

# سایت اختصاصی مهندسی کنترل



<https://controlengineers.ir>



<https://t.me/controlengineers>



<https://www.instagram.com/controlengineers.ir>



---

# COMPUTATIONAL AIDS IN CONTROL SYSTEMS USING MATLAB

---

**Hadi Saadat**

Professor of Electrical Engineering  
Milwaukee School of Engineering  
Milwaukee, Wisconsin

controlengineers.ir

## Computational Aids in Control Systems Using MATLAB

Copyright © **Hadi Saadat** This eBook is distributed free of charge for personal use. You are hereby licensed to make copies of this eBook and the accompanying MATLAB files as long as you do not charge money or request donations for such copies. You may not alter this eBook in any way. You may not modify, rent or sell it, or create derivative works based upon this eBook. This eBook and the accompanying MATLAB files are provided “as is.” No warranty, express or implied, is made by the author as to the accuracy of the content of this eBook and functioning of the related program material, nor shall the fact of distribution constitute any such warranty, and no responsibility is assumed by the author in connection therewith.

# CONTENTS

<b>PREFACE</b>	<b>vi</b>
<b>1 INTRODUCTION TO MATLAB</b>	<b>1</b>
1.1 INSTALLING THE TEXT TOOLBOX . . . . .	2
1.2 RUNNING MATLAB . . . . .	2
1.3 VARIABLES . . . . .	4
1.4 OUTPUT FORMAT . . . . .	4
1.5 CHARACTER STRING . . . . .	7
1.6 VECTOR OPERATIONS . . . . .	7
1.7 ELEMENTARY MATRIX OPERATIONS . . . . .	10
1.7.1 UTILITY MATRICES . . . . .	12
1.7.2 EIGENVALUES . . . . .	12
1.8 COMPLEX NUMBERS . . . . .	13
1.9 POLYNOMIAL ROOTS AND CHARACTERISTIC POLYNOMIAL . . . . .	15
1.9.1 PRODUCT AND DIVISION OF POLYNOMIALS . . . . .	16
1.9.2 POLYNOMIAL CURVE FITTING . . . . .	17
1.9.3 POLYNOMIAL EVALUATION . . . . .	18
1.9.4 PARTIAL-FRACTION EXPANSION . . . . .	18
1.10 GRAPHICS . . . . .	19
1.11 GRAPHICS HARD COPY . . . . .	21
1.12 THREE-DIMENSIONAL PLOTS . . . . .	27
1.13 HANDLE GRAPHICS . . . . .	30
1.14 LOOPS AND LOGICAL STATEMENTS . . . . .	30
1.15 SIMULATION DIAGRAM . . . . .	36
1.16 INTRODUCTION TO SIMULINK . . . . .	38
1.16.1 SIMULATION PARAMETERS AND SOLVER . . . . .	39
1.16.2 THE SIMULATION PARAMETERS DIALOG BOX . . . . .	40
1.16.3 BLOCK DIAGRAM CONSTRUCTION . . . . .	41
1.16.4 USING THE TO WORKSPACE BLOCK . . . . .	47

iv CONTENTS

1.16.5	LINEAR STATE-SPACE MODEL FROM SIMULINK DIAGRAM . . . . .	47
1.16.6	SUBSYSTEMS AND MASKING . . . . .	49
<b>2</b>	<b>Mathematical Models of Systems</b>	<b>51</b>
2.1	Differential Equations of Physical Systems . . . . .	51
2.2	Numerical Solution . . . . .	51
2.3	Nonlinear Systems . . . . .	56
2.4	Linearization . . . . .	58
2.5	Transfer Function . . . . .	58
2.5.1	Polynomial Roots and Characteristic Polynomial . . . . .	59
2.5.2	Poles and Zeros of a Transfer Function . . . . .	60
2.5.3	Partial-Fraction Expansion . . . . .	61
<b>3</b>	<b>State-Space Representation</b>	<b>63</b>
3.1	State-Variable Modeling . . . . .	63
3.2	Equations of Electrical Networks . . . . .	64
3.3	Transfer Function to State-Space Conversion . . . . .	66
3.4	State-Space to Transfer Function Conversion . . . . .	66
3.5	Similarity Transformation . . . . .	67
3.5.1	Diagonalization of the A Matrix . . . . .	67
3.5.2	Transformation to Phase-variable . . . . .	68
3.6	Solution of the State Equation . . . . .	70
3.7	Laplace Transform of State Transition Matrix, $\Phi(s)$ . . . . .	70
3.8	Evaluation of $\phi(t)$ from the Characteristic Values of <b>A</b> . . . . .	71
3.8.1	Cayley-Hamilton Method . . . . .	71
3.9	Numerical Solution of the State Equation . . . . .	73
3.10	Block Diagram Reduction . . . . .	74
<b>4</b>	<b>System Responses</b>	<b>77</b>
4.1	The Response of Second-Order Systems . . . . .	77
4.2	Time-Domain Performance Specifications . . . . .	78
4.3	Effects of Additional Poles and Zeros . . . . .	79
4.3.1	Addition of a Zero . . . . .	79
4.3.2	Addition of a Pole . . . . .	79
4.4	Frequency Response of Systems . . . . .	83
4.5	Control System Toolbox	
	LTI Models and LTI Viewer . . . . .	91
4.5.1	LTI Models . . . . .	91
4.5.2	The LTI Viewer . . . . .	92
<b>5</b>	<b>Control System Characteristics</b>	<b>94</b>
5.1	Stability . . . . .	94
5.2	The Routh-Hurwitz Stability Criterion . . . . .	95
5.2.1	Special Cases . . . . .	96
5.3	Sensitivity . . . . .	98

5.4	Steady-State Error and System Type . . . . .	100
<b>6</b>	<b>Root-Locus Analysis and Design</b>	<b>104</b>
6.1	Root-Locus Method . . . . .	105
6.2	Summary of General Rules for Constructing Root-Loci . . . . .	106
6.3	Root-Locus Design . . . . .	117
6.4	Gain Factor Compensation or P Controller . . . . .	118
6.5	Phase-Lead Design . . . . .	120
6.6	Phase-Lag Design . . . . .	123
6.7	PID Design . . . . .	127
6.7.1	PD Controller . . . . .	127
6.7.2	PI Controller . . . . .	129
6.7.3	PID Controller . . . . .	130
6.8	Minor-Loop Feedback Control . . . . .	132
6.9	Rate Feedback or Tachometer-Feedback Control . . . . .	132
6.10	Feedback Compensation using Passive Elements . . . . .	135
6.11	GUI Program for Root-locus Design . . . . .	137
<b>7</b>	<b>Frequency Response Analysis and Design</b>	<b>142</b>
7.1	Frequency Response . . . . .	142
7.1.1	Bode Plot . . . . .	143
7.1.2	Polar Plot . . . . .	144
7.1.3	Log-magnitude versus Phase Plot . . . . .	145
7.2	Relative Stability . . . . .	145
7.2.1	Gain and Phase Margins . . . . .	146
7.2.2	Nyquist Stability Criterion . . . . .	148
7.2.3	Simplified Nyquist Criterion . . . . .	148
7.3	Closed-Loop Frequency Response . . . . .	150
7.3.1	Nichols Chart . . . . .	150
7.4	Frequency-Response Design . . . . .	152
7.5	Gain Factor Compensation or $P$ Controller . . . . .	154
7.6	Phase-Lead Design . . . . .	157
7.7	Phase-Lag Design . . . . .	161
7.8	PID Design . . . . .	163
7.8.1	PD Controller . . . . .	164
7.8.2	PI Controller . . . . .	166
7.8.3	PID Controller . . . . .	167
<b>8</b>	<b>Modern Control Design</b>	<b>170</b>
8.1	Pole-Placement Design . . . . .	171
8.2	Controllability . . . . .	175
8.3	Observer Design . . . . .	175
8.4	Observability . . . . .	178
8.5	Combined Controller-Observer Design . . . . .	178
8.6	Optimal Regulator Design . . . . .	180

---

## PREFACE

---

This text is intended to provide assistance in solving computational problems associated with the study and application of linear control systems. It is written expressly to support the use of *MATLAB* as a part of an introductory course in automatic control systems. *MATLAB*, developed by Math Works, Inc., is an interactive system for scientific and engineering computation and must be purchased separately from this book.

The objective is to introduce the user to some of the capabilities of *MATLAB*, and the associated *Control System Toolbox*, so that it can be used to aid in the design and analysis of control systems. The text contains sufficient detail and over 75 examples. These examples are designed with two objectives in mind. First, to teach the student how to write *MATLAB* programs, and second, to allow the user to advance rapidly in solving more advanced control system problems. The organization of this text is as follows:

**Chapter 1** is an introduction to *MATLAB* and its capabilities. It describes several examples dealing with matrix manipulations, complex algebra, and the graphics features of *MATLAB*.

**Chapter 2** shows how to obtain the numerical solution of differential equations, polynomial roots and characteristic polynomials, poles and zeros of transfer functions, and partial fraction expansion.

**Chapter 3** explains how to perform state-space transformations such as state-space to transfer function, state-space to zero-pole, transfer function to state-space, conversion of block diagrams to state-space models and reduced transfer functions.

**Chapter 4** deals with time response of second-order systems, time domain performance specifications, and the effects of adding poles and zeros to the closed-loop transfer function. An example of model reduction is also presented. Examples of frequency response, including some frequency domain specifications, are given.

**Chapter 5** covers some of the essential characteristics of feedback control systems such as stability, sensitivity, and steady-state errors.

**Chapter 6** includes Routh-Hurwitz Stability Criterion, root-locus analysis, design of phase-lead, phase-lag, and PID controllers by root-locus techniques.

**Chapter 7** deals with gain and phase margin, Bode, Nyquist, Nichols plots, and design of control systems in frequency domain.

**Chapter 8** deals with the design of control systems based on modern control theory. Pole-placement, state estimator and optimal regulator designs are presented.

I wish to express my sincere appreciation to my colleague Professor George Smith, of Milwaukee School of Engineering for his careful and meticulous review of the manuscript and his numerous constructive suggestions, which improved its content. I am especially grateful to Professor Ray Palmer, chairman of the department of Electrical Engineering and Computer Science, Milwaukee School of Engineering, for his aid and encouragement to begin the project in the first place. I express my profound gratitude to Ms. Lynn Kallas who spent many hours editing and correcting the manuscript. I am appreciative of the assistance given by the reviewers of this book: Professor Ashok Ambardar, Michigan Technological University; Professor Thomas J. Csermely, Syracuse University; Professor Shankar P. Bhattacharyya, Texas A&M; Professor Lee Keel, Tennessee State University; Professor Surjit S. Mahil, Purdue University Calumet. The constructive comments and encouragement provided by my editor Anne Brown of McGraw-Hill are greatly appreciated. Finally, I express my gratitude and love to my family, without whom this undertaking would not have been possible.

Hadi Saadat



---

# CHAPTER 1

---

## INTRODUCTION TO *MATLAB*

*MATLAB*, developed by Math Works Inc., is a software package for high performance numerical computation and visualization. The combination of analysis capabilities, flexibility, reliability, and powerful graphics makes *MATLAB* the premier software package for electrical engineers.

*MATLAB* provides an interactive environment with hundreds of reliable and accurate built-in mathematical functions. These functions provide solutions to a broad range of mathematical problems including matrix algebra, complex arithmetic, linear systems, differential equations, signal processing, optimization, nonlinear systems, and many other types of scientific computations. The most important feature of *MATLAB* is its programming capability, which is very easy to learn and to use, and which allows user-developed functions. It also allows access to Fortran algorithms and C codes by means of external interfaces. There are several optional toolboxes written for special applications such as signal processing, control systems design, system identification, statistics, neural networks, fuzzy logic, symbolic computations, and others. *MATLAB* has been enhanced by the very powerful *SIMULINK* program. *SIMULINK* is a graphical mouse-driven program for the simulation of dynamic systems. *SIMULINK* enables students to simulate linear, as well as nonlinear, systems easily and efficiently.

The following section describes the use of *MATLAB* and is designed to give a quick familiarization with some of the commands and capabilities of *MATLAB*. For a description of all other commands, *MATLAB* functions, and many other useful features, the reader is referred to the *MATLAB User's Guide*.

## 1.1 INSTALLING THE TEXT TOOLBOX

The software diskette included with the book contains all the developed functions and chapter examples. The file names for chapter examples begin with the letters **ch**. For example, the M-file for Example 2.4 is **ch2ex04**. Create a subdirectory, such as **HS**, where the *MATLABR11* toolbox resides. Copy all the files on the diskette to the subdirectory *MATLABR11 \TOOLBOX\HS*.

In the *MATLAB 5.3 Command Window* open the Path Browser by selecting **Set Path** from the **File** menu. Press to open the **Add to Path** window. Open the **toolbox** folder and double-click on the **HS** folder. Choose **Add to Back** option and click on **Save Settings** to save the new path permanently.

## 1.2 RUNNING *MATLAB*

*MATLAB* supports almost every computational platform. *MATLAB* for *WINDOWS* is started by clicking on the *MATLAB* icon. The **Command window** is launched, and after some messages such as intro, demo, help help, info, and others, the prompt “>>” is displayed. The program is in an interactive command mode. Typing **who** or **whos** displays a list of variable names currently in memory. Also, the **dir** command lists all the files on the default directory. *MATLAB* has an on-line help facility, and its use is highly recommended. The command **help** provides a list of files, built-in functions and operators for which on-line help is available. The command

`help function name`

will give information on the specified function as to its purpose and use. The command

`help help`

will give information as to how to use the on-line help.

*MATLAB* has a demonstration program that shows many of its features. The command **demo** brings up a menu of the available demonstrations. This will provide a presentation of the most important *MATLAB* facilities. Follow the instructions on the screen – it is worth trying.

*MATLAB 5.3* includes a Help Desk facility that provides access to on line help topics, documentation, getting started with *MATLAB*, online reference materials, *MATLAB* functions, real-time Workshop, and several toolboxes. The online documentation is available in HTML, via either Netscape Navigator or Microsoft Internet Explorer. The command **helpdesk** launches the Help Desk, or you can use the **Help** menu to bring up the Help Desk.

If an expression with correct syntax is entered at the prompt in the Command window, it is processed immediately and the result is displayed on the screen. If an expression requires more than one line, the last character of the previous line must

contain three dots "...". Characters following the percent sign are ignored. The (%) may be used anywhere in a program to add clarifying comments. This is especially helpful when creating a program. The command **clear** erases all variables in the Command window.

*MATLAB* is also capable of executing sequences of commands that are stored in files, known as script files or *M-files*. Clicking on **File, Open M-file**, opens the **Edit window**. A program can be written and saved in ASCII format with a filename having extension .m in the directory where *MATLAB* runs. To run the program, click on the Command window and type the filename without the .m extension at the *MATLAB* command "»". You can view the text Edit window simultaneously with the Command window. That is, you can use the two windows to edit and debug a script file repeatedly and run it in the Command window without ever quitting *MATLAB*.

In addition to the Command window and Edit window are the **Graphic windows** or **Figure windows** with grey (default) background. The plots created by the graphic commands appear in these windows.

Another type of M-file is a *function file*. A function provides a convenient way to encapsulate some computation, which can then be used without worrying about its implementation. In contrast to the script file, a *function file* has a name following the word "function" at the beginning of the file. The filename must be the same as the "function" name. The first line of a function file must begin with the function statement having the following syntax

function [output arguments] = function name (input arguments)

The output argument(s) are variables returned. A function need not return a value. The input arguments are variables passed to the function. Variables generated in function files are local to the function. The use of **global** variables make defined variables common and accessible between the main script file and other function files. For example, the statement **global R S T** declares the variables *R*, *S*, and *T* to be global without the need for passing the variables through the input list. This statement goes before any executable statement in the script and function files that need to access the values of the global variables.

Normally, while an M-file is executing, the commands of the file are not displayed on the screen. The command **echo** allows M-files to be viewed as they execute. **echo off** turns off the echoing of all script files. Typing **what** lists M-files and Mat-files in the default directory.

*MATLAB* follows conventional Windows procedure. Information from the command screen can be printed by highlighting the desired text with the mouse and then choosing the **print Selected ...** from the **File** menu. If no text is highlighted the entire Command window is printed. Similarly, selecting **print** from the Figure window sends the selected graph to the printer. For a complete list and help on general purpose commands, type **help general**.

### 1.3 VARIABLES

Expressions typed without a variable name are evaluated by *MATLAB*, and the result is stored and displayed by a variable called **ans**. The result of an expression can be assigned to a variable name for further use. Variable names can have as many as 19 characters (including letters and numbers). However, the first character of a variable name must be a letter. *MATLAB* is case-sensitive. Lower and uppercase letters represent two different variables. The command **casesen** makes *MATLAB* insensitive to the case. Variables in script files are global. The expressions are composed of operators and any of the available functions. For example, if the following expression is typed

```
x = exp(-0.2696*.2)*sin(2*pi*0.2)/(0.01*sqrt(3)*log(18))
```

the result is displayed on the screen as

```
x =
    18.0001
```

and is assigned to **x**. If a variable name is not used, the result is assigned to the variable **ans**. For example, typing the expression

```
250/sin(pi/6)
```

results in

```
ans =
    500.0000
```

If the last character of a statement is a semicolon (;), the expression is executed, but the result is not displayed. However, the result is displayed upon entering the variable name. The command **disp** may be used to display a variable without printing its name. For example, **disp(x)** displays the value of the variable without printing its name. If **x** contains a text string, the string is displayed.

### 1.4 OUTPUT FORMAT

While all computations in *MATLAB* are done in double precision, the default format prints results with five significant digits. The format of the displayed output can be controlled by the following commands.

MATLAB Command	Display
<b>format</b>	Default. Same as <b>format short</b>
<b>format short</b>	Scaled fixed point format with 5 digits
<b>format long</b>	Scaled fixed point format with 15 digits
<b>format short e</b>	Floating point format with 5 digits
<b>format long e</b>	Floating point format with 15 digits
<b>format short g</b>	Best of fixed or floating point with 5 digits
<b>format long g</b>	Best of fixed or floating point with 15 digits
<b>format hex</b>	Hexadecimal format
<b>format +</b>	The symbols +, - and blank are printed for positive, negative, and zero elements
<b>format bank</b>	Fixed format for dollars and cents
<b>format rat</b>	Approximation by ratio of small integers
<b>format compact</b>	Suppress extra line feeds
<b>format loose</b>	Puts the extra line feeds back in

For more flexibility in the output format, the command **fprintf** displays the result with a desired format on the screen or to a specified filename. The general form of this command is the following.

```
fprintf{fstr, A,...)
```

writes the real elements of the variable or matrix  $A, \dots$  according to the specifications in the string argument of **fstr**. This string can contain format characters like *ANCI C* with certain exceptions and extensions. **fprintf** is "vectorized" for the case when  $A$  is nonscalar. The format string is recycled through the elements of  $A$  (columnwise) until all the elements are used up. It is then recycled in a similar manner through any additional matrix arguments. The characters used in the format string of the commands **fprintf** are listed in the table below.

Format codes	Control characters
%e scientific format, lower case e	\n new line
%E scientific format, upper case E	\r beginning of the line
%f decimal format	\b back space
%s string	\t tab
%u integer	\g new page
%i follows the type	// apostrophe
%x hexadecimal, lower case	\\ back slash
%X hexadecimal, upper case	\a bell

A simple example of the **fprintf** is

```
fprintf('Area = %7.3f Square meters \n', pi*4.5^2)
```

The results is

## 6 1. INTRODUCTION TO MATLAB

Area = 63.617 Square meters

The %7.3f prints a floating point number seven characters wide, with three digits after the decimal point. The sequence \n advances the output to the left margin on the next line.

The following command displays a formatted table of the natural logarithmic for numbers 10, 20, 40, 60, and 80

```
x = [10; 20; 40; 60; 80];
y = [x, log(x)];
fprintf('\n Number    Natural log\n')
fprintf('%4i \t %8.3f\n',y')
```

The result is

Number	Natural log
10	2.303
20	2.996
40	3.689
60	4.094
80	4.382

An M-file can prompt for input from the keyboard. The command **input** causes the computer to request data from the keyboard. For example, the command

```
R = input('Enter radius in meter ')
```

displays the text string

```
Enter radius in meter
```

and waits for a number to be entered. If a number, say 4.5 is entered, it is assigned to variable R and displayed as

```
R =
    4.5000
```

The command **keyboard** placed in an M-file will stop the execution of the file and permit the user to examine and change variables in the file. Pressing **ctrl-z** terminates the keyboard mode and returns to the invoking file. Another useful command is **diary A:filename**. This command creates a file on drive A, and all output displayed on the screen is sent to that file. **diary off** turns off the diary. The contents of this file can be edited and used for merging with a word processor file. Finally, the command **save filename** can be used to save the expressions on the screen to a file named *filename.mat*, and the statement **load filename** can be used to load the file *filename.mat*.

**MATLAB** has a useful collection of transcendental functions, such as exponential, logarithm, trigonometric, and hyperbolic functions. For a complete list and help on operators, type **help ops**, and for elementary math functions, type **help elfun**.

## 1.5 CHARACTER STRING

A sequence of characters in single quotes is called a *character string* or *text variable*.

```
c = 'Good'
```

results in

```
c = Good
```

A text variable can be augmented with more text variables, for example,

```
cs = [c, ' luck']
```

produces

```
cs =  
    Good luck
```

## 1.6 VECTOR OPERATIONS

An  $n$  vector is a row or a column array of  $n$  numbers. In *MATLAB*, elements enclosed by brackets and separated by semicolons generate a column vector.

For example, the statement

```
X = [ 2; -4; 8]
```

results in

```
X =  
    2  
   -4  
    8
```

If elements are separated by blanks or commas, a row vector is produced. Elements may be any expression. The statement

```
R = [tan(pi/4) sqrt(9) -5]
```

results in the output

```
R =  
    1.0000    3.0000   -5.0000
```

The transpose of a column vector results in a row vector, and vice versa. For example

```
Y=R'
```

will produce

## 8 1. INTRODUCTION TO MATLAB

```
Y =
    1.0000
    3.0000
   -5.0000
```

*MATLAB* has two different types of arithmetic operations. Matrix arithmetic operations are defined by the rules of linear algebra. Array arithmetic operations are carried out element-by-element. The period character (.) distinguishes the array operations from the matrix operations. However, since the matrix and array operations are the same for addition and subtraction, the character pairs .+ and .- are not used.

Vectors of the same size can be added or subtracted, where addition is performed componentwise. However, for multiplication, specific rules must be followed in order to obtain the correct resulting values. The operation of multiplying a vector  $X$  with a scalar  $k$  (scalar multiplication) is performed componentwise. For example  $P = 5 * R$  produces the output

```
P =
    5.0000    15.0000   -25.0000
```

The inner product or the *dot product* of two vectors  $X$  and  $Y$  denoted by  $\langle X, Y \rangle$  is a scalar quantity defined by  $\sum_{i=1}^n x_i y_i$ . If  $X$  and  $Y$  are both column vectors defined above, the inner product is given by

```
S = X' * Y
```

and results in

```
S =
   -50
```

The operator  $.*$  performs element-by-element operation. For example, for the previously defined vectors,  $X$  and  $Y$ , the statement

```
E = X .* Y
```

results in

```
E =
     2
    -12
    -40
```

The operator  $./$  performs element-by-element division. The two arrays must have the same size, unless one of them is a scalar. Array powers or element-by-element powers are denoted by  $(.^)$ . The trigonometric functions, and other elementary mathematical functions such as **abs**, **sqrt**, **real**, and **log**, also operate element by element.

Various norms (measure of size) of a vector can be obtained. For example, the *Euclidean norm* is the square root of the inner product of the vector and itself. The command



## 1.6. VECTOR OPERATIONS 9

```
N = norm(X)
```

produces the output

```
N =
    9.1652
```

The angle between two vectors  $X$  and  $Y$  is defined by  $\cos \theta = \frac{\langle X, Y \rangle}{\|X\| \|Y\|}$ . The statement

```
Theta = acos( X'*Y/(norm(X)*norm(Y)) )
```

results in the output

```
Theta =
    2.7444
```

where Theta is in radians.

The *zero vector*, also referred to as origin, is a vector with all components equal to zero. For example, to build a zero row vector of size 4, the following command

```
Z = zeros(1, 4)
```

results in

```
Z =
    0    0    0    0
```

The *one vector* is a vector with each component equal to one. To generate a one vector of size 4, use

```
I = ones(1, 4)
```

The result is

```
I =
    1    1    1    1
```

In *MATLAB*, the colon (:) can be used to generate a row vector. For example

```
x = 1:8
```

generates a row vector of integers from 1 to 8.

```
x =
    1    2    3    4    5    6    7    8
```

For increments other than unity, the following command

```
z = 0 : pi/3 : pi
```

results in

## 10 1. INTRODUCTION TO MATLAB

```
z =
    0000    1.0472    2.0944    3.1416
```

For negative increments

```
x = 5 : -1:1
```

results in

```
x =
     5     4     3     2     1
```

Alternatively, special vectors can be created, the command **linspace(x, y, n)** creates a vector with  $n$  elements that are spaced linearly between  $x$  and  $y$ . Similarly, the command **logspace(x, y, n)** creates a vector with  $n$  elements that are spaced in even logarithmic increments between  $10^x$  and  $10^y$ .

## 1.7 ELEMENTARY MATRIX OPERATIONS

In *MATLAB*, a matrix is created with a rectangular array of numbers surrounded by brackets. The elements in each row are separated by blanks or commas. A semicolon must be used to indicate the end of a row. Matrix elements can be any *MATLAB* expression. The statement

```
A = [ 6  1  2; -1  8  3;  2  4  9]
```

results in the output

```
A =
     6     1     2
    -1     8     3
     2     4     9
```

If a semicolon is not used, each row must be entered in a separate line as shown below.

```
A = [ 6  1  2
    -1  8  3
     2  4  9]
```

The entire row or column of a matrix can be addressed by means of the symbol **(:)**. For example

```
r3 = A(3, :)
```

results in

```
r3 =
     2     4     9
```

Similarly, the statement  $A(:, 2)$  addresses all elements of the second column in  $A$ .

Matrices of the same dimension can be added or subtracted. Two matrices,  $A$  and  $B$ , can be multiplied together to form the product  $AB$  if they are conformable. Two symbols are used for nonsingular matrix division.  $A \setminus B$  is equivalent to  $A^{-1}B$ , and  $A/B$  is equivalent to  $AB^{-1}$ .

### Example 1.1

For the matrix equation below,  $AX = B$ , determine the vector  $X$ .

$$\begin{bmatrix} 4 & -2 & -10 \\ 2 & 10 & -12 \\ -4 & -6 & 16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -10 \\ 32 \\ -16 \end{bmatrix}$$

The following statements

```
A = [4 -2 -10; 2 10 -12; -4 -6 16];
B = [-10; 32; -16];
X = A\B
```

result in the output

```
X =
    2.0000
    4.0000
    1.0000
```

In addition to the built-in functions, numerous mathematical functions are available in the form of M-files. For the current list and their applications, see the *MATLAB User's Guide*.

### Example 1.2

Use the **inv** function to determine the inverse of matrix  $A$  in Example 1.1 and then determine  $X$ . The following statements

```
A = [4 -2 -10; 2 10 -12; -4 -6 16];
B = [-10; 32; -16];
C = inv(A)
X = C*B
```

result in the output

```
C =
    2.2000    2.3000    3.1000
    0.4000    0.6000    0.7000
    0.7000    0.8000    1.1000
X =
    2.0000
    4.0000
    1.0000
```

### Example 1.3

Use the **lu** factorization function to express the matrix  $A$  of Example 1.2 as the product of upper and lower triangular matrices,  $A = LU$ . Then find  $X$  from  $X = U^{-1}L^{-1}B$ .  
Typing

## 12 1. INTRODUCTION TO MATLAB

```

A = [ 4 -2 -10; 2 10 -12; -4 -6 16 ]
B = [-10; 32 -16];
[L,U] = lu(A)

```

results in

```

L =
    1.0000         0         0
    0.5000    1.0000         0
   -1.0000   -0.7273    1.0000

U =
    4.0000   -2.0000  -10.0000
         0   11.0000   -7.0000
         0         0    0.9091

```

Now entering

```
X = inv(U)*inv(L)*B
```

results in

```

X =
    2.0000
    4.0000
    1.0000

```

Dimensioning is automatic in *MATLAB*. You can find the dimensions and rank of an existing matrix with the **size** and **rank** statements. For vectors, use the command **length**.

### 1.7.1 UTILITY MATRICES

There are many special utility matrices which are useful for matrix operations. A few examples are

<b>eye(m, n)</b>	Generates an $m$ -by- $n$ identity matrix.
<b>zeros(m, n)</b>	Generates an $m$ -by- $n$ matrix of zeros.
<b>ones(m, n)</b>	Generates an $m$ -by- $n$ matrix of ones.
<b>diag(x)</b>	Produces a diagonal matrix with the elements of <b>x</b> on the diagonal line.

For a complete list and help on elementary matrices and matrix manipulation, type **help elmat**. There are many other special built-in matrices. For a complete list and help on specialized matrices, type **help specmat**.

### 1.7.2 EIGENVALUES

If  $A$  is an  $n$ -by- $n$  matrix, the  $n$  numbers  $\lambda$  that satisfy  $Ax = \lambda x$  are the eigenvalues of  $A$ . They are found using **eig(A)**, which returns the eigenvalues in a column vector.

Eigenvalues and eigenvectors can be obtained with a double assignment statement  $[X, D] = \text{eig}(A)$ . The diagonal elements of  $D$  are the eigenvalues and the columns of  $X$  are the corresponding eigenvectors such that  $AX = XD$ .

#### Example 1.4

Find the eigenvalues and the eigenvectors of the matrix  $A$  given by

$$A = \begin{bmatrix} 0 & 1 & -1 \\ -6 & -11 & 6 \\ -6 & -11 & 5 \end{bmatrix}$$

```
A = [ 0 1 -1; -6 -11 6; -6 -11 5];
[X,D] = eig(A)
```

The eigenvalues and the eigenvectors are obtained as follows

```
X =
-0.7071    0.2182   -0.0921
 0.0000    0.4364   -0.5523
-0.7071    0.8729   -0.8285

D =
-1     0     0
 0    -2     0
 0     0    -3
```

## 1.8 COMPLEX NUMBERS

All the *MATLAB* arithmetic operators are available for complex operations. The imaginary unit  $\sqrt{-1}$  is predefined by two variables  $i$  and  $j$ . In a program, if other values are assigned to  $i$  and  $j$ , they must be redefined as imaginary units, or other characters can be defined for the imaginary unit.

```
j = sqrt(-1)    or i = sqrt(-1)
```

Once the complex unit has been defined, complex numbers can be generated.

#### Example 1.5

Evaluate the following function  $V = Zc \cosh g + \sinh g / Zc$ , where  $Zc = 200 + j300$  and  $g = 0.02 + j1.5$

```
i = sqrt(-1); Zc = 200 + 300*i; g = 0.02 + 1.5*i;
v = Zc *cosh(g) + sinh(g)/Zc
```

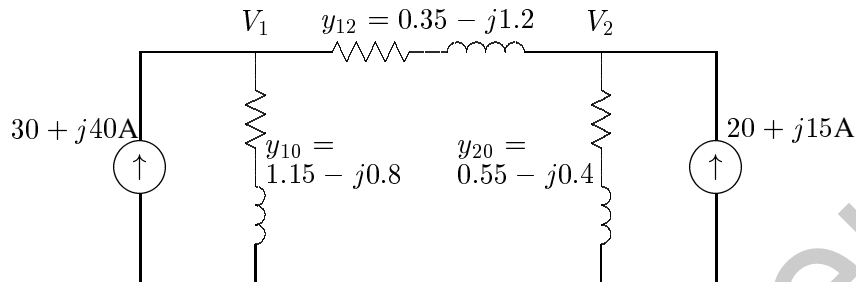
results in the output

```
v =
8.1672 + 25.2172i
```

It is important to note that, when complex numbers are entered as matrix elements within brackets, we avoid any blank spaces. If spaces are provided around the complex number sign, it represents two separate numbers.

### Example 1.6

In the circuit shown in Figure 1.1, determine the node voltages  $V_1$  and  $V_2$  and the power delivered by each source.



**FIGURE 1.1**  
Circuit for Example 1.6.

Kirchhoff's current law results in the following matrix node equation.

$$\begin{bmatrix} 1.5 - j2.0 & -0.35 + j1.2 \\ -0.35 + j1.2 & 0.9 - j1.6 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 30 + j40 \\ 20 + j15 \end{bmatrix}$$

and the complex power of each source is given by  $S = VI^*$ . The following program is written to yield solutions to  $V_1$ ,  $V_2$  and  $S$  using *MATLAB*.

```

j=sqrt(-1)                                % Defining j
I=[30+j*40; 20+j*15]                      % Column of node current phasors
Y=[1.5-j*2   -.35+j*1.2; -.35+j*1.2   .9-j*1.6]
                                           % Complex admittance matrix Y
disp('The solution is') V=inv(Y)*I         % Node voltage solution
S=V.*conj(I)                              % complex power at nodes
    
```

result in

The solution is

```

V =
    3.5902 + 35.0928i
    6.0155 + 36.2212i
    
```

```

S =
   1511.4 + 909.2i
    663.6 + 634.2i
    
```

In *MATLAB*, the conversion between polar and rectangular forms makes use of the following functions:

Operation	Description
$z = a + bi$ or $z = a + j * b$	Rectangular form
<code>real(z)</code>	Returns real part of $z$
<code>imag(z)</code>	Returns imaginary part of $z$
<code>abs(z)</code>	Absolute value of $z$
<code>angle(z)</code>	Phase angle of $z$
<code>conj(z)</code>	Conjugate of $z$
$z = M * \exp(j * \theta)$	converts $M \angle \theta$ to rectangular form

The prime (') transposes a real matrix; but for complex matrices, the symbol (.'') must be used to find the transpose.

## 1.9 POLYNOMIAL ROOTS AND CHARACTERISTIC POLYNOMIAL

If  $p$  is a row vector containing the coefficients of a polynomial, **roots(p)** returns a column vector whose elements are the roots of the polynomial. If  $r$  is a column vector containing the roots of a polynomial, **poly(r)** returns a row vector whose elements are the coefficients of the polynomial.

### Example 1.7

Find the roots of the following polynomial.

$$s^6 + 9s^5 + 31.25s^4 + 61.25s^3 + 67.75s^2 + 14.75s + 15$$

The polynomial coefficients are entered in a row vector in descending powers. The roots are found using **roots**.

```
p = [ 1 9 31.25 61.25 67.75 14.75 15 ]
r = roots(p)
```

The polynomial roots are obtained in column vector

```
r =
-4.0000
-3.0000
-1.0000 + 2.0000i
-1.0000 - 2.0000i
0.0000 + 0.5000i
0.0000 - 0.5000i
```

### Example 1.8

The roots of a polynomial are  $-1, -2, -3 \pm j4$ . Determine the polynomial equation.

Complex numbers may be entered using function  $i$  or  $j$ . The roots are then entered in a column vector. The polynomial equation is obtained using **poly** as follows

## 16 1. INTRODUCTION TO MATLAB

```

i = sqrt(-1)
r = [-1 -2 -3+4*i -3-4*i ]
p = poly(r)
    
```

The coefficients of the polynomial equation are obtained in a row vector.

```

p =
    1     9    45    87    50
    
```

Therefore, the polynomial equation is

$$s^4 + 9s^3 + 45s^2 + 87s + 50 = 0$$

### Example 1.9

Determine the roots of the characteristic equation of the following matrix.

$$A = \begin{bmatrix} 0 & 1 & -1 \\ -6 & -11 & 6 \\ -6 & -11 & 5 \end{bmatrix}$$

The characteristic equation of the matrix is found by **poly**, and the roots of this equation are found by **roots**.

```

A = [ 0  1 -1; -6 -11 6; -6 -11 5];
p = poly(A)
r = roots(p)
    
```

The result is as follows

```

p =
    1.0000    6.0000   11.0000    6.0000
r =
   -3.0000
   -2.0000
   -1.0000
    
```

The roots of the characteristic equation are the same as the eigenvalues of matrix  $A$ . Thus, in place of the **poly** and **roots** function, we may use

```
r = eig(A)
```

### 1.9.1 PRODUCT AND DIVISION OF POLYNOMIALS

The product of polynomials is the convolution of the coefficients. The division of polynomials is obtained by using the deconvolution command.



### Example 1.10

(a) Given  $A = s^2 + 7s + 12$ , and  $B = s^2 + 9$ , find  $C = AB$ .

(b) Given  $Z = s^4 + 9s^3 + 37s^2 + 81s + 52$ , and  $Y = s^2 + 4s + 13$ , find  $X = \frac{Z}{Y}$ .

The commands

```

A = [1 7 12]; B = [1 0 9];
C = conv(A, B)
Z = [1 9 37 81 52]; Y = [1 4 13];
[X, r] = deconv(Z, Y)
    
```

result in

```

C =
    1    7   21   63  108
X =
    1    5    4
r =
    0    0    0
    
```

### 1.9.2 POLYNOMIAL CURVE FITTING

In general, a polynomial fit to data in vector  $x$  and  $y$  is a function  $p$  of the form

$$p(x) = c_1 x^d + c_2 x^{d-1} + \dots + c_n$$

The degree is  $d$ , and the number of coefficients is  $n = d + 1$ . Given a set of points in vectors  $x$  and  $y$ , **polyfit(x, y, d)** returns the coefficients of  $d$ th order polynomial in descending powers of  $x$ .

### Example 1.11

Find a polynomial of degree 3 to fit the following data

$x$	0	1	2	4	6	10
$y$	1	7	23	109	307	1231

```

x = [ 0  1  2  4  6 10];
y = [ 1  7 23 109 307 1231];
c = polyfit(x,y,3)
    
```

The coefficients of a third degree polynomial are found as follows

```

c =
    1.0000    2.0000    3.0000    1.0000
    
```

i.e.,  $y = x^3 + 2x^2 + 3x + 1$ .

### 1.9.3 POLYNOMIAL EVALUATION

If  $c$  is a vector whose elements are the coefficients of a polynomial in descending powers, the **polyval(c, x)** is the value of the polynomial evaluated at  $x$ . For example, to evaluate the above polynomial at points 0, 1, 2, 3, and 4, use the commands

```
c = [1 2 3 1];
x = 0:1:4;
y = polyval(c, x)
```

which result in

```
y =
    7    23    55   109
```

### 1.9.4 PARTIAL-FRACTION EXPANSION

**[r, p, k] = residue[b, a]** finds the residues, poles, and direct terms of a partial fraction expansion of the ratio of two polynomials

$$\frac{P(s)}{Q(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0}$$

Vectors **b** and **a** specify the coefficients of the polynomials in descending powers of  $s$ . The residues are returned in column vector **r**, the pole locations in column vector **p**, and the direct terms in row vector **k**.

#### Example 1.12

Determine the partial fraction expansion for

$$F(s) = \frac{2s^3 + 9s + 1}{s^3 + s^2 + 4s + 4}$$

```
b = [2 0 9 1];
a = [1 1 4 4];
[r,p,k] = residue(b,a)
```

The result is as follows

```
r =
    0.0000   -0.2500i
    0.0000    +0.2500i
   -2.0000

p =
    0.0000   +2.0000i
    0.0000   -2.0000i
   -1.0000

K =
    2.0000
```

Therefore the partial fraction expansion is

$$2 + \frac{-2}{s+1} + \frac{j0.25}{s+j2} + \frac{-j0.25}{s-j2} = 2 + \frac{-2}{s+1} + \frac{1}{s^2+4}$$

**[b, a] = residue(r, p, K)** converts the partial fraction expansion back to the polynomial  $P(s)/Q(s)$ .

For a complete list and help on matrix analysis, linear equations, eigenvalues, and matrix functions, type **help matfun**.

## 1.10 GRAPHICS

*MATLAB* can create high-resolution, publication-quality 2-D, 3-D, linear, semilog, log, polar, bar chart and contour plots on plotters, dot-matrix printers, and laser printers. Some of the 2-D graph types are **plot**, **loglog**, **semilogx**, **semi -logy**, **polar**, and **bar**. The syntax for the above plots includes the following optional symbols and colors.

COLOR SPECIFICATION		LINE STYLE-OPTION	
Long name	Short name	Style	Symbol
black	k	solid	—
blue	b	dashed	--
cyan	c	dotted	:
green	g	dash-dot	-.
magenta	m	point	.
red	r	circle	o
white	w	x-mark	x
yellow	y	plus	+
		star	*

Some of the Specialized 2-D plots are listed below:

area	Filled area plot
bar	Bar graph
barh	Horizontal bar graph
comet	Comet-like trajectory
ezplot	Easy to use function plotter
ezpolar	Easy to use polar coordinate plotter
feather	Feather plot
fill	Filled 2-D polygons
fplot	Plot function
hist	Histogram
pareto	Pareto chart
pie	Pie chart
plotmatrix	Scatter plot matrix
stem	Discrete sequence or "stem" plot
stairs	Stairstep plot

You have three options for plotting multiple curves on the same graph. For example,

```
plot(x1, y1, 'r', x2, y2, '+b', x3, y3, '--')
```

plots  $(x_1, y_1)$  with a solid red line,  $(x_2, y_2)$  with a blue + mark, and  $(x_3, y_3)$  with a dashed line. If **X** and **Y** are matrices of the same size, **plot(X, Y)** will plot the columns of **Y** versus the column of **X**.

Alternatively, the **hold** command can be used to place new plots on the previous graph. **hold on** holds the current plot and all axes properties; subsequent plot commands are added to the existing graph. **hold off** returns to the default mode whereby a new plot command replaces the previous plot. **hold**, by itself, toggles the hold state.

Another way for plotting multiple curves on the same graph is the use of the **line** command. For example, if a graph is generated by the command `plot(x1, y1)`, then the commands

```
line(x2, y2, '+b')
line(x3, y3, '--')
```

Add curve  $(x_2, y_2)$  with a blue + mark, and  $(x_3, y_3)$  with a dashed line to the existing graph generated by the previous plot command. Multiple figure windows can be created by the **figure** command. **figure**, by itself, opens a new figure window, and returns the next available figure number, known as the figure handle. **figure(h)** makes the figure with handle **h** the current figure for subsequent plotting commands. Plots may be annotated with title,  $x - y$  labels and grid. The command **grid** adds a grid to the graph. The commands **title('Graph title')** titles the plot, and **xlabel('x-axis label')**, **ylabel('y-axis label')** label the plot with the specified string argument. The command **text(x-coordinate, y-coordinate, 'text')** can be used for placing text on the graph, where the coordinate values are taken from the current plot. For example, the statement

```
text(3.5, 1.5, 'Voltage')
```

will write Voltage at point (3.5, 1.5) in the current plot. Alternatively, you can use the **gtext('text')** command for interactive labeling. Using this command after a plot provides a crosshair in the Figure window and lets the user specify the location of the text by clicking the mouse at the desired location. Finally, the command **legend(string1, string2, string3, ...)** may be used to place a legend on the current plot using the specified strings as labels. This command has many optional arguments. For example, **legend(linetype1, string1, linetype2, string2, linetype3, string3, ...)** specifies the line types/color for each label at a suitable location. However, you can move the legend to a desired location with the mouse. **legend off** removes the legend from the current axes.

MATLAB provides automatic scaling. The command **axis([x min. x max. y min. y max.])** enforces the manual scaling. For example

```
axis([-10 40 -60 60])
```

produces an  $x$ -axis scale from  $-10$  to  $40$  and a  $y$ -axis scale from  $-60$  to  $60$ . Typing **axis** again or **axis('auto')** resumes auto scaling. Also, the aspect ratio of the plot can be made equal to one with the command **axis('square')**. With a square aspect ratio, a line with slope 1 is at a true 45 degree angle. **axis('equal')** will make the  $x$ - and  $y$ -axis scaling factor and tic mark increments the same. For a complete list and help on general purpose graphic functions, and two- and three-dimensional graphics, see **help graphics**, **help plotxy**, and **help plotxyz**.

There are many other specialized commands for two-dimensional plotting. Among the most useful are the **semilogx** and **semilogy**, which produce a plot with an  $x$ -axis log scale and a  $y$ -axis log scale. An interesting graphic command is the **comet** plot. The command **comet(x, y)** plots the data in vectors **x** and **y** with a comet moving through the data points, and you can see the curve as it is being plotted. For a complete list and help on general purpose graphic functions and two-dimensional graphics, see **help graphics** and **help plotxy**.

## 1.11 GRAPHICS HARD COPY

The easiest way to obtain hard-copy printout is to make use of the Windows built-in facilities. In the Figure window, you can pull down the file menu and click on the **Print** command to send the current graph directly to the printer. You can also import a graph to your favorite word processor. To do this, select **Copy options** from the **Edit** pull-down menu, and check mark the **Invert background** option in the dialog box to invert the background. Then, use **Copy** command to copy the graph into the clipboard. Launch your word processor and use the **Paste** command to import the graph.

Some word processors may not provide the extensive support of the Windows graphics and the captured graph may be corrupted in color. To eliminate this problem use the command

```
system_dependent(14, 'on')
```

which sets the metafile rendering to the lowest common denominator. To set the metafile rendering to normal, use

```
system_dependent(14, 'off')
```

In addition *MATLAB* provides a function called **print** that can be used to produce high resolution graphic files. For example,

```
print -dhpgl [filename]
```

saves the graph under the specified *filename* with extension *hgl*. This file may be processed with an HPGL-compatible plotter. Similarly, the command

```
print -dilll [filename]
```

produces a graphic file compatible with the Adobe Illustrator®88. Another **print** option allows you to save and reload a figure. The command

```
print -dmfile [ filename ]
```

produces a MAT file and M-file to reproduce the figure again.

In the Figure window, from the File pull-down menu you can use Save As... to save the figure with extension fig. This file can be opened in the Figure window again. Also, from the File pull-down menu you can use Expert... to save the graph in several different format, such as: emf, bmp, eps, ai, jpg, tiff, png, pcx, pbm, pgm, and ppm extensions.

### Example 1.13

Create a linear  $X$ - $Y$  plot for the following variables.

$x$	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
$y$	10	10	16	24	30	38	52	68	82	96	123

For a small amount of data, you can type in data explicitly using brackets.

```
x = [ 0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0];
y = [10  10  16  24  30  38  52  68  82  96  123];
plot(x, y), grid
xlabel('x'), ylabel('y'), title('A simple plot example')
```

**plot(x, y)** produces a linear plot of  $y$  versus  $x$  on the screen, as shown in Figure 1.2.

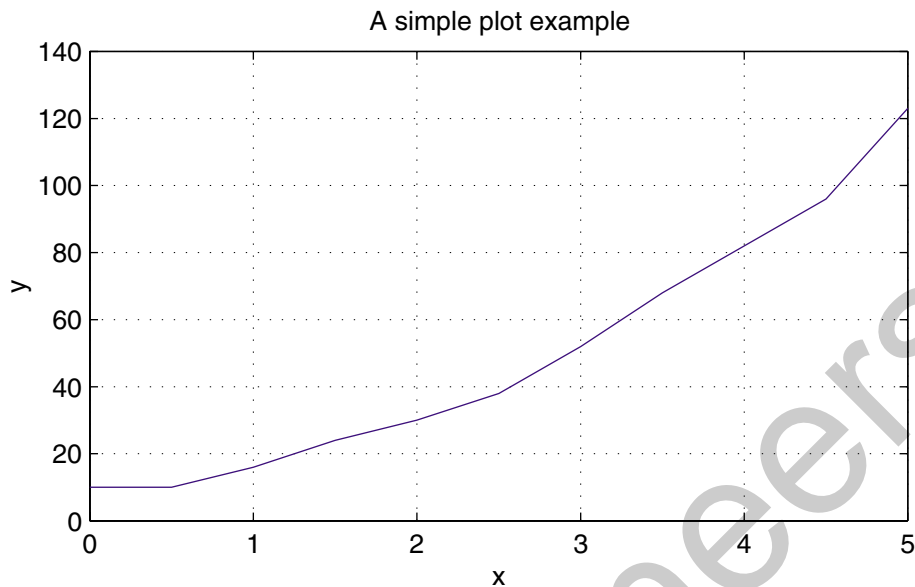
For large amounts of data, use the text editor to create a file with extension **m**. Typing the *filename* creates your data in the workspace.

### Example 1.14

Fit a polynomial of order 2 to the data in Example 1.13. Plot the given data point with symbol  $x$ , and the fitted curve with a solid line. Place a boxed legend on the graph.

The command **p = polyfit(x, y, 2)** is used to find the coefficients of a polynomial of degree 2 that fits the data, and the command **yc = polyval(p, x)** is used to evaluate the polynomial at all values in **x**. We use the following command.

```
x = [ 0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0];
y = [10  10  16  24  30  38  52  68  82  96  123];
p = polyfit(x, y, 2) % finds the coefficients of a polynomial
                     % of degree 2 that fits the data
yc = polyval(p, x); %polynomial is evaluated at all points in x
plot(x, y, 'x', x, yc) %plots data with x and fitted polynomial
xlabel('x'), ylabel('y'), grid
title('Polynomial curve fitting')
legend('Actual data', 'Fitted polynomial', 4)
```



**FIGURE 1.2**  
Example of X-Y plot.

The result is the array of coefficients of the polynomial of degree 2, and is

$$p = \begin{bmatrix} 4.0232 & 2.0107 & 9.6783 \end{bmatrix}$$

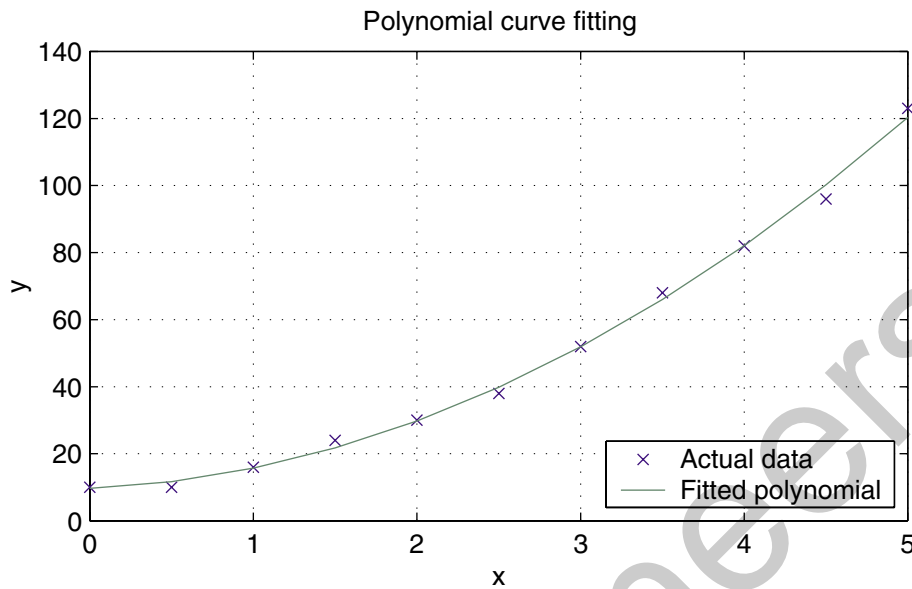
Thus, the parabola  $4.0x^2 + 2.0x + 9.68$  is found that fits the given data in the least-square sense. The plots are shown in Figure 1.3.

### Example 1.15

Plot function  $y = 1 + e^{-2t} \sin(8t - \pi/2)$  from 0 to 3 seconds. Find the time corresponding to the peak value of the function and the peak value. The graph is to be labeled, titled, and have grid lines displayed.

Remember to use `.*` for the element-by-element multiplication of the two terms in the given equation. The command `[cp, k] = max(c)` returns the peak value and the index `k` corresponding to the peak time. We use the following commands.

```
t=0:.005:3; c = 1+ exp(-2*t).*sin(8*t - pi/2);
[cp, k] = max(c) % cp is the maximum value of c at interval k
tp = t(k) % tp is the peak time
plot(t, c), xlabel(' t - sec'), ylabel('c'), grid
title('Damped sine curve')
text(0.6, 1.4, ['cp =', num2str(cp)]) %Text in quote & the value
% of cp are printed at the specified location
text(0.6, 1.3, ['tp = ', num2str(tp)])
```



**FIGURE 1.3**  
Fitting a parabola to the data in Example 1.13.

The result is

```

cp =
    1.4702
k =
    73
tp =
    0.3600
    
```

and the plot is shown in Figure 1.4.

