax of the function is

```
[x, fopt, key, c]=fmincon(Fun, x0, A, B, Aeq, Beq, xm, xM, CFun, OPT)
```

where `Fun` again could be M-functions, inline functions, or anonymous functions for the objective function, and $\boldsymbol{x}_0$ is the starting search point. The nonlinear constraints can be described by the MATLAB function `CFun`.

Example 6.18. Consider the following nonlinear programming problem:

$$\min_{\boldsymbol{x}\ \text{s.t.}} \begin{cases} 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3. \\ x_1^2+x_2^2+x_3^2-25=0 \\ 8x_1+14x_2+7x_3-56=0 \\ x_1,x_2,x_3 \ge 0 \end{cases}$$

The objective function can be expressed with an anonymous function

```
>> f=@(x)1000-x(1)*x(1)-2*x(2)*x(2)-x(3)*x(3)-x(1)*x(2)-x(1)*x(3);
```

Also, the two constraints are equalities, one of which is nonlinear. The nonlinear constraints can be described in the following MATLAB function, where two constraint variables `ceq` and `c` are returned. Since there is no inequality constraint, the variable `c` returns an empty matrix.

```
function [c,ceq]=opt_con(x)
ceq=x(1)*x(1)+x(2)*x(2)+x(3)*x(3)-25; c=[];
```

The linear equality constraint can be expressed by the $\boldsymbol{A}_{eq}$, $\boldsymbol{B}_{eq}$ matrices, while the linear inequality matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ should be empty ones, since there is no linear inequalities in the problem. Selecting an initial search position at $\boldsymbol{x}_0 = [1, 1, 1]^{\mathbf{T}}$, the problem can then be solved using the following statements:

```
>> x0=[1;1;1]; xm=[0;0;0]; xM=[]; A=[]; B=[]; Aeq=[8,14,7]; Beq=56;
   [x,f_opt,c,d]=fmincon(f,x0,A,B,Aeq,Beq,xm,xM,'opt_con')
```

The optimum solution can then be found, where $\boldsymbol{x}^* = [3.5121, 0.2170, 3.5522]^{\mathbf{T}}$ and $f_{\text{opt}} = 961.7151$.

6.6.2 Optimal Controller Design

With the powerful tools provided in MATLAB, many optimal control problems can be converted into conventional optimization problems. With the above-mentioned functions, some optimal controller design problems can be easily solved. Although not allowing elegant analytical solutions, numerical methods are extremely powerful practical techniques for controller design.

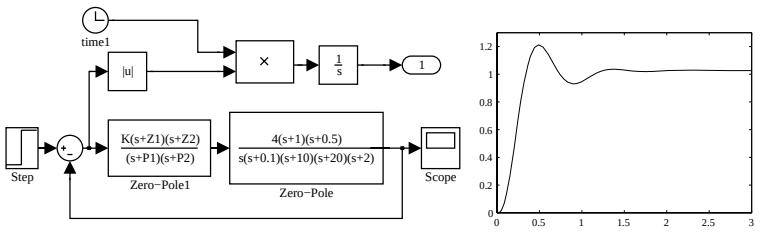
Example 6.19. Assume that

$$G(s) = \frac{10(s+1)(s+0.5)}{s(s+0.1)(s+2)(s+10)(s+20)}.$$

The phase lead-lag controllers can be designed using the method in Sec. 5.1. Here optimal controller design is explored. Integral-type criteria are very suitable for servo control problems. Given the plant model, a Simulink block diagram can be established as shown in Figure 6.25(a), where the ITAE criterion can be evaluated as shown.

In order to minimize the ITAE criterion, the following MATLAB function can be written to describe the objective function:

```
function y=c6optm1(x)
assignin('base','Z1',x(1)); assignin('base','P1',x(2));
assignin('base','Z2',x(3)); assignin('base','P2',x(4));
assignin('base','K',x(5));    % assign variable into MATLAB workspace
[t,xx,yy]=sim('c6moptm1.mdl',3); y=yy(end);   % evaluate objective function
```



(a) Simulink model (file: c6moptm1.mdl) (b) closed-loop response

Figure 6.25. *Phase lead-lag controller and system response.*

The `assignin()` function can be used to assign the variables in the MATLAB workspace, and the model parameters can be defined in the optimization variable vector $\boldsymbol{x}$. The following MATLAB statements can be used to solve the optimization problem:

```
>> x0=20*ones(5,1); x=fminsearch('c6optm1',x0)
```

and the parameters are returned in the variable $\boldsymbol{x}$, from which the controller model can be written as

$$G_c(s) = 243.77\frac{(s+53)(s+66.58)}{(s+38.28)(s+62.09)}.$$

Under this controller, the step response of the system is shown in Figure 6.25(b).

In practical calculation, when the zero of the controller is very small, the computation may become extremely slow. To solve the problem, a suitable constraint to ensure that all the five variables do not become smaller than 0.01 can be introduced. The following statements can then be used to solve the problem:

```
>> x=fmincon('c6optm1',x0,[],[],[],[],0.01*ones(5,1))
```

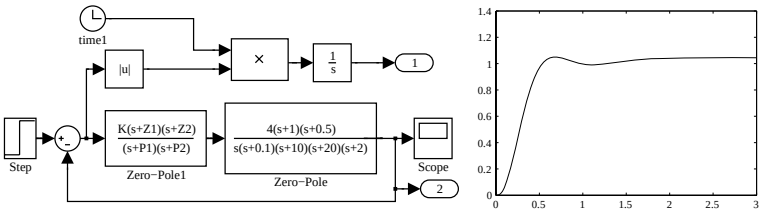
Based on the numerical optimization technique, an extra constraint can be introduced. For instance, if one wants to reduce the overshoot such that $\sigma \leq 3\%$, a new Simulink model can be established as shown in Figure 6.26(a). The objective function can be rewritten as

```
function y=c6optm2(x)
assignin('base','Z1',x(1)); assignin('base','P1',x(2));
assignin('base','Z2',x(3)); assignin('base','P2',x(4));
assignin('base','K',x(5));    % Assign variables to MATLAB workspace
[t,xx,yy]=sim('c6moptm2.mdl',3); y=yy(end,1);   % Evaluate objective function
if max(yy(:,2))>1.03, y=1.2*y; end % update objective function
```

It can be seen from the last line that if the overshoot is too large, one can increase the objective function purposely as a penalty.

The following statements can be given to solve the problem, and the closed-loop step response of the system is shown in Figure 6.26(b).

```
>> x=fmincon('c6optm2',x0,[],[],[],[],0.01*ones(5,1))
```



(a) modified Simulink model (file:c6moptm2.mdl) (b) closed-loop step response

Figure 6.26. *Modified simulation model and response.*

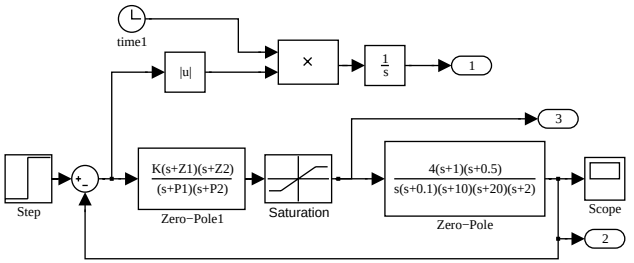


Figure 6.27. *The Simulink model with saturation (file: c6moptm3.mdl).*

The controller model

$$G_{c_2}(s) = 161.4965\frac{(s+43.1203)(s+55.7344)}{(s+28.4746)(s+61.0652)}$$

can be designed.

In ordinary control theory, hardware-related implementation of the PID controller is often not considered; i.e., in theory, extremely large signals are acceptable. In real-time control, however, the signal cannot be too large in order to avoid hardware failure. It can be seen that for a unit step input, this controller gives an initial output of 200, which is too high. It could cause hardware problems with a bad design and saturate the actuator leading to nonlinear operation. However, if saturation is included in the actuator, the resulting response can be easily solved using numerical methods, since one can simply add a saturation block in the Simulink model.

Example 6.20. Consider again the controller design problem. Assuming that the control signal should be kept within ± 20, the Simulink model can be modified as shown in Figure 6.27, and the objective function can be rewritten as

```
function y=c6optm3(x)
assignin('base','Z1',x(1)); assignin('base','P1',x(2));
assignin('base','Z2',x(3)); assignin('base','P2',x(4));
assignin('base','K',x(5));    % assign variables in MATLAB workspace
[t,xx,yy]=sim('c6moptm3.mdl',15); y=yy(end,1);   %   evaluate objective function
if max(yy(:,2))>1.03, y=1.4*y; end     % update the objective function
```

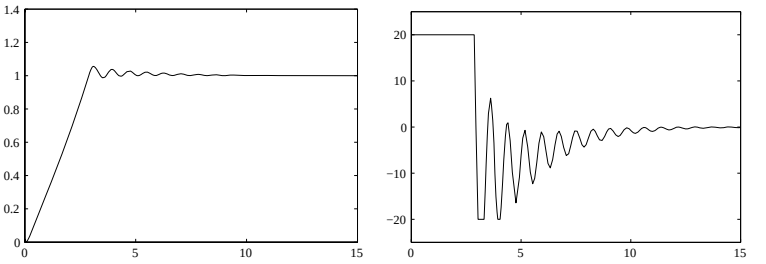
The following statements can be used to search for the optimum controller for the system:

```
>> x=fmincon('c6optm3',x0,[],[],[],[],0.01*ones(5,1))
```

and the controller

$$G_c(s) = 37.1595\frac{(s+142.6051)(s+62.6172)}{(s+20.3824)(s+27.6579)}$$

can be designed. The output signal and the control signal under such a controller can be obtained as shown in Figure 6.28. It can be seen that the control results are satisfactory.



(a) output signal (b) control signal

Figure 6.28. *Step response of the system when saturations are introduced.*

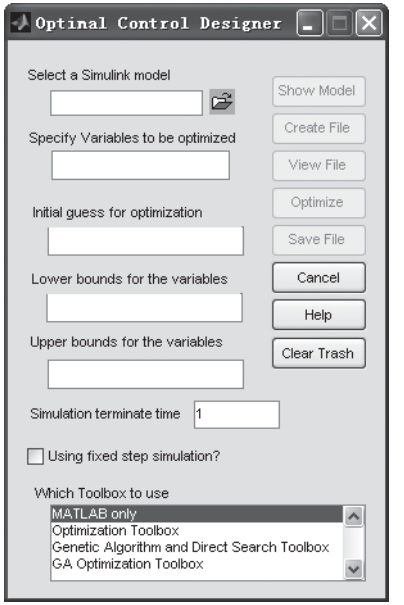


Figure 6.29. *OCD interface.*

6.6.3 A MATLAB/Simulink-Based Optimal Controller Designer and Its Applications

From the examples in the previous section, using numerical optimization algorithms, optimal controller design can be made simple. In this section, we will introduce a MATLAB/Simulink-based optimal controller designer (OCD) with some application examples.

The procedures for applying the OCD program are as follows:

1. Type `ocd` at the MATLAB prompt; the main interface is shown in Figure 6.29. The program can be used in optimal controller design.

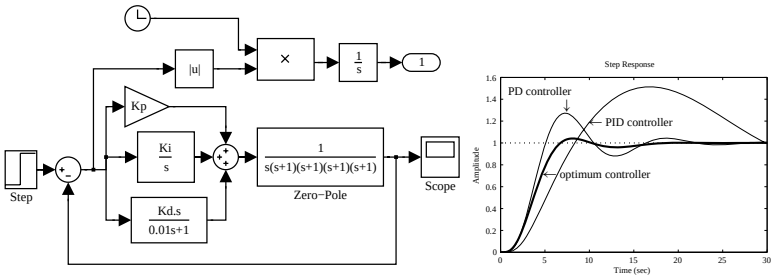
2. A Simulink model can be established and the model should contain at least two elements: the undetermined variable names and the outport reflecting the optimum criterion. For instance, in the PI controller design problem, the two variables K_p and K_i can be assigned. The ITAE criterion can be represented in the Simulink model as outport 1.
3. Fill in the Simulink model name in the Select a Simulink model edit box.
4. Fill in the variable names to be optimized in the Specify Variables to be optimized edit box, with variable names separated with commas.
5. Estimate the terminate time for the error to become zero and enter it in the Simulation terminate time edit box.
6. Click Create File to automatically generate a MATLAB function `optfun_*.m` and click Clear Trash to delete the temporary objective function files.
7. Click Optimize to start the optimization process. The optimal variables can be obtained. Sometimes, the button should be clicked again to ensure more accurate optimum solutions. The functions `fminsearch()`, `nonlin()` and `fmincon()` can be called automatically for parameter optimization.
8. The upper and lower bounds to the variables can also be used, and an initial search point can be specified, if necessary.

Example 6.21. Consider the FOIPDT-type plant model in Example 6.15; i.e., the plant model is given by

$$G(s) = \frac{1}{s(s+1)^4}.$$

The Simulink model for the PID control, with ITAE descriptions, is established as shown in Figure 6.30(a), and it is saved in the file `c6mopt4.mdl`.

Fill in the Simulink model name in the Select a Simulink model edit box, for instance, fill in `c6mopt4` for this example. The variable names to be optimized, `Kp,Ki,Kd` should be entered in the Specify Variables to be optimized edit box, and enter 30 in the Simulation terminate time edit box. Then click the Create File button to automatically generate the MATLAB function to describe the objective function



(a) Simulink model (file: c6mopt4.mdl) (b) comparisons

Figure 6.30. *PID control model and response comparisons.*


```
function y=optfun_2(x)
assignin('base','Kp',x(1));
assignin('base','Ki',x(2));
assignin('base','Kd',x(3));
[t_time,x_state,y_out]=sim('c6mopt4.mdl',[0,30.000000]);
y=y_out(end);
```

where the second, third, and fourth lines in the code will assign the variables in vector $\boldsymbol{x}$ to the variables K_p, K_i, K_d in the MATLAB workspace. Simulation is then performed to calculate the objective function.

Click the Optimize button to initiate the optimization process. In the meantime, the scope window should be opened to visualize the optimization process. After optimization, the optimum PID controller will be obtained as

$$G_c(s) = 0.2583 + \frac{0.0001}{s} + \frac{0.7159s}{0.01s+1}$$

which minimizes the ITAE criterion. It can be seen that $K_i = 0.0001$ is very small, which can be neglected, and thus a PD controller is sufficient for the system. The closed-loop step response is shown in Figure 6.30(b). It can be seen that the control response is highly superior to the one obtained in Example 6.15.

Example 6.22. The OCD program is not restricted to simple PID controller problems. It can also be used for complicated system models such as the cascade PI control system shown in Figure 2.11.

To solve the problem, the Simulink model shown in Figure 6.31 can be established, and saved as `c6model2.mdl`. Note that four undetermined parameters `Kp1`, `Ki1`, `Kp2`, `Ki2` should be optimized. The ITAE criterion can be defined. Starting the OCD, the model name `c6model2` should be entered into the Select a Simulink model edit box, and in the Specify Variables to be optimized edit box, `Kp1,Ki1,Kp2,Ki2` should be filled in. Also, in the Simulation terminate time edit box, one may fill in 0.6. Click the Create File to generate the MATLAB function. One may design the controllers by clicking the Optimize button, and the controllers, which minimize the ITAE criterion, can be found as

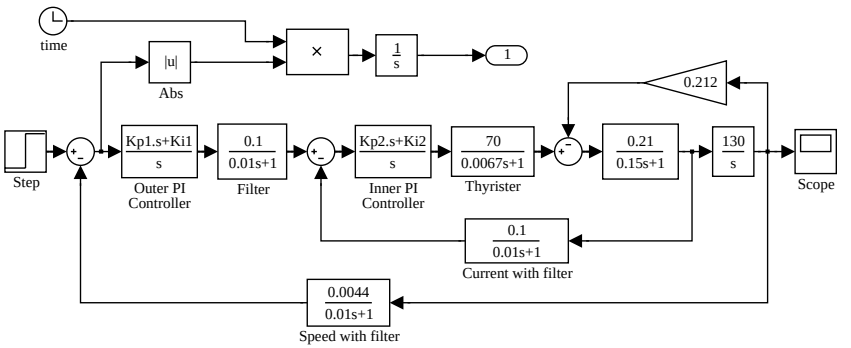


Figure 6.31. *Simulation model of cascade PI control (file: c6model2.mdl).*

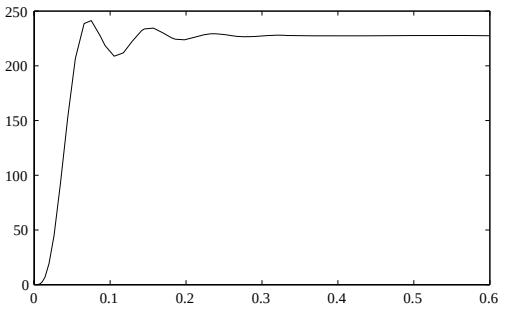


Figure 6.32. *Optimal control of the system.*

$K_{p_1} = 37.9118$, $K_{i_1} = 12.1855$, $K_{p_2} = 10.8489$, and $K_{i_2} = 0.9591$, i.e., the controllers are

$$G_{c_1}(s) = 37.9118 + \frac{12.1855}{s} \quad \text{and} \quad G_{c_2}(s) = 10.8489 + \frac{0.9591}{s}.$$

Under these controllers, the step response of the closed-loop system can be obtained as shown in Figure 6.32. It can be seen that the response is satisfactory.

It can be seen from the previous examples that the OCD program is quite versatile in finding the optimal controllers. However, in some applications, the OCD may not find a solution due to the poorly posed problem or because a good initial search point has not been found. This can be a drawback in conventional optimization algorithms, but many such problems can be avoided by intelligent use based on an understanding of the system behavior.

The genetic algorithm (GA) [77] allows the optimization search from many initial points in a parallel manner. The Genetic Algorithm Optimization Toolbox (GAOT) [78] provides a series of MATLAB-based functions for solving optimization problems using genetic algorithms. This toolbox is used with the OCD program, and the facility is useful in solving problems where conventional optimization methods cannot easily find an initial feasible search point. The GA Optimization Toolbox is the last list box in Figure 6.29.

Example 6.23. Consider an unstable plant model

$$G(s) = \frac{s+2}{s^4+8s^3+4s^2-s+0.4}.$$

By the direct use of the OCD program, a feasible PID controller cannot be designed. However, one may still establish a Simulink model as shown in Figure 6.33, which is the same as the previous examples.

In order to ensure that the control action is not too large, a saturation element can be appended to the controller, with the saturation width of $\Delta = 5$. From the OCD program, with the GAOT selection, the optimal PID controller can be designed as

$$G_c(s) = 47.8313 + \frac{0.2041}{s} + \frac{55.3632s}{0.01s+1}.$$

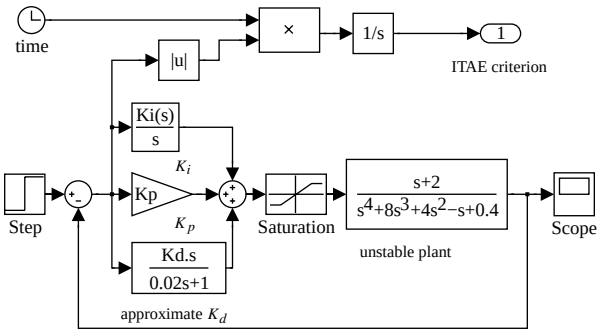


Figure 6.33. *Simulink model for PID control (file: c6munsta.mdl).*

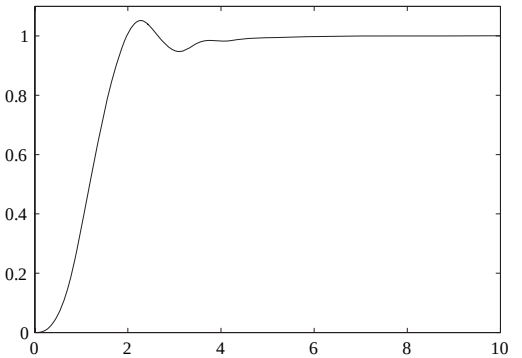


Figure 6.34. *Simulation results for an unstable plant with a PID controller.*

The step response of the closed-loop system under the optimal controller is shown in Figure 6.34. It can be seen that the PID controller can still be designed, with the help of GAs, and the transient response is satisfactory.

6.7 More Topics on PID Control

6.7.1 Integral Windup and Anti-Windup PID Controllers

A Simulink model for the study of the phenomenon of integrator windup is shown in Figure 6.35.

The plant model is given by

$$G(s) = \frac{10}{s^4 + 10s^3 + 35s^2 + 50s + 24},$$

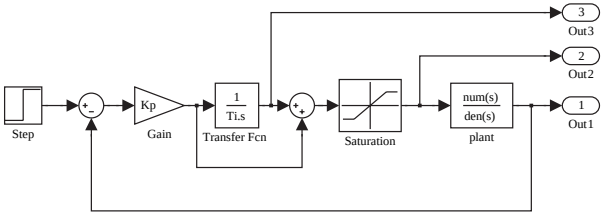


Figure 6.35. *Integrator windup demonstration (file: c6mwind.mdl).*

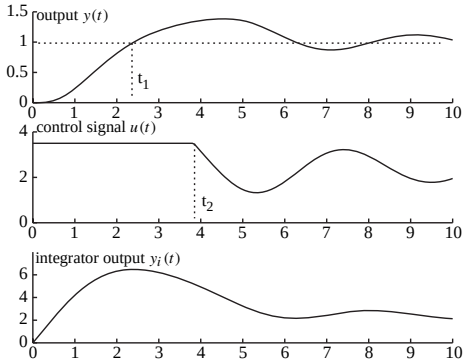


Figure 6.36. *Integrator windup demonstration.*

and the parameters for the PI controller are given by $K_p = 5.04$ and $T_i = 1.124$. With an actuator saturation nonlinear element given by $u_m = 3.5$, the related signals in the PI controlled system are shown in Figure 6.36. When there is an initial set-point change in $r(t)$, the error signal is initially so large that the control signal $u(t)$ quickly reaches its actuator saturation limit. Even when the output signal reaches the reference value at the time t_1, which gives a negative error signal due to the large value of the integrator output, the control signal still remains at the saturation value u_m, which causes the output of the system to continuously increase until it reaches the time t_2, and the negative action of the error signal begins to have effect. This phenomenon is referred to as the integrator windup action, which is undesirable in control applications. Therefore, we need to briefly introduce different antiwindup PID controllers for use in practice. We shall use Simulink for illustration.

An antiwindup PID controller is provided as an icon in the Simulink environment, and the internal structure is shown in Figure 6.37. The signal reflecting the actuator saturation is fed into the integrator action, which is determined by a ratio $1/T_t$. For instance, one can simulate the PID control system in the previous example using the Simulink model as shown in Figure 6.38(a). For different T_t, the output signals are compared in Figure 6.38(b). It can be seen that for smaller values of T_t, the windup phenomenon can be reduced more significantly.

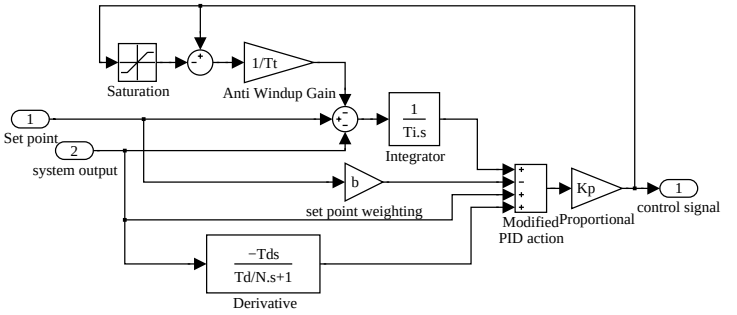
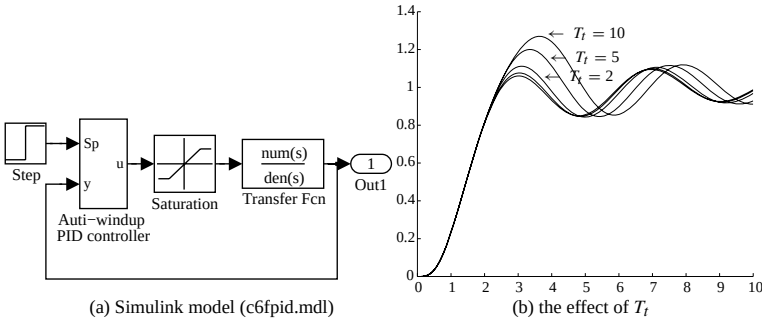


Figure 6.37. *Anti-windup PID structure (file: c6awpid.mdl).*



(a) Simulink model (c6fpid.mdl) (b) the effect of T_t

Figure 6.38. *Effect of anti-windup PI controllers.*

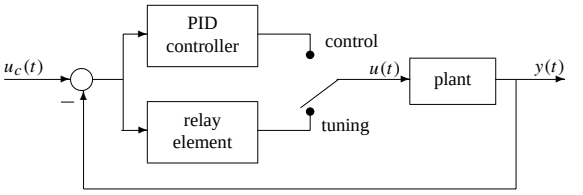
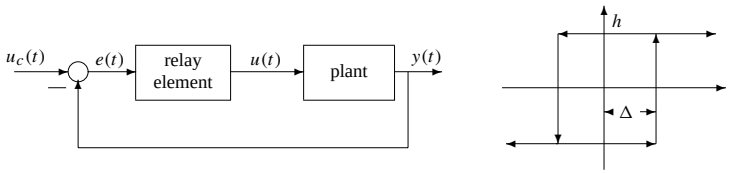


Figure 6.39. *Structure of a relay automatic tuning PID controller.*

6.7.2 Automatic Tuning of PID Controllers

An automatic tuning (also known as autotuning or autotuner) PID controller strategy is proposed by Åström and Hägglund [61]. Now the commercial automatic tuning PID controllers are available from most hardware manufacturers.

The structure of the relay-type of automatic tuning is shown in Figure 6.39, and it can be seen that the two modes are alternated by the use of switching. When the operator feels the need to adjust the parameters of the PID controller, he or she can simply press a button



(a) relay control block diagram (b) typical relay element

Figure 6.40. *Nonlinear model of relay control.*

to switch the process to the tuning mode, and the parameters can be tuned automatically. When this tuning task is completed, the process can be switched back to normal feedback control mode.

Under the tuning mode, the system is equivalent to the structure shown in Figure 6.40(a), and the typical relay nonlinearity is shown in Figure 6.40(b). Several approaches can be used to determine the crossover frequency ω_c and the ultimate gain K_c. The describing function approach is the theoretical basis for relay autotuning analysis, and Tsypkin's method (see Atherton [51]) can also be applied as described below.

Determining ω_c and K_c with the describing function method

In the describing function approach [51], one can approximately represent the static nonlinear element by an equivalent gain in analyzing the so-called limit cycles. Such a gain is referred to as the describing function of the nonlinearity and is in fact input amplitude dependent. For different nonlinear functions, the describing functions may also be different; a comprehensive study of describing functions can be found in [51].

The limit cycle, or oscillation, can be approximately determined by finding the intersection of the Nyquist plot of the plant model with the negative reciprocal of the describing function $N(a)$, as illustrated in Figure 6.41(a), which means that the conditions when the oscillation occurs are

$$1 + N(a)G(s)\mid_{s=\mathrm{j}\omega_c} = 0, \quad \text{i.e.,} \quad G(\mathrm{j}\omega_c) = -\frac{1}{N(a)}. \tag{6.44}$$

The describing function of the system with relay nonlinearity given in Figure 6.40(b) is that

$$N(a) = \frac{4h}{\pi a^2}\left(\sqrt{a^2-\Delta^2} - \mathrm{j}\Delta\right), \tag{6.45}$$

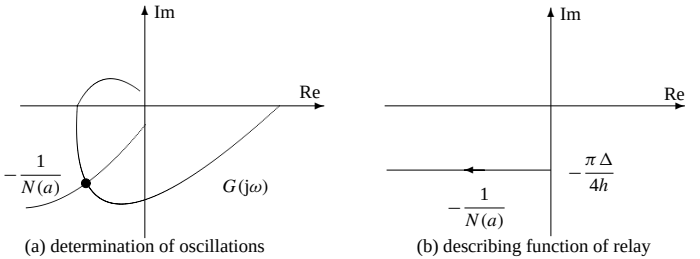
from which the negative reciprocal of the describing function $N(a)$ is simply

$$-\frac{1}{N(a)} = -\frac{\pi}{4h}\sqrt{a^2-\Delta^2} - \mathrm{j}\frac{\pi\Delta}{4h}, \tag{6.46}$$

which is just a straight line as shown in Figure 6.41(b).

The crossover frequency ω_c and the ultimate gain K_c can be obtained. For simplicity, assume that $\Delta = 0$. Then, the describing function can be simplified to $N(a) = 4h/(\pi a)$. So, immediately, one has

$$K_c = \frac{4h}{\pi a}, \quad T_c = \frac{2\pi}{\omega_c}. \tag{6.47}$$



(a) determination of oscillations (b) describing function of relay

Figure 6.41. *Determination of the magnitude and frequency of oscillations.*

Determining ω_c and K_c with Tsypkin's method

The describing function method is essentially based on the principle of fundamental harmonic equivalence. Tsypkin's method, on the other hand, can be used when more accurate analysis of relay systems is required, where the higher-order harmonics need to be considered apart from the fundamental one, for relay nonlinearities.

The Fourier series expansion of the square wave signal, which is the output of the relay action, can be written as

$$y(t) = \sum_{n=1(2)}^{\infty} \frac{4h}{n\pi}\sin n\omega(t-t_1), \tag{6.48}$$

where "(2)" represents a step of 2; i.e., only odd harmonics are considered since the relay function is an odd function. The Fourier series expansion of the output signal can then be written as

$$c(t) = \sum_{n=1(2)}^{\infty} \frac{4h}{n\pi} g_n \sin[n\omega(t-t_1)+\phi_n] \tag{6.49}$$

with g_n and ϕ_n the magnitude and phase of the plant model, respectively, i.e., $G(nj\omega) = g_n e^{j\phi_n}$. If the external input to the system is 0, then $x(t) = -c(t)$, and the switching point satisfies $x(t_1) = \delta, \dot{x}(t_1) < 0$. The locus $A(\omega)$ can be defined as

$$\mathrm{Re}[A_G(\theta,\omega)] = \sum_{n=1(2)}^{\infty}\Big[V_G(n\theta)\sin(n\theta) + U_G(n\theta)\cos(n\theta)\Big], \tag{6.50}$$

$$\mathrm{Im}[A_G(\theta,\omega)] = \sum_{n=1(2)}^{\infty}\left[\frac{1}{n}V_G(n\theta)\cos(n\theta) - U_G(n\theta)\sin(n\theta)\right], \tag{6.51}$$

where $G(nj\omega) = U_G(n\omega) + \mathrm{j}V_G(n\omega)$. Assume that $t_1 = 0$. The magnitude and frequency of the limit cycles can be solved from

$$\mathrm{Im}[A_G(0,\omega) + A_G(\omega\Delta t,\omega)] = -\frac{\pi\delta}{2h} \tag{6.52}$$

and with the constraints $\mathrm{Re}[A_G(0,\omega) - A_G(\omega\Delta t,\omega)] < 0$. If the relay element is symmetrical, then one has

$$\mathrm{Im}[A_G(0,\omega)] = -\frac{\pi\delta}{4h}. \tag{6.53}$$

6.7.3 Control Strategy Selection

It has been pointed out in some references, such as [60], that PID controllers can be used only for plants with relatively small time delay (or equivalent delays). When the delay constant increases, the PID controller cannot guarantee good responses. In fact, apart from the traditional PID control structure, other control strategies may also be used to deal with such cases. This leaves us with the following question: In practical applications, what kind of controller structure should be used to design a usable controller for a given plant model?

Such a question is well studied in [79], where the normalized parameters τ and κ are introduced, from which different control strategies are suggested, as summarized in Table 6.16, where apart from the τ and κ parameters, τ_2 and κ_2 are also introduced for the plant model given by

$$G(s) = \frac{K_v}{s(1+sT_v)}\mathrm{e}^{-sL}$$

with the relations

$$\tau_1 = \frac{L}{T_v}, \quad \kappa_2 = \frac{\lim\limits_{s\to 0} sG(s)}{\omega_c |G(j\omega_c)|} = \frac{1}{2\pi} K_v K_c T_c, \quad \text{and} \quad \tau_2 = \frac{\frac{2}{\pi} + \operatorname{atan}\sqrt{\kappa_2^2 - 1}}{\sqrt{\kappa_2^2 - 1}}. \tag{6.54}$$

It can be seen that Table 6.16, in some sense, can be used as a guide for choosing a suitable controller structure for a given plant model.

Table 6.16. *Controller selection from the plant model.*

Ranges of τ or κ	No precise control necessary	Precise control needed		
		High noise	Low saturation	Low measurement noise
$\tau > 1, \kappa < 1.5$	I control	I+B+C	PI+B+C	PI+B+C
$0.6 < \tau < 1$ $1.5 < \kappa < 2.25$	I or PI control	I+A	PI+A	PI+A+C or PID+A+C
$0.15 < \tau < 0.6$ $2.25 < \kappa < 15$	PI control	PI	PI or PID	PID
$\tau < 0.15, \kappa > 15$or $\tau_2 > 0.3, \kappa_2 < 2$	P or PI control	PI	PI or PID	PI or PID
$\tau_2 < 0.3, \kappa_2 > 2$	PD+E	F	PD+E	PD+E

A represents forward compensation suggested

B represents forward compensation necessary

C represents dead-zone compensation suggested

D represents dead-zone compensation necessary

E represents set-point weighting necessary

F represents for pole placement

Problems

1. For the plant models

   (a) $G_a(s) = \dfrac{1}{(s+1)^3}$, (b) $G_b(s) = \dfrac{1}{(s+1)^5}$, (c) $G_c(s) = \dfrac{-1.5s+1}{(s+1)^3}$,

   design PID (or PI) controllers using different design algorithms from this chapter and compare the closed-loop behaviors of the controlled systems.

2. Find the FOPDT approximations to the plant models given by

   (a) $G(s) = \dfrac{12(s^2-3s+6)}{(s+1)(s+5)(s^2+3s+6)(s^2+s+2)}$,

   (b) $G(s) = \dfrac{-5s+2}{(s+1)^2(s+3)^3}\mathrm{e}^{-0.5s}$,

   (c) $G(z) = \dfrac{1.0569\times 10^{-5}(z+18.42)(z+1.841)(z+0.3406)(z+0.03405)}{(z-0.8025)(z-0.7866)(z-0.7711)(z-0.7558)(z-0.6703)}$, $T=0.1$,

   using various algorithms discussed in this chapter. Compare the closeness of the approximation using relevant time and frequency domain analysis techniques.

3. Investigate the disturbance rejection properties of the controllers designed for the plants in Problem 1. Assume that the disturbances are added in the steady-state responses. If any of the controllers does not perform well for disturbance rejection, design a new PID controller to improve the disturbance rejection performance and check whether the new PID controller is suitable for set-point control.

4. For different PID controllers in problem 1, analyze the compensated systems with time and frequency domain tools. When the derivative term in the controller is disabled, what will happen with the control performance?

5. Using the PID tuner program, compare the PID controllers designed from different design approaches for the plant model

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

   and find a good PID controller.

6. Construct a Simulink model for PID control system structures with the plant model containing a pure delay term. Design different PID controllers for the plant models given below:

   (a) $G_a(s) = \dfrac{1}{(s+1)(2s+1)}\mathrm{e}^{-s}$, (b) $G_b(s) = \dfrac{1}{(17s+1)(6s+1)}\mathrm{e}^{-30s}$,

   (c) $G_c(s) = \dfrac{s+2}{(s+1)(4s+1)}\mathrm{e}^{-0.1s}$, (d) $G(z) = \dfrac{0.01752z+0.01534}{z^2-1.637z+0.6703}z^{-10}$, $T=0.2$.

   Compare the simulation results with the approximate results when the pure delay term is replaced by a Padé approximation.

7. Design PID controllers for the plants

$$\text{(a) } G(s) = \frac{15}{s(s+1)(s+2)^2(s+5)}, \quad \text{(b) } G(s) = \frac{5(s-5)}{s(s+5)^4}.$$

8. Solve the unconstrained optimization problem

$$\min_{\boldsymbol{x}} \begin{array}{l} 100(x_2 - x_1^2)^2 + (1-x_1)^2 + 90(x_4 - x_3^2) + (1-x_3^2)^2 \\ \quad +10.1\left[(x_2-1)^2 + (x_4-1)^2\right] + 19.8(x_2-1)(x_4-1). \end{array}$$

9. Solve the constrained optimization problems

(a)
$$\min_{\boldsymbol{x} \text{ s.t.}} \begin{cases} x_1 + x_2 \le 0 \\ -x_1 x_2 + x_1 + x_2 \ge 1.5 \\ x_1 x_2 \ge -10 \\ -10 \le x_1, x_2 \le 10 \end{cases} \mathrm{e}^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1),$$

(b)
$$\max_{\boldsymbol{x} \text{ s.t.}} \begin{cases} 0.003079 x_1^3 x_2^3 x_5 - \cos^3 x_6 \ge 0 \\ 0.1017 x_3^3 x_4^3 - x_5^2 \cos^3 x_6 \ge 0 \\ 0.09939(1+x_5) x_1^3 x_2^2 - \cos^2 x_6 \ge 0 \\ 0.1076(31.5+x_5) x_3^3 x_4^2 - x_5^2 \cos^2 x_6 \ge 0 \\ x_3 x_4 (x_5 + 31.5) - x_5 [2(x_1+5)\cos x_6 + x_1 x_2 x_5] \ge 0 \\ 0.2 \le x_1 \le 0.5, 14 < \le x_2 \le 22, 0.35 \le x_3 \le 0.6 \\ 16 \le x_4 \le 22, 5.8 \le x_5 \le 6.5, 0.14 \le x_6 \le 0.2618 \end{cases} \frac{1}{2\cos x_6}\left[x_1 x_2 (1+x_5) + x_3 x_4 \left(1 + \frac{31.5}{x_5}\right)\right].$$

10. Using ITAE, IAE, and ISE criteria, design optimal PID controllers for the open-loop plants

$$\text{(a) } G_a(s) = \frac{1}{(s+1)(2s+1)}\mathrm{e}^{-s}, \quad \text{(b)} G_b(s) = \frac{1}{(17s+1)(6s+1)}\mathrm{e}^{-30s}$$

and comment on which criterion will usually lead to the best control results.

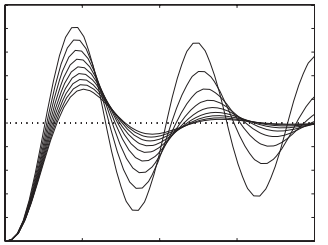
11. For a time varying plant model $\ddot{y}(t) + \mathrm{e}^{-0.2t}\dot{y}(t) + \mathrm{e}^{-5t}\sin(2t+6)y(t) = u(t)$, design an optimal PI control which minimizes the ITAE criterion. Analyze the closed-loop behavior of the system.

12. For the plant model

$$G(s) = \frac{1 + \frac{3\mathrm{e}^{-s}}{s+1}}{s+1},$$

design an optimal PID controller and analyze the step response of the closed-loop system.

13. In the OCD examples, the selection of simulation terminate time t_f is quite important. Please summarize how the t_f should be selected.



Chapter 7

Robust Control Systems Design

So far, we have presented some model-based controller design techniques in Chapter 5 and PID controller design methods in Chapter 6 which require only a rough model of the plant to be controlled. It is natural and practically important to ask, "Is the designed controller robust against the uncertainty and disturbance?"

We should note that the control system design methods discussed in Chapters 5 and 6 did not explicitly and quantitatively take into consideration the disturbance and uncertainty. It was assumed in Chapter 5 that an accurate plant model is available. This is usually not true in practice. The model is essentially an approximation of the actual physical plant. There may exist modeling uncertainty, which is the difference between the model and the actual plant property, also known as model mismatch. If the designed controller can tolerate the model mismatch, the controller is called "robust." This implies that the control system performance will not degrade significantly in the presence of model mismatch using the "robust controller." In Chapter 6, to design a PID controller with a reasonably good performance, an accurate model of the plant to be controlled was not required. There, PID controllers were considered to have certain robustness in the sense of tolerating model uncertainty. However, we must be clear that in the design of PID controllers, no quantitative information about the model mismatch is used. Therefore, PID controllers sometimes may not be robust.

In this chapter, we present a new framework within which uncertainty and disturbance can be explicitly and quantitatively taken into account during the design of the controller. This is referred to as the robust controller design and has been the focus of research for decades. We believe that robust control will continue to be a topic of further research since the robustness issue of any controller design is an inherent problem that must always be addressed. Moreover, the research on robust control will be multifaceted. The reason is obvious: different types of knowledge about uncertainty and disturbance will lead to different robust controller design methods. In this chapter, we cover some more advanced materials on robust control. Specifically, the presentation will be closely coupled with the Robust Control Toolbox for MATLAB. In Sec. 7.1, we introduce the linear quadratic Gaussian (LQG) problem, and in particular, the loop

transfer recovery (LTR) technique. $\mathcal{H}_2$- and $\mathcal{H}_\infty$-norm design problems are summarized in Sec. 7.2. In Sec. 7.3, we focus on the $\mathcal{H}_\infty$ design technique with detailed MATLAB solutions. The optimal $\mathcal{H}_\infty$ controller design technique will also be discussed. Sec. 7.4 covers the $\mathcal{H}_2$-norm controller design technique with relevant MATLAB solution methods. More problems on $\mathcal{H}_\infty$ control, weighting function selections, and so on are presented in Sec. 7.5.

