The admissible states and control values are unconstrained.
(b) Define C as the identity matrix of rank 4, i.e., $C = \text{eye}(4)$ and $D = \text{zeros}(4, 1)$. Use the *MATLAB* function **[K, P] = lqr2 (A, B, Q, R)** to design a state feedback controller in response to an initial condition offset of $\theta(0) = 0.1$ rad and $x(0) = 0.1$m (i.e., $x_0 = [0.1 \quad 0 \quad 0.1 \quad 0]$). Use the *MATLAB* function $[\mathbf{y}, \mathbf{x}] = \text{initial}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{X_0}, \mathbf{t})$ to simulate the system for 20 seconds. Obtain a plot of θ, x, and the control law $u = -kx'$.

12.15. Construct the *SIMULINK* block diagram for the linearized model of the inverted pendulum described in Problem 12.12(a) using the *SIMULINK* LQR model. Obtain the response for θ, x, and u for the initial condition offset described in Problem 12.14.

12.16. Obtain the state variable representation of the LFC system of Problem 12.4 with one input ΔP_L, and perform the following analysis.
(a) Use the *MATLAB* **step** function to obtain the frequency deviation step response for a sudden load change of $\Delta P_L = 0.2$ per unit.
(b) Construct the *SIMULINK* block diagram and obtain the frequency deviation response for the condition in part (a).
(c) Use **placepol(A, B, C, p)** function to place the compensated closed-loop pole at $-4 \pm j6$ and -4. Obtain the frequency deviation step response of the compensated system.
(d) Construct the *SIMULINK* block diagram and obtain the frequency deviation response for the condition in part (c).

12.17. Design a LQR state feedback for the system described in Problem 12.16.
(a) Find the optimal feedback gain vector to minimize the performance index

$$\mathbf{J} = \int_0^\infty \left(40{x_1}^2 + 20{x_2}^2 + 10{x_3}^2 + 0.2u^2\right) dt$$

The admissible states and control values are unconstrained. Obtain the frequency deviation step response for a sudden load change of $\Delta P_L = 0.2$ per unit.
(b) Construct the *SIMULINK* block diagram and obtain the frequency deviation step response for the condition in part (a).

APPENDIX A

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*.

A.1 INSTALLING THE TEXT TOOLBOX

The CD-Rom included with the book contains all the developed functions and chapter examples. The file names for chapter examples begin with the letters **chp**. For example, the M-file for Example 11.4 is **chp11ex4**. The appendix examples begin with **exa** and **exb** and a number. The CD contains a file called **PowerToolbox_V4.exe** that automates installation of all the files. Insert the CD-Rom in the disk drive and use Windows Explorer to view its contents. Double click on the **PowerToolboxV_4.exe** file to start the installation. The installation program will prompt you for the location of the *MATLAB* directory and the name of the directory where you would like the files to be installed.

To run m-files, load *MATLAB*. In the **Command Window** from **File menu**, select **Set Path** to open the Set Path Window. Press the **Add Folder** button to open the Browse for Folder. Select the folder where you have installed the Toolbox. Press **Move to Bottom** option, and press **Save** button to save the new path permanently. If you get the message "*MATLAB* cannot save changes to the path," it will ask you "Would you like to save pathdef.m to another location?" Select "Yes", the pathdef.m will be saved in **MATLAB** startup directory and will be used in the future sessions.

A.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 6.1 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**.

A.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.

A.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
format	Default. Same as **format short**
format short	Scaled fixed point format with 5 digits
format long	Scaled fixed point format with 15 digits
format short e	Floating point format with 5 digits
format long e	Floating point format with 15 digits
format short g	Best of fixed or floating point with 5 digits
format long g	Best of fixed or floating point with 15 digits
format hex	Hexadecimal format
format +	The symbols +, - and blank are printed for positive, negative, and zero elements
format bank	Fixed format for dollars and cents
format rat	Approximation by ratio of small integers
format compact	Suppress extra line feeds
format loose	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,...` 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

```
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 **cntrl-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**.

A.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
```

A.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


```
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

```
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

```
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.

A.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. $A\backslash B$ is equivalent to $A^{-1}B$, and A/B is equivalent to AB^{-1}

Example A.1 (exa1)

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 A.2 (exa2)

Use the **inv** function to determine the inverse of matrix A in Example A.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 A.3 (exa3)

Use the **lu** factorization function to express the matrix A of Example A.2 as the product of upper and lower triangular matrices, $A = LU$. Then find X from $X = U^{-1}L^{-1}B$. Typing

```
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**.