An interactive way to find the data points on the curve is by using the **ginput** command. Entering **[x, y] = ginput** will put a crosshair on the graph. Position the crosshair at the desired location on the curve, and click the mouse. You can repeat this procedure for extracting coordinates for as many points as required. When the return key is pressed, the input is terminated and the extracted data is printed on the command menu. For example, to find the peak value and the peak time for the function in Example 1.15, try

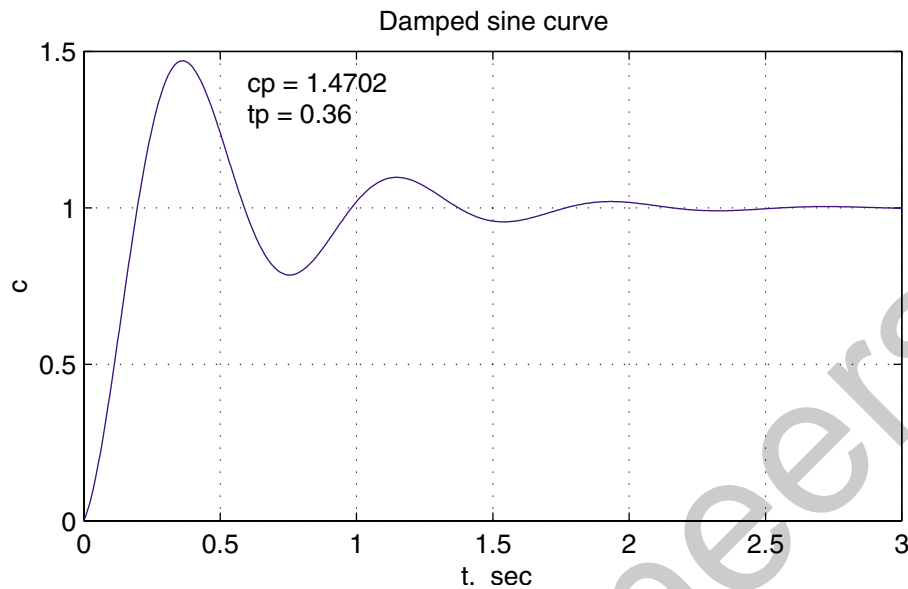
```
[tp, cp] = ginput
```

A crosshair will appear. Move the crosshair to the peak position, and click the mouse. Press the return key to get

```

cp =
    1.47
tp =
    0.36
    
```





**FIGURE 1.4**  
Graph of Example 1.15.

**subplot** splits the Figure window into multiple portions, in order to show several plots at the same time. The statement **subplot(m, n, p)** breaks the Figure window into an  $m$ -by- $n$  box and uses the  $p$ th box for the subsequent plot. Thus, the command **subplot(2, 2, 3)**, **plot(x,y)** divides the Figure window into four subwindows and plots  $y$  versus  $x$  in the third subwindow, which is the first subwindow in the second row. The command **subplot(111)** returns to the default Figure window. This is demonstrated in the next example.

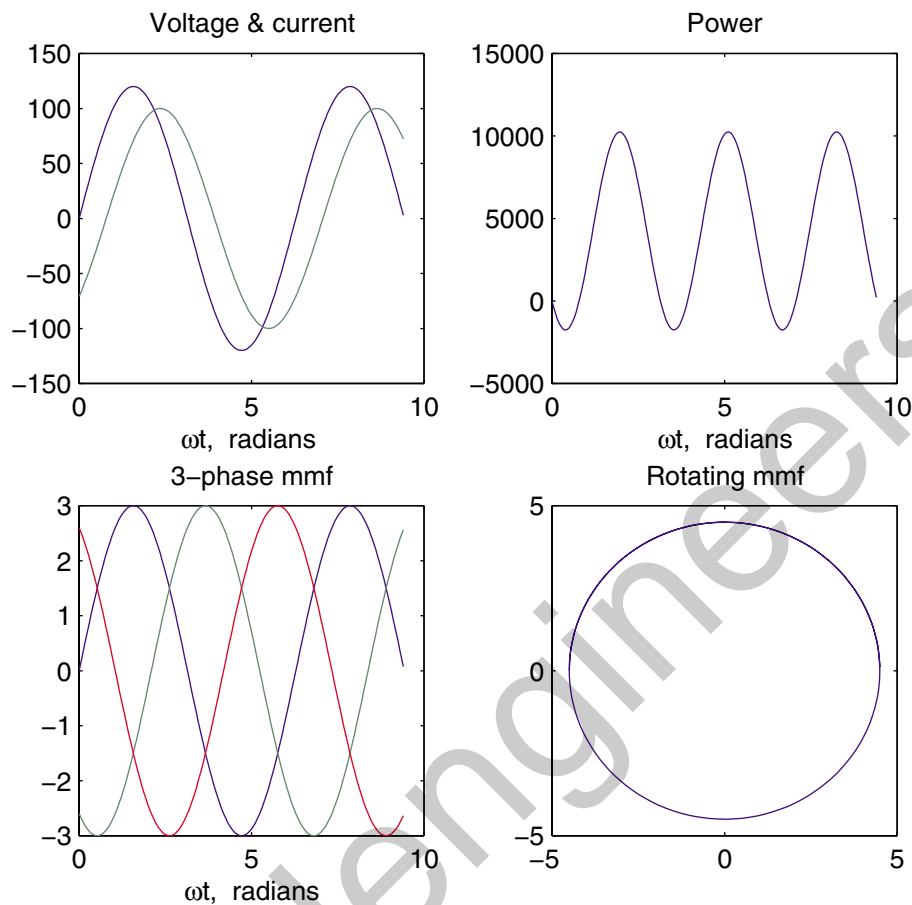
#### Example 1.16

Divide the Figure window into four partitions, and plot the following functions for  $\omega t$  from 0 to  $3\pi$  in steps of 0.05.

1. Plot  $v = 120 \sin \omega t$  and  $i = 100 \sin(\omega t - \pi/4)$  versus  $\omega t$  on the upper left portion.
2. Plot  $p = vi$  on the upper right portion.
3. Given  $F_m = 3.0$ , plot  $f_a = F_m \sin \omega t$ ,  $f_b = F_m \sin(\omega t - 2\pi/3)$ , and  $f_c = F_m \sin(\omega t - 4\pi/3)$  versus  $\omega t$  on the lower left portion.
4. For  $f_R = 3F_m$ , construct a circle of radius  $f_R$  on the lower right portion.

```

wt = 0: 0.05: 3*pi; v=120*sin(wt);           %Sinusoidal voltage
i = 100*sin(wt - pi/4);                       %Sinusoidal current
    
```



**FIGURE 1.5**  
Subplot demonstration.

```

p = v.*i; %Instantaneous power
subplot(2, 2, 1), plot(wt, v, wt, i); %Plot of v & i versus wt
title('Voltage & current'), xlabel('\omegat, radians');
subplot(2, 2, 2), plot(wt, p); % Instantaneous power vs. wt
title('Power'), xlabel(' \omegat, radians ')
Fm=3.0;
fa = Fm*sin(wt); % Three-phase mmf's fa, fb, fc
fb = Fm*sin(wt - 2*pi/3); fc = Fm*sin(wt - 4*pi/3);
subplot(2, 2, 3), plot(wt, fa, wt, fb, wt, fc)
title('3-phase mmf'), xlabel(' \omegat, radians ')
fR = 3/2*Fm;
subplot(2, 2, 4), plot(-fR*cos(wt), fR*sin(wt))
title('Rotating mmf'), subplot(111)
    
```

Example 1.16 results are shown in Figure 1.5.

## 1.12 THREE-DIMENSIONAL PLOTS

*MATLAB* provides extensive facilities for visualization of three-dimensional data. The most common are plots of curves in a three-dimensional space, mesh plots, surface plots, and contour plots. The command **plot3(x, y, z, 'style option')** produces a curve in the three-dimensional space. The viewing angle may be specified by the command **view(azimuth, elevation)**. The arguments *azimuth*, and *elevation* specifies the horizontal and vertical rotation in degrees, respectively. The **title**, **xlabel**, **ylabel**, etc., may be used for three-dimensional plots. The **mesh** and **surf** commands have several optional arguments and are used for plotting meshes and surfaces. The **contour(z)** command creates a contour plot of matrix **z**, treating the values in **z** as heights above the plane. The statement **mesh(z)** creates a three-dimensional plot of the elements in matrix **z**. A mesh surface is defined by the **z** coordinates of points above a rectangular grid in the **x-y** plane. The plot is formed by joining adjacent points with straight lines. **meshgrid** transforms the domain specified by vector **x** and **y** into arrays **X** and **Y**. For a complete list and help on general purpose Graphic functions and three-dimensional graphics, see **help graphics** and **help plotxyz**. Also type **demo** to open the *MATLAB Expo Menu Map* and visit *MATLAB*. Select and observe the demos in the Visualization section.

Following is a list of elementary 3-D plots and some specialized 3-D graphs.

plot3	Plot lines and points in 3-D space
mesh	3-D mesh surface
surf	3-D colored surface
fill3	Filled 3-D polygons
comet3	3-D comet-like trajectories
ezgraph3	General purpose surface plotter
ezmesh	Easy to use 3-D mesh plotter
ezmeshc	Easy to use combination mesh/contour plotter
ezplot3	Easy to use 3-D parametric curve plotter
ezsurf	Easy to use 3-D colored surface plotter
ezsurfc	Easy to use combination surf/contour plotter
meshc	Combination mesh/contour plot
meshz	3-D mesh with curtain
scatter3	3-D scatter plot
stem3	3-D stem plot
surfc	Combination surf/contour plot
trisurf	Triangular surface plot
trimesh	Triangular mesh plot
cylinder	Generate cylinder
sphere	Generate sphere

### Example 1.17 ch1ex17.m

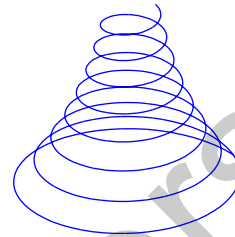
Few examples of 3-D plots and mesh plots are given in Figure 1.6.

Plot of a parametric space curve

$$x(t) = e^{-0.03t} \cos t, \quad y(t) = e^{-0.03t} \sin t, \quad z(t) = t$$

```

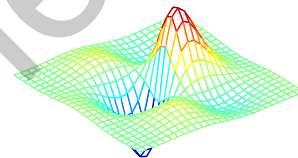
t= 0:0.1:16*pi;
x=exp(-0.03*t).*cos(t);
y=exp(-0.03*t).*sin(t);
z=t;
plot3(x, y, z), axis off
    
```



Plot of function  $z = \sin x \cos y e^{-(x^2 + y^2)^{0.5}}$  using mesh

```

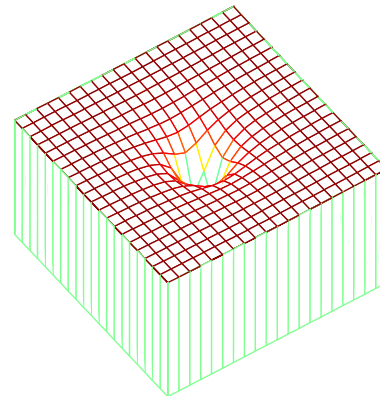
t = -4:0.3:4;
[x,y] = meshgrid(t,t);
z = sin(x).*cos(y).*exp(-(x.^2+y.^2).^0.5);
mesh(x,y,z), axis off
    
```



Plot of function  $z = -0.1/(x^2 + y^2 + 1)$  using meshz

```

x= -3:0.3:3; y=x;
[x, y]=meshgrid(x,y);
z=-0.1./(x.^2+y.^2+.1);
meshz(z) , axis off
view(-35, 60)
    
```



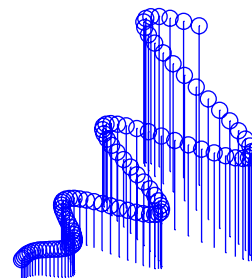
Discrete plot of

$$x=t, \quad y=t \cos t, \quad z=e^{0.1t}$$

using stem3

```

t=0:.2:20;
x=t; y=t.*cos(t);
z=exp(0.1*t);
stem3(x,y,z), axis off
    
```



Cylindrical surface created by

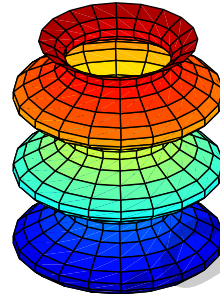
$$p=3+\sin t$$

using cylinder function

`t=0:pi/5:6*pi;`

`p=3+sin(t);`

`cylinder(p), axis off`



Plot of a unit sphere and a scaled sphere  
using sphere function

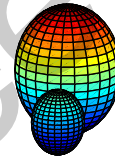
`[x,y,z]=sphere(24);`

`subplot(2,2,2), surf(x-2, y-2, z-1);`

`hold on`

`surf(2*x, 2*y, 2*z);`

`axis off`



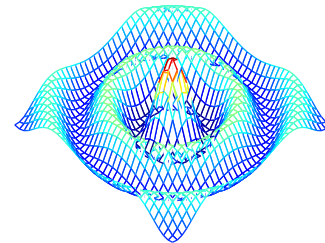
Cartezian plot of Bessel function

$$J_0[x^2+y^2]^{1/2} \quad -12 < x < 12, \quad -12 < y < 12$$

`[x,y]=meshgrid(-12:.7:12, -12:.7:12);`

`r=sqrt(x.^2+y.^2); z= bessel(0,r);`

`m=[-45 60]; mesh(z,m), axis off`



Contour lines and directional vectors  
using contour and quiver functions

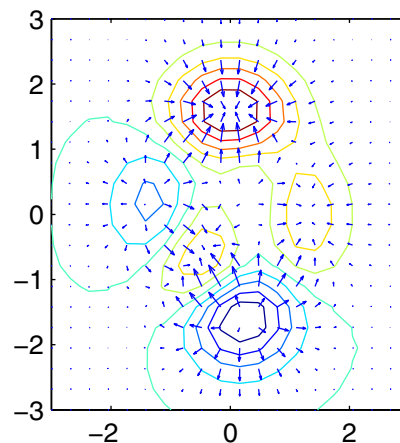
`[x,y,z]=peaks(20);`

`[nx, ny]=gradient(z,1,1);`

`contour(x,y,z,10)`

`hold on`

`quiver(x,y,nx,ny)`



**FIGURE 1.6**

Graphs of Example 1.17.

For more specialized 3-D graphs and color related functions see **help specgraph**.

### 1.13 HANDLE GRAPHICS

It is often desirable to be able to customize the graphical output. *MATLAB* allows object-oriented programming, enabling the user to have complete control over the details of a graph. *MATLAB* provides many low-level commands known as *Handle Graphics*. These commands makes it possible to access individual objects and their properties and change any property of an object without affecting other properties or objects. Handle Graphics provides a graphical user interface (GUI) in which the user interface includes push buttons and menus. These topics are not discussed here; like *MATLAB* syntax, they are easy to follow, and we leave these topics for the interested reader to explore.

### 1.14 LOOPS AND LOGICAL STATEMENTS

*MATLAB* provides loops and logical statements for programming, like **for**, **while**, and **if** statements. The **for** statement instructs the computer to perform all subsequent expressions up to the **end** statement for a specified number of counted times. The expression may be a matrix. The following is an example of a nested loop.

```
for i = 1:n, for j = 1:n
    expression
end, end
```

The **while** statement allows statements to be repeated an indefinite number of times under the control of a logic statement. The **if**, **else**, and **elseif** statements allow conditional execution of statements. *MATLAB* has six relational operators and four logical operators, which are defined in the following table.

<i>Relational Operator</i>	<i>Logical Operator</i>
<b>==</b> equal	<b>&amp;</b> logical AND
<b>~=</b> not equal	<b> </b> logical OR
<b>&lt;</b> less than	<b>~</b> logical complement
<b>&lt;=</b> less than or equal to	<b>xor</b> exclusive OR
<b>&gt;</b> greater than	
<b>&gt;=</b> greater than or equal to	

*MATLAB* is an interpreted language and macro operations such as matrices multiplies faster than micro operations such as incrementing an index, and use of loops are somewhat inefficient. Since variables in *MATLAB* are arrays and matrices, try to use vector operations as much as possible instead of loops. The loops should be used mainly for control operations. The use of loops are demonstrated in the next four examples.

### Example 1.18

A rectangular signal can be represented as a series sum of harmonically related sine or cosine signals. Consider the partial sum of the following periodic signals (Fourier series).

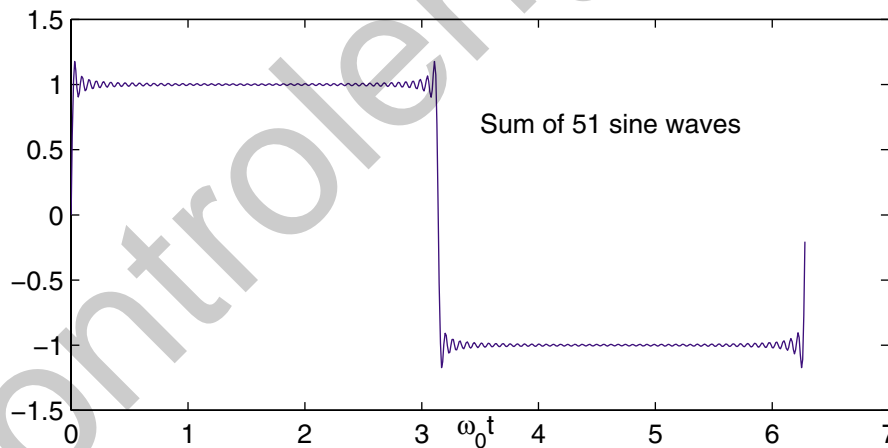
$$\begin{aligned}
 x(t) &= \frac{4}{\pi} \left[ \sin \omega_0 t + \frac{1}{3} \sin 3\omega_0 t + \frac{1}{5} \sin 5\omega_0 t + \cdots \right] \\
 &= \frac{4}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin n\omega_0 t \quad n \text{ is an odd integer}
 \end{aligned}$$

The following simple MATLAB statements uses a loop to generate this sum for any given odd integer.

```

n = input('Enter an odd integer');
w_0t = 0:.01:2*pi;
x = 0;
for k = 1:2:n;
    x = x + 1/k*sin(k*w_0t);
end
x=4/pi*x;
plot(w_0t, x), xlabel('\omega_0t')
text(3.5,.7,['Sum of ', num2str((n+1)/2),' sine waves'])
    
```

The result for  $n = 101$  is plotted as shown in Figure 1.7.



**FIGURE 1.7**  
Graph Of Example 1.18.

### Example 1.19

A network function known as transfer function is expressed by

$$F(s) = \frac{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0}{b_n s^n + b_{n-1} s^{n-1} + \cdots + b_1 s + b_0}$$

### 32 1. INTRODUCTION TO MATLAB

(a) write a *MATLAB* function named 'mybode' to evaluate the magnitude and phase angle of the above function for  $s = j\omega$  where  $0 < \omega < \infty$ .

(b) Write a script function that uses 'mybode' to evaluate the magnitude and phase angle of

$$F(s) = \frac{1000(s+1)}{s^3 + 110s^2 + 1000s}$$

over a logarithmic range of  $0.1 \leq \omega \leq 1000$ .

The following function make use of two simple loops to sum up the numerator and denominator terms and returns an array containing magnitudes and phase angles in degree over the specified range.

```

% The function mybode returns the magnitude and phase
% angle of the frequency response transfer function.
% num and den are the numerator and denominator
% coefficients in descending order. w is the frequency
% array in rad/sec.
function[mag, phase] = mybode(num, den, w);
m=length(num); n=length(den);
N=num(m); D=den(n);
s=j*w;
for k=1:m-1
    N=N+num(m-k)*s.^k;
end
for k=1:n-1
    D=D+den(n-k)*s.^k;
end
H=N./D;
mag=abs(H);
phase=angle(H)*180/pi;
    
```

(b) The following script file uses 'mybode' to evaluate and plot the magnitude and phase angle of the function given (b) over the specified range of frequency.

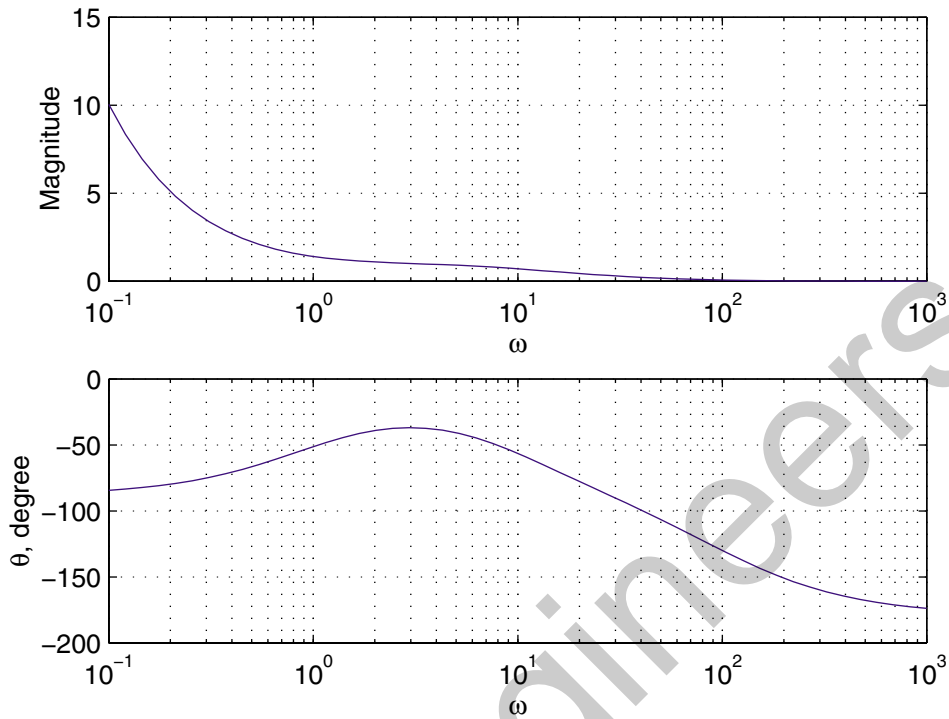
```

num=[1000 1000], den=[1 110 1000 1];
w=logspace(-1, 3); % logarithmic range from 0.1 to 1000
[mag, phase]=mybode(num, den, w);
subplot(2,1,1), semilogx(w, mag), grid
xlabel('\omega'), ylabel('Magnitude')
subplot(2,1,2), semilogx(w, phase), grid
xlabel('\omega'), ylabel('\theta, degree')
    
```

The result is shown in Figure 1.8.

The *MATLAB* control system toolbox has a function called **bode** which obtains the frequency response plots of a given transfer function. This function is used in Chapters 4 and 7.





**FIGURE 1.8**  
Magnitude and phase angle plots for Example 1.19.

### Example 1.20

Another example of loops is in the numerical solution of differential equations. Euler's method is the simplest and the least accurate of all numerical methods. Consider the following simple one-dimensional first-order system.

$$a_1 \frac{dx}{dt} + a_0 x = c$$

rewriting in the form

$$\frac{dx}{dt} = \frac{c}{a_1} - \frac{a_0}{a_1} x$$

If at  $t_0$  the value of  $x(t_0)$  denoted by  $x_0$  is given, the subsequent values of  $x$  can be determined by

$$x_{k+1} = x_k + \left. \frac{dx}{dt} \right|_{x_k} \Delta t$$

By applying the above algorithm successively, we can find approximate values of  $x(t)$  at enough points from an initial state  $(t_0, x_0)$  to a final state  $(t_f, x_f)$ . Euler's method assumes that the derivative is constant over the entire interval  $\Delta t$ . An improvement

can be obtained by calculating the derivative at both the beginning and end of intervals, and then using their average value. This algorithm known as the *modified Euler's method* is given by

$$x_{k+1}^c = x_k + \left( \frac{\frac{dx}{dt}\big|_{x_k} + \frac{dx}{dt}\big|_{x_{k+1}^p}}{2} \right) \Delta t$$

Use the above algorithm to find the numerical solution of the following differential equation, and plot the result up to a final time of  $t = 15$  seconds in steps of 0.01 second.

$$4 \frac{dx(t)}{dt} + 2x(t) = 10 \sin 8\pi t \quad \text{given, } x_0 = 1$$

we use the following statements

```

a1 = 4; a0 = 2;
x0 = 1; t0 = 0;                                % Initial state
Dt = 0.01; tf=15;                               % Step size and final time
t=[]; x=[];                                     % Initializing all the arrays
np = (tf -t0)/Dt;
t(1) = t0; x(1) = x0;
for k=1:np
    c=10*sin(pi*t(k));
    t(k+1)=t(k)+Dt;
    Dx1= c/a1 - a0/a1*x(k);                    % Derivative at the beginning
    x(k+1)=x(k)+Dx1*Dt;                        % Predicted value
    Dx2=c/a1 - a0/a1*x(k+1); % Derivative at the end of interval
    Dxavg=(Dx1+Dx2)/2; % Average value of the two derivatives
    x(k+1)=x(k) + Dxavg*Dt;                    % Corrected value
end
plot(t, x), grid
xlabel('t, sec'), ylabel('x(t)')
    
```

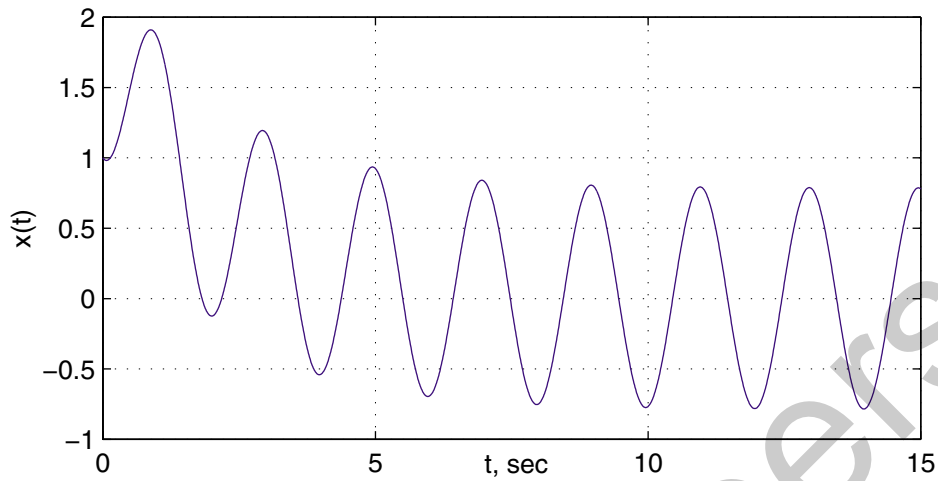
The result is shown in Figure 1.9.

*MATLAB* provides several powerful functions for the numerical solution of differential equations. Two of the functions employing the Runge-Kutta-Fehlberg methods are **ode23** and **ode45**, based on the Fehlberg second- and third-order pair of formulas for medium accuracy and forth- and fifth-order pair for higher accuracy. These functions are described and utilized in Chapter 2.

### Example 1.21

Consider the system of  $n$  equations in  $n$  variables

$$\begin{aligned}
 f_1(x_1, x_2, \dots, x_n) &= c_1 \\
 f_2(x_1, x_2, \dots, x_n) &= c_2 \\
 &\dots\dots\dots \\
 f_n(x_1, x_2, \dots, x_n) &= c_n
 \end{aligned}$$



**FIGURE 1.9**  
Numerical solution of Example 1.20.

The most widely used method for solving simultaneous nonlinear algebraic equations is the Newton-Raphson method. Newton's method is a successive approximation procedure based on an initial estimate of the unknown and the use of Taylor's series expansion and is given by

$$X^{(k+1)} = X^{(k)} + [J^{(k)}]^{-1} \Delta C^{(k)}$$

where  $J^{(k)}$  is a matrix whose elements are the partial derivatives of  $F^{(k)}$  and  $\Delta C^{(k)}$  is

$$\Delta C^{(k)} = C - F^{(k)}$$

The iteration procedure is initiated by assuming an approximate solution for each of the independent variables ( $x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}$ ). At the end of each iteration, the calculated values of all variables are tested against the previous values. If all changes in the variables are within the specified accuracy, a solution has converged, otherwise another iteration must be performed.

Starting with the initial values,  $x_1 = 1$ ,  $x_2 = 1$ , and  $x_3 = 1$ , solve the following system of equations by the Newton-Raphson method.

$$\begin{aligned} x_1^2 - x_2^2 + x_3^2 &= 11 \\ x_1 x_2 + x_2^2 - 3x_3 &= 3 \\ x_1 - x_1 x_3 + x_2 x_3 &= 6 \end{aligned}$$

Taking partial derivatives of the above functions results in the Jacobian matrix

$$J = \begin{bmatrix} 2x_1 & -2x_2 & 2x_3 \\ x_2 & x_1 + 2x_2 & -3 \\ 1 - x_3 & x_3 & -x_1 + x_2 \end{bmatrix}$$

The following statements solve the given system of equations by the Newton-Raphson algorithm

```

Dx=[10;10;10];    %Change in variable is set to a high value
x=[1; 1; 1];      % Initial estimate
C=[11; 3; 6];
iter = 0;          % Iteration counter
while max(abs(Dx))>=.0001 & iter<10; % Test for convergence
iter = iter + 1    % No. of iterations
F = [x(1)^2-x(2)^2+x(3)^2          % Functions
     x(1)*x(2)+x(2)^2-3*x(3)
     x(1)-x(1)*x(3)+x(2)*x(3)];
DC =C - F          % Residuals
J = [2*x(1)  -2*x(2)    2*x(3)    % Jacobian matrix
     x(2)    x(1)+2*x(2)  -3
     1-x(3)  x(3)        -x(1)+x(2)]
Dx=J\DC
x=x+Dx             % Successive solution
end
    
```

The program results for the first iteration are

```

DC =          J =
    10         2    -2     2
     4         1     3    -3
     5         0     1     0

Dx =          x =
    4.750        5.750
    5.000        6.000
    5.250        6.250
    
```

After six iterations, the solution converges to  $x_1 = 2.0000$ ,  $x_2 = 3.0000$ , and  $x_3 = 4.0000$ .

## 1.15 SIMULATION DIAGRAM

The differential equations of a lumped linear network can be written in the form

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}u(t) \end{aligned} \quad (1.1)$$

This system of first-order differential equations is known as the state equation of the system, and  $\mathbf{x}$  is the state vector. One advantage of the state-space method is

that the form lends itself easily to the digital and/or analog computer methods of solution. Further, the state-space method can be easily extended to analysis of nonlinear systems. State equations may be obtained from an  $n$ th-order differential equation or directly from the system model by identifying appropriate state variables.

To illustrate how we select a set of state variables, consider an  $n$ th-order linear plant model described by the differential equation

$$\frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \dots + a_1 \frac{dy}{dt} + a_0 y = u(t) \quad (1.2)$$

where  $y(t)$  is the plant output and  $u(t)$  is its input. A state model for this system is not unique, but depends on the choice of a set of state variables. A useful set of state variables, referred to as *phase variables*, is defined as

$$x_1 = y, \quad x_2 = \dot{y}, \quad x_3 = \ddot{y}, \quad \dots, \quad x_n = y^{n-1}$$

We express  $\dot{x}_k = x_{k+1}$  for  $k = 1, 2, \dots, n-1$ , and then solve for  $d^n y/dt^n$ , and replace  $y$  and its derivatives by the corresponding state variables to give

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}_{n-1} &= x_n \\ \dot{x}_n &= -a_0 x_1 - a_1 x_2 - \dots - a_{n-1} x_n + u(t) \end{aligned} \quad (1.3)$$

or in matrix form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_2 & \dots & -a_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u(t) \quad (1.4)$$

and the output equation is

$$y = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix} \mathbf{x} \quad (1.5)$$

The M-file **ode2phv.m** is developed which converts an  $n$ th-order ordinary differential equation to the state-space phase variable form. **[A, B, C] = ode2phv(ai, k)** returns the matrices **A**, **B**, **C**, where **ai** is a row vector containing coefficients of the equation in descending order, and **k** is the coefficient of the right-hand side.

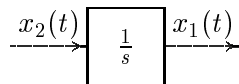
Equation (1.3) indicates that state variables are determined by integrating the corresponding state equation. A diagram known as the simulation diagram can be constructed to model the given differential equations. The basic element of the simulation diagram is the integrator. The first equation in (1.3) is

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

Integrating, we have

$$x_1 = \int x_2 dx$$

The above integral is shown by the following time-domain symbol. The integrating block is identified by symbol  $\frac{1}{s}$ . Adding an integrator for the remaining state variables and completing the last equation in (1.3) via a summing point and feedback paths, a simulation diagram is obtained.



## 1.16 INTRODUCTION TO SIMULINK

*SIMULINK* is an interactive environment for modeling, analyzing, and simulating a wide variety of dynamic systems. *SIMULINK* provides a graphical user interface for constructing block diagram models using “drag-and-drop” operations. A system is configured in terms of block diagram representation from a library of standard components. *SIMULINK* is very easy to learn. A system in block diagram representation is built easily and the simulation results are displayed quickly.

Simulation algorithms and parameters can be changed in the middle of a simulation with intuitive results, thus providing the user with a ready access learning tool for simulating many of the operational problems found in the real world. *SIMULINK* is particularly useful for studying the effects of nonlinearities on the behavior of the system, and as such, it is also an ideal research tool. The key features of *SIMULINK* are

- Interactive simulations with live display.
- A comprehensive block library for creating linear, nonlinear, discrete or hybrid multi-input/output systems.
- Seven integration methods for fixed-step, variable-step, and stiff systems.
- Unlimited hierarchical model structure.
- Scalar and vector connections.
- Mask facility for creating custom blocks and block libraries.

*SIMULINK* provides an open architecture that allows you to extend the simulation environment:

- You can easily perform “what if” analyses by changing model parameters – either interactively or in batch mode – while your simulations are running.
- Creating custom blocks and block libraries with your own icons and user interfaces from *MATLAB*, Fortran, or C code.
- You can generate C code from *SIMULINK* models for embedded applications and for rapid prototyping of control systems.
- You can create hierarchical models by grouping blocks into subsystems. There are no limits on the number of blocks or connections.
- *SIMULINK* provides immediate access to the mathematical, graphical, and programming capabilities of *MATLAB*, you can analyze data, automate procedures, and optimize parameters directly from *SIMULINK*.
- The advanced design and analysis capabilities of the toolboxes can be executed from within a simulation using the mask facility in *SIMULINK*.
- The *SIMULINK* block library can be extended with special-purpose blocksets. The DSP Blockset can be used for DSP algorithm development, while the Fixed-Point Blockset extends *SIMULINK* for modeling and simulating digital control systems and digital filters.

### 1.16.1 SIMULATION PARAMETERS AND SOLVER

You set the simulation parameters and select the solver by choosing **Parameters** from the Simulation menu. *SIMULINK* displays the **Simulation Parameters** dialog box, which uses three “pages” to manage simulation parameters. **Solver**, **Workspace I/O**, and **Diagnostics**.

#### SOLVER PAGE

The Solver page appears when you first choose **Parameters** from the **Simulation menu** or when you select the Solver tab. The Solver page allows you to:

- Set the start and stop times – You can change the start time and stop time for the simulation by entering new values in the Start time and Stop time fields. The default start time is 0.0 seconds and the default stop time is 10.0 seconds.
- Choose the solver and specify solver parameters – The default solver provide accurate and efficient results for most problems. Some solvers may be more efficient than others at solving a particular problem; you can choose between variable-step and fixed-step solvers. Variable-step solvers can modify their step sizes during the simulation. These are **ode45**, **ode23**, **ode113**, **ode15s**, **ode23s**, and **discrete**. The default is **ode45**. For variable-step solvers, you can set the maximum and suggested initial step size parameters. By default, these parameters are automatically determined, indicated by the value auto. For fixed-step solvers, you can choose **ode5**, **ode4**, **ode3**, **ode2**, **ode1**, and **discrete**.

- **Output Options** – The Output options area of the dialog box enables you to control how much output the simulation generates. You can choose from three popup options. These are: Refine output, Produce additional output, and Produce specified output only.

## WORKSPACE I/O PAGE

The Workspace I/O page manages the input from and the output to the *MATLAB* workspace, and allows:

- **Loading input from the workspace** – Input can be specified either as *MATLAB* command or as a matrix for the Import blocks.
- **Saving the output to the workspace** – You can specify return variables by selecting the Time, State, and/or Output check boxes in the Save to workspace area.

## DIAGNOSTICS PAGE

The Diagnostics page allows you to select the level of warning messages displayed during a simulation.

### 1.16.2 THE SIMULATION PARAMETERS DIALOG BOX

Table below summarizes the actions performed by the dialog box buttons, which appear on the bottom of each dialog box page.

Button	Action
Apply	Applies the current parameter values and keeps the dialog box open. During a simulation, the parameter values are applied immediately.
Revert	Changes the parameter values back to the values they had when the Dialog box was most recently opened and applies the parameters.
Help	Displays help text for the dialog box page.
Close	Applies the parameter values and closes the dialog box. During a simulation, the parameter values are applied immediately.

To stop a simulation, choose Stop from the Simulation menu. The keyboard shortcut for stopping a simulation is Ctrl-T. You can suspend a running simulation by choosing Pause from the Simulation menu. When you select Pause, the menu item changes to Continue. You proceed with a suspended simulation by choosing Continue.



### 1.16.3 BLOCK DIAGRAM CONSTRUCTION

At the *MATLAB* prompt, type *SIMULINK*. The *SIMULINK* BLOCK LIBRARY, containing seven icons, and five pull-down menu heads, appears. Each icon contains various components in the titled category. To see the content of each category, double click on its icon. The easy-to-use pull-down menus allow you to create a *SIMULINK* block diagram, or open an existing file, perform the simulation, and make any modifications. Basically, one has to specify the model of the system (state space, discrete, transfer functions, nonlinear ode's, etc), the input (source) to the system, and where the output (sink) of the simulation of the system will go. Generally when building a model, design it first on the paper, then build it using the computer. When you start putting the blocks together into a model, add the blocks to the model window before adding the lines that connect them. This way, you can reduce how often you need to open block libraries. An introduction to *SIMULINK* is presented by constructing the *SIMULINK* diagram for the following examples.

#### MODELING EQUATIONS

Here are some examples that may improve your understanding of how to model equations.

##### Example 1.22

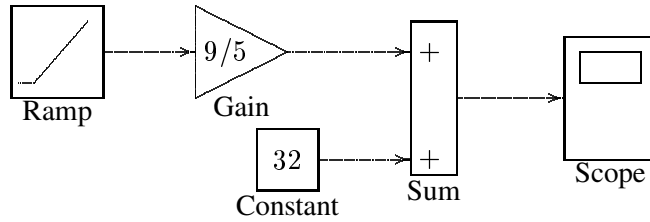
Model the equation that converts Celsius temperature to Fahrenheit. Obtain a display of Fahrenheit-Celsius temperature graph over a range of 0 to 100°C.

$$T_F = \frac{9}{5}T_C + 32 \quad (1.6)$$

First, consider the blocks needed to build the model. These are:

- A ramp block to input the temperature signal, from the source library.
- A constant block, to define the constant of 32, also from the source library.
- A gain block, to multiply the input signal by 9/5, from the Linear library.
- A sum block, to add the two quantities, also from the Linear library.
- A scope block to display the output, from the sink library.

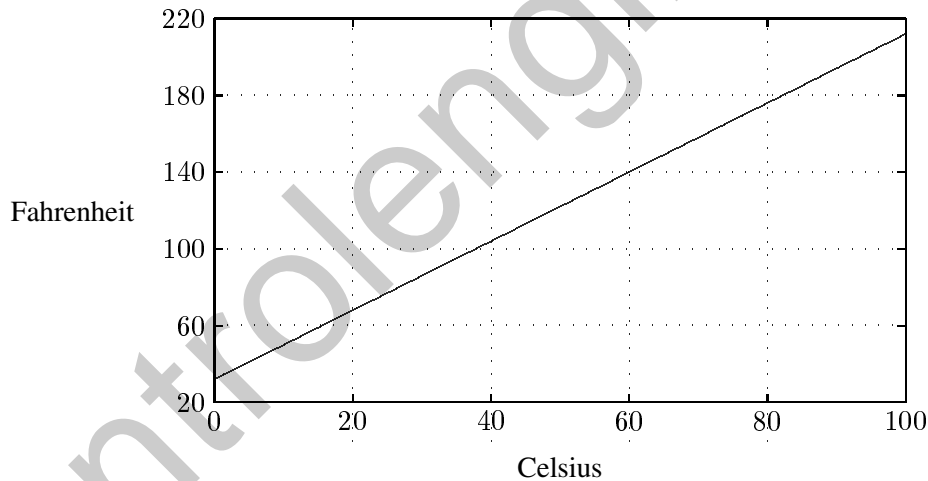
To create a *SIMULINK* block diagram presentation select **new...** from the **File** menu. This provides an untitled blank window for designing and simulating a dynamic system. Copy the above blocks from the block libraries into the new window by depressing the mouse button and dragging. Assign the parameter values to the Gain and Constant blocks by opening (double clicking on) each block and entering the appropriate value. Then click on the **close** button to apply the value and close the dialog box. The next step is to connect these icons together by drawing lines connecting the



**FIGURE 1.10**  
Simulink diagram for the system of Example 1.22.

icons using the left mouse button (hold the button down and drag the mouse to draw a line). You should now have the *SIMULINK* block diagram as shown in Figure 1.10.

The Ramp block inputs Celsius temperature. Open this block, set the Slope to 1, Start time to 0, and the Initial output to 0. The Gain block multiplies that temperature by the constant 9/5. The sum block adds the value 32 to the result and outputs the Fahrenheit temperature. Pull down the Simulation dialog box and select Parameters. Set the Start time to zero and the Stop Time to 100. Pull down the **File** menu and use **Save** to save the model under **simexa22** Start the simulation. Double click on the Scope, click on the **Auto Scale**, the result is displayed as shown in Figure 1.11.



**FIGURE 1.11**  
Fahrenheit-Celsius temperature graph for Example 1.22.

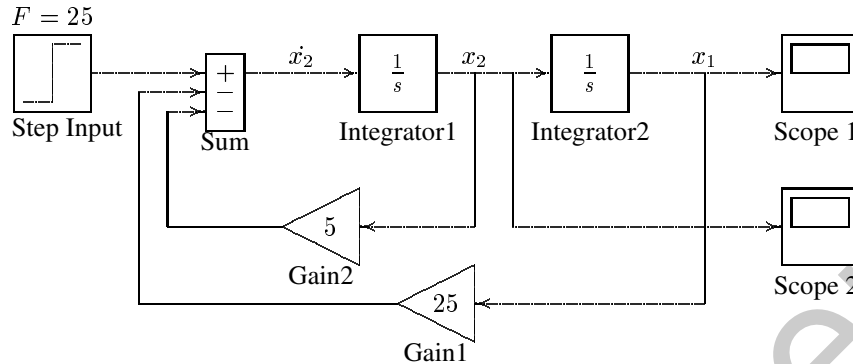
### Example 1.23

Construct a simulation diagram for the state equation described by

$$\begin{aligned}\frac{dx_1}{dt} &= x_2 \\ \frac{dx_2}{dt} &= \frac{1}{M}[f(t) - Bx_2 - Kx_1]\end{aligned}$$

where  $M = 1$  kg,  $B = 5$  N/m/sec,  $K = 25$  N/m, and  $f(t) = 25u(t)$ .

The simulation diagram is drawn from the above equations by inspection and is shown in Figure 1.12.



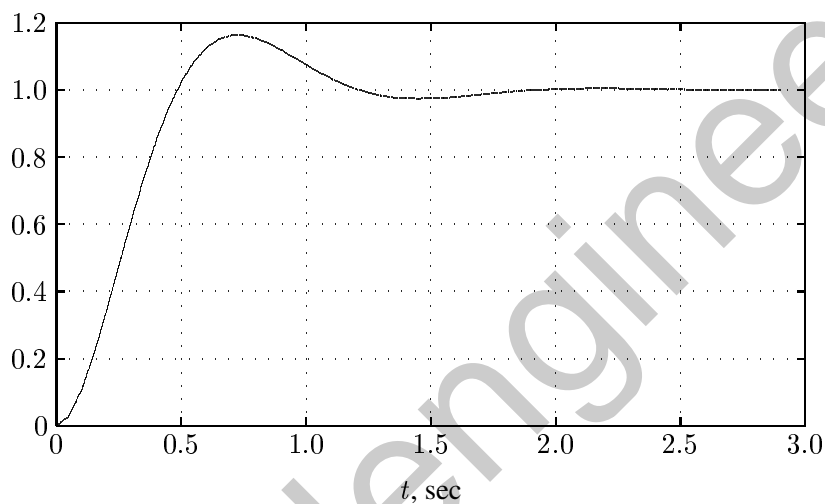
**FIGURE 1.12**  
Simulink diagram for the system of Example 1.23.

To create a *SIMULINK* block diagram presentation select **new...** from the **File** menu. This provides an untitled blank window for designing and simulating a dynamic system. You can copy blocks from within any of the seven block libraries or other previously opened windows into the new window by depressing the mouse button and dragging. Open the **Source Library** and drag the Step Input block to your window. Double click on Step Input to open its dialog box. Set the step time to 0, and set the Initial Value to 0 and the Final Value to 25 to represent the step input. Open the **Linear Library** and drag the **Sum** block to the right of the Step Input block. Open the Sum dialog box and enter + - - under List of Signs. Using the left mouse button, click and drag from the Step output port to the Summing block input port to connect them. Drag a copy of the Integrator block from the **Linear Library** and connect it to the output port of the Sum block. Click on the Integrator block once to highlight it. Use the Edit command from the menu bar to copy and paste a second Integrator. Next drag a copy of the Gain block from the **Linear Library**. Highlight the Gain block, and from the pull-down **Options** menu, click on the Flip Horizontal to rotate the Gain block by 180°. Double click on Gain block to open its dialog box and set the gain to 5. Make a copy of this block and set its gain to 25. Connect the output ports of the Gain blocks to the Sum block and their input ports to the locations shown in Figure 1.12. Finally, get two Auto-Scale Graphs from the **Sink Library**, and connect them to the output of each Integrator. Before starting simulation, you must set the simulation parameters. Pull down the **Simulation** dialog box and select **Parameters**. Set the Start Time to zero, the Stop Time to 3, and for a more accurate integration, set the Maximum Step Size to 0.1. Leave the other parameters at their default values. Press OK to close the dialog box.

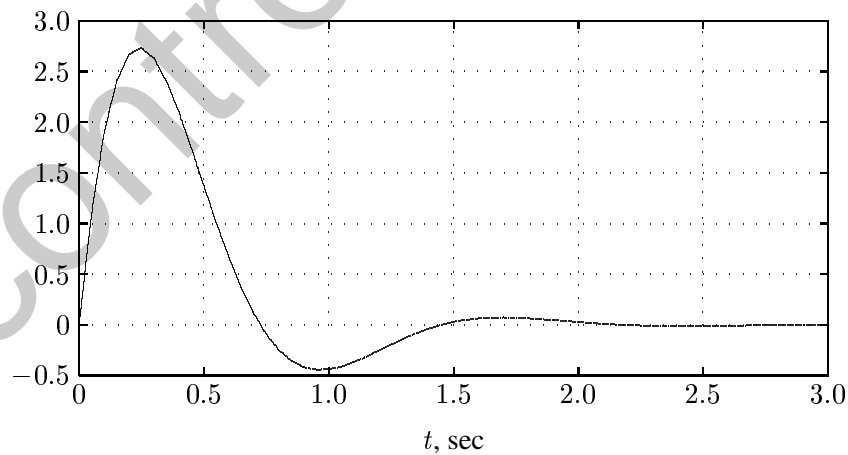
If you don't like some aspect of the diagram, you can change it in a variety of ways. You can move any of the icons by clicking on its center and dragging. You can move any of the lines by clicking on one of its corners and dragging. You can change the size and the shape of any of the icons by clicking and dragging on its corners. You can remove any line or icon by clicking on it to select it and using the

**cut** command from the **edit** menu. You should now have exactly the same system as shown in Figure 1.12. Pull down the **File** menu and use **Save as** to save the model under a file name **simexa23**. Start the simulation. *SIMULINK* will create the Figure windows and display the system responses. To see the second Figure window, click and drag the first one to a new location. The simulation results are shown in Figures 1.13 and 1.14.

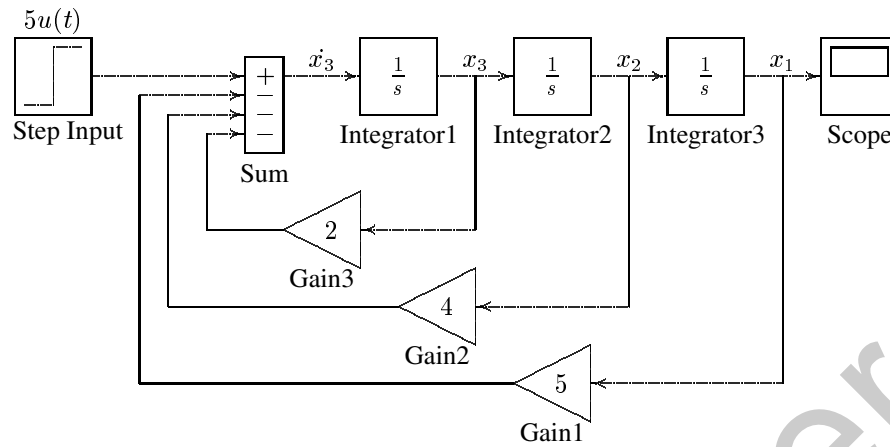
*SIMULINK* enables you to construct and simulate many complex systems, such as control systems modeled by block diagram with transfer functions including the effect of nonlinearities. In addition, *SIMULINK* provides a number of built-in state variable models and subsystems that can be utilized easily.



**FIGURE 1.13**  
Displacement response of the system described in Example 1.23.



**FIGURE 1.14**  
Velocity response of the system described in Example 1.23.



**FIGURE 1.15**  
Simulation diagram for the system of Example 1.24.

### Example 1.24

Consider the system defined by

$$2 \frac{d^3 y}{dt^3} + 4 \frac{d^2 y}{dt^2} + 8 \frac{dy}{dt} + 10y = 10u(t)$$

We have a third-order system; thus there are three state variables. Let us choose the state variables as

$$\begin{aligned} x_1 &= y \\ x_2 &= \dot{y} \\ x_3 &= \ddot{y} \end{aligned}$$

Then we obtain

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ \dot{x}_3 &= -5x_1 - 4x_2 - 2x_3 + 5u(t) \end{aligned}$$

The last of these three equations was obtained by solving the original differential equation for the highest derivative term  $\ddot{y}$  and then substituting  $y = x_1$ ,  $\dot{y} = x_2$ , and  $\ddot{y} = x_3$  into the resulting equation. Using matrix notation, the state equation is

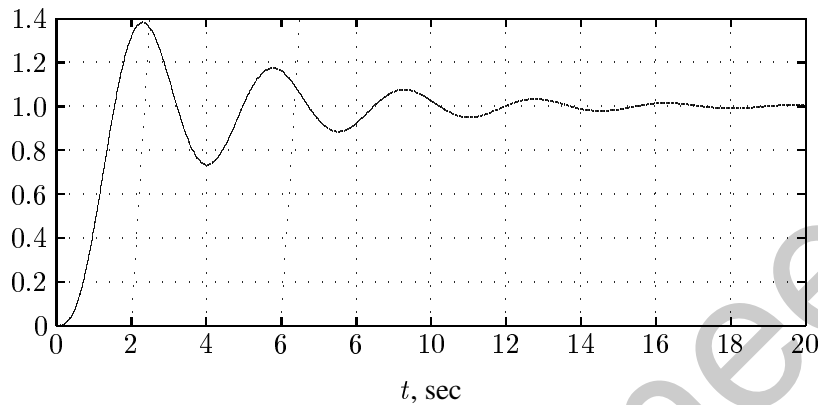
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -5 & -4 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} u(t)$$

and the output equation is given by

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{x}$$

The simulation diagram is obtained from the system differential equations and is given in Figure 1.15.

A *SIMULINK* Block diagram is constructed and saved as **simexa24**. The simulation response is shown in Figure 1.16.

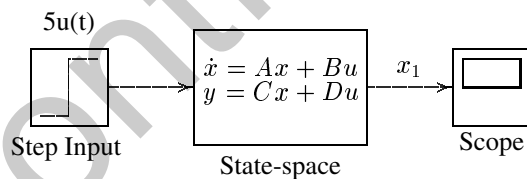


**FIGURE 1.16**  
Simulation result for the system in Example 1.24.

### Example 1.25

Use the **state-space** model to simulate the state and output equations described in Example 1.24.

The **State-Space** model provides a dialog box where the  $A$ ,  $B$ ,  $C$ , and  $D$  matrices can be entered in *MATLAB* matrix notation, or by variables defined in Workspace. A *SIMULINK* diagram using the **State-Space** model is constructed as shown in Figure 1.17, and saved as **simexa25**.



**FIGURE 1.17**  
State-space model for system in Example 1.25.

Note that in this example, the output is given by  $y = x_1$ , and we define  $C$  as  $C = [1 \ 0 \ 0]$ . If it is desired to access all the states, then we can define  $C$  as an identity matrix, in this case a third order, i.e.,  $C = \text{eye}(3)$ , and  $D$  as  $D = \text{zeros}(3, 1)$ . The output is a vector of state variables. A **DeMux** block may be added to produce individual states for graphing separately.