7.1 Linear Quadratic Gaussian Control

LQG control is considered a robust control method since noise in the state and output equations is explicitly considered. Furthermore, quantitative information about the noise is used in the controller design.

7.1.1 LQG Problem

Consider the state space model of the plant

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) + \boldsymbol{\Gamma}\boldsymbol{\xi}(t), \quad \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{\theta}(t), \tag{7.1}$$

where $\boldsymbol{\xi}(t)$ and $\boldsymbol{\theta}(t)$ are random noises in the state equation and the output measurements, respectively. Assume that $\boldsymbol{\xi}(t)$ and $\boldsymbol{\theta}(t)$ are zero mean Gaussian random processes with covariance matrices given by

$$\mathrm{E}[\boldsymbol{\xi}(t)\boldsymbol{\xi}^{\mathbf{T}}(t)] = \boldsymbol{\Xi} \geq 0, \quad \mathrm{E}[\boldsymbol{\theta}(t)\boldsymbol{\theta}^{\mathbf{T}}(t)] = \boldsymbol{\Theta} > 0, \tag{7.2}$$

where $\mathrm{E}[\boldsymbol{x}]$ denotes the mean value of $\boldsymbol{x}$ and $\mathrm{E}[\boldsymbol{x}\boldsymbol{x}^{\mathbf{T}}]$ is the covariance matrix of the zero mean Gaussian signal $\boldsymbol{x}$. The random signals $\boldsymbol{\xi}(t)$ and $\boldsymbol{\theta}(t)$ are further assumed to be mutually independent, i.e., $\mathrm{E}[\boldsymbol{\xi}(t)\boldsymbol{\theta}^{\mathbf{T}}(t)] = 0$. The performance index for optimal control is defined as

$$J = \mathrm{E}\left\{\int_0^\infty \left[\boldsymbol{z}^{\mathbf{T}}(t)\boldsymbol{Q}\boldsymbol{z}(t) + \boldsymbol{u}^{\mathbf{T}}(t)\boldsymbol{R}\boldsymbol{u}(t)\right]\mathrm{d}t\right\}, \tag{7.3}$$

where $\boldsymbol{z}(t) = \boldsymbol{M}\boldsymbol{x}(t)$ is the linear combination of state vector $\boldsymbol{x}(t)$ with $\boldsymbol{M}$ defined by the user to measure the performance. The constant weighting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ are, respectively, a symmetrical semipositive-definite and a symmetrical positive-definite matrix, that is, $\boldsymbol{Q} = \boldsymbol{Q}^{\mathbf{T}} \geq 0$, $\boldsymbol{R} = \boldsymbol{R}^{\mathbf{T}} > 0$. Note that $\boldsymbol{R}$ is a scalar when (7.1) is a single input–single output (SISO) feedback control system—the main theme of this book.

The LQG problem can be divided into the following two subproblems:

1. The LQ optimal state feedback control, as discussed in Sec. 5.2, and
2. the state estimation with disturbances.

7.1.2 LQG Problem Solutions Using MATLAB

LQG control with Kalman filters

The states can be estimated optimally if a Kalman filter, rather than an observer, is used.

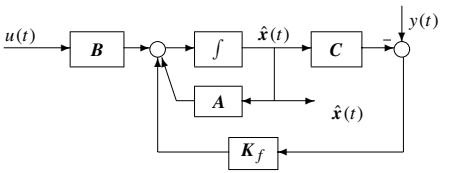


Figure 7.1. *Kalman filter.*

One can first find an optimal state estimation signal $\hat{x}(t)$, which minimizes the covariance $\mathrm{E}[(\boldsymbol{x} - \hat{\boldsymbol{x}})(\boldsymbol{x} - \hat{\boldsymbol{x}})^{\mathbf{T}}]$, and then use the estimated signal $\hat{\boldsymbol{x}}(t)$ to replace the actual state variables such that the original problem can be reduced to an ordinary LQ optimal control problem.

The block diagram of the Kalman filter is shown in Figure 7.1, where the Kalman filter gain matrix $\boldsymbol{K}_f$ is given by

$$\boldsymbol{K}_f = \boldsymbol{P}_f\boldsymbol{C}^{\mathbf{T}}\boldsymbol{\Theta}^{-1}, \tag{7.4}$$

where $\boldsymbol{P}_f$ satisfies the algebraic Riccati equation (ARE)

$$\boldsymbol{P}_f\boldsymbol{A}^{\mathbf{T}} + \boldsymbol{A}\boldsymbol{P}_f - \boldsymbol{P}_f\boldsymbol{C}^{\mathbf{T}}\boldsymbol{\Theta}^{-1}\boldsymbol{C}\boldsymbol{P}_f + \boldsymbol{\Gamma}\boldsymbol{\Xi}\boldsymbol{\Gamma}^{\mathbf{T}} = \boldsymbol{0}, \tag{7.5}$$

and $\boldsymbol{P}_f$ is a symmetrical semipositive-definite matrix, i.e., $\boldsymbol{P}_f = \boldsymbol{P}_f^{\mathbf{T}} \geq 0$.

A MATLAB function `kalman()` provided in the Control Systems Toolbox can be used to find the $\boldsymbol{K}_f$ matrix of the Kalman filter. The syntax of the function is

```
[Gk,Kf,Pf]=kalman(G,Ξ,Θ)
```

where $\boldsymbol{G}$ is the extended state space model object with Gaussian disturbances, i.e., $\boldsymbol{G} = (\boldsymbol{A}, \widetilde{\boldsymbol{B}}, \boldsymbol{C}, \widetilde{\boldsymbol{D}})$. $\boldsymbol{G}$ can be regarded as an extended model with two input matrices with $\widetilde{\boldsymbol{B}} = [\boldsymbol{B}, \boldsymbol{\Gamma}]$ and $\widetilde{\boldsymbol{D}} = [\boldsymbol{D}, \boldsymbol{D}]$. $\boldsymbol{G}_k$ is the state space object of the Kalman filter. $\boldsymbol{K}_f$ is the state feedback matrix and $\boldsymbol{P}_f$ is the solution of the Riccati equation of the Kalman filter given in (7.5).

Example 7.1. For the system given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} -0.02 & 0.005 & 2.4 & -32 \\ -0.14 & 0.44 & -1.3 & -30 \\ 0 & 0.018 & -1.6 & 1.2 \\ 0 & 0 & 1 & 0 \end{bmatrix}\boldsymbol{x} + \begin{bmatrix} 0.14 \\ 0.36 \\ 0.35 \\ 0 \end{bmatrix}u + \begin{bmatrix} -0.12 \\ -0.86 \\ 0.009 \\ 0 \end{bmatrix}\xi(t), \quad y = \boldsymbol{x}_2 + \theta(t),$$

where $\Xi = 10^{-3}$ and $\Theta = 10^{-7}$, the Kalman filter can be designed using the following MATLAB statements:

```
>> A=[-0.02,0.005,2.4,-32; -0.14,0.44,-1.3,-30;
      0,0.018,-1.6,1.2; 0,0,1,0];
   B=[0.14; 0.36; 0.35; 0]; G=[-0.12; -0.86; 0.009; 0];C=[0,1,0,0];
   G=ss(A,[B,G],C,[0,0]); Xi=1e-3; Theta=1e-7;
   [Gk,Kf,Pf]=kalman(G,Xi,Theta)
```

and it can be found that $\boldsymbol{K}_f^{\mathrm{T}} = [215.33, 87.371, -2.5369, -3.5741]$ and

$$\boldsymbol{P}_f = \begin{bmatrix} 0.0044357 & 2.1533\times10^{-5} & -3.6456\times10^{-5} & -7.7729\times10^{-5} \\ 2.1533\times10^{-5} & 8.7371\times10^{-6} & -2.5369\times10^{-7} & -3.5741\times10^{-7} \\ -3.6456\times10^{-5} & -2.5369\times10^{-7} & 3.0037\times10^{-7} & 6.3871\times10^{-7} \\ -7.7729\times10^{-5} & -3.5741\times10^{-7} & 6.3871\times10^{-7} & 1.3623\times10^{-6} \end{bmatrix}.$$

Separation principle for LQG design

When the optimal filter signal $\hat{\boldsymbol{x}}(t)$ is obtained, the block diagram of the LQG compensator can be constructed, as shown in Figure 7.2, with the optimal control $\boldsymbol{u}^*(t)$ given by

$$\boldsymbol{u}^*(t) = -\boldsymbol{K}_c\hat{\boldsymbol{x}}(t) \tag{7.6}$$

and the optimal state feedback matrix $\boldsymbol{K}_c$ given by

$$\boldsymbol{K}_c = \boldsymbol{R}^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{P}_c, \tag{7.7}$$

where the symmetrical semipositive-definite matrix satisfies the following ARE:

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{P}_c + \boldsymbol{P}_c\boldsymbol{A} - \boldsymbol{P}_c\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{P}_c + \boldsymbol{M}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{M} = \boldsymbol{0}. \tag{7.8}$$

From the above discussions, we can observe that, in the LQG optimal control problem, the optimal estimation and optimal control problems are solved separately. This is the well-known "separation principle." That is, to design an LQG controller, one can first design a state estimator and then use the estimated states, as if the states are exactly measurable, to design the LQR state feedback controller.

Observer-based LQG controller

For the state space plant model

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) + \boldsymbol{\xi}(t), \\ \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) + \boldsymbol{\theta}(t) \end{cases} \tag{7.9}$$

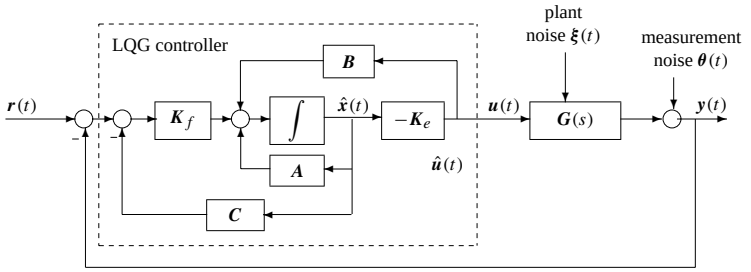


Figure 7.2. *LQG control structure.*

and the optimization criterion

$$J = \lim_{t_f\to\infty} \mathrm{E}\left\{\int_0^{t_f} [\boldsymbol{x}^{\mathrm{T}}\ \boldsymbol{u}^{\mathrm{T}}] \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{N}_c \\ \boldsymbol{N}_c^{\mathrm{T}} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{bmatrix} \mathrm{d}t\right\}, \tag{7.10}$$

where $\boldsymbol{N}_c$ can normally be selected as a zero matrix, the observer-based LQG controller is illustrated in Figure 7.3. Suppose that the state feedback gain matrix $\boldsymbol{K}_c$ and the Kalman filter gain matrix $\boldsymbol{K}_f$ have been obtained via the separation principle. Then, the Kalman filter dynamic equation is written as

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{K}_f(\boldsymbol{y} - \boldsymbol{C}\hat{\boldsymbol{x}} - \boldsymbol{D}\boldsymbol{u}). \tag{7.11}$$

So, the observer-based LQG controller can be compactly formulated as follows:

$$\boldsymbol{G}_c(s) = \left[\begin{array}{c|c} \boldsymbol{A} - \boldsymbol{K}_f\boldsymbol{C} - \boldsymbol{B}\boldsymbol{K}_c + \boldsymbol{K}_f\boldsymbol{D}\boldsymbol{K}_c & \boldsymbol{K}_f \\ \hline \boldsymbol{K}_c & \boldsymbol{0} \end{array}\right]. \tag{7.12}$$

A MATLAB function `lqg()` provided in the Robust Control Toolbox can be used to design an observer-based LQG controller. The syntax of the function is

```
Gc = -lqg(G,W,V)
[Af,Bf,Cf,Df] = lqg(A,B,C,D,W,V)
```

where $(\boldsymbol{A}_f,\boldsymbol{B}_f,\boldsymbol{C}_f,\boldsymbol{D}_f)$ is the state space model of the LQG controller $\boldsymbol{G}_c(s)$. Here, $\boldsymbol{W}$ and $\boldsymbol{V}$ can be constructed as follows:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{N}_c \\ \boldsymbol{N}_c^{\mathrm{T}} & \boldsymbol{R} \end{bmatrix}, \quad \boldsymbol{V} = \begin{bmatrix} \boldsymbol{\Xi} & \boldsymbol{N}_f \\ \boldsymbol{N}_f^{\mathrm{T}} & \boldsymbol{\Theta} \end{bmatrix}, \tag{7.13}$$

where $\boldsymbol{\Xi}$ and $\boldsymbol{\Theta}$ are, respectively, the covariances of the plant noise $\boldsymbol{\xi}(t)$ and measurement noise $\theta(t)$, with $\boldsymbol{N}_c$ and $\boldsymbol{N}_f$ often assumed to be zero matrices. It can be easily seen that the matrix $\boldsymbol{V}$ is in fact the joint correlation function of signals $\boldsymbol{\xi}(t)$ and $\boldsymbol{\theta}$ with

$$\mathrm{E}\left\{\begin{bmatrix} \boldsymbol{\xi}(t) \\ \boldsymbol{\theta}(\tau) \end{bmatrix} [\boldsymbol{\xi}(t)\ \boldsymbol{\theta}(\tau)]^{\mathrm{T}}\right\} = \begin{bmatrix} \boldsymbol{\Xi} & \boldsymbol{N}_f \\ \boldsymbol{N}_f^{\mathrm{T}} & \boldsymbol{\Theta} \end{bmatrix} \delta(t-\tau). \tag{7.14}$$

Note that $\boldsymbol{\Xi}$ is the covariance of $\boldsymbol{\xi}(t)$. If the plant model given in (7.1) is used, $\boldsymbol{\Xi}$ should be replaced by $\boldsymbol{\Gamma}\boldsymbol{\Xi}\boldsymbol{\Gamma}^{\mathrm{T}}$.

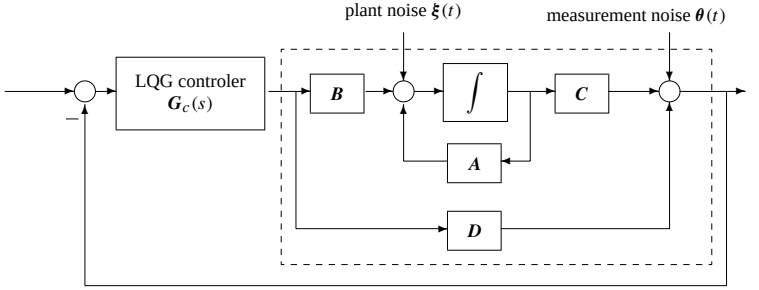


Figure 7.3. *Observer-based LQG control structure.*

Example 7.2. Consider the following system with Gaussian noises ξ and θ:

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -5000 & -100/3 & 500 & 100/3 \\ 0 & -1 & 0 & 1 \\ 0 & 100/3 & -4 & -60 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 25/3 \\ 0 \\ -1 \end{bmatrix} u + \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xi(t),$$
$$y(t) = [0, 0, 1, 0]\boldsymbol{x} + \theta(t)$$

with covariances $\Xi = 7 \times 10^{-4}$ and $\Theta = 10^{-8}$. Taking the weighting functions $\boldsymbol{Q} = diag(5000, 0, 50000, 1)$ and $R = 0.001$, the LQG problem can be solved using the following MATLAB statements:

```
>> A=[0,1,0,0;-5000,-100/3,500,100/3;0,-1,0,1;0,100/3,-4,-60];
   B=[0; 25/3; 0; -1];  C=[0,0,1,0]; D=0; G=[-1; 0; 0; 0];
   Q=diag([5000,0,50000,1]); R=0.001; Sys=ss(A,B,C,D);
   Xi=7e-4; Theta=1e-8; W=[Q,zeros(4,1); zeros(1,4),R];
   V=[Xi*G*G',zeros(4,1); zeros(1,4),Theta];
   Gf=-lqg(Sys,W,V); Gc=zpk(Gf)
```

and the controller can be designed as

$$G_c(s) = \frac{-1231049.0702(s + 40.47)(s^2 + 105.5s + 5000)}{(s^2 + 39.17s + 868.2)(s^2 + 493.9s + 1.234 \times 10^5)}.$$

Using the above designed LQG controller, the closed-loop step response of the system, with the random signals neglected, can be obtained as shown in Figure 7.4(a) using the following MATLAB statements:

```
>> step(feedback(Gc*Sys,1)), figure, bode(Sys,':',Gc*Sys,'-')
   [Gm,Pm,Wcg,Wcp]=margin(Sys*Gc)
```

The gain and phase margins are, respectively, 4.3730 and 43.0440°, at frequencies 323.2318 and 125.1567 rad/sec. The open-loop Bode diagrams for both the original and the compensated systems are shown in Figure 7.4(b). It can be seen that the closed-loop behavior is significantly improved with the LQG controller.

Now, let us assign R with different values denoted by ρ. A series of LQG controllers can be designed using the following MATLAB statements:

```
>> G=ss(A,B,C,D); f1=figure; f2=figure;
   for rho=[100,10,1,0.1,0.01,0.001]
      W=[Q,zeros(4,1); zeros(1,4),rho*R]; G=ss(A,B,C,D);
      Gc=-lqg(G,W,V);figure(f1),step(feedback(G*Gc,1),0.5),hold on
      figure(f2), bode(G*Gc,{0.1,10000}); hold on;
   end
```

with the closed-loop step responses and open-loop Bode diagrams compared in Figures 7.5(a) and (b), respectively. We can observe that when ρ decreases, the dynamic behavior of the closed-loop system improves. Meanwhile, the gain and phase margins, as well as the crossover frequencies, tend to increase, which indicates the improvements in the dynamic behavior of the controlled system.

7.1.3 LQG Control with Loop Transfer Recovery

LQG/LTR design algorithms

The story sounds good so far: with Kalman filters, the optimal LQG control design amounts to solving two independent Riccati equations, (7.5) and (7.8), which, with MATLAB, is a fairly easy task.

However, things are never as simple as that. It has been pointed out in [80] that the controller thus designed may have very small stability margins, implying that if the system is subjected to very small disturbance, the overall system may become unstable.

There is a seemingly correct intuition in control engineering practice that the dynamics of the signal filtering block should be much faster than the plant dynamics. It would therefore appear that if the dynamics of the Kalman filter were made fast, a satisfactory design would be achieved. This is, unfortunately, not true. We will show, through analysis and examples, that the LQG controller may not increase the stability margin of the overall system, but can significantly reduce it.

Suppose that accurate state measurement is possible, as in the LQR case. With the optimal LQR controller, the open-loop transfer function is simply $\boldsymbol{G}_{LQSF}(s) = \boldsymbol{K}_c(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}$. However, under LQG control, the open-loop transfer function becomes

$$\boldsymbol{G}_{L,LQG}(s) = \boldsymbol{K}_c(s\boldsymbol{I} - \boldsymbol{A} + \boldsymbol{B}\boldsymbol{K} + \boldsymbol{L}\boldsymbol{C})^{-1}\boldsymbol{L}\boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}. \tag{7.15}$$

The following example demonstrates the difference between $\boldsymbol{G}_{LQSF}(s)$ and $\boldsymbol{G}_{L,LQG}(s)$.

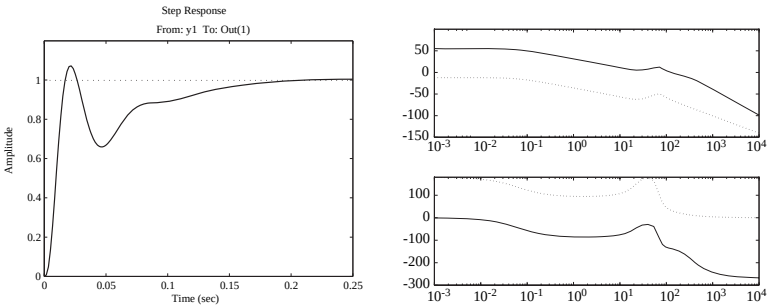
Example 7.3. Consider the plant

$$G(s) = \frac{-(948.12s^3 + 30325s^2 + 56482s + 1215.3)}{s^6 + 64.554s^5 + 1167s^4 + 3728.6s^3 - 5495.4s^2 + 1102s + 708.1}.$$

Its state space model can be obtained by the following MATLAB statements:

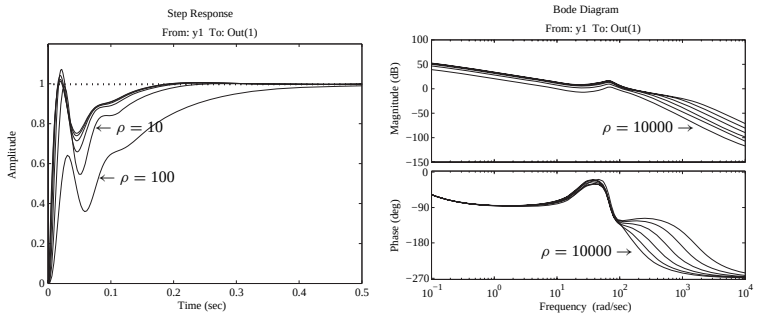
```
>> num=-[948.12, 30325, 56482, 1215.3];
   den=[1,64.554,1167,3728.6,-5495.4,1102,708.1];G=ss(tf(num,den));
```

this model is the state space object G.



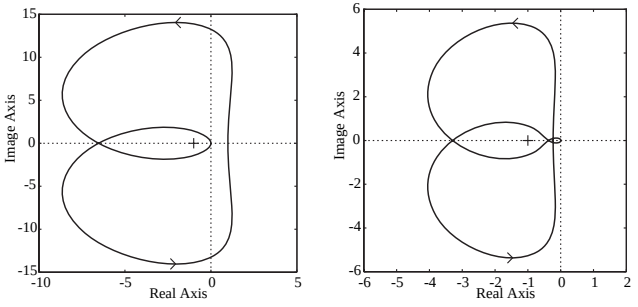
(a) step response (b) Bode diagram

Figure 7.4. *System responses under LQG control.*



(a) step response (b) Bode diagram

Figure 7.5. *Closed-loop system responses with different ρ.*



(a) $G_{\mathrm{L}QSF}(s)$ (b) $G_{\mathrm{L},LQG}(s)$

Figure 7.6. *Open-loop Nyquist plot comparison.*

With the weighting matrices $\boldsymbol{Q} = \boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}$ and $R = 1$, the optimal LQ controller can be obtained, and the open-loop Nyquist plot of the system $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{K}, 0)$ can be drawn, as shown in Figure 7.6(a), using the following MATLAB statements:

```
>> Q=G.c'*G.c;R=1;[Kc,P]=lqr(G.a,G.b,Q,R);
   nyquist(ss(G.a,G.b,Kc,0))
```

It can be seen that the original system is closed-loop stable since there are two encirclements around the point $(-1, \mathrm{j}0)$, which equals the number of unstable poles in the open-loop model.

If Gaussian noises are present in the system, the $\boldsymbol{\Gamma}$ vector is defined as $\boldsymbol{\Gamma} = \boldsymbol{B}$, and $\boldsymbol{\Xi} = 10^{-4}$ and $\Theta = 10^{-5}$, then a Kalman filter can be obtained using the following MATLAB statements:

```
>> Xi=1e-4; Theta=1e-5; G1=ss(G.a,[G.b, G.b],G.c,[G.d,G.d]);
   [K_Sys,L,P2]=kalman(G1,Xi,Theta); a1=G.a-G.b*Kc-L*G.c;
   G_o=G*ss(a1,L,Kc,0); Nyquist(G_o), [Gm,Pm,Wcg,Wcp]=margin(G_o)
```


The gain and phase margins are 2.6882 and 33.7375°, at frequencies 10.8799 and 4.4401 rad/sec, respectively. The resulting Nyquist plot of $G_{\mathrm{L},LQG}(s)$, which can be regarded as two subsystems $(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K} - \boldsymbol{L}\boldsymbol{C}, \boldsymbol{L}, \boldsymbol{K}, 0)$ and $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, 0)$ in series connections, is different from that of $G_{\mathrm{L}QSF}(s)$, as shown in Figure 7.6.

Clearly, if the weighting functions are not suitably chosen, there will be a difference, possibly large, between the open-loop transfer functions as demonstrated in Figure 7.6. To effectively reduce this difference, use the loop transfer recovery (LTR) technique. The basic idea is to make the loop transfer function in the LQG structure approach as closely as possible that using the direct full state feedback.

Let $\boldsymbol{\Xi}' = q\,\boldsymbol{\Xi}$. It can be shown that when $q \to \infty$, the open-loop transfer function of the LQG control problem with $\boldsymbol{\Xi}'$ will approach that for the LQR problem, i.e.,

$$\lim_{q\to\infty} \boldsymbol{K}_c(s\boldsymbol{I} - \boldsymbol{A} + \boldsymbol{B}\boldsymbol{K} + \boldsymbol{L}\boldsymbol{C})^{-1}\boldsymbol{L}\boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} = \boldsymbol{K}_c(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}. \tag{7.16}$$

Obviously, the key point in LQG/LTR controller design is to select a large q.

Alternatively, one can first solve the standard LQR problem to get a suitable state feedback gain vector and then use the LTR technique to make the final system with the open-loop transfer function, including the Kalman filter, approach that of the LQR system as closely as possible. This leads to the following two-step algorithm:

- Design an optimal LQR controller with the specified weighting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, and adjust the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ such that the open-loop transfer function $-\boldsymbol{K}_c(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}$ is satisfactory. A common practice is to set $\boldsymbol{Q} = \boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}$ and change $\boldsymbol{R}$ to make the open-loop transfer function close to a target transfer function, with the sensitivity and complementary sensitivity functions having the desired shapes.
- Set $\boldsymbol{\Gamma} = \boldsymbol{B}$, $\boldsymbol{W} = \boldsymbol{W}_0 + q\boldsymbol{I}$, and $\boldsymbol{V} = \boldsymbol{I}$. Increase the value of q so that the return difference of the compensated system approaches $-\boldsymbol{K}_c(\mathrm{j}\omega\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}$. With the selected q, the observer Riccati equation is then changed to

$$\frac{\boldsymbol{P}_f\boldsymbol{A}^{\mathrm{T}}}{q} + \frac{\boldsymbol{A}\boldsymbol{P}_f}{q} - \frac{\boldsymbol{P}_f\boldsymbol{C}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{C}\boldsymbol{P}_f}{q} + \frac{\boldsymbol{\Gamma}\boldsymbol{W}_0\boldsymbol{\Gamma}^{\mathrm{T}}}{q} + \boldsymbol{\Gamma}\boldsymbol{\Theta}\boldsymbol{\Gamma}^{\mathrm{T}} = \boldsymbol{0}, \tag{7.17}$$

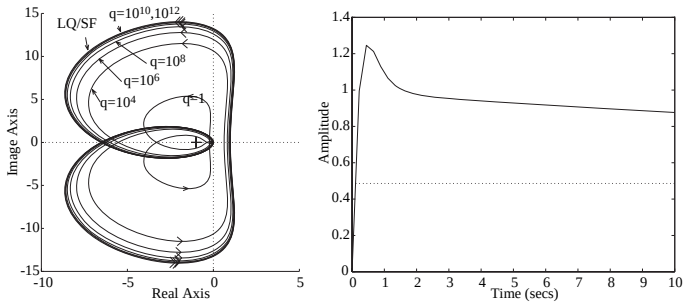
  where q is referred to as the fictitious-noise coefficient. When the original system $\boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}$ has no transmission zeros on the right-hand side of the s-plant, the filter gain matrix can then be evaluated from

$$\boldsymbol{K}_f \to q^{1/2}\boldsymbol{B}\boldsymbol{V}^{-1/2} \text{ when } q \to \infty. \tag{7.18}$$

  In practice, q should not be too large. Too large a q will introduce numerical truncating errors, which may in turn affect the robustness of the overall system.

Example 7.4. Consider again the problem in Example 7.3. Let us apply the LTR technique with different values of q using the following MATLAB statements:

```
>> num=-[948.12, 30325, 56482, 1215.3]; marg=[];
   den=[1,64.554,1167,3728.6,-5495.4,1102,708.1];G=ss(tf(num,den));
   Xi=1e-4; Theta=1e-5; Q=G.c'*G.c; R=1; [Kc,P]=lqr(G.a,G.b,Q,R);
```



(a) Nyquist plots (b) closed-loop step response

Figure 7.7. *LQG/LTR results.*

```
nyquist(ss(G.a,G.b,Kc,0)), hold on
for q0=[1,1e4,1e6,1e8,1e10,1e12,1e14]
   G1=ss(G.a,[G.b, G.b],G.c,[G.d,G.d]);
   [K_Sys,L,P2]=kalman(G1,q0*Xi,Theta);
   a1=G.a-G.b*Kc-L*G.c; G_o=G*ss(a1,L,Kc,0); nyquist(G_o)
   [Gm,Pm,Wcg,Wcp]=margin(G_o); marg=[marg; [Gm,Pm,Wcg,Wcp]];
end
```

The Nyquist diagrams of the open-loop system for different values of q are shown in Figure 7.7(a). It can be seen that when $q = 10^{10}$, the loop transfer function can almost be recovered. For this case, the closed-loop step response, obtained using the following MATLAB statements, is shown in Figure 7.7(b).

```
>> q=1e10; [K_Sys,L,P2]=kalman(G1,q*Xi,Theta); a1=G.a-G.b*Kc-L*G.c;
   G_o=G*ss(a1,L,Kc,0); G_c=feedback(G_o,1);
   t=0:0.01:10; figure; step(G_c,t)
```

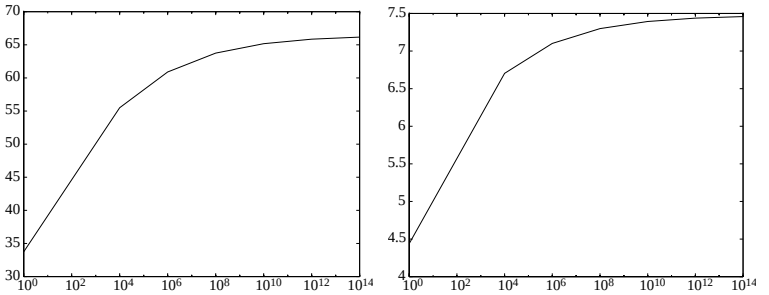
The phase margin and crossover frequency versus q plots, drawn using the following MATLAB statements:

```
>> q0=[1,1e4,1e6,1e8,1e10,1e12,1e14];
   semilogx(q0,marg(:,2)), figure, semilogx(q0,marg(:,4))
```

are compared in Figure 7.8(a) and (b), respectively. It can be seen that by the LQG/LTR controller, the phase margin and crossover frequency are significantly increased. Further increasing q will not contribute much to the gain margin and the crossover frequency. For this example, it is enough to set $q = 10^{10}$.

Linear quadratic Gaussian/loop transfer recovery problem solution using MATLAB

Two functions, `ltru()` and `ltry()`, are provided in the Robust Control Toolbox for effective LQG/ LTR controller design.



(a) phase margin plots (b) crossover frequency plots

Figure 7.8. *The phase margin and crossover frequency versus q.*

The function `ltru()` performs the LTR design at the input of the plant model, while `ltry()` does so at the output. Recovering the loop transfer function at the input side means that

$$\lim_{q\to\infty} \boldsymbol{\Gamma}\boldsymbol{K}_c(s\boldsymbol{I}-\boldsymbol{A}+\boldsymbol{B}\boldsymbol{K}_c+\boldsymbol{K}_f\boldsymbol{C})^{-1}\boldsymbol{K}_f = \boldsymbol{K}_c(s\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{B}. \tag{7.19}$$

The recovery of the loop transfer function at the output implies that

$$\lim_{q\to\infty} \boldsymbol{\Gamma}\boldsymbol{K}_c(s\boldsymbol{I}-\boldsymbol{A}+\boldsymbol{B}\boldsymbol{K}_c+\boldsymbol{K}_f\boldsymbol{C})^{-1}\boldsymbol{K}_f = \boldsymbol{C}(s\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{K}_f. \tag{7.20}$$

The syntax of function `ltru()` is

```
Gc=ltru(G,Kc,Ξ,Θ,q,ω);
[Af,Bf,Cf,Df]=ltru(A,B,C,D,Kc,Ξ,Θ,q,ω);
```

where $\boldsymbol{q}$ is a vector containing the selected values of q for the LTR process. The loop transfer function can then be expressed as $\boldsymbol{G}_c(s)\boldsymbol{C}(s\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{B}$. In the function call, the Nyquist plots of the open-loop transfer function for different specified q will be displayed automatically.

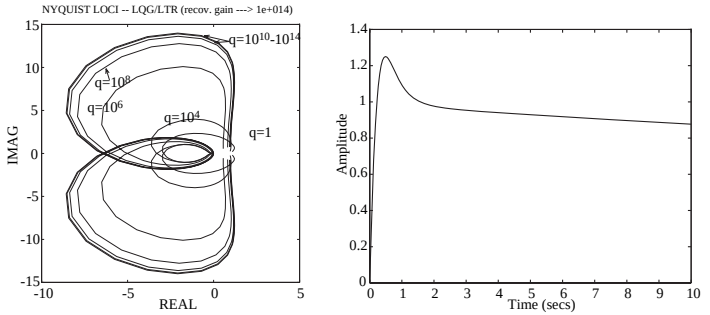
Similarly, the syntax for the `ltry()` function is

```
Gc=ltry(G,Kf,Q,R,q,ω)
[Af,Bf,Cf,Df]=ltry(A,B,C,D,Kf,Q,R,q,ω)
```

where $\boldsymbol{K}_f$ is the Kalman filter gain matrix.

Example 7.5. Consider again the plant model in Example 7.3. With different values of q, the corresponding LTR controller can be designed using the following MATLAB statements:

```
>> num=-[948.12, 30325, 56482, 1215.3];
   den=[1, 64.554, 1167, 3728.6, -5495.4, 1102, 708.1];
   G=ss(tf(num,den)); Xi=1e-4; Theta=1e-5;
```



(a) LTR for different q (b) closed-loop step response

Figure 7.9. *LQG/LTR control results.*

```
Q=G.c'*G.c; R=1; [Kc,P]=lqr(G.a,G.b,Q,R);
q0=[1,1e4,1e6,1e8,1e10,1e12,1e14];  w=logspace(-2,2);
Gc=ltru(G,Kc,Xi,Theta,q0,w); Gc=zpk(Gc))
```

and the controller designed is

$$G(s)=\frac{-22026697072.6516(s+30.22)(s+29.71)(s+6.763)(s+1.315)(s+0.01261)}{(s+1.445\times10^4)(s+30)(s+1.963)(s+0.0218)(s^2+1.444\times10^4 s+2.087\times10^8)}.$$

The Nyquist plots for different values of q are compared in Figure 7.9(a). It can be observed once again that when q is relatively large, for instance, $q > 10^{10}$, the loop transfer at the input side of the plant approaches the LQR solutions asymptotically.

The closed-loop step response under the LQG/LTR controller can be obtained using the following MATLAB statements:

```
>> q=10e10; Gc=ltru(G,Kc,Xi,Theta,q,w);
   G_c=feedback(G*Gc,1); figure; step(G_c,10)
```

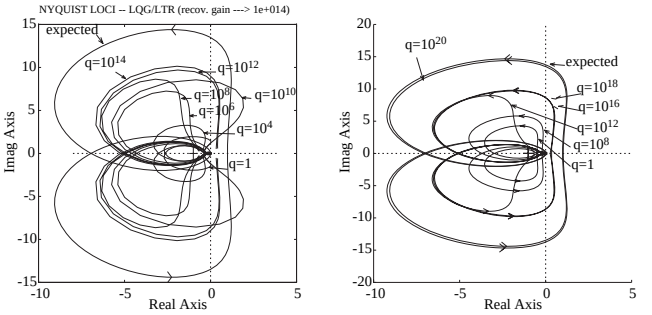
and is shown in Figure 7.9(b). It can be seen that the response is similar to the one shown in Example 7.4.

Example 7.6. Now, let us consider a nonminimum phase unstable plant model

$$G(s)=\frac{-948.12s^3+30325s^2-56482s-1215.3}{s^6+64.554s^5+1167s^4+3728.6s^3-5495.4s^2+1102s+708.1}$$

with $\mathbf{\Gamma}=\boldsymbol{B}$, $\Xi=10^{-4}$, $\Theta=10^{-5}$. To design an LQG/LTR controller, the following MATLAB statements can be issued:

```
>> num=[-948.12, 30325, 56482, -1215.3];
   den=[1,64.554,1167,3728.6,-5495.4,1102,708.1];G=ss(tf(num,den));
   Xi=1e-4; Th=1e-5; Q=diag([100,10,20,30,40,100]); R=1;
   [Kc,P]=lqr(G.a,G.b,Q,R); q0=[1,1e4,1e6,1e8,1e10,1e12,1e14];
   w=logspace(-2,2); Gc=ltru(G,Kc,Xi,Th,q0,w); hold on; nyquist(G);
```

(a) results of `ltru()` function (b) LTR design results

Figure 7.10. *LQG/LTR controller design using different methods.*

The Nyquist plots for different values of q are shown in Figure 7.10(a). It can be seen that when q is increased to a certain value, say, 10^{12}, the algorithm does not consistently converge to the expected one, i.e., the LQR solution. It can also be seen that even when q increases further, the converged shape of the Nyquist plot is in fact far away from the expected loop transfer function. Thus, we may conclude that the loop recovery is not achievable in this example.

We can perform a similar LTR design without using the `ltru()` function. Here, let us use the simple computation procedures introduced in Sec. 7.1.3 by the following MATLAB statements:

```
>> nyquist(ss(G.a,G.b,Kc,0)); hold on;
   for q0=[1,1e8,1e12,1e16,1e18,1e20]
      G1=ss(G.a,[G.b,G.b],G.c,[G.d,G.d]);
      [K_Sys,L,P2]=kalman(G1,q0*Xi,Theta);
      a1=G.a-G.b*Kc-L*G.c; Gc=ss(a1,L,Kc,0); G_o=G*Gc; nyquist(G_o)
   end
```

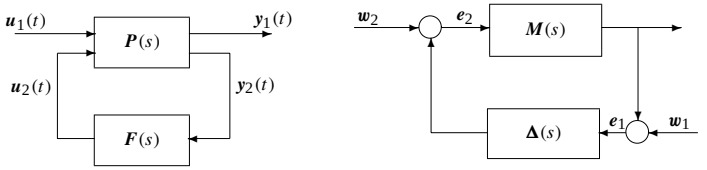
The resulting Nyquist diagrams for different values of q are shown in Figure 7.10(b). Interestingly, when q approaches 10^{20}, a satisfactory LTR can be achieved which is different than what is shown in Figure 7.10(a). This indicates that `ltru()` may not be numerically reliable when q is too large. Note that, however, in this example, with $q=10^{20}$ the closed-loop system may not be stable.

7.2 General Descriptions of the Robust Control Problems

The small gain theorem plays a pivot role in robust control. In this section, we will first briefly introduce the small gain theorem and the uncertainty description. Then, the robust controller structures and the model representation in MATLAB will be discussed.

7.2.1 Small Gain Theorem

A general description of robust control system structure is shown in Figure 7.11(a), where $\boldsymbol{P}(s)$ is the augmented plant model and $\boldsymbol{F}(s)$ is the controller model. The transfer function



(a) standard feedback control (b) sketch for small gain theorem

Figure 7.11. *General structure of $\mathcal{H}_2$ and $\mathcal{H}_\infty$ control.*

from the input $\boldsymbol{u}_1(t)$ to the output $\boldsymbol{y}_1(t)$ is denoted by $\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}(s)$. It should be emphasized at this point that the block diagram shown in Figure 7.11(a) is fairly general. The signal vector $\boldsymbol{u}_1(t)$ can include both reference and disturbance signals. $\boldsymbol{P}(s)$ can include both the plant model and the disturbance generation model. Moreover, uncertainties can also be included in $\boldsymbol{P}(s)$. The key idea of robust control is to separate the known part and the uncertain part from the knowledge about the uncertain system under investigation. This is illustrated in Figure 7.11(b), where $\boldsymbol{M}(s)$ denotes the known part of the uncertain system and $\boldsymbol{\Delta}(s)$ denotes the uncertain part. Usually, we have some limited knowledge about $\boldsymbol{\Delta}(s)$ such as the upper bound information. Note that $\boldsymbol{M}(s)$, after some transformations from Figure 7.11(a), contains both the plant and the controller. By designing $\boldsymbol{F}(s)$, we can change $\boldsymbol{M}(s)$. The bottom line is how to design $\boldsymbol{F}(s)$ such that the overall system is stable for all possible $\boldsymbol{\Delta}(s)$. This is the so-called small gain theorem summarized below.

Theorem 7.1 (*small gain theorem*). Suppose that $\boldsymbol{M}(s)$ is stable and let $\gamma > 0$. The interconnected system shown in Figure 7.11(b) is well-posed and internally stable for all stable $\boldsymbol{\Delta}(s)$ if the small gain condition

$$\|\boldsymbol{M}(s)\|_\infty \|\boldsymbol{\Delta}(s)\|_\infty < 1 \tag{7.21}$$

is satisfied.