A.7.1 UTILITY MATRICES

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

eye(m, n)	Generates an m-by-n identity matrix.
zeros(m, n)	Generates an m-by-n matrix of zeros.
ones(m, n)	Generates an m-by-n matrix of ones.
diag(x)	Produces a diagonal matrix with the elements of **x** 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**.

A.7.2 EIGENVALUES

If A is an n-by-n matrix, the n numbers λ 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 A.4 (exa4)

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 =                                    D =
   -0.7071    0.2182   -0.0921            -1     0     0
    0.0000    0.4364   -0.5523             0    -2     0
   -0.7071    0.8729   -0.8285             0     0    -3
```

A.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 A.5 (exa5)

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 A.6 (exa6)

In the circuit shown in Figure A.1, determine the node voltages V_1 and V_2 and the power delivered by each source.

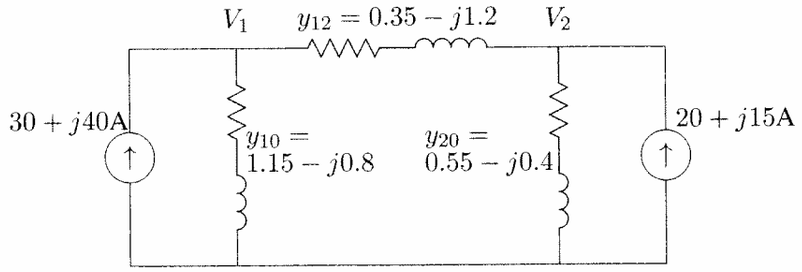


FIGURE A.1
Circuit for Example A.6.

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

$$\begin{bmatrix} 1.5 - j2.0 & -.35 + j1.2 \\ -.35 + j1.2 & 0.9 - j1.6 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 30 + j40 \\ 20 + j15 \end{bmatrix}$$

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 from
real(z)	Returns real part of z
imag(z)	Returns imaginary part of z
abs(z)	Absolute value of z
angle(z)	Phase angle of z
conj(z)	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.

A.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 A.7 (exa7)

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 A.8 (exa8)

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

```
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 A.9 (exa9)

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)
```


A.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 A.10 (exa10)

(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
```

A.9.2 POLYNOMIAL CURVE FITTING

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

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

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 dth order polynomial in descending powers of x.

Example A.11 (exa11)

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

A.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
```

A.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

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

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 A.12 (exa12)

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**.

A.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 `(x1, y1)` with a solid red line, `(x2, y2)` with a blue + mark, and `(x3, y3)` 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 `(x2, y2)` with a blue + mark, and `(x2, y2)` 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**.

A.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 A.13 (exa13)

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 A.2.

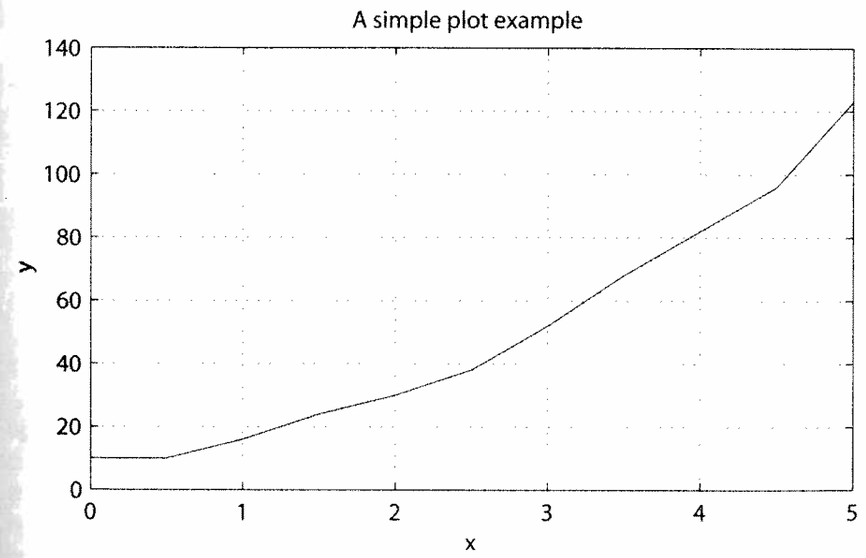


FIGURE A.2
Example of X-Y plot.

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 A.14 (exa14)

Fit a polynomial of order 2 to the data in Example A.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)
```

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

```
p =
    4.0232      2.0107      9.6783
```

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 A.3.