### 1.16.4 USING THE TO WORKSPACE BLOCK

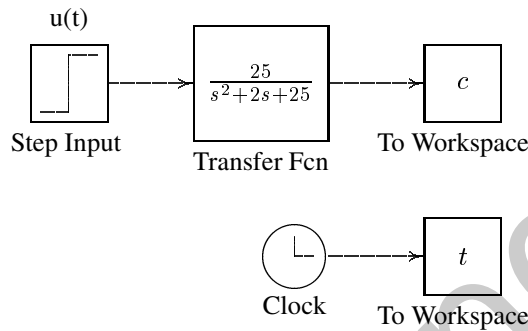
The To Workspace block can be used to return output trajectories to the *MATLAB* Workspace. Example 1.26 illustrates this use.

#### Example 1.26

Obtain the step response of the following transfer function, and send the result to the *MATLAB* Workspace.

$$\frac{C(s)}{R(s)} = \frac{25}{s^2 + 2s + 25}$$

where  $r(t)$  is a unit step function. The *SIMULINK* block diagram is constructed and saved in a file named **simexa26** as shown in Figure 1.18.



**FIGURE 1.18**  
Simulink model for system in Example 1.26.

The To Workspace block can accept a vector input, with each input element's trajectories stored as a column vector in the resulting workspace variable. To specify the variables open the To Workspace block and for the variable name enter  $c$ . The time vector is stored by feeding a Clock block into To Workspace block. For this block variable name specify  $t$ . The vectors  $c$  and  $t$  are returned to *MATLAB* Workspace upon simulation.

### 1.16.5 LINEAR STATE-SPACE MODEL FROM SIMULINK DIAGRAM

*SIMULINK* provides the **linmod**, and **dlinmod** functions to extract linear models from the block diagram model in the form of the state-space matrices  $A$ ,  $B$ ,  $C$ , and  $D$ . State-space matrices describe the linear input-output relationship as

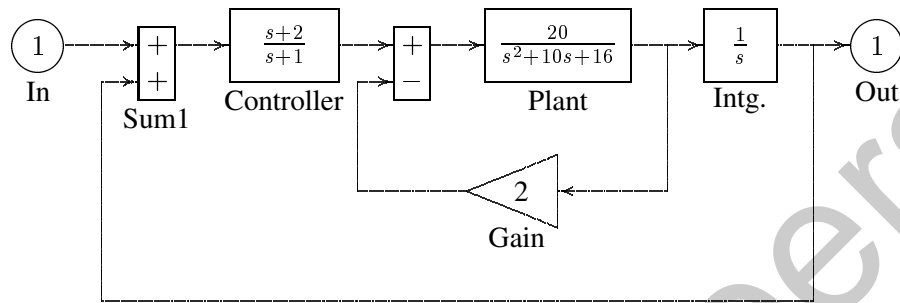
$$\dot{x}(t) = Ax(t) + Bu(t) \quad (1.7)$$

$$y(t) = Cx(t) + Du(t) \quad (1.8)$$

The following Example illustrates the use of **linmod** function. The input and outputs of the *SIMULINK* diagram must be defined using **Inport** and **Outport** blocks in place of the **Source** and **Sink** blocks.

### Example 1.27

Obtain the state-space model for the system represented by the block diagram shown in Figure 1.19. The model is saved with a filename **simexa27**. Run the simulation and to extract the linear model of this *SIMULINK* system, in the Command Window, enter the command



**FIGURE 1.19**  
Simulink model for system in Example 1.27.

```
[A,B,C,D] = linmod('simexa27')
```

The result is

```

A =      0      0      0      20
     -1     -1      0      0
     -1      1     -10     -56
      0      0      1      0
C =
     1      0      0      0
D =
     0
    
```

In order to obtain the transfer function of the system from the state-space model, we use the command

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

the result is

```

num =
    0.0000    0.0000    0.0000   20.0000   40.0000
den =
    1.0000   11.0000   66.0000   76.0000   40.0000
    
```



Thus, the transfer function model is

$$T(s) = \frac{20s + 40}{s^4 + 11s^3 + 66s^2 + 76s + 40}$$

Once the data is in the state-space form, or converted to a transfer function model, you can apply functions in Control System Toolbox for further analysis:

- Bode phase and magnitude frequency plot:

`bode(A, B, C, D)` or `bode(num, den)`

- Linearized time response:

`step(A, B, C, D)` or `step(num, den)`  
`lsim(A, B, C, D)` or `lsim(num, den)`  
`impulse(A, B, C, D)` or `impulse(num, den)`

#### 1.16.6 SUBSYSTEMS AND MASKING

*SIMULINK* subsystems, provide a capability within *SIMULINK* similar to subprograms in traditional programming languages.

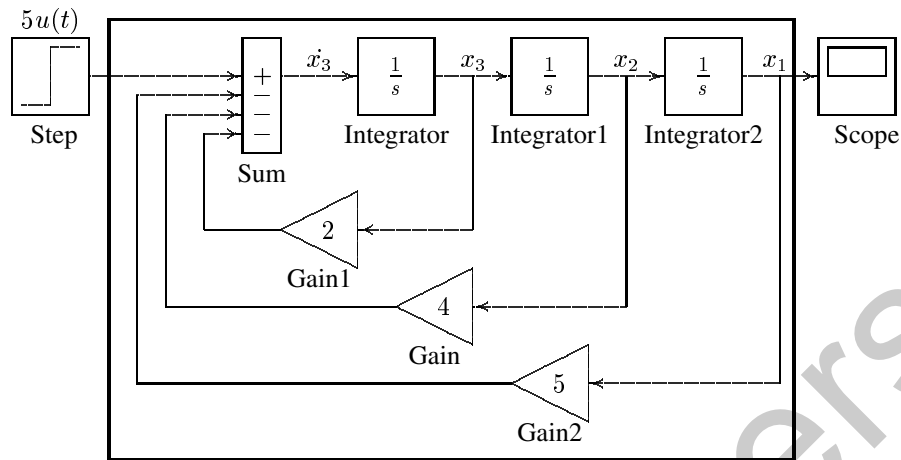
Masking is a powerful *SIMULINK* feature that enables you to customize the dialog box and icon for a block or subsystem. With masking, you can simplify the use of your model by replacing many dialog boxes in a subsystem with a single one.

##### Example 1.28

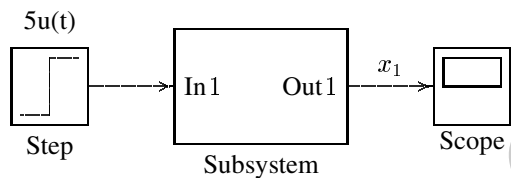
To encapsulate a portion of an existing *SIMULINK* model into a subsystem, consider the *SIMULINK* model of Example 1.24 shown in Figure 1.20, and proceed as follows:

1. Select all the blocks and signal lines to be included in the subsystem with the bounding box as shown.
2. Choose Edit and select Create Subsystem from the model window menu bar. *SIMULINK* will replace the select blocks with a subsystem block that has an input port for each signal entering the new subsystem and an output port for each signal leaving the new subsystem. *SIMULINK* will assign default names to the input and output ports.

To mask a block, select the block, then choose Create Mask from the Edit menu. The Mask Editor appears. The Mask Editor consists of three pages, each handling a different aspect of the mask.



**FIGURE 1.20**  
Simulation diagram for the system of Example 1.24.



**FIGURE 1.21**  
Simulation diagram for the system of Example 1.24.

- The Initialization page enables you to define and describe mask dialog box parameter prompts, name the variables associated with the parameters, and specify initialization commands.
- The Icon page enables you to define the block icon.
- The Documentation page enables you to define the mask type, and specify the block description and the block help.

In this example for icon the system transfer function is entered with command

```
dpoly([10], [2 4 8 10])
```

A short description of the system and relevant help topics can be entered in the Documentation page. The subsystem block is saved in a file named **simexa29**.

---

## CHAPTER 2

---

# MATHEMATICAL MODELS OF SYSTEMS

### 2.1 Differential Equations of Physical Systems

The dynamic performance of physical systems is obtained by utilizing the physical laws of mechanical, electrical, fluid and thermodynamic systems. We generally model physical systems with linear differential equations with constant coefficients when possible. Other models can be derived from more general differential equations.

### 2.2 Numerical Solution

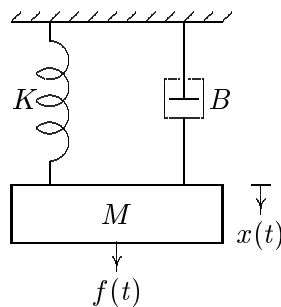
Analytical solutions of linear time-invariant equations are obtained through the Laplace transform and its inversion. There are other techniques which use the state transition matrix  $\phi(t)$  to provide a solution. These analytical methods are normally restricted to linear differential equations with constant coefficients. Numerical techniques solve differential equations directly in the time domain; they apply not only to linear time-invariant but also to nonlinear and time varying differential equations. The value of the function obtained at any step is an approximation of the value which would have been obtained analytically, whereas the analytical solution is exact. However, an analytical solution may be difficult, time consuming or even impossible to find.

*MATLAB* provides two functions for numerical solutions of differential equations employing the Runge-Kutta method. These are **ode23** and **ode45**, based on the

Fehlberg second and third order pair of formulas for medium accuracy and fourth and fifth order pair for high accuracy. The  $n$ th-order differential equation must be transformed into  $n$  first order differential equations and must be placed in an M-file that returns the state derivatives of the equations. The following examples demonstrate the use of these functions.

### Example 2.1

Consider the simple mechanical system of Figure 2.1. Three forces influence the motion of the mass, namely, the applied force, the frictional force, and the spring force.



**FIGURE 2.1**  
Mechanical translational system.

Applying Newton's law of motion, the force equation of the system is

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

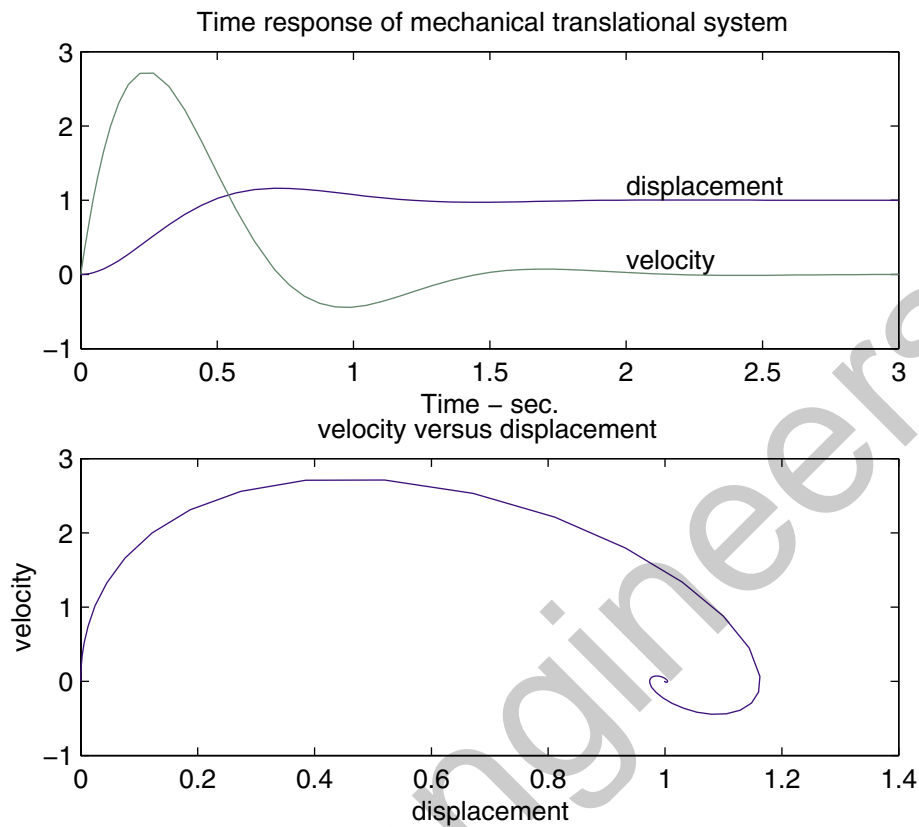
Let  $x_1 = x$  and  $x_2 = \frac{dx}{dt}$ , then

$$\begin{aligned} \frac{dx_1}{dt} &= x_2 \\ \frac{dx_2}{dt} &= \frac{1}{M} [f(t) - Bx_2 - Kx_1] \end{aligned}$$

With the system initially at rest, a force of 25 Newton is applied at time  $t = 0$ . Assume that the mass  $M = 1$  Kg, frictional coefficient  $B = 5$  N/m/sec., and the spring constant  $K = 25$  N/m. The above equations are defined in an M-file **mechsys.m** as follows:

```
function xdot = mechsys(t,x); % returns the state derivatives
F = 25; % Step input
M = 1; B = 5; K = 25;
xdot = [x(2) ; 1/M*( F - B*x(2) - K*x(1) ) ];
```

The following M-file, **ch2ex01.m** uses **ode23** to simulate the system over an interval of 0 to 3 sec., with zero initial conditions.



**FIGURE 2.2**  
Response of the mechanical system of Example 2.1.

```

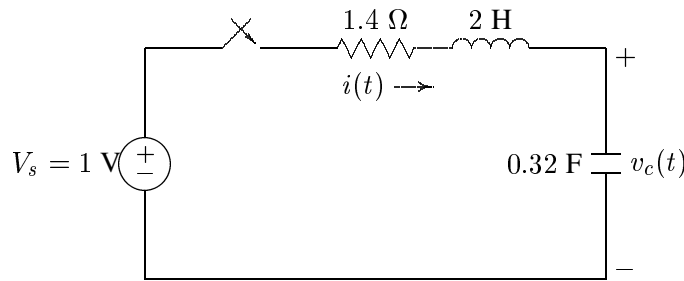
tspan = [0, 3] ; % time interval
x0 = [0, 0]; % initial conditions
[t,x] = ode23('mechsys', tspan, x0);
subplot(2,1,1), plot(t,x)
title('Time response of mechanical translational system')
xlabel('Time - sec.')
text(2,1.2,'displacement')
text(2,.2,'velocity')

d= x(:,1); v = x(:,2);
subplot(2,1,2), plot(d, v)
title('velocity versus displacement ')
xlabel('displacement')
ylabel('velocity')
subplot(111)
    
```

Results of the simulation are shown in Figure 2.2.

### Example 2.2

The circuit elements in Figure 2.3 are  $R = 1.4\Omega$ ,  $L = 2\text{H}$ , and  $C = 0.32\text{F}$ , the initial inductor current is zero, and the initial capacitor voltage is .5 volts. A step voltage of 1 volt is applied at time  $t = 0$ . Determine  $i(t)$  and  $v(t)$  over the range  $0 < t < 15$  sec. Also, obtain a plot of current versus capacitor voltage.



**FIGURE 2.3**  
RLC circuit for time-domain solution example.

Applying KVL

$$Ri + L \frac{di}{dt} + v_c = V_s$$

and

$$i = C \frac{dv_c}{dt}$$

Let

$$x_1 = v_c$$

and

$$x_2 = i$$

then

$$\dot{x}_1 = \frac{1}{C}x_2$$

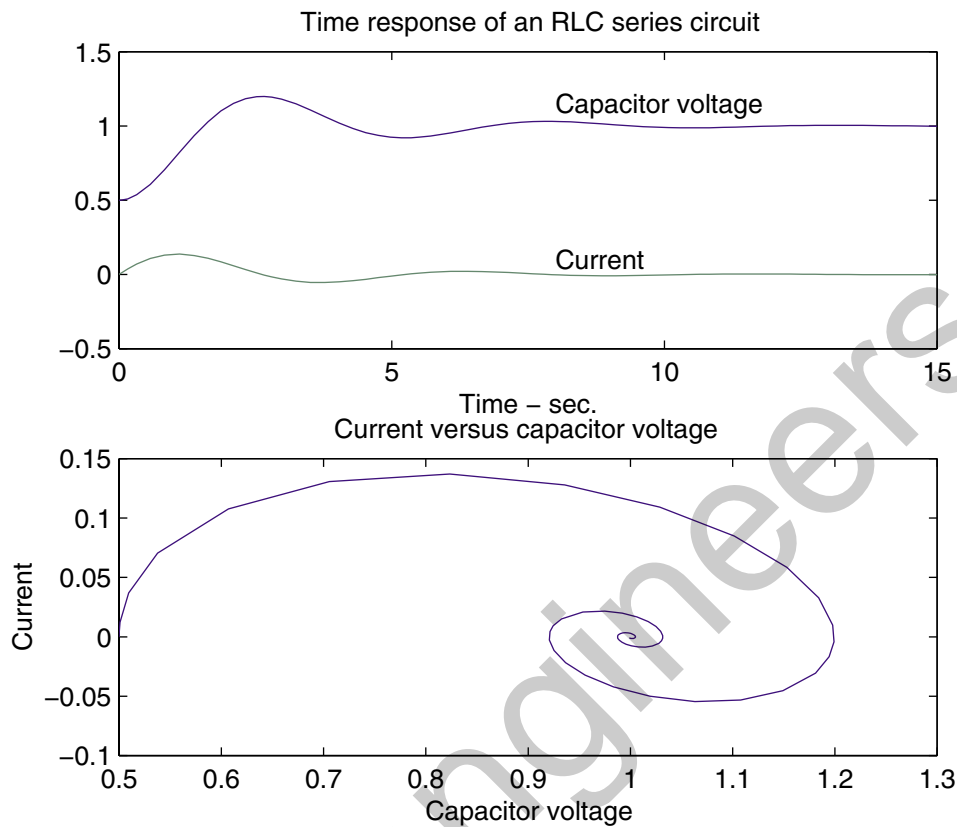
and

$$\dot{x}_2 = \frac{1}{L}(V_s - x_1 - Rx_2)$$

The above equations are defined in an M-file **electsys.m** as follows:

```

function xdot = electsys(t,x);
                                % returns the state derivatives
                                % Step input
V = 1;
R = 1.4; L = 2; C = 0.32;
xdot = [x(2)/C ; 1/L*( V - x(1) - R*x(2) ) ];
    
```



**FIGURE 2.4**  
Response of the series RLC circuit of Example 2.2.

The following M-file, **ch2ex02.m**, uses **ode23** to simulate the system over an interval of 0 to 15 sec.

```

x0 = [0.5, 0];           % initial conditions
tspan=[0, 15];          % time interval
[t,x] = ode23('electsys',tspan, x0);
subplot(2, 1, 1),plot(t,x)
title('Time response of an RLC series circuit')
xlabel('Time - sec.')
text(8,1.15, 'Capacitor voltage')
text(8, .1, 'Current')
vc= x(:,1);   i = x(:,2);
subplot(2, 1, 2),plot(vc, i)
title('Current versus capacitor voltage ')
xlabel('Capacitor voltage')
ylabel('Current'), subplot(111)
    
```

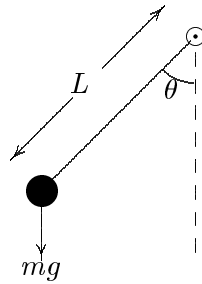
Results of the simulation are shown in Figure 2.4.

## 2.3 Nonlinear Systems

A great majority of physical systems are linear within some range of the variables. However, all systems ultimately become nonlinear as the ranges are increased without limit. For the nonlinear systems, the principle of superposition does not apply. **ode23** and **ode45** simplify the task of solving a set of nonlinear differential equations as demonstrated in Example 2.3.

### Example 2.3

Consider the simple pendulum illustrated in Figure 2.5 where a weight of  $W = mg$  kg is hung from a support by a weightless rod of length  $L$  meters. While usually approximated by a linear differential equation, the system really is nonlinear and includes viscous damping with a damping coefficient of  $B$  kg/m/sec.



**FIGURE 2.5**  
Pendulum oscillator.

If  $\theta$  in radians is the angle of deflection of the rod, the velocity of the weight at the end will be  $L\dot{\theta}$  and the tangential force acting to increase the angle  $\theta$  can be written:

$$F_T = -W \sin \theta - BL\dot{\theta}$$

From Newton's law

$$F_T = mL\ddot{\theta}$$

Combining the two equations for the force, we get:

$$mL\ddot{\theta} + BL\dot{\theta} + W \sin \theta = 0$$

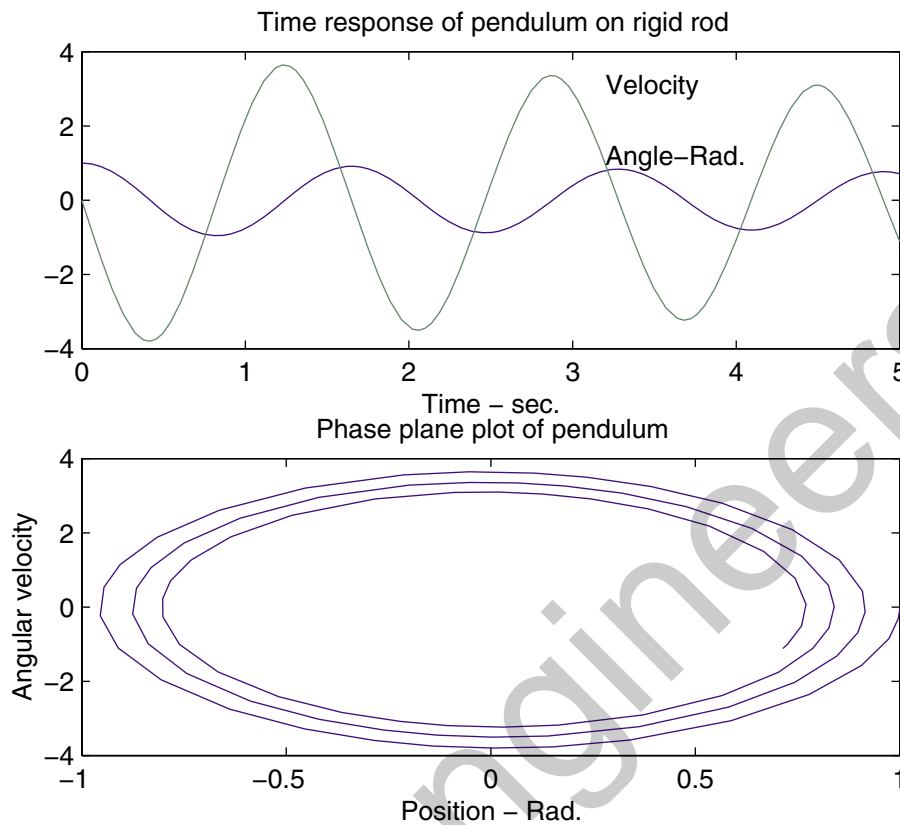
Let  $x_1 = \theta$  and  $x_2 = \dot{\theta}$  (angular velocity), then

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

$$\dot{x}_2 = -\frac{B}{m}x_2 - \frac{W}{mL} \sin x_1$$

The above equations are defined in an M-file **pendulum.m** as follows:





**FIGURE 2.6**  
Response of the pendulum described in Example 2.3.

```
function xdot = pendulum(t,x);%returns the state derivatives
W = 2; L = .6; B = 0.02; g = 9.81; m = W/g;
xdot = [x(2) ; -B/m*x(2)-W/(m*L)*sin(x(1)) ];
```

The following M-file, **ch2ex03.m**, uses **ode23** to simulate the system over an interval of 0 to 5 sec.

```
tspan = [0, 5]; % time interval
x0 = [1, 0]; % initial conditions
[t,x] = ode23('pendulum', tspan, x0);
subplot(2,1,1),plot(t,x)
title('Time response of pendulum on rigid rod')
xlabel('Time - sec.')
text(3.2,3.1,'Velocity'), text(3.2,1.2,'Angle-Rad.')
th= x(:,1); w = x(:,2);
subplot(2,1,2),plot(th, w)
title('Phase plane plot of pendulum')
xlabel('Position - Rad.'), ylabel('Angular velocity')
```

Results of the simulation are shown in Figure 2.6.

## 2.4 Linearization

Nonlinear systems are often linearized assuming small signal conditions. The nonlinear differential equation describing the motion of the pendulum in Example 2.3 can be linearized if the initial angle of deflection is small. When  $\theta = \theta_0 + \Delta\theta$ , the pendulum equation can be written as

$$mL(\ddot{\theta} + \Delta\ddot{\theta}) + BL(\dot{\theta} + \Delta\dot{\theta}) + W \sin(\theta + \Delta\theta) = 0 \quad (2.1)$$

For small  $\Delta\theta$  assuming  $\sin \Delta\theta \simeq 0$ ,  $\cos \Delta\theta \simeq 1$  and expanding the sine term yields the following linear differential equation.

$$mL\Delta\ddot{\theta} + BL\Delta\dot{\theta} + W\Delta\theta = 0 \quad (2.2)$$

It is left as an exercise to show that the above linearized equation will yield approximately the same response as long as  $\Delta\theta$  is small.

## 2.5 Transfer Function

The transfer function of a linear, time-invariant, differential equation system is defined as the ratio of the Laplace transform of the output variable to the Laplace transform of the input variable, with all initial conditions assumed to be zero. Although the transfer function can be used only for linear systems, it yields more intuitive information than the differential equation. The characteristic equation is obtained by setting the denominator polynomials of the transfer function to zero. The roots of the denominator are the system poles, and the roots of the numerator are the system zeros. The system transfer function can then be specified to within a constant by specifying the system poles and zeros. The constant, usually denoted by  $K$ , is the system gain factor. The transfer function model enables us to change system parameters and rapidly sense the effect of these changes on the system response. The transfer function is also useful in modeling the interconnection of subsystems by forming a block diagram representation. The time response of a system is obtained by the inverse transform of the  $s$ -domain response. This usually requires expansion of the rational function using partial fractions.

In this section, several examples are presented to demonstrate the use of *MATLAB* in finding the roots of the characteristic equation, poles and zeros of a transfer function, partial fraction expansion, and transformation of poles and zeros to transfer function.

The Control System Toolbox function `sys = tf(num, den)` creates a continuous-time transfer function. The output `sys` is a `tf` object. For SISO models, `num` and `den` are row vectors listing the numerator and denominator coefficients in descending powers of  $s$ . For example, the commands

```
num=[1 4]; den=[1 2 10];
sys=tf(num, den)
```

results in

Transfer function:

$$\frac{s + 4}{s^2 + 2s + 10}$$

### 2.5.1 Polynomial Roots and Characteristic Polynomial

If  $p$  is a row vector containing the coefficients of a polynomial, **roots(p)** returns a column vector whose elements are the roots of the polynomial. If  $r$  is a column vector containing the roots of a polynomial, **poly(r)** returns a row vector whose elements are the coefficients of the polynomial.

#### Example 2.4

Find the roots of the following polynomial.

$$s^6 + 9s^5 + 31.25s^4 + 61.25s^3 + 67.75s^2 + 14.75s + 15$$

The polynomial coefficients are entered in a row vector in descending powers. The roots are found using **roots**.

```
p = [ 1 9 31.25 61.25 67.75 14.75 15 ]
r = roots(p)
```

The polynomial roots are obtained in column vector

```
r =
-4.0000
-3.0000
-1.0000 + 2.0000i
-1.0000 - 2.0000i
0.0000 + 0.5000i
0.0000 - 0.5000i
```

#### Example 2.5

The roots of a polynomial are  $-1, -2, -3 \pm j4$ . Determine the polynomial equation.

In order to enter a complex number, it is first necessary to generate a complex unit. The roots are then entered in a column vector. The polynomial equation is obtained using **poly** as follows:

```
i = sqrt(-1)
r = [-1 -2 -3+4*i -3-4*i ]
p = poly(r)
```

The coefficients of the polynomial equation are obtained in a row vector.

```
p =
1 9 45 87 50
```

Therefore, the polynomial equation is

$$s^4 + 9s^3 + 45s^2 + 87s + 50 = 0$$

### Example 2.6

Determine the roots of the characteristic equation of the following matrix.

$$A = \begin{bmatrix} 0 & 1 & -1 \\ -6 & -11 & 6 \\ -6 & -11 & 5 \end{bmatrix}$$

The characteristic equation of the matrix is found by **poly**, and the roots of this equation are found by **roots**.

```
A = [ 0  1  -1; -6 -11  6; -6 -11  5];
p = poly(A)
r = roots(p)
```

The result is as follows:

```
p =
    1.0000    6.0000   11.0000    6.0000
r =
   -3.0000
   -2.0000
   -1.0000
```

### 2.5.2 Poles and Zeros of a Transfer Function

**tf2zp** finds the zeros, poles and gains of a transfer function.

### Example 2.7

Find the poles and zeros of the following transfer function:

$$H(s) = \frac{s^3 + 11s^2 + 30s}{s^4 + 9s^3 + 45s^2 + 87s + 50}$$

```
num = [ 1  11  30  0];
den = [ 1  9  45  87  50];
[z,p,k] = tf2zp(num,den)
```

The zeros, poles and gains are:

```
z =
   -6.0000
   -5.0000
    0.0000
    inf
p =
   -3.0000    +4.0000i
   -3.0000    -4.0000i
   -2.0000
   -1.0000
k =
    1
```

Therefore

$$H(s) = \frac{s(s+5)(s+6)}{(s+1)(s+2)(s+3+j4)(s+3-j4)}$$

**zp2tf** forms transfer function polynomials from the zeros, poles and gains of systems.

### Example 2.8

A system has zeros at  $-6, -5, 0$ , poles at  $-3 \pm j4, -2, -1$ , and a gain of 1. Determine the system transfer function.

```
z = [-6; -5; 0]; k=1;
i = sqrt(-1);
p = [-3+4*i; -3-4*i; -2; -1];
[num, den] = zp2tf(z,p,k)
```

The above program results in

```
num =
    1    11    30    0
den =
    1     9    45    87    50
```

which yields the following transfer function

$$H(s) = \frac{s^3 + 11s^2 + 30s}{s^4 + 9s^3 + 45s^2 + 87s + 50}$$

### 2.5.3 Partial-Fraction Expansion

**[r,p,k] = residue[b,a]** finds the residues, poles, and direct terms of a partial fraction expansion of the ratio of two polynomials

$$\frac{P(s)}{Q(s)} = \frac{b_ms^m + b_{m-1}s^{m-1} + \dots + b_1s + b_0}{a_ns^n + a_{n-1}s^{n-1} + \dots + a_1s + a_0} \quad (2.3)$$

Vectors **b** and **a** specify the coefficients of the polynomials in descending powers of  $s$ . The residues are returned in column vector **r**, the pole locations in column vector **p**, and the direct terms in row vector **k**.

### Example 2.9

Determine the partial fraction expansion for

$$F(s) = \frac{2s^3 + 9s + 1}{s^3 + s^2 + 4s + 4}$$

## 62 2. Mathematical Models of Systems

```

num = [ 2  0  9  1];
den = [ 1  1  4  4];
[res, poles ,k] = residue(num, den)
    
```

The result is as follows

```

res  =
    0.0000    -0.2500i
    0.0000    +0.2500i
   -2.0000
poles =
    0.0000    +2.0000i
    0.0000    -2.0000i
   -1.0000
K  =
    2.0000
    
```

Therefore the partial fraction expansion is

$$2 + \frac{-2}{s+1} + \frac{j0.25}{s+j2} + \frac{-j0.25}{s-j2} = 2 + \frac{-2}{s+1} + \frac{1}{s^2+4}$$

**[num, den] = residue(res, poles, K)** converts the partial fraction expansion back to the polynomial  $P(s)/Q(s)$ .

# CHAPTER 3

## STATE-SPACE REPRESENTATION

The differential equations of a lumped linear network can be written in the form

$$\dot{\mathbf{x}}(\mathbf{t}) = \mathbf{A}\mathbf{x}(\mathbf{t}) + \mathbf{B}\mathbf{u}(\mathbf{t}) \quad (3.1)$$

This system of first-order differential equations is known as the state equation of the system and  $\mathbf{x}$  is the state vector. One advantage of the state-space method is that the form lends itself easily to the digital and/or analog computer methods of solution. Further, the state-space method can be easily extended to analysis of nonlinear systems. State equations may be obtained from an  $n$ th-order differential equation or directly from the system model by identifying appropriate state variables.

### 3.1 State-Variable Modeling

To illustrate how we select a set of state variables, consider an  $n$ th-order linear plant model described by the differential equation

$$\frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \dots + a_1 \frac{dy}{dt} + a_0 y = u(t) \quad (3.2)$$

where  $y(t)$  is the plant output and  $u(t)$  is its input. A state model for this system is not unique but depends on the choice of a set of state variables. A useful set of state variables, referred to as *phase variables*, is defined as

$$x_1 = y, \quad x_2 = \dot{y}, \quad x_3 = \ddot{y}, \quad \dots, \quad x_n = y^{n-1}$$

### 64 3. State-Space Representation

We express  $\dot{x}_k = x_{k+1}$  for  $k = 1, 2, \dots, n-1$ , and then solve for  $d^n y/dt^n$  and replace  $y$  and its derivatives by the corresponding state variables to give

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}_{n-1} &= x_n \\ \dot{x}_n &= -a_0 x_1 - a_1 x_2 - \dots - a_{n-1} x_n + u(t)\end{aligned}\quad (3.3)$$

or in matrix form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_2 & \dots & -a_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u(t)\quad (3.4)$$

and the output equation is

$$y = [1 \quad 0 \quad 0 \quad \dots \quad 0] \mathbf{x}\quad (3.5)$$

#### Example 3.1

Obtain the state equation in phase variable form for the following differential equation.

$$2 \frac{d^3 y}{dt^3} + 4 \frac{d^2 y}{dt^2} + 6 \frac{dy}{dt} + 8y = 10u(t)$$

The M-file **ode2phv.m** is developed which converts an  $n$ th-order ordinary differential equation to the state-space phase variable form. **[A,B,C] = ode2phv(ai,k)** returns the matrices **A,B,C**, where **ai** is a row vector containing coefficients of the equation in descending order and **k** is the coefficient of the right-hand side.

```
ai = [ 2 4 6 8];
k = 10;
[A,B,C] = ode2phv(ai,k)
```

produces the following phase variable state representation

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -4 & -3 & -2 \end{bmatrix} & \mathbf{B} &= \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} & \mathbf{C} &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \end{aligned}$$

### 3.2 Equations of Electrical Networks

The state variables are directly related to the energy-storage elements of a system. It would seem, therefore, that the number of independent initial conditions is equal



to the number of energy-storing elements. This is true provided that there is no loop containing only capacitors and voltage sources and there is no cut set containing only inductive and current sources. In general, if there are  $n_C$  loops and  $n_L$  cut sets, the number of state variables is

$$n = e_L + e_C - n_C - n_L \quad (3.6)$$

where

- $e_L$  = number of inductors
- $e_C$  = number of capacitors
- $n_C$  = number of all capacitive and voltage source loops
- $n_L$  = number of all inductive and current source cut sets

### Example 3.2

Write the state equation for the network shown in Figure 3.1.

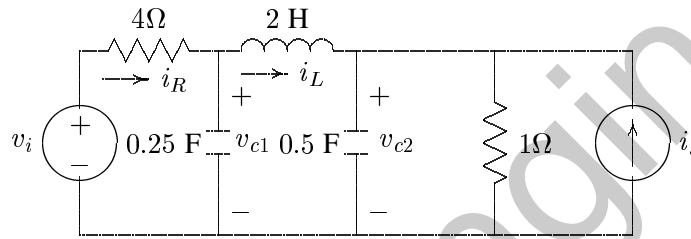


FIGURE 3.1

Circuit of Example 3.2.

Define the state variables as current through the inductors and the voltage across the capacitors. Write node equations containing capacitors and a loop equation containing an inductor. The state variables are  $v_{c1}$ ,  $v_{c2}$  and  $i_L$ . Node equations are

$$0.25 \frac{dv_{c1}}{dt} + i_L - i_R = 0$$

where

$$i_R = \frac{v_i - v_{c1}}{4}$$

$$0.5 \frac{dv_{c2}}{dt} - i_L + \frac{v_{c2}}{1} - i_s = 0$$

and the loop equation is

$$2 \frac{di_L}{dt} + v_{c2} - v_{c1} = 0$$

or

$$\begin{bmatrix} \dot{v}_{c1} \\ \dot{v}_{c2} \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} -1 & 0 & -4 \\ 0 & -2 & 2 \\ 0.5 & -0.5 & 0 \end{bmatrix} \begin{bmatrix} v_{c1} \\ v_{c2} \\ i_L \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_i \\ i_s \end{bmatrix}$$

### 3.3 Transfer Function to State-Space Conversion

The *Control System Toolbox* contains a set of functions for model conversion.  $[A, B, C, D] = \text{tf2ss}(\text{num}, \text{den})$  converts the system in transfer function form to state-space form.

#### Example 3.3

Find the state-space representation of the following transfer function.

```
num = [ 1 7 2]; den = [ 1 9 26 24 ];  
[A, B, C, D] = tf2ss(num, den)
```

results in the following matrices

$$A = \begin{bmatrix} -9 & -26 & -24 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 7 & 2 \end{bmatrix} \quad D = 0$$

### 3.4 State-Space to Transfer Function Conversion

Given the state and output equations

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \quad (3.7)$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}u \quad (3.8)$$

taking the Laplace transform and rearranging

$$\mathbf{Y}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s) + \mathbf{D}\mathbf{U}(s)$$

or

$$\mathbf{G}(s) = \frac{\mathbf{Y}(s)}{\mathbf{U}(s)} = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \quad (3.9)$$

$[\text{num}, \text{den}] = \text{ss2tf}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, i)$  converts the state equation to a transfer function for the  $i$ th input.

#### Example 3.4

A system is described by the following state-space equations

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ -1 & -2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{x}$$

Find the transfer function,  $G(s) = Y(s)/X(s)$

```
A = [0 1 0; 0 0 1; -1 -2 -3]; B = [10; 0; 0];
C = [1 0 0]; D = [0];
[num,den] = ss2tf(A,B,C,D,1)
```

results in the following coefficients

```
num =
    0.0000    10.0000    30.0000    20.0000
den =
    1.0000    3.0000    2.0000    1.0000
```

Therefore, the transfer function is

$$G(s) = \frac{10(s^2 + 3s + 2)}{s^3 + 3s^2 + 2s + 1}$$

Also, `[z,p] = ss2tf(A,B,C,D,1)` converts the state equation to transfer function in factored form.

### 3.5 Similarity Transformation

#### 3.5.1 Diagonalization of the A Matrix

One of the reasons for diagonalizing the **A** matrix, assuming we have distinct eigenvalues, is that they are all located on the main diagonal. It follows that the state transition matrix also is diagonal with elements  $e^{\lambda_1 t}, e^{\lambda_2 t}, \dots, e^{\lambda_n t}$ .

Given the linear system  $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}(t)$ , where **A** has distinct eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_n$  it is desired to find a nonsingular matrix **P** such that the transformation matrix **P**

$$\mathbf{x}(t) = \mathbf{P}\mathbf{y}(t) \quad (3.10)$$

transforms the above state equation into the canonical form

$$\dot{\mathbf{y}} = \mathbf{a}\mathbf{y} + \mathbf{b}\mathbf{u}(t) \quad (3.11)$$

with **a** given by the diagonal matrix

$$\mathbf{a} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P} \quad \text{and} \quad \mathbf{b} = \mathbf{P}^{-1}\mathbf{B}$$

In general, there are several methods of finding **P**. **P** can be formed by the use of the eigenvectors of **A**.

#### Example 3.5

Given the system represented in the state space by the following equation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \\ -6 & -11 & 6 \\ -6 & -11 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{x}$$

Find the transformation matrix **P** that transforms the above state equation into the canonical (diagonal) form.

```

A = [0 1 -1; -6 -11 6; -6 -11 5]; B = [0; 0; 1];
[P,L] = eig(A); % L is a diagonal matrix of eigenvalues
               % P is a modal matrix whose columns are
               % the corresponding eigenvectors
P
a = inv(P)*A*P % Diagonalization of the A matrix
b = inv(P)*B
    
```

The result is

```

P =
    -0.7071    0.2182   -0.0921
    -0.0000    0.4364   -0.5523
    -0.7071    0.8729   -0.8285

a =
    -1.0000   -0.0000    0.0000
     0.0000   -2.0000    0.0000
     0.0000   -0.0000   -3.0000

b =
    -2.8284
   -13.7477
    10.8628
    
```

### 3.5.2 Transformation to Phase-variable

In the state equation of a linear time invariant system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \quad (3.12)$$

if the matrix

$$\mathbf{S} = \begin{bmatrix} \mathbf{B} & \mathbf{A}\mathbf{B} & \mathbf{A}^2\mathbf{B} & \dots & \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix} \quad (3.13)$$

is nonsingular, then there exists a nonsingular transformation

$$\mathbf{y}(t) = \mathbf{Q}\mathbf{x}(t) \quad (3.14)$$

or

$$\mathbf{x}(t) = \mathbf{Q}^{-1}\mathbf{y}(t) \quad (3.15)$$

which transforms the above state equation to the phase-variable form.

$$\dot{\mathbf{y}}(t) = \mathbf{a}\mathbf{y}(t) + \mathbf{b}u(t) \quad (3.16)$$

where

$$\mathbf{a} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \dots & 1 \\ -a_1 & -a_2 & -a_3 & \dots & -a_n \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \quad (3.17)$$

$$\mathbf{a} = \mathbf{Q}\mathbf{A}\mathbf{Q}^{-1} \quad \text{and} \quad \mathbf{b} = \mathbf{Q}\mathbf{B}$$

The transformation  $\mathbf{Q}$  is given by

$$\mathbf{Q} = \begin{bmatrix} Q_1 \\ Q_1 A \\ \vdots \\ Q_1 A^{n-1} \end{bmatrix} \quad (3.18)$$

where

$$\mathbf{Q}_1 = [0 \quad 0 \quad \dots \quad 1] [B \quad AB \quad AB^2 \quad \dots \quad A^{n-1}B]^{-1} \quad (3.19)$$

The M-file **ss2phv** is developed which performs the above transformation. **[a,b] = ss2phv(A,B)** returns **a** and **b** in phase-variable form.

### Example 3.6

Transform the system given below into phase-variable form.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 3 & 0 & 2 \\ -12 & -7 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} -1 \\ 2 \\ 3 \end{bmatrix} u$$

$\mathbf{A} = [0 \quad 1 \quad 0; 3 \quad 0 \quad 2; -12 \quad -7 \quad -6];$

$\mathbf{B} = [-1; 2; 3];$

$[\mathbf{a}, \mathbf{b}] = \text{ss2phv}(\mathbf{A}, \mathbf{B})$

results in

$$\mathbf{Q} = \begin{bmatrix} 0.2500 & 0.0714 & 0.0357 \\ -0.2143 & 0.0000 & -0.0714 \\ 0.8571 & 0.2857 & 0.4286 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 0.0000 & 1.0000 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 \\ -6.0000 & -11.0000 & -6.0000 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

### 3.6 Solution of the State Equation

The solution of the linear nonhomogenous state equation

$$\dot{\mathbf{x}}(\mathbf{t}) = \mathbf{A}\mathbf{x}(\mathbf{t}) + \mathbf{B}\mathbf{u}(\mathbf{t}) \quad (3.20)$$

can be obtained by the Laplace transform approach.

$$s\mathbf{X}(s) - \mathbf{x}(0) = \mathbf{A}\mathbf{X}(s) + \mathbf{B}\mathbf{U}(s)$$

or

$$\mathbf{X}(s) = \Phi(s)\mathbf{x}(0) + \Phi(s)\mathbf{B}\mathbf{U}(s) \quad (3.21)$$

where

$$\Phi(s) = (s\mathbf{I} - \mathbf{A})^{-1} \quad (3.22)$$

$\phi(\mathbf{t}) = \mathcal{L}^{-1}[\Phi(s)]$  is known as the state transition matrix. Thus, solution of the state equation is

$$\mathbf{X}(\mathbf{t}) = \mathcal{L}^{-1}[\Phi(s)]\mathbf{x}(0) + \mathcal{L}^{-1}[\Phi(s)\mathbf{B}\mathbf{u}(s)] \quad (3.23)$$

We can also express the above equation in terms of  $\phi(\mathbf{t})$  and the convolution integral.

$$\mathbf{x}(\mathbf{t}) = \phi(\mathbf{t})\mathbf{x}(0) + \int_0^{\mathbf{t}} \phi(\tau)\mathbf{B}\mathbf{u}(\mathbf{t} - \tau)\mathbf{d}\tau \quad (3.24)$$

If  $\mathbf{A}$  is nonsingular, then the above equation can be simplified to give the following impulse, step and ramp responses.

For impulse input  $u(t) = K\delta(t)$ , the response is

$$\mathbf{x}(\mathbf{t}) = \phi(\mathbf{t})\mathbf{x}(0) + \phi(\mathbf{t})\mathbf{B}\mathbf{K} \quad (3.25)$$

For step input  $u(t) = K$ , the response is

$$\mathbf{x}(\mathbf{t}) = \phi(\mathbf{t})\mathbf{x}(0) + \mathbf{A}^{-1}[\phi(\mathbf{t}) - \mathbf{I}]\mathbf{B}\mathbf{K} \quad (3.26)$$

For ramp input  $u(t) = Kt$ , the response is

$$\mathbf{x}(\mathbf{t}) = \phi(\mathbf{t})\mathbf{x}(0) + (\mathbf{A}^2)^{-1}[\phi(\mathbf{t}) - \mathbf{I} - \mathbf{A}\mathbf{t}]\mathbf{B}\mathbf{K} \quad (3.27)$$

### 3.7 Laplace Transform of State Transition Matrix, $\Phi(s)$

$\Phi(s)$  is obtained from the Faddeeva algorithm given by

$$\Phi(s) = (s\mathbf{I} - \mathbf{A})^{-1} = \frac{s^{n-1}E_{n-1} + s^{n-2}E_{n-2} + \dots + E_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0} \quad (3.28)$$

and the  $E$  matrices are

$$E_{n-1} = \mathbf{I}, E_{n-1-k} = \mathbf{A}E_{n-k} + a_{n-k}\mathbf{I} \quad K = 1, \dots, n-1$$

The M-file **ltstm**, is developed which computes  $\Phi(s)$  according to the above algorithm.

### Example 3.7

Determine  $\Phi(s)$  for the system given below

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -2 & -1 \\ 2 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r(t)$$

A = [-2 -1; 2 -5];  
ltstm(A)

results in

$\phi(s) = \text{inv}(SI - A) = P / q$  where,

P = s\*\*(n-1)E(n-1) + S\*\*(n-2)E(n-2) + . . . + E(0)

q = a(n)s\*\*n + a(n-1)s\*\*(n-1) + a(1)s + . . . + a(0)

a(i) = coefficients of the characteristic equation q

The E matrices in descending power of s are :

E =

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

E =

$$\begin{bmatrix} 5 & -1 \\ 2 & 2 \end{bmatrix}$$

a =

$$\begin{bmatrix} 1 & 7 & 12 \end{bmatrix}$$

Therefore  $\Phi(s)$  is given by

$$\Phi(s) = \frac{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} s + \begin{bmatrix} 5 & -1 \\ 2 & 2 \end{bmatrix}}{s^2 + 7s + 12} = \frac{\begin{bmatrix} s+5 & -1 \\ 2 & s+2 \end{bmatrix}}{(s+3)(s+4)}$$

## 3.8 Evaluation of $\phi(t)$ from the Characteristic Values of A

### 3.8.1 Cayley-Hamilton Method

The *Cayley-Hamilton theorem* states that if the characteristic equation of any square matrix A is

$$\lambda^n + \alpha_1 \lambda^{n-1} + \alpha_2 \lambda^{n-2} + \dots + \alpha_n = 0 \quad (3.29)$$

then A satisfies the matrix equation,

$$\mathbf{A}^n + \alpha_1 \mathbf{A}^{n-1} + \alpha_2 \mathbf{A}^{n-2} + \dots + \alpha_n \mathbf{I} = 0 \quad (3.30)$$

That is, every square matrix satisfies its own characteristic equation.

Let

$$e^{\mathbf{A}t} = k_1(t)\mathbf{I} + k_2(t)\mathbf{A} + k_3(t)\mathbf{A}^2 + \dots + K_n(t)\mathbf{A}^{n-1} \quad (3.31)$$

It can be shown that a scalar equation, equivalent to the above equation, is satisfied when  $\mathbf{A}$  is replaced by  $\lambda$ ; that is,

$$\mathbf{e}^{\lambda t} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^{n-1} \\ 1 & \lambda_3 & \lambda_3^2 & \dots & \lambda_3^{n-1} \\ \vdots & & & & \\ 1 & \lambda_n & \lambda_n^2 & \dots & \lambda_n^{n-1} \end{bmatrix} \mathbf{K} \quad (3.32)$$

where  $\lambda$ 's are the distinct eigenvalues of  $\mathbf{A}$ . When two eigenvalues are equal, for example when  $\lambda_2 = \lambda_3$ , then the third row of the above matrix is replaced by

$$\frac{de^{\lambda t}}{d\lambda} = \frac{d}{dt}(k_1 + \lambda k_2 + \lambda^2 k_3 + \dots + \lambda^{n-1} K_n) \quad (3.33)$$

The M-file **stm** is developed based on the Cayley-Hamilton method. This function evaluates the state transition matrix in closed form. Repeated eigenvalues are of multiplicity two. The following example demonstrates the use of this function.

### Example 3.8

Find the state transition matrix  $\phi(t)$  for the system of Example 3.7.

```
A=[-2  -1; 2  -5];
stm(A)
```

results in

The state transition matrix is given by:

$$\phi(t) = \sum C_i \exp(L_i t) + \sum \{D_j t \exp(L_j t)\} \quad i=1, \dots, n-j$$

where

$L_i$  = eigenvalues &  $C_i$  = the corresponding constituent matrix

$L_j$  = repeated eigenvalues &  $D_j$  = the constituent matrix.

```
L_i =
-3
```

```
C_i =
2  -1
2  -1
```

```
L_i =
-4
```

```
C_i =
-1  1
-2  2
```

Thus, the state transition matrix is

$$\phi(t) = \begin{bmatrix} 2 & -1 \\ 2 & -1 \end{bmatrix} e^{-3t} + \begin{bmatrix} -1 & 1 \\ -2 & 2 \end{bmatrix} e^{-4t} = \begin{bmatrix} 2e^{-3t} - e^{-4t} & -e^{-3t} + e^{-4t} \\ 2e^{-3t} - 2e^{-4t} & -e^{-3t} + 2e^{-4t} \end{bmatrix}$$



### 3.9 Numerical Solution of the State Equation

The practical procedure for finding the time response of a system is through digital simulation. State-space representation enables us to simulate control systems on the computer. We have already seen the numerical solution of differential equations using functions **ode23** and **ode45** in Chapter 2. In fact, in order to solve an  $n$ th-order differential equation, it was necessary to transform it into  $n$  first-order differential equations, namely the state-variable representation.

For continuous-time linear systems, the *Control System Toolbox* provides the functions  $[y,x] = \text{impz}(\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{i},t)$  and  $[y,x] = \text{step}(\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{i},t)$  which obtain the impulse response and step response of the state equation. The function  $[y,x] = \text{lsim}(\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{U},t)$  simulates the state equation with arbitrary input.

#### Example 3.9

The state equation of a linear time-invariant system is represented by

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} r(t)$$

$$y = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \mathbf{x}$$

Given

$$\mathbf{x}(0) = \begin{bmatrix} 1 \\ 0.5 \\ -0.5 \end{bmatrix}$$

Determine  $\mathbf{x}(t)$  and  $y(t)$ , where  $r(t)$  is a unit step function.

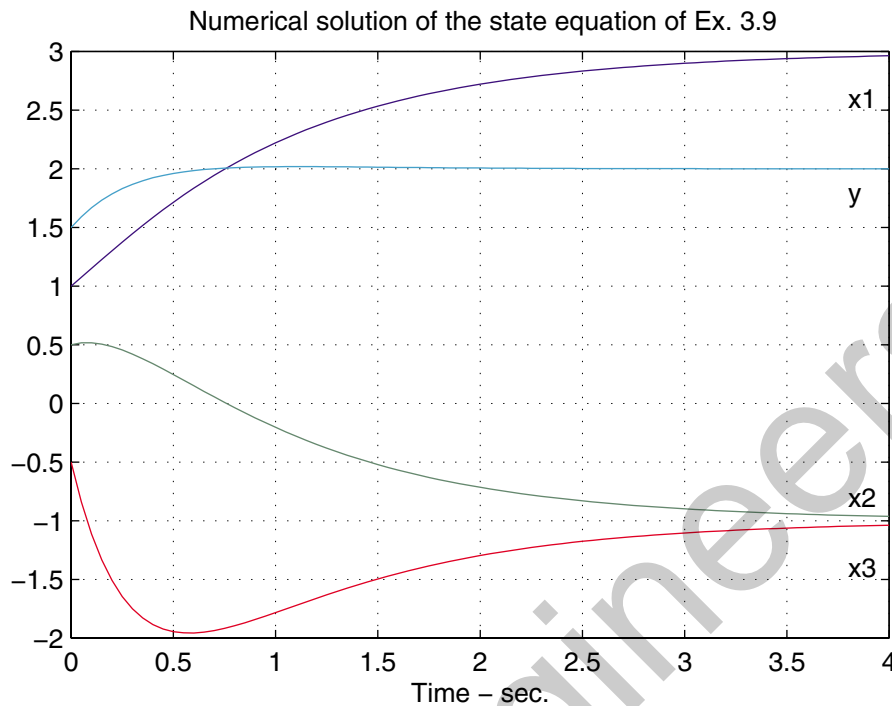
```
A = [0 1 0; 0 0 1; -6 -11 -6];
B = [1;1;1]; C=[1 1 0]; D=0;
x0=[1 .5 -.5]; t=0:.05:4;
U=ones(1,length(t)); % generates a row vector u(t)
[y,x]=lsim(A,B,C,D,U,t,x0);
plot(t,x,t,y)
title('Numerical solution of the state equation of Ex. 3.9')
xlabel('Time - sec.')
text(3.8,1.8,'y'), text(3.8,2.6,'x1'), text(3.8,-.8,'x2'),
text(3.8,-1.4,'x3')
```

The output is shown graphically in Figure 3.2.