Clearly, if we know that $\|\boldsymbol{\Delta}(s)\|_\infty < \gamma$, we should properly design $\boldsymbol{F}(s)$ to ensure $\|\boldsymbol{M}(s)\|_\infty < 1/\gamma$ such that the overall system is robustly stable, according to the small gain theorem.

7.2.2 Unstructured Uncertainties

The unstructured uncertainties can be classified into the additive and multiplicative uncertainties. The feedback system structure with uncertainties is shown in Figure 7.12. In general, the uncertain model can be represented by

$$\boldsymbol{G}_p(s) = \boldsymbol{\Delta}_A(s) + \boldsymbol{G}(s)[\boldsymbol{I} + \boldsymbol{\Delta}_M(s)]. \tag{7.22}$$

If $\boldsymbol{\Delta}_A(s) \equiv \boldsymbol{0}$, one has $\boldsymbol{G}_p(s) = \boldsymbol{G}(s)[\boldsymbol{I} + \boldsymbol{\Delta}_M(s)]$, and the uncertainty is referred to as the multiplicative uncertainty. When $\boldsymbol{\Delta}_M(s) \equiv \boldsymbol{0}$, the uncertainty is referred to as the additive uncertainty with the model $\boldsymbol{G}_p(s) = \boldsymbol{G}(s) + \boldsymbol{\Delta}_A(s)$.

Based on the discussions of the small gain theorem in the last subsection, starting from here, with no loss of generality, if we assume that the uncertainty norm bound shown

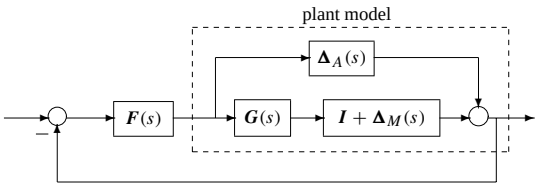


Figure 7.12. *Feedback control with uncertainties.*

in Figure 7.11(b) is unity, then we can concentrate on Figure 7.11(a) as if there were no uncertainty in $\boldsymbol{P}(s)$. But now our control design task amounts to designing $\boldsymbol{F}(s)$ such that $\|\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}(s)\|_\infty < 1$. We should understand that it is always possible to scale $\boldsymbol{\Delta}(s)$ such that the scaled uncertainty bound is less than 1.

In what follows, we shall focus on the robust control problem based on Figure 7.11(a). Our robust controller design task is simply to make the norm of $\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}(s)$ small. The popular measures of the smallness are the $\mathcal{H}_\infty$-norm and the $\mathcal{H}_2$-norm. This leads to the two popular robust controllers, the $\mathcal{H}_\infty$ controller and the $\mathcal{H}_2$ controller, which are the main topics of this chapter. However, at this point, we should emphasize that different measures will lead to different robust control design methods or even different frameworks. For example, the μ-synthesis technique, not covered in this book, is one of the other alternatives. For more complete coverage of robust control, we refer to [81]. This chapter serves only as an entry-level introduction to robust control techniques.

7.2.3 Robust Control Problems

Based on the above arguments, we now focus on the configuration shown in Figure 7.11(a), where an augmented plant model can be constructed as

$$\boldsymbol{P}(s) = \left[\begin{array}{c:cc} \boldsymbol{A} & \boldsymbol{B}_1 & \boldsymbol{B}_2 \\ \hdashline \boldsymbol{C}_1 & \boldsymbol{D}_{11} & \boldsymbol{D}_{12} \\ \boldsymbol{C}_2 & \boldsymbol{D}_{21} & \boldsymbol{D}_{22} \end{array}\right] \tag{7.23}$$

with the augmented state space description as follows:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + [\boldsymbol{B}_1\ \boldsymbol{B}_2]\begin{bmatrix}\boldsymbol{u}_1 \\ \boldsymbol{u}_2\end{bmatrix}, \quad \begin{bmatrix}\boldsymbol{y}_1 \\ \boldsymbol{y}_2\end{bmatrix} = \begin{bmatrix}\boldsymbol{C}_1 \\ \boldsymbol{C}_2\end{bmatrix}\boldsymbol{x} + \begin{bmatrix}\boldsymbol{D}_{11} & \boldsymbol{D}_{12} \\ \boldsymbol{D}_{21} & \boldsymbol{D}_{22}\end{bmatrix}\begin{bmatrix}\boldsymbol{u}_1 \\ \boldsymbol{u}_2\end{bmatrix}. \tag{7.24}$$

Straightforward manipulations give the following closed-loop transfer function:

$$\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}(s) = \boldsymbol{P}_{11}(s) + \boldsymbol{P}_{12}(s)[\boldsymbol{I} - \boldsymbol{F}(s)\boldsymbol{P}_{22}(s)]^{-1}\boldsymbol{F}(s)\boldsymbol{P}_{21}(s). \tag{7.25}$$

The above expression is also known as the linear fractional transformation (LFT) of the interconnected system. The objective of robust control is to find a stabilizing controller $\boldsymbol{u}_2(s) = \boldsymbol{F}(s)\boldsymbol{y}_2(s)$ such that $\|\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}(s)\| < 1$. Based on (7.25), the following three robust control problems are particularly interesting in control engineering practice:

- The $\mathcal{H}_2$ optimal control problem: $\min\limits_{\boldsymbol{F}(s)} \|\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}\|_2$;
- the $\mathcal{H}_\infty$ optimal control problem: $\min\limits_{\boldsymbol{F}(s)} \|\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}\|_\infty$;
- the standard $\mathcal{H}_\infty$ robust control: $\|\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}\|_\infty < 1$.

7.2.4 Model Representation Under MATLAB

A MATLAB function `mksys()`, provided in the Robust Control Toolbox, can be used to describe the system model with a single variable name. The syntax of the function is `S=mksys(A,B,C,D)`, where $(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{D})$ is the state space model of the system. The model variable S is also referred to as the tree variable in the Robust Control Toolbox. The tree S can then be used directly throughout the robust control analysis and design functions in the Robust Control Toolbox. Generally, `mksys()` can be called in the format `S=mksys(`$V_1, V_2, \ldots, V_N$`, TY)`, where $(V_1, V_2, \ldots, V_N)$ are the model parameters of different types identified by the argument `TY`. The details of these variables are listed in Table 7.1. Note that the `'tss'` format uses the representation in (7.23). The descriptor system takes a more general state space representation described as

$$\begin{cases} \boldsymbol{E}\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u, \\ y = \boldsymbol{C}\boldsymbol{x} + Du, \end{cases} \tag{7.26}$$

where matrix $\boldsymbol{E}$ can be either singular or nonsingular; in the former case, the system is referred to as the singular system. The details of the singular system will not be covered in this book.

Apart from the model descriptions given in Table 7.1, some other model formats are allowed, such as the multivariable transfer function model, impulse response model, and the generalized state space model.

Example 7.7. Consider a two input–two output (TITO) state space model

$$\dot{\boldsymbol{x}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -3 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 & 0 \\ 2 & 3 \\ -3 & -3 \end{bmatrix} \boldsymbol{u}, \quad y = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \boldsymbol{x}.$$

The model can be entered into MATLAB with the following statements and then packed into a single tree variable `S`:

```
>> A1=[-1,0,0; 0,-2,0; 0,0,-3]; B1=[1,0; 2,3; -3,-3];
   C1=[1,0,0; 1,1,1]; D1=zeros(2,2); S=mksys(A1,B1,C1,D1,'ss');
```

For a given transfer function model

$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24},$$

Table 7.1. *Variable description of the* `mksys` *function.*

`TY`	Reserved names of $V_1, V_2, \ldots, V_N$	Description
`'ss'`	`a, b, c, d`	Standard state space (default)
`'des'`	`a, b, c, d, e`	Descriptor system
`'tss'`	`a, b1, b2, c1, c2, d11, d12, d21, d22`	Two-port state space
`'tf'`	`num, den`	Transfer function

it can be packed into a tree variable `S1` using the following MATLAB statements:

```
>> num=[1,7,24,24]; den=[1,10,35,50,24]; S1=mksys(num,den,'tf');
```

A MATLAB function `branch()` is provided to retrieve the variables from a given model (tree) variable S, `[`$V_1, V_2, \ldots, V_N$`]=branch(S)`, where S is the existing tree variable and $V_1, V_2, \ldots, V_N$ are the variables packed into S, as shown in Table 7.1. For instance, the state space model can be retrieved from the tree structure S as follows: `[a,b,c,d]=branch(S)`. For the tree structure S_1, the transfer function parameters can be retrieved from `[n,d]=branch(`S_1`)`.

With tree variable structures to represent the plant models, we can call some of the functions in the Robust Control Toolbox in alternative formats. For instance, `ltru()` and `ltry()` can be called in the following way:

```
Gf=ltru(G,Kc,Ξ,Θ,r,ω,svk)
Gf=ltry(G,Kf,Q,R,q,ω,svk)
```

where G and G_f are the tree variables for the plant and the controller, respectively.

Moreover, the individual parameter of the state space model can be retrieved by using the same `branch()` function. For instance, the $\boldsymbol{a}$ and $\boldsymbol{c}$ matrices within the tree variable S can be retrieved using the MATLAB statement `[a,c]=branch(S,'a,c')`. It should be noted that although the original names $\boldsymbol{A}_1$, $\boldsymbol{C}_1$ were used in packing the tree variable S, one still needs to use $\boldsymbol{a}$, $\boldsymbol{c}$ to retrieve them from the tree.

7.2.5 Dealing with Poles on the Imaginary Axis

If the plant model contains purely imaginary poles, the robust control design techniques cannot be used directly. In this case, a new variable z can be introduced with $s = (\alpha z + \delta)/(\gamma z + \beta)$. Replace the s term in the plant model and transform it into a transfer function of z. This transformation is referred to as the bilinear transformation, also known as the frequency domain or complex plane bilinear transformation.

With the bilinear transformation, the poles will be shifted away from the imaginary axis. The basic idea is to take the shifted model as the new plant model and design a robust controller based on it. Suppose that a controller $\boldsymbol{F}(z)$ is designed. Then, one can set $z = (-\beta s + \delta)/(\gamma s + \alpha)$ and substitute the z term in the controller $\boldsymbol{F}(z)$ to retrieve the corresponding controller $\boldsymbol{G}_c(s)$ for the system.

The bilinear transformation can simply be represented as

$$\boldsymbol{G}(s) = \boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D} \;\Rightarrow\; \boldsymbol{G}(z) = \boldsymbol{C}_b(z\boldsymbol{I} - \boldsymbol{A}_b)^{-1}\boldsymbol{B}_b + \boldsymbol{D}_b, \tag{7.27}$$

and the state space description $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ under bilinear transformation can be written as

$$\left[\begin{array}{c|c} (\beta\boldsymbol{A} - \delta\boldsymbol{I})(\alpha\boldsymbol{I} - \gamma\boldsymbol{A})^{-1} & (\alpha\beta - \gamma\delta)(\alpha\boldsymbol{I} - \gamma\boldsymbol{A})^{-1}\boldsymbol{B} \\ \hline \boldsymbol{C}(\alpha\boldsymbol{I} - \gamma\boldsymbol{A})^{-1} & \boldsymbol{D} + \gamma\boldsymbol{C}(\alpha\boldsymbol{I} - \gamma\boldsymbol{A})^{-1}\boldsymbol{B} \end{array}\right]. \tag{7.28}$$

Table 7.2. *Available bilinear transformation methods.*

Method	Method name	Mathematical	Variables `aug`
`'Tustin'`	Tustin Transform	$s = \frac{2(z-1)}{T(z+1)}$	T
`'P_Tust'`	Prewarped Tustin	$s = \frac{w_0(z-1)}{\tan(w_0T/2)(z+1)}$	$[T, w_0]$
`'BwdRec'`	Backward Rectangular	$s = \frac{z-1}{Tz}$	T
`'FwdRec'`	Forward Rectangular	$s = \frac{z-1}{T}$	T
`'S_Tust'`	Shifted Tustin	$s = \frac{2(z-1)}{T(z/\gamma+1)}$	$[T, \gamma]$
`'G_Bili'`	General Bilinear	$s = \frac{\alpha z+\delta}{\gamma z+\beta}$	$[\alpha, \beta, \gamma, \delta]$

A MATLAB function `bilin()`, provided in the Robust Control Toolbox, can be used to perform the bilinear and the reversed bilinear transformations for a given system model. The syntax is as follows:

```
Sb=bilin(Sa,revs,method,aug)
[Ab,Bb,Cb,Db]=bilin(A,B,C,D,revs,method,aug)
```

where S_a is the tree structure or state space object, of the original model, and S_b is the bilinearly transformed model tree as explained in the above. The variable `revs` is used to describe the direction in the bilinear transformation, with `revs=1` for $s \Rightarrow z$ transformation (the default one), and -1 for $z \Rightarrow s$. The `method` describes the bilinear transformation method used as listed in Table 7.2.

Example 7.8. Consider a transfer function model

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

Note that there is a pole at $s = 0$. One can get a state space model from $G(s)$ first and then use the Tustin method to get the bilinear transformation under $T = 0.5$. From the new transfer function obtained using the following MATLAB statements:

```
>> s=tf('s'); G=10/s/(s+1)/(s+2); G=ss(G); T=0.5;
   Gf=bilin(G,1,'Tustin',T), eig(G), eig(Gf)
```

it can be seen that the poles of the original model G are $0, -1, -2$, with one of the poles on the imaginary axis. Taking Tustin's bilinear transformation, the poles of G_f are shifted to 1, 0.6000, 0.3333, all moved from the imaginary axis.

When calling `bilin()` with `revs` set to -1, the model can be shifted back:

```
>> G1=bilin(Gf,-1,'Tustin',T); G2=tf(G1)
```


7.3 $\mathcal{H}_\infty$ Controller Design

7.3.1 Augmentations of the Model with Weighting Functions

In this section, we will focus on the weighted control structure shown in Figure 7.13, where $\boldsymbol{W}_1(s)$, $\boldsymbol{W}_2(s)$, and $\boldsymbol{W}_3(s)$ are weighting functions or weighting filters. We assume that $\boldsymbol{G}(s)$, $\boldsymbol{W}_1(s)$, and $\boldsymbol{W}_3(s)\boldsymbol{G}(s)$ are all proper; i.e., they are bounded when $s \to \infty$. It can be seen that the weighting function $\boldsymbol{W}_3(s)$ is not required to be proper. By slightly rearranging the block diagram in Figure 7.13, we obtain the control structure shown in Figure 7.14, which agrees with the standard structure in Figure 7.11(a).

One may wonder why we need to use three weighting functions in Figure 7.13. First, we note that the weighting functions are, respectively, for the three signals, namely, the error, the input, and the output. In the two-port state space structure, the output vector $\boldsymbol{y}_1 = [\boldsymbol{y}_{1a}, \boldsymbol{y}_{1b}, \boldsymbol{y}_{1c}]^{\mathbf{T}}$ is not used directly to construct the control signal vector $\boldsymbol{u}_2$. We should understand that $\boldsymbol{y}_1$ is actually for the control system performance measurement. So, it is not strange to include the filtered "input signal" $\boldsymbol{u}(t)$ in the "output signal" $\boldsymbol{y}_1$ because one may need to measure the control energy to assess whether the designed controller is good or not. Clearly, Figure 7.13 represents a more general picture of optimal and robust control systems. The weighting functions can also be regarded as filters. This type of

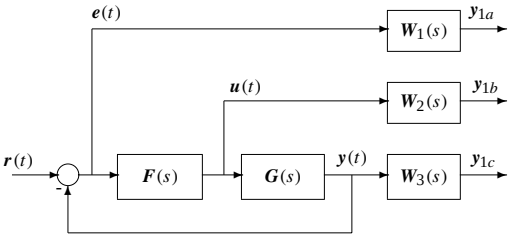


Figure 7.13. *Block diagram of general weighted sensitivity functions.*

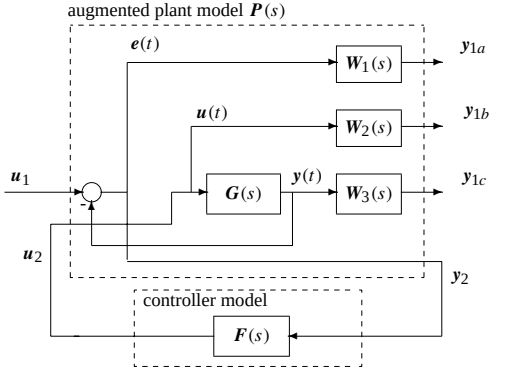


Figure 7.14. *Two-port diagram with weighting functions.*

frequency-dependent weighting is more practical. For example, if one wishes to emphasize the tracking error in the low frequency band, $\boldsymbol{W}_1(s)$ can be simply chosen as a low-pass filter. We will show next that, given the weighting transfer functions, we can design an $\mathcal{H}_\infty$ by using the idea of the augmented state space model.

Assume that the state space representation of the plant model is given by $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$. Denote by $(\boldsymbol{A}_{\boldsymbol{W}_1}, \boldsymbol{B}_{\boldsymbol{W}_1}, \boldsymbol{C}_{\boldsymbol{W}_1}, \boldsymbol{D}_{\boldsymbol{W}_1})$ the state space representation for $\boldsymbol{W}_1(s)$ and by $(\boldsymbol{A}_{\boldsymbol{W}_2}, \boldsymbol{B}_{\boldsymbol{W}_2}, \boldsymbol{C}_{\boldsymbol{W}_2}, \boldsymbol{D}_{\boldsymbol{W}_2})$ that for $\boldsymbol{W}_2(s)$. For $\boldsymbol{W}_3(s)$, which may possibly be improper, it is denoted as follows:

$$\boldsymbol{W}_3(s) = \boldsymbol{C}_{\boldsymbol{W}_3}(s\boldsymbol{I} - \boldsymbol{A}_{\boldsymbol{W}_3})^{-1}\boldsymbol{B}_{\boldsymbol{W}_3} + \boldsymbol{P}_m s^m + \cdots + \boldsymbol{P}_1 s + \boldsymbol{P}_0. \tag{7.29}$$

Under the above setup, (7.24) can be written as

$$\boldsymbol{P}(s) = \left[\begin{array}{cccc:c:c} \boldsymbol{A} & 0 & 0 & 0 & 0 & \boldsymbol{B} \\ -\boldsymbol{B}_{\boldsymbol{W}_1}\boldsymbol{C} & \boldsymbol{A}_{\boldsymbol{W}_1} & 0 & 0 & \boldsymbol{B}_{\boldsymbol{W}_1} & -\boldsymbol{B}_{\boldsymbol{W}_1}\boldsymbol{D} \\ 0 & 0 & \boldsymbol{A}_{\boldsymbol{W}_2} & 0 & 0 & \boldsymbol{B}_{\boldsymbol{W}_2} \\ \boldsymbol{B}_{\boldsymbol{W}_3}\boldsymbol{C} & 0 & 0 & \boldsymbol{A}_{\boldsymbol{W}_3} & 0 & \boldsymbol{B}_{\boldsymbol{W}_3}\boldsymbol{D} \\ -\boldsymbol{D}_{\boldsymbol{W}_1}\boldsymbol{C} & \boldsymbol{C}_{\boldsymbol{W}_1} & 0 & 0 & \boldsymbol{D}_{\boldsymbol{W}_1} & -\boldsymbol{D}_{\boldsymbol{W}_1}\boldsymbol{D} \\ \hdashline 0 & 0 & \boldsymbol{C}_{\boldsymbol{W}_2} & 0 & 0 & \boldsymbol{D}_{\boldsymbol{W}_2} \\ \widetilde{\boldsymbol{C}} + \boldsymbol{D}_{\boldsymbol{W}_3}\boldsymbol{C} & 0 & 0 & \boldsymbol{C}_{\boldsymbol{W}_3} & 0 & \widetilde{\boldsymbol{D}} + \boldsymbol{P}_0\boldsymbol{D} \\ \hdashline -\boldsymbol{C} & 0 & 0 & 0 & \boldsymbol{I} & -\boldsymbol{D} \end{array}\right], \tag{7.30}$$

where

$$\begin{aligned} \widetilde{\boldsymbol{C}} &= \boldsymbol{P}_0\boldsymbol{C} + \boldsymbol{P}_1\boldsymbol{C}\boldsymbol{A} + \cdots + \boldsymbol{P}_m\boldsymbol{C}\boldsymbol{A}^{m-1}, \\ \widetilde{\boldsymbol{D}} &= \boldsymbol{P}_0\boldsymbol{D} + \boldsymbol{P}_1\boldsymbol{C}\boldsymbol{B} + \cdots + \boldsymbol{P}_m\boldsymbol{C}\boldsymbol{A}^{m-2}\boldsymbol{B}. \end{aligned} \tag{7.31}$$

We remark that each weighting function can be assumed to be empty, i.e., in MATLAB terminology, $\boldsymbol{W}_i(s) = [\,]$. Among robust control problems, we focus on the following three cases:

1. *Sensitivity problem*: $\boldsymbol{W}_2(s)$ and $\boldsymbol{W}_3(s)$ are not specified.
2. *Mixed robust stability and performance*: $\boldsymbol{W}_3(s)$ is empty.
3. *General mixed sensitivity problem*: All three weighting functions are present. In general, the augmented plant model $\boldsymbol{P}(s)$ can be written in the following form:

$$\boldsymbol{P}(s) = \left[\begin{array}{c:c} \boldsymbol{W}_1 & -\boldsymbol{W}_1\boldsymbol{G} \\ \boldsymbol{0} & \boldsymbol{W}_2 \\ \boldsymbol{0} & \boldsymbol{W}_3\boldsymbol{G} \\ \hdashline \boldsymbol{I} & -\boldsymbol{G} \end{array}\right], \tag{7.32}$$

which in $\mathcal{H}_\infty$ design is called the general mixed sensitivity problem. The linear fractional transformation representation of such a problem can be written as

$$\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1} = \left[\begin{array}{c} \boldsymbol{W}_1\boldsymbol{S} \\ \boldsymbol{W}_2\boldsymbol{F}\boldsymbol{S} \\ \boldsymbol{W}_3\boldsymbol{T} \end{array}\right], \tag{7.33}$$

where $\boldsymbol{F}(s)$ is the controller to be designed, $\boldsymbol{S}(s)$ is the sensitivity transfer function defined as

$$\boldsymbol{S}(s) = \boldsymbol{E}(s)\boldsymbol{R}^{-1}(s) = [\boldsymbol{I} + \boldsymbol{F}(s)\boldsymbol{G}(s)]^{-1}, \tag{7.34}$$

and $\boldsymbol{T}(s)$ is the complementary sensitivity transfer function defined as

$$\boldsymbol{T}(s) = \boldsymbol{I} - \boldsymbol{S}(s) = \boldsymbol{F}(s)\boldsymbol{G}(s)[\boldsymbol{I} + \boldsymbol{F}(s)\boldsymbol{G}(s)]^{-1}. \tag{7.35}$$

7.3.2 Model Augmentation with Weighting Function Under MATLAB

First, the tree structures of the weighting functions should be created using the `mksys()` as follows:

$S_{\boldsymbol{W}_1}$=mksys($\boldsymbol{A}_{\boldsymbol{W}_1}$,$\boldsymbol{B}_{\boldsymbol{W}_1}$,$\boldsymbol{C}_{\boldsymbol{W}_1}$,$\boldsymbol{D}_{\boldsymbol{W}_1}$)
$S_{\boldsymbol{W}_2}$=mksys($\boldsymbol{A}_{\boldsymbol{W}_2}$,$\boldsymbol{B}_{\boldsymbol{W}_2}$,$\boldsymbol{C}_{\boldsymbol{W}_2}$,$\boldsymbol{D}_{\boldsymbol{W}_2}$)
$S_{\boldsymbol{W}_3}$=mksys($\boldsymbol{A}_{\boldsymbol{W}_3}$,$\boldsymbol{B}_{\boldsymbol{W}_3}$,$\boldsymbol{C}_{\boldsymbol{W}_3}$,$\boldsymbol{D}_{\boldsymbol{W}_3}$)

where the state space tree variables for the weighting functions $\boldsymbol{W}_1(s)$, $\boldsymbol{W}_2(s)$ and the proper part of $\boldsymbol{W}_3(s)$ can be established, respectively. The final plant augmentation is usually in `'tss'` (i.e., two-port state space model) format. This can be obtained using the `augss()` function provided in the Robust Control Toolbox with syntax

S_{Tss}=augss(S,$S_{\boldsymbol{W}_1}$,$S_{\boldsymbol{W}_2}$,$S_{\boldsymbol{W}_3}$,w3poly)

where S is the plant tree variable, and `w3poly` is the polynomial of $\boldsymbol{W}_3(s)$ if it is not proper. Of course, different ways of calling `augss()` are allowed. If any of the weighting functions is not provided, the variable in the function call can be replaced by an empty system. In recent versions of MATLAB, the models S and others can also be provided in state space models.

Example 7.9. Consider the plant model in Example 7.2. The weighting functions are selected as $W_1(s) = 100/(s+1)$ and $W_3(s) = s/1000$. Then, the augmented plant model can be constructed by the following MATLAB scripts:

```
>> A=[0,1,0,0; -5000,-100/3,500,100/3; 0,-1,0,1; 0,100/3,-4,-60];
   B=[0; 25/3; 0; -1];  C=[0,0,1,0]; D=0; S=mksys(A,B,C,D,'ss');
   W1num=[100]; W1den=[1,1]; [a,b,c,d]=tf2ss(W1num,W1den);
   S1=mksys(a,b,c,d); S2=mksys([],[],[],[]); S3=mksys([],[],[],[]);
   W3poly=[1/1000]; SysTss=augss(S,S1,S2,S3,W3poly);
```

If the weighting functions are given in transfer function format, the MATLAB function `augtf()` provided in the Robust Control Toolbox can also be used. The syntax is S_T=augtf(S,$\boldsymbol{W}_1$,$\boldsymbol{W}_2$,$\boldsymbol{W}_3$), where S_T is the tree variable of the plant model, and $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, $\boldsymbol{W}_3$ are the transfer function representations of the weighting functions which can be represented by two-row matrices, whose first row is the numerator and second row the denominator. The transfer function augmentation accepts the improper transfer functions. The returned variable S_T is the two-port state space representation of the augmented system. In new versions of the Robust Control Toolbox, the plant and weighting functions can also be described by linear time-invariant (LTI) objects, thus the augmentation is even simpler. In the following linear time-invariant examples, such augmentations are demonstrated.

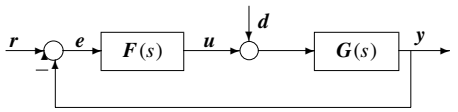


Figure 7.15. *Typical feedback control structure.*

Example 7.10. Consider again the problem in Example 7.9. The augmented system can be obtained with the following MATLAB statements:

```
>> A=[0,1,0,0; -5000,-100/3,500,100/3; 0,-1,0,1; 0,100/3,-4,-60];
   B=[0; 25/3; 0; -1];  C=[0,0,1,0]; D=0; G=ss(A,B,C,D);
   W1=tf(100,[1,1]); W2=[]; W3=tf([1 0],1000);
   SysTss1=augtf(G,W1,W2,W3);
```

It can be seen that the statements are quite straightforward.

The two-port format of the augmented system can be retrieved by the direct use of the `branch()` function as demonstrated in the following:

```
>> [a,b1,b2,c1,c2,d11,d12,d21,d22]=branch(SysTss1);
```

and the results can be represented mathematically as

$$P(s) = \left[\begin{array}{ccccc:c:c} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -5000 & -100/3 & 500 & 100/3 & 0 & 0 & 25/3 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 100/3 & -4 & -60 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & -1 & 1 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 100 & 0 & 0 \\ 0 & -1/1000 & 0 & 1/1000 & 0 & 0 & 0 \\ \hdashline 0 & 0 & -1 & 0 & 0 & 1 & 0 \end{array}\right].$$

7.3.3 Weighted Sensitivity Problems: A Simple Case

The weighted sensitivity problem is a simple case of $\mathcal{H}_\infty$ controller design. In the general weighting case, $\boldsymbol{W}_2(s)$ and $\boldsymbol{W}_3(s)$ are set to `[ ]` for this weighted sensitivity problem. Denoting the sensitivity transfer function by $\boldsymbol{S}(s)$, and introducing a weighting function $\boldsymbol{W}(s)$, our design objective is to find a controller such that $\|\boldsymbol{W}(s)\boldsymbol{S}(s)\|_\infty < 1$.

A typical feedback control structure under discussion in this subsection is shown in Figure 7.15. Before we go further into controller design, we present the following theorem.

Theorem 7.2. If the plant model $\boldsymbol{G}(s)$ is a stable and proper transfer function, the set of all controllers $\boldsymbol{F}(s)$ which internally stabilize the open-loop system can be written as

$$\boldsymbol{F}(s) = \boldsymbol{Q}(s)[\boldsymbol{I} - \boldsymbol{G}(s)\boldsymbol{Q}(s)]^{-1}, \tag{7.36}$$

where $\boldsymbol{Q}(s)$ is any stable proper transfer function. Equation (7.36) is also referred to as the Youla parameterization formula.

Thus, our controller design task is to pick up, from a set of controllers defined by the above Youla parameterization, a subset of controllers ensuring that $\|\boldsymbol{W}(s)\boldsymbol{S}(s)\|_\infty < 1$. Note that, this subset consists of controllers that are all called robust controllers, as explained earlier in Sec. 7.2.1. Here, $\boldsymbol{W}(s)$ has two roles. The first role is the scaling so that $\|\boldsymbol{\Delta}\|_\infty < 1$. The second, more practical role is that $\boldsymbol{W}(s)$ can be used in the frequency domain to put a different emphasis on the shape of $\boldsymbol{S}(s)$, known as sensitivity shaping. If we can define an optimal performance index so that we can pick up a unique controller from this subset of robust controllers, this uniquely determined controller is then the robust and optimal controller.

Stable minimum phase plant model considerations

Define a kth-order transfer function $J(s) = 1/(\tau s + 1)^k$, with k a positive integer. For any given stable and strictly proper transfer function $G(s)$, it can be shown that

$$\lim_{\tau \to 0} \|G(1 - J)\|_\infty = 0. \tag{7.37}$$

From (7.36), $WS = W(1 - GQ)$. With the introduced $J(s)$, and k selected as the pole-zero excess, of the plant model, set

$$Q(s) = G^{-1}J \tag{7.38}$$

such that $Q(s)$ is a stable proper transfer function. Then, $WS = W(1 - J)$. Thus, for relatively small τ, $\|W(s)S(s)\|_\infty < 1$.

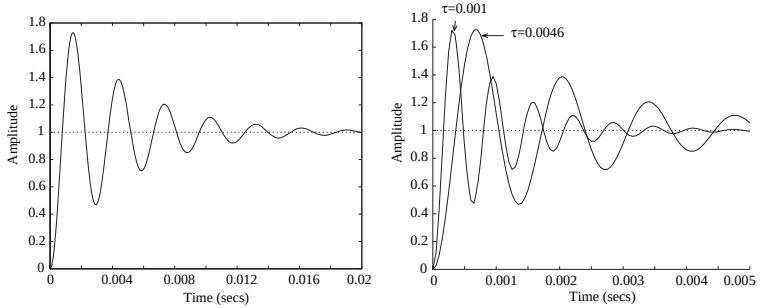
A MATLAB function `minsens()` is written which can be used to design a minimum sensitivity stabilizing controller for stable minimum phase plant model $G(s)$:

```
function [Gc,tau]=minsens(G,W,options)
t1=options(1); t2=options(2);G=tf(G); W=tf(W); num=G.num{1};
den=G.den{1}; ii=find(abs(num>eps)); num=num(ii(1):end);
k=length(den)-length(num); zr=roots(num); norms=[]; JJ=[];
tt=logspace(log10(t1),log10(t2),10);
if ~any(real(zr)>=0)
   for i=1:length(tt), Jden=1;
      for j=1:k,Jden=conv(Jden,[tt(i),1]);end
      nn=Jden-[zeros(1,k),1]; JJ=[JJ; Jden]; g1=tf(nn,Jden);
      g=ss(g1*W); norms=[norms,normhinf(g.a,g.b,g.c,g.d)];
   end
end
norms, key=input('Select a number n=> ');
tau=tt(key); Qnum=den; Qden=JJ(key,:); nn=JJ(key,:)-[zeros(1,k),1];
g1=tf(Qnum,Qden); g2=tf(JJ(key,:),nn); Gc=minreal(g1*g2);
```

The syntax of the function is `[`G_c`,`τ^*`]=minsens(`G`,`W`,Tau_range)`, where G is the transfer function object of the plant and W is the weighting function $W(s)$. `Tau_range` is used to specify the minimum and maximum values of τ. G_c returned is the stabilizing controller $F(s)$, and τ^* returns the best value of τ.

Example 7.11. Consider a stable minimum phase plant model

$$G(s) = \frac{100}{s^2 + 7s + 2}$$



(a) for $\tau = 0.0022$ (b) for different values of τ

Figure 7.16. *Step responses for different τ.*

with a weighting function of $W(s) = 100/(s+1)$. The following MATLAB statements are used to design a robust controller for the minimum sensitivity problem. The step response under such a controller is shown in Figure 7.16(a):

```
>> G=tf(100,[1,7,2]); W=tf(100,[1,1]);
   [Gc,tau]=minsens(G,W,[0.001,1]); tau, zpk(Gc)
```

As an intermediate result, a series of norm values 0.1996, 0.4294, 0.9221, 1.9722, 4.1836, 8.7499, 17.6146, 33.3642, 56.5321, 82.0190 is displayed. If 0.4294 is selected, the optimum value of τ is 0.0022, and the optimal controller obtained is

$$G_c(s) = \frac{215443.469(s+6.702)(s+0.2984)}{s(s+928.3)}.$$

The step response under the controller can be drawn with the following statements, as shown in Figure 7.16(a):

```
>> G_o=G*Gc; G_c=feedback(G_o,1); step(G_c,0:0.00005:0.02)
```

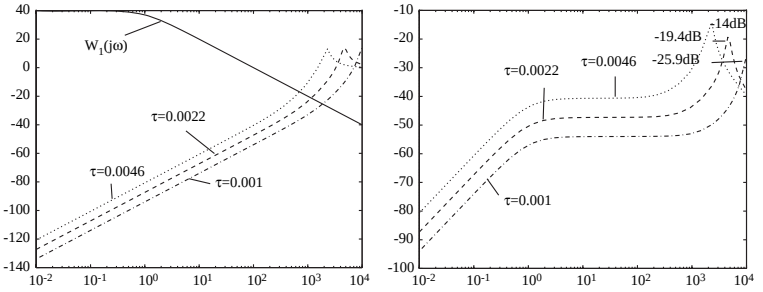
Now, let us check the other two values of τ: $\tau = 0.0046$ and $\tau = 0.001$. By the following MATLAB statements:

```
>> [Gc1,tau]=minsens(G,W,[0.001,1]);
Select a number n=> 3
>> G_c1=feedback(G*Gc1,1); [Gc2,tau]=minsens(G,W,[0.001,1]);
Select a number n=> 1
>> G_c2=feedback(G*Gc2,1); step(G_c1,G_c2,0.01);
```

the obtained step responses are compared in Figure 7.16(b). It can be seen that the smaller the τ, the quicker the response. However, too small a τ may induce the numerical roundoff problem.

The following MATLAB statements can be used to perform the frequency domain analysis:

```
>> g1=feedback(1,G*Gc); [m1,p1,w]=bode(g1);
   g2=feedback(1,G*Gc1); [m2,p2]=bode(g2,w);
```

(a) sensitivity functions (b) weighted sensitivity functions

Figure 7.17. *Bode magnitude plots for different controllers.*

```
g3=feedback(1,G*Gc2); [m3,p3]=bode(g3,w); [m,p]=bode(W,w);
m1=20*log10(m1(:)); m2=20*log10(m2(:));
m3=20*log10(m3(:)); m=20*log10(m(:));
semilogx(w,m,'-',w,m1,'--',w,m2,':',w,m3,'-.'),
figure; semilogx(w,m+m1,'--',w,m+m2,':',w,m+m3,'-.')
```

The magnitude Bode plot of the sensitivity functions, together with the weighting function $W(s)$, is shown in Figure 7.17(a). The plots for $\|W(j\omega)S(j\omega)\|_\infty$ are also obtained and shown in Figure 7.17(b). It can be seen that for all three controllers, $\|W(j\omega)S(j\omega)\|_\infty < 1$ for all the frequencies, i.e., below the 0dB line. The peak values representing the $\mathcal{H}_\infty$-norms are also labeled on the plots.

Coprime factorization of transfer functions

It is obvious that the above algorithm with $Q(s) = G^{-1}J$ is applicable only for stable minimum phase plants. If $G(s)$ is nonminimum phase, $Q(s) = G^{-1}J$ will contain unstable poles which will cause the system to be internally unstable. The design algorithm for nonminimum phase plants can be done using the coprime factorization technique.

Definition 7.1. For a given transfer function which can be written as $G(s) = N(s)/M(s)$, where $N(s)$ and $M(s)$ are stable proper transfer functions, if there exist two other stable proper transfer functions $X(s)$ and $Y(s)$ such that

$$N(s)X(s) + M(s)Y(s) = 1, \tag{7.39}$$

the transfer functions $N(s)$ and $M(s)$ are called coprime. Finding the four transfer functions (N, M, X, Y) is called the coprime factorization of $G(s)$. Equation (7.39) is also called the Bezout equation.

To understand the concept of coprimeness of two transfer functions $N(s)$ and $M(s)$, we note the following:

(1) $N(s)$ and $M(s)$ are both stable proper transfer functions.
(2) The zeros, including the infinity ones, of $N(s)$ cannot be canceled with those of $M(s)$.

If $G(s)$ is stable, there is no need to perform coprime factorization. In this case, a straightforward solution to the Bezout equation is that

$$N(s) = G(s), \quad M(s) = Y(s) = 1, \quad X(s) = 0. \tag{7.40}$$

Example 7.12. Let us illustrate the concept of coprimeness through an example. Consider an unstable transfer function $G(s) = 1/(s-1)$. If two transfer functions $N(s) = 1/(s+1)$ and $M(s) = (s-1)/(s+1)$ are chosen, it is clear that the above two conditions are satisfied. Thus $N(s)$ and $M(s)$ are coprime. Consider another two transfer functions $N_1(s) = 1/(s+1)^2$ and $M_1(s) = (s-1)/(s+1)^2$. It can be seen that a zero of $M_1(s)$ is at $s = \infty$, which is the same as one zero of $N_1(s)$. Thus, the (N_1, M_1) pair is not coprime.

It should also be noted that the pair $N(s) = 1/(s+2)$ and $M(s) = (s-1)/(s+2)$ is also a coprime representation of $G(s)$. So, the coprime factorization of a given transfer function is not unique.

From the above example, it is readily seen that obtaining a coprime representation $N(s)/M(s)$ is very simple. However, one may have difficulty finding the transfer functions $X(s)$ and $Y(s)$ satisfying the Bezout equation. From among different methods for coprime factorization, we shall only present the state space approach.

Theorem 7.3. Let $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, D)$ be the state space representation of $G(s)$. Pick a matrix $\boldsymbol{F}$ stabilizing $\boldsymbol{A} + \boldsymbol{B}\boldsymbol{F}$. Then, the state space representations of $M(s)$ and $N(s)$ are given as

$$M(s) = \left[\begin{array}{c|c} \boldsymbol{A}+\boldsymbol{B}\boldsymbol{F} & \boldsymbol{B} \\ \hline \boldsymbol{F} & 1 \end{array}\right], \quad N(s) = \left[\begin{array}{c|c} \boldsymbol{A}+\boldsymbol{B}\boldsymbol{F} & \boldsymbol{B} \\ \hline \boldsymbol{C}+D\boldsymbol{F} & D \end{array}\right]. \tag{7.41}$$

Choose a matrix $\boldsymbol{H}$ to ensure that $\boldsymbol{A} + \boldsymbol{H}\boldsymbol{C}$ is stable. Then, the state space representations of $X(s)$ and $Y(s)$ are as follows:

$$X(s) = \left[\begin{array}{c|c} \boldsymbol{A}+\boldsymbol{H}\boldsymbol{C} & \boldsymbol{H} \\ \hline \boldsymbol{F} & 0 \end{array}\right], \quad Y(s) = \left[\begin{array}{c|c} \boldsymbol{A}+\boldsymbol{H}\boldsymbol{C} & -\boldsymbol{B}-\boldsymbol{H}D \\ \hline \boldsymbol{F} & 1 \end{array}\right]. \tag{7.42}$$

A MATLAB function `coprime()` is written for coprime factorization given a transfer function $G(s)$, where the pole placement algorithm is used to determine F and H:

```
function [N,M,X,Y]=coprime(G,K1,K2)
G=ss(G); a=G.a; b=G.b; c=G.c; d=G.d;
if length(K1)==1, K1=K1*ones(size(c)); end
if length(K2)==1, K2=K2*ones(size(c)); end
F=-acker(a,b,K1); H=-acker(a',c',K2)'; M=ss(a+b*F,b,F,1);
N=ss(a+b*F,b,c+d*F,d); X=ss(a+H*c,H,F,0);Y=ss(a+H*c,-b-H*d,F,1);
```

The syntax of the function is `[N,M,X,Y]=coprime(G,K1,K2)`, where G is the transfer function of $G(s)$, and the returned variables are the transfer function objects for N, M, X, Y, respectively. For SISO systems, $\boldsymbol{K}_1$, $\boldsymbol{K}_2$ are defined as follows:

(1) If $\boldsymbol{K}_1$ or $\boldsymbol{K}_2$ is a column vector, it contains the desired pole positions:

(2) If $\boldsymbol{K}_1$ or $\boldsymbol{K}_2$ is a scalar, it will contain the expected repeated pole position of the relevant stabilization problems.