Example A.15 (exa15)

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)])
```

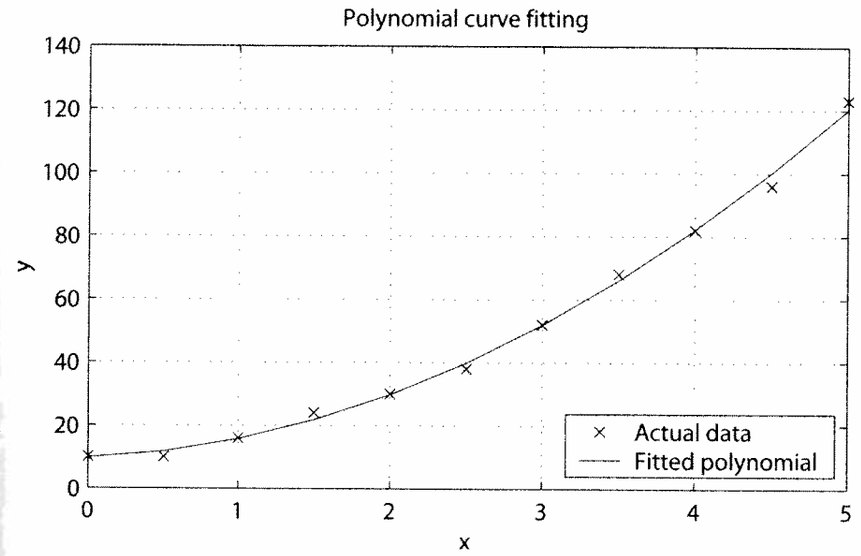



FIGURE A.3
Fitting a parabola to the data in Example A.13.

The result is

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

and the plot is shown in Figure A.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 A.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

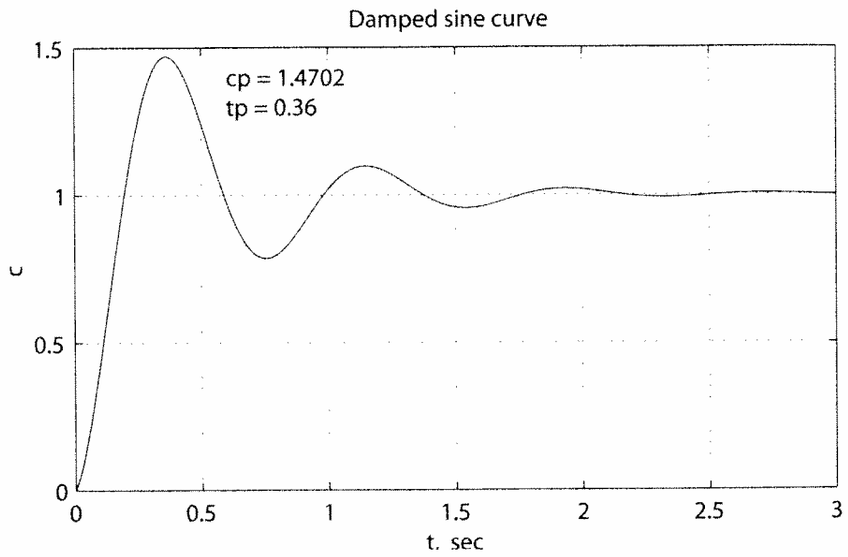


FIGURE A.4
Graph of Example A.15.