#### Example 3.10

For Example 3.9, determine and plot  $y(t)$  and  $\mathbf{x}(t)$  if the input is given by  $r(t) = \sin(2\pi t)$ .

```
A = [0 1 0; 0 0 1; -6 -11 -6];
B = [1; 1; 1]; C=[1 1 0]; D=0;
t = 0:.05:4; % time interval
```



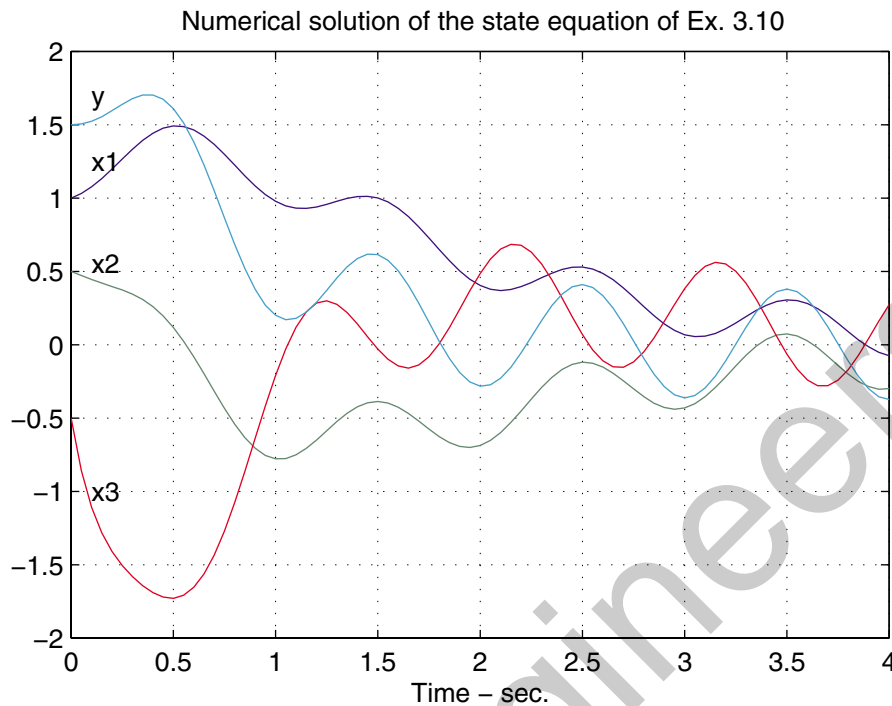
**FIGURE 3.2**  
Numerical solution for Example 3.9.

```
U = sin(2*pi*t);
x0 = [1 0.5 -0.5]; % row vector of initial conditions
[y,x] = lsim(A,B,C,D,U,t,x0);
plot(t,x,t,y)
title('Numerical solution of the state equation of Ex. 3.10')
xlabel('Time - sec.')
text(.1, 1.7, 'y'), text(.1, 1.25, 'x1'),
text(.1, .55, 'x2'), text(.1, -1, 'x3')
```

The output is shown graphically in Figure 3.3.

### 3.10 Block Diagram Reduction

The *MATLAB Control System Toolbox* script file **blkbuild** and the function **connect** convert block diagrams to state-space models. The transfer function blocks are numbered sequentially from 1 to the number of the blocks. **nblocks** defines the total number of blocks and **bldblock** converts each block to an unconnected state-space representation. The statement **[A,B,C,D] = connect(a,b,c,d,q,iu,iy)** connects up the blocks according to a predefined matrix **q** that specifies the interconnections. The first element of each row of the **q** matrix is the block number. The remaining elements indicate the source of the block's summing input. When the input to the summing

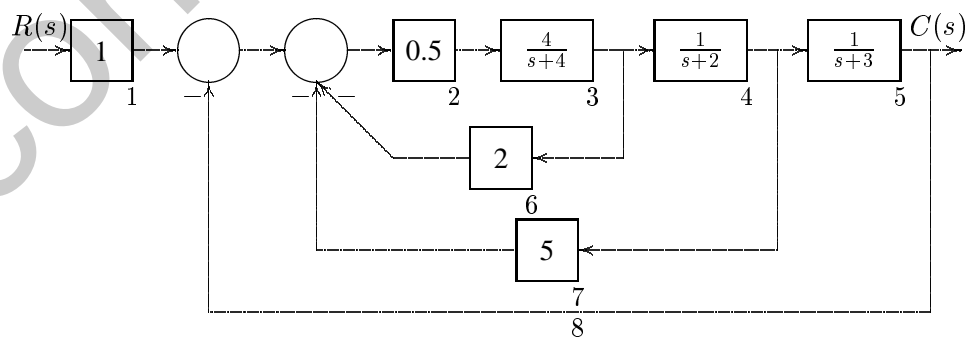


**FIGURE 3.3**  
Numerical solution for Example 3.10.

junction is negative, the block number is entered with a negative sign. The **iu** and **iy** are two row vectors indicating retaining input and output blocks. Finally, to obtain the overall transfer function, **[num,den] = ss2tf(A,B,C,D,iu)** calculates the transfer function from the **iu**'th input.

### Example 3.11

Determine the state-space representation and the overall transfer function for a system with the following block diagram representation.



$$n1 = 1; \quad d1=1; \quad n2 = .5; \quad d2=1; \quad n3 = 4; \quad d3=[1 \ 4];$$

### 76 3. State-Space Representation

```
n4 = 1; d4=[1 2]; n5 = 1; d5=[1 3]; n6 = 2; d6=1;
n7 = 5; d7=1; n8 = 1; d8=1;
nblocks=8; blkbuild
q= [ 1 0 0 0 0 % q matrix indicates the block
     2 1 -6 -7 -8 % diagram configurations
     3 2 0 0 0
     4 3 0 0 0
     5 4 0 0 0
     6 3 0 0 0
     7 4 0 0 0
     8 5 0 0 0 ];
iu = [1]; % input for the connected system
iy = [8]; % output for the connected system
[A ,B,C,D]=connect(a,b,c,d,q,iu,iy) % connect the block
[num,den]=ss2tf(A,B,C,D,1) % convert to transfer function
```

The result is

```
A =      B =      C =      D =
    -8.0    0.5    0    0
     4.0    0    0    1
     0    0    0    0
     0    0    0    0
    num =      Den =
     0    0    0    2    1.0    13.0    56.0    80.0
```

Thus, the overall transfer function is

$$\frac{C(s)}{R(s)} = \frac{2}{s^3 + 13s^2 + 56s + 80}$$

The *Control System Toolbox* contains four more functions which are useful in model building. **append** lumps together the dynamics of two state-space systems, forming an augmented model. **parallel** and **series** connect two state-space systems in parallel and series, respectively. Finally, **ode** generates  $A$ ,  $B$ ,  $C$ ,  $D$  for a second order system.

A much better method of finding the overall transfer function of a control system is to build the block diagram as a *SIMULINK* model. Run the simulation and to extract the linear model of this *SIMULINK* system, in the Command Window, enter the command

```
[A,B,C,D] = linmod('Simulink file name')
[num, den]=ss2tf(A, B, C, D)
```

See Example 1.27.

---

# CHAPTER 4

---

## SYSTEM RESPONSES

Assessing the time-domain performance of closed-loop system models is important because control systems are inherently time-domain systems. The performance of dynamic systems in the time domain can be defined in terms of the time response to standard test inputs. One very common input to control systems is the step function. If the response to a step input is known, it is mathematically possible to compute the response to any input. Another input of major importance is the sinusoidal function. A sinusoidal steady-state output is obtained when an asymptotically stable linear system is subjected to a sinusoidal input. Thus, if we know the response of a linear time-invariant system to sinusoids of all frequencies, we have a complete description of the system.

### 4.1 The Response of Second-Order Systems

The standard form of the second-order transfer function is given by

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (4.1)$$

where  $\omega_n$  is the natural frequency. The natural frequency is the frequency of oscillation if all of the damping is removed. Its value gives us an indication of the speed of the response.  $\zeta$  is the dimensionless damping ratio. The damping ratio gives us an idea about the nature of the transient response. It gives us a feel for the amount of overshoot and oscillation that the response undergoes.

The transient response of a practical control system often exhibits damped oscillations before reaching steady-state. The underdamped response ( $\zeta < 1$ ) to a unit step input, subject to zero initial condition, is given by

$$c(t) = 1 - \frac{1}{\beta} e^{-\zeta \omega_n t} \sin(\beta \omega_n t + \theta) \quad (4.2)$$

where  $\beta = \sqrt{1 - \zeta^2}$  and  $\theta = \tan^{-1}(\beta/\zeta)$ .

## 4.2 Time-Domain Performance Specifications

The performance criteria that are used to characterize the transient response to a unit step input include rise time, peak time, overshoot, and settling time. We define the rise time  $t_r$  as the time required for the response to rise from 10 percent of the final value to 90 percent of the final value. The time to reach the peak value is  $t_p$ . The swiftness of the response is measured by  $t_r$  and  $t_p$ . The similarity with which the actual response matches the step input is measured by the percent overshoot and settling time  $t_s$ . For underdamped systems the percent overshoot P.O. is defined as

$$P.O. = \frac{\text{maximum value} - \text{final value}}{\text{final value}} \quad (4.3)$$

The peak time is obtained by setting the derivative of (4.2) to zero.

$$t_p = \frac{\pi}{\omega_n \sqrt{1 - \zeta^2}} \quad (4.4)$$

The peak value of the step response occurs at this time, and evaluating the response in (4.2) at  $t = t_p$  yields

$$C(t_p) = M_{pt} = 1 + e^{-\zeta \pi / \sqrt{1 - \zeta^2}} \quad (4.5)$$

Therefore, from (4.3), the percent overshoot is

$$P.O. = e^{-\zeta \pi / \sqrt{1 - \zeta^2}} \times 100 \quad (4.6)$$

Settling time is the time required for the step response to settle within a small percent of its final value. Typically, this value may be assumed to be  $\pm 2$  percent of the final value. For the second-order system, the response remains within 2 percent after 4 time constants, that is

$$t_s = 4\tau = \frac{4}{\zeta \omega_n} \quad (4.7)$$

A function called **timespec(num, den)** is written which obtains the time-domain performance specifications,  $P.O.$ ,  $t_p$ ,  $t_r$  and  $t_s$ . **num** and **den** are the numerator and denominator of the system closed-loop transfer function.

## 4.3 Effects of Additional Poles and Zeros

### 4.3.1 Addition of a Zero

The zeros of a transfer function affect the amplitude of a response component but do not affect the nature of the response. The closer the zero is to the dominant poles, the more effect it has on the transient response. The rise time and the peak time are decreased, while the overshoot is increased. As the zero moves away from the dominant poles, the response approaches that of the second-order system.

### 4.3.2 Addition of a Pole

Since the poles of the closed-loop transfer function are the roots of the characteristic equation, they control the transient response of the system directly. The rise time and the peak time are increased with a reduction in the overshoot, resulting in a more sluggish response. As the pole moves away from the dominant pole, it has less effect. Since this additional exponential term decays after five time-constants and if the pole is five times farther to the left than the dominant poles, the system can be represented by a second-order model.

The transfer function of a third-order system with one zero may be written in the following standard form

$$C(s) = \frac{\omega_n^2(1 + as)}{(1 + Ts)(s^2 + 2\zeta\omega_n s + \omega_n^2)} K/s \quad (4.8)$$

Taking the inverse Laplace transform, the underdamped transient response ( $\zeta < 1$ ) is

$$c(t) = K[1 + \frac{1}{\beta} \sqrt{\frac{\sigma}{\rho}} e^{-\zeta\omega_n t} \sin(\beta\omega_n t + \phi) + \frac{\chi}{\rho} e^{-\frac{t}{T}}] \quad (4.9)$$

where

$$\beta = \sqrt{1 - \zeta^2}, \sigma = 1 - 2\zeta a\omega_n + a^2\omega_n^2, \rho = 1 - 2T\zeta\omega_n + T^2\omega_n^2, \chi = \omega_n^2 T(a - T),$$

and

$$\phi = \tan^{-1} \left[ \frac{a\beta\omega_n}{1 - a\zeta\omega_n} \right] - \tan^{-1} \frac{\beta\omega_n T}{1 - T\zeta\omega_n} - \tan^{-1} \frac{\beta}{-\zeta}.$$

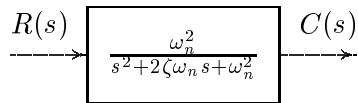
The function **c = stepzwn(z, ω<sub>n</sub>, R, a, T, t)** is developed which obtains the step response of (4.9) where **z** is the damping factor, **ω<sub>n</sub>** is the natural frequency, and **R** is the magnitude of the step function. For second-order systems, **a** and **T** are set to zero, and **t** is the specified time interval. Time-response formulas for the critically damped and overdamped cases are also obtained and included in the above function.

Given a transfer function of a closed-loop control system, the *Control System Toolbox* function **step(num, den)** produces the step response plot with the time vector automatically determined. If the closed-loop system is defined in state space, we use **step(A, B, C, D)**. **step(num, den, t)** or **step(A, B, C, D, iu, t)** uses the user-supplied time vector **t**. The scalar **iu** specifies which input is to be used for the step response. If the above commands are invoked with the left-hand arguments [**y, x, t**], the output

vector, the state response vectors, and the time vector **t** are returned, and we need to use **plot** function to obtain the plot. See also **initial** and **lsim** functions. A function called **timespec(num, den)** is written which obtains the time-domain performance specifications,  $P.O.$ ,  $t_p$ ,  $t_r$ , and  $t_s$ . **num** and **den** are the numerator and denominator of the system closed-loop transfer function.

#### Example 4.1

Obtain the step response and the time-domain specifications for the system shown below, where  $\zeta = 0.6$  and  $\omega_n = 5$ .

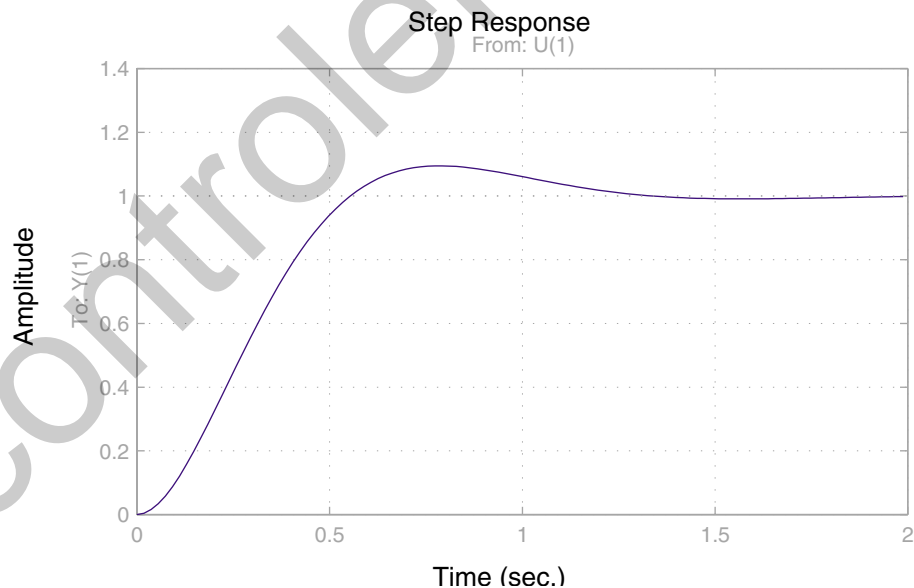


The commands

```
num = 25; den = [1 6 25];
step(num, den), grid
timespec(num, den)
```

result in

```
Peak time = 0.786667      Percent overshoot = 9.47783
Rise time = 0.373333
Settling time = 1.18667
```



**FIGURE 4.1**

Unit step response of Example 4.1.

The result is shown in Figure 4.1.



### Example 4.2

Obtain the unit step response, rise time, peak time and percent overshoot for the system whose closed-loop transfer function is given below

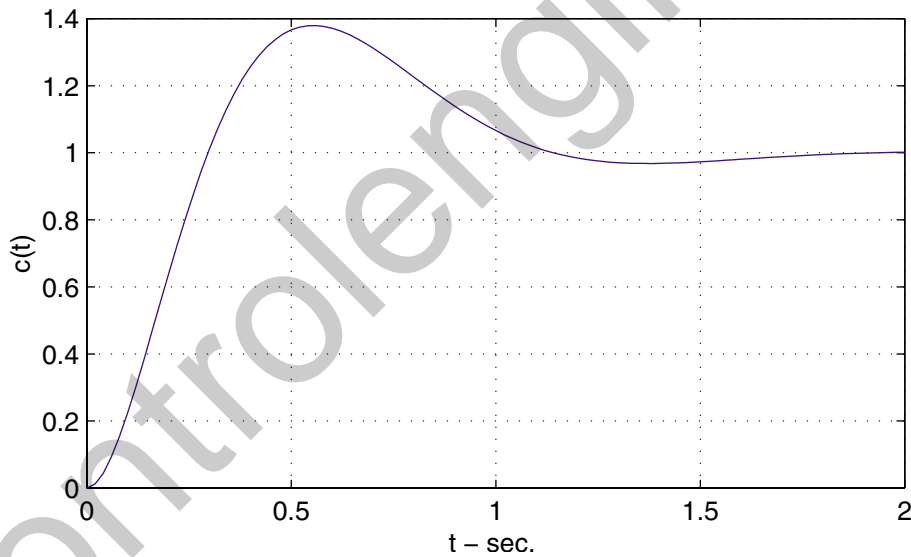
$$\frac{C(s)}{R(s)} = \frac{25(1 + 0.4s)}{(1 + 0.16s)(s^2 + 6s + 25)} = \frac{10s + 25}{0.16s^3 + 1.96s^2 + 10s + 25}$$

The following commands

```
num = [10, 25];
den = [0.16 1.96 10 25];
t = 0:0.02:2;
c = step(num, den, t); plot(t, c),
xlabel('t - sec. '), ylabel('c(t)'), grid
timespec(num, den)
```

result in

Peak time = 0.553333                      Percent overshoot = 37.9675  
Rise time = 0.206667  
Settling time = 1.59



**FIGURE 4.2**

Unit step response of Example 4.2.

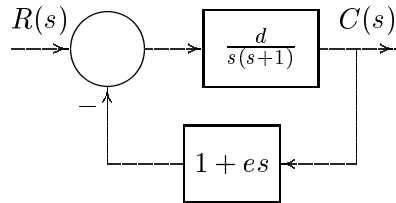
The result is shown in Figure 4.2.

### Example 4.3

The block diagram of a servomechanism is shown below. Determine values of  $d$  and  $e$  so that the maximum overshoot in unit step response is 40 percent and peak time is 0.8 second.

The following commands

## 82 4. System Responses



```

os = 40;   tmax=.80;
z = log(100/os)/sqrt( pi^2 +(log(100/os))^2 ) %From Eq. (4.6)
wn = pi/(tmax*sqrt(1-z^2)) %From Eq. (4.4)
num = wn^2;   den = [1 2*z*wn wn^2];
t = 0:0.02:4;
c = step(num, den, t); plot(t, c),
xlabel('t - sec. '), ylabel('c(t)'), grid
timespec(num, den),
    
```

result in

```

z = 0.2800
wn = 4.0906
    
```

```

Peak time = 0.803239      Percent overshoot = 39.9965
Rise time = 0.314311
Settling time = 3.37011
    
```

From the block diagram we have

$$\frac{C(s)}{R(s)} = \frac{d}{s^2 + (de + 1)s + d}$$

The characteristic equation is

$$s^2 + (de + 1)s + d = s^2 + 2\zeta\omega_n s + \omega_n^2$$

equating the coefficient

$$d = \omega_n^2 = 4.0906^2 = 16.733$$

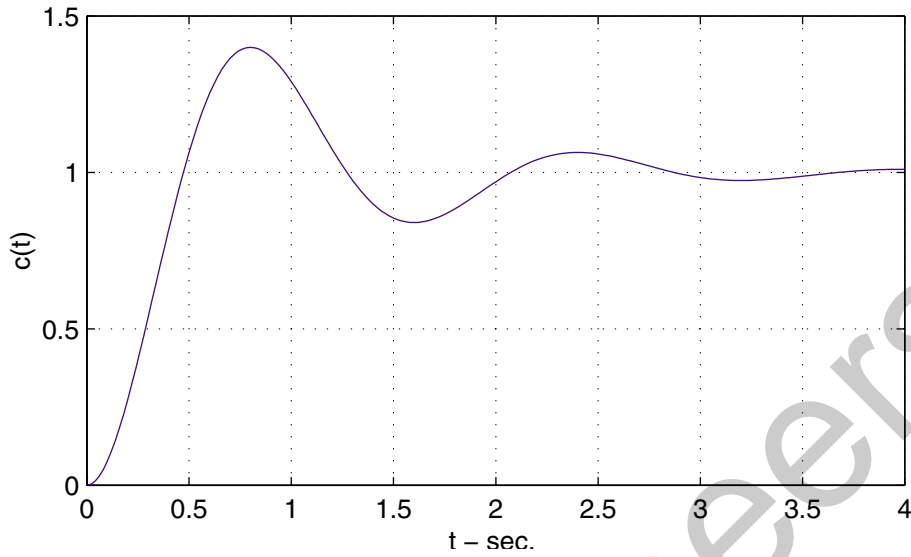
and

$$de + 1 = 2(0.28)(4.0906)$$

Therefore

$$e = 0.077$$

The result is shown in Figure 4.3.



**FIGURE 4.3**  
Unit step response of Example 4.3.

#### 4.4 Frequency Response of Systems

The frequency response of a system is defined as the steady-state response of the system to a sinusoidal input signal. Consider a system with transfer function  $G(s)$  and a sinusoidal input

$$r(t) = A \cos \omega t \quad (4.10)$$

Using the transform of  $r(t)$ , the transform  $C(s)$  of the system output is

$$C(s) = \frac{As G(s)}{s^2 + \omega^2} \quad (4.11)$$

The partial fraction expansion results in

$$C(s) = \frac{k_1}{s - j} + \frac{k_1^*}{s + j} + \Sigma \text{ terms generated by the poles of } G(s) \quad (4.12)$$

The poles of  $G(s)$  are the natural frequencies (or natural modes). They govern the waveform of the transient component of the response. For the linear lumped network, the terms generated by the poles of  $G(s)$  will not contribute to the steady-state response  $c(t)$ . Therefore, the steady-state response is given by the inverse Laplace transform of the first two terms of  $C(s)$ .

$$c(t) = A |G(j\omega)| \cos(\omega t + \theta) \quad (4.13)$$

From this equation it can be seen that system output has the same frequency as the input and can be obtained by multiplying the magnitude of the input by  $|G(j\omega)|$  and shifting the phase angle of the input by the angle of  $G(j\omega)$ . The magnitude  $G(j\omega)$  and

its angle  $\theta$  for all  $\omega$  constitute the *system frequency response*, and provide a significant insight for the analysis and design of control systems. The correlation between frequency and transient responses is indirect, except in the case of second-order systems. In practice, the frequency response characteristic is adjusted by using various design criteria which will normally result in an acceptable transient response.

Consider first the frequency response of a first-order system with the following transfer function

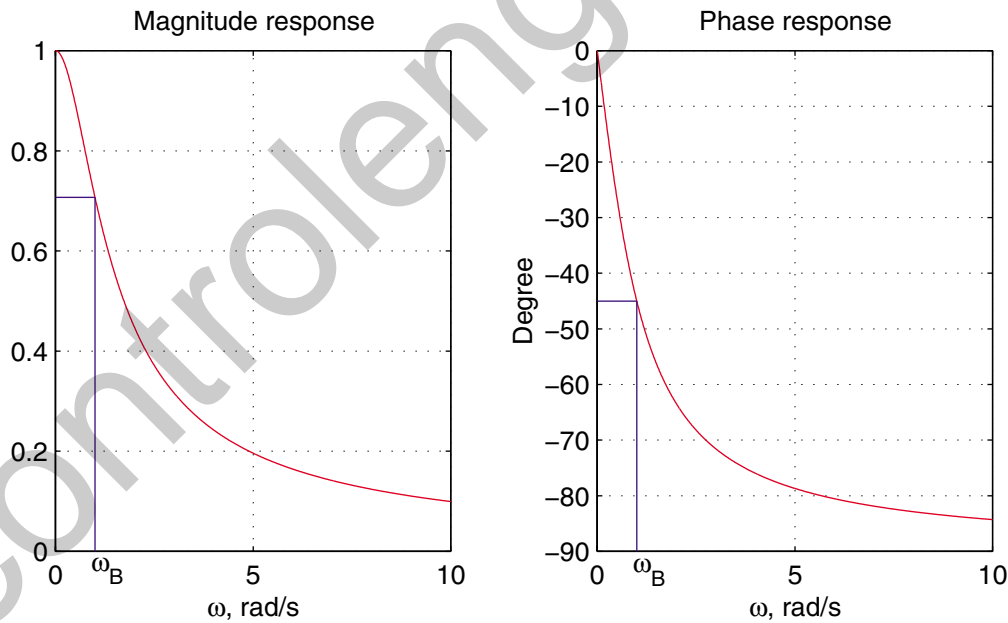
$$G(s) = \frac{1}{\tau s + 1} \quad (4.14)$$

The sinusoidal steady-state transfer function is given by

$$G(j\omega) = \frac{1}{[1 + \tau^2 \omega^2]^{1/2}} \angle \phi(\omega) \quad (4.15)$$

where  $\phi(\omega) = -\tan^{-1} \tau \omega$ .

Plots of  $|G(j\omega)|$  and  $\phi(\omega)$  are given in Figure 4.4. The *system bandwidth (BW)* is defined as that value of frequency at which the magnitude of the frequency response is reduced to  $1/\sqrt{2}$  of its low frequency value. This frequency is denoted by  $\omega_B$ . For the first-order system the bandwidth is given by  $\omega_B = 1/\tau$ .



**FIGURE 4.4**  
Frequency response of a first-order system.

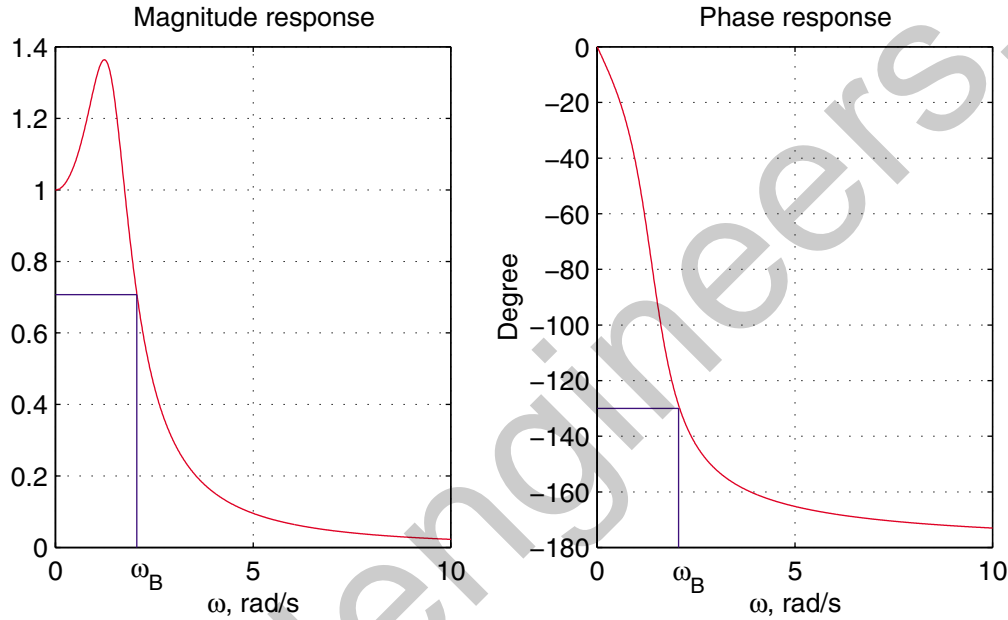
Now consider the standard second-order transfer function

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (4.16)$$

The frequency response is given by

$$G(j\omega) = \frac{1}{\left( \left(1 - \frac{\omega}{\omega_n}\right)^2 + \left(2\zeta \frac{\omega}{\omega_n}\right)^2 \right)^{1/2}} \angle \phi(\omega) \quad (4.17)$$

A plot of frequency response for a given  $\zeta$  and  $\omega_n$  is given in Figure 4.5.



**FIGURE 4.5**  
Frequency response of a second-order system.

For a constant  $\zeta$ , increasing  $\omega_n$  causes the bandwidth to increase by the same factor. This corresponds to a decrease in the peak time  $t_p$  and the rise time  $t_r$  of the transient response. Therefore, to increase the speed of response of a system, it is necessary to increase the system bandwidth. For a particular system, an approximate relationship is given by

$$\omega_B t_r \simeq \text{constant} \quad (4.18)$$

where this constant has a value approximately equal to 2.

The frequency at which the peak occurs is obtained by setting the derivative of (4.16) to zero. For  $\zeta < 0.707$ , the *resonance frequency*  $\omega_r$  is given by

$$\omega_r = \omega_n \sqrt{1 - 2\zeta^2} \quad (4.19)$$

The maximum value of the magnitude of the step response, denoted by  $M_{p\omega}$ , is

$$M_{p\omega} = \frac{1}{2\zeta \sqrt{1 - \zeta^2}} \quad (4.20)$$

The peak in the frequency response is directly related to the amount of overshoot in the transient response. The greater the peak, the more overshoot that occurs. Thus, in the control system design the  $M_{p\omega}$  is usually restricted to a maximum allowable value.

Given a transfer function of a system, the *Control System Toolbox* function **bode(num, den)** produces the frequency response plot with the frequency vector automatically determined. If the system is defined in state space, we use **bode(A, B, C, D)**. **bode(num, den,  $\omega$ )** or **bode(A, B, C, D,  $\omega$ )** uses the user-supplied frequency vector  $\omega$ . The scalar **iu** specifies which input is to be used for the frequency response. If the above commands are invoked with the left-hand arguments [**mag, phase,  $\omega$** ], the frequency response of the system in the matrices **mag**, **phase**, and  $\omega$  are returned, and we need to use **plot** or **semilogx** functions to obtain the plot.

For second-order systems with  $\zeta < 1$ , (4.18) and (4.19) can be utilized to compute the frequency  $\omega_r$  and the peak value  $M_{p\omega}$  of the frequency response. However, the function **frqspec(w, mag)** is developed which will return  $\omega_r$ ,  $M_{p\omega}$ , and  $\omega_B$  based on the functional values of **w** and **mag**.

#### Example 4.4

A system is described by the closed-loop transfer function

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

Obtain the frequency response, peak amplitude  $M_{p\omega}$ , frequency  $\omega_r$ , and the bandwidth  $\omega_B$  of the system.

The following commands

```

num = 4;
den = [1 2 4];
w=0:.01:3;
[mag, phase]=bode(num, den, w);
frqspec(w,mag)
plot(w, mag)
ylabel('Magnitude'),xlabel('\omega, rad/s'), grid
    
```

will result in

Peak Mag. = 1.15       $\omega_r$  = 1.41      Bandwidth = 2.54

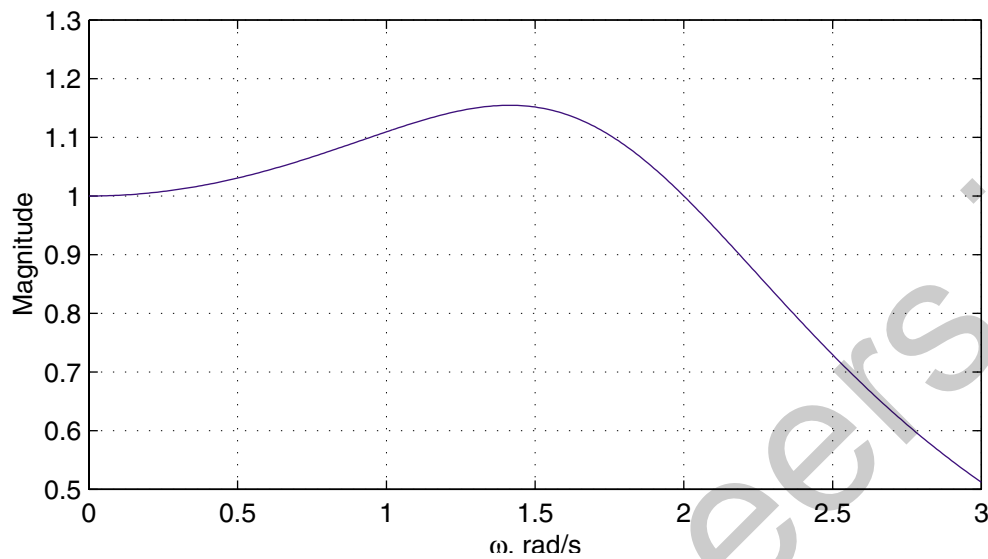
The result is shown in Figure 4.6.

#### Example 4.5

A pole is added to the closed-loop transfer function of Example 4.4 and the new transfer function is

$$G(s) = \frac{2.5 \times 4}{(s + 2.5)(s^2 + 2s + 4)} = \frac{10}{s^3 + 4.5s^2 + 9s + 10}$$

Determine:



**FIGURE 4.6**  
Frequency response of Example 4.4.

1. the step response and the rise time  $t_r$ .
2. the frequency response and the bandwidth  $\omega_B$ .
3. the approximate value of the bandwidth to have a rise time of 0.5 s in the transient response.

The following commands

```

num = 10;
den = [1 4.5 9 10];
t=0:.02:4;
c = step(num, den, t);
timespec(num, den)
w=0:.01:3;
[mag, phase]=bode(num, den, w);
frqspec(w, mag)
subplot(2,1,1), plot(t,c), title(' Step response')
ylabel('c(t)'),xlabel('Time, sec'), grid
subplot(2,1,2), plot(w,mag), title(' Frequency response')
ylabel('Magnitude'),xlabel('\omega, rad/s'), grid
    
```

result in

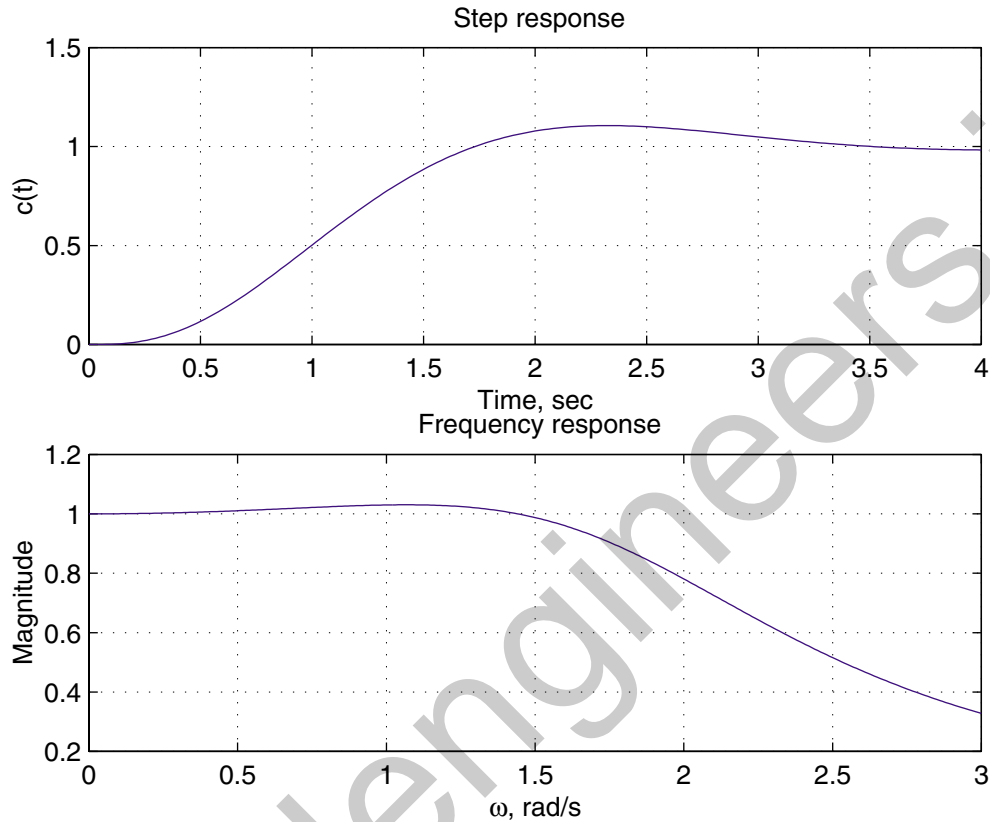
```

Peak time = 2.32          Percent overshoot = 10.6117
Rise time = 1.06
Settling time = 3.26
    
```

```

Peak Mag. = 1.03    wr = 1.07    Bandwidth = 2.13
    
```

The result is shown in Figure 4.7.



**FIGURE 4.7**  
Step response and frequency response of Example 4.5.

The bandwidth rise time product is  $\omega_B t_r = 2.14 \times 1.06 = 2.268$ . Thus from (4.18), for a rise time  $t_r = 0.5$  s, it is necessary to increase the system bandwidth to  $2.268/0.5 \simeq 4.54$ .

#### Example 4.6

Given the system described by the third-order transfer function

$$G(s) = \frac{750}{s^3 + 36s^2 + 205s + 750}$$

1. Find the dominant poles of the system.
2. Find a reduced-order model of the system. Determine  $t_r$ ,  $t_p$  and percent overshoot in the step response. Also, find  $\omega_r$ ,  $M_{p\omega}$  and the bandwidth  $\omega_B$  in the frequency response.
3. Find the exact values for the parameters and compare them with the values in



2).

The commands

```
a = [ 1 36 205 750];
r = roots(a)
```

result in

```
p =
    -30
    -3 + 4i
    -3 - 4i
```

Therefore the transfer function is

$$G(s) = \frac{750}{(s + 30)(s^2 + 6s + 25)} = \frac{25}{(1 + 0.0333s)(s^2 + 6s + 25)}$$

The dominant poles are  $-3 \pm j4$ . Since the real pole  $s = -30$  is far from the dominant poles, its effect is minimal and may be neglected. Therefore the approximate transfer function is

$$G(s) \simeq \frac{25}{s^2 + 6s + 25}$$

```
num1 = 25; den1 = [1 6 25]; % Approximate 2nd-order system
t=0:.02:2;
c1 = step(num1, den1, t);
timespec(num1, den1)
```

```
w=0:.02:8;
[mag1, phase1]=bode(num1, den1, w);
frqspect(w, mag1)
```

```
num2 =750; den2 =[1 36 205 750]; % 3rd-order system
c2= step(num2, den2, t);
timespec(num2, den2)
```

```
[mag2, phase2]=bode(num2, den2, w);
frqspect(w, mag2)
```

```
subplot(2,2,1), plot(t,c1), xlabel('t, sec.'), grid
title(' Approximate 2nd order system')
subplot(2,2,2), plot(w, mag1), xlabel('\omega, rad/s'), grid
subplot(2,2,3), plot(t,c2), xlabel('t, sec.'), grid
title(' Actual 3rd order system')
subplot(2,2,4), plot(w,mag2), xlabel('\omega, rad/s'), grid
subplot(111)
```

The result for the reduced-order system is

#### 90 4. System Responses

Peak time = 0.786667      Percent overshoot = 9.47783  
 Rise time = 0.373333  
 Settling time = 1.18667

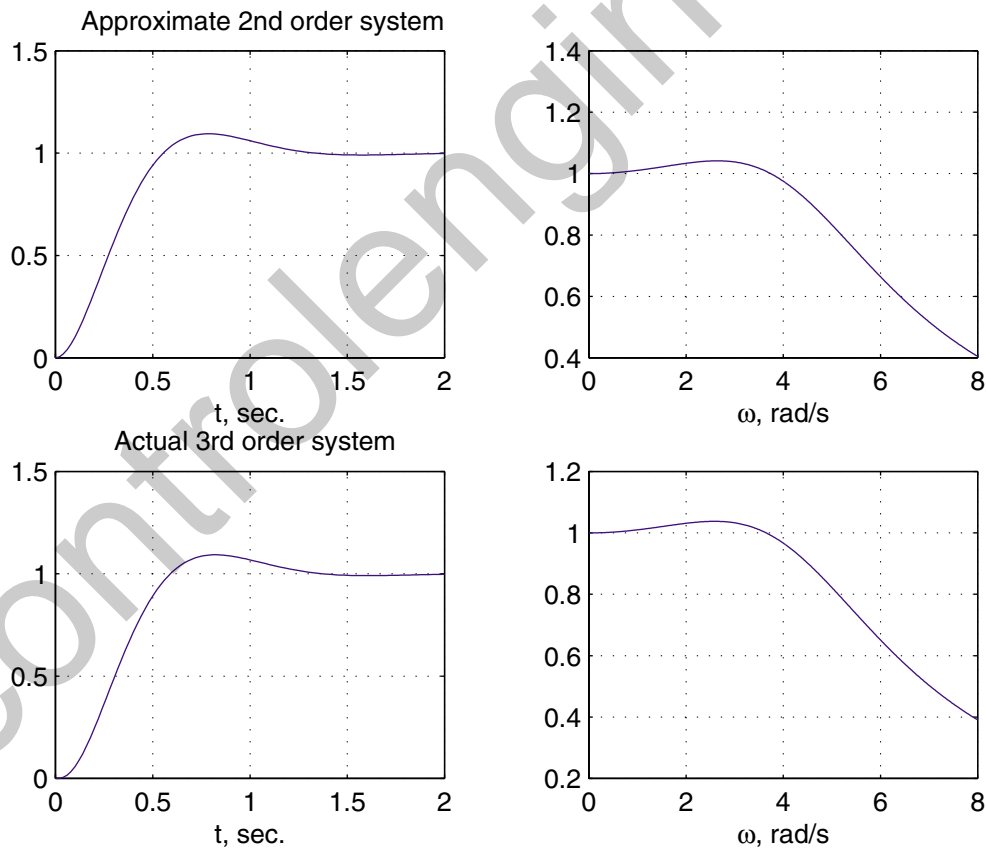
Peak Mag. = 1.04       $\omega_r = 2.64$       Bandwidth = 5.75

The third-order system parameters are

Peak time = 0.823333      Percent overshoot = 9.32926  
 Rise time = 0.376667  
 Settling time = 1.22333

Peak Mag. = 1.04       $\omega_r = 2.58$       Bandwidth = 5.67

The step and frequency response for the actual system and the approximate second-order system are shown in Figure 4.8. Comparison of the results shows little change in the parameters of the reduced-order model.



**FIGURE 4.8**  
Step response and frequency response of Example 4.6.

## 4.5 Control System Toolbox

### LTI Models and LTI Viewer

The Control System Toolbox provides many tools for analysis and design of control systems. In this section the LTI viewer is briefly described. For additional information and use of the control design tools refer to the Control System Toolbox version 4.

#### 4.5.1 LTI Models

The following commands forms the transfer function, zero/pole/gain, or state-space models

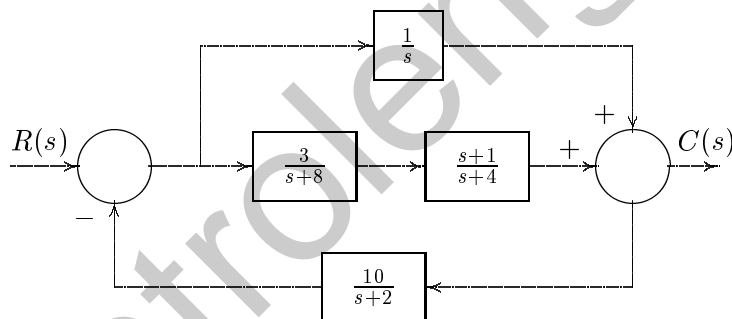
```

G1=tf(num, den)    % transfer function
G2=zpk(z, p, k)    % zero/pole/gain
T=ss(A, B, C, D)   % state space
    
```

The command **T = feedback(G, H)** returns the transfer function of a simple negative feedback control system. For positive feedback, an additional argument of +1 is used. That is, for positive feedback the syntax is **T = feedback(G, H, +1)**. You can perform summing and cascading operations on LTI systems.

#### Example 4.7

Obtain the closed-loop transfer function for the control system shown in Figure 4.9.



**FIGURE 4.9**  
Block diagram for Example 4.7.

The following commands

```

G=tf(3,[1 8])*tf([1 1],[1 4]) + tf(1,[1 0]);
H=tf(10,[1 2]);
T=feedback(G, H)
    
```

result in

Transfer function:

$$\frac{4s^3 + 23s^2 + 62s + 64}{s^4 + 14s^3 + 96s^2 + 214s + 320}$$

### 4.5.2 The LTI Viewer

The LTI Viewer is an interactive user interface that can be utilized to obtain various system responses. The command syntax is

**ltiview('plot type', sys, Extra)**

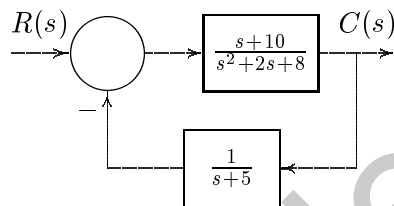
where sys is the transfer function name and 'plot type' is one of the following responses:

step	bode
impulse	nyquist
initial	nichols
lsim	sigma

Extra is an optional argument specifying the final time. Once an LTI Viewer is opened, the right-click on the mouse allows you to change the response type and obtain the system time-domain and frequency-domain characteristics.

#### Example 4.8

Use the LTI Viewer to obtain the step response and the closed-loop frequency response for the control system shown in Figure 4.10.



**FIGURE 4.10**  
Block diagram for Example 4.8.

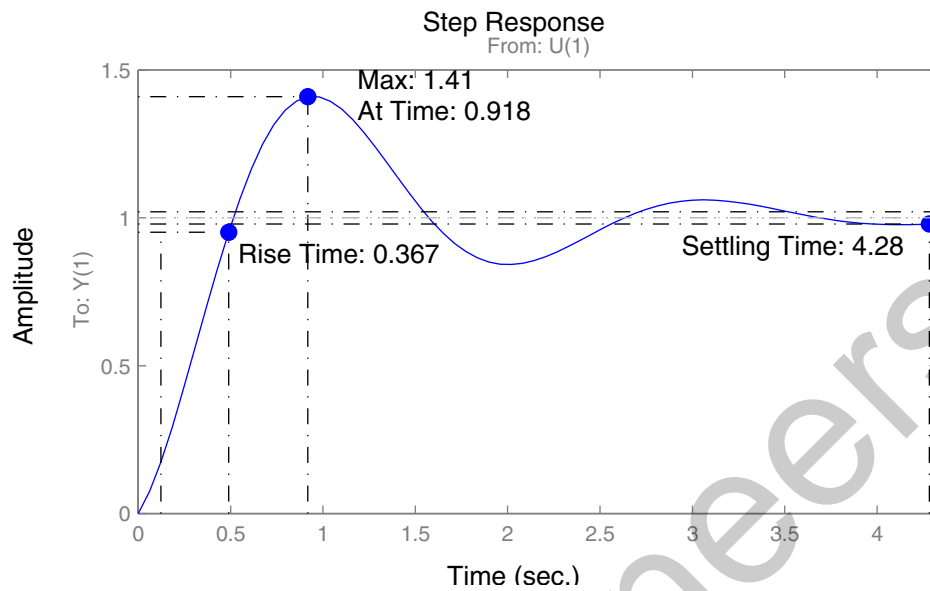
We use the following commands

```

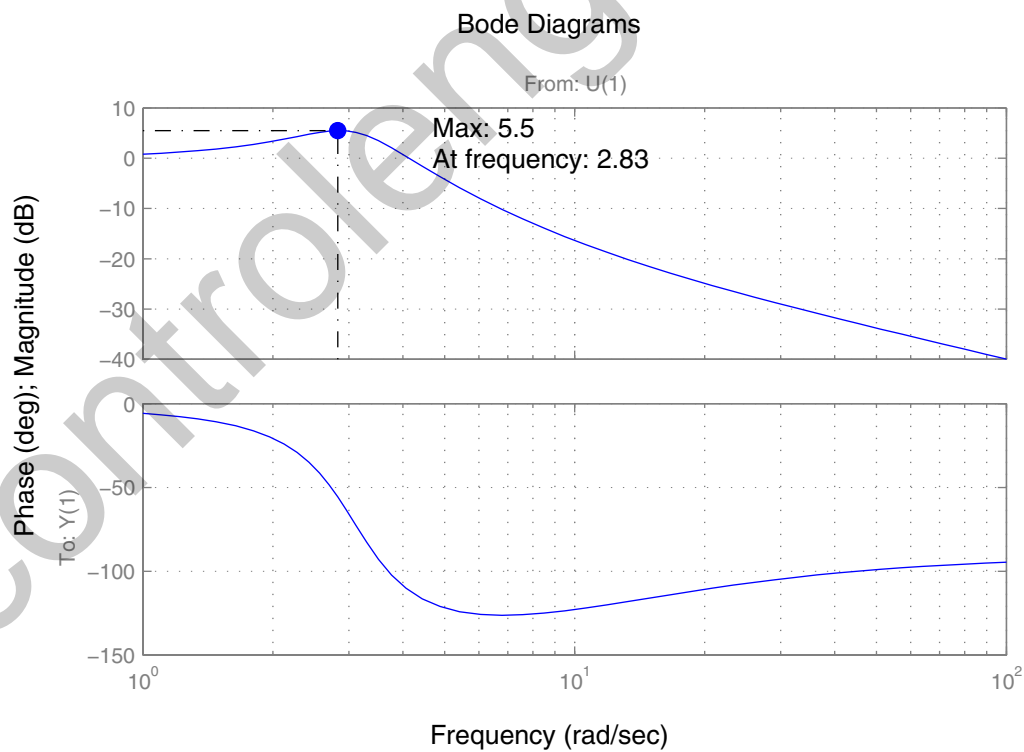
G=tf([1 10],[1 2 8]);
H=tf(1,[1 5]);
T=feedback(G, H)
ltiview('step', T)
    
```

The system step response is obtained as shown in Figure 4.11. The mouse right-click is used to obtain the time-domain specifications. From File menu you can select Print to Figure option to obtain a Figure Window for the LTI Viewer for editing the graph.

In the LTI Viewer, hold on the mouse right button and select the **Plot Type** pop-up menu, scroll down and select **Bode** plot. This will produce the amplitude and phase angle frequency response as shown in Figure 4.12. The mouse right-click is used to obtain the response peak amplitude.



**FIGURE 4.11**  
Step response of Example 4.8.



**FIGURE 4.12**  
Step response of Example 4.8.

---

# CHAPTER 5

---

## CONTROL SYSTEM CHARACTERISTICS

The objective of the control system is to control the output  $c$  in some prescribed manner by the input  $u$  through the elements of the control system. Some of the essential characteristics of feedback control systems are investigated in the following sections.

### 5.1 Stability

For a system to be usable it must be stable. A linear time-invariant system is stable if every bounded input produces a bounded output. We call this characteristic *stability*. The response is bounded if the poles of the closed loop system are in the left-hand portion of the  $s$ -plane. Thus, a necessary and sufficient condition for a feedback system to be stable is that all the poles of the system transfer function have negative real parts.

The stability of a linear time-invariant system may be checked by using the *Control System Toolbox* function **impz** to obtain the impulse response of the system. The system is stable if its impulse response approaches zero as time approaches infinity. One way to determine the stability of a system is by simulation. The function **lsim** can be used to observe the output for typical inputs. This is particularly useful for non-linear systems. Alternatively, the *MATLAB* function **roots** can be utilized to obtain the roots of the characteristic equations. In the classical control theory, several techniques have been developed requiring little computation for stability analysis. One of these techniques is the *Routh-Hurwitz criterion*. In this chapter an answer to the

question of absolute stability is demonstrated using a simple program based on the Routh-Hurwitz criterion. Consideration of the *degree* of stability of a system often provides valuable information about its behavior. That is, if it is stable, how close is it to being unstable? This is the concept of *relative stability*. Usually, relative stability is expressed in terms of the speed of response and overshoot. Other methods frequently used for stability studies are the *Bode diagram*, *Root-locus plot*, *Nyquist criterion*, and *Lyapunov's stability criterion*. Some of these techniques are presented in later chapters.