Example 7.13. Perform a coprime factorization for the unstable nonminimum phase system

$$G(s) = \frac{(s-1)^2(s^2-s+1)}{(s-2)^2(s+1)^3}.$$

To perform a coprime factorization of $G(s)$, let us assume that the desired poles are all located at $s = -2$ for $\boldsymbol{F}$ and at $s = -0.1$ for $\boldsymbol{H}$. Then, the following MATLAB statements can be used:

```
>> s=tf('s'); G=(s-1)^2*(s^2-s+1)/((s-2)^2*(s+1)^3);
   [N,M,X,Y]=coprime(G,-2,-0.1); zpk(N),zpk(M),zpk(X),zpk(Y)
```

It can then be found that

$$M(s) = \frac{(s-1)^2(s^2-s+1)}{(s+2)^5}, \quad N(s) = \frac{(s+1)^3(s-2)^2}{(s+2)^5},$$

$$X(s) = \frac{-190.5279(s-4.71)(s+0.9637)(s^2+2.036s+1.038)}{(s+0.1)^5},$$

$$Y(s) = \frac{(s-0.823)(s^2-1.036s+0.9908)(s^2+13.36s+275.1)}{(s+0.1)^5}.$$

To design a robust controller for an unstable minimum phase plant $G(s)$, the following steps can be used to ensure that $\|W(s)S(s)\|_\infty < 1$:

1. Perform a coprime factorization to get (N, M, X, Y);
2. select a τ which is small enough to make $\|WMY(1-J)\|_\infty < 1$, with k in $J(s)$ equal to the pole-zero excess of $G(s)$;
3. set $Q = YN^{-1}J$;
4. get a robust controller from $F(s) = (X + MQ)/(Y - NQ)$.

Nonminimum phase plant model consideration

For a nonminimum phase system model, the following alternative design procedure [82], which is slightly different from the design steps presented in the above, was proposed to design a robust controller to achieve $\|W(s)S(s)\|_\infty < 1$:

1. Perform coprime factorization (N, M, X, Y);
2. find a stable Q_0 such that $\|WM(Y - NQ_0)\|_\infty < 1$;
3. find a relatively small τ such that $\|WM(Y - NQ_0J)\|_\infty < 1$;
4. select a Q such that $Q = Q_0J$;
5. obtain a robust controller from $F(s) = (X + MQ)/(Y - NQ)$.

7.3.4 $\mathcal{H}_\infty$ Controller Design: The General Case

To consider the general case, let us focus on the two-port system structure for $\mathcal{H}_\infty$ control described in Figure 7.11(a). The design objective is to find a robust controller $\boldsymbol{F}_c(s)$ guaranteeing the closed-loop system with an $\mathcal{H}_\infty$-norm bounded by a given positive number γ,

i.e., $\|\boldsymbol{T}_{y_1u_1}(s)\|_\infty < \gamma$. The controller can be represented by

$$\boldsymbol{F}_c(s) = \left[\begin{array}{c|c} \boldsymbol{A}_f & -\boldsymbol{Z}\boldsymbol{L} \\ \hline \boldsymbol{F} & 0 \end{array}\right], \tag{7.43}$$

where

$$\begin{aligned} \boldsymbol{A}_f &= \boldsymbol{A} + \gamma^{-2}\boldsymbol{B}_1\boldsymbol{B}_1^{\mathrm{T}}\boldsymbol{X} + \boldsymbol{B}_2\boldsymbol{F} + \boldsymbol{Z}\boldsymbol{L}\boldsymbol{C}_2, \\ \boldsymbol{F} &= -\boldsymbol{B}_2^{\mathrm{T}}\boldsymbol{X}, \quad \boldsymbol{L} = -\boldsymbol{Y}\boldsymbol{C}_2^{\mathrm{T}}, \quad \boldsymbol{Z} = (\boldsymbol{I} - \gamma^{-2}\boldsymbol{Y}\boldsymbol{X})^{-1}, \end{aligned} \tag{7.44}$$

and $\boldsymbol{X}$ and $\boldsymbol{Y}$ are, respectively, the solutions of the following two AREs:

$$\begin{aligned} &\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X} + \boldsymbol{X}\boldsymbol{A} + \boldsymbol{X}(\gamma^{-2}\boldsymbol{B}_1\boldsymbol{B}_1^{\mathrm{T}} - \boldsymbol{B}_2\boldsymbol{B}_2^{\mathrm{T}})\boldsymbol{X} + \boldsymbol{C}_1\boldsymbol{C}_1^{\mathrm{T}} = \boldsymbol{0}, \\ &\boldsymbol{A}\boldsymbol{Y} + \boldsymbol{Y}\boldsymbol{A}^{\mathrm{T}} + \boldsymbol{Y}(\gamma^{-2}\boldsymbol{C}_1^{\mathrm{T}}\boldsymbol{C}_1 - \boldsymbol{C}_2^{\mathrm{T}}\boldsymbol{C}_2)\boldsymbol{Y} + \boldsymbol{B}_1^{\mathrm{T}}\boldsymbol{B}_1 = \boldsymbol{0}. \end{aligned} \tag{7.45}$$

The conditions for the existence of an $\mathcal{H}_\infty$ controller are as follows:

- $\boldsymbol{D}_{11}$ is small enough such that $\boldsymbol{D}_{11} < \gamma$;
- the solution $\boldsymbol{X}$ of the controller ARE is positive-definite;
- the solution $\boldsymbol{Y}$ of the observer ARE is positive-definite;
- $\lambda_{\max}(\boldsymbol{X}\boldsymbol{Y}) < \gamma^2$, which indicates that the eigenvalues of the product of the two Riccati equation solution matrices are all less than γ^2.

A MATLAB function `hinf()` is provided in the Robust Control Toolbox for $\mathcal{H}_\infty$ controller design for the general mixed stability and performance problem. The syntax is `[`$\boldsymbol{F}_c$`,`$\boldsymbol{G}_{cl}$`]=hinf(`$\boldsymbol{G}$`)`, where $\boldsymbol{G}$ is the two-port state space description of the plant model, including the specifications of the weighting functions. The returned tree variable $\boldsymbol{F}_c$ is the designed $\mathcal{H}_\infty$ controller in state space form and $\boldsymbol{G}_{cl}$ is the state space object of the closed-loop system.

The existence of the $\mathcal{H}_\infty$ controller is checked first. Then, an $\mathcal{H}_\infty$ controller is designed if all the conditions are satisfied. Otherwise, the error messages will be given to report the failure of the function call.

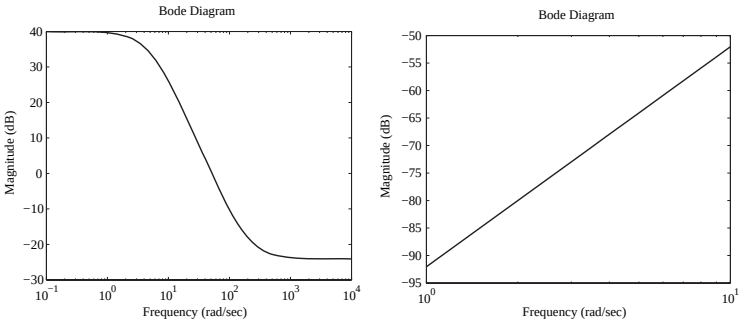
Example 7.14. Consider a plant model given by

$$G(s) = \frac{400}{s^2 + \delta s + 400},$$

where the uncertain parameter δ varies within a certain range. The nominal value of δ is 2. Define the weighting functions as

$$W_1(s) = \frac{100(0.005s+1)^2}{\rho(0.2s+1)^2}, \quad W_3(s) = \frac{s^2}{40000}.$$

Let us first consider $\rho = 1$. The magnitude Bode diagrams of the weighting functions $W_1(s)$ and $W_3(s)$ are shown in Figures 7.18(a) and (b), respectively, using the following



(a) $W_1(\mathrm{j}\omega)$ (b) $W_3(\mathrm{j}\omega)$

Figure 7.18. *Bode magnitude plots for weighting functions.*

MATLAB statements:

```
>> num=400; den=[1,2,400]; G=tf(num,den); s=tf('s');
   W1=100*(0.005*s+1)^2/(0.2*s+1)^2; W3=s^2/40000;
   bodemag(W1); figure, bodemag(W3)
```

The two-port augmented system can be established, and the $\mathcal{H}_\infty$ controller can then be designed by the direct use of the `hinf()` function,

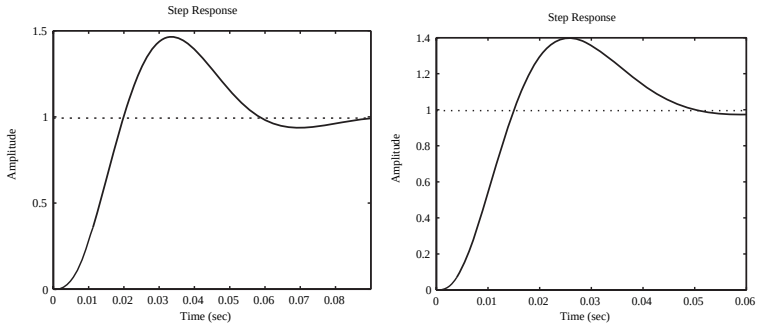
```
>> S_T=augtf(G,W1,[],W3); Gc1=hinf(S_T); zpk(Gc1)
```

Then the following messages are displayed:

```
         << H-inf Optimal Control Synthesis >>
     Computing the 4-block H-inf optimal controller
    using the S-L-C loop-shifting/descriptor formulae
Solving for the H-inf controller F(s) using U(s) = 0 (default)
Solving Riccati equations and performing H-infinity existence tests:
       1.  Is D11 small enough?                          OK
       2.  Solving state-feedback (P) Riccati ...
           a.  No Hamiltonian jw-axis roots?          OK
           b.  A-B2*F stable (P >= 0)?                OK
       3.  Solving output-injection (S) Riccati ...
           a.  No Hamiltonian jw-axis roots?          OK
           b.  A-G*C2 stable (S >= 0)?                OK
       4.  max eig(P*S) < 1 ?                         OK
     ---------------------------------------------------------
       all tests passed -- computing H-inf controller ...
                          DONE!!!
     ---------------------------------------------------------
```

We can see that all the existence conditions are satisfied and the design process of the $\mathcal{H}_\infty$ controller is successful. The controller $G_{c1}(s)$ is obtained as in the following format:

$$G_{c1}(s) = \frac{8597.8554(s+42.84)(s^2+2s+400)}{(s+5)^2(s^2+308.5s+4.388\times10^4)}.$$



(a) with $\rho = 1$ (b) with $\rho = 0.5$

Figure 7.19. *Closed-loop step responses for different ρ.*

The closed-loop step response of the system obtained with the MATLAB statements

```
>> G_o=G*Gc; G_c=feedback(G_o,1); step(G_c)
```

is shown in Figure 7.19(a). Let us see what will happen if ρ is reduced to 0.5. Using the following MATLAB statements:

```
>> W1=2*W1; S_T1=augtf(G,W1,[],W3); F=hinf(S_T1);
   Gc2=zpk(F), G_o=G*Gc2; G_c=feedback(G_o,1); step(G_c);
```

we find that and the controller is

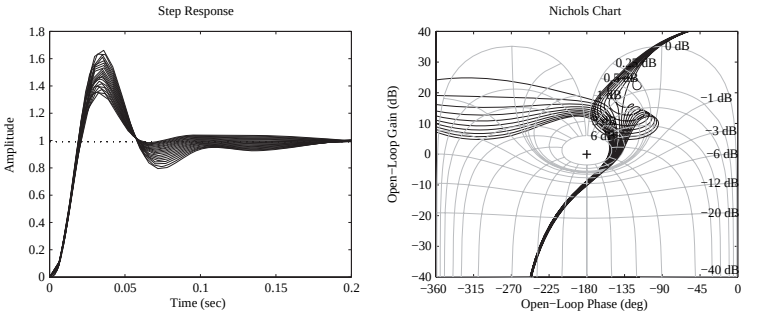
$$G_{c2}(s) = \frac{27751.2951(s+51.54)(s^2+2s+400)}{(s+5)^2(s^2+542s+1.078\times 10^5)}.$$

The closed-loop step response under the new ρ is shown in Figure 7.19(b). Clearly, by decreasing ρ, the step response of the system will be faster with the overshoot reduced while keeping approximately the same shape in the dynamic response curves.

Now let us perturb the parameter $\delta \in (-10, 10)$ in the plant model. Note that when $\delta < 0$, the open-loop plant model is unstable. The step response under the same $\mathcal{H}_\infty$ controller $F_1(s)$ can be obtained using the following MATLAB statements:

```
>> f1=figure; f2=figure;
   for delta=-10:1:10
      den(2)=delta; G=tf(num,den); Go=G*Gc1; Gc=feedback(Go,1);
      figure(f1);step(Gc),hold on;figure(f2); nichols(Go);hold on
   end
   figure(f1); xlim([0,0.2]); figure(f2),axis([-360,0,-40,40]),grid
```

The step responses of the controlled system with different values of δ are compared in Figure 7.20(a). It can be observed that although the plant model is greatly perturbed, e.g., from unstable to stable and with large change in pole positions, the step responses are rather close. The controlled system is indeed robust.



(a) step responses (b) Nichols charts

Figure 7.20. *System responses for different δ.*

To further appreciate of the robustness, see the Nichols charts for different values of δ in Figure 7.20(b). In the display window, the Nichols charts for $\delta \le 0$ have been split into two segments. Obviously, despite the great variation of δ, the Nichols charts are all kept away from the 6dB constant M contour, which ensures the satisfactory dynamic behavior of the closed-loop system.

Example 7.15. Consider the sensitivity problem for the nonminimum phase plant model given by $G(s) = (s-1)/(s+1)^2$, with

$$W(s) = 0.62\frac{s^2+1.2s+1}{(s+0.001)(s+1.2)(0.001s+1)}$$

as the weighting function. Using the following MATLAB statements:

```
>> num=[1,-1]; den=[1,2,1]; G=tf(num,den); s=tf('s');
   GW=0.62*(s^2+1.2*s+1)/(s+0.001)/(s+1.2)/(0.001*s+1);
   S_T=augtf(G,GW,[],[]); F1=hinf(S_T);
```

it can be seen that an error message is given complaining that the matrix $\boldsymbol{D}_{12}$ in (7.30) is not full ranked. To meet the full rank requirement, it can be seen that at least one of the following three conditions should be satisfied:

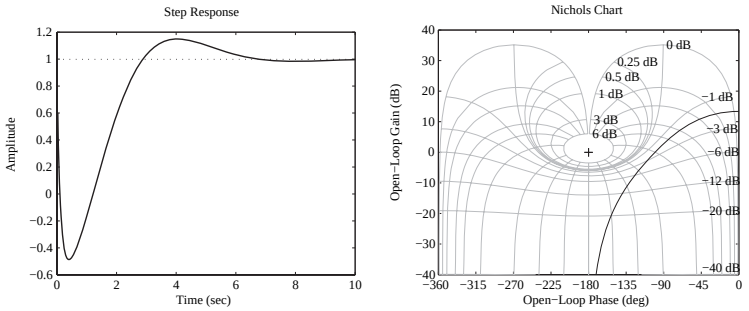
$$-\boldsymbol{D}\boldsymbol{D}_{W_1}\boldsymbol{D} \neq 0, \quad -\boldsymbol{D}_{W_2} \neq 0, \quad \tilde{\boldsymbol{D}} + \boldsymbol{D}_{W_3}\boldsymbol{D} \neq \boldsymbol{0}. \tag{7.46}$$

As a remedy, let us choose the weighting function $W_2(s)$ as a small constant value ϵ, for instance, $\epsilon = 10^{-5}$. In this way, the problem can be solved as shown below:

```
>> S_T1=augtf(G,GW,1e-5,[]); F1=hinf(S_T1); zpk(F1)
```

Therefore, an $\mathcal{H}_\infty$ controller can now be designed:

$$F_1(s) = \frac{51328980.8773(s-5.848)(s+1.778)(s+1)^2}{(s+1.014\times 10^4)(s+1000)(s+84.42)(s+1.2)(s+0.001)}.$$



(a) step responses (b) Nichols charts

Figure 7.21. *System responses with an $\mathcal{H}_\infty$ controller.*

The closed-loop step response and the open-loop Nichols chart of the system under controller `F1` are shown, respectively, in Figures 7.21(a) and (b) by using the following MATLAB statements:

```
>> G_o=G*F1; G_c=feedback(G_o,1); step(G_c,10),
   figure, nichols(G_o), grid; axis([-360,0,-40,40])
```

It can be seen that the dynamic behavior of the controlled system is satisfactory with a small undershoot.

Example 7.16. Consider a double integrator plant model $G(s) = 1/s^2$. Let the weighting functions be $W_1(s) = 0.5(s+4)/(s+1)$ and $W_3(s) = s^2/100$. An $\mathcal{H}_\infty$ controller can be designed using the following MATLAB statements:

```
>> G=tf(1,[1 0 0]); W1=tf([0.5,2],[1,1]); W3=tf([1 0 0],100);
   S_T=augtf(G,W1,[],W3); C=zpk(hinf(S_T)), step(feedback(G*C,1))
```

and the controller designed is

$$C(s) = \frac{185.132(s + 2.831\times10^{-8})(s - 2.831\times10^{-8})}{(s+1)(s^2 + 14.67s + 105.3)}.$$

The step response of the closed-loop system is shown in Figure 7.22. Since the controller just cancels the poles of the plant at $s = 0$ directly, this is usually not suggested as a good controller design. It can also be seen that the steady-state value of the closed-loop system will never reach 1.

Since there exist poles at $s = 0$, one may shift the poles away from the imaginary axis and design an $\mathcal{H}_\infty$ controller for the shifted plant. Then, shift the designed $\mathcal{H}_\infty$ controller back. A common shift algorithm uses a special bilinear transformation $z = (s + p_1)/(s/p_2 + 1)$, with $p_2 = \infty$ and $p_1 < 0$. Then, the plant model will be shifted from $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ to $(\boldsymbol{A} - p_1\boldsymbol{I}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$. After the controller design, use the inverse bilinear transformation to convert $(\boldsymbol{A}_F, \boldsymbol{B}_F, \boldsymbol{C}_F, \boldsymbol{D}_F)$ back into $(\boldsymbol{A}_F + p_1\boldsymbol{I}, \boldsymbol{B}_F, \boldsymbol{C}_F, \boldsymbol{D}_F)$.

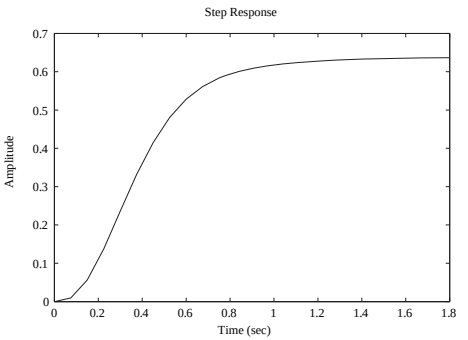


Figure 7.22. *System responses with an $\mathcal{H}_\infty$ controller.*

Assume $p_1 = -0.1$. The $\mathcal{H}_\infty$ controller can be designed using the following MATLAB statements:

```
>> G1=ss(G); [a,b,c,d]=ssdata(G1); p1=-0.1; a1=a-p1*eye(size(a));
   S_shift=ss(a1,b,c,d); W3=tf([1 0 0],100);
   TSS_shift=augtf(S_shift,W1,[],W3); Gc=hinf(TSS_shift);
   af=Gc.a+p1*eye(size(Gc.a)); Gc=zpk(ss(af,Gc.b,Gc.c,Gc.d))
```

The controller is then designed as

$$G_c(s) = \frac{242.6903(s + 0.426)(s + 0.1241)}{(s + 1.1)(s^2 + 15.76s + 118.7)}.$$

To compare the step responses and Nichols charts, use the following MATLAB statements:

```
>> G_o=G*Gc; G_c=feedback(G_o,1); step(G_c,t),
   figure, nichols(G_o), axis([-360,0,-40,40]), grid
```

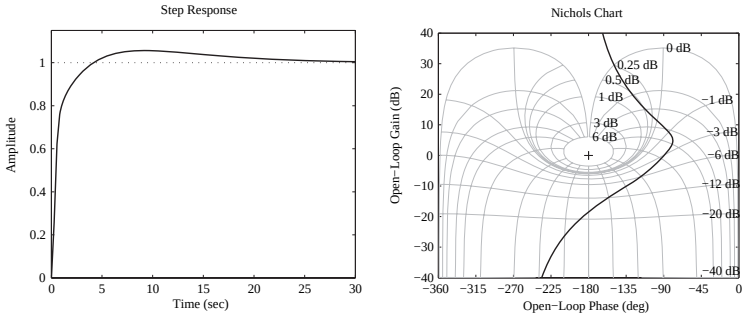
The results are shown in Figures 7.23(a) and (b), respectively. It can be seen that the speed of the responses, for this example, is rather slow.

7.3.5 Optimal $\mathcal{H}_\infty$ Controller Design

Consider the case in Example 7.14. When ρ in the weighting function decreases, the system response increases. This leaves us to question how small the value of ρ can be to ensure that one gets the best response.

In optimal $\mathcal{H}_\infty$ controller design, the optimal criterion is defined as

$$\max_\gamma \|\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}\| < \frac{1}{\gamma}, \quad \text{and in general,} \quad \max_\gamma \begin{bmatrix} \boldsymbol{W}_1\boldsymbol{S} \\ \boldsymbol{W}_2\boldsymbol{F}\boldsymbol{S} \\ \boldsymbol{W}_3\boldsymbol{T} \end{bmatrix} \le \frac{1}{\gamma}. \tag{7.47}$$



(a) step responses (b) Nichols charts

Figure 7.23. *System responses when the poles are shifted.*

Even more generally, all three terms in the matrix on the left-hand side of (7.47) can be individually weighted by γ. An iteration method, known as the γ-iteration method, can be used in finding the optimal γ.

A MATLAB function `hinfopt()`, provided in the Robust Control Toolbox, can be used to perform the optimal $\mathcal{H}_\infty$ controller design. A bisectional algorithm is used in the iteration process. The syntax of the function is `[`γ^*`,`G_f`,`G_{cl}`]=hinfopt(`G`)`, where G is again the augmented two-port state space system, which can be either the tree structure or an LTI model. The variable γ^* is the optimal value for γ.

Example 7.17. Consider again the problem in Example 7.14. The following MATLAB statements can be used to design an optimal $\mathcal{H}_\infty$ controller:

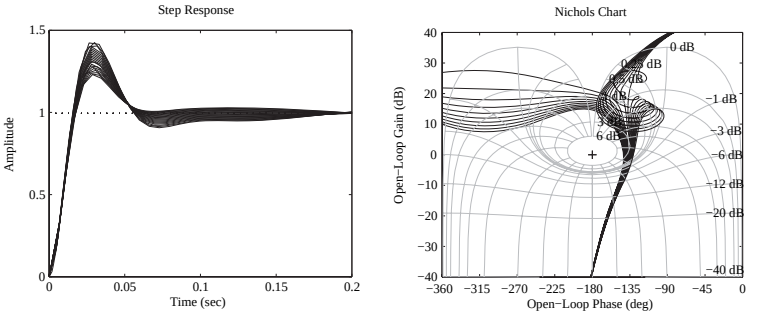
```
>> G=tf(400,[1,2,400]); s=tf('s');
   W1=100*(0.005*s+1)^2/(0.2*s+1)^2; W3=s^2/40000;
   G_T=augtf(G,W1,[],W3); [g,F,ccL]=hinfopt(G_T); Gc=zpk(F)
```

The intermediate iteration results are

```
                << H-Infinity Optimal Control Synthesis >>
 No   Gamma    D11<=1  P-Exist P>=0  S-Exist  S>=0  lam(PS)<1 C.L.
-------------------------------------------------------------------
 1 1.0000e+000   OK      OK     OK     OK      OK       OK    STAB
 2 2.0000e+000   OK      OK    FAIL    OK      OK       OK    UNST
 3 1.5000e+000   OK      OK     OK     OK      OK       OK    STAB
 4 1.7500e+000   OK      OK     OK     OK      OK       OK    STAB
 5 1.8750e+000   OK      OK     OK     OK      OK       OK    STAB
 6 1.9375e+000   OK      OK    FAIL    OK      OK       OK    UNST
 7 1.9062e+000   OK      OK    FAIL    OK      OK       OK    UNST
 8 1.8906e+000   OK      OK     OK     OK      OK       OK    STAB
Iteration no. 8 is your best answer under the tolerance:   0.0100.
```

and it is found that $\gamma^* = 1.8906$, and the controller can be written as

$$G_c(s) = \frac{505249.1691(s+48.42)(s^2+2s+400)}{(s+9617)(s+216.4)(s+5)^2}.$$



(a) step responses (b) Nichols charts

Figure 7.24. *System responses for different δ.*

Now, let us investigate the effect of uncertain parameter δ. With different values of δ, by using the following MATLAB statements:

```
>> f1=figure; f2=figure;
   for delta=-10:1:10
      G.den{1}(2)=delta; G_o=G*Gc; G_c=feedback(G_o,1);
      figure(f1); step(G_c), hold on;
      figure(f2), nichols(G_o), hold on
   end
   figure(f2), axis([-360,0,-40,40]), grid
```

the closed-loop step responses and the open-loop Nichols charts can be obtained as compared in Figures 7.24(a), and (b), respectively. It can be seen that the dynamic responses of the system are improved compared to those shown in (7.20), where only a robust $\mathcal{H}_\infty$ controller is employed.

Example 7.18. Assume that a multivariable system is given by

$$\boldsymbol{G}(s) = \begin{bmatrix} \dfrac{0.806s+0.264}{s^2+1.15s+0.202} & \dfrac{-15s-1.42}{s^3+12.8s^2+13.6s+2.36} \\ \dfrac{1.95s^2+2.12s+0.49}{s^3+9.15s^2+9.39s+1.62} & \dfrac{7.15s^2+25.8s+9.35}{s^4+20.8s^3+116.4s^2+111.6s+18.8} \end{bmatrix}.$$

The model can be entered easily into the MATLAB environment. Now consider the mixed sensitivity problem, with the weighting functions chosen as

$$\boldsymbol{W}_1(s) = \begin{bmatrix} \dfrac{100}{s+0.5} & 0 \\ 0 & \dfrac{100}{s+1} \end{bmatrix}, \quad \boldsymbol{W}_3(s) = \begin{bmatrix} \dfrac{s}{1000} & 0 \\ 0 & \dfrac{s}{200} \end{bmatrix}. \tag{7.48}$$

Let $\boldsymbol{W}_2(s)$ be an empty matrix. In fact, to avoid the singularity problem, one may still assume that $\boldsymbol{W}_2(s) = \text{diag}([10^{-5}, 10^{-5}])$. Thus, the following statements can be used to

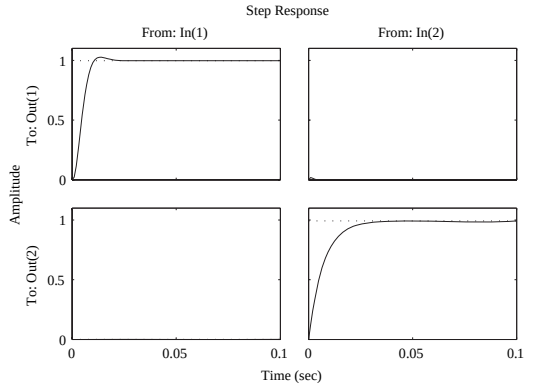


Figure 7.25. *Step response under the optimal $\mathcal{H}_\infty$ controller.*

augment the system to a two-port system. The optimal $\mathcal{H}_\infty$ can then be designed. The step response of the multivariable can be obtained as shown in Figure 7.25.

```
>> g11=tf([0.806 0.264],[1 1.15 0.202]); s=tf('s');
   g12=tf([-15 -1.42],[1 12.8 13.6 2.36]);
   g21=tf([1.95 2.12 0.49],[1 9.15 9.39 1.62]);
   g22=tf([7.15 25.8 9.35],[1 20.8 116.4 111.6 18.8]);
   G=[g11, g12; g21, g22]; W1=[100/(s+0.5),0; 0,100/(s+1)];
   W2=[tf(1e-5),0; 0,tf(1e-5)]; W3=[s/1000,0; 0,s/200];
   Tss=augtf(G,W1,W2,W3); [g,Gc]=hinfopt(Tss); zpk(Gc(1,2))
   step(feedback(G*Gc,eye(2)),0.1)
```

It can be seen that the control results are satisfactory. The decoupling problem of multivariable control was solved successfully and the performance of the responses are well acceptable. Unfortunately, the orders of the designed controller are extremely high. For instance, the off-diagonal term $g_{12}(s)$ in the controller is a 14^{th}-order model given by

$$g_{12}(s) = \frac{\begin{array}{c}1522968.8928(s+1222)(s+763)(s+11.54)(s+8.096)\\(s+8.002)(s+0.9354)(s+0.9336)(s+0.9306)\\(s+0.5)(s+0.2175)(s+0.2164)(s+0.2147)(s+0.09511)\end{array}}{\begin{array}{c}(s+2.132\times10^4)(s+1677)(s+657.8)(s+11.55)(s+8.1)\\(s+1.052)(s+1)(s+0.9331)(s+0.9218)(s+0.5)\\(s+0.3369)(s+0.2467)(s+0.2263)(s+0.2167)\end{array}}.$$

It is also obvious from the result that the response of the $y_{22}(t)$ signal is relatively slow, compared with that of output $y_{11}(t)$. Weighting must be added to $w_{1,22}(s)$ in $\boldsymbol{W}_1(s)$. For instance, let $w_{1,22}(s) = 1000/(s+1)$. A new optimal $\mathcal{H}_\infty$ controller can be designed, and the improved control result can be obtained as shown in Figure 7.26.

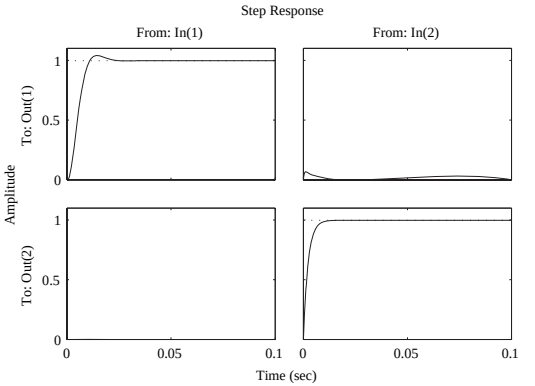


Figure 7.26. *New result with modified $\boldsymbol{W}_1(s)$ function.*

```
>> W1(2,2)=1000/(s+1); Tss=augtf(G,W1,W2,W3);
   [g,Gc1]=hinfopt(Tss); step(feedback(G*Gc1,eye(2)),0.1);
```

Since the order of the controller is extremely high, which is not easily implemented in real control applications, model reduction techniques should be used. The model reduction technique in Chapter 3 cannot be used for each individual item of the multivariable controller only. Instead, a closed-loop controller reduction technique should be used [83, 84].

7.4 Optimal $\mathcal{H}_2$ Controller Design

The $\mathcal{H}_2$ optimal control problem is to find a stabilizing controller $\boldsymbol{F}_c(s)$ for the augmented system model $\boldsymbol{P}(s)$ given by

$$\boldsymbol{P}(s) = \left[\begin{array}{c|cc} \boldsymbol{A} & \boldsymbol{B}_1 & \boldsymbol{B}_2 \\ \hline \boldsymbol{C}_1 & \boldsymbol{D}_{11} & \boldsymbol{D}_{12} \\ \boldsymbol{C}_2 & \boldsymbol{D}_{21} & \boldsymbol{D}_{22} \end{array}\right] \tag{7.49}$$

such that the $\mathcal{H}_2$-norm of the linear fractional transformation $\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}(s)$,

$$\|\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}(s)\|_2 = \|\boldsymbol{P}_{11}(s) + \boldsymbol{P}_{12}(s)[\boldsymbol{I} - \boldsymbol{F}(s)\boldsymbol{P}_{22}(s)]^{-1}\boldsymbol{F}(s)\boldsymbol{P}_{21}(s)\|_2 < 1, \tag{7.50}$$

is minimized. For SISO systems, the $\mathcal{H}_2$ optimal control problem can be represented as

$$\min_{\boldsymbol{F}(s)} \|\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}\|_2 = \min_{\boldsymbol{F}(s)} \sqrt{\frac{1}{\pi}\int_0^\infty \boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}(-\mathrm{j}\omega)\boldsymbol{T}_{\boldsymbol{y}_1\boldsymbol{u}_1}(\mathrm{j}\omega)\mathrm{d}\omega}. \tag{7.51}$$

The $\mathcal{H}_2$ optimal control problem is equivalent to the LQG/LTR problem as explained in the following:

- The Kalman filter is expressed by

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}_2\boldsymbol{u}_2 + \boldsymbol{K}_f(\boldsymbol{y}_2 - \boldsymbol{C}_2\hat{\boldsymbol{x}} - \boldsymbol{D}_{22}\boldsymbol{u}_2) \tag{7.52}$$

  with

$$\boldsymbol{K}_f = (\boldsymbol{\Sigma}\boldsymbol{C}_2^{\mathrm{T}} + \boldsymbol{N}_f)\boldsymbol{\Theta}^{-1} = (\boldsymbol{\Sigma}\boldsymbol{C}_2^{\mathrm{T}} + \boldsymbol{B}_1\boldsymbol{D}_{21}^{\mathrm{T}})(\boldsymbol{D}_{21}\boldsymbol{D}_{21}^{\mathrm{T}})^{-1}, \tag{7.53}$$

  where $\boldsymbol{\Sigma}$ is a symmetrical matrix satisfying the ARE

$$\boldsymbol{\Sigma}\boldsymbol{A}^{\mathrm{T}} + \boldsymbol{A}\boldsymbol{\Sigma} - (\boldsymbol{\Sigma}\boldsymbol{C}_2^{\mathrm{T}} + \boldsymbol{N}_f)\boldsymbol{\Theta}^{-1}(\boldsymbol{C}_2\boldsymbol{\Sigma} + \boldsymbol{N}_f^{\mathrm{T}}) + \boldsymbol{\Xi} = \boldsymbol{0}. \tag{7.54}$$

- The full state feedback $\boldsymbol{u}_2 = -\boldsymbol{K}_c\hat{\boldsymbol{x}}$ with

$$\boldsymbol{K}_c = \boldsymbol{R}^{-1}(\boldsymbol{B}_2^{\mathrm{T}} + \boldsymbol{N}_c^{\mathrm{T}}) = (\boldsymbol{D}_{12}^{\mathrm{T}}\boldsymbol{D}_{12})^{-1}(\boldsymbol{B}_2^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{D}_{12}^{\mathrm{T}}\boldsymbol{C}_1), \tag{7.55}$$

  where

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} - (\boldsymbol{P}\boldsymbol{B}_2 + \boldsymbol{N}_c)\boldsymbol{R}^{-1}(\boldsymbol{B}_2^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{N}_c^{\mathrm{T}}) + \boldsymbol{Q} = \boldsymbol{0}. \tag{7.56}$$

The observer-based $\mathcal{H}_2$ optimal controller can be compactly denoted by

$$\boldsymbol{F}_c(s) = \left[\begin{array}{c:c} \boldsymbol{A} - \boldsymbol{K}_f\boldsymbol{C}_2 - \boldsymbol{B}_2\boldsymbol{K}_c + \boldsymbol{K}_f\boldsymbol{D}_{22}\boldsymbol{K}_c & \boldsymbol{K}_f \\ \hdashline \boldsymbol{K}_c & 0 \end{array}\right]. \tag{7.57}$$

Furthermore, it can be seen that by sending $\boldsymbol{D}_{21} \to 0$, the $\mathcal{H}_2$ optimal control problem can be made equivalent to the LQG/LTR problem.

A MATLAB function `h2lqg()`, provided in the Robust Control Toolbox, is for the design of the optimal $\mathcal{H}_2$ controller. It should be noted that this optimal controller is also robust. The syntax of the function is

$$[\boldsymbol{F}_c, \boldsymbol{G}_{cl}] = \texttt{h2lqg}(\boldsymbol{G})$$

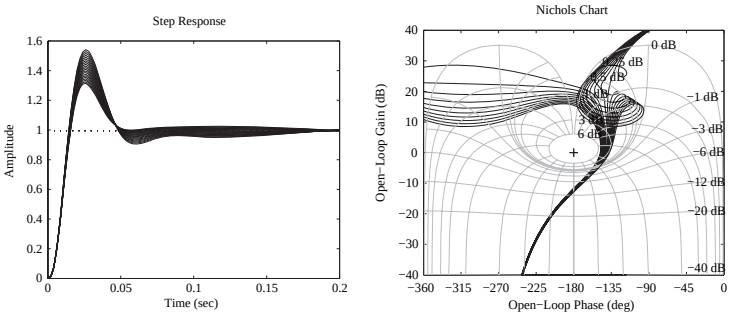
where, as before, $\boldsymbol{G}$ is the two-port state space description of the robust control problem, and the returned variables $\boldsymbol{F}_c$ and $\boldsymbol{G}_{cl}$ are the state space representations of the controller and the closed-loop system, respectively.

Example 7.19. Consider again the problem in Example 7.14. The $\mathcal{H}_2$ optimal controller can be designed by using the following MATLAB statements:

```
>> G=tf(400,[1,2,400]); s=tf('s');
   W1=100*(0.005*s+1)^2/(0.2*s+1)^2; W3=s^2/40000;
   G_T=augtf(G,W1,[],W3); [F,ccL]=h2lqg(G_T); Gc=zpk(F)
```

with the controller

$$G_c(s) = \frac{6086.1502(s+40.73)(s^2+2s+400)}{(s+5)^2(s^2+258.4s+3.339\times 10^4)}.$$



(a) step responses (b) Nichols charts

Figure 7.27. *System responses for different δ.*

With the $\mathcal{H}_2$ controller designed in the above, the closed-loop step response of the system with different values of δ can be obtained using the following MATLAB statements:

```
>> f1=figure; f2=figure;
   for delta=-10:1:10
      G.den{1}(2)=delta; G_o2=G*Gc2; G_c2=feedback(G_o2,1);
      figure(f1); step(G_c2,0.2), hold on;
      figure(f2), nichols(G_o2), hold on
   end
   figure(f2), axis([-360,0,-40,40]), grid
```

The results are summarized in Figure 7.27(a), and the Nichols charts for different values of δ are compared in Figure 7.27(b). It can be seen that the controller is robust to the changes in the plant model. Unfortunately, yet interestingly, for this example the dynamic behavior is not as good as the $\mathcal{H}_\infty$ optimal controller obtained in the previous section. However, we remark that there is no definite conclusion that the $\mathcal{H}_\infty$ optimal controller is, in general, better than the $\mathcal{H}_2$ optimal controller under the same robust control setup.

7.5 The Effects of Weighting Functions in $\mathcal{H}_\infty$ Control

In this section, via several examples we will illustrate how the weighting functions affect the performance of a robust control system.

Example 7.20. Consider the system model given in [82]

$$G(s) = \frac{-6.4750s^2 + 4.0302s + 175.7700}{s(5s^3 + 3.5682s^2 + 139.5021s + 0.0929)}.$$

For the sensitivity problem, select a weighting function as

$$W_1(s) = \frac{0.9(s^2+1.2s+1)}{1.021(s+0.001)(s+1.2)(0.001s+1)}.$$

Since there is a pole at $s = 0$, the shift technique should be used in the robust controller design. We can easily design an $\mathcal{H}_\infty$ control with the following MATLAB statements:

```
>> G=tf([-6.4750,4.0302,175.77],[5,3.5682,139.5021,0.0929,0]);
   s=tf('s'); W1=0.9*(s^2+1.2*s+1)(1.012*(s+1e-3)*(s+1.2)*(1e-3*s+1));
   [a,b,c,d]=ssdata(ss(G)); p1=-0.1; a1=a-p1*eye(size(a));
   S=mksys(a1,b,c,d); TSS_=augtf(S,W1,1e-5,[]);
   C_shift=hinf(TSS_); [a2,bf,cf,df]=branch(C_shift);
   af=a2+p1*eye(size(a2)); Gc=zpk(ss(af,bf,cf,df))
```

The user will be prompted that

```
 ---------------------------------------------------------
    all tests passed -- computing H-inf controller ...
                        DONE!!!
 ---------------------------------------------------------
```

and the controller can be designed as

$$G_c(s) = \frac{-7658521.09(s-72.32)(s+2.491)(s^2+0.3334s+0.03009)(s^2+0.713s+27.9)}{(s+1000)(s+41.05)(s+4.908)(s+1.3)(s+0.101)(s^2+605.9s+1.814\times 10^5)}.$$

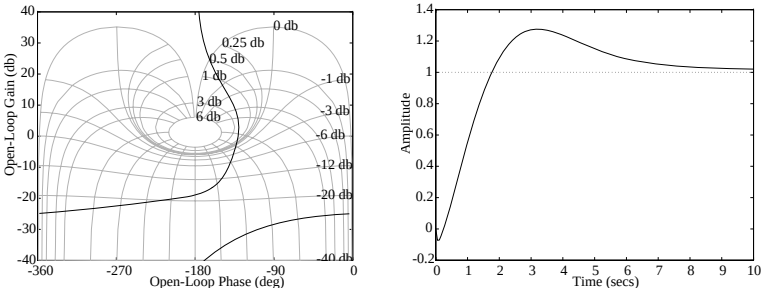
With this controller, let us draw the open-loop Nichols chart and the closed-loop step response of the system using the following MATLAB statements:

```
>> G_o=G*Gc; G_c=feedback(G_o,1); figure; step(G_c),
   figure, nichols(G_o), grid; axis([-360,0,-40,40])
```

The results are shown, respectively, in Figures 7.28(a) and (b).