```
cp =
    1.47
tp =
    0.36
```

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 pth 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 A.16 (exa16)

Divide the Figure window into four partitions, and plot the following functions for ωt from 0 to 3π in steps of 0.05.

1. Plot $v = 120\sin\omega t$ and $i = 100\sin(\omega t - \pi/4)$ versus ωt 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 ωt on the lower left portion.
4. For $f_R = 3F_m$, construct a circle of radius f_R on the lower right portion.

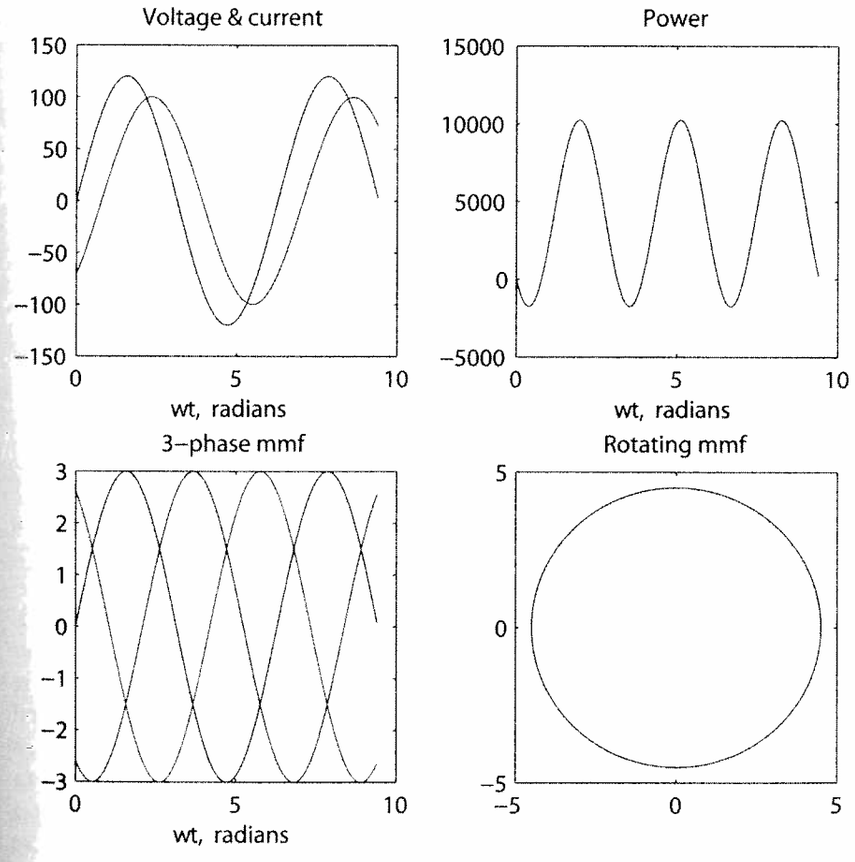


FIGURE A.5
Subplot demonstration.

```
wt = 0: 0.05: 3*pi; v=120*sin(wt);          %Sinusoidal voltage
i = 100*sin(wt - pi/4);                     %Sinusoidal current
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 A.16 results are shown in Figure A.5.

A.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 A.17 (exa17)

Few examples of 3-D plots and mesh plots are given in Figure A.6.
For more specialized 3-D graphs and color related functions see **help specgraph**.

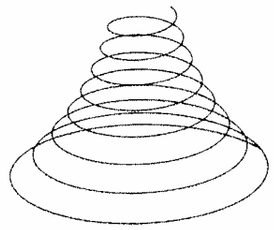
A.13 HANDLE GRAPHICS

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.

Plot of parametric space curve

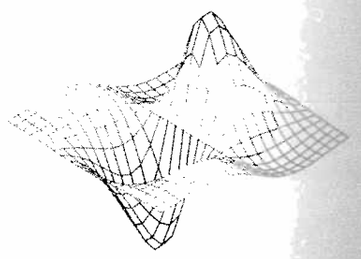
$x(t) = e^{-0.03t}\cos t,\ y(t) = e^{-0.03t}\sin t,\ z(t) = t$

```
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
```



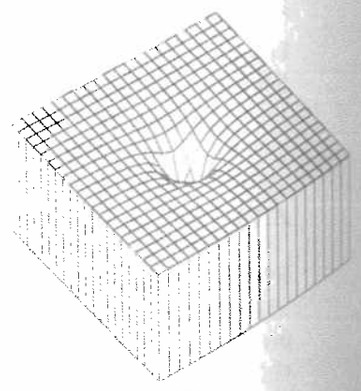
Plot of function $z = \sin x \cos y\, e^{-(x^2+y^2)^{0.5}}$
using mesh

```
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
```



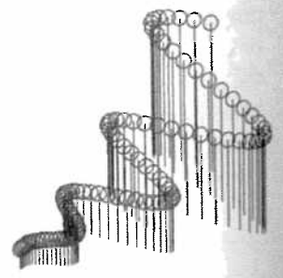
Plot of function $z = -0.1/(x^2+y^2+1)$
using meshz

```
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)
```