## 5.2 The Routh-Hurwitz Stability Criterion

The Routh-Hurwitz criterion provides a quick method for determining absolute stability that can be applied to an  $n$ th-order characteristic equation of the form

$$a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0 = 0 \quad (5.1)$$

The criterion is applied through the use of a *Routh table* defined as

$s^n$	$a_n$	$a_{n-2}$	$a_{n-4}$	$\dots$
$s^{n-1}$	$a_{n-1}$	$a_{n-3}$	$a_{n-5}$	$\dots$
$s^{n-2}$	$b_1$	$b_2$	$b_3$	$\dots$
$s^{n-3}$	$c_1$	$c_2$	$c_3$	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

$a_n, a_{n-1}, \dots, a_0$  are the coefficients of the characteristic equation and

$$b_1 = \frac{a_{n-1}a_{n-2} - a_n a_{n-3}}{a_{n-1}}, \quad b_2 = \frac{a_{n-1}a_{n-4} - a_n a_{n-5}}{a_{n-1}}, \quad \text{etc.}$$

$$c_1 = \frac{b_1 a_{n-3} - a_{n-1} b_2}{b_1}, \quad c_2 = \frac{b_1 a_{n-5} - a_{n-1} b_3}{b_1}, \quad \text{etc.}$$

Calculations in each row are continued until only zero elements remain. The necessary and sufficient condition that all roots of (5.1) lie in the left half of the  $s$ -plane is that the elements of the first column of the Routh array have the same sign. If there are changes of signs in the elements of the first column, the number of sign changes indicates the number of roots with positive real parts.

A function called **routh(a)** is written that forms the Routh array and determines if any roots have positive real parts. **a** is a row vector containing the coefficients of the characteristic equation.

### Example 5.1

Determine if the following characteristic equation represents a stable system.

$$s^4 + 10s^3 + 35s^2 + 50s + 24 = 0$$

## 96 5. Control System Characteristics

```
a = [1 10 35 50 24];
routh(a)
```

results in

```
Routh-Hurwitz Array
1      35      24
10     50       0
30     24       0
42      0       0
24      0       0
```

System is stable

### Example 5.2

Apply the Routh-Hurwitz criterion to the following characteristic equation to determine the number of roots in the right half  $s$ -plane.

$$s^4 + 4s^3 - 7s^2 - 22s + 24 = 0$$

```
a = [1 4 -7 -22 24];
routh(a)
```

results in

```
Routh-Hurwitz Array
1.0000  -7.0000  24.0000
4.0000 -22.0000   0
-1.5000  24.0000   0
42.0000   0       0
24.0000   0       0
```

There are 2 roots in the right half  $s$ -plane

### 5.2.1 Special Cases

#### Case 1

If the first element in a row is zero, it is replaced by a very small positive number  $\epsilon$ , and the calculation of the array is completed. This case is demonstrated in Example 5.3.

### Example 5.3

Use the Routh-Hurwitz criterion to determine the number of roots in the right half  $s$ -plane for

$$s^4 - 5s^2 + 20s + 24 = 0$$

```
a = [1 0 -5 20 24];
routh(a)
```



results in

Zero in the first column is replaced by 0.00001

Routh-Hurwitz Array			
1.0000e+000	-5.0000e+000	2.4000e+001	
1.0000e-005	2.0000e+001	0	
-2.0000e+006	2.4000e+001	0	
2.0000e+001	0	0	
2.4000e+001	0	0	

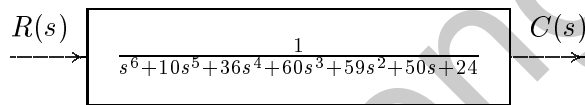
There are 2 roots in the right half s-plane

### Case 2

If all elements in a row are zero, the system has poles on the imaginary axis, pairs of complex conjugate roots forming symmetry about the origin of the  $s$ -plane, or pairs of real roots with opposite signs. In this case, an auxiliary equation is formed from the preceding row. The all-zero row is then replaced with coefficients obtained by differentiating the auxiliary equation. This case is demonstrated in Example 5.4.

### Example 5.4

Construct the Routh array for the characteristic equation of the system given in Figure 5.1.



**FIGURE 5.1**

System for Example 5.4.

```
a = [1 10 36 60 59 50 24];  
routh(a)
```

results in

Elements of row 6 are all zero.

They are replaced by the auxiliary Eq. coefficients

Routh-Hurwitz Array			
1	36	59	24
10	60	50	0
30	54	24	0
42	42	0	0
24	24	0	0
48	0	0	0
24	0	0	0

The characteristic equation includes roots on the  $j\omega$ -axis or pairs of real or complex roots symmetrical about the  $j\omega$ -axis

### 5.3 Sensitivity

One of the objectives of using feedback in a control system is to reduce the system's sensitivity to parameter variations and noise. A good control system should be insensitive to parameter changes or external disturbances. The *sensitivity* of a system can be measured as the ratio of the percentage change in the system transfer function to the percentage change in a parameter  $b$ . For example, the sensitivity of the transfer function  $T(s)$  to the variation in the parameter  $b$  is defined as

$$S_b^T = \frac{\Delta T(s)/T(s)}{\Delta b/b} = \frac{\Delta T(s)}{\Delta b} \frac{b}{T(s)} \quad (5.2)$$

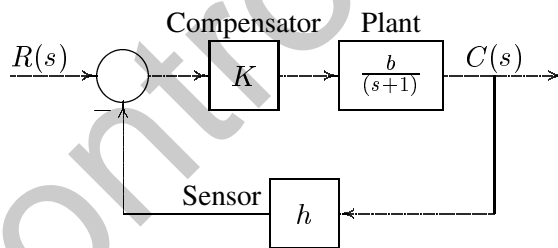
As  $\Delta b$  approaches zero, the sensitivity of  $T$  with respect to  $b$  is

$$S_b^T = \frac{\partial T(s)}{\partial b} \frac{b}{T(s)} \quad (5.3)$$

The *static sensitivity* is the value of  $S$  for  $s \rightarrow 0$ . *Dynamic sensitivities* are usually calculated by replacing  $s$  by  $j\omega$  and plotting  $S$  as a function of frequency,  $\omega$ . The magnitude of  $S(j\omega)$ , in fact, measures system errors. So the aim is to minimize it. This is comparatively easy to do at low frequencies because of the finite bandwidth of physical devices. Any physical system has a finite bandwidth. Therefore, the transfer function of a real control system always tends to zero, and its sensitivity function tends to unity at large  $\omega$  (i.e.,  $S(j\omega) \rightarrow 1$  as  $\omega \rightarrow \infty$ ). This condition leads to large system error and no disturbances can be rejected.

#### Example 5.5

Consider the control system represented by the block diagram shown in Figure 5.2, where  $b$  has a nominal value of 4 and  $h$  has a nominal value of 0.5.



**FIGURE 5.2**  
System for Example 5.5.

1. Find the sensitivity of  $T(s)$  with respect to  $b$ . Plot the magnitude of the sensitivity function as a function of frequency for  $K = 2$  and  $K = 0.5$ .
2. Find the sensitivity of  $T(s)$  with respect to  $h$ . Plot the magnitude of the sensitivity function as a function of frequency for  $K = 2$  and  $K = 0.5$ .

The system transfer function is

$$T(s) = \frac{Kb}{s + 1 + Kbh}$$

for  $b = 4$  and  $h = 0.5$ , the system bandwidth  $\omega_B$  is  $1 + 2K$ .

Sensitivity of  $T$  with respect to  $b$  evaluated at the nominal values  $b = 4$  and  $h = 0.5$  is

$$S_b^T = \frac{\partial T}{\partial b} \frac{b}{T} = \frac{s + 1}{s + 1 + Kbh} = \frac{s + 1}{s + 1 + 2K}$$

Similarly, sensitivity of  $T$  with respect to  $h$  evaluated at the nominal values  $b = 4$  and  $h = 0.5$  is

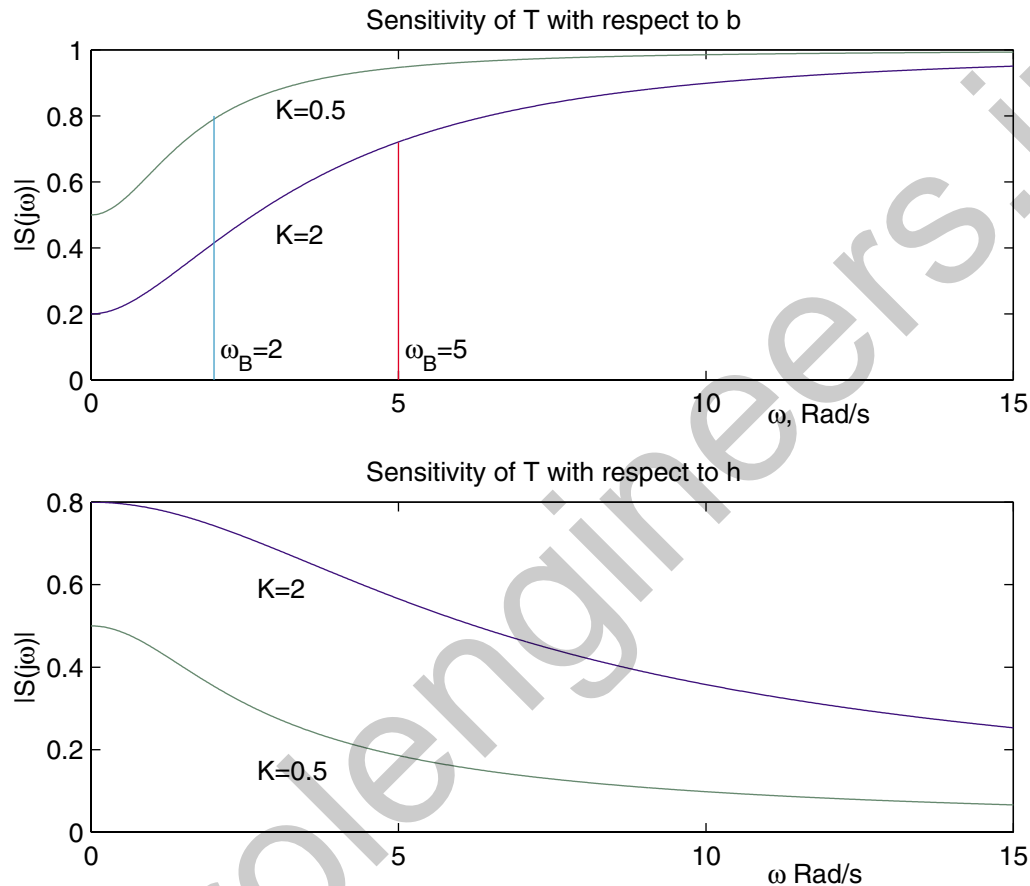
$$S_h^T = \frac{\partial T}{\partial h} \frac{h}{T} = \frac{-Kbh}{s + 1 + Kbh} = \frac{-2K}{s + 1 + 2K}$$

The following program computes and plots  $|S_b^T(j\omega)|$  and  $|S_h^T(j\omega)|$  for  $K = 2$  and  $K = 0.5$ .

```
k1=2;k2=0.5;
num=[1 1];
den1=[1 1+2*k1];
den2=[1 1+2*k2];
w=0:.02:15;
STb1 = bode(num,den1,w); % Magnitude of S(jw)
STb2 = bode(num,den2,w); % Magnitude of S(jw)
subplot(2,1,1),plot(w,STb1, w, STb2), grid
title('Sensitivity of T with respect to b')
xlabel('\omega, Rad/s'), ylabel('|S(j\omega)|')
text(3, 0.83,'K=0.5'),text(3,.44,'K=2')
num1 = -2*k1; num2 = -2*k2;
STb1 = bode(num1,den1,w); % Magnitude of S(jw)
STb2 = bode(num2,den2,w); % Magnitude of S(jw)
subplot(2,1,2),plot(w,STb1, w, STb2), grid
text(11,-.1,'\omega Rad/s'),text(2.7,.15,'K=0.5'),text(2.7,.59,'K=2')
title('Sensitivity of T with respect to h')
xlabel('\omega, Rad/s'), ylabel('|S(j\omega)|')
subplot(111)
```

From Figure 5.3 we can see that sensitivity of the system to  $b$  decreases with increasing open-loop gain  $K$ ; whereas the sensitivity to  $h$  increases with increasing  $K$ . It is clear that sensitivity to  $b$  increases rapidly outside the system bandwidth  $\omega_B$ , and sensitivity to  $h$  increases for frequency below bandwidth. More applications of *sensitivity*

in the design and analysis of control system will be presented in later chapters. The result is shown in Figure 5.3.



**FIGURE 5.3**  
Example 5.5.

## 5.4 Steady-State Error and System Type

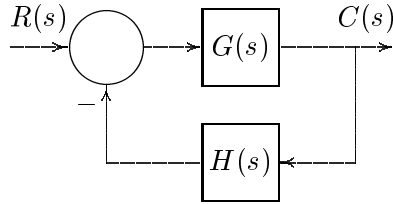
In addition to being stable, a control system is also expected to meet a specified performance requirement when it is commanded by a set-point change or disturbed by an external force. The performance of the control system is judged not only by the transient response, but also by *steady-state error*. The steady-state error is the error as the transient response decays leaving only the continuous response. High loop gains, in addition to sensitivity reduction, will also reduce the steady-state error. The steady-state error for a control system is classified according to its response characteristics to a polynomial input. A system may have no steady-state error to a step input, but the same system may exhibit nonzero steady-state error to a ramp input. This depends on

the type of the open-loop transfer function.

Consider the system shown in Figure 5.4. The closed-loop transfer function is

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s)H(s)} \quad (5.4)$$

The error of the closed-loop system is



**FIGURE 5.4**

Nonunity feedback control system.

$$E(s) = R(s) - H(s)C(s) = \frac{1}{1 + G(s)H(s)} R(s) \quad (5.5)$$

Using the final-value theorem, we have

$$e_{ss} = \lim_{s \rightarrow 0} \frac{sR(s)}{1 + G(s)H(s)} \quad (5.6)$$

For the polynomial inputs, such as step, ramp and parabolas, the steady-state error from the above equation will be

**Unit step input**

$$e_{ss} = \frac{1}{1 + \lim_{s \rightarrow 0} G(s)H(s)} = \frac{1}{1 + K_p} \quad (5.7)$$

**Unit ramp input**

$$e_{ss} = \frac{1}{\lim_{s \rightarrow 0} sG(s)H(s)} = \frac{1}{K_v} \quad (5.8)$$

**Unit parabolic input**

$$e_{ss} = \frac{1}{\lim_{s \rightarrow 0} s^2 G(s)H(s)} = \frac{1}{K_a} \quad (5.9)$$

In order to define the *system type*, the general open loop transfer function is written in the following form:

$$G(s)H(s) = \frac{K(1 + T_1 s)(1 + T_2 s) \dots (1 + T_m s)}{s^j (1 + T_a s)(1 + T_b s) \dots (1 + T_n s)} \quad (5.10)$$

The *type* of feedback control system refers to the *order* of the pole of  $G(s)H(s)$  at  $s = 0$ . The steady-state error for type 0, 1 and 2 systems are tabulated in Table 5.1.

**Table 5.1 Steady-state Errors**

	$R(s)$	$\frac{1}{s}$	$\frac{1}{s^2}$	$\frac{1}{s^3}$	
$j$					
0		$\frac{1}{1 + K_p}$	$\infty$	$\infty$	$K_p = \lim_{s \rightarrow 0} G(s)H(s)$
1		0	$\frac{1}{K_v}$	$\infty$	$K_v = \lim_{s \rightarrow 0} sG(s)H(s)$
2		0	0	$\frac{1}{K_a}$	$K_a = \lim_{s \rightarrow 0} s^2G(s)H(s)$

Two functions, **errorzp(z,p,k)** and **errortf(num, den)**, are written for computation of system steady-state error due to typical inputs, namely unit step, unit ramp, and unit parabolic. **errorzp(z,p,k)** finds the steady-state error when the system is represented by the zeros, poles, and gain. **z** is a column vector containing the transfer function zeros, **p** is a column vector containing the poles, and **k** is the gain. If the numerator power,  $m$ , is less than the denominator power,  $n$ , then there are  $n - m$  zeros at infinity and vector **z** must be padded with  $(n - m)$  **inf**. **errortf(num, den)** finds the steady-state error when the transfer function is expressed as the ratio of two polynomials. These functions are demonstrated in Examples 5.6 and 5.7.

### Example 5.6

Determine the step, ramp and parabolic error constants and the steady-state error for the following transfer function.

$$G(s) = \frac{10(s + 4)}{s(s + 1)(s + 2)(s + 5)}$$

The following commands

```
k=10;
z = [-4; inf; inf; inf];
p = [0; -1; -2; -5];
errorzp(z,p,k)
```

result in

System type is 1

Error Constants:

Kp	Kv	Ka
$\infty$	4	0

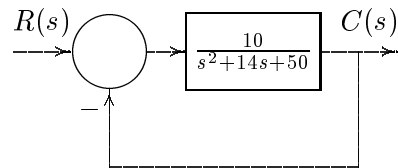
Steady-state Errors:

step	Ramp	Parabolic
0	0.25	$\infty$

### Example 5.7

Find the error constants and the steady-state error for the system shown in Figure 5.5.

The commands



**FIGURE 5.5**

System for Example 5.7.

```

num = 10;
den = [1 14 50];
error_tf(num, den)
    
```

result in

System type is 0

Error Constants:

K <sub>p</sub>	K <sub>v</sub>	K <sub>a</sub>
----------------	----------------	----------------

0.2	0	0
-----	---	---

Steady-state Errors:

step	Ramp	Parabolic
------	------	-----------

0.833	∞	∞
-------	---	---

---

## CHAPTER 6

---

### ROOT-LOCUS ANALYSIS AND DESIGN

This chapter deals with the *root-locus method* developed by W. R. Evans. The root-locus method enables us to find the closed-loop poles from the open-loop poles for all the values of the gain of the open-loop transfer function. The root-locus of a system is a plot of the roots of the system characteristic equation as the gain factor  $K$  is varied. Therefore, the designer can select a suitable gain factor to achieve the desired performance criteria. If the required performance cannot be achieved, a controller can be added to the system to alter the root-locus in the required manner.

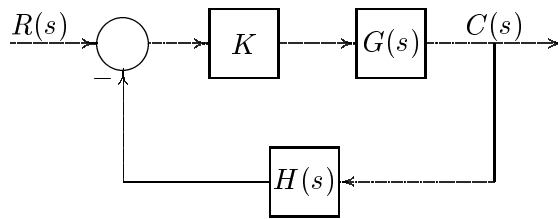
Given a transfer function of an open-loop control system, the *Control System Toolbox* function **rlocus(num, den)** produces a root-locus plot with the gain vector automatically determined. If the open-loop system is defined in state space, we use **rlocus(A, B, C, D)**. **rlocus(num, den, K)** or **rlocus(A, B, C, D, K)** uses the user-supplied gain vector  $K$ . If the above commands are invoked with the left hand arguments **[r, K]**, the matrix  $r$  and the gain vector  $K$  are returned, and we need to use **plot(r, 'r')** to obtain the plot. **rlocus** function is accurate, and we use it to obtain the root-locus. The command **axis('equal')** will make the  $x$ - and  $y$ -axis scaling factor the same. The command **rlocfind** finds the root locus gains for a given set of roots. **[K, Poles] = rlocfind(num, den)** is used for interactive gain selection from the root locus plot of a system generated by **rlocus**. **rlocfind** puts up a crosshair cursor in the graphics window which is used to select a pole location on an existing root locus. The root locus gain associated with this point is returned in  $K$  and all the system poles for this gain



are returned in POLES. A good knowledge of the characteristics of the root loci offers insights into the effects of adding poles and zeros to the system transfer function. It is important to know how to construct the root locus by hand, so we can design a simple system and be able to understand and develop the computer-generated loci. Therefore, the basic construction rules for sketching the root-locus are summarized below:

## 6.1 Root-Locus Method

Consider the feedback control system given in Figure 6.1.



**FIGURE 6.1**

Control system for root-locus.

The closed-loop transfer function of this system is

$$T(s) = \frac{C(s)}{R(s)} = \frac{KG(s)}{1 + KG(s)H(s)} \quad (6.1)$$

In general, the open-loop transfer function is given by

$$KG(s)H(s) = \frac{K(s + z_1)(s + z_2) \cdots (s + z_m)}{(s + p_1)(s + p_2) \cdots (s + p_n)} \quad (6.2)$$

where  $m$  is the number of finite zeros and  $n$  is the number of finite poles of the loop transfer function. If  $n > m$ , there are  $(n - m)$  zeros at infinity.

The characteristic equation of the closed-loop transfer function is

$$1 + KG(s)H(s) = 0 \quad (6.3)$$

Therefore

$$\frac{(s + p_1)(s + p_2) \cdots (s + p_n)}{(s + z_1)(s + z_2) \cdots (s + z_m)} = -K \quad (6.4)$$

From (6.4), it follows that for a point in the  $s$ -plane to be on the root-locus, when  $0 < K < \infty$ , it must satisfy the following two conditions.

$$K = \frac{\text{product of vector lengths from finite poles}}{\text{product of vector lengths from finite zeros}} \quad (6.5)$$

and

$$\sum \text{angles of zeros of } G(s)H(s) - \sum \text{angles of poles of } G(s)H(s) = r(180)^\circ, \quad (6.6)$$

where  $r = \pm 1, \pm 3, \pm 5, \dots$

## 6.2 Summary of General Rules for Constructing Root-Loci

1. **Number of loci.** For  $n > m$ , the number of loci, that is, the number of branches of the root-locus, is equal to the number of poles of the open-loop transfer function  $G(s)H(s)$ . The root-locus is symmetrical with respect to the real axis.
2. **Starting and ending points.** As  $K$  is increased from zero to infinity, the loci of the closed-loop poles originate from the open-loop poles ( $K = 0$ ), and proceed toward and terminate at the open-loop zeros, ( $K \rightarrow \infty$ ). Zeros tend to attract root-loci toward them and poles tend to repel them.
3. **Root-locus segments on the real axis.** For  $K > 0$ , root-loci occurs on a particular segment of the real axis if and only if there are an odd number of total poles and zeros of the open-loop transfer function laying to the right of that segment.
4. **Imaginary axis intersections.** Use Routh-Hurwitz criterion to determine  $j\omega$ -axis crossings of the root-locus points. Both the gain  $K$  and the value of  $\omega$  may be found from the Routh array.
5. **Asymptotes (For  $n \neq m$ ).** For most systems of interest,  $n$  is greater than or equal to  $m$ . For  $n > m$  there are  $(n - m)$  zeros at infinity, thus for  $0 < K < \infty$ ,  $(n - m)$  root-locus ends at zeros at infinity.

Root-locus points are asymptotic to straight lines with angles given by

$$\theta = \frac{r180^\circ}{n - m}, \quad r = \pm 1, \pm 3, \pm 5, \dots \quad (6.7)$$

as  $s$  approaches infinity. Table 6.1 should be helpful.

**Table 6.1 Angles of asymptotes**

$n - m$	Angles of asymptotes
0	no asymptotes
1	$180^\circ$
2	$\pm 90^\circ$
3	$180^\circ, \pm 60^\circ$
4	$\pm 45^\circ, \pm 135^\circ$

These straight lines emanate from a point  $s$  on the real axis specified by

$$\sigma_a = \frac{\sum \text{poles of } G(s)H(s) - \sum \text{zeros of } G(s)H(s)}{n - m} \quad (6.8)$$

6. **Angles of departure and arrival.** Assume a point  $s_\theta$  arbitrarily near the poles (for departure) or the zeros (for arrival) and then apply the fundamental angle relationship, (6.6) to obtain

$$\theta_d = \sum_i \theta_{zi} - \sum_i \theta_{pi} + r(180^\circ) \quad (6.9)$$

$$\theta_a = \sum_i \theta_{pi} - \sum_i \theta_{zi} + r(180^\circ) \quad (6.10)$$

where  $r = \pm 1, \pm 3, \dots$

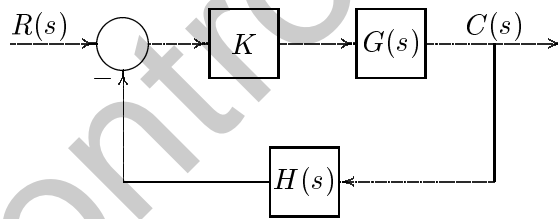
7. **Breakaway and re-entry points.** These are points on the real axis where two or more branches of the root-locus depart from or arrive at the real axis. Breakaway points may be determined by expressing the characteristic equation for the gain  $K$  as a function of  $s$ ,  $K = -1/G(s)H(s)$ , and then solving for the breakaway points  $s$  from

$$\left. \frac{dK(s)}{ds} \right|_{s=s_B} = 0 \quad (6.11)$$

The real roots of this equation which satisfy rule 3 are the breakaway or re-entry points. Root-loci for second-order systems occurs as straight lines, circles, or segments thereof.

### Example 6.1

Obtain the root-locus for  $K > 0$  for the simple motor position servo system shown in Figure 6.2.



**FIGURE 6.2**  
Control system for Example 6.1.

The open-loop transfer function is

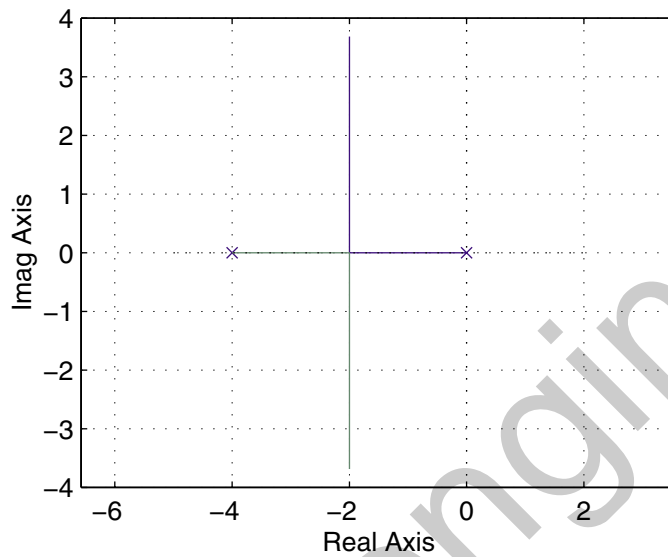
$$KG(s)H(s) = \frac{K}{s(s+4)} = \frac{K}{s^2 + 4s}$$

The *Control System Toolbox* function **rlocus(num, den, K)** is used to obtain the closed-loop root-locus for varying  $K$ .

```

num=1;
den=[1 4 0];
rlocus(num, den),grid;
axis('equal')
    
```

The first statement overrides the automatic scaling, and the second statement produces a graph with square aspect ratio. The results are shown in Figure 6.3.



**FIGURE 6.3**  
Root-locus for Example 6.1.

- The root-loci on the real axis are to the left of an odd number of finite poles and zeros.
- $n - m = 2$  zeros at infinity.
- Two asymptotes with angles  $\theta = \pm 90^\circ$ .
- The asymptotes intersect on the real axis at

$$\sigma_a = \frac{-4 - 0}{2} = -2$$

- Breakaway point on the real axis is given by

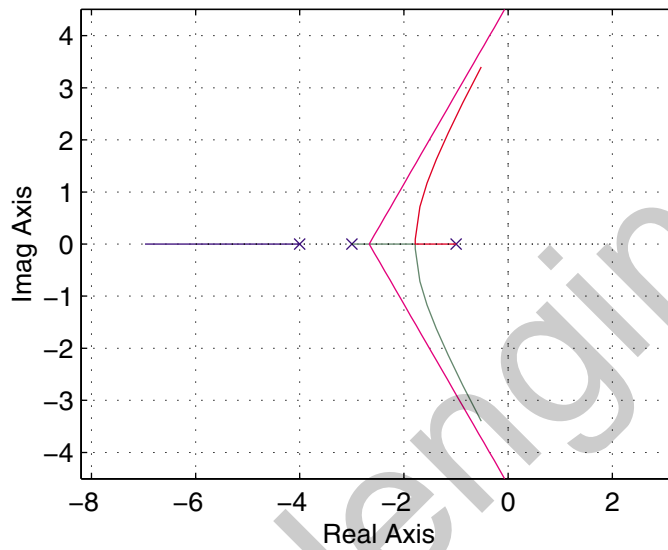
$$\frac{dK}{ds} = -\frac{d}{ds}(s^2 + 4s) = 0 \quad \text{i.e.} \quad s = -2$$

### Example 6.2

Obtain the root-locus for  $K > 0$  for a system whose open-loop transfer function is given below.

$$KG(s)H(s) = \frac{K}{(s+1)(s+3)(s+4)} = \frac{K}{s^3 + 8s^2 + 19s + 12}$$

Statements written to obtain root-locus are similar to Example 6.1. The results are given in Figure 6.4.



**FIGURE 6.4**  
Root-locus for Example 6.2.

- The root-loci on the real axis are to the left of an odd number of finite poles and zeros.
- $n - m = 3$  zeros at infinity.
- Three asymptotes with angles  $\theta = 180^\circ, \pm 60^\circ$ .
- The asymptotes intersect on the real axis at

$$\sigma_a = \frac{-4 - 3 - 1}{3} = -2.66$$

- Breakaway point on the real axis is given by

$$\frac{dK}{ds} = -\frac{d}{ds}(s^3 + 8s^2 + 19s + 12) = 0$$

## 110 6. Root-Locus Analysis and Design

The roots of this equation are  $s_1 = -3.55$  and  $s_2 = -1.78$ , but  $s_2 = -3.55$  is not part of the root-locus for  $K > 0$ , therefore the breakaway point is at  $s = -1.78$ .

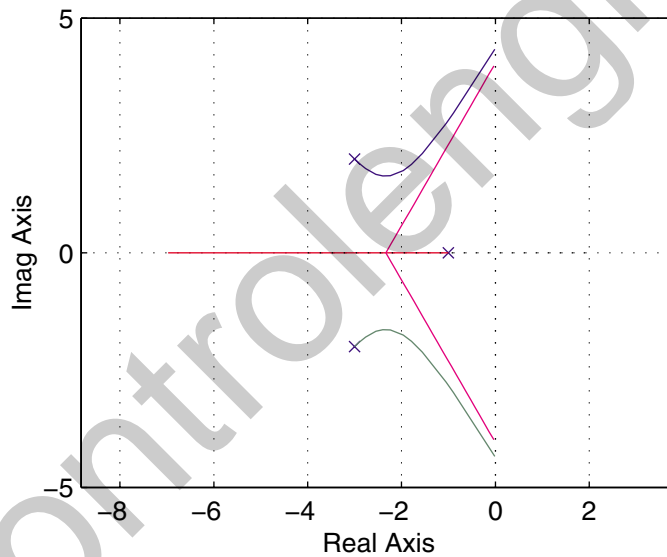
- The Routh array gives the location of the  $j\omega$ -axis crossing.

$$\begin{array}{cc|c}
 1 & 19 & \\
 8 & 12 + K & \\
 \hline
 140 - K & 0 & \\
 8 & & 
 \end{array}
 \quad
 \begin{array}{l}
 s = \pm j4.36 \\
 K = 140
 \end{array}$$

### Example 6.3

Obtain the root-locus for  $K > 0$  for a system whose open-loop transfer function is given below. The results are shown in Figure 6.5.

$$KG(s)H(s) = \frac{K}{(s+1)(s+3-j2)(s+3+j2)} = \frac{K}{s^3 + 7s^2 + 19s + 13}$$



**FIGURE 6.5**  
Root-locus for Example 6.3.

- The root-loci on the real axis are to the left of an odd number of poles and zeros.
- $n - m = 3$  zeros at infinity.
- Three asymptotes with angles  $\theta = 180^\circ, \pm 60^\circ$ .

- The asymptotes intersect on the real axis at

$$\sigma_a = \frac{-3 - 3 - 1}{3} = -2.33$$

- Breakaway point on the real axis is given by

$$\frac{dK}{ds} = -\frac{d}{ds}(s^3 + 7s^2 + 19s + 13) = 0$$

Roots are  $s_1, s_2 = -2.33 \pm j0.94$ , which shows no intersection with the real axis.

- The angles of departure from the complex poles are  $\theta_{d1} = 0 - (135 + 90) + 180 = -45^\circ$ , and  $\theta_{d2} = 0 - (45 + 90) + 180 = 45^\circ$ .
- The Routh array gives the location of the  $j\omega$ -axis crossing and the value of  $K$  at that point.

1	19	
7	$13 + K$	$s = \pm j4.36$
$\frac{120 - K}{7}$	0	$K = 120$

#### Example 6.4

Obtain the root-locus for  $K > 0$  for a system whose open-loop transfer function is given below. The results are shown in Figure 6.6.

$$KG(s)H(s) = \frac{K}{(s+1)(s+3-j1)(s+3+j1)} = \frac{K}{s^3 + 7s^2 + 16s + 10}$$

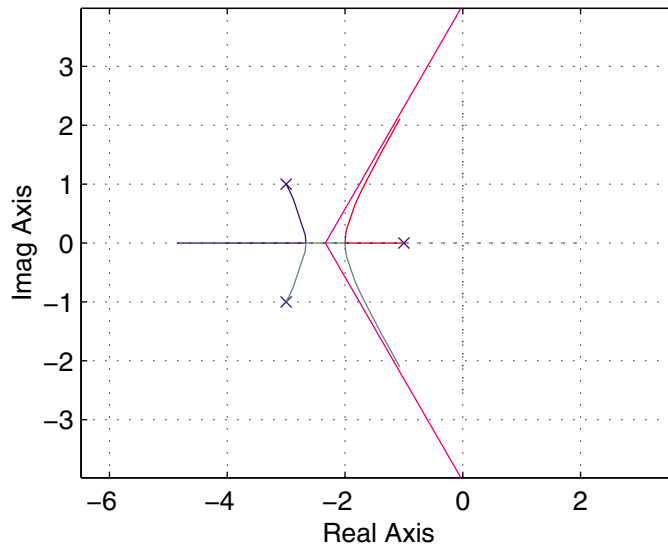
- The root-loci on the real axis are to the left of an odd number of poles and zeros.
- $n - m = 3$  zeros at infinity.
- Three asymptotes with angles  $\theta = 180^\circ, \pm 60^\circ$ .
- The asymptotes intersect on the real axis at

$$\sigma_a = \frac{-3 - 3 - 1}{3} = -2.33$$

- Breakaway points on the real axis are given by

$$\frac{dK}{ds} = -\frac{d}{ds}(s^3 + 7s^2 + 16s + 10) = 0$$

The roots of this equation are  $s_1 = -2.0$  and  $s_2 = -2.66$ . Both points lie on the root-locus for  $K > 0$ ; they are the breakaway point and re-entry point.



**FIGURE 6.6**  
Root-locus for Example 6.4.

- The angles of departure from the complex poles are  $\theta_{d1} = 0 - (153.43 + 90) + 180 = -63.43^\circ$ , and  $\theta_{d2} = 0 - (26.56 + 90) + 180 = 63.43^\circ$ .
- The Routh array gives the location of the  $j\omega$ -axis crossing.

1	16	
7	$10 + K$	$s = \pm j4$
$\frac{102 - K}{7}$	0	$K = 102$

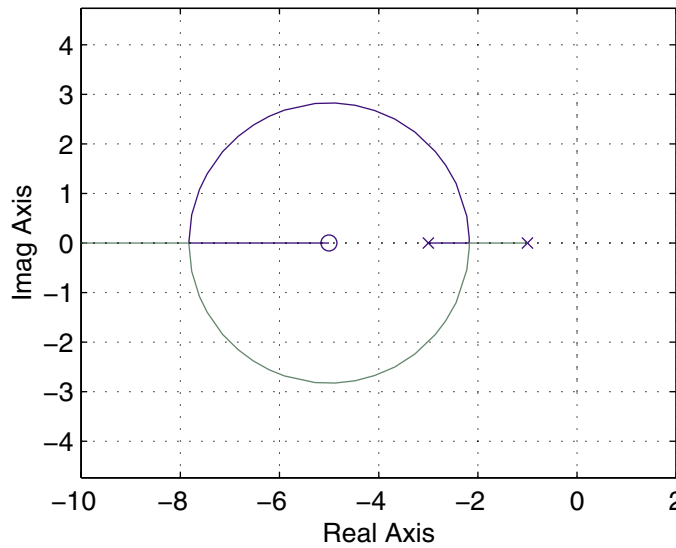
### Example 6.5

Obtain the root-locus for  $K > 0$  for a system whose open-loop transfer function is given below. The results are shown in Figure 6.7.

$$KG(s)H(s) = \frac{K(s + 5)}{(s + 1)(s + 3)} = \frac{K(s + 5)}{(s^2 + 4s + 3)}$$

- The root-loci on the real axis are to the left of an odd number of finite poles and zeros.
- $n - m = 1$  zero at infinity.
- One asymptote with angle  $\theta = 180^\circ$ .





**FIGURE 6.7**  
Root-locus for Example 6.5.

- Breakaway points on the real axis are given by

$$\frac{dK}{ds} = -\frac{d}{ds} \frac{(s^2 + 4s + 3)}{(s + 5)} = 0$$

The roots of this equation are  $s_1 = -2.17$  and  $s_2 = -7.83$ . Both points lie on the root-locus for  $K > 0$ ; therefore they are the breakaway point and re-entry point, respectively.

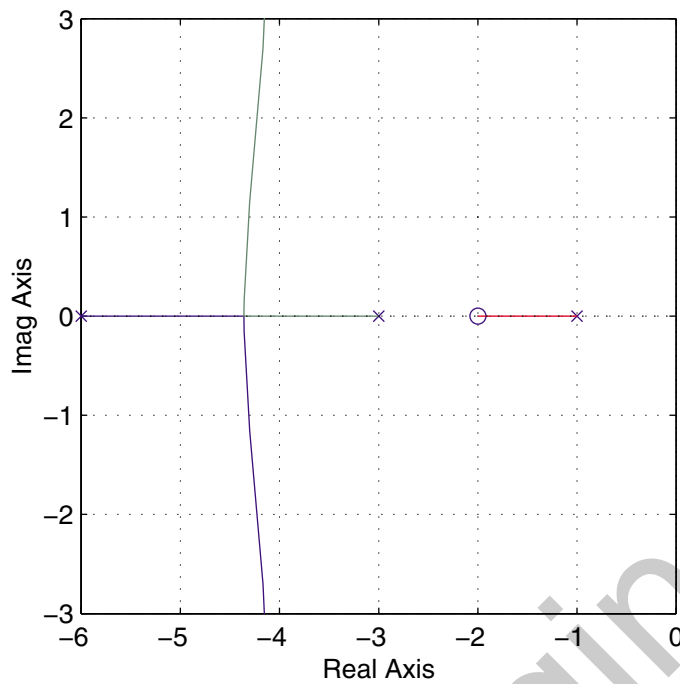
- No intersection on the  $j\omega$ -axis for  $K > 0$ .

### Example 6.6

Obtain the root-locus for  $K > 0$  for a system whose open-loop transfer function is given below. The results are shown in Figure 6.8.

$$KG(s)H(s) = \frac{K(s + 2)}{(s + 1)(s + 3)(s + 6)} = \frac{K(s + 2)}{s^3 + 10s^2 + 27s + 18}$$

- The root-loci on the real axis are to the left of an odd number of finite poles and zeros.
- $n - m = 2$  zeros at infinity.
- Two asymptotes with angles  $\theta = \pm 90^\circ$ .



**FIGURE 6.8**  
Root-locus for Example 6.6.

- The asymptotes intersect on the real axis at

$$\sigma_a = \frac{(-1 - 3 - 6) - (-2)}{2} = -4$$

- Breakaway point on the real axis is given by

$$\frac{dK}{ds} = -\frac{d}{ds} \frac{(s^3 + 10s^2 + 27s + 18)}{(s + 2)} = 0$$

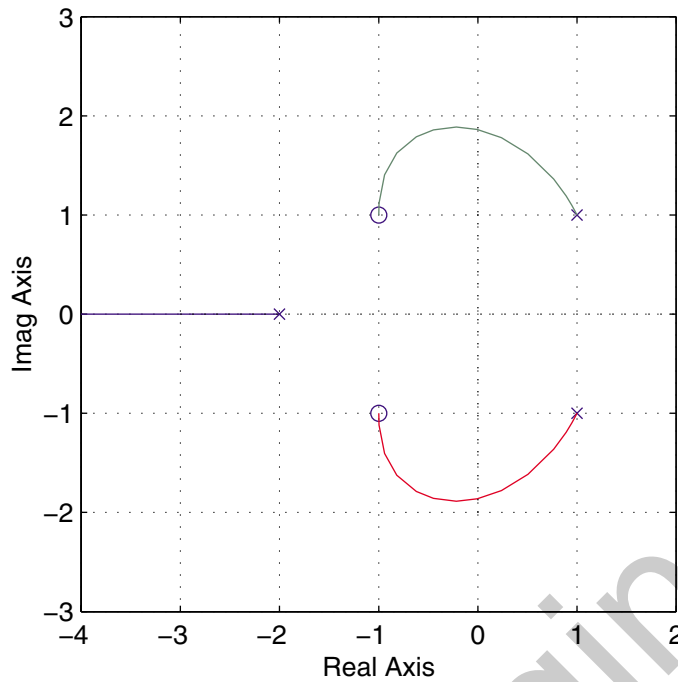
The roots of this equation are  $-4.36$  and  $-1.8 \pm j0.903$ ;  $s = -4.36$  lies on the root-locus for  $K > 0$ , therefore this is the breakaway point.

- No intersection on the  $j\omega$  - axis for  $K > 0$ .

### Example 6.7

Obtain the root-locus for  $K > 0$  for a system whose open-loop transfer function is given below. The results are shown in Figure 6.9.

$$KG(s)H(s) = \frac{K(s + 1 - j1)(s + 1 + j1)}{(s + 2)(s - 1 - j1)(s - 1 + j1)} = \frac{K(s^2 + 2s + 2)}{(s^3 - 2s + 4)}$$



**FIGURE 6.9**  
Root-locus for Example 6.7.

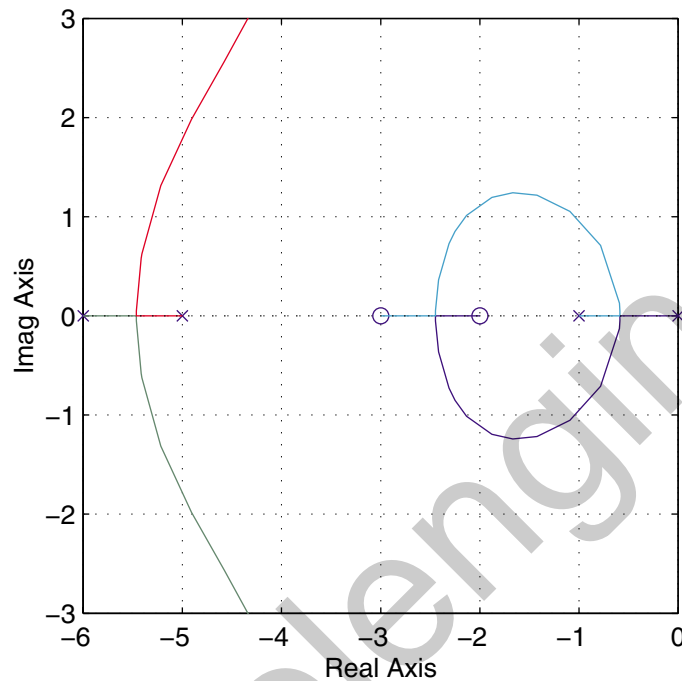
- The root-loci on the real axis are to the left of an odd number of finite poles and zeros.
- $n - m = 1$  zero at infinity.
- One asymptote with angle  $\theta = 180^\circ$ .
- No breakaway point on the real axis for  $K > 0$ .
- The angles of departure from the complex poles are  $\theta_{d1} = (0 + 45) - (90 + 18.43) + 180 = 116.56^\circ$ , and  $\theta_{d2} = (-45 + 0) - (-90 - 18.43) + 180 = -116.56^\circ$ .
- The angles of arrival from the complex zeros are  $\theta_{a1} = (180 + 135 + 45) - (90) + 180 = 90^\circ$ , and  $\theta_{a2} = (180 + 45 - 45) - (90) + 180 = -90^\circ$ .
- The Routh array gives the location of the  $j\omega$ -axis crossing and the value of  $K$  at that point.

$$\begin{array}{r|rr} 1 & 2K - 2 & \\ K & 2K + 4 & \\ \hline 2K^2 - 4K - 4 & 0 & \end{array} \quad \begin{array}{l} s = \pm j1.86 \\ K = 2.732 \end{array}$$

### Example 6.8

Obtain the root-locus for  $K > 0$  for a system whose open-loop transfer function is given below. The results are shown in Figure 6.10.

$$KG(s)H(s) = \frac{K(s+2)(s+3)}{s(s+1)(s+5)(s+6)} = \frac{K(s^2 + 5s + 6)}{s^4 + 12s^3 + 41s^2 + 30s}$$



**FIGURE 6.10**  
Root-locus for Example 6.8.

- The root-loci on the real axis are to the left of an odd number of finite poles and zeros.
- $n - m = 2$  zeros at infinity.
- Two asymptotes with angles  $\theta = \pm 90^\circ$ .
- The asymptotes intersect on the real axis at

$$\sigma_a = \frac{(0 - 1 - 5 - 6) - (-2 - 3)}{2} = -3.5$$

- Breakaway points on the real axis are given by

$$\frac{dK}{ds} = -\frac{d}{ds} \frac{(s^4 + 12s^3 + 41s^2 + 30s)}{(s^2 + 5s + 6)} = 0$$

The real roots of this equation are  $-0.586$ ,  $-5.46$  which are the breakaway points and  $-2.247$  which is the re-entry point.

- No intersection on the  $j\omega$ -axis for  $K > 0$ .

### 6.3 Root-Locus Design

The design specifications considered here are limited to those dealing with system accuracy and time-domain performance specifications. These performance specifications can be defined in terms of the desirable location of the dominant closed-loop poles.

The root-locus can be used to determine the value of the loop gain  $K$ , which results in a satisfactory closed-loop behavior. This is called the proportional controller and provides gradual response to deviations from the set point. There are practical limits as to how large the gain can be made. In fact, very high gains lead to instabilities. If the root-locus plot is such that the desired performance cannot be achieved by the adjustment of the gain, then it is necessary to reshape the root-loci by adding the additional controller  $G_c(s)$  to the open-loop transfer function.  $G_c(s)$  must be chosen so that the root-locus will pass through the proper region of the  $s$ -plane.

The proportional controller ( $P$ ) has no sense of time, and its action is determined by the present value of the error. An appropriate controller must make corrections based on the past and future values. This can be accomplished by combining proportional with integral action ( $PI$ ) or proportional with derivative action ( $PD$ ). There is also a proportional-plus-integral-plus-derivative controller ( $PID$ ).

$$G_c(s) = K_P + \frac{K_I}{s} + K_D s \quad (6.12)$$

The ideal integral and differential compensators require the use of active amplifiers.

Other compensators which can be realized with only passive network elements are lead, lag, and lead-lag compensators. A first-order compensator having a single zero and pole in its transfer function is

$$G_c(s) = \frac{K_c(s + z_0)}{s + p_0} \quad (6.13)$$

Several functions are developed for the root-locus design. These are

Function	Controller
<code>[numopen, denopen, denclsd] = pcomp(num, den, ζ)</code>	Proportional
<code>[numopen, denopen, denclsd] = phlead(num, den, s<sub>1</sub>)</code>	Phase-Lead
<code>[numopen, denopen, denclsd] = phlag(num, den, ζ)</code>	Phase-Lag
<code>[numopen, denopen, denclsd] = pdcomp(num, den, s<sub>1</sub>)</code>	PD
<code>[numopen, denopen, denclsd] = picomp(num, den, s<sub>1</sub>)</code>	PI
<code>[numopen, denopen, denclsd] = pidcomp(num, den, s<sub>1</sub>)</code>	PID

Alternatively, the function `[numopen, denopen, denclsd] = rldesign(num, den, s1)` allows the user to select any of the above controller designs.  $s_1 = \sigma + j\omega$  is a desired pole of the closed-loop transfer function, except for the **pcomp** and **phlag** controllers where  $\zeta$ , the damping ratio of the dominant poles, is substituted for  $s_1$ . **num** and **den** are row vectors of polynomial coefficients of the uncompensated open-loop plant transfer function. The function **phlead(num, den, s1)** may also be used to design phase-lag controllers. To do this, the desired pole location  $s_1$  must be assumed slightly to the right of the uncompensated pole position. The function returns the open-loop and closed-loop numerators and denominators of the compensated system transfer function.

## 6.4 Gain Factor Compensation or P Controller

The proportional controller is a pure gain controller. The design is accomplished by choosing a value of  $K_0$  which results in a satisfactory transient response. The developed function `[numopen, denopen, denclsd] = rldesign(num, den, ζ)` displays six options for root-locus design. For a proportional controller, option 1 must be selected. This option calls upon the function `[numopen, denopen, denclsd] = pcomp(num, den, ζ)`. The function computes the required gain  $K_0$ , that will result in a desirable damping ratio in the step response.

### Example 6.9

Obtain the gain  $K_0$  of a proportional controller for the system with the open-loop transfer function

$$G_p(s) = \frac{1}{s(s+1)(s+4)}$$

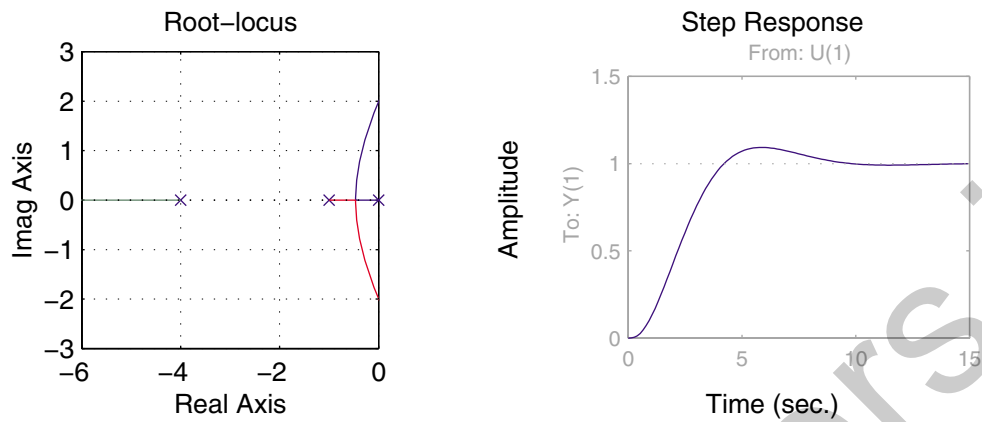
such that the damping ratio of the dominant poles will be equal to 0.6. Obtain root-locus, step response and the time-domain specifications for the compensated system.

The following commands

```
num=1;
den=[1 5 4 0];
zeta=0.6;
[numopen, denopen, denclsd]=rldesign(num,den,zeta); % compensated
% open-loop & closed-loop transfer function

subplot(221), rlocus(numopen, denopen), grid % Comp. Root-locus
axis('equal'), axis([-6 0 -3 3]), title('Root-locus')
subplot(222), step(numopen, denclsd), grid % Comp. step response
timespec(numopen, denclsd); % Time-domain specifications
subplot(111)
```

result in



**FIGURE 6.11**  
Root-locus plot and the step response for Example 6.9.

Compensator type	Enter
Gain compensation	1
Phase-lead (or phase-lag )	2
Phase-lag (Approximate $K = K_0/K_c$ )	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice → 1

Controller gain:  $K_0 = 2.05$

Compensated open-loop  
Transfer function:  
2.05

-----  
 $s^3 + 5s^2 + 4s$

Compensated closed-loop  
Transfer function:  
2.05

-----  
 $s^3 + 5s^2 + 4s + 2.05$

Roots of the compensated characteristic equation:  
-4.1563  
-0.4219 + 0.5615i  
-0.4219 - 0.5615i

Peak time = 5.855  
Rise time = 2.702

Percent overshoot = 9.2856

Settling time = 8.699

The results are shown in Figure 6.11.

From the above results the controller gain is  $K_0 = 2.05$ . This gain will result in the velocity error constant of  $K_v = 2.05/4 = 0.5125$ . Thus, the steady-state error due to a ramp input is  $e_{ss} = 1/K_v = 1.95$ . The compensated closed-loop transfer function is

$$\frac{C(s)}{R(s)} = \frac{2.05}{s^3 + 5s^2 + 4s + 2.05}$$

## 6.5 Phase-Lead Design

In (6.13) the compensator is a high-pass filter or phase-lead if  $p_0 > z_0$ . The phase-lead network contributes a positive angle to the root-locus angle criterion of (6.6) and tends to shift the root-locus of the plant toward the left in the  $s$ -plane. The lead network acts mainly to modify the dynamic response to raise bandwidth and to increase the speed of response. In a sense, a lead network approximates derivative control. If  $p_0 < z_0$ , the compensator is a low-pass filter or phase-lag. The phase-lag compensator adds a negative angle to the angle criterion and tends to shift the root-locus to the right in the  $s$ -plane. The compensator angle must be small to maintain the stability of the system. The lag network is usually used to raise the low-frequency gain and thus to improve the steady-state accuracy of the system. The lag network is an approximate integral control.