To design the optimal $\mathcal{H}_\infty$ controller, use the following MATLAB statements:

```
>> [gg,C_shopt]=hinfopt(TSS_);
   [af1,bf,cf,df]=branch(C_shopt);
   af=af_shift+p1*eye(size(af1)); Gc1=zpk(ss(af,bf,cf,df))
```



(a) Nichols chart (b) step response

Figure 7.28. *Dynamic behavior of a system under the $\mathcal{H}_\infty$ controller.*

The following message will be displayed

```
                  << H-Infinity Optimal Control Synthesis >>
No    Gamma  D11<=1 P-Exist P>=0 S-Exist S>=0 lam(PS)<1 C.L.
-------------------------------------------------------------------
6 3.2500e+000  OK     OK     OK     OK     OK      OK     STAB
Iteration no. 6 is your best answer under the tolerance: 0.0100
-------------------------------------------------------------------
```

and the controller can then be designed as

$$G_c(s) = \frac{-3562725.71(s-604.2)(s+2.616)(s^2+0.3342s+0.03021)(s^2+0.713s+27.9)}{(s+1000)(s+197)(s+4.908)(s+1.3)(s+0.101)(s^2+516.5s+1.521\times 10^5)}.$$

Then, use the following MATLAB statements to perform system analysis:

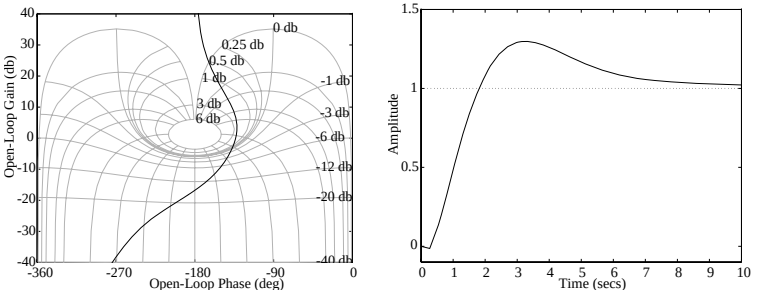
```
>> G_o1=G*Gc1; G_c1=feedback(G_o1,1); step(G_c1),
   figure, nichols(G_o1), grid; axis([-360,0,-40,40])
```

The obtained open-loop Nichols chart and the closed-loop step response of the system are shown in Figures 7.29(a) and (b), respectively.

For the above two controllers, the magnitude Bode diagrams for the sensitivity functions and the weighted sensitivity functions are obtained by the following MATLAB statements:

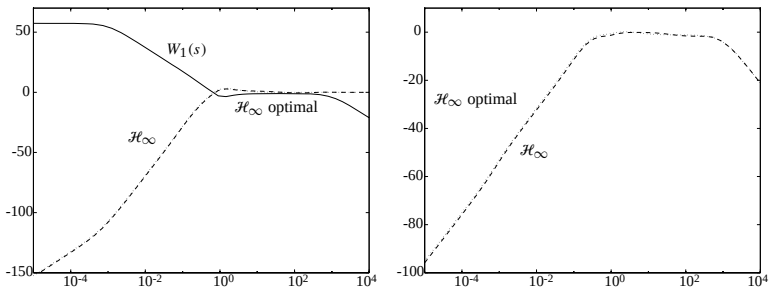
```
>> w=logspace(-5,4); g1=feedback(1,G*Gc);
   g2=feedback(1,G*Gc1);
   [m1,p1]=bode(g1,w); m1=20*log10(m1(:));
   [m2,p2]=bode(g2,w); m2=20*log10(m2(:));
   [m,p]=bode(tf(nW1,dW1),w); m=20*log10(m(:));
   semilogx(w,m,'-',w,m1,'--',w,m2,':'),
   figure; semilogx(w,m+m1,'--',w,m+m2,':')
```

with the results compared in Figures 7.30(a) and (b), respectively. It can be seen that the weighted sensitivity functions are all below the 0dB line, which means that the design specifications are met.



(a) Nichols chart (b) step response

Figure 7.29. *Dynamic behavior of a system with the optimal $\mathcal{H}_\infty$ controller.*



(a) sensitivity (b) weighted sensitivity

Figure 7.30. *Magnitude Bode diagrams with different controllers.*

Example 7.21. Consider again the double integrator problem given in Example 7.16. Let us change the weighting functions such that [85]

$$W_1(s) = \frac{\beta(\alpha s^2 + 2\zeta_1\omega_{1_c}\sqrt{\alpha}s + \omega_{1_c}^2)}{\beta s^2 + 2\zeta_2\omega_{1_c}\sqrt{\beta}s + \omega_{1_c}^2} \text{ with } \alpha\text{=1.5, } \beta\text{=100, } \omega_{1_c}\text{=3, } \zeta_1 = \zeta_2\text{=0.7.}$$

Using the following MATLAB statements:

```
>> G=tf(1,[1,0,0]); s=tf('s'); [a,b,c,d]=ssdata(ss(G));
   p1=-0.1; a1=a-p1*eye(size(a)); G_shift=ss(a1,b,c,d);
   beta=100; alpha=1.5; w1c=3; zeta1=0.7; zeta2=0.7;
   w1=tf(beta*[alpha 2*zeta1*w1c*sqrt(alpha) w1c*w1c],...
       [beta 2*zeta2*w1c*sqrt(beta) w1c*w1c]);
   TSS_shift=augtf(G_shift,w1,[],s^2/100);
   [gg,ss_Fopt_shift]=hinfopt(TSS_shift);
   [a2,bf,cf,df]=branch(ss_Fopt_shift);
   af=af_shift+p1*eye(size(a2)); Gc=zpk(ss(af,bf,cf,df))
```

the messages displayed are

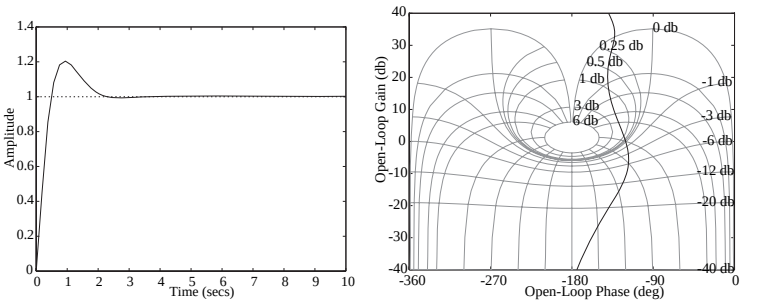
```
                << H-Infinity Optimal Control Synthesis >>
No   Gamma     D11<=1  P-Exist  P>=0  S-Exist  S>=0  lam(PS)<1 C.L.
-------------------------------------------------------------------
7 5.7812e-001   OK      OK       OK     OK       OK       OK    STAB
Iteration no. 7 is your best answer under the tolerance:   0.0100.
-------------------------------------------------------------------
```

and the optimal $\mathcal{H}_\infty$ controller can then be designed such that

$$G_c(s) = \frac{42944.8733(s+1.73)(s+0.1798)(s+0.2235)}{(s+497.9)(s+27.19)(s^2+0.62s+0.142)}.$$

To check the time and frequency performance, use the following MATLAB commands:

```
>> G_o=G*Gc; G_c=feedback(G_o,1); step(G_c),
   figure, nichols(G_o), grid; axis([-360,0,-40,40])
```

(a) step responses (b) Nichols charts

Figure 7.31. *System responses with optimal $\mathcal{H}_\infty$ controllers.*

to get the closed-loop step response and the open-loop Nichols chart shown in Figures 7.31(a) and (b), respectively. It can be seen that with the newly assigned weighting functions, the speed of the system responses is significantly increased.

Let us further change ω_{1_c} to different values. By the following MATLAB commands:

```
>> f1=figure; f2=figure;
   for w1c=1:10
      w1=tf(beta*[alpha 2*zeta1*w1c*sqrt(alpha) w1c*w1c],...
             [beta 2*zeta2*w1c*sqrt(beta) w1c*w1c]);
      TSS_shift=augtf(G_shift,w1,[],s^2/100);
      [gg,ss_Fopt_shift]=hinfopt(TSS_shift);
      [af_shift,bf,cf,df]=branch(ss_Fopt_shift);
      af=af_shift+p1*eye(size(af_shift));
      Gc=ss(af,bf,cf,df); G_o=G*Gc; G_c=feedback(G_o,1);
      figure(f1); step(G_c); hold on;
      figure(f2); nichols(G_o), hold on
   end
   figure(f2), axis([-360,0,-40,40]); grid
```

the message for each ω_{1_c} is displayed and summarized as

```
                << H-Infinity Optimal Control Synthesis >>
No   Gamma     D11<=1  P-Exist  P>=0  S-Exist  S>=0  lam(PS)<1 C.L.
-------------------------------------------------------------------
9 6.3672e-001 OK      OK       OK     OK       OK       OK    STAB 7
6.0938e-001   OK      OK       OK     OK       OK       OK    STAB 7
5.7812e-001   OK      OK       OK     OK       OK       OK    STAB 7
5.4688e-001   OK      OK       OK     OK       OK       OK    STAB 9
5.1953e-001   OK      OK       OK     OK       OK       OK    STAB 9
4.8828e-001   OK      OK       OK     OK       OK       OK    STAB 8
4.6094e-001   OK      OK       OK     OK       OK       OK    STAB 5
4.3750e-001   OK      OK       OK     OK       OK       OK    STAB 8
4.1406e-001   OK      OK       OK     OK       OK       OK    STAB 7
3.9062e-001   OK      OK       OK     OK       OK       OK    STAB
-------------------------------------------------------------------
```

The closed-loop step responses and the open-loop Nichols charts for different values of ω_{1_c} are compared in Figures 7.32(a) and (b), respectively. Note that the above displayed

messages about $\mathcal{H}_\infty$ design are only for the converged value of γ for each ω_{1c}. Clearly, with a different ω_{1c}, a different $\mathcal{H}_\infty$ controller is designed using a different weighting function. As shown in Figures 7.32(a) and (b), the dynamic behavior of the system is greatly improved. Specifically, when ω_{1c} increases, the dynamic response tends to be faster.

From the above examples, it can be concluded that the performance of the system is largely dependent upon the selection of the weighting functions.

We have shown that the performance of the robust control system is largely dependent upon the selection of the weighting functions. However, a proper choice of the weighting functions is not an easy task. One has to embed certain design experience, intuition, and domain knowledge in the robust controller design. In this section, we provide a practical and useful method for determining the weighting functions.

Recall the "standard functions" defined in Sec. 5.4. Assume that the expected standard function, which minimizes the ITAE criterion, is represented by $G_T(s)$. Then, we can set $G_T = \widehat{G}/(1+\widehat{G})$, where $\widehat{G}$ is the equivalent open-loop model for the standard function if the unity negative feedback is assumed. The sensitivity function can then be written as $S_T = 1/(1+\widehat{G})$. From the above two formulae, it is readily seen that the "standard sensitivity function" $S_T(s)$ can be written as $S_T(s) = 1 - G_T(s)$, which is a proper transfer function.

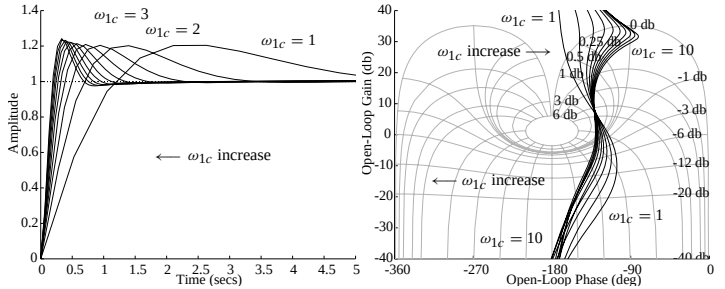
To improve the plant with a dynamic behavior similar to $G_T(s)$, a reasonable weighting function $W_1(s)$ can be selected as $W_1(s) = S_T^{-1}(s)$.

It is worth mentioning that, theoretically, there is a pole at $s = 0$ in the weighting function $W_1(s)$ thus selected. In the current version of the Robust Control Toolbox, this will cause computational problem. A practical solution is to replace the constant term in the denominator by a small positive constant, e.g., $\rho = 0.001$.

Example 7.22. Consider the unstable plant model given by

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

Select a third-order standard transfer function with an ITAE criterion for the type I format in Table 5.1. The weighting function $W_1(s)$ with different specified natural frequencies ω_n



(a) step responses (b) Nichols charts

Figure 7.32. *System responses with an $\mathcal{H}_\infty$ controller.*

can be obtained using the following MATLAB statements:

```
>> for wn=[1,10,50,100],
      GT=std_tf(wn,3); W1=inv(1-GT);
      [m,p,w]=bode(W1); semilogx(w,20*log10(m(:)')); hold on
   end
```

The magnitude Bode diagrams for different ω_n are shown in Figure 7.33(a), and it can be seen that with larger values of ω_n, the penalties on the low frequency response are increased, which will reduce the sensitivity of the system in low frequencies.

The optimal $\mathcal{H}_\infty$ controllers with different $W_1(s)$ under different ω_n can be designed by using the following MATLAB statements:

```
>> G=tf([1,5],[1,-2,4]); [a,b,c,d]=ssdata(ss(G));
   for wn=[1,10,50,100]
      GT=std_tf(wn,3); W1=inv(1-GT); W1.den{1}(end)=1e-3;
      TSS_=augtf(G,W1,1e-5,[]); [gg,Gc]=hinfopt(TSS_);
      step(feedback(G*Gc,1),1); hold on
   end
   xlim([0,1]), zpk(Gc)
```

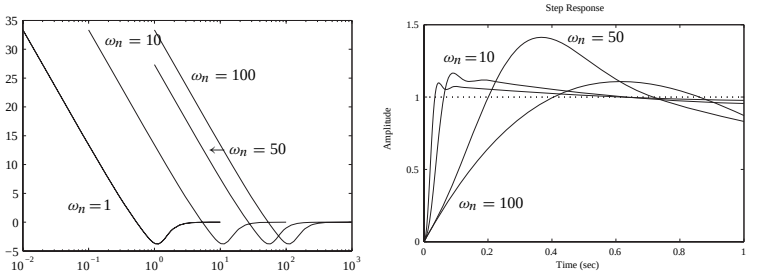
The messages displayed are

```
                << H-Infinity Optimal Control Synthesis >>
No    Gamma    D11<=1  P-Exist  P>=0  S-Exist  S>=0  lam(PS)<1  C.L.
-------------------------------------------------------------------------
8 9.9219e-001   OK      OK       OK     OK       OK      OK       STAB
8 9.9219e-001   OK      OK       OK     OK       OK      OK       STAB
8 9.9219e-001   OK      OK       OK     OK       OK      OK       STAB
8 9.9219e-001   OK      OK       OK     OK       OK      OK       STAB
-------------------------------------------------------------------------
```

The controller is then designed as

$$G_c(s) = \frac{3207304.3993(s^2 + 1.841s + 3.688)(s^2 + 173.5s + 2.712\times 10^5)}{(s + 8.018\times 10^5)(s+5)s(s^2 + 175s + 2.15\times 10^4)}.$$

The closed-loop step responses are compared in Figure 7.33(b). It can be seen that for larger ω_n, the system responses are more satisfactory.



(a) Bode magnitude of $W_1(s)$ (b) step response

Figure 7.33. *System responses with optimal $\mathcal{H}_\infty$ controllers.*

Example 7.23. Consider the double integrator problem again. Let us now use the second-order standard ITAE type I system as its reference. For different values of natural frequencies, with the following MATLAB statements:

```
>> G=tf(1,[1,0,0]); f1=figure; f2=figure; p1=-0.1;
   [a,b,c,d]=ssdata(ss(G)); a1=a-p1*eye(size(a));
   G1=ss(a1,b,c,d); for wn=[10,50,100]
      GT=std_tf(wn,2); W1=inv(1-GT); W1.den{1}(end)=1e-3;
      TSS_=augtf(G1,W1,1e-5,[]); [gg,Gc_shift]=hinfopt(TSS_);
      [af_shift,bf,cf,df]=branch(Gc_shift);
      af=af_shift+p1*eye(size(af_shift));
      Gc=ss(af,bf,cf,df); G_o=G*Gc; G_c=feedback(G_o,1);
      figure(f2); nichols(G_o); hold on;
      figure(f1); step(G_c,2); hold on;
   end
   figure(f2), grid, axis([-360,0,-40,40]), zpk(Gc)
```

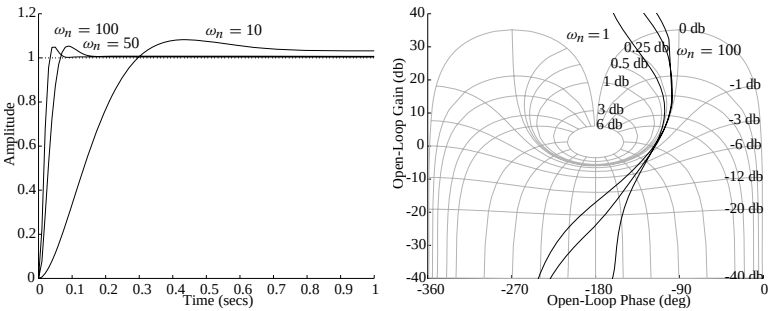
the message for each ω_n is displayed and summarized as

```
          << H-Infinity Optimal Control Synthesis >>
No    Gamma    D11<=1  P-Exist  P>=0  S-Exist  S>=0  lam(PS)<1  C.L.
-------------------------------------------------------------------------
8 9.9219e-001   OK      OK      OK     OK      OK       OK      STAB
8 9.9219e-001   OK      OK      OK     OK      OK       OK      STAB
8 9.9219e-001   OK      OK      OK     OK      OK       OK      STAB
-------------------------------------------------------------------------
```

The controller designed is

$$G_c(s) = \frac{43351967.8308(s+917)(s^2+0.3989s+0.03983)}{(s+3218)(s+1238)(s+141.1)(s+0.1)}.$$

We can design a family of optimal $\mathcal{H}_\infty$ controllers. The closed-loop step responses and open-loop Nichols charts with different ω_n are compared, respectively, in Figures 7.34(a) and (b). Clearly, when $\omega_n = 100$ rad/sec, the response of the system is satisfactory. In this case, the response is faster compared with that using the weighting functions $W_1(s)$ and $W_3(s)$ given in Example 7.21.



(a) step response (b) Nichols charts

Figure 7.34. *System responses with optimal $\mathcal{H}_\infty$ controllers.*

Problems

1. Design a Kalman filter for the system given by

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1.25 & 1.25 & 0 & 0 \\ 1.25 & -1.25 & 0 & 0 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} [u(t) + \xi(t)], \\ y(t) = [2, 1, 3, 4]\boldsymbol{x}(t) + \theta(t), \end{cases}$$

   where the variances of disturbance signals $\xi(t)$ and $\sigma(t)$ are, respectively, $\mathrm{E}[\xi^2] = 1.25 \times 10^{-3}$ and $\mathrm{E}[\theta^2] = 2.25 \times 10^{-5}$, and furthermore, $\xi(t)$ and $\theta(t)$ are mutually independent.

2. Select a weighting matrix $\boldsymbol{Q}$ and assume that $R = 1$. Design an LQG controller for the system given in the previous problem. Furthermore, design an observer-based controller for the same plant and find the gain and phase margins of the compensated system. Compare the time and frequency domain responses using MATLAB.

3. In the above problem, check whether the return difference transfer function with the LQG controller matches that under direct state feedback. If there is a mismatch, design an LQG/LTR controller (find a suitable value of q) and then compare the responses.

4. Create a tree variable in a MATLAB workspace for the following state space model:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & -2 & 0 \\ -1 & 0 & 2 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} u, \quad y = [1, 2, 3]\boldsymbol{x} + 4u.$$

5. Perform the coprime factorization for the following models:

$$\text{(a)} \quad G_1(s) = \frac{5s+2}{s^2+5s+4},$$
$$\text{(b)} \quad G_2(s) = \frac{s-1}{s(s-2)},$$
$$\text{(c)} \quad G_3(s) = \frac{(s-2)^2(s^2-s-1)}{(s-1)^2(s+1)^2+2}.$$

   Find the transfer functions $X(s)$ and $Y(s)$ satisfying the Bezout equation.

6. For the plant model $G(s) = \dfrac{1}{(0.01s+1)^2}$, with the weighting functions $W_1(s) = \dfrac{10}{s^3+2s^2+2s+1}$ and $W_3(s) = \dfrac{10s+1}{20(0.01s+1)}$, do the following:

(a) Write the two-port description of the system with weighting functions.
(b) Design the optimal $\mathcal{H}_\infty$ controller.
(c) Draw the closed-loop step response and open-loop Nichols chart.
(d) Evaluate the dynamic performance of the controlled system.
(e) Design the $\mathcal{H}_2$ optimal controller and compare the control performance.

7. Consider the following plant models:

$$\text{(a)} \quad G(s) = \frac{10}{(s+1)(s+2)(s+3)(s+4)},$$

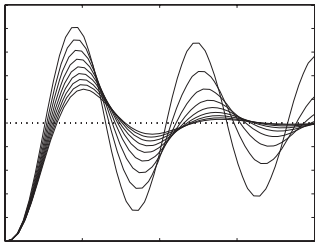
$$\text{(b)} \quad G(s) = \frac{10(-s+3)}{s(s+1)(s+2)}.$$

Design the optimal $\mathcal{H}_\infty$ controllers for the minimum sensitivity problems. (Note: The standard target transfer functions can be used in the controller design.) Perform time and frequency domain analyses for the system. Draw the magnitude Bode plots for the sensitivity and complementary sensitivity functions.

8. In Problem 7(b), if the optimal $\mathcal{H}_\infty$ controller is designed, check the stability if the numerator in the plant model is changed to $10(s+3)$. Verify the result using time and frequency domain analysis tools.

9. Using the weighting function based on the standard transfer functions, compare the robust controller designed for the sensitivity problem for the plant models given in Problem 7. Find qualitatively the effect of natural frequency of the standard transfer functions.

10. Design $\mathcal{H}_\infty$ and $\mathcal{H}_2$ controllers for the plant models given in Problem 7.5 for the sensitivity problem. Perform frequency domain analysis of the controlled system.



Chapter 8

Fractional-Order Controller: An Introduction

Using the notion of fractional-order may be a more realistic step because real processes are generally "fractional" [86]. However, for many real processes, fractionality is very low. A typical example of a noninteger, (fractional-) order system is the voltage–current relationship of a semi-infinite lossy resistor and capacitor (RC) line or the diffusion of heat in a semi-infinite solid, where the heat flow $q(t)$ is naturally equal to the semiderivative of temperature $T(t)$ [87], as described by the following simple fractional-order differential equation (FODE):

$$\frac{\mathrm{d}^{0.5}T(t)}{\mathrm{d}t^{0.5}} = q(t).$$

Clearly, using an integer-order ordinary differential equation (ODE) description for the above system may differ significantly from the actual situation. However, the fact that the integer-order dynamic models are more welcome is probably due to the absence of solution methods for FODEs. Details of past and present progress in the analysis of dynamic systems modeled by FODEs can be found in [88–95]. For example, PID (proportional integral derivative) controllers, which have been dominating industrial controllers, have been modified using the notion of a fractional-order integrator and differentiator. It has been shown that two extra degrees of freedom from the use of a fractional-order integrator and differentiator make it possible to further improve the performance of traditional PID controllers. In addition, the plant to be controlled can also be modeled as a dynamic system described by an FODE. For fractional-order systems, the fractional controller CRONE was developed in [96], while [89, 97, 98] presented the PD^δ controller and [99] proposed the $\mathrm{PI}^\lambda\mathrm{D}^\delta$ controller.

In theory, control systems can include both the fractional-order dynamic system or plant to be controlled and the fractional-order controller. However, in control engineering, it is a common practice to consider only the fractional-order controller. This is due to the fact that the plant model may have already been obtained as an integer-order model in a classical sense. In most cases, our objective is to apply fractional-order control (FOC) to enhance system control performance. Therefore, in this chapter we will concentrate on the scenario in which the controller is fractional-order.

This chapter serves as an introduction to the essentials of FOC for control engineering practice, with an emphasis on how to analyze and realize fractional-order systems using MATLAB. For a broader introductory coverage of fractional-order calculus and its applications in engineering, we refer the interested reader to the textbook [100].

This chapter is organized as follows. In Sec. 8.1, definitions and properties of fractional-order calculus are briefly introduced, followed by frequency and time domain analysis of fractional-order linear systems in Sec. 8.2. Then, in Sec. 8.3 filter approximations to fractional-order differentiators are introduced using Oustaloup's recursive scheme and its refined version. With this filter approximation, using Simulink, a simulation method for a general nonlinear fractional-order dynamic system is proposed with an illustrative example. Since the fractional-order controller after finite dimensional approximation is usually of a very high order, controller order reduction is discussed and demonstrated in Sec. 8.4. Finally, we present some controller design case studies for fractional-order systems in Sec. 8.5.

Note that this chapter, like previous chapters, is designed so that the text and illustrative MATLAB scripts flow in a natural and smooth manner. We hope that this design enables readers to quickly get started on problem solving. It is worth mentioning that the design of a MATLAB class for a fractional-order transfer function is demonstrated thoroughly in the chapter.

8.1 Fractional-Order Calculus and Its Computations

In a letter to Hôpital in 1695, Leibniz raised the following question: Can the meaning of derivatives with integer order $\mathrm{d}^n y(x)/\mathrm{d}x^n$ be generalized to derivatives with noninteger orders, so that in general $n \in \mathscr{C}$? (Here $\mathscr{C}$ is the set for all complex numbers.) Hôpital was a bit curious about this question and replied with another question to Leibniz: What if $n = 1/2$? Leibniz, in a letter dated September 30, 1695, replied: It will lead to a paradox, from which one day useful consequences will be drawn.

The question raised by Leibniz for a fractional-order derivative has been a topic of ongoing study in the last 300 years. Several mathematicians contributed to this subject over the years. People like Liouville, Riemann, and Weyl made major contributions to the theory of fractional-order calculus. So, the term "fractional-order calculus" is by no means new. It is a generalization of ordinary differentiation by noninteger derivatives. The subject is as old as the calculus of differentiation and goes back to the 17th century when Leibniz and Newton invented calculus. The theory of fractional-order derivatives was developed mainly in the 19th century. For more information, see [91, 93, 101, 102].

In the development of fractional-order calculus, there appeared different definitions of fractional-order differentiations and integrations. Some of the definitions extend directly from integer-order calculus. The well-established definitions include the Cauchy integral formula, the Grünwald–Letnikov definition, the Riemann–Liouville definition, and the Caputo definition. The definitions will be summarized first, and then properties will be given.

8.1.1 Definitions of Fractional-Order Calculus

Definition 8.1 (*Cauchy's fractional-order integration formula*). This definition is a general extension of the integer-order Cauchy formula

$$\mathscr{D}^{\gamma} f(t) = \frac{\Gamma(\gamma+1)}{2\pi \mathrm{j}} \int_C \frac{f(\tau)}{(\tau - t)^{\gamma+1}} \mathrm{d}\tau, \tag{8.1}$$

where C is the smooth curve encircling the single-valued function $f(t)$.

Definition 8.2 (*Grünwald–Letnikov definition*). The definition is defined as

$${}_a\mathscr{D}_t^{\alpha} f(t) = \lim_{h\to 0} \frac{1}{h^{\alpha}} \sum_{j=0}^{[(t-a)/h]} (-1)^j \binom{\alpha}{j} f(t - jh), \tag{8.2}$$

where $w_j^{(\alpha)} = (-1)^j \binom{\alpha}{j}$ represents the coefficients of the polynomial $(1-z)^{\alpha}$. The coefficients can also be obtained recursively from

$$w_0^{(\alpha)} = 1, \quad w_j^{(\alpha)} = \left(1 - \frac{\alpha+1}{j}\right) w_{j-1}^{(\alpha)}, \ j = 1, 2, \ldots. \tag{8.3}$$

Based on the Definition 8.2, the fractional-order differentiation can easily be calculated from

$${}_a\mathscr{D}_t^{\alpha} f(t) = \lim_{h\to 0} \frac{1}{h^{\alpha}} \sum_{j=0}^{[(t-a)/h]} (-1)^j \binom{\alpha}{j} f(t - jh) \approx \frac{1}{h^{\alpha}} \sum_{j=0}^{[(t-a)/h]} w_j^{(\alpha)} f(t - jh). \tag{8.4}$$

Assuming that the step size h is small enough, we see that (8.4) can be used to evaluate the differentiations of the given function. It can be shown [93] that the accuracy of the method is $\mathrm{o}(h)$. Thus, based on the Grünwald–Letnikov definition, the following MATLAB function can be written to evaluate the fractional-order differentiation [103]:

```
function dy=glfdiff(y,t,gam)
h=t(2)-t(1); dy(1)=0; y=y(:); t=t(:);
w=1; for j=2:length(t), w(j)=w(j-1)*(1-(gam+1)/(j-1)); end
for i=2:length(t), dy(i)=w(1:i)*[y(i:-1:1)]/h^gam; end
```

The syntax of the function is `dy=glfdiff(y,t,γ)`, where $\boldsymbol{y}, \boldsymbol{t}$ are, respectively, the vectors composed of the samples and the time instances. The time vector $\boldsymbol{t}$ is assumed to be evenly distributed. γ is the order of fractional-order differentiation. The returned vector $\boldsymbol{d}_y$ is the vector of the fractional-order derivatives.

Definition 8.3 (*Riemann–Liouville fractional-order differentiation*). The fractional-order integration is defined as

$${}_a\mathscr{D}_t^{-\alpha} f(t) = \frac{1}{\Gamma(\alpha)} \int_a^t (t-\tau)^{\alpha-1} f(\tau)\mathrm{d}\tau, \tag{8.5}$$

where $0 < \alpha < 1$, and a is the initial time instance, often assumed to be zero, i.e., $a = 0$. The differentiation is then denoted as ${}_{\mathscr{D}_t^{-\alpha}} f(t)$.

The Riemann–Liouville definition is the most widely used definition in fractional-order calculus. The subscripts on both sides of $\mathscr{D}$ represent, respectively, the lower and upper bounds in the integration [104].

Such a definition can also be extended to fractional-order differentiations when the order satisfies $n-1 < \beta \leq n$. The fractional-order differentiation is then defined as

$$ {}_a\mathscr{D}_t^{\beta} f(t) = \frac{\mathrm{d}^n}{\mathrm{d}t^n}\left[{}_a\mathscr{D}_t^{-(n-\beta)} f(t)\right] = \frac{1}{\Gamma(n-\beta)}\frac{\mathrm{d}^n}{\mathrm{d}t^n}\left[\int_a^t \frac{f(\tau)}{(t-\tau)^{\beta-n+1}}\mathrm{d}\tau\right]. \tag{8.6} $$

Definition 8.4 (*Caputo's definition of fractional-order differentiation*). Caputo's definition is given by

$$ {}_0\mathscr{D}_t^{\alpha} y(t) = \frac{1}{\Gamma(1-\gamma)}\int_0^t \frac{y^{(m+1)}(\tau)}{(t-\tau)^{\gamma}}\mathrm{d}\tau, \tag{8.7} $$

where $\alpha = m + \gamma$, m is an integer, and $0 < \gamma \leq 1$. Similarly, Caputo's fractional-order integration is defined as

$$ {}_0\mathscr{D}_t^{\gamma} = \frac{1}{\Gamma(-\gamma)}\int_0^t \frac{y(\tau)}{(t-\tau)^{1+\gamma}}\mathrm{d}\tau, \quad \gamma < 0. \tag{8.8} $$

It can be shown [93] that for a class of real functions, the fractional-order differentiations from the Grünwald–Letnikov and Riemann–Liouville definitions are identical.

8.1.2 Properties of Fractional-Order Differentiations

The fractional-order differentiation has the following properties [105]:

1. The fractional-order differentiation ${}_0\mathscr{D}_t^{\alpha} f(t)$, with respect to t of an analytic function $f(t)$, is also analytical.
2. The fractional-order differentiation is exactly the same with integer-order one, when $\alpha = n$ is an integer. Also ${}_0\mathscr{D}_t^{0} f(t) = f(t)$.
3. The fractional-order differentiation is linear; i.e., for any constants a, b, one has
$$ {}_0\mathscr{D}_t^{\alpha}\left[af(t) + bg(t)\right] = a\,{}_0\mathscr{D}_t^{\alpha} f(t) + b\,{}_0\mathscr{D}_t^{\alpha} g(t). \tag{8.9} $$
4. Fractional-order differentiation operators satisfy the commutative-law, and also satisfy
$$ {}_0\mathscr{D}_t^{\alpha}\left[{}_0\mathscr{D}_t^{\beta} f(t)\right] = {}_0\mathscr{D}_t^{\beta}\left[{}_0\mathscr{D}_t^{\alpha} f(t)\right] = {}_0\mathscr{D}_t^{\alpha+\beta} f(t) \tag{8.10} $$
5. The Laplace transform of fractional-order differentiation is defined as
$$ \mathscr{L}\left[{}_0\mathscr{D}_t^{\alpha} f(t)\right] = s^{\alpha}\mathscr{L}[f(t)] - \sum_{k=1}^{n-1} s^k\left[{}_0\mathscr{D}_t^{\alpha-k-1} f(t)\right]_{t=0}. \tag{8.11} $$
In particular, if the derivatives of the function $f(t)$ are all equal to 0 at $t = 0$, one has $\mathscr{L}[{}_0\mathscr{D}_t^{\alpha} f(t)] = s^{\alpha}\mathscr{L}[f(t)]$.

8.2 Frequency and Time Domain Analysis of Fractional-Order Linear Systems

The fractional-order system is the direct extension of classical integer-order systems. The fractional-order system is established upon the fractional-order differential equations, and the fractional-order transfer function of a single variable system can be defined as

$$ G(s) = \frac{b_1 s^{\gamma_1} + b_2 s^{\gamma_2} + \cdots + b_m s^{\gamma_m}}{a_1 s^{\eta_1} + a_2 s^{\eta_2} + \cdots + a_{n-1} s^{\eta_{n-1}} + a_n s^{\eta_n}}, \tag{8.12} $$

where b_i, a_i are real numbers and the orders γ_i, η_i of the numerator and the denominator can also be real numbers. The analysis of the fractional-order Laplace transformations and their inverse is very complicated. The closed-form solutions to the problems are not possible in general.

8.2.1 Fractional-Order Transfer Function Modeling

For the fractional-order transfer function model in (8.12), it can be seen that if the coefficients and the orders of the numerator and denominator are given, the model can be established. Thus, an "`fotf`" class can be constructed by creating the `@fotf` directory and writing in the directory an `fotf()` function as follows:

```
function G=fotf(a,na,b,nb)
if nargin==0,
   G.a=[]; G.na=[]; G.b=[]; G.nb=[]; G=class(G,'fotf');
elseif isa(a,'fotf'), G=a;
elseif nargin==1 & isa(a,'double'), G=fotf(1,0,a,0);
else,
   ii=find(abs(a)<eps); a(ii)=[]; na(ii)=[];
   ii=find(abs(b)<eps); b(ii)=[]; nb(ii)=[];
   G.a=a; G.na=na; G.b=b; G.nb=nb; G=class(G,'fotf');
end
```

The syntax of the function is `G=fotf(a,η,b,γ)`, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the coefficients of the denominator and the numerator, respectively, while $\boldsymbol{\eta}$ and $\boldsymbol{\gamma}$ are the order sequences in the denominator and the numerator, respectively.

A display function should also be created for the `fotf` class. The file should also be saved in the `@fotf` directory such that

```
function display(G)
sN=polydisp(G.b,G.nb); sD=polydisp(G.a,G.na); s=' ';
nm=max([length(sN),length(sD)]); nn=length(sN); nd=length(sD);
disp([char(s*ones(1,floor((nm-nn)/2))) sN]), disp(char('-'*ones(1,nm)));
disp([char(s*ones(1,floor((nm-nd)/2))) sD])
function strP=polydisp(p,np)
P=''; [np,ii]=sort(np,'descend'); p=p(ii);
for i=1:length(p), P=[P,'+',num2str(p(i)),'s^{',num2str(np(i)),'}']; end
P=P(2:end); P=strrep(P,'s^{0}',''); P=strrep(P,'+-','-');
P=strrep(P,'^{1}',''); P=strrep(P,'+1s','+s'); strP=strrep(P,'-1s','-s');
if length(strP)>=2, if strP(1:2)=='1s', strP=strP(2:end); end,end,
```

Example 8.1. Suppose that the fractional-order transfer function is given by

$$G(s)=\frac{-2s^{0.63}-4}{2s^{3.501}+3.8s^{2.42}+2.6s^{1.798}+2.5s^{1.31}+1.5}.$$

With the following statement, the fractional-order transfer function can be entered into the MATLAB environment:

```
>> b=[-2,-4]; nb=[0.63,0]; a=[2 3.8 2.6 2.5 1.5];
   na=[3.501,2.42,1.798,1.31,0]; G=fotf(a,na,b,nb)
```

The display of the fractional-order transfer function is

```
                    -2s^{0.63}-4
-----------------------------------------------------
2s^{3.501}+3.8s^{2.42}+2.6s^{1.798}+2.5s^{1.31}+1.5
```

A function `fotf()` can be written in the `@tf` directory to convert an integer-order transfer function to an `fotf` object:

```
function G1=fotf(G)
n=G.num{1}; d=G.den{1}; i1=find(abs(n)<eps); i2=find(abs(d)<eps);
if length(i1)>0 & i1(1)==1, n=n(i1(1)+1:end); end
if length(i2)>0 & i2(1)==1, d=d(i2(1)+1:end); end
G1=fotf(d,length(d)-1:-1:0,n,length(n)-1:-1:0);
```

8.2.2 Interconnections of Fractional-Order Blocks

Based on the newly defined `fotf` class, the `plus()`, `mtimes()` and `feedback()` functions can be written as follows:

- Plus function `plus()` for block parallel connections:

```
function G=plus(G1,G2)
a=kron(G1.a,G2.a); b=[kron(G1.a,G2.b), kron(G1.b,G2.a)]; na=[]; nb=[];
for i=1:length(G1.a), na=[na G1.na(i)+G2.na]; nb=[nb, G1.na(i)+G2.nb]; end
for i=1:length(G1.b), nb=[nb G1.nb(i)+G2.na]; end
G=unique(fotf(a,na,b,nb));
```

- Multiplication function `mtimes()` for block series connections:

```
function G=mtimes(G1,G2)
G2=fotf(G2); a=kron(G1.a,G2.a);
b=kron(G1.b,G2.b); na=[]; nb=[];
for i=1:length(G1.na), na=[na,G1.na(i)+G2.na]; end
for i=1:length(G1.nb), nb=[nb,G1.nb(i)+G2.nb]; end
G=unique(fotf(a,na,b,nb));
```

- Feedback function `feedback()` for block negative feedback connections:

```
function G=feedback(F,H)
H=fotf(H);
```



```
b=kron(F.b,H.a); a=[kron(F.b,H.b), kron(F.a,H.a)]; na=[]; nb=[];
for i=1:length(F.b), nb=[nb F.nb(i)+H.nb]; na=[na,F.nb(i)+H.nb]; end
for i=1:length(F.a), na=[na F.na(i)+H.na]; end
G=unique(fotf(a,na,b,nb));
```

- Simplification function `unique()`:

```
function G=unique(G)
[a,n]=polyuniq(G.a,G.na); G.a=a; G.na=n;
[a,n]=polyuniq(G.b,G.nb); G.b=a; G.nb=n;
function [a,an]=polyuniq(a,an)
[an,ii]=sort(an,'descend'); a=a(ii); ax=diff(an); key=1;
for i=1:length(ax)
   if ax(i)==0, a(key)=a(key)+a(key+1); a(key+1)=[]; an(key+1)=[];
   else, key=key+1; end
end
```

Other functions should also be designed, such as `minus()`, `uminus()`, `inv()`, and the files should be placed in the `@fotf` directory to overload the existing ones. The listings of these functions are not given in this text but available from the book's companion Website.

Example 8.2. Suppose in the unity negative feedback system, the system models are given by

$$G(s)=\frac{0.8s^{1.2}+2}{1.1s^{1.8}+0.8s^{1.3}+1.9s^{0.5}+0.4},\ G_c(s)=\frac{1.2s^{0.72}+1.5s^{0.33}}{3s^{0.8}}.$$

The plant and controller can be easily entered and the closed-loop system can be directly obtained with the commands

```
>> G=fotf([1.1,0.8 1.9 0.4],[1.8 1.3 0.5 0],[0.8 2],[1.2 0]);
   Gc=fotf(3,[0.8],[1.2 1.5],[0.72 0.33]); H=fotf(1,0,1,0);
   GG=feedback(G*Gc,H)
```

and the result is given by

$$G(s)=\frac{0.96s^{1.92}+1.2s^{1.53}+2.4s^{0.72}+3s^{0.33}}{3.3s^{2.6}+2.4s^{2.1}+0.96s^{1.92}+1.2s^{1.53}+5.7s^{1.3}+1.2s^{0.8}+2.4s^{0.72}+3s^{0.33}}.$$

It can be seen from the above illustrations that, although the plant and controllers are relatively simple, an extremely complicated closed-loop model may be obtained. This makes the analysis and design of the fractional-order system a difficult task.