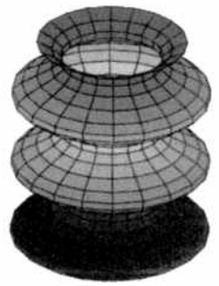
Discrete plot of $x = t,\ y = t\cos t,\ z = e^{0.1t}$
using stem3

```
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 + sint
using cylinder function

```
t = 0 : pi/5 : 6*pi;
p = 3 + sin(t);
cylinder(p), axis off
```



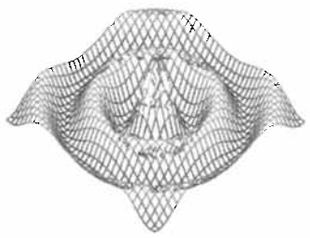
Plot of a unit sphere and a scaled sphere
using cylinder function

```
[x,y,z] = sphere(24);
surf(x–2, y–2, z–1);
hold on
surf(2*x, 2*y, 2*z); axis off
```

Cartezian plot of Bessel function

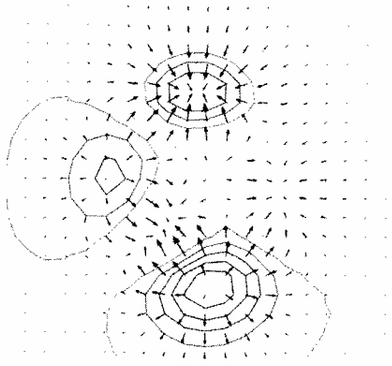
$J_0(\mathrm{sqrt}(x^2+y^2))$

```
[x,y]=meshgrid(–12:0.7:12, –12:0.7:12);
r = sqrt(x.^2+y.^2); z = bessel(0, r);
m = [–45 60];
mesh(z,m), axis off
```



Counter 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);
```



A.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.

Relational Operator		*Logical Operator*	
==	equal	&	logical AND
~=	not equal	|	logical OR
<	less than	~	logical complement
<=	less than or equal to	xor	exclusive OR
>	greater than		
>=	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 A.18 (exa18)

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).

$$x(t) = \frac{4}{\pi}[\sin\omega_0 t + \frac{1}{3}\sin 3\omega_0 t + \frac{1}{5}\sin 5\omega_0 t + \cdots]$$
$$= \frac{4}{\pi}\sum_{n=1}^{\infty}\frac{1}{n}\sin n\omega_0 t \qquad n \text{ is an odd integer}$$

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 A.7.

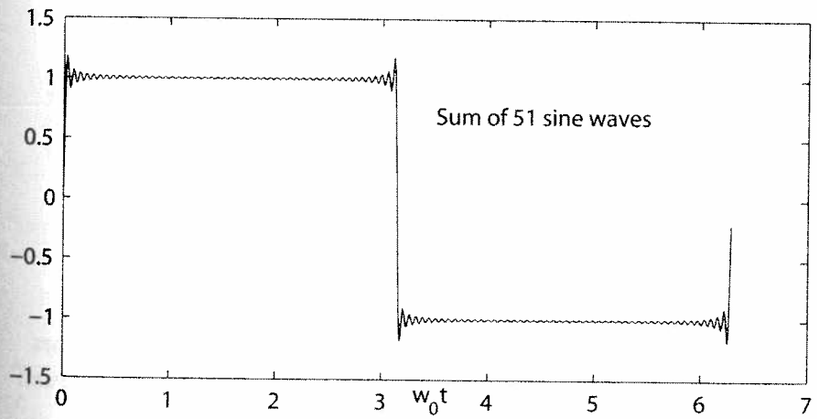


FIGURE A.7
Graph 0f Example A.18.

Example A.19 (axa19)

A network function known as transfer function is expressed by

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

(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 A.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.

Example A.20 (exa20)

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.