The DC gain of the compensator is

$$a_0 = G_c(0) = \frac{K_c z_0}{p_0} \quad (6.14)$$

For a given desired location of a closed-loop pole  $s_1$ , the design can be accomplished by trial and error. Select a proper value of  $z_0$  and use the angle criterion of (6.6) to determine  $p_0$ . Then, the gain  $K_c$  is obtained by applying the magnitude criterion of (6.5). Alternatively, if the compensator DC gain,  $a_0 = (K_c z_0)/p_0$ , is specified, then for a given location of the closed-loop pole

$$s_1 = |s_1| \angle \beta \quad (6.15)$$

$z_0$  and  $p_0$  are obtained such that the equation

$$1 + G_c(s_1)G_p(s_1) = 0 \quad (6.16)$$

is satisfied. It can be shown that the above parameters are found from the following equations [12].

$$z_0 = \frac{a_0}{a_1}, \quad p_0 = \frac{1}{b_1} \quad \text{and} \quad K_c = \frac{a_0 p_0}{z_0} \quad (6.17)$$



where

$$\begin{aligned} a_1 &= \frac{\sin \beta + a_0 M \sin(\beta - \psi)}{|s_1| M \sin \psi} \\ b_1 &= -\frac{\sin(\beta + \psi) + a_0 M \sin \beta}{|s_1| \sin \psi} \end{aligned} \quad (6.18)$$

where  $M$  and  $\psi$  are the magnitude and phase angle of the open-loop plant transfer function evaluated at  $s_1$ , i.e.,

$$G_p(s_1) = M \angle \psi \quad (6.19)$$

For the case that  $\psi$  is either  $0^\circ$  or  $180^\circ$ , (6.18) is given by

$$a_1 |s_1| \cos \beta \pm \frac{b_1 |s_1|}{M} \cos \beta \pm \frac{1}{M} + a_0 = 0 \quad (6.20)$$

where the plus sign applies for  $\psi = 0^\circ$  and the minus sign applies for  $\psi = 180^\circ$ . For this case the zero of the compensator must also be assigned.

Based on the above equations, the function **[numopen, denopen, denclsd] = phlead (num, den,  $s_1$ )** is developed for the phase-lead controller design. **num** and **den** are the numerator and denominator of the polynomial coefficients of the open-loop plant transfer function and  $s_1$  is the desired dominant closed-loop pole. The user enters the compensator DC gain. The function obtains the controller transfer function and roots of the compensated characteristic equation. Also, the function returns the open-loop and closed-loop numerators and denominators of the compensated system transfer function.

### Example 6.10

The system of Example 6.9 whose open-loop transfer function is

$$G_p(s) = \frac{1}{s(s+1)(s+4)}$$

is required to have a faster response and a smaller steady-state error due to a ramp input. A phase-lead compensator is to be designed to meet the following specifications.

- Time constant  $\tau = 1/\zeta\omega_n = 0.6667$  sec.
- Damping ratio  $\zeta = 0.6$ .
- Steady-state error due to a unit ramp input  $e_{ss} = 0.5$ .

Obtain root-locus, step response, and the time-domain specifications for the compensated system.

From the first two specifications  $\zeta\omega_n = 1/\tau = 1.5$  and  $\theta = \cos^{-1} 0.6 = 53.13^\circ$ . Thus, the required closed-loop pole is  $s_1 = -1.5 + j2$ . The third specification requires

$$e_{ss} = \frac{1}{K_v} = 0.5$$

or

$$K_v = 2$$

where

$$K_v = \lim_{s \rightarrow 0} s \frac{K_c(s + z_0)}{(s + p_0)} \frac{1}{s(s + 1)(s + 4)} = \frac{K_c z_0}{p_0} \frac{1}{4}$$

Therefore, the compensator gain required is  $a_0 = (k_c z_0)/p_0 = 4 \times 2 = 8$ . The function **[numopen, denopen, denclsd] = rldesign(num, den, s1)** with option 2 is used for the phase-lead design. The following commands

```

num=1;
den=[1 5 4 0];
j=sqrt(-1);
s1=-1.5+j*2;
[numopen, denopen, denclsd]=rldesign(num, den, s1); %compensated
                                % open-loop & closed-loop transfer function
subplot(2,2,1), rlocus(numopen, denopen), grid %Comp.Root-locus
subplot(2,2,2), step(numopen, denclsd), grid % Comp. step resp
timespec(numopen, denclsd); % Time-domain specifications
subplot(111)
    
```

Compensator type	Enter
Gain compensation	1
Phase-lead (or phase-lag )	2
Phase-lag (Approximate $K = K_0/K_c$ )	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice → 2

Enter the compensator DC Gain → 8

$G_c(0) = 8$ ,  $G_c = 82.2812(s + 1.11407)/(s + 11.4583)$

Compensated open-loop

Transfer function:

$$82.28 s + 91.67$$

$$s^4 + 16.46 s^3 + 61.29 s^2 + 45.83 s$$

Compensated closed-loop

Transfer function:

$$82.28 s + 91.67$$

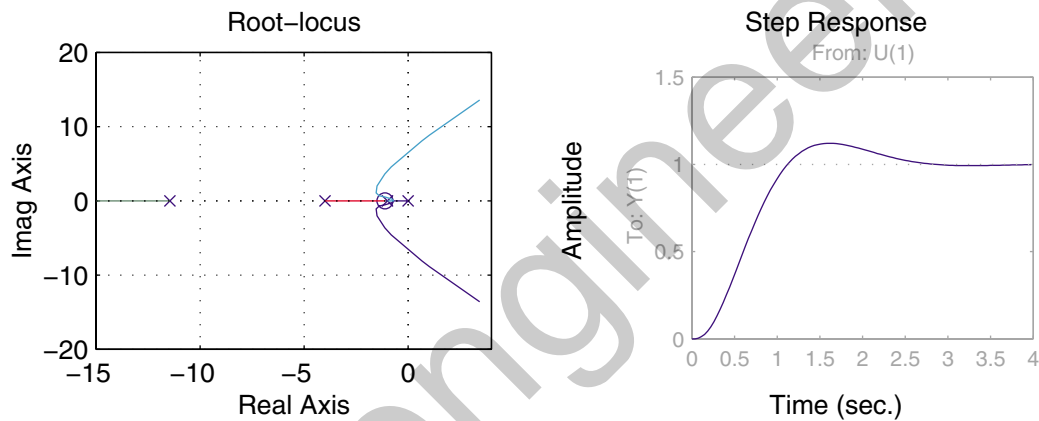
$$s^4 + 16.46 s^3 + 61.29 s^2 + 128.1 s + 91.67$$

Roots of the compensated characteristic equation:

-12.2623  
 -1.5000 + 2.0000i  
 -1.5000 - 2.0000i  
 -1.1961

Peak time = 1.613                      Percent overshoot = 12.083  
 Rise time = 0.710  
 Settling time = 2.533

The results are shown in Figure 6.12.



**FIGURE 6.12**  
Root-locus plot and the step response for Example 6.10.

From the above results, the compensated close-loop transfer function is

$$\frac{C(s)}{R(s)} = \frac{82.28(s + 1.114)}{s^4 + 16.46s^3 + 61.3s^2 + 128.1s + 91.66}$$

## 6.6 Phase-Lag Design

In the phase-lag control, the poles and zeros of the controller are placed very close together, and the combination is located relatively close to the origin of the  $s$ -plane. Thus, the root-loci in the compensated system are shifted only slightly from their original locations. Hence, the phase-lag compensator is used when the system transient response is satisfactory but requires a reduction in the steady-state error. The function **[numopen, denopen, denclsd] = phlead(num, den,  $s_1$ )** can be used for phase-lag compensation by specifying the desired pole  $s_1$  slightly to the right of the uncompensated pole location. Alternatively, phase-lag compensation can be obtained by assuming a DC gain of unity for the compensator based on the following approximate method.

$$a_0 = G_c(0) = \frac{K_c z_0}{p_0} = 1 \quad (6.21)$$

Therefore,

$$K_c = \frac{p_0}{z_0}, \quad \text{since } p_0 < z_0 \text{ then } K_c < 1 \quad (6.22)$$

If  $K_0$  is the gain required for the desired closed-loop pole  $s_1$ , then from (6.3)

$$K_0 = -\frac{1}{G_p(s_1)} \quad (6.23)$$

If we place the pole and zero of the lag compensator very close to each other with their magnitude much smaller than  $s_1$ , then

$$G_c(s_1) = \frac{K_c(s + z_0)}{s + p_0} \simeq K_c \quad (6.24)$$

Now, the gain  $K$  required to place a closed-loop pole at approximately  $s_1$  is given by

$$K = -\frac{1}{G_c(s_1)G_p(s_1)} \simeq -\frac{1}{K_c G_p(s_1)} \simeq \frac{K_0}{K_c} \quad (6.25)$$

Since  $K_c < 1$ , then  $K > K_0$ . Next, select the compensator zero  $z_0$ , arbitrarily small. Then from (6.21) the compensator pole is

$$p_0 = K_0 z_0 \quad (6.26)$$

The compensated system transfer function is then given by

$$K G_p G_c = K K_c \frac{s + z_0}{s + p_0} G_p \quad (6.27)$$

The function **[numopen, denopen, dencldsd] = phlag(num, den, ζ)** is developed for the phase-lag controller design, based on the above approximate criteria. **num** and **den** are the numerator and denominator of the polynomial coefficients of the open-loop plant transfer function, and  $\zeta$  is the desired damping ratio of the dominant closed-loop poles. The user is prompted to enter the gain  $K$  to realize the steady-state error and the compensator zero,  $z_0$ . The function obtains the controller transfer function and roots of the compensated characteristic equation. Also, the function returns the open-loop and closed-loop numerators and denominators of the compensated system transfer function.

A lag-lead controller may be obtained by appropriately combining a lag and a lead network in series.

### Example 6.11

The system of Example 6.9 whose open-loop transfer function is

$$K G_p(s) = \frac{K}{s(s+1)(s+4)}$$

is required to have approximately the same dominant closed-loop pole locations and the same damping ratio ( $\zeta = 0.6$ ) as in Example 6.9. Design a phase-lag compensator such that the steady-state error due to a unit ramp input,  $e_{ss}$ , will be equal to 0.125.

Obtain root-locus, step response, and the time-domain specifications for the compensated system.

The gain  $K$  which results in  $e_{ss} = 0.125$  is given by

$$K_v = \frac{1}{e_{ss}} = 8 = \lim_{s \rightarrow 0} s \frac{K}{s(s+1)(s+4)}$$

Thus the gain to realize the steady-state error specification is  $K = 32$ .

The function `[numopen, denopen, denclsd] = rldesign(num, den,  $\zeta$ )` with option 3 is used to obtain the parameters of the phase-lag compensator. Choose a small value for the compensator zero, e.g.,  $z_0 = 0.1$ .

The following commands

```
num=1;
den=[1 5 4 0];
zeta=0.6;
[numopen, denopen, denclsd]=rldesign(num,den,zeta); % compensated
% open-loop & closed-loop transfer function
subplot(221), rlocus(numopen, denopen), grid % Comp. Root-locus
subplot(222), step(numopen, denclsd), grid % Comp. step response
timespec(numopen, denclsd); % Time-domain specifications
subplot(111)
```

result in

Compensator type	Enter
Gain compensation	1
Phase-lead (or phase-lag )	2
Phase-lag (Approximate $K = K_0/K_c$ )	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice  $\rightarrow 3$

Enter gain  $K$  required for the steady-state error specification  $\rightarrow 32$

Enter magnitude of the compensator zero  $\rightarrow 0.1$

Gain for the desired closed-loop poles = 2.05

Gain for the desired steady-state response  $K = 32$

$$G_c(0) = 1, \quad G_c = 0.0640625(s + 0.1)/(s + 0.00640625)$$

## 126 6. Root-Locus Analysis and Design

Compensated open-loop

Transfer function:

$$2.05 s + 0.205$$

$$s^4 + 5.006 s^3 + 4.032 s^2 + 0.02562 s$$

Compensated closed-loop

Transfer function:

$$2.05 s + 0.205$$

$$s^4 + 5.006 s^3 + 4.032 s^2 + 2.076 s + 0.205$$

Roots of the compensated characteristic equation:

$$-4.1530$$

$$-0.3646 + 0.5142i$$

$$-0.3646 - 0.5142i$$

$$-0.1242$$

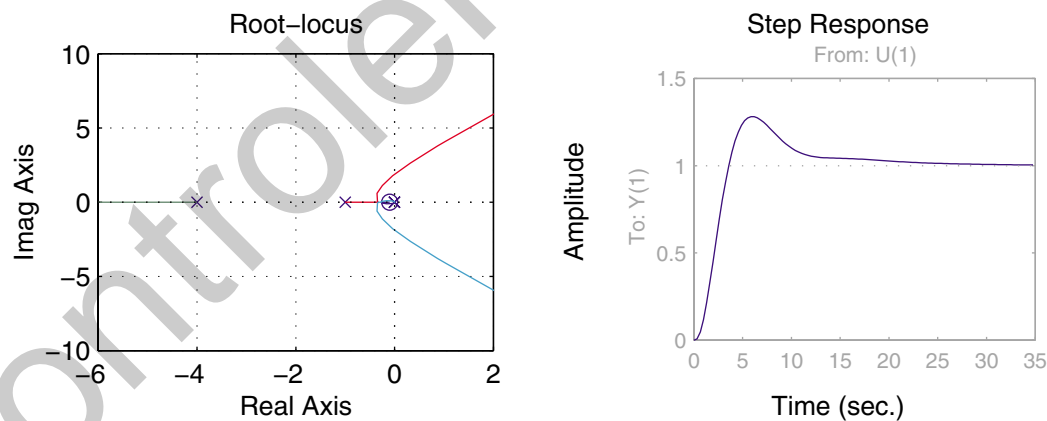
$$\text{Peak time} = 6.037$$

$$\text{Percent overshoot} = 28.194$$

$$\text{Rise time} = 2.334$$

$$\text{Settling time} = 22.054$$

The results are shown in Figure 6.13. We can see that the complex poles are located



**FIGURE 6.13**

Root-locus plot and the step response for Example 6.11.

approximately in the same location as in Example 6.9. The steady-state error is greatly reduced, but the percent overshoot is increased. The compensated closed-loop transfer function is

$$\frac{C(s)}{R(s)} = \frac{2.05(s + 0.1)}{s^4 + 5.006s^3 + 4.032s^2 + 2.075s + 0.205}$$

## 6.7 PID Design

One of the most common controllers available commercially is the PID controller. Different processes are suited to different combinations of proportional, integral, and derivative control. The control engineer's task is to adjust the three gain factors to arrive at an acceptable degree of error reduction simultaneously with acceptable dynamic response. For a desired location of the closed-loop pole  $s_1$ , as given by (6.15), the following equations [12] are obtained to satisfy (6.16).

$$K_P = \frac{-\sin(\beta + \psi)}{M \sin \beta} - \frac{2K_I \cos \beta}{|s_1|}$$

$$K_D = \frac{\sin \psi}{|s_1| M \sin \beta} + \frac{K_I}{|s_1|^2} \quad (6.28)$$

For PD or PI controllers, the appropriate gain is set to zero. The above equations can be used only for the complex pole  $s_1$ . For the case that  $s_1$  is real, the zero of the PD controller ( $z_0 = K_P/K_D$ ) and the zero of the PI controller ( $z_0 = K_I/K_P$ ) are specified and the corresponding gains to satisfy angle and magnitude criteria are obtained accordingly. For the PID design, the value of  $K_I$  to achieve a desired steady-state error is specified. Again, (6.28) is applied only for the complex pole  $s_1$ .

Based on the above equations, three functions called **[numopen, denopen, denclsd] = pdcomp(num, den,  $s_1$ )**, **[numopen, denopen, denclsd] = picomp(num, den,  $s_1$ )** and **[numopen, denopen, denclsd] = pidcomp(num, den,  $s_1$ )** are written for the PID controller design. **num** and **den** are the numerator and denominator of the polynomial coefficients of the open-loop plant transfer function and  $s_1$  is the desired dominant closed-loop pole. The function obtains the controller transfer function and roots of the compensated characteristic equation. Also, the function returns the open-loop and closed-loop numerators and denominators of the compensated system transfer function.

### 6.7.1 PD Controller

Here both the error and its derivative are used for control, and the compensator transfer function is

$$G_c(s) = K_P + K_D s = K_D \left( s + \frac{K_P}{K_D} \right) \quad (6.29)$$

From above, it can be seen that the PD controller is equivalent to the addition of a simple zero at  $s = -K_P/K_D$  to the open-loop transfer function which improves the transient response. From a different point of view, the PD controller may also be used to improve the steady-state error, because it anticipates large errors and attempts corrective action before they occur. The function **[numopen, denopen, denclsd] = rldesign(num, den,  $s_1$ )** with option 4 is used for the PD controller design. Its use is demonstrated in the following example.

### Example 6.12

For the system of Example 6.9, design a PD controller to place the dominant closed-loop poles at the same location as in Example 6.10, i.e.,  $s_1 = -1.5 \pm j2$ . Obtain the time-domain specifications for the compensated system.

The following commands

```

num=1;
den=[1 5 4 0];
j=sqrt(-1);
s1=-1.5+j*2;
[numopen, denopen, denclsd]=rldesign(num,den,s1); % Compensated
                                     % open-loop & closed-loop transfer function
timespec(numopen, denclsd);           % Time-domain specifications
result in
    
```

Compensator type	Enter
Gain compensation	1
Phase-lead (or phase-lag )	2
Phase-lag (Approximate $K = K_0/K_c$ )	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice → 4

$G_c = 12.5 + 8.25s$

Compensated open-loop  
Transfer function:

$$\frac{8.25 s + 12.5}{s^3 + 5 s^2 + 4 s}$$

Compensated closed-loop

Transfer function:

$$\frac{8.25 s + 12.5}{s^3 + 5 s^2 + 12.25 s + 12.5}$$

Roots of the compensated characteristic equation:

$$\begin{aligned} &-1.5000 + 2.0000i \\ &-1.5000 - 2.0000i \\ &-2.0000 \end{aligned}$$

Peak time = 1.393

Percent overshoot = 17.237

Rise time = 0.613

Settling time = 2.373



Thus the compensated open-loop transfer function is

$$G_p G_c = \frac{8.25s + 12.5}{s^3 + 5s^2 + 4s}$$

The steady-state error due to a ramp input is

$$e_{ss} = \frac{1}{k_v} = \frac{4}{12.5} = 0.32$$

We have approximately the same speed of response, but slightly higher percent overshoot compared to the phase-lead design of Example 6.10.

### 6.7.2 PI Controller

The integral of the error as well as the error itself is used for control, and the compensator transfer function is

$$G_c(s) = K_P + \frac{K_I}{s} = \frac{K_P(s + K_I/K_P)}{s} \quad (6.30)$$

The PI controller is common in process control or regulating systems. Integral control bases its corrective action on the cumulative error integrated over time. The controller increases the type of system by 1 and is used to reduce the steady-state errors. The function **[numopen, denopen, denclsd] = rldesign(num, den, s1)** with option 5 is used for the PI controller design. It is demonstrated in the following example.

#### Example 6.13

For the system of Example 6.9, design a PI controller that places the dominant closed-loop poles at the same location as in the phase-lag design of Example 6.11, i.e.,  $s_1 = -0.3646 + j0.5142$ .

The following commands

```
num = 1; den = [1 5 4 0];
s1 = -.3646+j*.5142;
[numopen, denopen, denclsd]=rldesign(num,den,s1);
timespec(numopen, denclsd); % Time-domain specifications.
```

result in

Compensator type	Enter
Gain compensation	1
Phase-lead (or phase-lag )	2
Phase-lag (Approximate $K = K_0/K_c$ )	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice → 5

$G_c = 2.05308 + 0.194057/s$

Row vectors of polynomial coefficients of the compensated system:

Open-loop num.	0	0	0	2.053	0.194
Open-loop den.	1	5	4	0	0
Closed-loop den	1	5	4	2.053	0.194

Roots of the compensated characteristic equation:

-4.1532  
 -0.3646 + 0.5142i  
 -0.3646 - 0.5142i  
 -0.1176

Peak time = 6.035	Percent overshoot = 28.70
Rise time = 2.295	
Settling time = 23.12	

Thus the compensated open-loop transfer function is

$$G_p G_c = \frac{2.053s + 0.194}{s^2(s^2 + 5s + 4)}$$

The PI controller increases the system type by 1. That is, we have a type 2 system and the steady-state error due to a ramp input is zero. Speed of response and percent overshoot are almost the same as the phase-lag design of Example 6.11.

### 6.7.3 PID Controller

The PID controller is used to improve the dynamic response as well as to reduce or eliminate the steady-state error. The function **[numopen, denopen, denclsd] = rldesign(num, den, s1)** with option 6 is used for the PID controller design. Its use is demonstrated in the following example.

#### Example 6.14

For the system of Example 6.9, design a PID controller to place the dominant closed-loop poles at the same location as in Example 6.10, i.e.,  $s_1 = -1.5 \pm j2$ . Obtain the time-domain specifications for the compensated system.

The following commands

```

num = 1;    den = [1 5 4 0];
s1=-1.5+j*2;
[numopen, denopen, denclsd]=rldesign(num,den,s1);           %Returns
    
```

```

                                % compensated open-loop & closed-loop
                                % transfer function
timespec(numopen, denclsd);    % Time-domain specifications
    
```

result in

Compensator type	Enter
Gain compensation	1
Phase-lead (or phase-lag )	2
Phase-lag (Approximate $K = K_0/K_c$ )	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice → 6

Enter the integrator gain KI → 0.1

$G_c = 12.548 + 0.1/s + 8.266s$

Compensated open-loop

Transfer function:

$8.266 s^2 + 12.55 s + 0.1$

-----

$s^4 + 5 s^3 + 4 s^2$

Compensated closed-loop

Transfer function:

$8.266 s^2 + 12.55 s + 0.1$

-----

$s^4 + 5 s^3 + 12.27 s^2 + 12.55 s + 0.1$

Roots of the compensated characteristic equation:

-1.5000 + 2.0000i

-1.5000 - 2.0000i

-1.9920

-0.0080

Peak time = 1.393

Percent overshoot = 17.555

Rise time = 0.613

Settling time = 2.400

Thus the compensated open-loop transfer function is

$$G_p G_c = \frac{8.266s^2 + 12.55s + 0.1}{s^2(s^2 + 5s + 4)}$$

The PID controller increases the system type by 1. That is, we have a type 2 system and the steady-state error due to a ramp input is zero. The transient response is also improved.

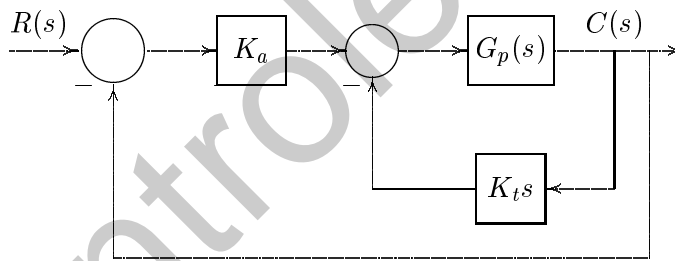
## 6.8 Minor-Loop Feedback Control

The controllers discussed in the preceding section are placed in cascade (series) with the system to be controlled. Another technique is to place the compensator in a minor feedback loop. The scheme is called *minor-loop feedback compensation* or *parallel compensation*. The selection of the compensation scheme depends largely on the control system, power level of the available signals, and the required design specifications. Feedback compensation is used to improve the system's tracking of a desired input and to yield a system that is less sensitive to disturbances and parameter variations. Generally, an amplifier may not be necessary since the controller is placed in a minor loop which is at a higher power level than cascade compensation.

The feedback compensation technique is primarily concerned with the addition of rate or acceleration feedback. In general, the PID controller or the phase-lead and phase-lag controllers discussed earlier can all be applied as minor-loop feedback controllers.

## 6.9 Rate Feedback or Tachometer-Feedback Control

The most common example of feedback compensation is *rate feedback*. The rate signal is commonly implemented with a tachometer. The rate feedback is used to damp out oscillations and it is usually used with a unity feedback as shown in Figure 6.14. The effects of rate feedback are similar to those of a cascade lead controller. It therefore has the effect of moving the root-locus to the left and improving the time response.



**FIGURE 6.14**  
Minor-loop feedback control with tachometer.

The closed-loop transfer function of the system is

$$\frac{C(s)}{R(s)} = \frac{K_a G_p(s)}{1 + (K_a + K_t s) G_p(s)} \quad (6.31)$$

and the characteristic equation is

$$1 + H(s) G_p(s) = 0 \quad (6.32)$$

where  $H(s) = K_a + K_t s$  is the same as the PD controller. For a desired location of the closed-loop pole  $s_1$  as given by (6.15), the following equations are obtained to

satisfy (6.32).

$$K_a = \frac{-\sin(\beta + \psi)}{M \sin \beta}$$

$$K_t = \frac{\sin \psi}{|s_1| M \sin \beta} \quad (6.33)$$

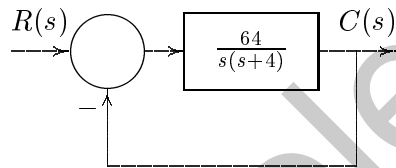
where  $M$  and  $\psi$  are the magnitude and phase angle of the open-loop plant transfer function evaluated at  $s_1$ .  $|s_1|$  and  $\beta$  are the magnitude and phase angle of the desired closed-loop pole.

In the rate feedback, no zero appears in the closed-loop transfer function. Therefore, the closed-loop step response will be more sluggish than the cascade PD controller.

Based on the above equations, function **[numopen, denopen, denclsd] = tachfdbk(num, den, s<sub>1</sub>)** is developed for the minor-loop feedback control with tachometer. Alternatively, the function **[numopen, denopen, denclsd] = fbdesign(num, den, s<sub>1</sub>)** with option 1 can be used for this design.  $s_1 = \sigma + j\omega$  is a desired pole of the closed-loop transfer function. **num** and **den** are the row vectors of polynomial coefficients of the uncompensated open-loop plant transfer function.

### Example 6.15

Determine the step response and the time-domain specifications for the system shown in Figure 6.15.



**FIGURE 6.15**  
Control system for Example 6.15.

The closed-loop transfer function is given by

$$\frac{C(s)}{R(s)} = \frac{64}{s^2 + 4s + 64}$$

The closed-loop poles are located at  $-2 + j7.746$  with a damping ratio of 0.25 and a time constant of  $\tau = 1/\zeta\omega_n = 2$  sec. The commands

```
num = 64; denc = [1 4 64]; timespec(num, denc);
```

result in the following time-domain specifications:

```
Peak time = 0.405          Percent overshoot = 44.339
Rise time = 0.160
Settling time = 1.760
```

The response is highly oscillatory with an overshoot of 44.4 percent.

A tachometer-feedback control, as shown in Figure 6.14, is used to improve the time response. Determine values of  $K_a$  and  $K_t$  which result in a system design having dominant poles with time-constant  $\tau = 0.125$  sec. and a damping ratio  $\zeta = 0.707$ . Obtain the time-domain specifications and plot the step response of the uncompensated and the compensated systems.

From the above specifications,  $\zeta\omega_n = 1/\tau = 8$  and  $\theta = \cos^{-1} 0.707 = 45^\circ$ . Thus the required closed-loop pole is  $s_1 = -8 + j8$ . The function **[numopen, denopen, denclsd] = fbdesign(num, den, s1)** with option 1 is used for the rate feedback compensation.

The following commands

```
num=64;
den=[1 4 0];dencu=[1 4 64];%uncompensated closed-loop denominator
s1=-8+j*8;
[numopen, denopen, denclsd]=fbdesign(num,den,s1); % compensated
                                % open-loop & closed-loop transfer function
t=0:.02:2;
c1=step(num, dencu, t); % Uncompensated system step response
timespec(num, dencu); % uncompensated time-domain specifications

c2=step(numopen, denclsd, t); % Compensated system step response
timespec(numopen,denclsd);%Compensated time-domain specifications
plot(t, c1,t, c2); xlabel('t, sec. '), ylabel('c(t)'), grid
text(.60, 1.2, 'Uncompensated response')
text(.205, 0.75, 'Compensated response')
```

result in

Compensator type	Enter
Rate feedback	1
Minor-loop design with passive network	2
To quit	0

Enter your choice  $\rightarrow$  1

$K_a = 2,$                        $K_t = 0.1875$  s

Compensated open-loop

Transfer function:

128

-----  
 $s^2 + 4$  s

Compensated closed-loop

Transfer function:

128

-----  
 $s^2 + 16$  s + 128

Roots of the compensated characteristic equation:

$$-8.0000 + 8.0000i$$

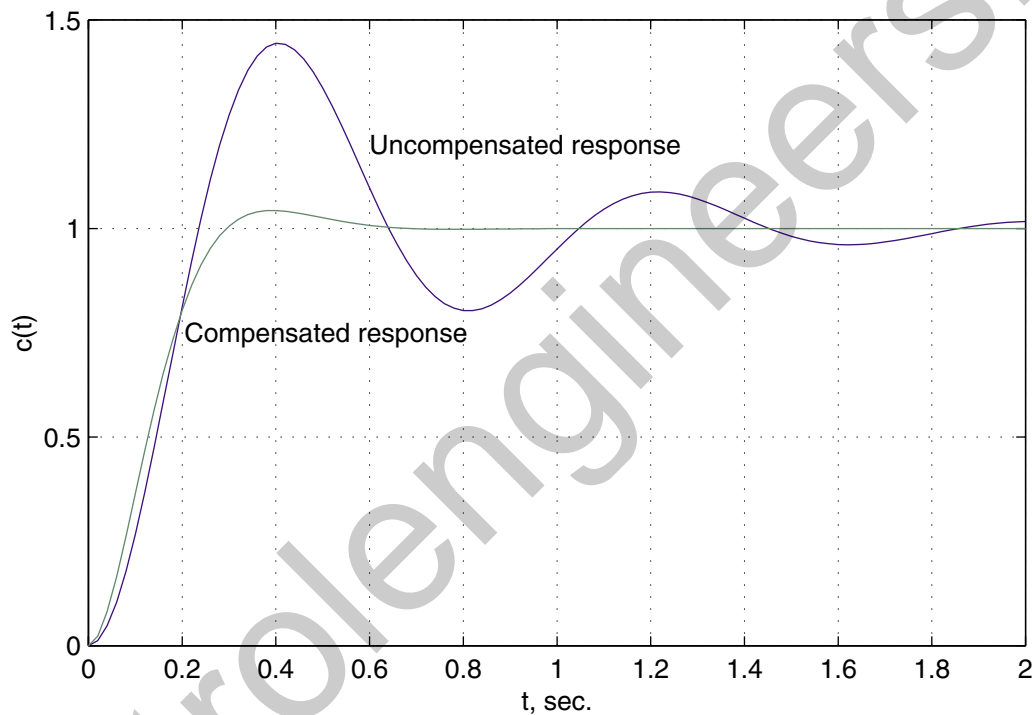
$$-8.0000 - 8.0000i$$

$$\text{Peak time} = 0.392$$

$$\text{Percent overshoot} = 4.321$$

$$\text{Rise time} = 0.190$$

$$\text{Settling time} = 0.526$$



**FIGURE 6.16**

Unit step response of the system in Example 6.15.

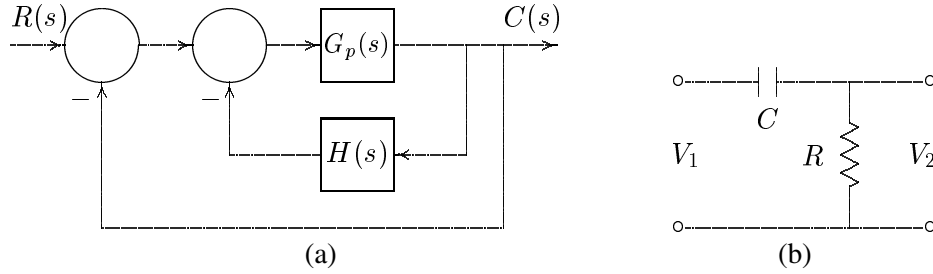
The results are shown in Figure 6.16.

## 6.10 Feedback Compensation using Passive Elements

To minimize cost, instead of using a tachometer, an RC network with phase-lead characteristics can be used in the minor feedback loop for compensation as shown in Figure 6.17.

The controller  $H(s)$  contains a simple RC network as shown in Figure 6.17 (b) with a transfer function given by

$$H(s) = \frac{a_1 s}{b_1 s + 1} \quad (6.34)$$



**FIGURE 6.17**  
Feedback compensation using passive elements.

The closed-loop characteristic equation of the system in Figure 6.17 is

$$1 + G_p(s) + H(s)G_p(s) = 0 \quad (6.35)$$

For a given location of the closed-loop pole

$$s_1 = |s_1| \angle \beta \quad (6.36)$$

$a_1$  and  $b_1$  are obtained such that (6.35) rewritten as

$$H(s)G_p(s) = -[1 + G_p(s)] = K \angle \gamma \quad (6.37)$$

is satisfied. It can be shown that the above parameters are found from the following equations:

$$\begin{aligned} a_1 &= \frac{\sin \beta}{|s_1| M \sin \theta} \\ b_1 &= -\frac{\sin(\theta - \beta)}{|s_1| \sin \theta} \end{aligned} \quad (6.38)$$

where  $M = |G_p(s_1)| / K$ ,  $\psi = \angle G_p(s_1)$ , and  $\theta = \gamma - \psi$ .

Based on the above equations, function **[numopen, denopen, denclsd] = pnetfdbk(num, den,  $s_1$ )** is developed for the minor-loop feedback control with a passive network. The function **[numopen, denopen, denclsd] = fbdesign(num, den,  $s_1$ )** with option 2 can be used for this design.  $s_1 = \sigma + j\omega$  is a desired pole of the closed-loop transfer function. **num** and **den** are the row vectors of polynomial coefficients of the uncompensated open-loop plant transfer function.

### Example 6.16

For the system of Example 6.15, instead of placing the tachometer in the minor-loop, use the passive network controller shown in Figure 6.17. Determine the controller parameters to place the dominant closed-loop poles at the same location as in Example 6.15, i.e.,  $s_1 = -8 + j8$ . Obtain the time-domain specifications for the compensated system.

The function **[numopen, denopen, denclsd] = fbdesign(num, den,  $s_1$ )** with option 2 is used.

The following commands



```

num = 64;
den = [1 4 0];
s1 = -8+j*8;
[numopen, denopen, denclsd]=fbdesign(num,den,s1);
                                % Returns compensated open-loop &
                                % closed-loop transfer function
t = 0:.02:2;
timespec(numopen,denclsd);      % Time-domain specifications
c = step(numopen, denclsd, t);  % Compensated system
plot(t, c); title('Step response'), grid % step response

```

result in

Compensator type	Enter
Rate feedback	1
Minor-loop design with passive network	2
To quit	0

Enter your choice → 2

$G_c = 2.5s / (s + 24)$

Compensated open-loop

Transfer function:

$64s + 1536$

-----  
 $s^3 + 28s^2 + 256s$

Compensated closed-loop

Transfer function:

$64s + 1536$

-----  
 $s^3 + 28s^2 + 320s + 1536$

Roots of the compensated characteristic equation:

-12.0000

-8.0000 + 8.0000i

-8.0000 - 8.0000i

Peak time = 0.437 hspace1.5cm Percent overshoot = 2.014

Rise time = 0.237

Settling time = 0.494

## 6.11 GUI Program for Root-locus Design

A graphical user interface program (GUI) has been developed for the design of a first-order controller in the forward path of a closed-loop control system. This GUI

program designs the following controllers: Proportional, phase-lag, phase-lead, PD, PI, and PID controllers. To run the GUI controller design program at the MATLAB prompt type **rldesigngui**.

### Example 6.17

Consider the system of Example 6.10 whose open-loop transfer function is

$$G_p(s) = \frac{1}{s(s+1)(s+4)}$$

Use the **rldesigngui** program to design a phase lead controller to meet the following specifications:

- Time constant  $\tau = 1/\zeta\omega_n = 0.6667$  sec.
- Damping ratio  $\zeta = 0.6$ .
- Steady-state error due to a unit ramp input  $e_{ss} = 0.5$ .

From the first two specifications  $\zeta\omega_n = 1/\tau = 1.5$  and  $\theta = \cos^{-1} 0.6 = 53.13^\circ$ . Thus, the required closed-loop pole is  $s_1 = -1.5 + j2$ . The third specification requires

$$e_{ss} = \frac{1}{K_v} = 0.5$$

or

$$K_v = 2$$

where

$$K_v = \lim_{s \rightarrow 0} s \frac{K_c(s+z_0)}{(s+p_0)} \frac{1}{s(s+1)(s+4)} = \frac{K_c z_0}{p_0} \frac{1}{4}$$

Therefore, the compensator gain required is  $a_0 = (k_c z_0)/p_0 = 4 \times 2 = 8$ .  
At the MATLAB prompt type

```
>> rldesigngui
```

The following graphic window is displayed.

**Root-Locus Design**

Enter the num and den coefficients in descending powers of s

**Plant TF,  $G_p(s)$ :**

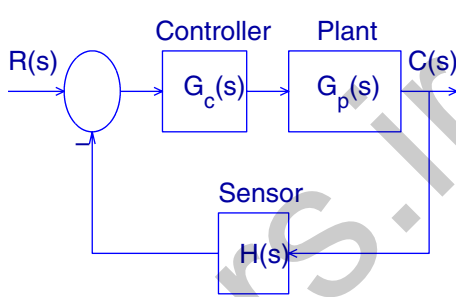
num:

den:

**Feedback path TF,  $H(s)$ :**

num:

den:



P Controller	Find K to satisfy the desired damping ratio
Phase Lag Controller	Design a phase lag controller for the specified damping ratio of dominant closed-loop poles and to satisfy the steady-state error
Phase Lead Controller	Design a phase lead controller for the desired dominant closed-loop poles
PD Controller	Design a PD controller for the desired dominant closed-loop poles
PI Controller	Design a PI controller for the desired dominant closed-loop poles
PID Controller	Design a PID controller for the desired dominant closed-loop poles

MS  
OE

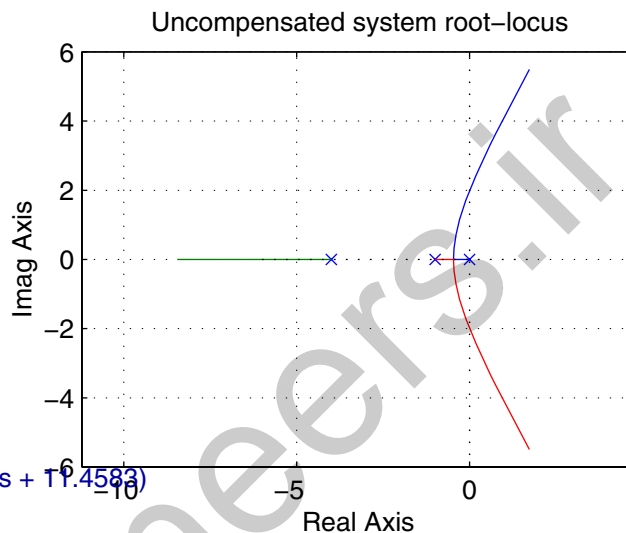
(c) Hadi Saadat

Exit

The plant transfer function is specified and clicking on the phase lead controller button, the following graphic window is displayed as shown in the next page. The desired dominant closed-loop poles are entered. Pressing the Find  $G_c(s)$  button, the controller transfer function, and the compensated open loop and closed loop transfer functions are obtained as shown in this figure.

### Phase Lead Controller

Enter the desired dominant closed loop pole	Enter the compensator DC gain, $G_c(0)$
$s_1 = -1.5+2i$	$G_c(0) = 8$
Find $G_c(s)$	



Controller:  $G_c(s) = 82.2813(s + 1.1141)/(s + 19.4583)$

Compensated Open-loop TF

$$KG_c G_p H(s) = \frac{82.2813s + 91.6667}{1s^4 + 16.4583s^3 + 61.2917s^2 + 45.8333s + 0}$$

Compensated Closed-loop TF

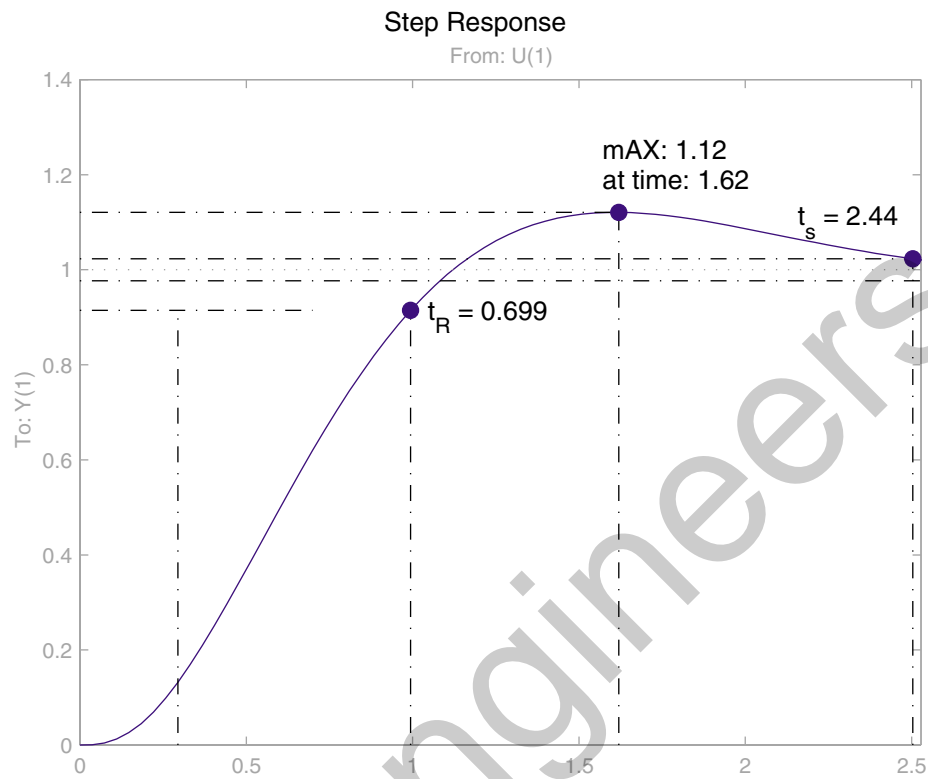
$$\frac{C(s)}{R(s)} = \frac{82.2813s + 91.6667}{1s^4 + 16.4583s^3 + 61.2917s^2 + 128.1146s + 91.6667}$$

Roots of the Characteristic Equation:

$-12.2623+0i$        $-1.5+2i$        $-1.5-2i$        $-1.19608+0i$

MS OE	(c) Hadi Saadat	Compensated System Responses	Close
----------	-----------------	------------------------------	-------

Pressing the Compensated System Responses Button, various responses can be obtained. The step response is shown in Figure 6.18. Right-clicking on the response opens a dialog box for the response characteristics. In this case the time-domain performance characteristics such as the response rise time, peak value, peak time and settling time are obtained as marked on the plot.



---

# CHAPTER 7

---

## FREQUENCY RESPONSE ANALYSIS AND DESIGN

The frequency response of a system is the steady-state response of the system to a sinusoidal input signal. The frequency response method and the root-locus method are simply two different ways of applying the same basic principles of analysis. These methods supplement each other, and in many practical design problems, both techniques are employed. One advantage of the frequency response method is that the transfer function of a system can be determined experimentally by frequency response tests. Furthermore, the design of a system in the frequency domain provides the designer with control over the system bandwidth and over the effect of noise and disturbance on the system response.

In this chapter *polar plot* and *Bode plot* of the open-loop transfer function, *gain margin*, and *phase margin* specifications are obtained using *MATLAB Control System Toolbox* functions. The relative stability of the closed-loop system based on the *Nyquist criterion* are examined. Closed-loop frequency response, the *peak amplitude*, and *bandwidth* are also obtained. In addition, several functions developed for the control system design in frequency domain are demonstrated.

### 7.1 Frequency Response

The response of a linear time-invariant system to sinusoidal input  $r(t) = A \sin(\omega t)$  is given by

$$c(t) = A |G(j\omega)| \sin[\omega t + \theta(\omega)] \quad (7.1)$$

where the transfer function  $G(j\omega)$  is obtained by substituting  $j\omega$  for  $s$  in the expression for  $G(s)$ . The resulting transfer function may be written in *polar form* as

$$G(j\omega) = |G(j\omega)| \angle \theta(\omega) \quad (7.2)$$

Alternatively, the transfer function can be represented in rectangular complex form as

$$G(j\omega) = \text{Re}G(j\omega) + j\text{Im}G(j\omega) = R(j\omega) + jX(j\omega) \quad (7.3)$$

The most common graphical representation of a frequency response function is the *Bode plot*. Other representations of sinusoidal transfer functions are *polar plot* and *log-magnitude versus phase plot*.

### 7.1.1 Bode Plot

The *Bode plot* consists of two graphs plotted on semi-log paper with linear vertical scales and logarithmic horizontal scales. The first graph is a plot of the magnitude of a frequency response function  $G(j\omega)$  in decibels versus the logarithm of  $\omega$ , the frequency. The second graph of a Bode plot shows the phase function  $\theta(\omega)$  versus the logarithm of  $\omega$ . The logarithmic representation is useful in that it shows both the low- and high-frequency characteristics of the transfer function in one diagram. Furthermore, the frequency response of a system may be approximated by a series of straight line segments.

Given a transfer function of a system, the *Control System Toolbox* function **bode(num, den)** produces the frequency response plot with the frequency vector automatically determined. If the system is defined in state space, we use **bode(A, B, C, D)**. **bode(num, den,  $\omega$ )** or **bode(A, B, C, D,  $\omega$ )** uses the user-supplied frequency vector  $\omega$ . The scalar **iu** specifies which input is to be used for the frequency response. If the above commands are invoked with the left-hand arguments [**mag, phase,  $\omega$** ], the frequency response of the system in the matrices **mag**, **phase**, and  $\omega$  are returned, and we need to use **plot** or **semilogx** functions to obtain the plot.

#### Example 7.1

Obtain the Bode plot for the unity feedback control system with the open-loop transfer function

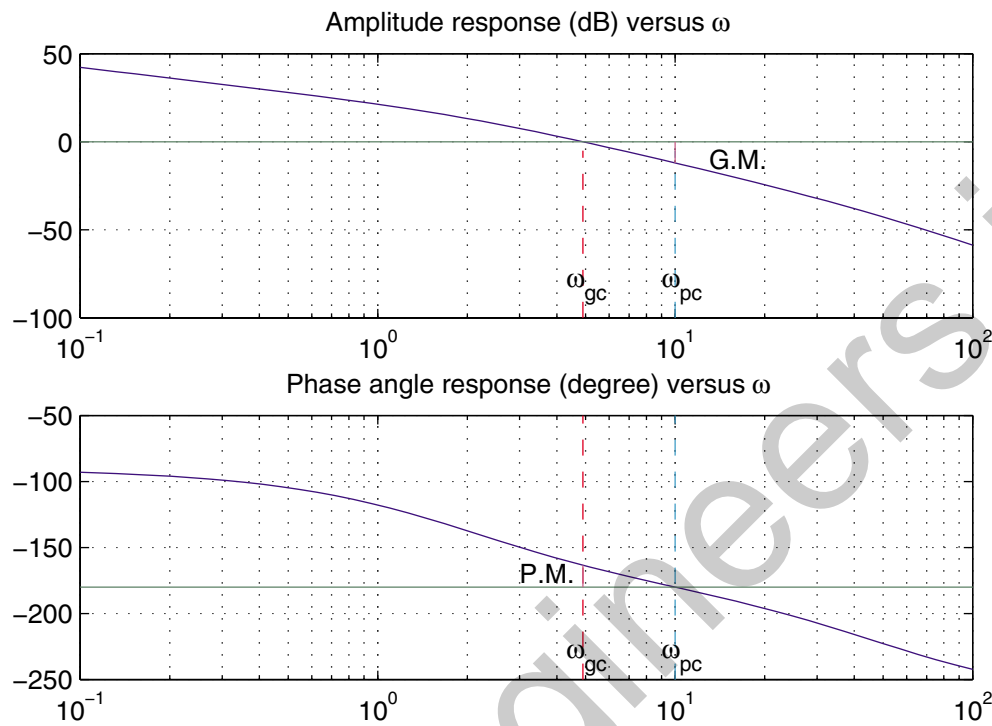
$$GH(s) = \frac{K}{s(s+2)(s+50)} = \frac{K}{s^3 + 52s^2 + 100s}$$

for  $K = 1300$ . We use the following commands

```

num=1300;
den=[1 52 100 0];
bode(num, den), grid           % Obtains the frequency response plots
                                % Magnitude in dB and phase angle in degrees
    
```

The resulting plot is shown in Figure 7.1. The commands used for displaying text ( $\omega_{gc}$ ,  $\omega_{pc}$ ,  $G.M.$ ,  $P.M.$ ) on the graphs are not included in the above statements.



**FIGURE 7.1**  
Bode plot of Example 7.1.

### 7.1.2 Polar Plot

A *polar plot*, also called the *Nyquist plot*, is a graph of  $\text{Im}G(j\omega)$  versus  $\text{Re}G(j\omega)$  with  $\omega$  varying from  $-\infty$  to  $+\infty$ . The polar plot may be directly graphed from sinusoidal steady-state measurements on the components of the open-loop transfer function.

The *MATLAB Control System Toolbox* function `[Re, Im]=nyquist(num, den,  $\omega$ )` returns the real and imaginary parts of a transfer function for the specified range of frequencies.

#### Example 7.2

Obtain the polar plot for the system of Example 7.1 with the gain  $K = 1300$  and  $K = 5200$ .

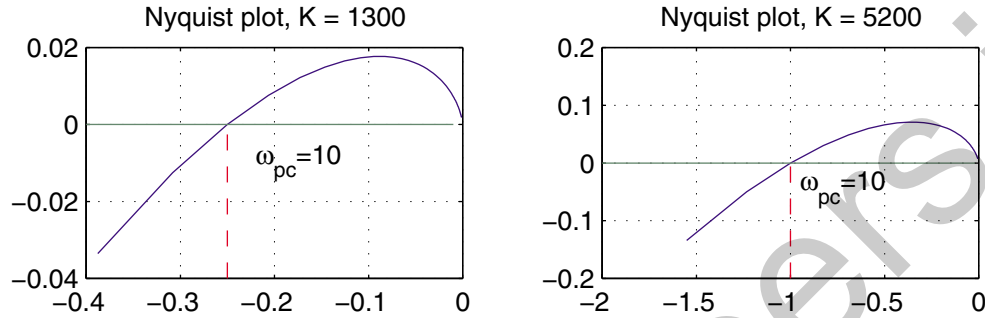
The following commands

```
k1=1300; k2=5200; w=8:1:80;
num1=[k1]; num2=[k2];
den=[1 52 100 0];
[Re1,Im1]=nyquist(num1,den,w);
[Re2,Im2]=nyquist(num2,den,w);
subplot(2,2,1),plot(Re1,Im1), title('Nyquist plot, K = 1300')
grid
```



```
subplot(2,2,2),plot(Re2,Im2), title('Nyquist plot, K = 5200')
grid, subplot(111)
```

produce the graph shown in Figure 7.2.



**FIGURE 7.2**  
Polar plot of Example 7.1.

### 7.1.3 Log-magnitude versus Phase Plot

The *log-magnitude versus phase plot* shows the logarithmic magnitude in decibels versus phase angle for a frequency range of interest, usually plotted on a *Nichols chart*. The Nichols chart contains lines of constant closed-loop magnitude and phase, showing the relationship between the open-loop and closed-loop frequency response. In Example 7.1, the addition of statement **plot(dB, phase)** will produce the log-magnitude versus phase plot.