8.2.3 Frequency Domain Analysis of Linear Fractional-Order Systems

It can be seen that, when jω is used to substitute for the variable s in the fractional-order transfer function model, the frequency domain response $G(\mathrm{j}\omega)$ can be easily evaluated. Thus, the fractional-order Bode diagrams, Nyquist plots, and Nichols charts can be easily evaluated with the function `bode()`, which is written as an overload function for the `fotf` object

```
function H=bode(G,w)
a=G.a; eta=G.na; b=G.b; g=G.nb; if nargin==1, w=logspace(-4,4); end
for i=1:length(w)
    P=b*((sqrt(-1)*w(i)).^g.'); Q=a*((sqrt(-1)*w(i)).^eta.'); H1(i)=P/Q;
end
H1=frd(H1,w); if nargout==0, bode(H1); else, H=H1; end
```

The syntax of the function is `H=bode(G,ω)`, where G is the fractional-order transfer function object and the optional argument $\boldsymbol{\omega}$ is the frequency vector.

If one wants to draw the Bode diagram, there is no need to return any variable. If frequency domain response data are needed, the response results can be found in the returned variable H. The variable H can be used in drawing the Nyquist plot and the Nichols chart by using `nyquist(H)` and `nichols(H)`, respectively.

8.2.4 Time Domain Analysis of Fractional-Order Systems

The evaluation of the time domain response of a fractional-order system is more complicated. Let us consider a special form of a fractional-order differential equation [93]

$$a_1 \mathscr{D}_t^{\eta_1} y(t) + a_2 \mathscr{D}_t^{\eta_2} y(t) + \cdots + a_{n-1} \mathscr{D}_t^{\eta_{n-1}} y(t) + a_n \mathscr{D}_t^{\eta_n} y(t) = u(t), \tag{8.13}$$

where $u(t)$ can be represented by a certain function and its fractional-order derivatives. Assume also that the output function $y(t)$ has zero initial conditions. The Laplace transform can be used to find the transfer function

$$G(s) = \frac{1}{a_1 s^{\eta_1} + a_2 s^{\eta_2} + \cdots + a_{n-1} s^{\eta_{n-1}} + a_n s^{\eta_n}}. \tag{8.14}$$

Consider the Grünwald–Letnikov definition in (8.4). The discrete form of it can be rewritten as

$${}_a\mathscr{D}_t^{\eta_i} y(t) \simeq \frac{1}{h^{\eta_i}} \sum_{j=0}^{[(t-a)/h]} w_j^{(\eta_i)} y_{t-jh} = \frac{1}{h^{\eta_i}} \left[y_t + \sum_{j=1}^{[(t-a)/h]} w_j^{(\eta_i)} y_{t-jh} \right], \tag{8.15}$$

where $w_0^{(\beta_i)}$ can be evaluated recursively from the formula (8.3). By substituting it into (8.13), the numerical solution to the fractional-order differential equation can be written as

$$y_t = \frac{1}{\sum_{i=1}^{n} \frac{a_i}{h^{\eta_i}}} \left[u_t - \sum_{i=1}^{n} \frac{a_i}{h^{\eta_i}} \sum_{j=1}^{[(t-a)/h]} w_j^{(\eta_i)} y_{t-jh} \right]. \tag{8.16}$$

For the general form of the fractional-order transfer function in (8.12), the right-hand side can equivalently be evaluated first by using the numerical method discussed earlier. The final solution can be obtained from (8.16). A MATLAB function can be written for the `fotf` object to evaluate the time domain response as follows:

```
function y=lsim(G,u,t)
a=G.a; eta=G.na; b=G.b; gamma=G.nb; nA=length(a);
h=t(2)-t(1); D=sum(a./[h.^eta]); W=[]; nT=length(t);
```



```
vec=[eta gamma]; D1=b(:)./h.^gamma(:);
y1=zeros(nT,1); W=ones(nT,length(vec));
for j=2:nT, W(j,:)=W(j-1,:).*(1-(vec+1)/(j-1)); end
for i=2:nT
   A=[y1(i-1:-1:1)]'*W(2:i,1:nA); y1(i)=(u(i)-sum(A.*a./[h.^eta]))/D;
end
for i=2:nT, y(i)=(W(1:i,nA+1:end)*D1)'*[y1(i:-1:1)]; end
```

The syntax of the function is `y=lsim(G,u,t)`, where the time vector and the input vector are defined in the variables $\boldsymbol{t}$ and $\boldsymbol{u}$, respectively. The returned vector $\boldsymbol{y}$ is the solution to the equations. If there are more points in the equation, the computation may be very slow.

An overloaded `step()` function can also be written, based on the `lsim()` function given above, as

```
function y=step(G,t)
u=ones(size(t)); y=lsim(G,u,t);
if nargout==0, plot(t,y); end
```

with `y=step(G,t)`, where G is an `fotf` object, and $\boldsymbol{t}$ should be given as an evenly distributed time vector. The step response of the system is returned in vector $\boldsymbol{y}$.

It is possible to solve the above fractional-order differential equation analytically by using the Mittag–Leffler function in two parameters, which is a generalization of the exponential function e^z. The Mittag–Leffler function in two parameters is defined as

$$\mathscr{E}_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + \beta)}, \quad (\alpha, \beta > 0). \tag{8.17}$$

Clearly, e^z is a particular case of the Mittag–Leffler function [92]:

$$\mathscr{E}_{1,1}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+1)} = \sum_{k=0}^{\infty} \frac{z^k}{k!} = \mathrm{e}^z.$$

Furthermore, one can get more particular cases for the Mittag–Leffler function in two parameters, for example,

$$\mathscr{E}_{2,1}(z) = \cosh(\sqrt{z}), \quad \mathscr{E}_{1,2}(z) = \frac{\mathrm{e}^z - 1}{z}, \quad \mathscr{E}_{2,2}(z) = \frac{\sinh(\sqrt{z})}{\sqrt{z}}, \tag{8.18}$$

$$\mathscr{E}_{1/2,1}(\sqrt{z}) = \frac{2}{\sqrt{\pi}} \mathrm{e}^{-z} \mathrm{erfc}(-\sqrt{z}). \tag{8.19}$$

The analytical solution of the n-term FODE is given in general form [92] by

$$y(t) = \frac{1}{a_n} \sum_{m=0}^{\infty} \frac{(-1)^m}{m!} \sum_{\substack{k_0+k_1+\cdots+k_{n-2}=m \\ k_0 \geq 0, \ldots, k_{n-2} \geq 0}} (m; k_0, k_1, \ldots, k_{n-2}) \prod_{i=0}^{n-2} \left(\frac{a_i}{a_n} \right)^{k_i} t^{(\beta_n - \beta_{n-1})m + \beta_n + \sum_{j=0}^{n-2} (\beta_{n-1} - \beta_j) k_j - 1} \tag{8.20}$$

$$\mathscr{E}^{(m)}_{\beta_n-\beta_{n-1},\beta_n+\sum_{j=0}^{n-2}(\beta_{n-1}-\beta_j)k_j}\left(-\frac{a_{n-1}}{a_n}t^{\beta_n-\beta_{n-1}}\right),$$

where $\mathscr{E}_{\lambda,\mu}(z)$ is the Mittag–Leffler function in two parameters as defined in (8.17) and

$$\mathscr{E}^{(n)}_{\lambda,\mu}(y)\equiv\frac{\mathrm{d}^n}{\mathrm{d}y^n}\mathscr{E}_{\lambda,\mu}(y)=\sum_{j=0}^{\infty}\frac{(j+n)!\,y^j}{j!\,\Gamma(\lambda j+\lambda n+\mu)}\quad\text{for } n=0,1,2,\ldots. \tag{8.21}$$

8.3 Filter Approximation to Fractional-Order Differentiations

It can be seen that the Grünwald–Letnikov definition gives a very good fitting to the fractional-order derivatives for given functions. However, in control system analysis and design, the definition is not useful, since the samples of the function should be known. On-line real-time fractional-order differentiation may be required in control systems. Using filters is one of the best ways to solve the problems.

8.3.1 Oustaloup's Recursive Filter

Some continuous filters have been summarized in [105]. Among the filters, the well-established Oustaloup recursive filter has a very good fitting to the fractional-order differentiators [106]. Assume that the expected fitting range is (ω_b,ω_h). The filter can be written as

$$G_f(s)=K\prod_{k=-N}^{N}\frac{s+\omega'_k}{s+\omega_k}, \tag{8.22}$$

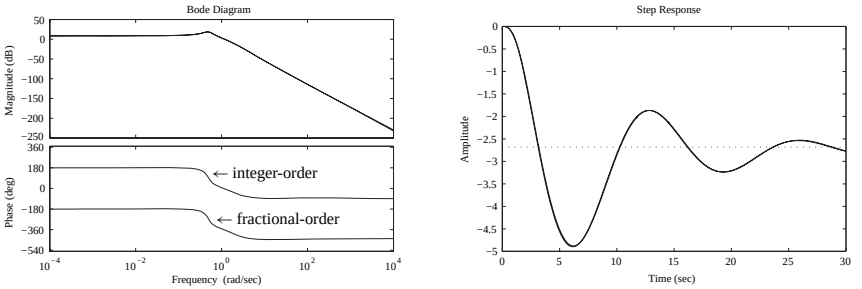
where the poles, zeros, and gain of the filter can be evaluated from (8.23) such that

$$\omega'_k=\omega_b\left(\frac{\omega_h}{\omega_b}\right)^{\frac{k+N+\frac{1}{2}(1-\gamma)}{2N+1}},\quad \omega_k=\omega_b\left(\frac{\omega_h}{\omega_b}\right)^{\frac{k+N+\frac{1}{2}(1+\gamma)}{2N+1}},\quad K=\omega_h^{\gamma}. \tag{8.23}$$

With the above algorithm, the following MATLAB function `oustafod()` can be written to design the continuous filter. Thus, the $y(t)$ signal can be filtered through the filter and the output of the filter can be regarded as an approximation to the $\mathscr{D}_t^{\gamma}y(t)$ signal.

```
function G=oustafod(r,N,wb,wh)
mu=wh/wb; k=-N:N; w_kp=(mu).^((k+N+0.5-0.5*r)/(2*N+1))*wb;
w_k=(mu).^((k+N+0.5+0.5*r)/(2*N+1))*wb;
K=wh^r; G=tf(zpk(-w_kp',-w_k',K));
```

The function can be called with `G_f=oustafod(γ,N,ω_b,ω_h)`, where γ is the order of the differentiation, $2N+1$ is the order of the filter, and the frequency fitting range is given by (ω_b,ω_h). The filter G_f can be designed such that it may fit very well within the frequency range of the fractional order differentiator.



(a) Bode diagrams (b) step response

Figure 8.1. *Time and frequency domain comparisons.*

Example 8.3. Consider a fractional-order model

$$G(s)=\frac{-2s^{0.63}-4}{2s^{3.501}+3.8s^{2.42}+2.6s^{1.798}+2.5s^{1.31}+1.5}.$$

Since the original orders are all fractional, it may not be easy to design controllers for them. Thus, a model reduction technique can be considered to reduce the order such that a low integer-order approximation can be achieved. Suppose that one wants to approximate the differentiators within the frequency range of $(10^{-3},10^4)$; the high-order term can also be approximated as $s^{3.501}=s^3s^{0.501}$, and the integer-order approximation can be obtained as

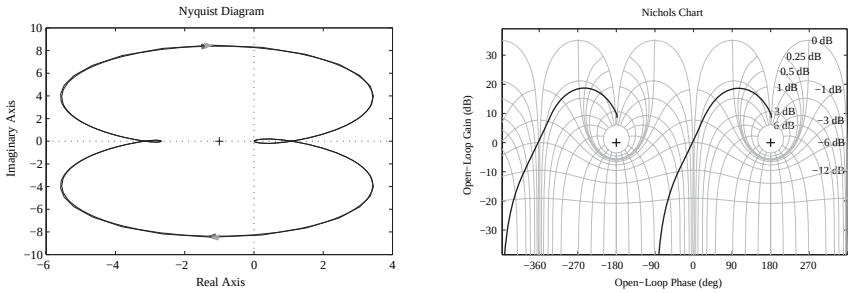
```
>> N=4; w1=1e-3; w2=1e4; g1=oustafod(0.501,N,w1,w2);
   s=tf('s');
   g2=oustafod(0.42,N,w1,w2); g3=oustafod(0.798,N,w1,w2);
   g4=oustafod(0.31,N,w1,w2); g5=oustafod(0.63,N,w1,w2);
   G1=(-2*g5-4)/(2*s^3*g1+3.8*s^2*g2+2.6*s*g3+2.5*s*g4+1.5);
```

It is found that the order of the approximation reaches 48. The exact Bode diagram and its 48th-order approximation are shown in Figure 8.1(a). The step responses of the system is obtained as shown in Figure 8.1(b). With the following MATLAB statements, it can be seen that the time response of the filter can accurately approximate the fractional-order derivatives of the system.

```
>> b=[-2 -4]; nb=[0.63 0]; a=[2 3.8 2.6 2.5 1.5];
   na=[3.501 2.42 1.798 1.31 0]; G=fotf(a,na,b,nb);
   w=logspace(-4,4,500); H=bode(G,w); bode(G1,H,{1e-4,1e4});
   figure; t=0:0.004:30; y=step(G,t); step(G1,30); line(t,y)
```

The open-loop Nyquist plots and Nichols charts can also be obtained as shown in Figure 8.2. It can be seen that the Nyquist plot accurately fits the theoretical one, while the Nichols chart is shifted by 360°, which means the two are identical:

```
>> H=bode(G,w); nyquist(G,H,{1e-4,1e4});
   figure; nichols(G,H,{1e-4,1e4}); grid
```



(a) Nyquist plots (b) Nichols plots

Figure 8.2. *Comparisons of other frequency domain plots.*

8.3.2 A Refined Oustaloup Filter

Here we introduce a new approximate realization method for the fractional-order derivative in the frequency range of interest $[\omega_b, \omega_h]$. Our proposed method here gives a better approximation than Oustaloup's method with respect to both low frequency and high frequency.

Assume that the frequency range to be fit is defined as (ω_b, ω_h). Within the prespecified frequency range, the fractional-order operator s^α can be approximated by the fractional-order transfer function as

$$K(s) = \left(\frac{1+\frac{bs}{d\omega_b}}{1+\frac{ds}{b\omega_h}}\right)^\alpha, \tag{8.24}$$

where $0 < \alpha < 1,\ s = \mathrm{j}\omega,\ b > 0,\ d > 0$, and

$$K(s) = \left(\frac{bs}{d\omega_b}\right)^\alpha \left(1+\frac{-ds^2+d}{ds^2+b\omega_h s}\right)^\alpha. \tag{8.25}$$

In the frequency range $\omega_b < \omega < \omega_h$, by using a Taylor series expansion, we obtain

$$K(s) = \left(\frac{bs}{d\omega_b}\right)^\alpha \left(1+\alpha p(s)+\frac{\alpha(\alpha-1)}{2}p^2(s)+\cdots\right) \tag{8.26}$$

with

$$p(s) = \frac{-ds^2+d}{ds^2+b\omega_h s}.$$

It is then found that

$$s^\alpha = \frac{(d\omega_b)^\alpha b^{-\alpha}}{\left[1+\alpha p(s)+\frac{\alpha(\alpha-1)}{2}p^2(s)+\cdots\right]} \left(\frac{1+\frac{bs}{d\omega_b}}{1+\frac{ds}{b\omega_h}}\right)^\alpha. \tag{8.27}$$

Truncating the Taylor series to 1 leads to

$$s^\alpha \approx \frac{(d\omega_b)^\alpha}{b^\alpha\big(1+\alpha p(s)\big)} \left(\frac{1+\frac{bs}{d\omega_b}}{1+\frac{ds}{b\omega_h}}\right)^\alpha. \tag{8.28}$$

Thus, the fractional-order differentiator is defined as

$$s^\alpha \approx \left(\frac{d\omega_b}{b}\right)^\alpha \left(\frac{ds^2+b\omega_h s}{d(1-\alpha)s^2+b\omega_h s+d\alpha}\right) \left(\frac{1+\frac{bs}{d\omega_b}}{1+\frac{ds}{b\omega_h}}\right)^\alpha. \tag{8.29}$$

Expression (8.29) is stable if and only if all the poles are on the left-hand side of the complex s-plane. It is easy to check that expression (8.29) has three poles:

- One of the poles is located at $-b\omega_h/d$, which is a negative real pole since $\omega_h > 0$, $b > 0, d > 0$;
- The two other poles are the roots of the equation

$$d(1-\alpha)s^2 + a\omega_{hs} + d\alpha = 0 \tag{8.30}$$

whose real parts are negative since $0 < \alpha < 1$.

Thus, all the poles of (8.29) are stable within the frequency range (ω_b, ω_h).

The irrational fractional-order part of expression (8.29) can be approximated by the continuous-time rational model

$$K(s) = \lim_{N\to\infty} K_N(s) = \lim_{N\to\infty} \prod_{k=-N}^{N} \frac{1+s/\omega_k'}{1+s/\omega_k}. \tag{8.31}$$

According to the recursive distribution of real zeros and poles, the zero and pole of rank k can be written as

$$\omega_k' = \left(\frac{d\omega_b}{b}\right)^{\frac{\alpha-2k}{2N+1}},\quad \omega_k = \left(\frac{b\omega_h}{d}\right)^{\frac{\alpha+2k}{2N+1}}. \tag{8.32}$$

Thus, the continuous rational transfer function model can be obtained [107] as

$$s^\alpha \approx \left(\frac{d\omega_h}{b}\right)^\alpha \left(\frac{ds^2+b\omega_h s}{d(1-\alpha)s^2+b\omega_h s+d\alpha}\right) \prod_{k=-N}^{N} \frac{s+\omega_k'}{s+\omega_k}. \tag{8.33}$$

Through confirmation by experimentation and theoretical analysis, the synthesis approximation can obtain the good effect when $b = 10$ and $d = 9$.

Through the approximation method, the fractional-order system may be approximated as the very high integer-order system. The high integer-order rational transfer function could be very tedious.

With the above algorithm, a MATLAB function `new_fod()` is written

```
function G=new_fod(r,N,wb,wh,b,d)
if nargin==4, b=10; d=9; end
mu=wh/wb; k=-N:N; w_kp=(mu).^((k+N+0.5-0.5*r)/(2*N+1))*wb;
w_k=(mu).^((k+N+0.5+0.5*r)/(2*N+1))*wb; K=(d*wh/b)^r;
G=zpk(-w_kp',-w_k',K)*tf([d,b*wh,0],[d*(1-r),b*wh,d*r]);
```

with the syntax G_f=`new_fod(`$\gamma, N, \omega_b, \omega_h, b, d$`)`.

Example 8.4. Consider a model

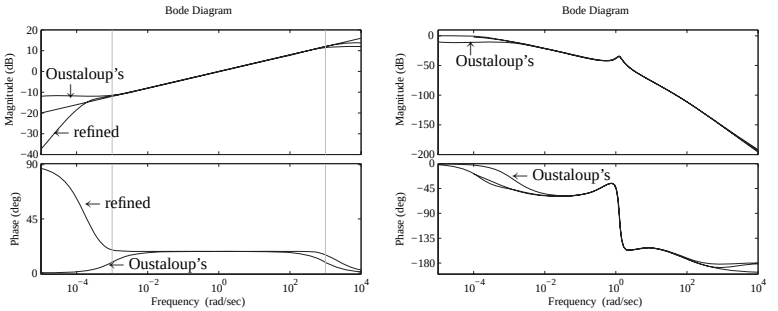
$$G(s) = \frac{s+1}{10s^{3.2} + 185s^{2.5} + 288s^{0.7} + 1}$$

which is a fractional-order model. The exact Bode diagram can be obtained with the `bode()` function. The approximations using the Oustaloup filter, and the refined Oustaloup filter, can be obtained as shown in Figure 8.3(a). The approximations to the $G(s)$ model are shown in Figure 8.3(b). It can be seen that the refined method provides a much better fit:

```
>> b=[1 1]; a=[10,185,288,1]; nb=[1 0]; na=[3.2,2.5,0.7,0];
   w=logspace(-4,4,200); G0=fotf(a,na,b,nb); H=bode(G0,w);
   s=zpk('s'); N=4; w1=1e-3; w2=1e3; b=10; d=9;
   g1=oustafod(0.2,N,w1,w2); g2=oustafod(0.5,N,w1,w2); a1=g1;
   g3=oustafod(0.7,N,w1,w2);
   G1=(s+1)/(10*s^3*g1+185*s^2*g2+288*g3+1);
   g1=new_fod(0.2,N,w1,w2,b,d); g2=new_fod(0.5,N,w1,w2,b,d);
   g3=new_fod(0.7,N,w1,w2,b,d); bode(g1,a1); figure
   G2=(s+1)/(10*s^3*g1+185*s^2*g2+288*g3+1); bode(H,G1,G2)
```

8.3.3 Simulink-Based Fractional-Order Nonlinear Differential Equation Solutions

From the previous discussions, it can be found that the refined Oustaloup recursive filter is an effective way to compute the fractional-order derivatives. It should be noted that the



(a) $s^{0.2}$ fittings (b) Bode diagram comparisons

Figure 8.3. *Bode diagram comparisons.*

orders of the numerator and the denominator in the refined Oustaloup filter are the same, which may cause algebraic loops in Simulink. To avoid the algebraic loops, the filter should be followed by a low-pass filter, with a crossover frequency ω_h. The constructed block is shown in Figure 8.4(a).

With the mask facilities provided in Simulink, the fractional-order differentiator block can be built, as shown in Figure 8.4(b). Double click the fractional-order differentiator block to display the dialog box in Figure 8.4(c), which allows the user to enter parameters into the refined Oustaloup filters:

```
wb=ww(1); wh=ww(2); G=new_fod(gam,n,wb,wh,10,9);
num=G.num{1}; den=G.den{1}; T=1/wh; str='Fractional\n';
if isnumeric(gam)
   if gam>0, str=[str, 'Der  s^' num2str(gam) ];
   else, str=[str, 'Int  s^{' num2str(gam) '}']; end
else, str=[str, 'Der  s^gam']; end
```

In practical simulation processes, the model established could be made up of stiff systems. Thus, ode15s or ode23tb algorithms should be selected to ensure high efficiency and accuracy. Examples will be given to demonstrate the solutions of FODEs.

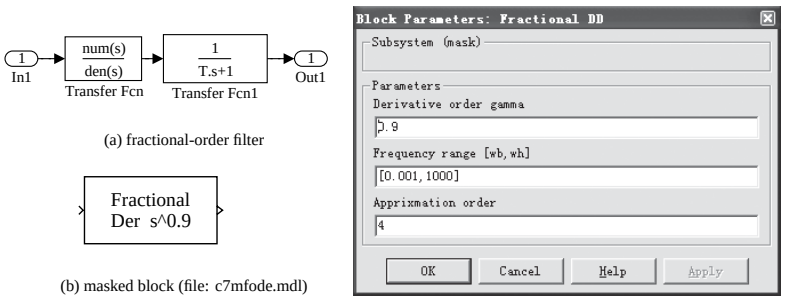
Example 8.5. Consider the nonlinear FODE described by

$$\frac{3\mathscr{D}^{0.9}y(t)}{3+0.2\mathscr{D}^{0.8}y(t)+0.9\mathscr{D}^{0.2}y(t)} + \left|2\mathscr{D}^{0.7}y(t)\right|^{1.5} + \frac{4}{3}y(t) = 5\sin(10t).$$

It can be seen that solving the original FODE is very complicated. From the original equation, the output signal $y(t)$ can explicitly be expressed as

$$y(t) = \frac{3}{4}\left[5\sin(10t) - \frac{3\mathscr{D}^{0.9}y(t)}{3+0.2\mathscr{D}^{0.8}y(t)+0.9\mathscr{D}^{0.2}y(t)} - \left|2\mathscr{D}^{0.7}y(t)\right|^{1.5}\right].$$

A Simulink model can then be established from the above equations, as shown in Figure 8.5(a). It can be seen from the model that each fractional-order differentiator can be modeled with the above designed block. In Figure 8.5(b), the simulation results are shown, with different parameters of the refined Oustaloup filter.

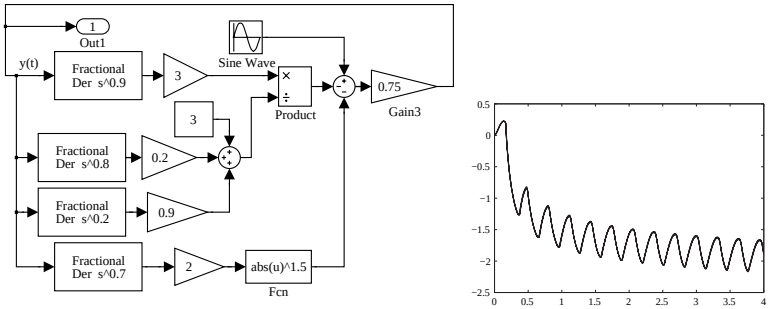


(a) fractional-order filter

(b) masked block (file: c7mfode.mdl)

(c) Dialog box of fractional-order differentiators

Figure 8.4. *Fractional-order differentiator block design.*



(a) Simulink model (file: c7mfod2.mdl) (b) simulation results

Figure 8.5. *Simulink modeling and results of a nonlinear FODE.*

It can be seen that the results are the same, and the only exception is the combination of $\omega_b = 0.001, \omega_h = 1000, N = 2$. However, even with this rough approximation, the error is still acceptable.

8.4 Model Reduction Techniques for Fractional-Order Systems

It has been shown that if the integer-order approximation is used to fit the fractional-order transfer function models with the use of the refined Oustaloup recursive filter, the order of the final system could be extremely high. Thus, a low-order approximation to the original problem can be found using the optimal model reduction method.

Recall the expected reduced-order model given by

$$G_{r/m,\tau}(s) = \frac{\beta_1 s^r + \cdots + \beta_r s + \beta_{r+1}}{s^m + \alpha_1 s^{m-1} + \cdots + \alpha_{m-1} s + \alpha_m} \mathrm{e}^{-\tau s}. \tag{8.34}$$

An objective function for minimizing the $\mathcal{H}_2$-norm of the reduction error signal $e(t)$ can be defined as

$$J = \min_{\theta} \left\| \widehat{G}(s) - G_{r/m,\tau}(s) \right\|_2, \tag{8.35}$$

where θ is the set of parameters to be optimized such that

$$\theta = [\beta_1, \ldots, \beta_r, \alpha_1, \ldots, \alpha_m, \tau]. \tag{8.36}$$

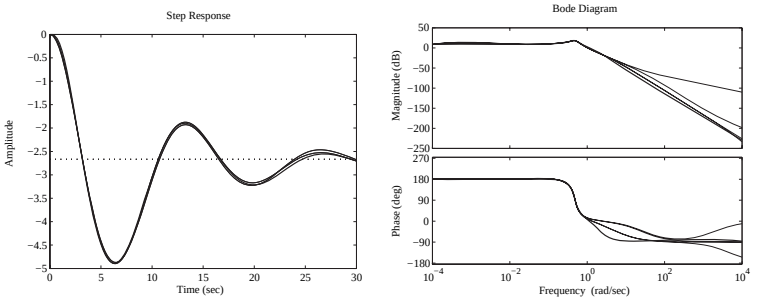
For an easy evaluation of the criterion J, the delayed term in the reduced-order model $G_{r/m,\tau}(s)$ can be further approximated by a rational function $\widehat{G}_{r/m}(s)$ using the Padé approximation technique [47]. Thus, the revised criterion can then be defined by

$$J = \min_{\theta} \left\| \widehat{G}(s) - \widehat{G}_{r/m}(s) \right\|_2 \tag{8.37}$$

and the $\mathcal{H}_2$-norm computation can be evaluated recursively using the algorithm in [108]. The function `opt_app()` discussed in Sec. 3.6 can still be used for fractional-order systems.

Table 8.1. *Comparisons of different order combinations.*

r	m	Reduced-order model	Error
2	3	$\dfrac{0.03147s^2 - 0.8141s - 0.07206}{s^3 + 0.3168s^2 + 0.2582s + 0.02703}$	0.2286
2	4	$\dfrac{-0.0119s^2 - 23.21s - 2.035}{s^4 + 28.78s^3 + 9.242s^2 + 7.365s + 0.7634}$	0.2308
2	5	$\dfrac{-4.932s^2 - 0.8602s - 0.00386}{s^5 + 5.741s^4 + 2.794s^3 + 1.596s^2 + 0.3134s + 0.001448}$	0.1342
2	6	$\dfrac{-2.327\times10^4 s^2 - 4059s - 18.21}{s^6 + 4719s^5 + 2.709\times10^4 s^4 + 1.318\times10^4 s^3 + 7534s^2 + 1479s + 6.831}$	0.1342



(a) step responses (b) Bode diagrams

Figure 8.6. *Comparisons of the reduced-order models.*

Example 8.6. Consider again the high-order fractional-order transfer function given in Example 8.3, where a 48th-order model was obtained, and with the refined Oustaloup filter, a 58th-order model can be obtained. Using optimal reduction techniques for different order combinations, the reduced-order models can be found as shown in Table 8.1. It can be seen that the $G_{2/5}(s)$ model is the best one. The step responses and Bode diagrams are compared in Figure 8.6. It can be seen that the approximation is satisfactory. It should be noted that in the code, the `opt_app()` function may be called several times since the original model should be used in these cases.

```
>> N=4; w1=1e-3; w2=1e3; s=tf('s'); g1=new_fod(0.501,N,w1,w2,9,10);
   g2=new_fod(0.42,N,w1,w2,9,10); g3=new_fod(0.798,N,w1,w2,9,10);
   g4=new_fod(0.31,N,w1,w2,9,10); g5=new_fod(0.63,N,w1,w2,9,10);
   G=(-2*g5-4)/(2*s^3*g1+3.8*s^2*g2+2.6*s*g3+2.5*s*g4+1.5);
   Gr1=opt_app(G,2,3,0);norm(G-Gr1),Gr2=opt_app(G,2,4,0);norm(G-Gr2)
   Gr3=opt_app(G,2,5,0); Gr3=opt_app(G,2,5,0,Gr3); norm(G-Gr3)
   Gr4=opt_app(G,2,6,0); Gr4=opt_app(G,2,6,0,Gr4);
   Gr4=opt_app(G,2,6,0,Gr4); norm(G-Gr4)
   step(G,Gr1,Gr2,Gr3,Gr4,30); figure; bode(G,Gr1,Gr2,Gr3,Gr4)
```

8.5 Controller Design Studies for Fractional-Order Systems

From the analysis given previously, it can be seen that the behaviors of fractional-order controllers may be different from their integer-order counterparts. For instance, if the widely used PID controller is considered, its fractional-order version $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$ controller can be expressed by [99]

$$G_c(s) = K_p + \frac{K_i}{s^{\lambda}} + K_d s^{\mu}. \tag{8.38}$$

In the illustration in Figure 8.7, the fractional-order PID controller is explained, with the horizontal axis as the order of the integrator and the vertical axis the order of the differentiator. It can be seen that the ordinary PI (proportional plus integral), PD, and PID controllers are special cases of the fractional-order PID controller since the values of λ and μ can be selected freely, which adds two more degree of freedom to the controller design. It has been shown that the control behavior of the best fractional-order PID controller is quite superior to the best conventional PID controller in some applications [109].

If the loop shaping technique is considered, it can be seen that the Bode magnitude diagrams is no longer restricted to $20k$ dB/decade slopes. Thus the shape of the loop transfer function can be set freely for better performance and robustness. In this section, several examples will be given to show the design of an integer-order controller and fractional-order controller for fractional-order plants.

Example 8.7. For a plant model

$$G(s) = \frac{1}{s^{2.6} + 2.2s^{1.5} + 2.9s^{1.3} + 3.32s^{0.9} + 1},$$

if an integer-order PID controller is expected, it is quite natural to first find an FOPDT approximate model,

$$G_p(s) = k\frac{\mathrm{e}^{-Ls}}{Ts+1}$$

and then design a PID controller for the FOPDT model. The designed controller can then be used in closed-loop control of the fractional-order plant $G(s)$. For instance, the Wang–Juang–Chan algorithm [69] in Sec. 6.3.4 can be used to design a PID controller for an

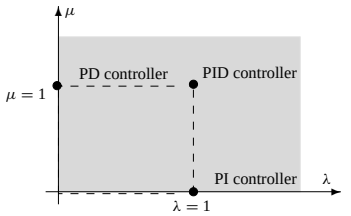


Figure 8.7. *Fractional-order PID controller.*

FOPDT model with an optimum ITAE criterion:

$$K_p = \frac{(0.7303 + 0.5307T/L)(T + 0.5L)}{K(T+L)}, \quad T_i = T + 0.5L, \quad T_d = \frac{0.5LT}{T+0.5L}. \tag{8.39}$$

The following statements can be used to extract the FOPDT model from the approximated high-order plant model:

```
>> N=4; w1=1e-3; w2=1e3; s=tf('s');
   g1=new_fod(0.6,N,w1,w2,9,10); g2=new_fod(0.5,N,w1,w2,9,10);
   g3=new_fod(0.3,N,w1,w2,9,10); g4=new_fod(0.9,N,w1,w2,9,10);
   G=1/(s^2*g1+2.2*s*g2+2.9*s*g3+3.32*g4+1); Gr=opt_app(G,0,1,1)
```

The reduced plant model is then

$$G_r(s) = \frac{0.1702}{s+0.1702}\mathrm{e}^{-0.612s}.$$

The PID controller can be designed such that

```
>> K=0.1702/0.1702; T=1/0.1702; L=0.612;
   Ti=T+0.5*L; Kp=(0.7303+0.5307*T/L)*Ti/(K*(T+L));
   Td=(0.5*L*T)/(T+0.5*L); Gc=Kp*(1+1/Ti/s+Td*s),
```

The integer-order PID controller is designed as

$$G_c(s) = 4.7960\left(1 + \frac{1}{5.6315s} + 0.3076s\right) = \frac{1.614s^2 + 5.55s + 0.8979}{s}.$$

Under such a controller, the closed-loop step response is obtained as shown in Figure 8.8. It can be seen that the integer-order PID controller can still be used in the fractional-order plant control. The control results are satisfactory. It is also seen that the high-order approximation to the closed-loop system is very accurate:

```
>> Gcf=fotf(1,1,[1.614 5.55 0.8979],[2,1,0]); H=fotf(1,0,1,0);
   a=[1 2.2 2.9 3.32 1]; an=[2.6,1.5,1.3 0.9 0]; G0=fotf(a,an,1,0);
   GG=feedback(Gcf*G0,H); t=0:0.005:15;
   step(feedback(G*Gc,1),t); hold on, step(feedback(G0*Gcf,H),t);
```

Example 8.8. Consider a fractional-order plant model

$$G(s) = \frac{10}{s^{\alpha} + 2.2},$$

where the order α is an undetermined parameter, within the interval $\alpha \in (1.2, 1.6)$. The nominal value of the variable is $\alpha_0 = 1.4$. In order to get a low-order robust controller, a relatively smaller value of N can be selected, for instance, $N = 2$. The following statements can be used to approximate the original model by integer-order approximation such that

```
>> N=2; w1=1e-3; w2=1e3; s=tf('s');
   g1=oustafod(0.4,N,w1,w2);  G=1/(s*g1+2.2);
```

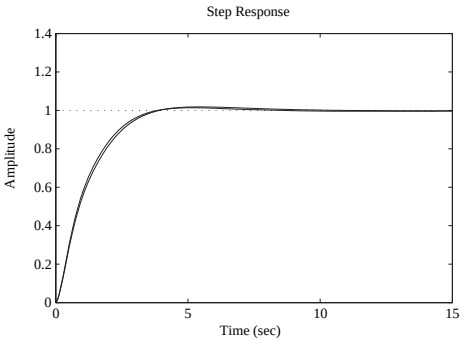


Figure 8.8. *Integer-order PID control of fractional-order plant.*

Select weighting functions $w_1(s) = 100/(s+1)$ and $w_3(s) = 10/(0.01s+100)$. The optimal $\mathcal{H}_\infty$ controller can be designed such that

```
>> W1=100/(s+1); W3=100/(0.01*s+100); Gc=mixsyn(G,W1,[],W3);
```

The controller can be designed as

$$G_c(s) = \frac{\begin{array}{c}71870205(s+1000)(s+144.3)(s+8.265)(s+0.1116)\\(s+0.006921)(s^2+1.73s+2.388)\end{array}}{\begin{array}{c}(s+9499)(s+9975)(s+346.4)(s+27.46)\\(s+1.738)(s+1)(s+0.1096)(s+0.006918)\end{array}}.$$

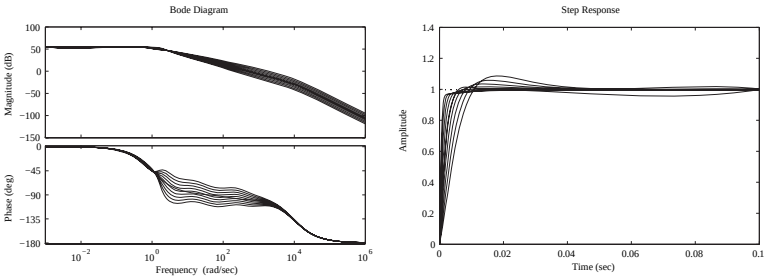
Under such a controller, the open-loop Bode diagrams and the closed-loop step response are obtained as shown in Figures 8.9(a) and (b), respectively:

```
>> f1=figure; bode(G*Gc); hold on
   f2=figure; step(feedback(G*Gc,1),0.1); hold on
   for a=[0.2:0.05:0.6]
      g1=oustafod(a,4,w1,w2); G1=1/(s*g1+2.2);
      figure(f1); bode(G1*Gc);
      figure(f2); step(feedback(G1*Gc,1),0.1)
   end
```

Example 8.9. Consider again the fractional-order plant model in Example 8.7. The integer-order approximation can be obtained such that

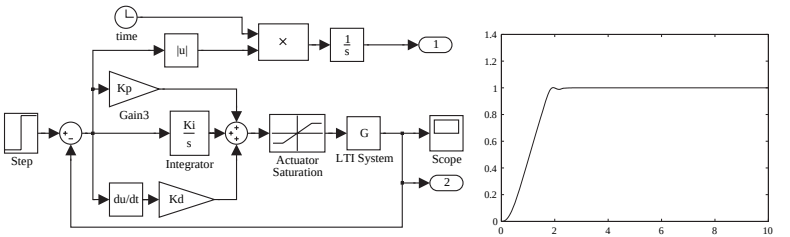
```
>> N=4; w1=1e-3; w2=1000; s=tf('s');
   g1=oustafod(0.6,N,w1,w2); g2=oustafod(0.5,N,w1,w2);
   g3=oustafod(0.3,N,w1,w2); g4=oustafod(0.9,N,w1,w2);
   G=1/(s^2*g1+2.2*s*g2+2.9*s*g3+3.32*g4+1);
```

Using the integer-order model, the Simulink model for optimal controller design with an integer-order PID controller is established as shown in Figure 8.10(a). A saturation actuator with limits ± 5 is also included in the Simulink model.



(a) Bode diagrams (b) closed-loop step responses

Figure 8.9. *Time and frequency domain analysis under robust controller.*



(a) Simulink model (file: c8mfpid2.mdl) (b) closed-loop response

Figure 8.10. *Optimal PID controller design for fractional-order plant.*

It can be found by using the Optimal Controller Designer (OCD) program that the parameters of the PID controller are $K_p = 14768.1007$, $K_i = 1.35636077$, $K_d = 2306.39271$. Under such a controller, the optimum step response of the closed-loop system can be obtained as shown in Figure 8.10(b). It can be seen that the controller obtained with the OCD is much better than the one obtained in Example 8.7. Also the control action is restricted within the specific range.