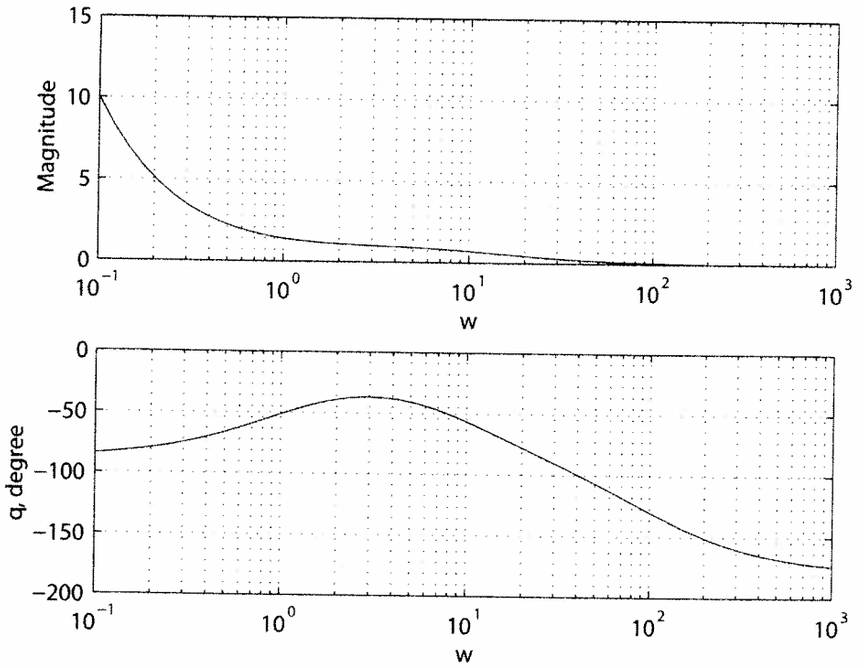


FIGURE A.8
Magnitude and phase angle plots for Example A.19.

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 Δt. 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

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

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.

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

we use the following statements

```
a1 = 4; a0 = 2;                                    % Initial state
x0 = 1; t0 = 0;                                % Step size and final time
Dt = 0.01;  tf=15;                           % Initializing all the arrays
t=[]; x=[];
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 A.9.

Example A.21 (exa21)

Consider the system of n equations in n variables

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

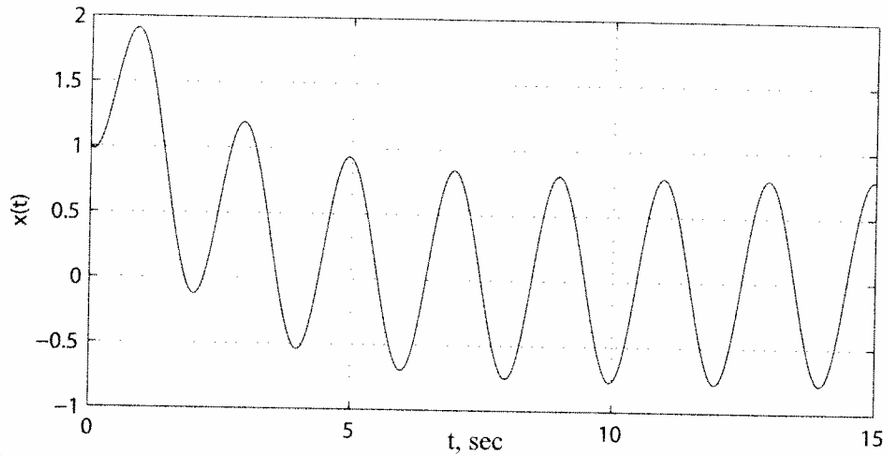


FIGURE A.9
Numerical solution of Example A.20.

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

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

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 $(x_1^{(0)}, x_2^{(0)} \cdots, x_n^{(0)})$. 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.

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

$$x_1 - x_1x_3 + x_2x_3 = 6$$

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 convergencc
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                                       %Change in variable
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 $x_3 = 4.0000$.

A.15 SOLUTION OF DIFFERENTIAL EQUATIONS

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 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. The nth-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 A.22 (exa22)

Consider the simple mechanical system of Figure A.10. Three forces influence the motion of the mass, namely, the applied force, the frictional force, and the spring force.

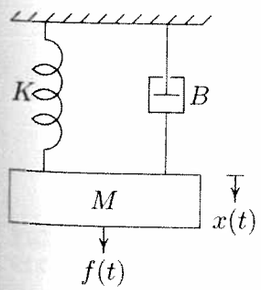


FIGURE A.10
Mechanical translational system.

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

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

Let

$$x_1 = x$$

and

$$x_2 = \frac{dx}{dt}$$

then

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

With the system initially at rest, a force of 25 newtons 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.

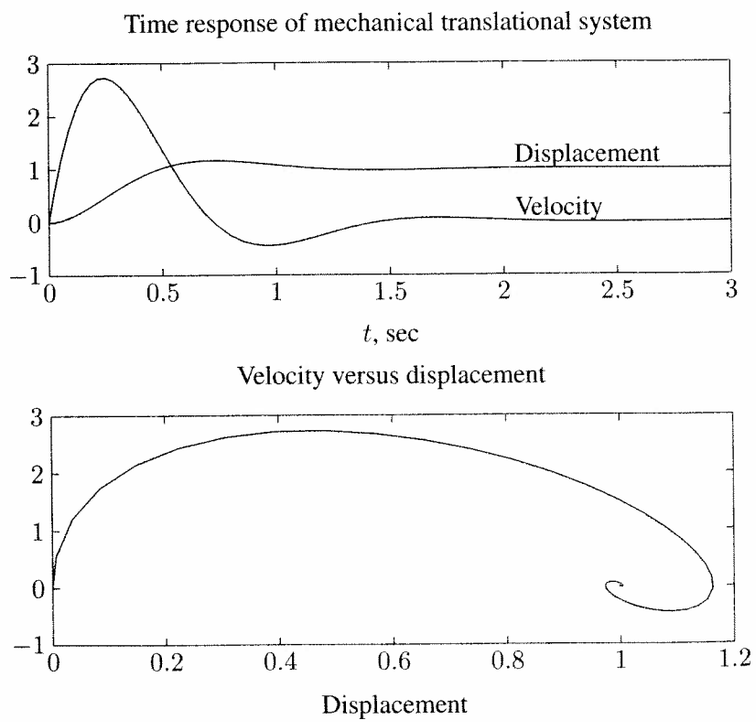


FIGURE A.11
Response of the mechanical system of Example A.22.