## 7.2 Relative Stability

The closed-loop transfer function of a control system is given by

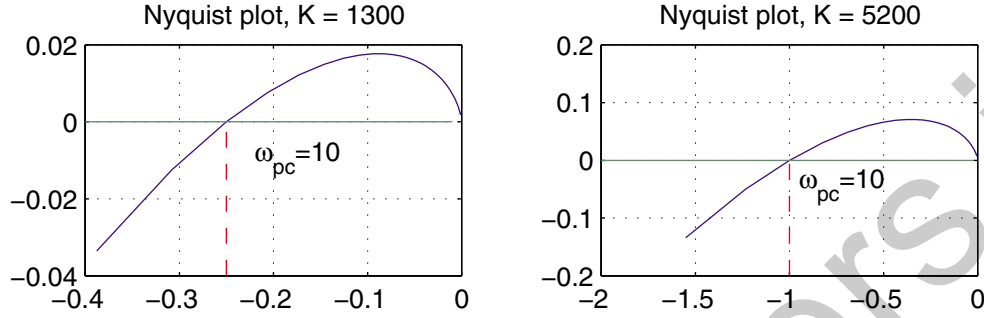
$$T(s) = \frac{C(s)}{R(s)} = \frac{KG(s)}{1 + KGH(s)} \quad (7.4)$$

For *BIBO* stability, poles of  $T(s)$  must lie in the left-half  $s$ -plane. Since zeros of  $1 + KGH(s)$  are poles of  $T(s)$ , the system is *BIBO* stable when the roots of the characteristic equation  $1 + KGH(s)$  lie in the left-half  $s$ -plane. The *root-locus*, which is the locus of the roots of the characteristic equation as  $K$  varies from zero to infinity, and is covered in Chapter 6. All points on the root-locus satisfy the following conditions:

$$|KGH(s)| = 1 \quad \text{and} \quad \angle GH(s) = -180^\circ \quad (7.5)$$

The root-locus for Example 7.1 is shown in Figure 7.3. As  $K$  is increased, the system becomes marginally stable when the real parts of the dominant complex roots are zero. This corresponds to the intersection of the root-loci with  $j\omega$ -axis; that is,

when  $s = j\omega$ . For this system, the critical gain for marginal stability is  $K_c = 5200$ . The polar plot of the above system for three values of  $K$  is also shown in Figure 7.3.



**FIGURE 7.3**

Root-locus and polar plot for three values of gain.

The intersection of the polar plot with the negative real axis has a phase angle of  $-180^\circ$ . The frequency  $\omega_{pc}$  corresponding to this point is known as the *phase crossover frequency*. In addition, as the loop gain is increased, the polar plot crossing  $(-1, 0)$  point has the property described by

$$|K_c GH(j\omega_{pc})| = 1 \quad \text{and} \quad \angle GH(j\omega_{pc}) = -180^\circ \quad (7.6)$$

The closed-loop response becomes marginally stable when the frequency response magnitude is unity and its phase angle is  $-180^\circ$ . The frequency at which the polar plot intersect  $(-1, 0)$  point is the same frequency that the root-locus crosses the  $j\omega$ -axis. For a still larger value of  $K$ , the polar plot will enclose the  $(-1, 0)$  point, and the system is unstable.

Thus, the system is stable if

$$|K GH(j\omega)| < 1 \quad \text{at} \quad \angle GH(j\omega_{pc}) = -180^\circ \quad (7.7)$$

The proximity of the  $K GH(j\omega)$  plot in the polar coordinates to the  $(-1, 0)$  point gives an indication of the stability of the closed-loop system.

### 7.2.1 Gain and Phase Margins

*Gain margin* and *phase margin* are two common design criteria related to the open-loop frequency response. The *gain margin* is the factor by which the gain of a stable system must be increased for the polar plot to pass through the  $(-1, 0)$  point. The gain margin is defined as

$$G.M. = \frac{K_c}{K} \quad (7.8)$$

where  $K_c$  is the critical loop gain for marginal stability and  $K$  is the actual loop gain. The above ratio can be written as

$$G.M. = \frac{K_c |GH(j\omega_{pc})|}{K |GH(j\omega_{pc})|} = \frac{1}{K |GH(j\omega_{pc})|} = \frac{1}{a} \quad (7.9)$$

In terms of decibels the gain margin is

$$G.M._{dB} = 20 \log_{10}(G.M.) = -20 \log_{10} |KGH(j\omega_{pc})| = -20 \log_{10} a \quad (7.10)$$

The gain margin is simply the factor by which  $K$  must be changed in order to render the system unstable. In this example, with  $K = 1300$ , the  $G.M. = 1/a = 1/0.25 = 4$ , or  $G.M._{dB} = 20 \log_{10} 4 = 12.04$  dB. Thus, the critical loop gain is  $K_c = (G.M.)K = (4)1300 = 5200$ . For  $K = 10400$ ,  $G.M. = 1/2 = 0.5 = -6.02$  dB and the system is unstable.

The gain margin alone is inadequate to indicate relative stability when system parameters affecting the phase of  $GH(j\omega)$  are subject to variation. Another measure called *phase margin* is required to indicate the degree of stability. Let  $\omega_{gc}$ , known as the *gain crossover frequency*, be the frequency at which the open-loop frequency response magnitude is unity. The phase margin is the angle in degrees through which the polar plot must be rotated about the origin in order to intersect the  $(-1, 0)$  point. The phase margin is given by

$$P.M. = \angle GH(j\omega_{gc}) - (-180^\circ) \quad (7.11)$$

In this example for  $K = 1300$ ,  $\omega_{gc} = 4.89$  and  $P.M. = -163.36 - (-180) = 16.64^\circ$ .

The gain and phase margins along with  $\omega_{pc}$  and  $\omega_{gc}$  are obtained more easily from the Bode plot. The phase margin may be read directly off the Bode plot at the frequency which the amplitude curve crosses the 0 dB line [ $\omega_{gc} = 4.89$ ], and the gain margin may be read (in decibels) at the frequency at which the phase angle curve crosses the  $-180^\circ$  line [ $\omega_{pc} = 10$ ]. From Figure 7.1, these are  $G.M. = 12.04$  dB and  $P.M. = 16.64^\circ$ .

For satisfactory performance, the phase margin should be between  $30^\circ$  and  $60^\circ$ , and the gain margin should be greater than 6 dB.

The **MATLAB Control System Toolbox** function **[Gm, Pm,  $\omega_{pc}$ ,  $\omega_{gc}$ ] = margin(mag, phase, w)** can be used with **bode** function for evaluation of gain and phase margins,  $\omega_{pc}$  and  $\omega_{gc}$ .

### Example 7.3

In Example 7.1 for  $K = 1300$ , evaluate the gain margin,  $\omega_{pc}$ , phase margin, and  $\omega_{gc}$ .

The following commands

```
k = 1300; num = [k]; den = [1 52 100 0]; w = .1:.1:20;
[mag, phase] = bode(num, den, w);
[Gm, Pm, wpc, wgc] = margin(mag, phase, w);
fprintf('Gain Margin = %7.3g', Gm), fprintf('..
Gain crossover w = %7.3g', wgc)
fprintf('Phase Margin = %7.3g', Pm), fprintf('..
Phase crossover w = %7.3g', wpc)
```

result in

```
Gain Margin =          4      Gain crossover w =      4.89
Phase Margin =      16.6    Phase crossover w =      10
```

### 7.2.2 Nyquist Stability Criterion

The *Nyquist stability criterion* provides a convenient method for finding the number of zeros of  $1 + GH(s)$  in the right-half  $s$ -plane directly from the Nyquist plot of  $GH(s)$ . The Nyquist stability criterion is defined in terms of the  $(-1, 0)$  point on the Nyquist plot or the *zero-dB*,  $180^\circ$  point on the Bode plot. The Nyquist criterion is based upon a theorem of complex variable mathematics due to Cauchy. The *Nyquist diagram* is obtained by mapping the *Nyquist path* into the complex plane via the mapping function  $GH(s)$ . The Nyquist path is chosen so that it encircles the entire right-half  $s$ -plane. When the  $s$ -plane locus is the Nyquist path, the Nyquist stability criterion is given by

$$Z = N + P \quad (7.12)$$

where

$P$  = number of poles of  $GH(s)$  in the right-half  $s$ -plane,

$N$  = number of clockwise encirclements of  $(-1, 0)$  point by the Nyquist diagram,

$Z$  = number of zeros of  $1 + GH(s)$  in the right-half  $s$ -plane.

For the closed-loop system to be stable,  $Z$  must be zero, that is

$$N = -P \quad (7.13)$$

### 7.2.3 Simplified Nyquist Criterion

If the open-loop transfer function  $GH(s)$  does not have poles in the right-half  $s$ -plane ( $P = 0$ ), it is not necessary to plot the complete Nyquist diagram; the polar plot for  $\omega$  increasing from  $0^+$  to  $\infty$  is sufficient. Such an open-loop transfer function is called *minimum-phase* transfer function. For minimum-phase open-loop transfer functions the closed-loop system is stable if and only if the polar plot lies to the right of  $(-1, 0)$  point. For a minimum-phase open-loop transfer function the criterion is defined in terms of the polar plot crossing with respect to  $(-1, 0)$  point as follows:

Right of $(-1, 0)$	stable	$\omega_{pc} > \omega_{gc}$	$ GH(j\omega)  < 1$ , $G.M._{dB} > 0$ , $P.M. > 0^\circ$
On $(-1, 0)$	marg. stable	$\omega_{pc} = \omega_{gc}$	$ GH(j\omega)  = 1$ , $G.M._{dB} = 0$ , $P.M. = 0^\circ$
Left of $(-1, 0)$	not stable	$\omega_{pc} < \omega_{gc}$	$ GH(j\omega)  > 1$ , $G.M._{dB} < 0$ , $P.M. < 0^\circ$

If  $P$  is not zero, the closed-loop system is stable if and only if the number of counter-clockwise encirclements of the Nyquist diagram about  $(-1, 0)$  point is equal to  $P$ .

The *MATLAB Control System Toolbox* function **[re, im] = nyquist(num, den,  $\omega$ )** can obtain the Nyquist diagram by mapping the Nyquist path. However, the argument  $\omega$  is specified as a real number. In order to map a complex number  $s = a + jb$ , we must specify  $\omega = -js$ , since the above function automatically multiplies  $\omega$  by the operator  $j$ . To avoid this, the developed function **[re, im] = cnyquist(num, den,  $s$ )** can be used, where the argument  $s$  must be specified as a complex number. In

defining the Nyquist path care must be taken for the path not to pass through any poles or zeros of  $GH(s)$ . The use of this function is demonstrated in the following example.

#### Example 7.4

The open-loop transfer function of a system is given by

$$GH(s) = \frac{K(s+1)}{(s-2)(s-4)} = \frac{K(s+1)}{s^2 - 6s + 8}$$

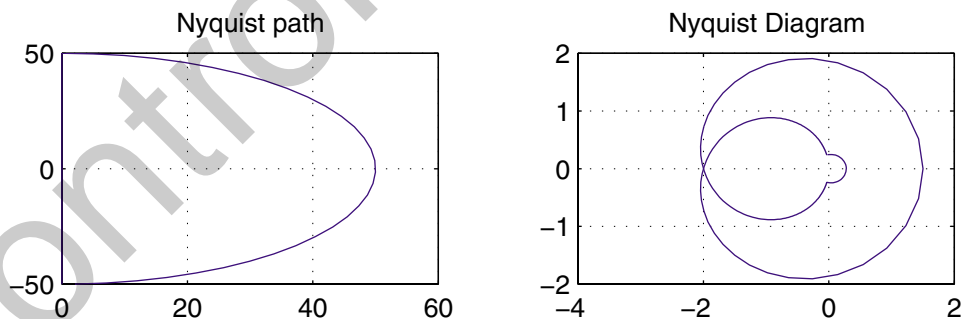
Obtain the Nyquist diagram for  $K = 12$  and determine if the system is stable.

```

th = pi/2:-.1:-pi/2;
                                % Select a path with a large radius, say
                                % 50 to ensure open-loop poles are
                                % enclosed.
s1 = j*(-50:.2:50); % Nyquist path on the jw-axis from -j50
                                % to j50
s2 = 50*exp(j*th); % Nyquist path from pi/2 to -pi/2 with
                                % radius 50
s = [s1 s2]; % Row vector containing the Nyquist path

num = [12 12];
den = [1 -6 8];
[Re, Im] = cnyquist(num,den, s);
subplot(221),plot(s),title('Nyquist path'), grid
subplot(222),plot(Re,Im),title('Nyquist Diagram'),grid
    
```

The results are shown in Figure 7.4.



**FIGURE 7.4**

Nyquist path and Nyquist diagram of the system in Example 7.4.

The open-loop transfer function has two poles in the right-half  $s$ -plane, that is  $P = 2$ . The Nyquist path encircles  $(-1, 0)$  point twice in the counterclockwise direction, that is  $N = -2$  and  $Z = P + N = 2 - 2 = 0$ . Therefore, for  $K = 12$ , the closed-loop system is stable. The  $G.M.$  is  $1/2$ , and gain for marginal stability is  $K_c = 1/2(12) = 6$ . The system is stable for all  $K > 6$ .

### 7.3 Closed-Loop Frequency Response

The *closed-loop frequency response* is the frequency response of the closed-loop transfer function  $T(j\omega)$ . The performance specifications in terms of closed-loop frequency response are the closed-loop system *bandwidth*,  $\omega_B$ , and the closed-loop system *resonant peak magnitude*,  $M_p$ .

The bandwidth,  $\omega_B$ , is defined as the frequency at which the  $|T(j\omega)|$  drops to 70.7 percent of its zero frequency value, or 3 dB down from the zero frequency value. The bandwidth indicates how well the system tracks an input sinusoid and is a measure of the speed of response. If the bandwidth is small, only signals of relatively low frequency are passed, and the response is slow; whereas a large bandwidth corresponds to a faster rise time. Therefore, the rise time and the bandwidth are inversely proportional to each other.

The frequency at which the peak occurs, the *resonant frequency*, is denoted by  $\omega_r$ , and the maximum amplitude,  $M_p$ , is called the resonant peak magnitude.  $M_p$  is a measure of the relative stability of the system. A large  $M_p$  corresponds to the presence of a pair of dominant closed-loop poles with small damping ratio, which results in a large maximum overshoot of the step response in the time domain. If the gain  $K$  is set so that the open-loop frequency response  $GH(j\omega)$  passes through the  $(-1, 0)$  point,  $M_p$  will be infinity. In general, if  $M_p$  is kept between 1.0 and 1.7, the transient response will be acceptable. The developed function **frqspec(w, mag)** calculates  $M_p$ ,  $\omega_r$ , and the bandwidth  $\omega_B$  from the frequency response data.

#### Example 7.5

The closed-loop transfer function of Example 7.1 for  $K = 438$  is

$$T(s) = \frac{438}{s^3 + 52s + 100s + 438}$$

Obtain the amplitude response and determine the system bandwidth and the resonant peak magnitude. The following commands

```

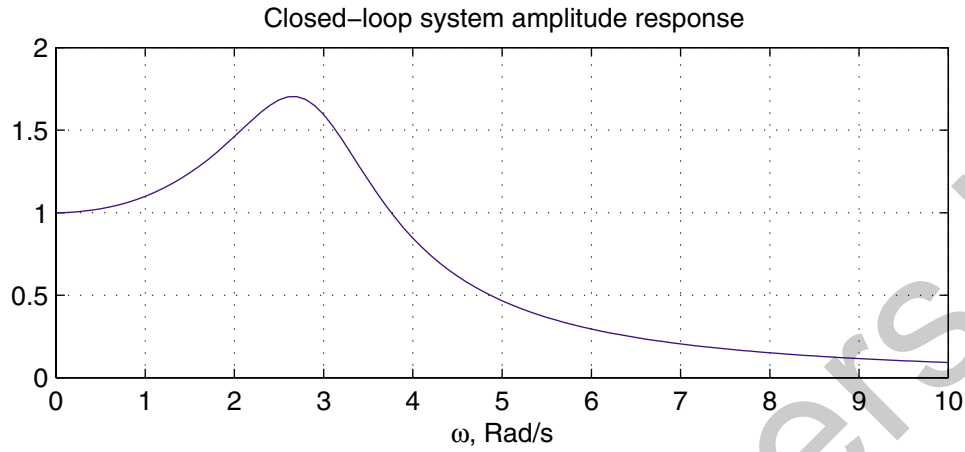
k = 438; num = k;    den = [1 52 100 k];
w = 0:.1:10;
[mag, phase] = bode(num, den, w);
subplot(211), plot(w, mag),
title('Closed-loop system amplitude response'),
grid, pause, frqspec(w, mag)
    
```

give the result shown in Figure 7.5.

Peak Mag. = 1.7       $\omega_r$  = 2.7      Bandwidth = 4.25

#### 7.3.1 Nichols Chart

The closed-loop frequency response obtained above is fast, accurate, and not limited to a unity feedback system. However, it is possible to use the *Nichols chart*, which



**FIGURE 7.5**  
Closed-loop frequency response of the system in Example 7.5.

provides a graphical technique for obtaining closed-loop response from the open-loop data. The chart is usually constructed for a unity feedback system.

The closed-loop frequency response of a unity feedback system is given by

$$M(j\omega) = \frac{G(j\omega)}{1 + G(j\omega)} = \frac{x + jy}{1 + x + jy} \quad (7.14)$$

Let  $M = |M(j\omega)|$  and  $N = \tan \phi_m(j\omega)$ , where  $\phi_m$  is the closed-loop transfer function phase angle. Upon substitution, the following two equations are obtained in terms of  $M$  and  $N$  [10].

$$\left(x - \frac{M^2}{1 - M^2}\right)^2 + y^2 = \left(\frac{M}{1 - M^2}\right)^2 \quad (7.15)$$

$$\left(x + \frac{1}{2}\right)^2 + \left(y - \frac{1}{2N}\right)^2 = \frac{1}{4} + \frac{1}{4N^2}. \quad (7.16)$$

For a given  $M$ , (7.15) represents a circle with radius  $r = M/(1 - M^2)$  and center at  $(M^2/(1 - M^2), 0)$  known as the  $M$  circle.

For a given  $N$ , (7.16) represents a circle with radius  $r = \sqrt{(N^2 + 1)/(4N^2)}$  and center at  $-1/2, 1/(2N)$ , known as the  $N$  circle.

The constant  $M$  and  $N$  circles can be used for analysis and design in the polar plane. However, it is common to transform the  $M$  and  $N$  circles on the polar plot into noncircular  $M$  and  $N$  contours on a log-magnitude phase diagram. The resulting chart is called the *Nichols chart*. The procedure for obtaining the closed-loop response from the Nichols chart is as follows:

1. The open-loop frequency response is superimposed on the Nichols chart.



2. From the intersection of this curve with the contours at various frequency points, values of  $M$  and  $\phi_m$  are read from the plot.
3. The closed-loop frequency response curves are obtained from the above values.
4. The resonant peak value of  $M_r$  and the corresponding  $\omega_r$  is given at a point where  $G(j\omega)$  is tangent to an  $M$  circle.
5. The bandwidth  $\omega_B$  is obtained by noting the frequency at which the  $G(j\omega)$  curve intersects the  $M = 0.707$  locus.

In order to use the Nichols chart for nonunity feedback systems, the block diagram must be rearranged to obtain an equivalent unity feedback system.

The *MATLAB Control System Toolbox* function **ngrid** is used to obtain the Nichols chart. **ngrid('off')** resumes normal auto-scaling.

#### Example 7.6

On the Nichols chart, draw the log-magnitude phase curve of the open-loop transfer function of Example 7.1 for  $k = 438$ .

```

k = 438;
num = k; den = [1 52 100 0]; % open-loop transfer function
w = .1:1:10;
[mag,phase] = bode(num,den,w);
ngrid, % generates Nichols chart
semilogy(phase, mag) % log-magnitude-phase plot
ngrid('off') % resumes normal auto-scaling
    
```

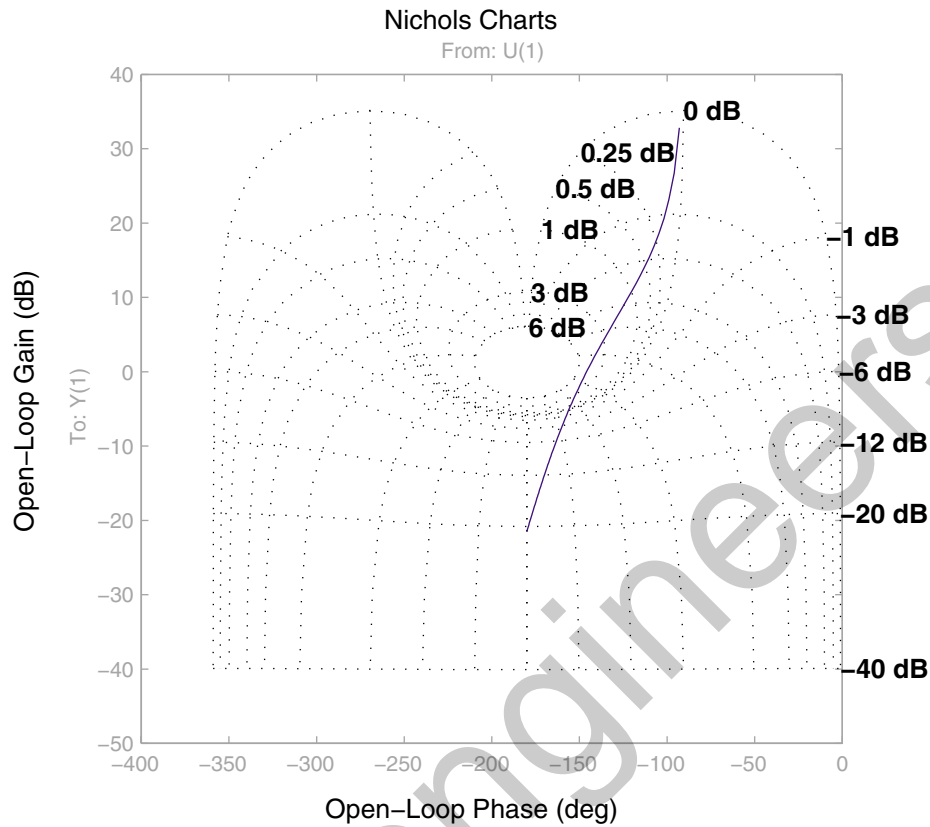
The results are shown in Figure 7.6.

From the Nichols chart we can see that the open-loop transfer function is tangent to the  $M = 1.7$  circle. This is the resonant peak magnitude  $M_r$  which agrees with the value found in Example 7.5.

## 7.4 Frequency-Response Design

In Chapter 6, root-locus analysis and design were presented. In the remainder of this chapter, the design of linear control systems is carried out in the frequency domain. The frequency response design provides information on the steady-state response, stability margin, and system bandwidth. The transient response performance can be estimated indirectly in terms of the phase margin, gain margin, and resonant peak magnitude. Percent overshoot is reduced with an increase in the phase margin, and the speed of response is increased with an increase in the bandwidth. Thus, the gain crossover frequency, resonant frequency, and bandwidth give a rough estimate of the speed of transient response.





**FIGURE 7.6**  
Plot of  $G(j\omega)$  superimposed on Nichols chart.

A common approach to the frequency response design is to adjust the open-loop gain so that the requirement on the steady-state accuracy is achieved. This is called the *proportional controller*. If the specifications on the phase margin and gain margin are not satisfied, then it is necessary to reshape the open-loop transfer function by adding the additional controller  $G_c(s)$  to the open-loop transfer function.  $G_c(s)$  must be chosen so that the system has certain specified characteristics. This can be accomplished by combining proportional with integral action (*PI*) or proportional with derivative action (*PD*). There are also proportional-plus-integral-plus-derivative (*PID*) controllers with the following transfer function

$$G_c(s) = K_P + \frac{K_I}{s} + K_D s \quad (7.17)$$

The ideal integral and differential compensators require the use of active amplifiers.

Other compensators which can be realized with only passive network elements are lead, lag and lead-lag compensators. A first-order compensator having a single

zero and pole in its transfer function is

$$G_c(s) = \frac{K_c(s + z_0)}{s + p_0} \quad (7.18)$$

Several functions have been developed for the selection of suitable controller parameters based on the satisfaction of frequency response criteria such as phase margin. These are

Function	Controller
<code>[numopen, denopen, denclsd] = frqp(num, den)</code>	Proportional
<code>[numopen, denopen, denclsd] = frqlead(num, den)</code>	Phase-lead
<code>[numopen, denopen, denclsd] = frqlag(num, den)</code>	Phase-lag
<code>[numopen, denopen, denclsd] = frqpd(num, den)</code>	PD
<code>[numopen, denopen, denclsd] = frqpi(num, den)</code>	PI
<code>[numopen, denopen, denclsd] = frqpid(num, den)</code>	PID

Alternatively, the function `[numopen, denopen, denclsd] = frdesign(num, den)` allows the user to select any of the above controller designs where **num** and **den** are row vectors of polynomial coefficients of the uncompensated open-loop plant transfer function. The function returns the open-loop and closed-loop numerators and denominators of the compensated system transfer function.

## 7.5 Gain Factor Compensation or $P$ Controller

The  $P$  controller is a pure gain controller. The design is accomplished by choosing the gain  $K_P$  for the uncompensated system to give the desired steady-state error. When the gain  $K_P$  is varied, the phase angle plot will not be affected. The Bode magnitude curve is shifted up or down to correspond to the increase or decrease in  $K_P$ . Similarly, the effect of changing  $K_P$  on the Nyquist diagram is to enlarge or reduce it; the shape of the Nyquist diagram cannot be changed.

The function `[numopen, denopen, denclsd] = frdesign(num, den)` displays six options for frequency response design. For proportional controller, option 1 must be selected, which calls upon the function `[numopen, denopen, denclsd] = frqp(num, den)`. The user enters the desired gain  $K_p$ . The open-loop and closed-loop frequency-domain specifications before and after compensation are found. Roots of the compensated characteristic equation are also computed. The function returns the open-loop and closed-loop numerators and denominators of the compensated system transfer function.

### Example 7.7

Obtain the Bode plot, gain and phase margins for the feedback control system with the open-loop transfer function

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

Determine the gain factor  $K_P$  of a proportional controller such that the steady-state error due to a ramp input will equal 0.25. Obtain the frequency response of the compensated system and the new gain and phase margins.

The steady-state error specification requires

$$e_{ss} = \frac{1}{K_v} = 0.25$$

or

$$K_v = 4$$

where the velocity error constant  $K_v$  is given by

$$K_v = \lim_{s \rightarrow 0} s \frac{8K_p}{s(s+1)(s+4)} = 2K_p$$

Therefore, the compensator gain required is  $K_p = 2$ .

The following commands

```
num = 8; den = [1 5 4 0]; w = 0.1:0.1:10;
[numopen,denopen,denclsd]=frdesign(num,den);%Design function
[mag, phase] = bode(num, den, w); dB = 20*log10(mag);
[magp, phase]= bode(numopen,denopen,w);dBp = 20*log10(magp);
subplot(211), semilogx(w, dB, w, dBp), grid
title('Uncompensated and gain compensated
Magnitude plot (dB) ') subplot(212),semilogx(w, phase),
title('Phase angle plot (degree) ')
grid
```

result in

<u>Compensator type</u>	<u>Enter</u>
Gain compensation	1
Phase-lead	2
Phase-lag	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice → 1

## 156 7. Frequency Response Analysis and Design

Uncompensated control system  
Gain Margin = 2.5 Gain crossover  $\omega = 1.22$   
Phase Margin = 22.5 Phase crossover  $\omega = 2$

Enter the desired gain factor  $K_p \rightarrow 2$

Gain & Phase Margins with gain compensation  
Gain Margin = 1.25 Gain crossover  $\omega = 1.79$   
Phase Margin = 5.19 Phase crossover  $\omega = 2$

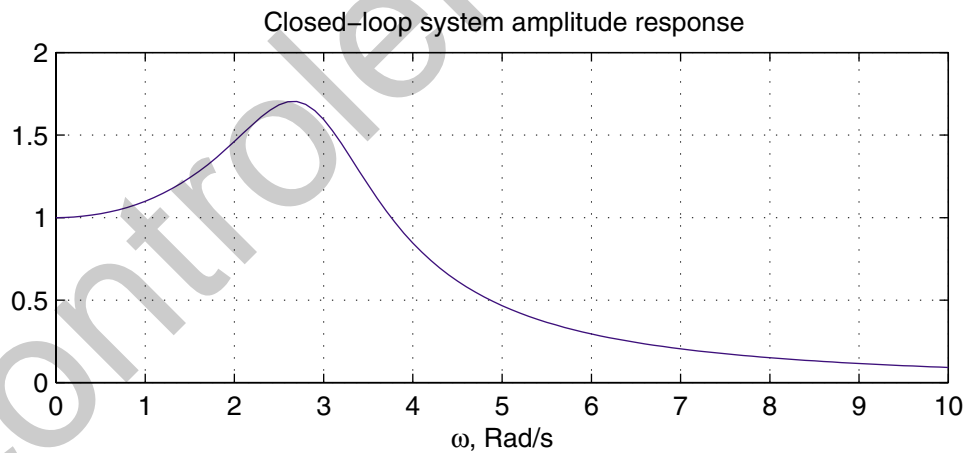
Peak Mag. = 11.6  $\omega_r = 1.8$  Bandwidth = 2.75

Row vectors of polynomial coefficients of the compensated system:

Open-loop num.	0	0	0	16
Open-loop den.	1	5	4	0
Closed-loop den.	1	5	4	16

Roots of the compensated characteristic equation:

-4.8549  
-0.0725 + 1.8139i  
-0.0725 - 1.8139i



**FIGURE 7.7**

Bode diagram of Example 7.7 with gain compensation.

The results are given in Figure 7.7. It can be seen that increasing the gain values improves the steady-state behavior but will decrease the phase and gain margins resulting in poor stability. It is then necessary to redesign the system by using a suitable controller to alter the frequency response so that the performance specifications will be met.

## 7.6 Phase-Lead Design

In (7.18) the compensator is a high-pass filter or phase-lead if  $p_0 > z_0$ . A phase-lead compensator contributes positive phase angles, and tends to increase phase margin and improve relative stability. Also, we need to increase the gain crossover frequency. This increases the bandwidth and results in a faster transient response.

If  $\omega_0$  and  $\omega_p$  are the corner frequencies of the phase-lead controller, the maximum value of the controller phase angle occurs at the frequency  $\omega_m$  given by the geometric mean of the two corner frequencies of the controller. That is

$$\omega_m = \sqrt{\omega_0 \omega_p} \quad (7.19)$$

We need to select the pole and zero of the phase-lead controller such that the maximum phase lead occurs at the new gain crossover frequency without greatly altering the magnitude curve near that frequency. This is accomplished by placing the corner frequency of the phase-lead controller such that  $\phi_m$  is located at the new gain crossover frequency,  $\omega_{gc}$ .

The design specifications simply include the phase margin specification and the steady-state error requirement.

The DC gain of the compensator is

$$a_0 = G_c(0) = \frac{K_c z_0}{p_0} \quad (7.20)$$

The controller transfer function may be written in the following form

$$G_c(s) = \frac{K_c(s + z_0)}{(s + p_0)} = \frac{a_1 s + a_0}{b_1 s + b_0} \quad (7.21)$$

where  $a_1 = K_c/p_0$ ,  $b_1 = 1/p_0$  and  $b_0 = 1$ .

The compensator DC gain  $a_0$  can be chosen to meet the steady-state error. For example, the error constant for a specified steady-state error  $e_{ss}$  due to an input ramp is given by

$$K_v = \frac{1}{e_{ss}} = \lim_{s \rightarrow 0} s G_c(s) GH(s) = a_0 \lim_{s \rightarrow 0} s GH(s) \quad (7.22)$$

Thus from the above equation, the controller DC gain  $a_0$  to realize the steady-state error specification is found.

The compensated characteristic equation of the control system is given by

$$1 + G_c(s) GH(s) = 0 \quad (7.23)$$

If  $PM$  is the desired phase margin at the new gain crossover frequency  $\omega_{gc}$ , then from (7.23) we have

$$G_c(j\omega_{gc}) GH(j\omega_{gc}) = 1 \angle (-180 + PM) \quad (7.24)$$

Substituting for  $G_c(j\omega_{gc})$  and equating real and imaginary parts of the above expression, the controller parameters are found [12] as follows

$$a_1 = \frac{1 - M \cos \theta}{\omega_{gc} M \sin \theta} \quad (7.25)$$

$$b_1 = \frac{\cos \theta - a_0 M}{\omega_{gc} \sin \theta} \quad (7.26)$$

where  $\theta$  is the angle of the controller transfer function evaluated at  $j\omega_{gc}$ ,

$$\theta = \angle G_c(j\omega_{gc}) = -180^\circ + PM - \psi \quad (7.27)$$

and  $M$  and  $\psi$  are the magnitude and phase angle of the open-loop plant transfer function evaluated at  $j\omega_{gc}$ , i.e.,

$$GH(j\omega_{gc}) = M \angle \psi \quad (7.28)$$

For a phase-lead controller, the phase-angle of the compensator must be positive; therefore, from (7.27)

$$\psi < -180^\circ + PM \quad (7.29)$$

Also, for a phase-lead compensator  $a_0 < G_c(j\omega_{gc})$ , then from (7.24)

$$a_0 M < 1 \quad (7.30)$$

For a stable controller,  $a_1$  and  $b_1$  must be greater than zero.

Based on the above equations, the function **[numopen, denopen, denclsd] = frqlead (num, den)** is used for the phase-lead controller design. **num** and **den** are the numerator and denominator of the polynomial coefficients of the open-loop plant transfer function. The user enters the desired phase margin and the controller DC gain,  $G_c(0) = K_c z_0 / p_0$ . The program finds and displays a compensated gain crossover frequency range for a stable controller. The user then specifies the crossover frequency in this range. The controller transfer function and the frequency domain specifications before and after compensation are found. Roots of the compensated characteristic equation are also computed. The function returns the open-loop and closed-loop numerators and denominators of the compensated system transfer function.

### Example 7.8

Design a phase-lead compensator for the system of Example 7.7 such that the system has a phase margin of  $45^\circ$  and a steady-state error of 0.25 due to a ramp input. Plot the frequency response before and after compensation. Find the time-domain specifications using the function **timespec(numopen, denclsd)**, and obtain a plot of the step response.

The function **[numopen, denopen, denclsd] = frdesign(num, den)** with option 2 is used for the phase-lead compensation.

The velocity error constant is  $K_v = 1/0.25 = 4$ , and from (7.22) the controller DC gain is given by  $a_0 = 4/2 = 2$ .

The following commands

```
num = 8; den = [1 5 4 0];
[numopen, denopen, denclsd] = frdesign(num, den);
                                % Design function
w = .1:.1:10;
[mag, phase] = bode(num, den, w);          dB = 20*log10(mag);
[magp, phasep] = bode(numopen, denopen, w);
```

```

dBp = 20*log10(magp);
subplot(211),semilogx(w, dB, w, dBp),
title('Uncompensated and compensated magnitude plot (dB)')
subplot(212),semilogx(w, phase, w, phasep), grid
title('Uncompensated and compensated
      phase angle plot (degree)')
t=0:.05:4; c=step(numopen, denclsd, t);
clg, subplot(211), plot(t, c), grid,
title('Step response of the compensated system')
timespec(numopen, denclsd);

```

result in

Compensator type	Enter
Gain compensation	1
Phase-lead	2
Phase-lag	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice → 2

Enter the compensator DC Gain → 2

Enter desired Phase Margin → 45

For a stable controller select a compensated gain crossover frequency wgc between 2.32 and 4.82

Suggested wgc for max. phase lead is 3.57

Enter wgc → 3.57

Uncompensated control system

Gain Margin = 2.5 Gain crossover w = 1.22

Phase Margin = 22.5 Phase crossover w = 2

Controller transfer function

$G_c(0) = 2$ ,  $G_c = 83.535(s + 0.821555)/(s + 34.3143)$

Row vectors of polynomial coefficients of the compensated system:

Open-loop num.	0	0	0	668.2798	549.0286
Open-loop den.	1.0000	39.3143	175.5714	137.2572	0
Closed-loop den	1.0000	39.3143	175.5714	805.5369	549.0286

Gain Margin = 8.27 Gain crossover w = 3.57

Phase Margin = 45 Phase crossover w = 12

Peak Mag. = 1.31 wr = 3.6 Bandwidth = 5.95

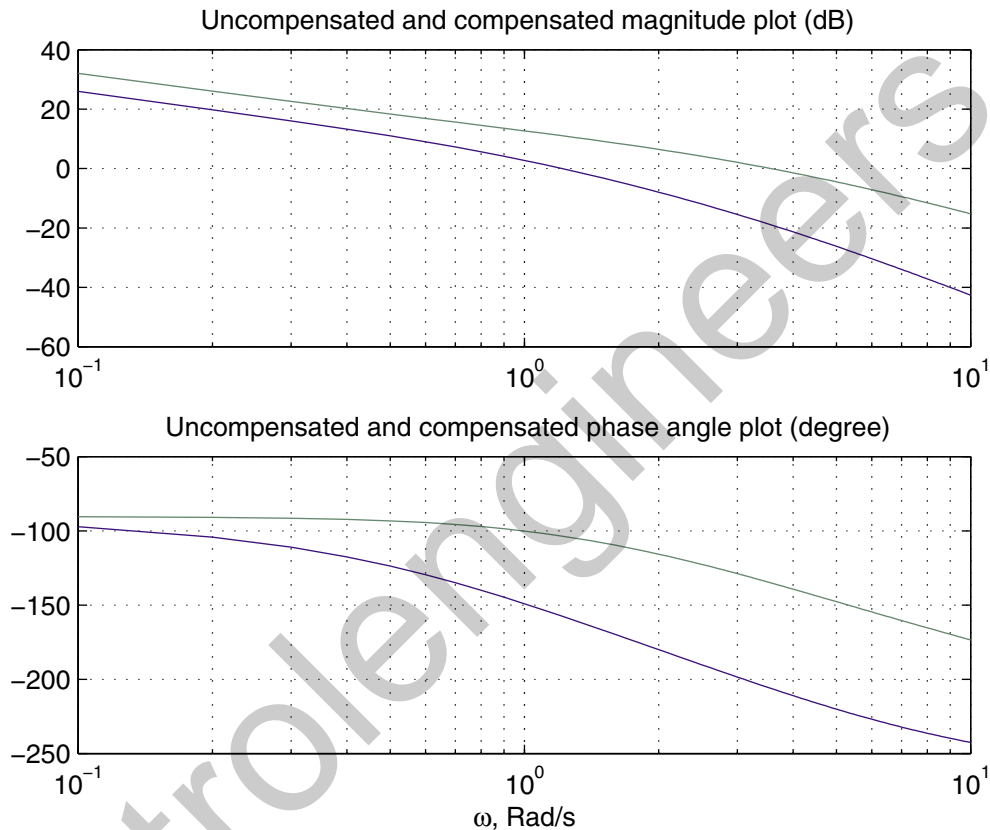
Roots of the compensated characteristic equation:

-34.9359

160 7. Frequency Response Analysis and Design

$$\begin{aligned} & -1.7915 + 4.0680i \\ & -1.7915 - 4.0680i \\ & -0.7954 \end{aligned}$$

The results are given in Figures 7.8a and 7.8b.



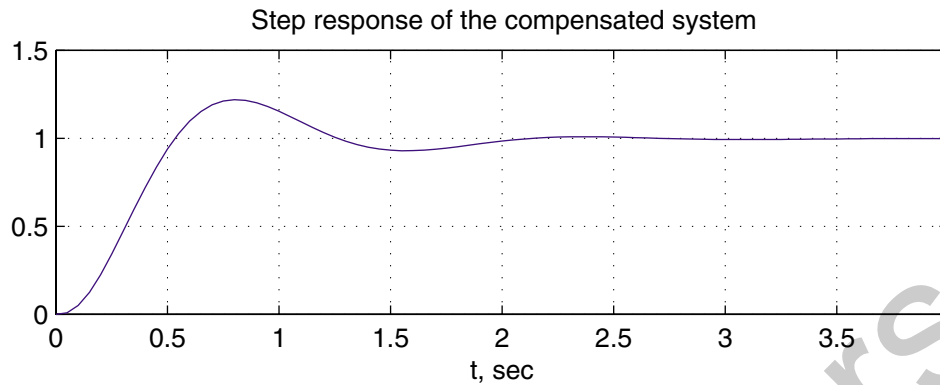
**FIGURE 7.8**  
Bode diagram of Example 7.8.

Peak time = 0.806 Percent overshoot = 22.0289  
Rise time = 0.338  
Settling time = 1.963

If the closed-loop frequency response specifications and the time-domain performance specifications are performed for the uncompensated system, we obtain the following values.

Peak Mag. = 2.63  $\omega_r$  = 1.3 Bandwidth = 1.95  
Peak time = 2.63 Percent overshoot = 22.8697





**FIGURE 7.9**  
Step response of the compensated system of Example 7.8.

Rise time = 0.97227  
Settling time = 15.198

From the above example we see that the use of a phase-lead controller to increase the phase margin will result in an increase in the crossover frequencies. In the closed-loop frequency response, the resonant peak magnitude is reduced while the bandwidth is increased. This corresponds to the reduction of the percent overshoot and the rise time in the step response.

One additional point can be made concerning the relationship between the bandwidth and the rise time of step response discussed in Chapter 4, Section 4.4. An approximate relationship was given by (4.18). That is,  $\omega_B t_r \simeq \text{constant}$ . Comparison of the uncompensated and compensated values shows that this product is approximately 2.

As a final note for the designed compensator, the ratio of the high-frequency gain ( $\omega \rightarrow \infty$ ) to the DC gain is  $83.535/2 = 41.76$ . This may produce high-frequency noise problems which may be objectionable for some systems. In this case, the ratio should be lowered (typically by a ratio of 10). To do this, redesign the system selecting a lower gain crossover frequency,  $\omega_{pc}$ . Alternatively, the phase margin may be reduced. However, this will increase the rise time and the settling time.

## 7.7 Phase-Lag Design

In (7.18), the compensator is a low-pass filter or phase-lag if  $p_0 < z_0$ . The phase-lag compensator adds a negative angle to the angle criterion and tends to destabilize the system. The design problem is to determine the compensator parameters to improve the steady-state error and to maintain a desired phase margin for a satisfactory transient response. The pole and zero of the lag compensator must be located substantially lower than the new gain crossover frequency. Therefore, we move the new gain

crossover frequency to a lower frequency while keeping the phase curve of the Bode plot relatively unchanged at the gain crossover frequency. The result is an increase in the low-frequency gain while the high-frequency range is attenuated.

Equations (7.25) and (7.26) are used for phase-lag design. However, since the compensator angle  $\theta$  must be negative, criterion (7.29) becomes

$$\psi > -180^\circ + PM \quad (7.31)$$

Also, for a phase-lag compensator, since  $a_0 > G_c(j\omega_{gc})$ , then from (7.24)

$$a_0 M > 1 \quad (7.32)$$

For a stable controller,  $a_1$  and  $b_1$  must be greater than zero.

Based on the above equations the function **[numopen, denopen, denclsd] = frqlag(num, den)** is developed for the phase-lag controller design. **num** and **den** are the numerator and denominator of the polynomial coefficients of the open-loop plant transfer function. The user enters the desired phase margin and the controller DC gain,  $G_c(0) = K_c z_0/p_0$ . The program finds and displays a compensated gain crossover frequency range for a stable controller. The user then specifies the crossover frequency in this range. The controller transfer function and the frequency domain specifications before and after compensation are found. Roots of the compensated characteristic equation are also computed. The function returns the open-loop and closed-loop numerators and denominators of the compensated system transfer function.

### Example 7.9

Design a phase-lag compensator for the system of Example 7.7 such that the system has a phase margin of  $45^\circ$  and a steady-state error of 0.05 due to a ramp input.

The function **[numopen, denopen, denclsd] = frdesign(num, den)** with option 3 is used for the phase-lag compensation.

The velocity error constant is  $K_v = 1/0.05 = 20$ , and from (7.22) the controller DC gain is given by  $a_0 = 20/2 = 10$ .

The following commands

```
num = 8; den = [1 5 4 0];
[numopen, denopen, denclsd] = frdesign(num,den);
% Design function
timespec(numopen, denclsd);
result in
```

Compensator type	Enter
Gain compensation	1
Phase-lead	2
Phase-lag	3
PD Controller	4
PI Controller	5
PID Controller	6

```

To quit                                0

Enter your choice → 3

Enter the compensator DC Gain → 10
Enter desired Phase Margin → 45
For a stable controller select a compensated gain crossover
frequency wgc between 0.070 and 0.631

Enter wgc → 0.4

Uncompensated control system
Gain Margin = 2.5 Gain crossover w = 1.22
Phase Margin = 22.5 Phase crossover w = 2

Controller transfer function
Gc(0) = 10, Gc = 0.206043(s + 0.128954)/(s + 0.00265701)

Row vectors of polynomial coefficients of the compensated
system:

Open-loop num.      0      0      0  1.6483  0.2126
Open-loop den.    1.0000  5.0027  4.0133  0.0106      0
Closed-loop den  1.0000  5.0027  4.0133  1.6590  0.2126

Gain Margin = 10.20 Gain crossover w = 0.4
Phase Margin = 45 Phase crossover w = 1.84
Peak Mag. = 1.38 wr = 0.3 Bandwidth = 0.65

Roots of the compensated characteristic equation:
-4.1240
-0.3370 + 0.3719i
-0.3370 - 0.3719i
-0.2047

Peak time = 7.105 Percent overshoot = 32.345
Rise time = 2.597
Settling time = 17.836
    
```

The advantage of the phase-lag design is that the steady-state error can be reduced to a low value; however, the disadvantage is that the bandwidth is lowered which results in a larger rise time and thus a slower response.

## 7.8 PID Design

One of the most common controllers available commercially is the PID controller. Different processes are suited to different combinations of proportional, integral, and

derivative control. The control engineer's task is to adjust the three gain factors to arrive at an acceptable degree of error reduction simultaneously with acceptable dynamic response.

For a desired phase margin  $PM$  the Nyquist plot must pass through the point  $1\angle(-180 + PM)$  at the new gain crossover frequency. The PID controller transfer function is given by (7.17). At  $j\omega_{gc}$  the controller transfer function is given by

$$G_c(j\omega_{gc}) = K_P + \frac{K_I}{j\omega_{gc}} + K_D j\omega_{gc} \quad (7.33)$$

The controller parameters are found [12] by substituting (7.33) in (7.24) and equating the real and imaginary parts of the resulting equation

$$K_P = \frac{\cos \theta}{M} \quad (7.34)$$

$$K_D = \frac{K_I}{\omega_{gc}^2} + \frac{\sin \theta}{M\omega_{gc}} \quad (7.35)$$

where  $\theta$ ,  $M$  and  $\psi$  are given by (7.27) and (7.28).

For a stable controller,  $K_P$  and  $K_D$  must be positive. Thus  $\omega_{gc}$  must be selected such that  $\theta$  in (7.27) will be less than  $90^\circ$ . Equations (7.34) and (7.35) are applied for the design of a PID controller. For PD or PI controllers, the appropriate gain is set to zero. Based on the above equations, the three functions **[numopen, denopen, denclsd] = frqpd(num, den)**, **[numopen, denopen, denclsd] = frqpi(num, den)** and **[numopen, denopen, denclsd] = frqpidd(num, den)** are developed for the PID controller design. **num** and **den** are the numerator and denominator of the polynomial coefficients of the open-loop plant transfer function. The user enters the desired phase margin. Each function finds and displays a compensated gain crossover frequency range for a stable controller. The user then specifies the crossover frequency in this range. For the PID controller the constant  $K_I$  must also be specified. The controller transfer function and the frequency domain specifications before and after compensation are found. Roots of the compensated characteristic equation are also computed. The function returns the open-loop and closed-loop numerators and denominators of the compensated system transfer function.

### 7.8.1 PD Controller

In a PD controller, both the error and its derivative are used for control, and the compensator transfer function is

$$G_c(s) = K_P + K_D s \quad (7.36)$$

From above, it can be seen that the PD controller is equivalent to the addition of a simple zero at  $s = -K_P/K_D$  to the open-loop transfer function. This improves the transient response. From a different point of view, the PD controller may also be used to improve the steady-state error, because it anticipates large errors, and attempts corrective action before they occur.

The function `[numopen, denopen, denclsd] = frdesign(num, den)` with option 4 is used for the PD controller design. Its use is demonstrated in the following example.

### Example 7.10

The open-loop transfer function of a control system is given by

$$G_c(s) = \frac{50}{s(s+1)(s+4)(s+5)} = \frac{50}{s^4 + 10s^3 + 29s^2 + 20s}$$

Design a PD controller for the above system to have a phase margin of  $50^\circ$ . Also find the time-domain performance specification for the compensated system. The commands

```
num = 50; den = [1 10 29 20 0];
[numopen, denopen, denclsd] = frdesign(num, den);
                                % Design function
timespec(numopen, denclsd);

result in
```

<u>Compensator type</u>	<u>Enter</u>
Gain compensation	1
Phase-lead	2
Phase-lag	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice → 4

Enter desired Phase Margin → 50

For a stable controller select a compensated gain crossover frequency wgc between 0.509 and 2.6

Enter wgc → 1.2

Uncompensated control system

Gain Margin = 1.08 Gain crossover w = 1.36

Phase Margin = 2.48 Phase crossover w = 1.42

Controller transfer function

Gc = 0.613155 + 0.4347 s

Row vectors of polynomial coefficients of the compensated system:

## 166 7. Frequency Response Analysis and Design

```
Open-loop num.    0    0    0  21.7350  30.6577
Open-loop den.    1   10   29      20      0
Closed-loop den   1   10   29  41.7350  30.6577
```

Gain Margin = 6.71 Gain crossover  $\omega = 1.2$   
Phase Margin = 50 Phase crossover  $\omega = 4.07$

Peak Mag. = 1.19  $\omega_r = 1.1$  Bandwidth = 2.04

Roots of the compensated characteristic equation:  
-6.3470  
-1.9390  
-0.8570 + 1.3254i  
-0.8570 - 1.3254i

Peak time = 2.304 Percent overshoot = 18.1044  
Rise time = 0.972  
Settling time = 5.064

### 7.8.2 PI Controller

In a PI controller, the integral of the error as well as the error itself is used for control, and the compensator transfer function is

$$G_c(s) = K_P + \frac{K_I}{s} \quad (7.37)$$

The PI controller is extremely common in process control or regulating systems. Integral control bases its corrective action on the cumulative error integrated over time. The controller increases the type of system by 1 and is used to reduce the steady-state errors. The function `[numopen, denopen, denclsd] = frdesign(num, den)` with option 5 is used for the PI controller design. Its use is demonstrated in the following example.

#### Example 7.11

Design a PI controller for the system of Example 7.10 for a compensated system phase margin of  $50^\circ$ . Also, obtain the time domain performance specifications.

The following commands

```
num = 50; den = [1 10 29 20 0];
[numopen, denopen, denclsd] = frdesign(num, den);
                                % Design function
timespec(numopen, denclsd);

result in
```

<u>Compensator type</u>	<u>Enter</u>
Gain compensation	1

```

Phase-lead                2
Phase-lag                  3
PD Controller              4
PI Controller              5
PID Controller             6
To quit                    0
    
```

Enter your choice → 5

Enter desired Phase Margin → 50

For a stable controller select a compensated gain crossover frequency  $w_{gc}$  between 0.0509 and 0.458

Enter  $w_{gc}$  → 0.35

Uncompensated control system

Gain Margin = 1.08 Gain crossover  $w = 1.36$

Phase Margin = 2.48 Phase crossover  $w = 1.42$

Controller transfer function

$G_c = 0.146155 + 0.0105983/s$

Row vectors of polynomial coefficients of the compensated system:

```

Open-loop num.    0    0    0    0    7.3077    0.5299
Open-loop den.    1   10   29   20         0         0
Closed-loop den   1   10   29   20    7.3077    0.5299
    
```

Gain Margin = 6.7 Gain crossover  $w = 0.35$

Phase Margin = 50 Phase crossover  $w = 1.34$

Peak Mag. = 1.24  $w_r = 0.225$  Bandwidth = 0.538

Roots of the compensated characteristic equation:

```

-4.6041 + 0.4247i
-4.6041 - 0.4247i
-0.3493 + 0.3796i
-0.3493 - 0.3796i
-0.0932
    
```

Peak time = 8.025 Percent overshoot = 24.842

Rise time = 2.996

Settling time = 31.458

### 7.8.3 PID Controller

The PID controller is used to improve the dynamic response as well as to reduce or eliminate the steady-state error. The function **[numopen, denopen, denclsd] =**

**frdesign(num, den)** with option 6 is used for the PID controller design. Its use is demonstrated in the following example.

### Example 7.12

Design a PID controller for the system of Example 7.10 for a compensated system phase margin of  $50^\circ$ . Also, obtain the time domain performance specifications. Choose  $K_I$  to have a value of 0.01.