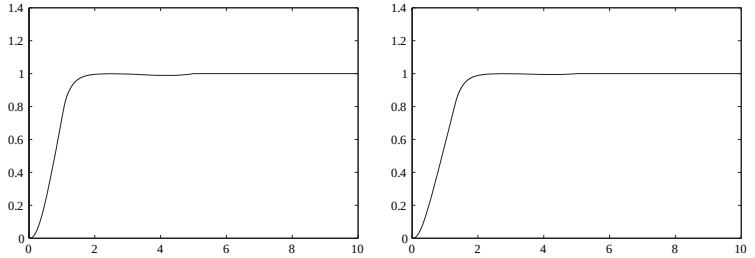
Due to the robustness of the PID controllers, the errors in the controller parameters may not cause any problem in the control results. For instance, if we had the erroneous parameters $K_p = 10000$, $K_i = 1$, $K_d = 2500$, where the errors reach 35%, the control results would be as shown in Figure 8.11(a). It can be seen that the system responses are almost the same with the optimal PID controller:

```
>> Kp=10000; Ki=1; Kd=2500;
   [t,x,y]=sim('c8mfpid2',[0,10]); plot(t,y(:,2))
```

Assume that plant model is changed to

$$G(s) = \frac{2}{s^{2.6}+5s^{1.5}+4s^{1.3}+5.32s^{0.9}+1},$$

where the parameters are all perturbed. If the erroneous PID controller is still used, the control results are as shown in Figure 8.11(b). It can be seen that, although the plant models



(a) controller parameters change (b) plant and controller change

Figure 8.11. *The robustness of the PID controller.*

change significantly, the PID controller can still behave perfectly. This demonstrates the robustness of the PID controller in fractional-order plant models:

```
>> G=2/(s^2*g1+5*s*g2+4*s*g3+5.32*g4+1);
   [t,x,y]=sim('c8mfpid2',[0,10]); plot(t,y(:,2))
```

Problems

1. Assume that a fractional-order linear differential equation is given by

$$0.8\mathscr{D}_t^{2.2}y(t)+0.5\mathscr{D}_t^{0.9}y(t)+y(t)=1,$$

with initial values $y(0)=y'(0)=y''(0)=0$. Solve numerically the FODE. If the order of 2.2 is approximated by 2, and 0.9 is approximated by 1, the original fractional-order differential equation can be approximated by an integer-order system. Compare the accuracy of the approximated integer-order systems.

2. For a fractional-order model given by

$$(a).\quad G(s)=\frac{5}{s^{2.3}+1.3s^{0.9}+1.25}$$

and

$$(b).\quad G(s)=\frac{5s^{0.6}+2}{s^{3.3}+3.1s^{2.6}+2.89s^{1.9}+2.5s^{1.4}+1.2},$$

approximate the fractional-order models with low-order integer-order models, and compare the accuracy of the frequency and time domain fittings. Discuss what order combination is most suitable for the original model.

3. Suppose that the plant model is

$$G(s)=\frac{1}{s^{2.6}+2.2s^{1.5}+2.9s^{1.3}+3.32s^{0.9}+1},$$

and an integer-order PID controller is

$$G_c(s)=\frac{1.614s^2+5.55s+0.8979}{s}.$$

Find the closed-loop fractional-order model.

4. Write a function to find the solutions to the FODE using the algorithm in (8.17)–(8.21), and compare the results with the Grünwald–Letnikov definition approach and the block diagram algorithm.

5. Consider the linear FODE given by

$$\mathscr{D}x(t)+\left(\frac{9}{1+2\lambda}\right)^{\alpha}\mathscr{D}^{\alpha}x(t)+x(t)=1,$$

where $\lambda=0.5$, $\alpha=0.25$ and $x(0)=0$. Solve the equation numerically.

6. Find a good approximation to $s^{0.7}$ with the revised Oustaloup filter and see which N can best fit the fractional-order differentiator.

7. Solve the following nonlinear FODE with the block diagram algorithm with $x(0)=0$:

$$\mathscr{D}^2x(t)+\mathscr{D}^{1.455}x(t)+\left[\mathscr{D}^{0.555}x(t)\right]^2+x^3(t)=\sin t.$$

8. For the plant model

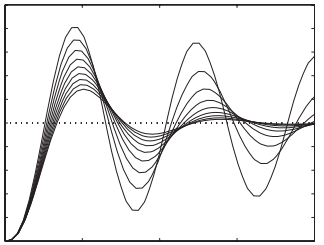
$$G(s)=\frac{5s^{0.6}+2}{s^{3.3}+3.1s^{2.6}+2.89s^{1.9}+2.5s^{1.4}+1.2},$$

design an integer-order PID controller and observe the control results.

9. For the fractional-order model

$$G=\frac{b}{as^{0.7}+1},$$

design an $\mathscr{H}_\infty$ controller which can tolerate the parameter changes in the fractional-order model, for instance, $a\in(0.2,5)$ and $b\in(0.2,1.5)$.



Appendix

CtrlLAB: A Feedback Control System Analysis and Design Tool

A.1 Introduction

A.1.1 What Is CtrlLAB?

CtrlLAB, a MATLAB-based toolkit with an integrated graphical user interface (GUI), was designed by the authors for solving the modeling, analysis, and design problems in SISO (single input–single output) feedback control systems. It is developed from the old Control Kit by the authors [110]. CtrlLAB has become a flexible and powerful tool for both teaching and engineering design and requires minimum user effort. It can be used as a companion to this book.

CtrlLAB, written and tested under MATLAB v4.2, was first made public on the MathWorks anonymous ftp site as a user-contributed MATLAB program. Since then, much useful feedback has been received. Over the years, CtrlLAB has been greatly improved. It has already been used as a CAI (computer aided instruction) tool in control courses at many universities worldwide. The latest version of CtrlLAB can also run under other versions of MATLAB, including MATLAB R2007b. It is still freely downloadable from MATLAB Central at

```
http://www.mathworks.com/matlabcentral/index.shtml
```

Currently, CtrlLAB is the most downloaded tool under the Controls and Systems Modeling file exchange category at MATLAB Central.

The main facilities provided by CtrlLAB are

- model entry, including Simulink model entry;
- model display;
- state space realizations;
- model reduction using various algorithms;
- system analysis in frequency and time domains;
- graphical display with figure editing and manipulation;
- a GUI matrix processor and editor;

- many controller design modules such as the model-based approaches (lead-lag, LQ (linear quadratic) optimal, pole-placement, etc.); PID (proportional integral derivative) parameter setting and PID tuning schemes; and robust controller design approaches (such as LQG (linear quadratic Gaussian), LQG/LTR (loop transfer recovery), $\mathcal{H}_2$, $\mathcal{H}_\infty$, etc.).

A.1.2 Installation and Requirements

With the downloaded `ctrllab.zip` file, unzip it to a directory using `WinZip` or `pkunzip` software. Before running CtrlLAB, the directory of CtrlLAB should be added to the MATLAB path. This can be set with the File | Set Path menu item in the MATLAB command window.

CtrlLAB is written for the PC Windows platform; however, it should also be able to run on other platforms. Although CtrlLAB has not been fully tested on other platforms, with a MATLAB version newer than 4.2c, the cross platform compatibility will be much better than what was experienced under MATLAB version 4.2c. We believe that CtrlLAB can run on any current version of other platforms with little modification.

A.1.3 Execution of CtrlLAB

To run CtrlLAB, simply type `ctrllab` under the MATLAB prompt, and a GUI with menus will pop up, as shown in Figure A.1. The user must first enter or to define the models, which include the plant, the controller, and the feedback element. The default models for the latter two are all unity. The possible time delay may also be specified. With the specified models, the analysis and design tasks can be performed.

Menus and dialog boxes are provided to invoke relevant functions to fulfill the user's own analysis and design tasks. Note that all the functions provided in CtrlLAB can be accessed through the efficient and user friendly GUI. There is no need to call these functions

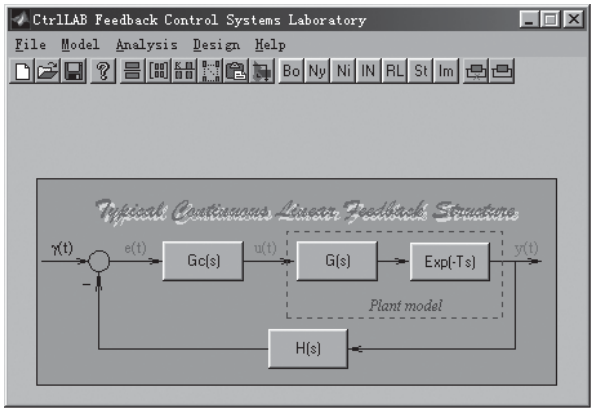


Figure A.1. *The GUI of CtrlLAB.*

manually. CtrlLAB is designed for linear feedback control system analysis and design using only mouse clicks and numeric key strokes. Great effort has been made in CtrlLAB to minimize the user involvement in the analysis and design of feedback control systems.

A.2 Model Entry and Model Conversion

A.2.1 Transfer Function Entry

To quickly enter a default model, the user can click one of the model icons in the block diagram shown in Figure A.1, and CtrlLAB will check whether the model exists in the work space. If it does not exist, a dialog box, shown in Figure A.2, will appear by default, which allows the user to enter the system model by specifying the numerator and denominator, respectively, in the appropriate edit boxes.

The transfer function model can be entered in two ways. The first is by entering the standard MATLAB vectors in descending order of the Laplace complex variable s. The second is by representing the polynomials in a "natural way." These two methods are demonstrated in Table A.1. It can be seen that for the factorized polynomials, the s polynomial representation is much more "natural" and simpler than a pure MATLAB expression.

A.2.2 Entering Other Model Representations

The state space model, or zero-pole-gain model, can also be entered if the corresponding item from the list box shown in Figure A.2 is selected.

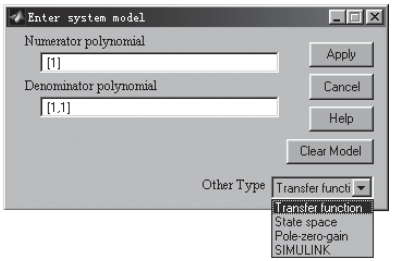


Figure A.2. *Dialog box for transfer function model entry.*

Table A.1. *Examples of polynomial representations.*

Mathematical	MATLAB commands	s polynomial
s^2+5s+4	`[1,5,4]`	`s2+5s+4`
$s^2(s+5)(s^2+7)$	`[conv([1,5],[1,0,7]),0,0]`	`s2(s+5)(s2+7)2`
$1.5s^3(s^3+7s^2+6s+2)^{12}$	too complicated	`1.5s3(s3+7s2+6s+2)12`

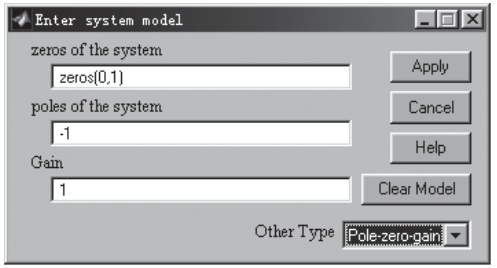


Figure A.3. *Dialog box for zero-pole-gain model entry.*

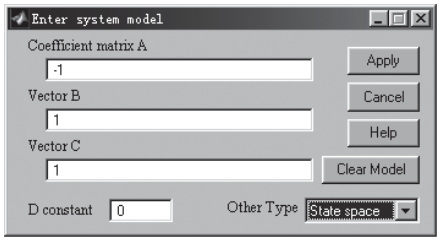
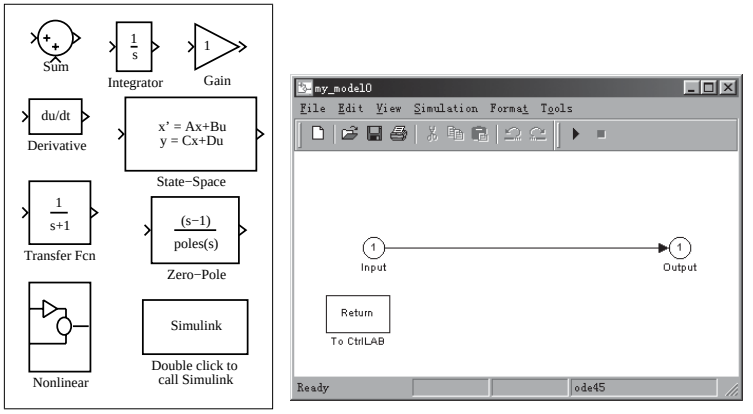


Figure A.4. *Dialog box for state space model entry.*

In Figure A.2, if the menu item pole-zero (for zero-pole-gain model) is selected, the dialog box shown in Figure A.3 will appear, where the zero-pole-gain model parameters can be entered in the corresponding edit boxes. Then, press the OK button to confirm. Internally, a transfer function object will be generated automatically from the user-specified zero-pole-gain model. For the state space item, the dialog box shown in Figure A.4 will appear, where the ($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, D) matrices of the system can be entered in the corresponding edit boxes. Then, a transfer function object of the block can be generated automatically from the given state space model.

A.2.3 A More Complicated Model Entry

If the system model under study has a more complicated structure, such as containing complex block diagrams or nonlinearities, the Simulink program should be used to construct the system model. In this case, the user can select the Simulink item from the dialog box shown in Figure A.2. A model name (an internal name) will be requested and then the Simulink editing environment will appear, as shown in Figures A.5(a) and (b), where Figure A.5(a) is the model library from which all the Simulink library models can be accessed. Figure A.5(b) is a blank Simulink model editing window in which the user can draw the system model between the input and output ports of the system. Once the model entry process is completed in the Simulink edit window, as shown in Figure A.5(b), double click Return to CtrlLAB to return the user system model to CtrlLAB. If the user model in Simulink is nonlinear, the linearized transfer function model of the user system will be created and saved, together with the original Simulink model, for CtrlLAB use. A simple nonlinear model entry example in



(a) model library (b) model entering window

Figure A.5. *Simulink model entering in CtrlLAB.*

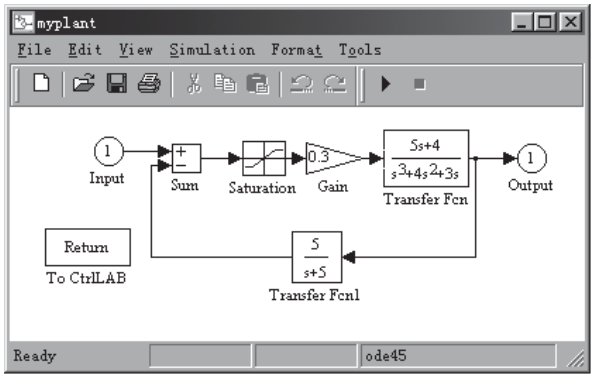


Figure A.6. *Complicated model entry in CtrlLAB via Simulink.*

CtrlLAB is shown in Figure A.6 which uses Simulink to describe the nonlinear part. Note the Return to CtrlLAB button in Figure A.6 for returning a linearized transfer function object for use with CtrlLAB.

A.3 Model Transformation and Reduction

A.3.1 Model Display

To display the model of a block in Figure A.1, select Model | Model Select in the menu shown in Figure A.7, or simply click the relevant block button in the main interface shown in Figure A.1.

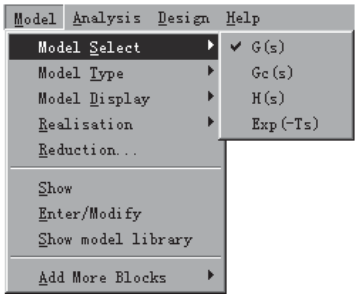


Figure A.7. *Model selecting menu.*

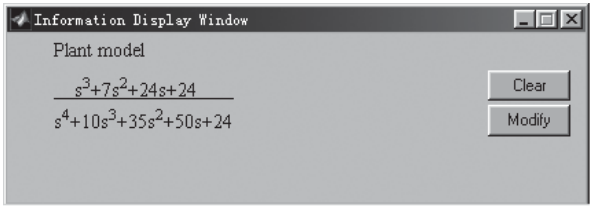


Figure A.8. *Transfer function display.*

Figure A.9. *Display format selection.*

As an example, consider the transfer function of the plant model given by

$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24}.$$

To display the transfer function model of the plant, simply press the G(s) button in the main interface shown in Figure A.1. The transfer function model will then be displayed in the Information Display Window as shown in Figure A.8. The displayed model can also be modified in the display window by pressing the Modify button. The dialog box shown in Figure A.2 will be displayed again for model parameter changes.

The block model can be displayed in various formats. This can be done by selecting the Model | Model Display menu, shown in Figure A.9, with the transfer function format as the default. Through the Model | Model Display | Factorized TF menu item, the transfer function in the factorized format will be displayed as shown in Figure A.10.

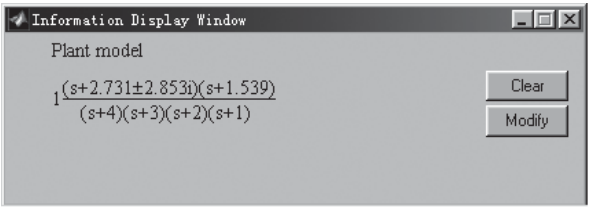


Figure A.10. *Factorized transfer function display format.*

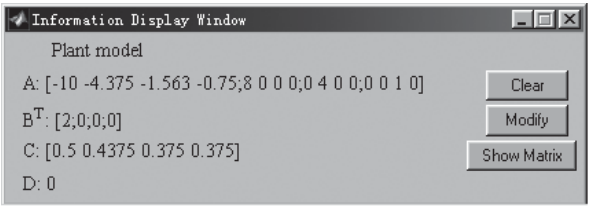


Figure A.11. *State space model display format.*

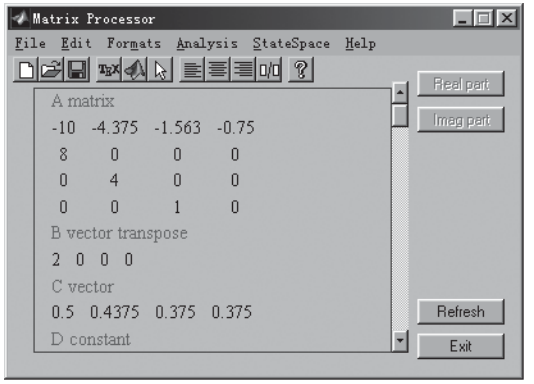


Figure A.12. *Display via the Matrix Processor.*

Moreover, the state space model can be displayed by the Model | Model Display | state space menu item as demonstrated in Figure A.11. When the Show button is clicked, the Matrix Processor is activated; the typical window is shown in Figure A.12. The zero-pole-gain format of the system is displayed by the Model | Model Display | Pole-Zero menu item which is shown in Figure A.13.

If the nonlinear system model is involved, only the linearized model will be displayed as in Figure A.14. To display the original Simulink model, simply press the to CtrlLAB button.

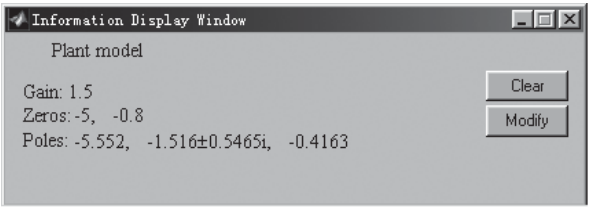


Figure A.13. *Zero-pole-gain display format.*

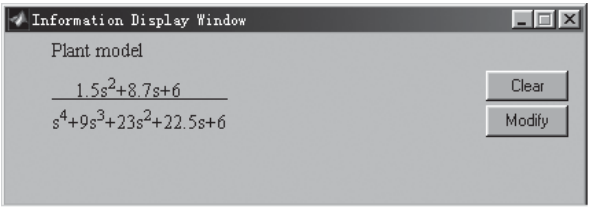


Figure A.14. *Linearized model display.*

A.3.2 State Space Realizations

Different state space realizations can be performed for a given transfer function plant model. This can be done by the Model | Realisation menu items shown in Figure A.15, and an example of the Jordanian canonical form of the system is obtained, as shown in Figure A.16, via the Matrix Processor interface.

A.3.3 Model Reduction

Reduced-order models of the system can also be obtained by the Model | Reduction menu item. The model reduction dialog box will appear as in Figure A.17, where various model reduction approaches are implemented such as the continued-fraction approach, the Padé method, the Routh method, the dominant mode method, the balanced realization method, the optimal reduction method, the FF-Padé method, the modal method, and the optimal Hankel approximation method.

For example, if the Padé approximation method is chosen from the list box of model reduction methods, the expected order of the reduced model can be specified as in Figure A.17. The reduced-order model is then obtained as shown in Figure A.18.

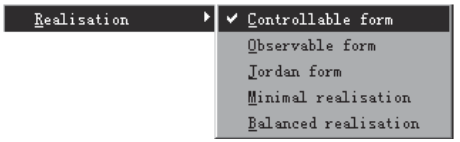


Figure A.15. *State space realization menu.*

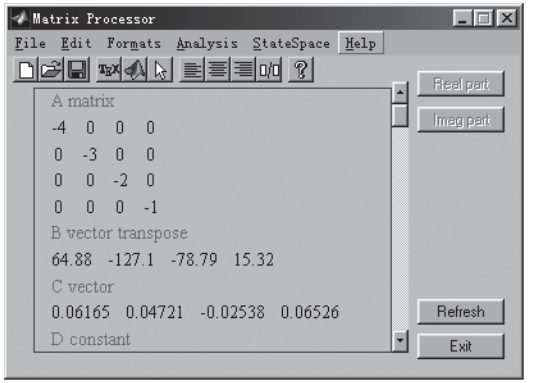


Figure A.16. *Jordan realization.*

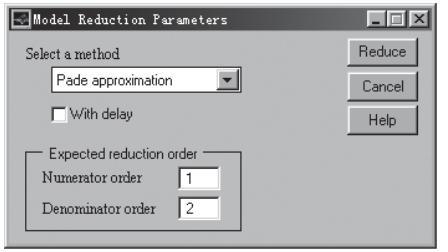


Figure A.17. *Model reduction dialog box.*

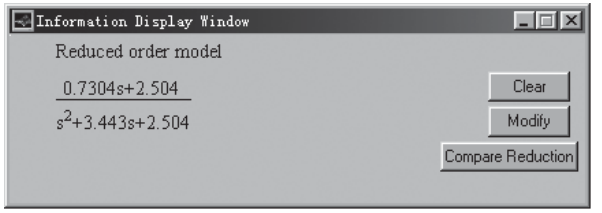
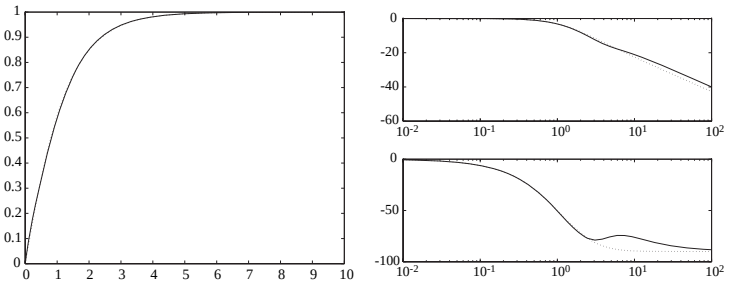


Figure A.18. *Model reduction result via the Padé approximation method.*

To compare the reduced-order model with the original model, click on Compare responses in the model display window. A new dialog box pops up for choosing a comparison plot from a list of responses which include the Bode diagrams, Nyquist plots, Nichols charts, as well as the step and impulse responses between the original model and the reduced-order model. For instance, the step response comparison, and the Bode diagram comparison, of the original system and the reduced model via the Padé approximation method are shown in Figures A.19(a) and (b), respectively, where the solid line represents for the original model and the dotted line the reduced-order model. It can be seen that the responses of the two



(a) step response comparison (b) Bode diagram comparison

Figure A.19. *Comparisons of the reduced order and the original models.*

models are quite close, especially in the step response comparison, where the two curves are almost indistinguishable.

A.4 Feedback Control System Analysis

Various linear system analysis tasks covered in this book can be performed by the direct use of CtrlLAB. After performing the model entry from Sec. A.2, select Analysis from the main menu shown in Figure A.1. The system analysis menu will appear as shown in Figure A.20. In this menu, plots for time domain, frequency domain, and root locus analysis can be generated by just using mouse clicks. In what follows, some detailed instructions are given in the subsections to follow.

A.4.1 Frequency Domain Analysis

The Bode diagram of the system can be obtained by the Analysis | Frequency Domain Analysis | Bode Diagram menu item. The result is shown as in Figure A.21(a).

Via the Options | Show asymptotes sub-menu in the Bode diagram window, the Bode plot asymptotes are drawn together with the exact Bode diagram, as demonstrated in Figure A.21(b).

The properties of the graphs can be modified by the Options | Plot preference sub-menu in the Bode diagram window, and a dialog box is then provided as shown in

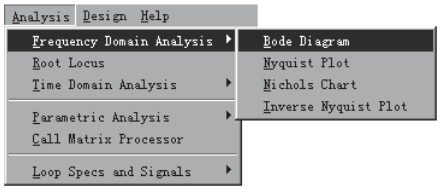
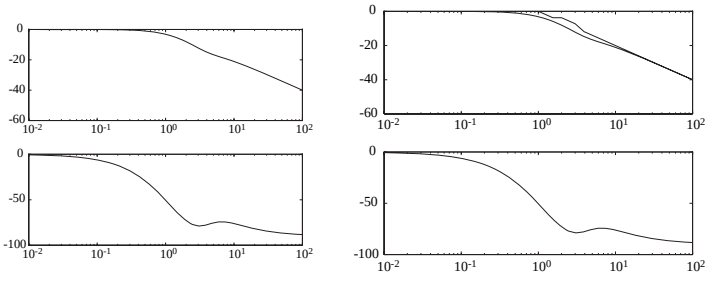


Figure A.20. *System analysis menu in CtrlLAB.*



(a) Bode diagram (b) with asymptotes

Figure A.21. *Bode diagram of a given linear system.*

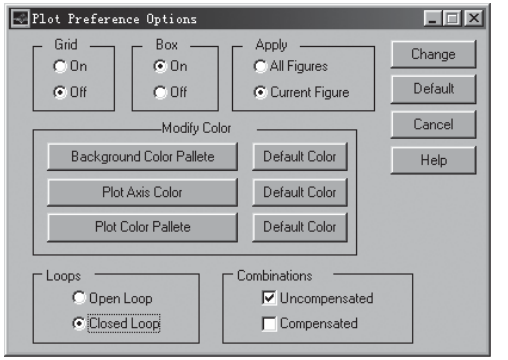


Figure A.22. *Graph properties setting dialog box.*

Figure A.22, where some of the details on the graph can be modified such as the boxes, grid, colors, etc. Moreover, the open-loop and closed-loop properties of the plots can also be changed. If a controller model is available, the Combinations group can be used to choose the Compensated as well as the Uncompensated frequency response. For instance, if the user checks the Closed Loop box, the closed-loop Bode diagram can then be obtained as shown in Figure A.23.

The Nyquist and Nichols charts can be obtained via the Analysis | Nyquist Plot and Analysis | Nichols Chart menu items. Results shown in Figures A.24(a) and (b), respectively.

The root locus plot can be obtained by using Analysis | Root Locus. For some particular systems, the directly obtained root locus of the system may not be very informative due to the poor quality of the automatically chosen plot ranges. In this case, the user can change the axis of the plot via the Options | Zoom | User Define menu item on the root locus window. A dialog box then appears as shown in Figure A.25(a). The ranges of the x and y axes can be changed until a good display result is obtained. For instance, with the properly chosen axes, the more informative root locus of the system can then be redrawn, as shown in Figure A.25(b).

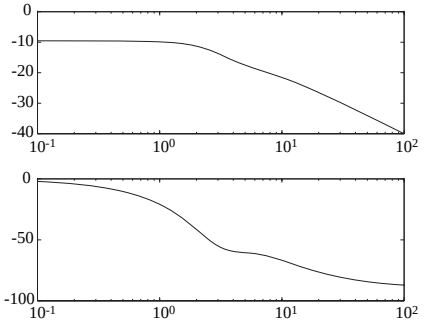
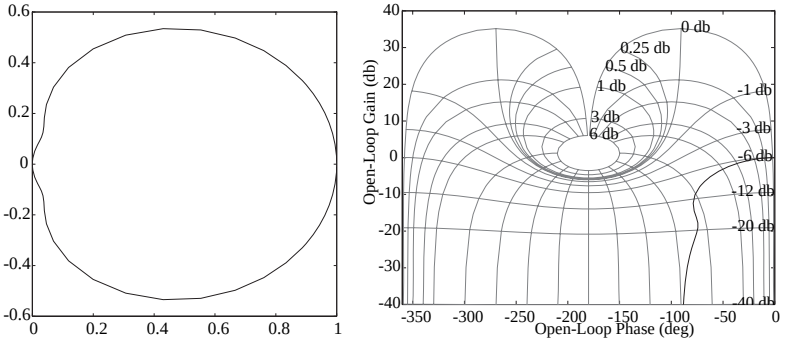
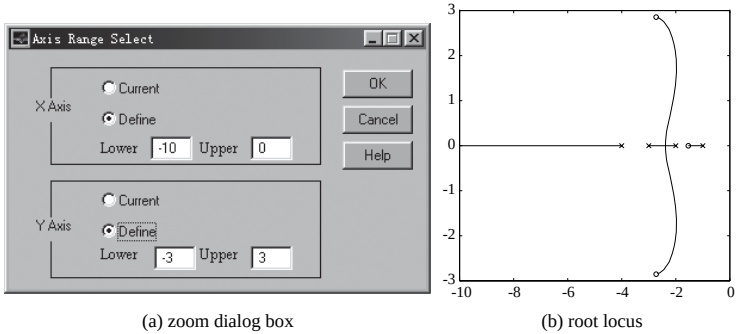


Figure A.23. *The modified graph.*



(a) Nyquist plots (b) Nichols chart

Figure A.24. *Frequency responses.*

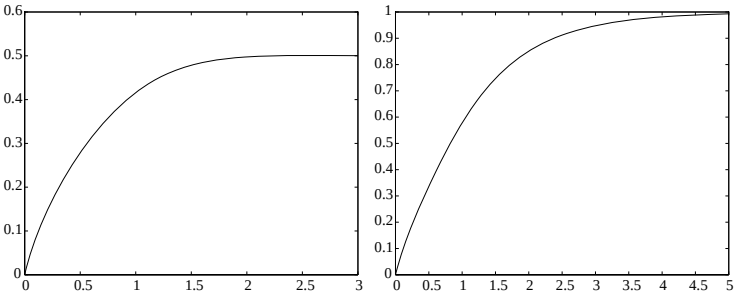


(a) zoom dialog box (b) root locus

Figure A.25. *Root locus analysis.*

A.4.2 Time Domain Analysis

The step and impulse responses of the system can be obtained directly from the menu Analysis | Step response, and Analysis | Impulse response, respectively. For instance, the



(a) closed-loop system (b) open-loop system

Figure A.26. *Step response analysis.*

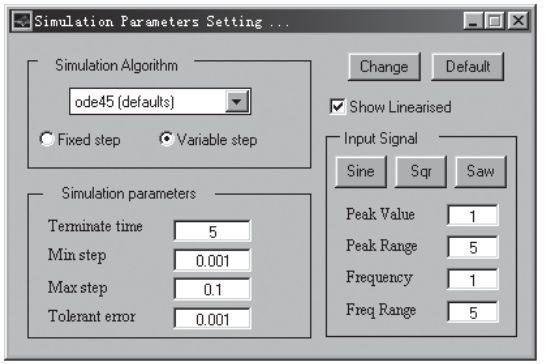


Figure A.27. *Simulation parameter setting dialog box.*

step response of the system can be obtained as shown in Figure A.26(a). This step response shown in Figure A.26(a) is the closed-loop step response. One can obtain the open-loop step response of the system by selecting the relevant submenu item in the Analysis menu and the open-loop step response of the system can then be redrawn in the step response window as shown in Figure A.26(b).

For nonlinear systems, one can also specify the type of input signals, via the Options | Simulation parameters menu item in the relevant graphics window. A dialog box will appear as shown in Figure A.27 which prompts the user to specify the input signals as well as the simulation parameters. For instance, when studying the system with the Simulink model, to display the step response of the linearized system and that of the original system,

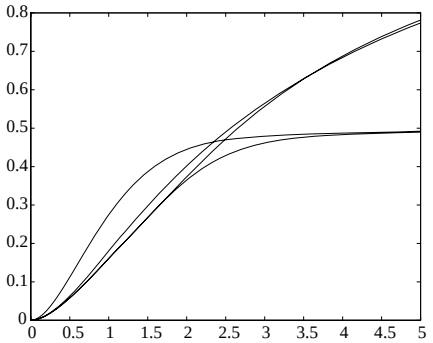
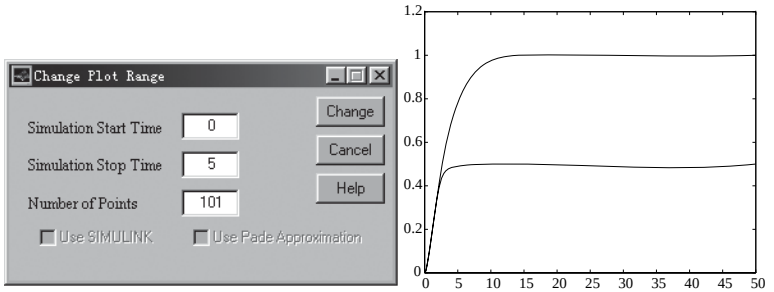


Figure A.28. *Step responses of a nonlinear system with linearization.*



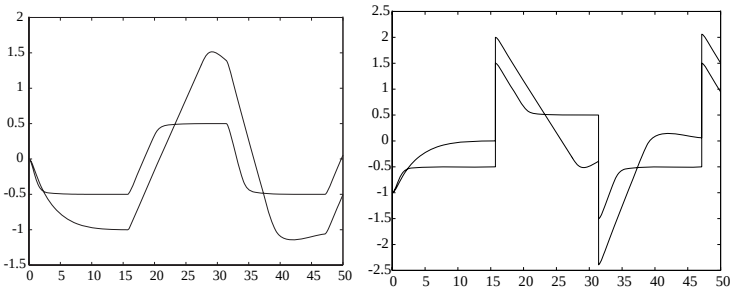
(a) plot range setting (b) with a new time range

Figure A.29. *Time range modifications.*

check the Show Linearised box. The time response of the system can then be displayed as shown in Figure A.28.

The plot range can also be set by the Options | Plot range menu item in the graphics window. A dialog box, shown in Figure A.29(a), prompts the user to select a new plot range. For instance, the user can set a new terminating time at 50, and the new system responses are then obtained as shown in Figure A.29(b).

Other signal types apart from the step and impulse signals can also be applied. For instance, the user can select square wave, saw tooth, wave and sine wave by using the dialog box shown in Figure A.27. Other parameters such as the frequency of the signal can also be changed. The time response to a square wave input is shown in Figure A.30(a). To display other signals such as the error signal $e(t)$, select the Options | Other signals menu item in the graphics window and click the error signal $e(t)$ in the block diagram of the feedback system. The error signal for a step input can then be obtained as shown in Figure A.30(b).



(a) square wave input response (b) error signal

Figure A.30. *Time response of other signals.*

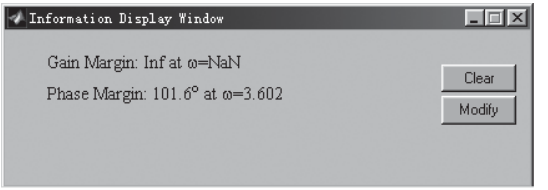


Figure A.31. *Gain and phase margins.*

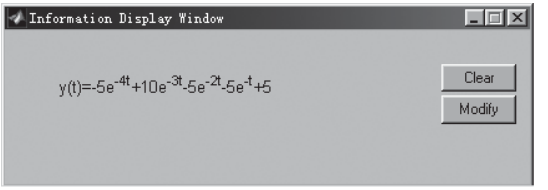


Figure A.32. *Analytical closed-loop step response.*

A.4.3 System Properties Analysis

The stability property, gain and phase margins, and the analytical solutions to step and impulse signals can also be obtained through the menu system. For instance, for the nonlinear system model, the gain and phase margins to the linearized model can be obtained as shown in Figure A.31, and the analytical solutions to the step response of the system can then be shown as in Figure A.32.

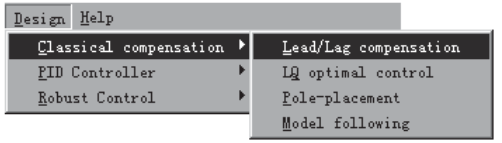


Figure A.33. *System design menu.*

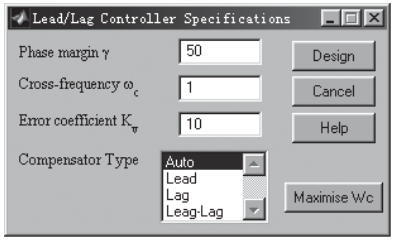


Figure A.34. *Lead-Lag compensator dialog box.*

A.5 Controller Design Examples

A.5.1 Model-Based Controller Designs

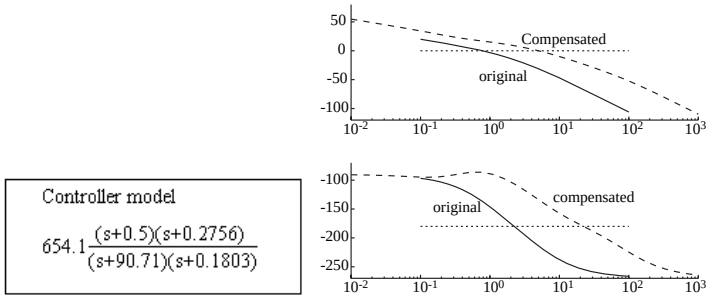
We shall use the phase lead-lag controller design problem as an example to illustrate the controller design for a given plant model via CtrlLAB. The model-based controller design menu is shown in Figure A.33, and it can be seen that several model-based design algorithms can be selected within the menu, as discussed in Chapter 5. For instance, with a typical lead-lag controller design dialog box, shown in Figure A.34, the user is requested to enter the parameters such as the expected phase margin γ, the crossover frequency ω_c, and the steady-state error tolerance K_v.

Let us try a plant model given by $G(s) = 1/[s(s+1)(0.2s+1)]$. Set the expected phase margin $\gamma = 50°$, the crossover frequency $\omega_c = 5$ rad/sec, and the steady-state error tolerance $K_v = 100$. Then, a lead-lag compensator can be designed as shown in Figure A.35(a). With a proper menu selection, the controller can be shown in the factorized form as in Figure A.35(a). The Bode diagrams of the system before and after lead-lag compensation can be obtained using the Analysis | Bode Diagram menu item, as shown in Figure A.35(b).

Via CtrlLAB, it is also very easy to design the LQ optimal controller and the pole-placement controller with either full state feedback or observer-based structures. The straightforward model-based controllers can also be designed with CtrlLAB.

A.5.2 Design of PID Controllers

Consider the PID controller design problem with the plant model $G(s) = 10/[(s+1)(s+2)(s+3)(s+4)]$ entered via CtrlLAB. By the Design | PID Controller menu item, the design menu will appear as shown in Figure A.36. It can be seen that different PID controller design algorithms have been implemented within CtrlLAB. The "one-shot" submenu item



(a) lead-lag controller (b) Bode diagram comparison

Figure A.35. *A lead-lag compensator.*

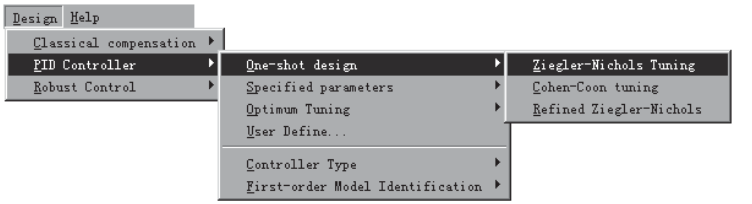
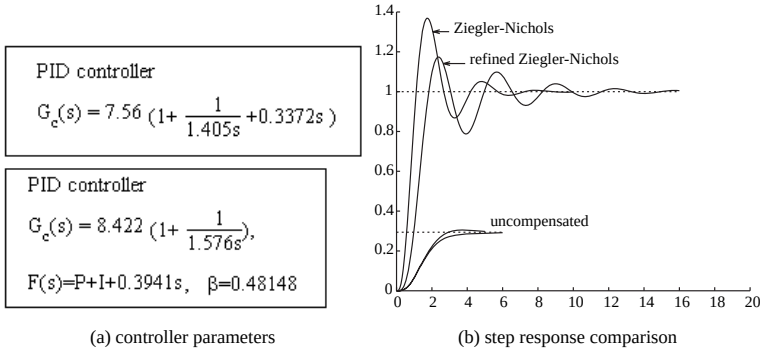


Figure A.36. *Main menu for PID controller design.*



(a) controller parameters (b) step response comparison

Figure A.37. *Ziegler–Nichols PID controller.*

in Figure A.36 means that the PID controller can be designed directly from the known plant model with no other extra specification needed. One may design a PID controller using the Ziegler–Nichols algorithm by selecting Design | PID controller | One-shot design | Ziegler–Nichols Tuning. This will immediately generate the PID controller as shown in Figure A.37(a). Furthermore, the refined Ziegler–Nichols controller can be designed, as also shown in Figure A.37(a), when the One-shot design | Refined Ziegler–Nichols menu is selected. By the Analysis | Step response menu item, the closed-loop step response of

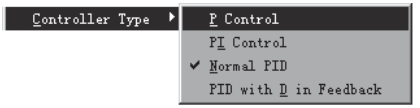


Figure A.38. *PID controller structures.*

Figure A.39. *FOPDT model fitting methods.*

Figure A.40. *PID with specified parameters.*

Figure A.41. *Optimum PID controller design.*

the system will be obtained as in Figure A.37(b) where it is shown together with the step response of the uncompensated system.

Apart from the standard PID controllers, other similar structures such as the P controller, the PI controller, and the PID controller with D in the feedback loop, can also be designed, which can be selected from the Design PID Controller | Controller Type menu item as shown in Figure A.38. We know that the PID controller parameter setting is based on the first-order plus dead time (FOPDT) model. Given a high-order plant model, we can select different approaches to fit the original plant model by a standard first-order model with dead time. The fitting algorithms can be selected from the menu shown in Figure A.39.