```
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, **exa18.m**, uses **ode23** to simulate the system over an interval of 0 to 3 sec., with zero initial conditions.

```
tspan = [0, 3];                                          % time interval
x0 = [0, 0];                                          % initial conditions
[t,x] = ode23('mechsys', tspan, x0); subplot(2, 1, 1), plot(t, x),
xlabel('t,  sec') title('Time response of mechanical translational
system') text(2, 1.2, 'Displacement'), text(2, 0.2, '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 A.11.

Example A.23 (exa23)

The circuit elements in Figure A.12 are $R = 1.4\ \Omega$, $L = 2$ H, and $C = 0.32$ F. The initial inductor current is zero, and the initial capacitor voltage is 0.5 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.

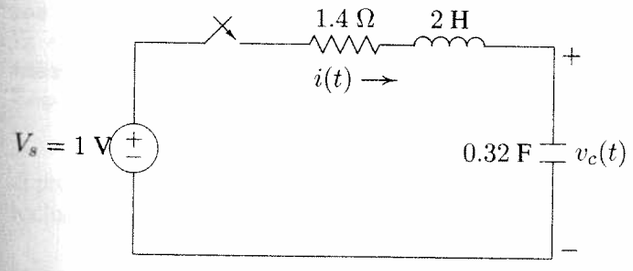


FIGURE A.12
RLC circuit for Example A.23.

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)
)];
```