The following commands

```
num = 50; den = [1 10 29 20 0];
[numopen,denopen, denclsd] = frdesign(num,den);
                                % Design function
timespec(numopen, denclsd);

result in
```

Compensator type	Enter
Gain compensation	1
Phase-lead	2
Phase-lag	3
PD Controller	4
PI Controller	5
PID Controller	6
To quit	0

Enter your choice → 6

Enter the integrator gain  $K_I$  → 0.01

Enter desired Phase Margin → 50

For a stable controller select a compensated gain crossover frequency  $w_{gc}$  between 0.407 and 2.61

Enter  $w_{gc}$  → 1.2

Uncompensated control system

Gain Margin = 1.08 Gain crossover  $w$  = 1.36

Phase Margin = 2.48 Phase crossover  $w$  = 1.42

Controller transfer function

$G_c = 0.613155 + 00.01/s + 0.441645 s$

Row vectors of polynomial coefficients of the compensated system:

Open-loop num.	0	0	0	22.0822	30.6577	0.5
Open-loop den.	1	10	29	20	0	0
Closed-loop den	1	10	29.	42.0822	30.6577	0.5



Gain Margin = 6.69 Gain crossover  $\omega = 1.2$   
 Phase Margin = 50 Phase crossover  $\omega = 4.09$

Peak Mag. = 1.19  $\omega_r = 1.1$  Bandwidth = 2.04

Roots of the compensated characteristic equation:

-6.3632  
 -0.8774 + 1.3246i  
 -0.8774 - 1.3246i  
 -1.8653  
 -0.0167

Peak time = 2.28 Percent overshoot = 18.75

Rise time = 0.96

Settling time = 3.72

controlengineers.ir

---

## CHAPTER 8

---

### MODERN CONTROL DESIGN

The classical design techniques of Chapters 6 and 7 are based on the root-locus and frequency response that utilize only the plant output for feedback with a dynamic controller. In this final chapter on design, we employ modern control designs that require the use of all state variables to form a linear static controller. Modern control design is especially useful in multivariable systems; however, in this chapter the ideas of state-space design are illustrated using single-input, single-output systems.

One approach in modern control systems accomplished by the use of state feedback is known as *pole-placement design*. The pole-placement design allows all roots of the system characteristic equation to be placed in desired locations. This results in a regulator with constant gain vector  $\mathbf{K}$ .

The state-variable feedback concept requires that all states be accessible in a physical system, but for most systems this requirement is not met; i.e., some of the states are inaccessible. For systems in which all states are not available for feedback, a state estimator (observer) may be designed to implement the pole-placement design.

The other approach to the design of regulator systems is the optimal control problem where a specified mathematical performance criterion is minimized.

## 8.1 Pole-Placement Design

The control is achieved by feeding back the state variables through a regulator with constant gains. Consider the control system presented in the state-variable form

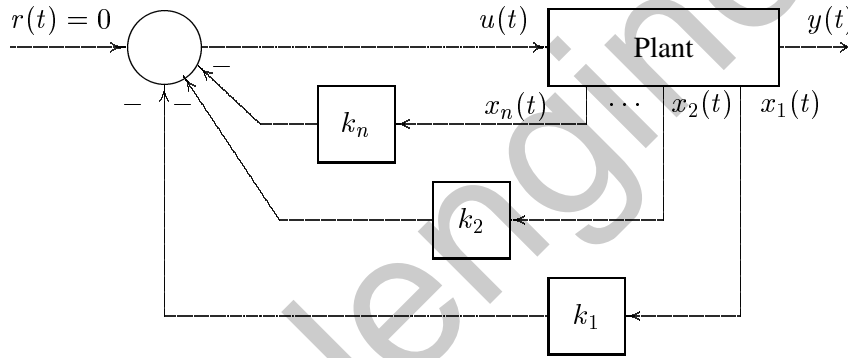
$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \quad (8.1)$$

$$y(t) = \mathbf{C}\mathbf{x}(t)$$

Consider the block diagram of the system shown in Figure 8.1 with the following state feedback control

$$u(t) = -\mathbf{K}\mathbf{x}(t) \quad (8.2)$$

where  $\mathbf{K}$  is a  $1 \times n$  vector of constant feedback gains. The control system input  $r(t)$  is assumed to be zero. The purpose of this system is to return all state variables to values of zero when the states have been perturbed.



**FIGURE 8.1**  
Control system design via pole placement.

Substituting (8.2) into (8.1), the closed-loop system state-variable representation is

$$\dot{\mathbf{x}}(t) = (\mathbf{A} - \mathbf{BK})\mathbf{x}(t) = \mathbf{A}_f\mathbf{x}(t) \quad (8.3)$$

The closed-loop system characteristic equation is

$$|s\mathbf{I} - \mathbf{A} + \mathbf{BK}| = 0 \quad (8.4)$$

Assume the system is represented in the phase variable canonical form as follows

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_2 & \dots & -a_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u(t) \quad (8.5)$$

Substituting for  $\mathbf{A}$  and  $\mathbf{B}$  into (8.4), the closed-loop characteristic equation for the control system is found

$$|s\mathbf{I} - \mathbf{A} + \mathbf{BK}| = s^n + (a_{n-1} + k_n)s^{n-1} + \cdots + (a_1 + k_2)s + (a_0 + k_1) = 0 \quad (8.6)$$

For the specified closed-loop pole locations  $-\lambda_1, \dots, -\lambda_n$ , the desired characteristic equation is

$$\alpha_c(s) = (s + \lambda_1) \cdots (s + \lambda_n) = s^n + \alpha_{n-1}s^{n-1} + \cdots + \alpha_1s + \alpha_0 = 0 \quad (8.7)$$

The design objective is to find the gain matrix  $\mathbf{K}$  such that the characteristic equation for the controlled system is identical to the desired characteristic equation. Thus, the gain vector  $\mathbf{K}$  is obtained by equating coefficients of equations (8.6) and (8.7).

$$k_i = \alpha_{i-1} - a_{i-1} \quad (8.8)$$

If the state model is not in the phase-variable canonical form, we can use the transformation technique of Chapter 3 Section 3.5 to transform the given state model to the phase-variable canonical form. The gain factor is obtained for this model and then transformed back to confirm with the original model. This procedure results in the following formula, known as *Ackermann's formula*.

$$\mathbf{K} = [0 \ 0 \ \cdots \ 0 \ 1] \mathbf{S}^{-1} \alpha_c(\mathbf{A}) \quad (8.9)$$

where the matrix  $\mathbf{S}$  is given by

$$\mathbf{S} = [B \ AB \ A^2B \ \cdots \ A^{n-1}B] \quad (8.10)$$

and the notation  $\alpha_c(\mathbf{A})$  is given by

$$\alpha_c(\mathbf{A}) = \mathbf{A}^n + \alpha_{n-1}\mathbf{A}^{n-1} + \cdots + \alpha_1\mathbf{A} + \alpha_0\mathbf{I} \quad (8.11)$$

The function  $[\mathbf{K}, \mathbf{A}_f] = \text{placepol}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{p})$  is developed for the pole-placement design.  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{C}$  are system matrices and  $\mathbf{p}$  is a row vector containing the desired closed-loop poles. This function returns the gain vector  $\mathbf{K}$  and the closed-loop system matrix  $\mathbf{A}_f$ . Also, the *MATLAB Control System Toolbox* contains two functions for pole-placement design. Function  $\mathbf{K} = \text{acker}(\mathbf{A}, \mathbf{B}, \mathbf{p})$  is for single input systems, and function  $\mathbf{K} = \text{place}(\mathbf{A}, \mathbf{B}, \mathbf{p})$ , which uses a more reliable algorithm, is for multi-input systems.

The condition that must exist to place the closed-loop poles at the desired location is to be able to transform the given state model into phase-variable canonical form. That is, the controllability matrix  $\mathbf{S}$ , given in (8.10), must have a nonzero determinant. This characteristic is known as *controllability*.

### Example 8.1

For the plant

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ -1 & -2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \mathbf{x}$$

design a state feedback control to place the closed-loop pole at  $-3 \pm j4$  and  $-8$ . Obtain the initial condition response, given  $x_1(0) = 1$ ,  $x_2(0) = 1$  and  $x_3(0) = -1$ . The following commands

```
A=[-1 0 0; -1 -2 0; 1 0 0];
B=[1; 0; 0];
C=[1 1 0]; D=0;
j=sqrt(-1);
P=[-3+j*4 -3-j*4 -8]; % desired closed-loop poles
[K,Af]=placepol(A,B,C,P); % returns gain K and closed-loop
                             % system matrix
% initial condition response
t=0:.02:2;
r=zeros(1,length(t)); % generates a row of zero input
x0=[1 1 -1]; % initial state
[y,x]=lsim(Af, B, C, D, r, t, x0); % initial state response
subplot(2,2,1), plot(t, x(:,1)),title('x_1(t)'), grid
subplot(2,2,2), plot(t, x(:,2)),title('x_2(t)'), grid
subplot(2,2,3), plot(t, x(:,3)),title('x_3(t)'), grid
subplot(2,2,4), plot(t, y),title('y(t)'), grid
subplot(111)
```

result in

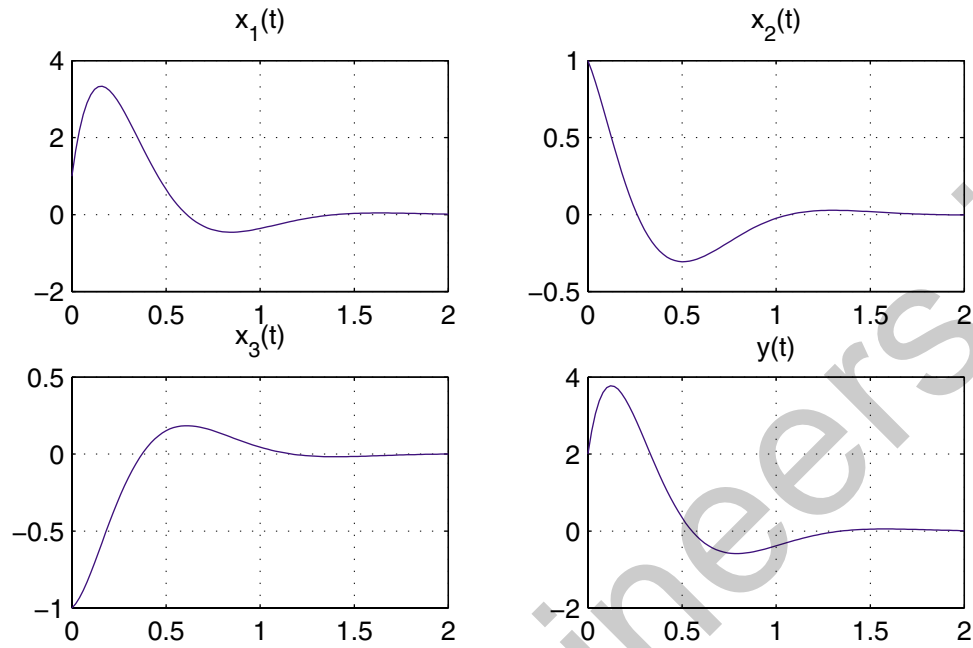
```
Feedback gain vector K
    11    51   100

Uncompensated Plant
Transfer function:
      s^2 + s
    -----
s^3 + 3 s^2 + 2 s

Compensated system closed-loop
Transfer function:
      s^2 + s
    -----
s^3 + 14 s^2 + 73 s + 200

Compensated system matrix A - B*K
   -12   -51  -100
    -1    -2     0
```

The results are given in Figure 8.2.



**FIGURE 8.2**  
Initial condition response for Example 8.1.

### Example 8.2

A single-input single-output plant has the transfer function

$$G(s) = \frac{4}{s(s+1)(s+4)}$$

Obtain a state model for the plant and design a state feedback that will place the closed-loop pole at  $-2 \pm j2$  and  $-5$ . Also, obtain the closed-loop transfer function for the controlled system.

The following commands

```

num=4; den=[1 5 4 0];
[A, B, C, D]=tf2ss(num, den)    % converts transfer function
                                % to state model
j=sqrt(-1);
P=[-2+j*2 -2-j*2 -5];          % desired closed-loop poles
[K,Af]=placepol(A,B,C,P);       % returns gain K & closed-loop
                                % system matrix
    
```

result in

$$A = \begin{bmatrix} -5 & -4 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

```

      0      1      0
B =
      1
      0
      0
C =
      0      0      4
D =
      0
Feedback gain vector K
      4      24      40
Uncompensated Plant
Transfer function:
              4
-----
s^3 + 5 s^2 + 4 s
Compensated system closed-loop
Transfer function:
              4
-----
s^3 + 9 s^2 + 28 s + 40
Compensated system matrix A - B*K
      -9      -28      -40
      1         0         0
      0         1         0

```

From the above results the closed-loop transfer function for the controlled plant is

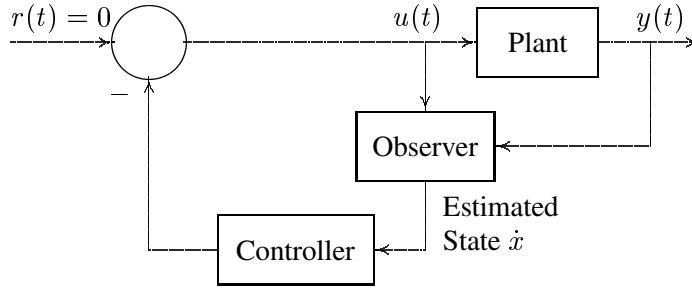
$$T(s) = \frac{C(s)}{R(s)} = \frac{4}{s^3 + 9s^2 + 28s + 40}$$

## 8.2 Controllability

A system is said to be controllable when the plant input  $u$  can be used to transfer the plant from any initial state to any arbitrary state in a finite time. The plant described by (8.1) with the system matrix having dimension  $n \times n$  is completely state controllable if and only if the controllability matrix  $\mathbf{S}$  in (8.10) has a rank of  $n$ . The function  $\mathbf{S} = \text{ctrb}(\mathbf{A}, \mathbf{B})$  is developed which returns the controllability matrix  $\mathbf{S}$  and determines whether or not the system is state controllable.

## 8.3 Observer Design

In the pole-placement approach to the design of control systems, it was assumed that all state variables are available for feedback. However, in practice it is impractical to install all the transducers which would be necessary to measure all of the states. If the state variables are not available because of system configuration or cost, an observer



**FIGURE 8.3**  
State feedback design with an observer.

or estimator may be necessary. The observer is an estimator algorithm based on the mathematical model of the system. The observer creates an estimate  $\hat{\mathbf{x}}(t)$  of the states from the measurements of the output  $y(t)$ . The estimated states, rather than the actual states, are then fed to the controller. One scheme is shown in Figure 8.3.

Consider a system represented by the state and output equations

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \quad (8.12)$$

$$y(t) = \mathbf{C}\mathbf{x}(t) \quad (8.13)$$

Assume that the state  $\mathbf{x}(t)$  is to be approximated by the state  $\hat{\mathbf{x}}(t)$  of the dynamic model

$$\dot{\hat{\mathbf{x}}}(t) = \mathbf{A}\hat{\mathbf{x}}(t) + \mathbf{B}u(t) + \mathbf{G}(y(t) - \hat{y}(t)) \quad (8.14)$$

$$\hat{y}(t) = \mathbf{C}\hat{\mathbf{x}}(t) \quad (8.15)$$

Subtracting (8.14) from (8.12), and (8.15) from (8.13), we have

$$(\dot{\mathbf{x}}(t) - \dot{\hat{\mathbf{x}}}(t)) = \mathbf{A}(\mathbf{x}(t) - \hat{\mathbf{x}}(t)) - \mathbf{G}(y(t) - \hat{y}(t)) \quad (8.16)$$

$$(y(t) - \hat{y}(t)) = \mathbf{C}(\mathbf{x}(t) - \hat{\mathbf{x}}(t)) \quad (8.17)$$

where  $\mathbf{x}(t) - \hat{\mathbf{x}}(t)$  is the error between the actual state vector and the estimated vector, and  $y(t) - \hat{y}(t)$  is the error between the actual output and the estimated output. Substituting the output equation into the state equation, we obtain the equation for the error between the estimated state vector and the actual state vector.

$$\dot{\mathbf{e}}(t) = (\mathbf{A} - \mathbf{G}\mathbf{C})\mathbf{e}(t) = \mathbf{A}_e\mathbf{e}(t) \quad (8.18)$$

where

$$\mathbf{e}(t) = \mathbf{x}(t) - \hat{\mathbf{x}}(t) \quad (8.19)$$

If  $\mathbf{G}$  is chosen such that eigenvalues of matrix  $\mathbf{A} - \mathbf{G}\mathbf{C}$  all have negative real parts, then the steady-state value of the estimated state vector error  $\mathbf{e}(t)$  for any initial condition will tend to zero. That is,  $\hat{\mathbf{x}}(t)$  will converge to  $\mathbf{x}(t)$ . The design of the observer is similar to the design of the controller. However, the eigenvalues of  $\mathbf{A} - \mathbf{G}\mathbf{C}$  must



be selected to the left of the eigenvalues of  $\mathbf{A}$ . This ensures that the observer dynamic is faster than the controller dynamic for providing a rapid updated estimate of the state vector.

The estimator characteristic equation is given by

$$|s\mathbf{I} - \mathbf{A} + \mathbf{G}\mathbf{C}| = 0 \quad (8.20)$$

For a specified speed of response, the desired characteristic equation for the estimator is

$$\alpha_e(s) = s^n + \alpha_{n-1}s^{n-1} + \cdots + \alpha_1s + \alpha_0 = 0 \quad (8.21)$$

Thus, the estimator gain  $\mathbf{G}$  is obtained by equating coefficients of (8.20) and (8.21). This is identical to the pole-placement technique, and  $\mathbf{G}$  is found by the application of Ackermann's formula

$$\mathbf{G} = \alpha_e(\mathbf{A}) \begin{bmatrix} \mathbf{C} \\ \mathbf{C}\mathbf{A} \\ \vdots \\ \mathbf{C}\mathbf{A}^{n-1} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \quad (8.22)$$

and the notation  $\alpha_e(\mathbf{A})$  is given by

$$\alpha_e(\mathbf{A}) = \mathbf{A}^n + \alpha_{n-1}\mathbf{A}^{n-1} + \cdots + \alpha_1\mathbf{A} + \alpha_0\mathbf{I} \quad (8.23)$$

The function  $[\mathbf{G}, \mathbf{A}_e] = \text{observer}(\mathbf{A}, \mathbf{B}, \mathbf{C}, p_e)$  is developed for the estimator.  $p_e$  is the desired estimator eigenvalues. This function returns the gain vector  $\mathbf{G}$  and the closed-loop system matrix  $\mathbf{A}_f$ .

The necessary condition for the design of an observer is that all the states can be observed from the measurements of the output. This characteristic is known as *observability*.

### Example 8.3

For the plant

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 16 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}$$

design a full-state observer such that the observer is critically damped with eigenvalues at  $-8$  and  $-8$ .

The following commands

```
A=[ 0 1; 16 0];
B=[0; 1];
C=[1 0]; D=0;
Pe=[-8 -8]; % desired observer eigenvalues
[K,Ae]= observer(A,B,C,Pe); % returns gain K and closed-loop
```

result in

Estimator gain vector  $G$

16

80

Open-loop Plant

Transfer function:

1

-----  
 $s^2 - 16$

Error matrix  $A - G \cdot C$

-16      1

-64      0

## 8.4 Observability

A system is said to be observable if the initial vector  $\mathbf{x}(t)$  can be found from the measurement of  $u(t)$  and  $y(t)$ . The plant described by (8.12) is completely state observable if the inverse matrix in (8.22) exists. The function **V=obsvable(A, C)** returns the observability matrix **V** and determines whether or not the system is state observable.

## 8.5 Combined Controller-Observer Design

Consider the system represented by the state and output equations (8.12) and (8.13) with the state feedback control based on the observed state  $\hat{\mathbf{x}}(t)$  given by

$$u(t) = -\mathbf{K}\hat{\mathbf{x}}(t) \quad (8.24)$$

Substituting in (8.12), the state equation becomes

$$\dot{\mathbf{x}}(t) = (\mathbf{A} - \mathbf{BK})\mathbf{x}(t) + \mathbf{BK}\mathbf{e}(t) \quad (8.25)$$

Combining the above equation with the error equation given by (8.19), we have

$$\begin{bmatrix} \dot{\mathbf{x}}(t) \\ \dot{\mathbf{e}}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A} - \mathbf{BK} & \mathbf{BK} \\ \mathbf{0} & \mathbf{A} - \mathbf{GC} \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{e}(t) \end{bmatrix} \quad (8.26)$$

The function **[K, G, A<sub>c</sub>] = placeobs(A, B, C, p, p<sub>e</sub>)** is developed for the combined controller-observer design. **A** is the system matrix, **B** is the input column vector, and **C** is the output row vector. **p** is a row vector containing the desired closed-loop poles and **p<sub>e</sub>** is the desired observer eigenvalues. The function displays the gain vectors **K** and **G**, open-loop plant transfer function, and the controlled system closed-loop transfer function. Also, the function returns the gain vector **K**, and the combined system matrix in (8.26).

### Example 8.4

For the system of Example 8.3, design a controller-observer system such that the desired closed-loop poles for the system are at  $-1 \pm j2$ . Choose the eigenvalues of the observer gain matrix to be  $p_{e1} = p_{e2} = -8$ .

The following commands

```
A=[ 0 1; 16 0];
B=[0; 1];
C=[1 0]; D=0;
j=sqrt(-1);
P=[-1+j*2 -1-j*2]; % desired regulator roots
Pe=[-8 -8]; % desired observer roots
[K,G,Af]= placeobs(A,B,C,P,Pe); % returns gain K,G & closed-loop
% system matrix
```

result in

```
Feedback gain vector K
    21    2
Estimator gain vector G:
    16
    80
Open-loop Plant
Transfer function:
    1
-----
s^2 - 16
Controller-estimator
Transfer function:
    496 s + 2192
-----
s^2 + 18 s + 117
Controlled system closed-loop
Transfer function:
    496 s + 2192
-----
s^4 + 18 s^3 + 101 s^2 + 208 s + 320
Combined controller observer system matrix
    0    1    0    0
   -5   -2   21    2
    0    0  -16    1
    0    0  -64    0
```

From the above results the controller-observer transfer function is

$$G_{ce} = \frac{496s + 2192}{s^2 + 18s + 117}$$

The closed-loop transfer function for the controlled plant is

$$T(s) = \frac{C(s)}{R(s)} = \frac{496s + 2192}{s^4 + 18s^3 + 101s^2 + 208s + 320}$$

## 8.6 Optimal Regulator Design

The object of the optimal regulator design is to determine the optimal control law  $\mathbf{u}^*(\mathbf{x}, t)$  which can transfer the system from its initial state to the final state (with zero system input) such that a given performance index is minimized. The performance index is selected to give the best trade-off between performance and cost of control. The performance index that is widely used in optimal control design is known as the *quadratic performance index* and is based on minimum-error and minimum-energy criteria.

Consider the plant described by

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \quad (8.27)$$

The problem is to find the vector  $\mathbf{K}(t)$  of the control law

$$\mathbf{u}(t) = -\mathbf{K}(t)\mathbf{x}(t) \quad (8.28)$$

which minimizes the value of a quadratic performance index  $\mathbf{J}$  of the form

$$\mathbf{J} = \int_{t_0}^{t_f} (\mathbf{x}'\mathbf{Q}\mathbf{x} + \mathbf{u}'\mathbf{R}\mathbf{u})dt \quad (8.29)$$

subject to the dynamic plant equation in (8.27). In (8.29)  $\mathbf{Q}$  is a positive semidefinite matrix and  $\mathbf{R}$  is a real symmetric matrix.  $\mathbf{Q}$  is positive semidefinite if all its principal minors are nonnegative. The choice of the elements of  $\mathbf{Q}$  and  $\mathbf{R}$  allows the relative weighting of individual state variables and individual control inputs.

To obtain a formal solution, we can use the method of *Lagrange multipliers*. The constraint problem is solved by augmenting (8.27) into (8.29) using an  $n$ -vector of *Lagrange multipliers*,  $\lambda$ . The problem reduces to the minimization of the following unconstrained function

$$\mathcal{L}(\mathbf{x}, \lambda, u, t) = [\mathbf{x}'\mathbf{Q}\mathbf{x} + \mathbf{u}'\mathbf{R}\mathbf{u}] + \lambda'[\mathbf{A}\mathbf{x} + \mathbf{B}u - \dot{\mathbf{x}}] \quad (8.30)$$

The optimal values (denoted by the subscript  $*$ ) are found by equating the partial derivatives to zero.

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \mathbf{A}\mathbf{X}^* + \mathbf{B}\mathbf{u}^* - \dot{\mathbf{x}}^* = \mathbf{0} \quad \Rightarrow \quad \dot{\mathbf{x}}^* = \mathbf{A}\mathbf{X}^* + \mathbf{B}\mathbf{u}^* \quad (8.31)$$

$$\frac{\partial \mathcal{L}}{\partial u} = 2\mathbf{R}\mathbf{u}^* + \lambda'\mathbf{B} = \mathbf{0} \quad \Rightarrow \quad \mathbf{u}^* = -\frac{1}{2}\mathbf{R}^{-1}\lambda'\mathbf{B} \quad (8.32)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = 2\mathbf{x}'^*\mathbf{Q} + \dot{\lambda}' + \lambda'\mathbf{A} = \mathbf{0} \quad \Rightarrow \quad \dot{\lambda} = -2\mathbf{Q}\mathbf{x}^* - \mathbf{A}'\lambda \quad (8.33)$$

Assume that there exists a symmetric, time varying positive definite matrix  $\mathbf{p}(t)$  satisfying

$$\lambda = 2\mathbf{p}(t)\mathbf{x}^* \quad (8.34)$$

Substituting (8.34) into (8.32) gives the optimal closed-loop control law

$$\mathbf{u}^*(t) = -\mathbf{R}^{-1}\mathbf{B}'\mathbf{p}(t)\mathbf{x}^* \quad (8.35)$$

Obtaining the derivative of (8.34), we have

$$\dot{\lambda} = 2(\dot{\mathbf{p}}\mathbf{x}^* + \mathbf{p}\dot{\mathbf{x}}^*) \quad (8.36)$$

Finally equating (8.33) with (8.36), we obtain

$$\dot{\mathbf{p}}(t) = -\mathbf{p}(t)\mathbf{A} - \mathbf{A}'\mathbf{p}(t) - \mathbf{Q} + \mathbf{p}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}'\mathbf{p}(t) \quad (8.37)$$

The above equation is referred to as the matrix *Riccati equation*. The boundary condition for (8.37) is  $\mathbf{p}(t_f) = \mathbf{0}$ . Therefore, (8.37) must be integrated backward in time. Since a numerical solution is performed forward in time, a dummy time variable  $\tau = t_f - t$  is replaced for time  $t$ . Once the solution to (8.37) is obtained the solution of the state equation (8.31) in conjunction with the optimum control equation (8.35) is obtained.

The function `[τ, p, K, t, x]=riccati` is developed for the time-domain solution of the Riccati equation. The function returns the solution of the matrix Riccati equation,  $\mathbf{p}(\tau)$ , the optimal feedback gain vector  $\mathbf{k}(\tau)$ , and the initial state response  $\mathbf{x}(t)$ . In order to use this function, the user must declare the function `[A, B, Q, R, t0, tf, x0]=system` ( $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}, t_0, t_f, \mathbf{x}_0$ ) containing system matrices and the performance index matrices in an M-file named **system.m**.

For linear time-invariant systems, since  $\dot{\mathbf{p}} = \mathbf{0}$ , when the process is of infinite duration, that is  $t_f = \infty$ , (8.37) reduces to the algebraic Riccati equation

$$\mathbf{p}\mathbf{A} + \mathbf{A}'\mathbf{p} + \mathbf{Q} - \mathbf{p}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}'\mathbf{p} = \mathbf{0} \quad (8.38)$$

The *MATLAB Control System Toolbox* function `[k, p]=lqr2(A, B, Q, R)` can be used for the solution of the algebraic Riccati equation.

### Example 8.5

Design a state feedback system for the plant described by

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -4 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$

$$y = [1 \quad 0 \quad 0] \mathbf{x}$$

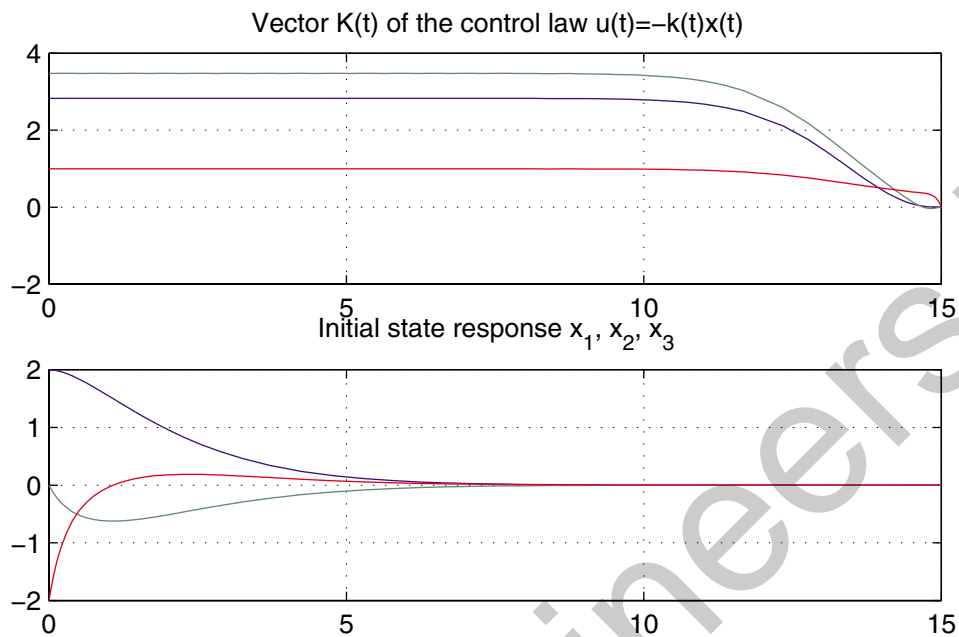
Find the optimal control law to minimize the performance index

$$\mathbf{J} = \int_0^{10} \left( 4x_1^2(t) + 3x_2^2 + 2x_3^2 + \frac{1}{2}u^2 \right) dt \quad (8.39)$$

The admissible states and control values are unconstrained. The states are initially at  $x_1(0) = 2$ ,  $x_2(0) = 0$  and  $x_3(0) = -2$ . For this system we have

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -4 & -5 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \quad \text{and} \quad R = \frac{1}{2}$$

First an M-file named **system.m** is created and the function `[A, B, Q, R, t0, tf, x0] = system(A, B, Q, R, t0, tf, x0)` is defined as follows.



**FIGURE 8.4**

Optimal feedback gain vector  $\mathbf{K}(t)$  and initial condition response  $\mathbf{x}(t)$ .

```
function [A,B,Q,R,t0,tf,x0]=system(A,B,Q,R,t0,tf,x0)
A=[0 1 0; 0 0 1; 0 -4 -5]; B=[0;0; 1];
Q=[4 0 0; 0 3 0; 0 0 2]; R=.5;
t0=0; tf=15;
x0=[2 0 -2];
```

The above function is saved in an M-file named **system.m**. Then, the following commands

```
[tt,p,k,t,x]=riccati
subplot(2,1,1), plot(tt,k),
title('Vector K(t) of the control law u(t)=-k(t)x(t)'),
grid,
subplot(2,1,2), plot(t,x),
title('Initial state response x_1, x_2, x_3'), grid
subplot(111)
```

result in the control law and the response is given in Figure 8.4.

### Example 8.6

For Example 8.5 use the *MATLAB Control System Toolbox* **lqr2** to obtain the solution to the algebraic Riccati equation.

The following commands

```

A=[0 1 0; 0 0 1;0 -4 -5]; B=[0;0; 1];
Q=[4 0 0; 0 3 0;0 0 2];R=.5;
[K, p]=lqr2(A,B,Q,R)
    
```

result in

```

K =
    2.8284    3.4780    0.9963
p =
    10.5755    8.4801    1.4142
     8.4801   11.0060    1.7390
     1.4142    1.7390    0.4982
    
```

controlengineers.ir

## List of Author-Developed Functions

<b>cntrable</b>	determines system controllability
<b>cnyquist</b>	maps the Nyquist path via complex function $GH(s)$
<b>electsys</b>	returns the state derivatives for Example 2.2
<b>errorrf</b>	steady-state error, transfer function in polynomial form
<b>errorzp</b>	steady-state error, transfer function in zero pole form
<b>fbdesign</b>	minor-loop feedback design program
<b>frcntrl</b>	frequency response design equations
<b>frdesign</b>	frequency response design program
<b>frqlag</b>	frequency response design phase-lag controller
<b>frqlead</b>	frequency response design phase-lead controller
<b>frqp</b>	frequency response design P controller
<b>frqpd</b>	frequency response design PD controller
<b>frqpi</b>	frequency response design PI controller
<b>frqpid</b>	frequency response design PID controller
<b>frqspec</b>	frequency response performance specifications
<b>ghs</b>	returns magnitude and phase of the complex function $GH(s)$
<b>ltstm</b>	Laplace transform of state transition matrix
<b>mechsys</b>	returns the state derivatives for Example 2.1
<b>observer</b>	observer design
<b>obsvable</b>	determines system observability
<b>pcomp</b>	root-locus design P controller
<b>pdcomp</b>	root-locus design PD controller
<b>pdlead</b>	root-locus design phase-lead controller
<b>pendulum</b>	returns the state derivatives for Example 2.3
<b>phlag</b>	root-locus design phase-lag controller
<b>picomp</b>	root-locus design PI controller
<b>pidcomp</b>	root-locus design PID controller
<b>placeobs</b>	combined controller-observer design
<b>placepol</b>	pole-placement design
<b>pnetfdbk</b>	feedback compensation using passive elements
<b>riccasim</b>	returns state derivative of Riccati equation
<b>riccati</b>	optimal regulator design
<b>rldesign</b>	root-locus design program
<b>routh</b>	Routh-Hurwitz array
<b>ss2phv</b>	transformation to control canonical form
<b>statesim</b>	returns state derivatives for use in Riccati equation
<b>stepzwn</b>	step response: $\frac{\omega_n^2(1+as)}{(1+Ts)(s^2+2\zeta\omega_ns+\omega_n^2)}$
<b>stm</b>	determines the state transition matrix $\phi(t)$
<b>system</b>	system matrices defined for use in Riccati equation
<b>tachfdbk</b>	tachometer feedback control
<b>timespec</b>	time-domain performance specifications
<b>rldesigngui</b>	GUI program for root-locus design



### List of M-files for Chapter Examples

CH1EX01	CH1EX02	CH1EX03	CH1EX04	CH1EX05
CH1EX06	CH1EX07	CH1EX08	CH1EX09	CH1EX10
CH1EX11	CH1EX12	CH1EX13	CH1EX14	CH1EX15
CH1EX16	CH1EX17	CH1EX18	CH1EX19	CH1EX20
CH2EX21	CH2EX01	CH2EX02	CH2EX03	CH2EX04
CH2EX05	CH2EX06	CH2EX07	CH2EX08	CH2EX09
CH3EX01	CH3EX02	CH3EX03	CH3EX04	CH3EX05
CH3EX06	CH3EX07	CH3EX08	CH3EX09	CH3EX10
CH3EX11	CH4EX01	CH4EX02	CH4EX03	CH4EX04
CH4EX05	CH4EX06	CH4EX07	CH4EX08	CH5EX01
CH5EX02	CH5EX03	CH5EX04	CH5EX05	CH5EX06
CH5EX07	CH6EX01	CH6EX02	CH6EX03	CH6EX04
CH6EX05	CH6EX06	CH6EX07	CH6EX08	CH6EX09
CH6EX10	CH6EX11	CH6EX12	CH6EX13	CH6EX14
CH6EX15	CH6EX16	CH6EX17	CH7EX01	CH7EX02
CH7EX03	CH7EX04	CH7EX05	CH7EX06	CH7EX07
CH7EX08	CH7EX09	CH7EX10	CH7EX11	CH7EX12
CH8EX01	CH8EX02	CH8EX03	CH8EX04	CH8EX05
CH8EX06	simexa22	simexa23	simexa24	simexa25
simexa26	simexa27	simexa28	simexa29	

### References

1. Brogan, W. L., *Modern Control Theory*, Prentice-Hall, 1982.
2. Clark, R. N., *Introduction to Automatic Control Systems*, John Wiley, 1962.
3. Close, D. R., *State Variables for Engineers*, John Wiley, 1965.
4. D'Azzo, J. J., *Linear Control System Analysis*, McGraw-Hill, 1988.
5. Dorf, R. C., *Modern Control Systems*, Addison-Wesley, 1989.
6. Franklin, G. F., Powell, J. D. and Emami-Naeini, E., *Feedback Control of Dynamic Systems*, Addison-Wesley, 1986.
7. Golten, J., and Verwer, A., *Control System Design and Simulation*, McGraw-Hill, 1991.
8. Hill, R. H., *Experiments in Computational Matrix Algebra*, Random House, 1988.
9. Kuo, B.C., *Automatic Control Systems*, Prentice-Hall, 1991.
10. Ogata, K., *Modern Control Engineering*, Prentice-Hall, 1990.
11. Palm, W. J., *Control Systems Engineering*, John Wiley, 1986.

12. Phillips, C. L. and Harbor, R. D., *Feedback Control Systems*, Prentice-Hall, 1991.
13. Rowland, J. R., *Linear Control Systems*, John Wiley, 1986.
14. Scharf, L. L., and Behrens, R. T., *A First Course in Electrical and Computer Engineering*, Addison-Wesley, 1990.
15. The Math Works Inc., *The Student Edition of MATLAB*, Prentice-Hall, 1992.

controlengineers.ir

## INDEX

- ABCD* constants, 144  
*B*-coefficients, 280  
*H* constant, 463  
 $\Delta$ -Y transformation, 35  
 $\Delta$ -connected loads, 34  
 Proportional controller, 662  
 AC resistance, 105  
 Acceleration factor, 198  
 Ackermann's formula, 571  
 Active power, 16  
 Admittance matrix, 192  
 All-aluminum alloy conductor, 104  
 All-aluminum conductor, 104  
 Alternators, 49  
 Aluminum conductor alloy-reinforced, 104  
 Aluminum conductor steel-reinforced, 104  
 Amplifier model, 555  
 Annual load factor, 8  
 ANSI, 104  
 Apparent power, 16  
 Area control error (ACE), 551  
 Armature mmf, 52  
 Armature reaction, 53  
 Armature short circuit time constants, 340  
 Array powers, 594  
 Attenuation constant, 153  
 Automatic generation control, 542  
 Automatic voltage regulator, 555  
 Autotransformers, 77  
 Average power, 16  
 Axis, 606  
 Balanced fault, 353  
 Balanced three-phase circuits, 30  
 Balanced three-phase fault, 354  
 Balanced three-phase power, 37  
 Balanced three-phase short circuit, 325  
 Bandwidth, 661  
 Base current, 89  
 Base impedance, 89  
 Base volt-ampere, 89  
 Base voltage, 89  
 Basic loops, 370  
 Bode plot, 657  
 Bolted fault, 354  
 Branches of a tree, 369  
 Brushless excitation, 49  
 Building algorithm, 369  
 Bundling, 105  
 Bus, 4  
 Bus admittance matrix, 192  
 Bus code, 223  
 Bus data file, 223  
 Bus impedance matrix, 193, 369  
 Bus voltages during fault, 366, 434  
 Capacitance of single-phase lines, 121  
 Capacitance of Three-phase lines, 124  
 Capacitance of three-phase two-circuit lines, 126

- Case-sensitive, 589
- Change of base, 90
- Character string, 592
- Characteristic impedance, 153
- Characteristic polynomial, 601
- Circle diagram, 163
- Circular mils, 104
- Closed-loop frequency response, 661
- Coherent, 511
- Colon, 595
- Column vector, 593
- Complex numbers, 599
- Complex power, 19
- Complex power balance, 21
- Complex power flow, 26
- Composite load, 530
- Control area, 545
- Coordination equations, 270
- Copper loss, 68
- Corona, 135
- Cost function, 268
- Cotree, 369
- Critical clearing angle, 493, 496
- Critical clearing time, 494, 507
- Current waves, 156
- Current-carrying capacity, 163
- Cut set, 370
- Cylindrical rotor generator, 56
  
- Daily-load curve, 8
- Daily-load factor, 8
- Damped frequency of oscillation, 474
- Damper, 49
- Damping power, 473
- Damping ratio, 474, 644
- DC component, 341
- DC components of stator currents, 340
- DC offset, 316
- DC resistance, 105
- DC transmission tie line, 2
- Decoupled power flow, 240
- Deregulation, 3
- Derivation of loss formula, 289
- Direct axis, 318
- Direct axis reactance, 64
- Direct axis reluctance, 63
- Direct axis subtransient reactance, 336
- Direct axis synchronous reactance, 338, 342
- Distribution, primary, 6
- Distribution, secondary, 8
- Diversity, 9
- Division of polynomials, 603
- Dot product, 594
- Double line-to-ground fault, 425, 434
- Driving point admittance, 192
- Dynamic stability, 460
  
- Economic dispatch neglecting losses, 268
- Economic dispatch, generator limits, 276
- Economic dispatch, transmission losses, 279
- Edison, Thomas, 1
- Effect of bundling on capacitance, 126
- Effect of earth on capacitance, 127
- Effect of load current, 347
- Eigenvalues, 259, 599
- Electric field intensity, 121
- Electric industry structure, 2
- Electrostatic induction, 135
- Element-by-element division, 594
- Element-by-element multiplication, 594
- Elementary matrix operation, 596
- Energy control center, 11, 528
- Equal-area criterion, 486
- Equivalent  $\pi$  model, 154
- Equivalent circuit of transformer, 64
- Equivalent leakage impedance, 69
- Excitation voltage, 53
- Exciter, 49
- Exciter model, 556
- Extra-high voltage, 2, 104
  
- Fast decoupled power flow solution, 240
- Fault analysis using  $Z_{bus}$ , 363
- Flux linkage, 50, 106, 320
- Frequency bias factor, 548
- Frequency response, 657
- Frequency response design, 662
- Fuel-cost curve, 267
- Function file, 588

- Fundamental cut set, 370
- Gain factor, 641
- Gain margin, 659
- Gauss-Seidel, 195
- Gauss-Seidel power flow solution, 209
- Generalized circuit constants, 144
- Generation, 4
- Generator model, 529, 557
- Generator voltage regulation, 55
- Geometric mean distance, 110
- Geometric mean radius, 110
- GMR of bundle conductors, 118
- Governor model, 532
- Gradient method, 263, 270, 283
- Gradient vector, 259
- Graph of network, 369
- Graphics, 605
- Graphics hard copy, 607
- Handle graphics, 614
- Heat rate, 267
- Help, 587
- Help Desk, 588
- Hessian matrix, 259
- Hyperbolic functions, 154
- Ideal transformer, 65
- Impedance matrix, 193, 369
- Impedance triangle, 20
- Incident wave, 156
- Incremental fuel cost, 267
- Incremental fuel-cost curve, 267
- Incremental production cost, 270
- Incremental transmission loss, 281
- Inductance due to external flux linkage, 108
- Inductance of single conductor, 106
- Inductance of composite conductors, 115
- Inductance of single-phase lines, 109
- Inductance of three-phase lines, 112
- Inductance of three-phase two-circuit lines, 119
- Inductance spacing factor, 110
- Inductances of salient-pole machines, 320
- Inequality constraints, 264
- Inertia constant, 463
- Infinite bus, 56
- Inner product, 594
- Input-output curve, 267
- Installed generation capacity, 4
- Installing Text toolbox, 587
- Instantaneous power, 15
- Integral controller, 542
- Internal flux linkage, 107
- Internal inductance, 107
- Iron loss, 69
- Jacobian matrix, 204, 233
- Kinetic energy, 462
- Kron reduction formula, 513
- Kron's loss formula, 279
- Kuhn-Tucker, 265, 276
- Lagrange multiplier, 260, 280, 577
- Line compensation, 165
- Line currents, 435
- Line data file, 223
- Line flows, 212
- Line inductance, 120
- Line loadability equation, 164
- Line losses, 212
- Line performance program, 171
- Line resistance, 105
- Line voltage, 32
- Line voltage regulation, 144
- Line-to-line fault, 423, 433
- Line-to-line short circuit, 330, 333
- Linear quadratic regulator, 576
- Links of a cotree, 369
- Load bus, 208
- Load flow, 189
- Load frequency control, 528
- Load impedance, 90
- Load model, 530
- Loads, 8
- Logical statements, 614
- Long line model, 151
- Loops, 614
- Loss coefficients, 280

- Machine model for transient analyses, 335  
 Magnetic field induction, 133  
 Magnetic field intensity, 106  
 Magnetic flux density, 107  
 Matrix division, 597  
 Matrix multiplication, 597  
 Medium length lines, 147  
 Medium line model, 147  
 Mil, 104  
 Minimum phase transfer function, 660  
 Moment of inertia, 461  
 Momentary duty, 341  
 Multimachine system, 511  
 Multimachine transient stability, 514  
 Mutual inductance, 111  
  
 Negative phase sequence, 30, 401  
 Newton-Raphson, 200  
 Newton-Raphson power flow solution, 232  
 Nominal  $\pi$  model, 147  
 Nonlinear algebraic equations, 195  
 Nonlinear function optimization, 258  
 Nonlinear programming, 258  
 Nonlinear systems, 620  
 Numerical solution of swing equation, 504  
 Nyquist, 661  
 Nyquist diagram, 660  
 Nyquist path, 660  
 Nyquist plot, 658  
 Nyquist stability criterion, 660  
  
 One vector, 595  
 One-line diagram, 91  
 One-machine system connected to infinite bus, 472  
 Open circuit transient time constant, 337  
 Open line, 167  
 Open-circuit test, 68  
 Operating cost of thermal plant, 267  
 Optimal control design, 576  
 Optimal dispatch of generation, 257  
 Output format, 590  
 Overhead transmission lines, 103  
  
 Overshoot, 644  
  
 P-Q bus, 208  
 P-V bus, 208  
 Pacific Intertie, 2  
 Park transformation, 321  
 Partial fraction expansion, 604  
 Path Browser, 587  
 PD controller, 646, 649  
 Peak load, 8  
 Peak time, 644  
 Penalty factor, 282  
 Per phase basis, 37  
 Per-unit system, 88  
 Permeability of free space, 107  
 Permittivity of free space, 121  
 Phase constant, 153  
 Phase margin, 659  
 Phase sequence, 30  
 Phase variables, 622  
 Phase voltage, 32  
 Phase-lag design, 648  
 Phase-lead design, 647  
 PI controller, 646, 650  
 PID controller, 564, 650  
 PID design, 649  
 Plant factor, 9  
 Plant output, 622  
 Polar plot, 658  
 Polynomial curve fitting, 603  
 Polynomial evaluation, 604  
 Polynomial roots, 601  
 Pools, 3  
 Positive phase sequence, 30, 400  
 Positive-sequence subtransient reactance, 411  
 Potential difference between two points, 121  
 Potential difference, multiconductors, 123  
 Power angle, 54, 461  
 Power angle characteristics, 57  
 Power circle diagram, 163  
 Power factor, 16  
 Power factor control, 56  
 Power factor correction, 23

- Power flow analysis, 189  
 Power flow data preparation, 223  
 Power flow equation, 189, 209  
 Power flow programs, 222  
 Power flow solution, 208  
 Power flow through transmission lines, 161  
 Power flow, decoupled, 240  
 Power flow, Gauss-Seidel, 209  
 Power flow, Newton-Raphson, 232  
 Power grid, 2  
 Power in single-phase ac circuit, 15  
 Power pool, 2  
 Power residuals, 234  
 Power system control, 527  
 Power system Toolbox, 665  
 Power transformers, 64  
 Power transmission capability, 163  
 Power triangle, 20  
 Power-angle curve, 466  
 Primary feeder, 8  
 Prime mover model, 531  
 Prime movers, 4  
 Product of polynomials, 603  
 Production cost, 268  
 Propagation constant, 153  
 Proportional controller, 646  
  
 Quadratic performance index, 576  
 Quadrature axis, 318  
 Quadrature axis reactance, 64  
 Quadrature axis reluctance, 63  
  
 Rate feedback, 562  
 Reactance of armature reaction, 54  
 Reactive power, 17  
 Reactive power and voltage control, 555  
 Reactive power flow, 59  
 Reactive transmission line loss, 163  
 Real power, 16  
 Real transmission line loss, 163  
 Reference bus, 208  
 Reflected wave, 156  
 Regulated bus, 208  
 Regulated bus data, 224  
 Regulating transformers, 86  
  
 Relative stability, 640, 658  
 Reluctance power, 64  
 Reset action, 542  
 Resonant frequency, 661  
 Resonant peak magnitude, 661  
 Riccati equation, 577  
 Rise time, 644  
 Root locus, 641  
 Root-locus design, 645  
 Rotor, 49  
 Round rotor, 49  
 Routh-Hurwitz array, 641  
 Row vector, 593  
 Running MATLAB, 587  
  
 Salient-pole rotor, 49  
 Salient-pole synchronous generator, 62  
 SCTM, 402  
 Self-inductance, 111  
 Semicolon, 590  
 Sensor model, 557  
 Sequence impedance of lines, 409  
 Sequence impedance of loads, 407  
 Sequence impedance of machines, 410  
 Sequence impedances, 406  
 Sequence impedances of transformer, 411  
 Sequence network of a generator, 418  
 Sequence networks, 420  
 Series capacitor compensation, 168  
 Settling time, 644  
 Short circuit current in lines, 366  
 Short circuit subtransient time constant, 336  
 Short circuit transient reactance, 337  
 Short circuit transient time constant, 337  
 Short length line, 143  
 Short line model, 143  
 Short-circuit test, 69  
 Shunt capacitor compensation, 168  
 Shunt reactors, 165  
 Simplified Nyquist criterion, 660  
 Simulation diagram, 623  
 SIMULINK, 623  
 Single line-to-ground fault, 421, 432  
 Slack bus, 208



- Small disturbances, 471
- Solid fault, 354
- Solution of differential equations, 615
- Sources of electricity, 5
- Space phasor, 51
- Sparse matrix, 193
- Speed governing system, 532
- Speed regulation, 533
- Stabilizer, 562
- Standard transmission voltages, 6
- Stanley, William, 1
- State equation, 622
- State feedback, 569
- State feedback control, 569
- State variables, 622
- Static stability limits, 58
- Stator, 49
- Steady-state error, 642
- Steady-state period, 315
- Steady-state stability limit, 466
- Steel towers, 103
- Subplot, 612
- Substation, 6
- Subtransient period, 315
- Subtransient reactance, 344
- Subtransient time constant, 344
- Subtransmission, 6
- Surge impedance, 157
- Surge impedance loading (*SIL*), 159
- Swing bus, 208
- Swing equation, 464
- Switchgear, 11
- Symmetrical components, 399
- Synchronizing coefficient, 472
- Synchronizing power coefficient, 477
- Synchronous condenser, 165
- Synchronous generator phasor diagram, 55
- Synchronous generators, 49
- Synchronous machine transient analysis, 314
- Synchronous machine transients, 318
- Synchronous reactance, 54
- Synchronous speed, 51
- Synchronously rotating reference frame, 318
- Tap changing transformers, 83, 220
- Taylor's series, 200
- Temperature constant, 105
- Tesla, Nikola, 1
- Thévenin's impedance, 356
- Thévenin's voltage, 356
- Thermal loading limit, 163
- Three-dimensional plots, 613
- Three-phase transformer connections, 74
- Three-phase transformer model, 76
- Three-winding transformer model, 82
- Three-winding transformers, 81
- Tie-line bias control, 549
- Time-domain performance specifications, 644
- Torque angle, 461
- Transfer admittance, 192
- Transfer function, 529, 638
- Transformer bus data, 224
- Transformer efficiency, 70
- Transformer equivalent circuit, 66
- Transformer leakage flux, 66
- Transformer magnetizing current, 66
- Transformer maximum efficiency, 71
- Transformer mutual flux, 66
- Transformer no-load current, 66
- Transformer performance, 70
- Transformer voltage regulation, 71
- Transformer zero-sequence impedance, 412
- Transient period, 315
- Transient phenomena, 315
- Transient reactance, 343
- Transient stability, 460
- Transient time constant, 343
- Transmission and distribution, 6
- Transmission line parameters, 102
- Transmission matrix, 149
- Transpose, 593
- Transposed line, 114
- Tree of network, 369
- Turbine model, 531



Ultra-high voltage, 104  
 Unbalanced faults, 399  
 Unbalanced short circuit, 330  
 Unconstrained parameter optimization,  
     258  
 Utility matrices, 599  
 Utilization factor, 9  
  
 V curve, 57  
 Var, 17  
 Variables, 589  
 Vector operation, 593  
 Velocity of propagation, 157  
 Voltage control of transformers, 83  
 Voltage regulation, 144  
 Voltage waves, 156  
 Voltage-controlled bus, 208  
  
 Wave length, 157  
  
 Y-connected loads, 32  
  
 Zero phase sequence, 401  
 Zero vector, 595  
 Zero-sequence reactance of lines, 410  
 Zero-sequence subtransient reactance,  
     411  
 Zero-sequence variable, 325