PID controllers can also be designed with other algorithms using the Specified parameters and Optimum Tuning menu items as shown in Figures A.40 and A.41, respectively.

With the above different tuning algorithms, we can design PID controllers that have better performance. For instance, the suboptimal first-order approximation to the plant model can be obtained using menu item First-order model identification | Optimal reduction, and from this an optimum PID controller can be designed. Using these controllers,

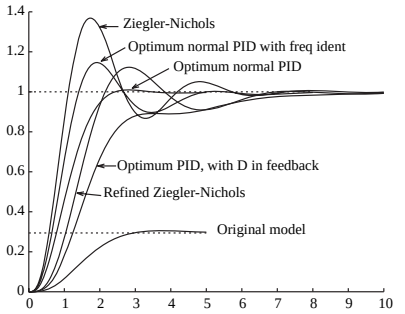


Figure A.42. *Step response comparison of different PID controllers.*

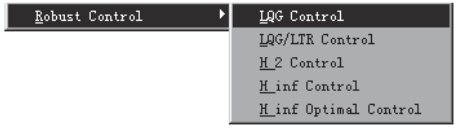


Figure A.43. *Robust control design menu.*

the closed-loop step responses are then compared as in Figure A.42. It can be seen that performance can be significantly improved, compared to the results from other "one-shot" PID controllers.

A.5.3 Robust Controller Design

In this section, only the $\mathcal{H}_\infty$ controller design via CtrlLAB will be demonstrated, although other design problems can also be solved in CtrlLAB. The example we shall use is the double integrator plant model as given in Example 7.16. The design submenus for the robust controllers can be obtained by selecting the Design | Robust Control menu item as shown in Figure A.43.

To get an $\mathcal{H}_\infty$ optimal controller, select the Design | Robust Control | H_inf Optimal Control menu item to obtain the dialog box shown in Figure A.44. Specify various weighting functions $W_1(s)$, $W_2(s)$, and $W_3(s)$ in the dialog box. To design an $\mathcal{H}_\infty$ controller for the sensitivity problem, check Sensitivity so that a new dialog box will appear as shown in Figure A.45(a). In Figure A.45(a), the expected order and the natural frequency for the ITAE standard reference model should be entered. For instance, if one selects $n = 2$ and $\omega_n = 10$ rad/sec, an optimal $\mathcal{H}_\infty$ controller can be designed as shown in Figure A.45(b).

The Nichols charts and the closed-loop step response of the system can then be obtained as shown in Figures A.46(a) and (b), respectively. Other types of robust controllers, such as the $\mathcal{H}_2$ controller and the LQG/LTR controllers, can also be designed and analyzed with little effort using the menus and dialog boxes.

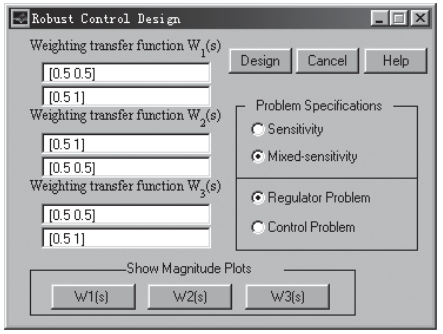
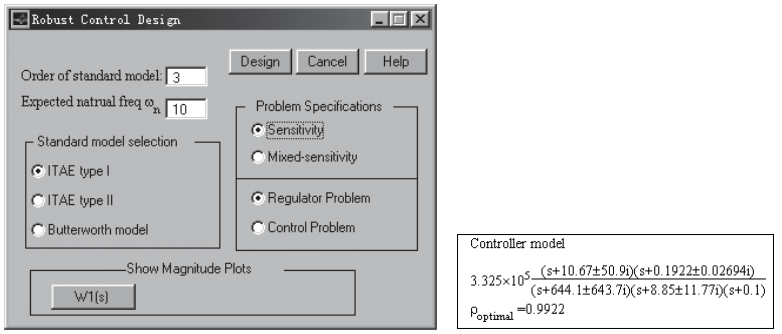
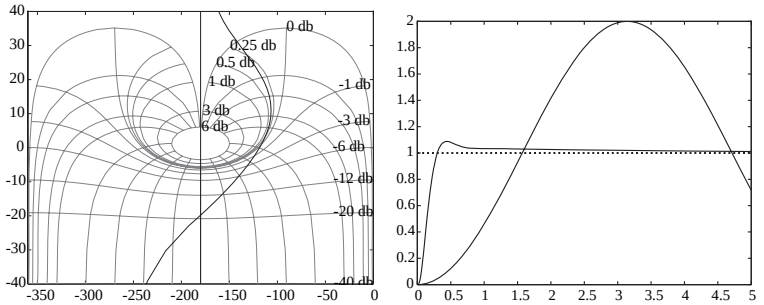


Figure A.44. *$\mathcal{H}$-norm-based dialog box.*



(a) dialog box for the sensitivity problem (b) optimal $\mathcal{H}_\infty$ controller

Figure A.45. *Robust control design results.*



(a) Nichols charts (b) closed-loop step response

Figure A.46. *Robust control system analysis.*

A.6 Graphical Interface-Based Tools

Two useful graphics-based tools are provided in CtrlLAB which can be used to process matrices and figures, respectively. In the following subsections, detailed descriptions of these two programs will be given.

A.6.1 A Matrix Processor

A matrix processor, `MatxProc()` is developed which can be used to process and edit matrices and state space models, and perform various kinds of matrix analyses in a visual way. The GUI facilities are extensively used to make the matrix processor very flexible and easy to use.

When `MatxProc` is typed in the MATLAB prompt, a GUI will appear as shown in Figure A.47. The program can also be called from within CtrlLAB. In MATLAB, `MatxProc()` can be called using the format `MatxProc(A)`, where `A` is a given matrix, or simply using `MatxProc`.

The File | New matrix menu can be selected to create a new matrix. The dialog box shown in Figure A.48 will appear to prompt the user to select from different matrix templates. For instance, if one selects a Hilbert matrix with 3 rows, the matrix will then be created by `MatxProc` as shown in Figure A.49.

Various display formats are allowed in `MatxProc()`. The user can select the Format menu as shown in Figure A.50(a). It can be seen that the user can specify different display precisions (high, normal, or rational), different alignment requirements (left, right, or center), and different truncating thresholds. For instance, the high precision display is given in Figure A.50(b), with part of the matrix elements hidden due to the limited size of the window. The hidden part of the matrix can be displayed via the horizontal scroll bar. The matrix can also be displayed in rational number format.

A matrix displayed can be analyzed and processed within `MatxProc()`. For instance, to analyze the matrix, simply select the Analysis to obtain the menu appearing in Figure A.51. To get the parameters of the given matrix, select the Analysis | Matrix

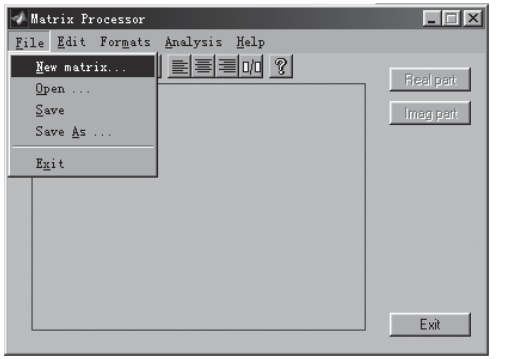


Figure A.47. *A matrix processor interface.*

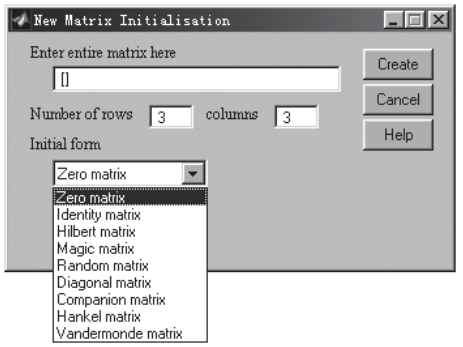


Figure A.48. *Matrix creating dialog box.*

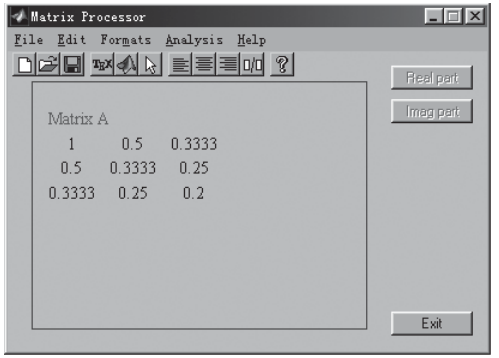
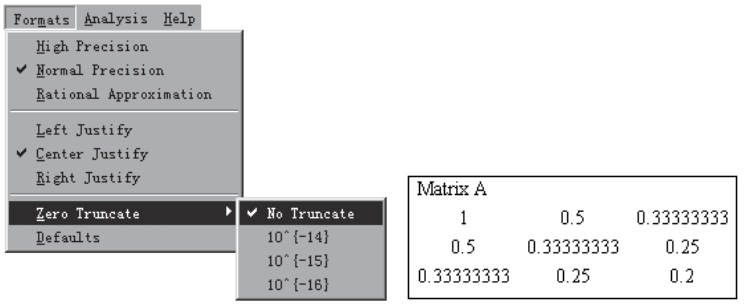


Figure A.49. *Creating a new matrix.*



(a) format menu (b) high precision display

Figure A.50. *Display formats of a matrix.*

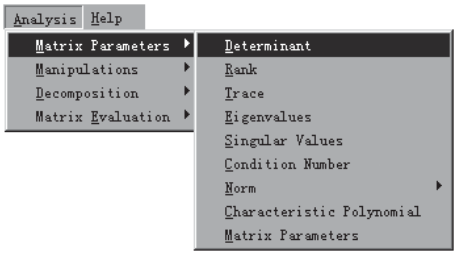


Figure A.51. *Matrix analysis menu.*

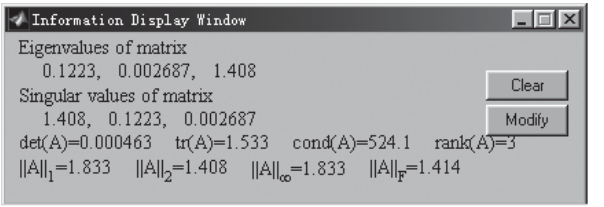
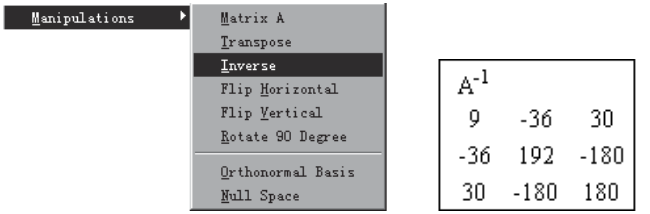


Figure A.52. *Matrix parameters display.*



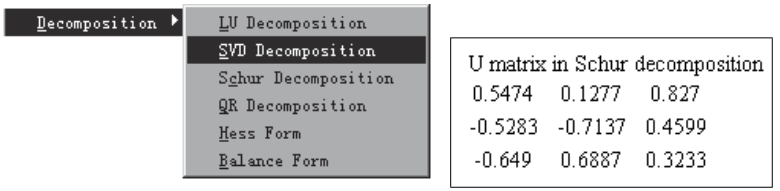
(a) manipulation menu (b) inverse matrix

Figure A.53. *Matrix manipulations.*

Parameters menu item. The analysis results will be obtained and displayed in the Information Display Window as shown in Figure A.52. Other analysis tasks such as evaluating the determinant, trace, norm, characteristic polynomial of the matrix can also be performed using the Analysis menu.

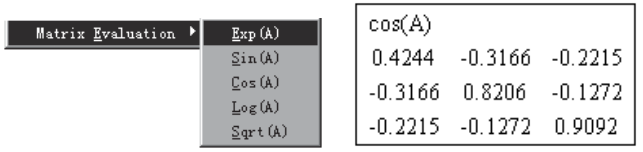
Matrix manipulation such as matrix inversion and rotation can be performed within `MatxProc()`. To manipulate the matrix, select the Analysis | Manipulations menu as shown in Figure A.53(a) to easily obtain, for example, the inversion of the matrix shown in Figure A.53(b).

Different decompositions for a given matrix can also be obtained, such as the QR decomposition, LU decomposition, singular value decomposition (SVD), etc. The Analysis | Decomposition menu is shown in Figure A.54(a), where the $\boldsymbol{U}$ matrix of the Schur decomposition can easily be obtained by selecting the relevant menu item, and the results are shown in Figure A.54(b). In addition, the button labeled T matrix in the GUI prompts the user to display the other matrix, for example, the $\boldsymbol{T}$ matrix, such that $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{T}\boldsymbol{U}^{\mathbf{T}}$.



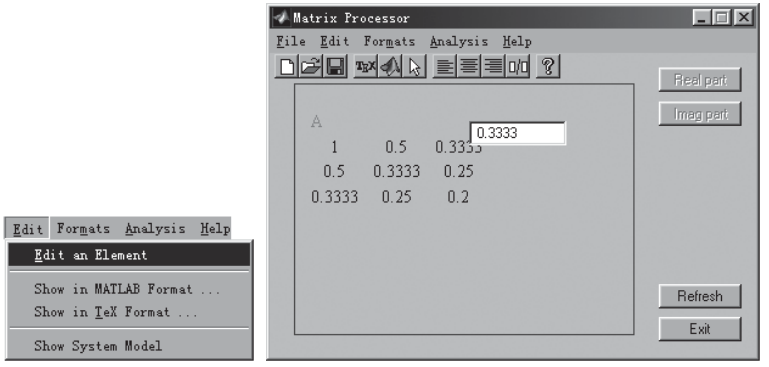
(a) decomposition menu (b) $\boldsymbol{U}$ matrix

Figure A.54. *Matrix decompositions.*



(a) matrix evaluation menu (b) cosine function

Figure A.55. *Matrix function evaluations.*

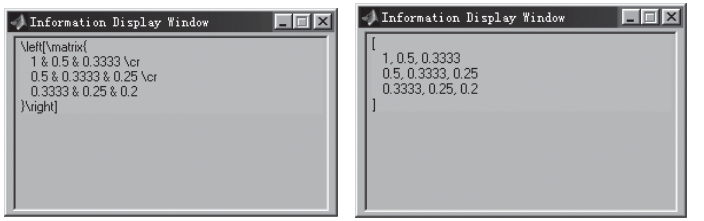


(a) matrix edit menu (b) matrix editing interface

Figure A.56. *Matrix editing facilities.*

Matrix function evaluations can be performed within `MatxProc()` by selecting the Analysis | Matrix Evaluation menu. Contents of the menu are displayed in Figure A.55(a). When the user selects the Cos(A) function display, the cosine of matrix $\boldsymbol{A}$ can be obtained as shown in Figure A.55(b).

A matrix can be edited using the Edit menu as shown in Figure A.56(a). By the Edit | Edit an Element menu item, the cursor will be changed to the cross sign, which prompts the user to select a matrix element. Once the user has selected an element to edit, the value of the element will be entered into the edit box for modification, as shown in Figure A.56(b). Once the edit process is done, the user can press the Accept button to confirm the change.



(a) TEX format (b) MATLAB format

Figure A.57. *Matrix display in other formats.*

The matrix can be shown in other formats as well, such as the TEX format and the MATLAB format. This is particularly useful in dealing with large and complicated matrices. For instance, the TEX format of the matrix can be obtained by selecting the Edit | Show in TeX Format menu item, and the result is as shown in Figure A.57(a), while the MATLAB format of the matrix is shown in Figure A.57(b).

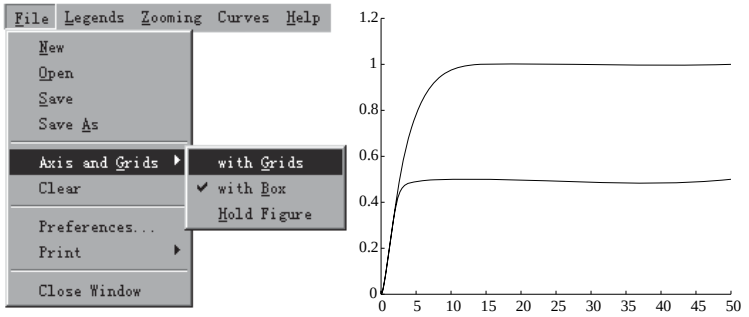
A.6.2 A Graphical Curve Processor

The graphical curve processor is not currently an independent MATLAB function. It has been integrated into CtrlLAB. It is mainly used to "decorate" the graphs obtained using CtrlLAB to any degree of complexity. It can be used to do simple things such as add or remove grids, add arrows, add floating legends to the graph, etc. Most of the figures in this book used this unique graphical curve processor within CtrlLAB. We remark that, although the current version of MATLAB has provided a plot editing toolbar for various graph editing utilities, the graphical curve processor within CtrlLAB has been working similarly and more powerfully with earlier versions of MATLAB (since version 4.2c) and is compatible with versions 5.x and 6.x. The ultimate objective of CtrlLab is to minimize user effort.

An Option menu in the standard MATLAB graphics window allows for some of the useful facilities to be called; this menu is shown in Figure A.58(a). For instance, via the Options | Axis and Grid | with Boxes off and Options | Axis and Grid | with Grid off menu items, the time response graph will then be changed to the display format shown in Figure A.58(b), where the grids and boxes are turned off.

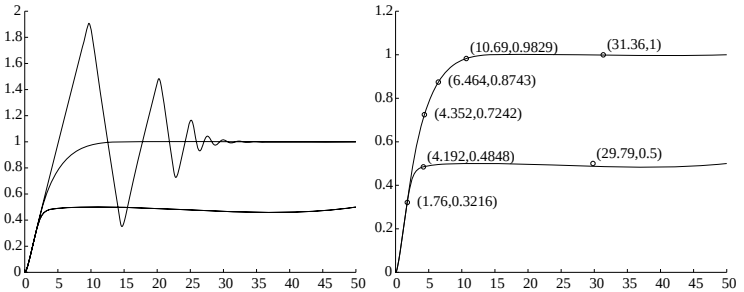
Note that, to turn off the grids, we can type `grid off` within the MATLAB command line. However, our objective here is to avoid such a user involvement. At this point, we remark again that CtrlLAB is designed for linear feedback control system analysis and design by *only mouse clicks* and some essential numeric key strokes. Great efforts have been made to minimize the user involvement in the analysis and design of feedback control systems. The Matrix Processor and Graph Processor described in this section are also part of the efforts to achieve this goal.

To draw several curves together with a common coordinate, select the Options | Axis and Grid | Hold on menu item to hold the current graph coordinate and then display another curve on the current plot. This is demonstrated in Figure A.59(a).



(a) Options menu (b) curve without box and grid

Figure A.58. *Graphics processor menu and results.*



(a) graph holding (b) cursor positioning

Figure A.59. *Screen hold and cursor.*

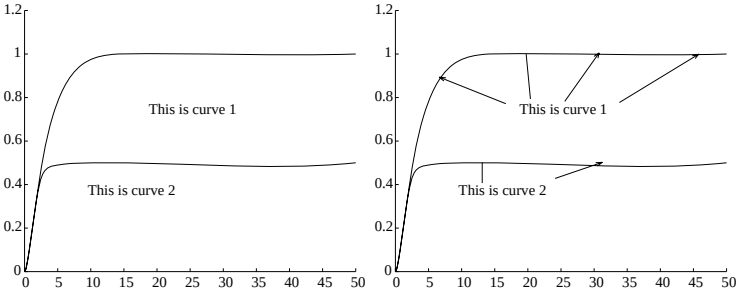
To cancel the hold protection, select the Options | Axis and Grid | Hold off menu item. To locate the specific points on the graph, use the Options | Cursor positions menu item. For instance, the curves with some points selected and marked are shown in Figure A.59(b).

Furthermore, various legends can be added to the graphs. The Options | Legends menu is shown in Figure A.60, where one can select to add, move, or edit text strings on the graphs, and also to draw lines or lines with arrows on the graph.

Two text legends are added on the graph shown in Figure A.61(a), and several lines and arrows can be further added on the graph as shown in Figure A.61(b). It can be seen that the legends (including lines and arrows) can be added or edited freely using the facilities provided. The user can also remove the legends by selecting Options | Legends | Delete a Legend to remove an existing legend.

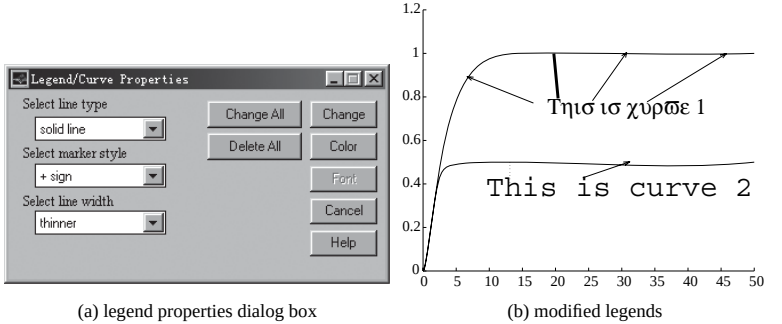
The properties of the legends can be modified if the user selects the Legends | Properties menu item, and a dialog box for assigning legend properties will be displayed as shown in Figure A.62(a). With proper settings, the modified version of the graph with different fonts, and line types will be obtained as shown in Figure A.62(b).

Figure A.60. *Legends menu.*



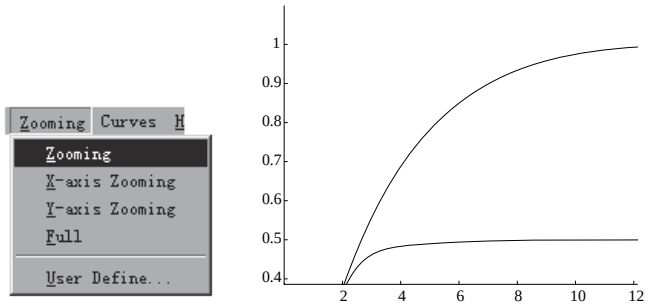
(a) examples of legends (b) examples of arrows and lines

Figure A.61. *Adding more legends on graphs.*



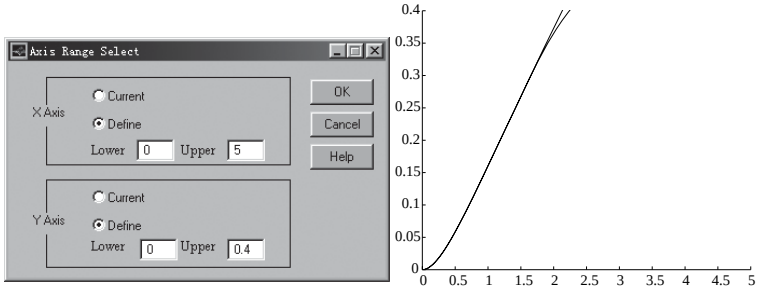
(a) legend properties dialog box (b) modified legends

Figure A.62. *Changing the properties of legends.*



(a) zoom menu (b) zoomed graphic display

Figure A.63. *Zoom facilities.*



(a) axis specification dialog box (b) zoomed graphic display

Figure A.64. *Axis range specifications.*

The user may also change the view in the graph window by selecting the Options | Zooming menu item as shown in Figure A.63(a), which allows the user to change the current coordinates using a mouse. For instance, the user can redefine the range for display by dragging the mouse, and the results can then be displayed as shown in Figure A.63(b).

Moreover, using the Zooming | User Define menu item, the dialog box shown in Figure A.64(a) will pop up to allow the user to select a reasonable display range. If the plot range in Figure A.64(a) is used, the zoomed output will be displayed as shown in Figure A.64(b).

Problems

1. Use the following plant models to test the previously described analysis and design tasks using CtrlLAB:

   (a) $G(s) = \dfrac{50000}{(s+1)(s+2)(s+3)(s+4)(s+5)(s+6)(s+7)(s+8)}$.

   (b) $\dot{\boldsymbol{x}} = \begin{bmatrix} 2.25 & -5 & -1.25 & -0.5 \\ 2.25 & -4.25 & -1.25 & -0.25 \\ 0.25 & -0.5 & -1.25 & -1 \\ 1.25 & -1.75 & -0.25 & -0.75 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 4 \\ 2 \\ 2 \\ 0 \end{bmatrix} u, \quad y = x_1 + 5x_2.$

   (c) The DC drive system given in Example 2.11. Use both the direct method and the Simulink method to create the system model.

2. Analyze the system matrix in problem 1(b). Find the norms, determinant, eigenvalues, and characteristic polynomial of $\boldsymbol{A}$, and do LU, QR, SVD decomposition of $\boldsymbol{A}$ within CtrlLAB. Find the matrices $e^{\boldsymbol{A}}$, $\sin(\boldsymbol{A})$, and $log(\boldsymbol{A})$.

3. Try to reproduce Figure 3.14(a) by using the graphics processor.

Bibliography

[1] Callier F. M., Desoer C. A. *Multivariable Feedback Systems*. New York: Springer-Verlag, 1982

[2] Freudenberg J. S., Looze D. P. *Frequency Domain Properties of Scalar and Multivariable Feedback Systems*, Lecture Notes in Control and Information Sciences, volume 104. Berlin: Springer-Verlag, 1988

[3] Maciejowski J. M. *Multivariable Feedback Design*. Wokingham, England: Addison-Wesley, 1989

[4] Postlethwaite I., MacFarlane A. G. J. *A Complex Variable Approach to the Analysis of Linear Multivariable Feedback Systems*. Berlin: Springer-Verlag, 1979

[5] Skogestad S., Postlethwaite I. *Multivariable Feedback Control*: *Analysis and Design*. Chichester, England: John Wiley & Sons, 1996

[6] Vardulakis A. I. G. *Linear Multivariable Control — Algebraic Analysis and Synthesis Methods*. Chichester, England: John Wiley & Sons, 1991

[7] Wonham W. M. *Linear Multivariable Control — A Geometric Approach*, Lecture Notes in Economics and Mathematical Systems, volume 101. Berlin: Springer-Verlag, 1974

[8] Mayr O. *The Origins of Feedback Control*. Cambridge, MA: MIT Press, 1970

[9] Minorsky N. *Directional stability of automatically steered bodies*. Journal of the American Society of Naval Engineering, 1922, 34(2):280–309

[10] Ziegler J. G., Nichols N. B. *Optimum settings for automatic controllers*. Transactions of the ASME, 1942, 64:759–768

[11] Nyquist H. *Regeneration theory*. Bell System Technology Journal, 1932, 11:126–147

[12] Bode H. W. *Network Analysis and Feedback Amplifier Design*. Princeton, NJ: Van Nostrand, 1945

[13] James H. M., Nichols N. B., Phillips R. S. *Theory of Servomechanisms*, MIT Radiation Laboratory Series, volume 25. New York: McGraw–Hill, 1947

[14] Evans W. R. *Graphical analysis of control systems*. Transactions of the AIEE, 1948, 67:547– 551

[15] Pontryagin L. S., Boltyanskii V. G., Gamkrelidze R. V., Mischenko E. F. The Mathematical Theory of Optimal Processes. New York: Interscience Publishers, 1962. Translated from the Russian by K. N. Trirogoff

[16] Bellman R. *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957

[17] Kalman R. E. *On the general theory of control systems*. IRE Transactions on Automatic Control, 1959, 4(3):110. Abstract. Full paper published in Proceedings of the 1st IFAC Congress, Moscow, 1960

[18] Kalman R. E. *Mathematical description of linear dynamical systems*. SIAM Journal of Control, 1963, 1(2):152–192

[19] Kalman R. E. *When is a linear control system optimal?* Transactions of ASME Journal of Basic Engineering Series D, 1964, 86:51–60

[20] Doyle J. C., Stein G. *Robustness with observers*. IEEE Transactions on Automatic Control, 1979, AC-24:607–611

[21] Zhang Z., Freudenberg J. S. *Loop transfer recovery for nonminimum phase plants*. IEEE Transactions on Automatic Control, 1990, 35(5):547–553

[22] Zames G. *Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms and approximate inverses*. IEEE Transactions on Automatic Control, 1981, AC-26(4):301–320

[23] Doyle J. C., Glover K., Khargoneckar P. P., Francis B. A. *State space solutions to standard H_2 and H_∞ control problems.* IEEE Transactions on Automatic Control, 1989, 34(8):831–847

[24] Melsa J. L., Jones S. K. *Computer Programmes for Computational Assistance in the Study of Linear Control Theory*. New York: McGraw–Hill, 1973

[25] Moler C. B. *MATLAB — An Interactive Matrix Laboratory*. Technical Report 369, University of New Mexico, Albuquerque, NM, 1980

[26] Åström K. J. *Computer aided tools for control system design*. In Jamshidi M, Herget C, eds., *Computer-Aided Control System Engineering*. Amsterdam: Elsevier Science Publishers B. V, 1985, 3–40

[27] *Using MATLAB version* 6.1. The MathWorks, Natick, MA, 2001

[28] Xue D. *Computer-aided Design of Control Systems with MATLAB*. Beijing: Tsinghua University Press (in Chinese), 1996

[29] Xue D., Chen Y. Q. *MATLAB/Simulink Based System Simulation Techniques.* Beijing: Tsinghua University Press (in Chinese), 2002

[30] Moore B. *Principal component analysis in linear systems: controllability, observability, and model reduction*. IEEE Transactions on Automatic Control, 1981, 26:17–32

[31] Ljung L. *System Identification — Theory for the User*. 2nd edition. Upper Saddle River, NJ: PTR Prentice Hall, 1999

[32] Akaike H. *A new look at the statistical model identification*. IEEE Transactions on Automatic Control, 1974, 19(6):716–723

[33] Levy E. C. *Complex-curve fitting*. IRE Transactions on Automatic Control, 1959, 4:37–43

[34] Andresen T. *A logarithmic-amplitude polar diagram*. Modeling, Identification and Control, 2001, 22(2):65–72

[35] Davison E. J. *A method for simplifying linear dynamic systems*. IEEE Transactions on Automatic Control, 1966, 11:93–101

[36] Atherton D. P., Borne P. *Concise Encyclopedia of Modelling and Simulation*. New York: Pergamon Press, 1992

[37] Bultheel A., van Barel M. *Padé techniques for model reduction in linear system theory: A survey*. Journal of Computational and Applied Mathematics, 1986, 14:401–438

[38] Decoster M., van Cauwenberghe A. R. *A comparative study of different reduction methods (Parts 1 & 2)*. Journal A, 1976, 17:68–74;125–134

[39] Hutton M. F. *Routh approximation for high-order linear systems*. In Proceedings of the 9th Allerton Conference. 1971, 160–169

[40] Shamash Y. *Linear system reduction using Padé approximation to allow retention of dominant modes*. International Journal of Control, 1975, 21:257–272

[41] Lucas T. N. *Some further observations on the differential method of model reduction*. IEEE Transactions on Automatic Control, 1992, 37:1389–1391

[42] Chen C. F., Chang C. Y., Han K. W. *Model reduction using the stability-equation method and the continued fraction method*. International Journal of Control, 1980, 32:81–94

[43] Hu X. H. *FF-Padé method of model reduction in frequency domain*. IEEE Transactions on Automatic Control, 1987, 32:243–246

[44] Hwang C., Lee Y. *Multi-frequency Padé approximation via Jordan continued-fraction expansion*. IEEE Transactions on Automatic Control, 1989, 34:444–446

[45] Xue D., Atherton D. P. *An optimal model reduction algorithm for linear systems*. In Proceedings of the American Control Conference. Boston, MA, 1991, 2128–2129

[46] Xue D. *Model Reduction Techniques and Applications*. Shenyang, China: Lecture Notes of Northeastern University, 1996

[47] Xue D., Atherton D. P. *A suboptimal reduction algorithm for linear systems with a time delay*. International Journal of Control, 1994, 60(2):181–196

[48] Gruca A., Bertrand P. *Approximation of high-order systems by low-order models with delays*. International Journal of Control, 1978, 28:953–965

[49] Glover K. *All optimal Hankel-norm approximations of linear multivariable systems and their $\mathcal{L}^\infty$-error bounds*. International Journal of Control, 1984, 39:1115–1193

[50] Stahl H., Hippe P. *Comments on "FF-Padé method of model reduction in frequency domain."* IEEE Transactions on Automatic Control, 1988, 33:415–416

[51] Atherton D. P. *Nonlinear Control Engineering — Describing Function Analysis and Design*. London: Van Nostrand Reinhold, 1975

[52] *Using Simulink Version* 4.1. The MathWorks, Natick, MA, 2001

[53] Franklin G. F., Powell J. D., Workman W. *Digital Control of Dynamic Systems*. 3rd edition. Reading, MA: Addison Wesley, 1988

[54] Frederick D. K., Rimer M. *Benchmark problem for CACSD packages*. In Abstracts of the Second IEEE Symposium on Computer-Aided Control System Design. Santa Barbara, CA, 1985

[55] Dorato P. *Linear Quadratic Control — An Introduction*. New York: McGraw–Hill, 1995

[56] Balasubramanian R. *Continuous Time Controller Design*. Stevenage, UK: Peter Peregrinus Ltd., 1989

[57] Kautskey J., Nichols N. K., Van Dooren P. *Robust pole-assignment in linear state feedback*. International Journal of Control, 1985, 41(5):1129–1155

[58] Dorf R. C., Bishop R. H. *Modern Control Systems*. 9th edition. Upper Saddle River, NJ: Prentice-Hall, 2001

[59] Bennett S. *Development of the PID controllers*. IEEE Control Systems Magazine, 1993, 13(2):58–65

[60] Åström K. J., Hägglund T. *PID Controllers: Theory, Design and Tuning*. Research Triangle Park: Instrument Society of America, 1995

[61] Åström K. J., Hägglund T. *Automatic Tuning of PID Controllers*. Research Triangle Park: Instrument Society of America, 1988

[62] Yu C. C. *Autotuning of PID Controllers: Relay Feedback Approach*. Advances in Industrial Control. London: Springer-Verlag, 1999

[63] Tan K. K., Wang Q.-G., Hang C. C., Hägglund T. *Advances in PID Control*. Advances in Industrial Control. London: Springer-Verlag, 2000

[64] Wang L. P., Cluett W. R. *From Plant Data to Process Control: Ideas for Process Identification and PID Design*. Research Triangle Park: Taylor & Francis, 2000

[65] Zhuang M. *Computer Aided PID Controller Design*. Ph.D. thesis, Sussex University, UK, 1992

[66] Chien K.-L., Hrones J. A., Reswick J. B. *On the automatic control of generalised passive systems*. Transactions of the ASME, 1952, 175–185

[67] Cohen G. H., Coon G. A. *Theoretical considerations of retarded control*. Transactions of the ASME, 1953, 827–834

[68] Hang C. C., Åström K. J., Ho W. K. *Refinement of the Ziegler–Nichols tuning formula*. Proceedings of the IEE, Part D, 1991, 138:111–118

[69] Wang F. S., Juang W. S., Chan C. T. *Optimal tuning of PID controllers for single and cascade control loops*. Chemical Engineering Communications, 1995, 132:15–34

[70] Zhuang M., Atherton D. P. *Automatic tuning of optimum PID controllers*. Proceedings of the IEE, Part D, 1993, 140:216–224

[71] O'Dwyer A. *Handbook of PI and PID Controller Tuning Rules*. London: Imperial College Press, 2003

[72] Visioli A. *Optimal tuning of PID controllers for integral and unstable processes*. Proceedings of the IEE, Part D, 2001, 148(2):180–184

[73] Haalman A. *Adjusting controllers for a deadtime process*. Control Engineering, 1965, 71–73

[74] McMillan G. K. *Control loop performance*. In Proceedings of the ISA/84 International Conference on Advances in Instrumentation. Houston, TX, 1984, 589–603

[75] O'Dwyer A. *PI and PID controller tuning rules for time delay processes: A summary*. Parts 1 & 2. In Proceedings of the Irish Signals and Systems Conference, 1999

[76] Nelder J. A., Mead R. *A simplex method for function minimization*. Computer Journal, 1965, 7:308–313

[77] Goldberg D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989

[78] Houck C. R., Joines J. A., Kay M. G. *A Genetic Algorithm for Function Optimization: A MATLAB Implementation*. Electronic Version of the GAOT Manual, 1995

[79] Åström K. J., Hang C. C., Persson P., Ho W. K. *Towards intellegient PID control*. Automatica, 1992, 28(1):1–9

[80] Stein G., Athans M. *The LQG/LTR procedure for multivariable feedback control design*. IEEE Transactions on Automatic Control, 1987, 32(2):105–114

[81] Zhou K. *Optimal and Robust Control*. Upper Saddle River, NJ: Prentice Hall, 1996

[82] Doyle J. C., Francis B. A., Tannerbaum A. R. *Feedback Control Theory*. New York: MacMillan Publishing Company, 1991

[83] Anderson B. D. O. *Controller design: Moving from theory to practice*. IEEE Control Systems Magazine, 1993, 13(4):16–25. Also, Bode Prize Lecture, CDC, 1992

[84] Anderson B. D. O., Liu Y. *Controller reduction: Concepts and approaches*. IEEE Transactions on Automatic Control, 1989, AC-34(8):802–812

[85] Chiang R. Y., Sofanov M. G. *Robust Control Toolbox User's Guide*. The MathWorks, Natick, MA, 1992

[86] Torvik P. J., Bagley R. L. *On the appearance of the fractional derivative in the behavior of real materials*. Transactions of the ASME, 1984, 51(4):294–298

[87] Podlubny I., Dorčák L., Misanek J. *Application of fractional-order derivatives to calculation of heat load intensity change in blast furnace walls*. Transactions of the Technical University of Kosice, 1995, 5(5):137–144

[88] Axtell M., Bise E. M. *Fractional calculus applications in control systems*. In Proceeding of the IEEE 1990 Natational Aerospace and Electronics Conference. New York, 1990, 563–566

[89] Dorčák L. *Numerical models for simulation the fractional-order control systems*. UEF SAV, The Academy of Sciences Institute of Experimental Physics. Kosice, Slovak Republic, 1994, 62–68

[90] Matignon D. *Stability result on fractional differential equations with applications to control processing*. In IMACS-SMC Proceedings. Lille, France, 1996, 963–968

[91] Oldham K. B., Spanier J. *The Fractional Calculus*. New York: Academic Press, 1974

[92] Podlubny I. *The Laplace transform method for linear differential equations of the fractional order*. In Proceedings of the 9th International BERG Conference. Kosice, Slovak Republic, 1997, 119–119 (in Slovak)

[93] Podlubny I. *Fractional Differential Equations*. San Diego: Academic Press, 1999

[94] Woon S. C. *Analytic continuation of operators — operators acting complex s-times. Applications: from number theory and group theory to quantum field and string theories*. Reviews in Mathematical Physics, 1999, 11(4):463–501

[95] Zavada P. *Operator of fractional derivative in the complex plane*. Communications in Mathematical Physics, 1998, 192(2):261–285

[96] Oustaloup A. *La Dérivation non Entière*. Paris: HERMES, 1995

[97] Petráš I., Dorčák L., Kostial I. *Control quality enhancement by fractional order controllers*. Acta Montanistica Slovaca, 1998, 2:143–148

[98] Podlubny I. *Fractional-Order Systems and Fractional-Order Controllers*. Technical Report UEF-03-94, The Academy of Sciences Institute of Experimental Physics, Kosice, Slovak Republic, 1994

[99] Podlubny I. *Fractional-order systems and $PI^{\lambda}D^{\mu}$-controllers*. IEEE Transactions on Automatic Control, 1999, 44(1):208–214

[100] Magin R. L. *Fractional Calculus in Bioengineering*. Redding, CT: Begell House Publishers, 2006

[101] Miller K. S., Ross B. *An Introduction to the Fractional Calculus and Fractional Differential Equations*. New York: Wiley, 1993

[102] Samko S. G., Kilbas A. A., Marichev O. I. *Fractional Integrals and Derivatives and Some of Their Applications*. Minsk: Nauka i Technika, 1987

[103] Xue D., Chen Y. Q. *MATLAB Solutions to Advanced Applied Mathematical Problems*. Beijing: Tsinghua University Press, 2004. (in Chinese)

[104] Hilfer R. *Applications of Fractional Calculus in Physics*. Singapore: World Scientific, 2000

[105] Petráš I., Podlubny I., O'Leary P. *Analogue Realization of Fractional Order Controllers*. Fakulta BERG, TU Košice, 2002

[106] Oustaloup A., Levron F., Mathiew B., Nanot F. *Frequency band complex noninteger differentiator: Characterization and synthesis*. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 2000, 47(1):25–39

[107] Xue D., Zhao C. N., Chen Y. Q. *A modified approximation method of fractional order system*. In Proceedings of the IEEE Conference on Mechatronics and Automation. Luoyang, China, 2006, 1043–1048

[108] Åström K. J. *Introduction to Stochastic Control Theory*. London: Academic Press, 1970

[109] Xue D., Zhao C. N., Chen Y. Q. *Fractional order PID control of a DC-motor with elastic shaft: A case study*. In Proceedings of the American Control Conference. Minneapolis, MN, 2006, 3182–3187

[110] Xue D., Goucem A., Atherton D. P. *A menu-driven interface to PC-MATLAB for a first course on control systems*. International Journal of Electrical Engineering Education, 1991, 28(1):21–33

Index of MATLAB Functions

Bold page numbers indicate where to find the syntax explanation of the function

Index