The following M-file, **exa19.m**, uses **ode23** to simulate the system over an interval of 0 to 15 sec.

```
tspan = [0, 15];                                    % time interval
x0 = [0.5, 0];                                  % initial conditions
[t,x] = ode23('electsys', tspan, x0); subplot(2, 1, 1), plot(t, x)
title('Time response of an RLC series circuit') xlabel('t,  sec')
text(8,1.05,'Capacitor voltage'), text(8, .05,'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 A.13.

A.16 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 A.20.

Example A.24 (exa24)

Consider the simple pendulum illustrated in Figure A.14, where a weight of $W = mg$ kg is hung from a support by a weightless rod of length L meters. While usually

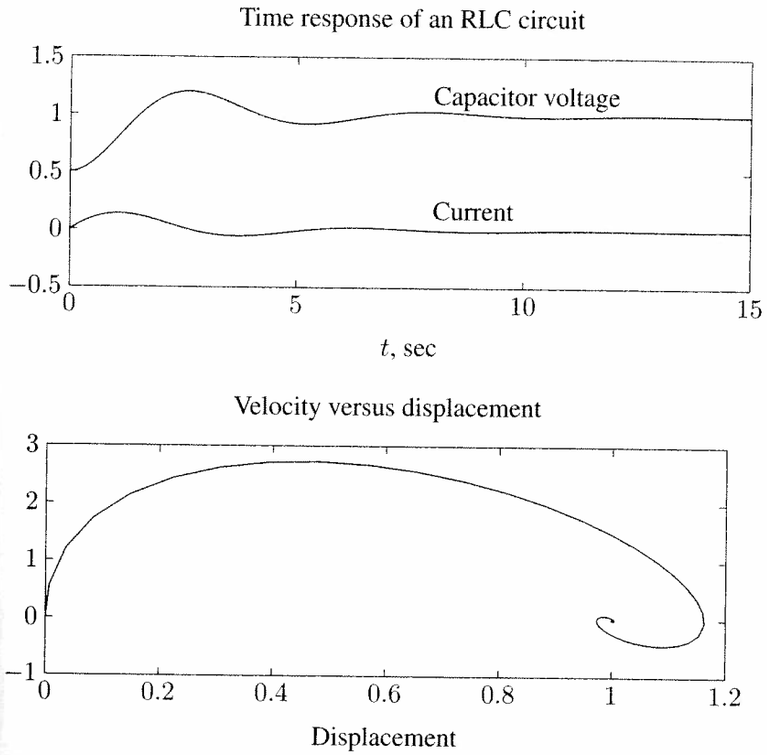


FIGURE A.13
Response of the series RLC circuit of Example A.23.

approximated by a linear differential equation, the system really is nonlinear and includes viscous damping with a damping coefficient of B kg/m/sec.

If θ 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 θ can be written as

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

From Newton's law

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

Combining the two equations for the force, we get

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

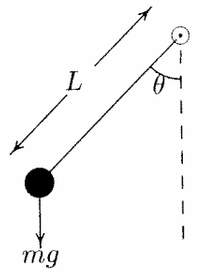


FIGURE A.14
Pendulum oscillator.

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.

```
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)) ];
```

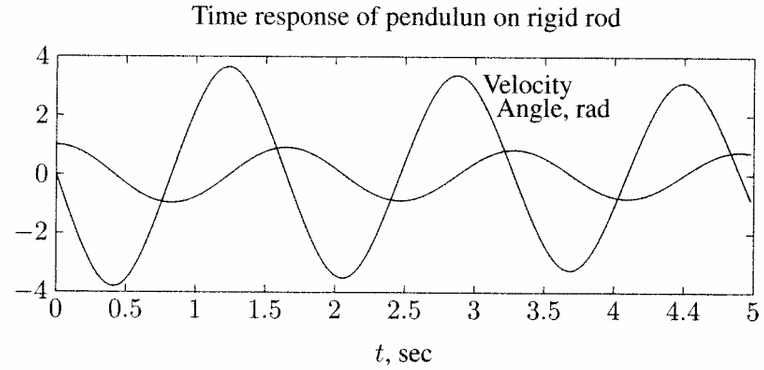
The following M-file, **exa20.m**, uses **ode23** to simulate the system over an interval of 0 to 5 sec. Results of the simulation are shown in Figure A.15.

```
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 a rigid pendulum') xlabel('t, sec')
text(3.2, 3.5, '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') subplot(111)
```

A.17 SIMULATION DIAGRAM

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

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \qquad \text{(A.1)}$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}u(t)$$



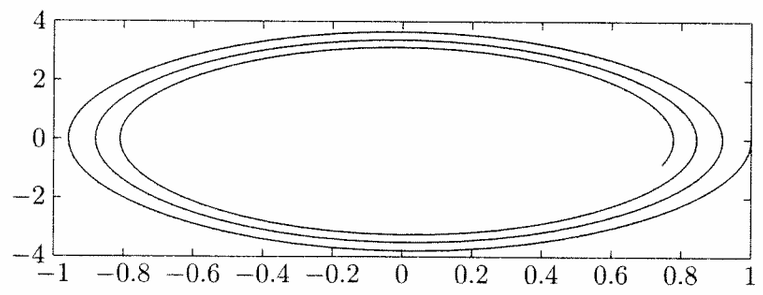


FIGURE A.15
Response of the pendulum described in Example A.24.

This system of first-order differential equations is known as the state equation of the system, and **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 nth-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 nth-order linear plant model described by the differential equation

$$\frac{d^n y}{dt^n} + a_{n-1}\frac{d^{n-1}y}{dt^{n-1}} + \ldots + a_1\frac{dy}{dt} + a_0 y = u(t) \qquad \text{(A.2)}$$

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

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

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

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}_{n-1} &= x_n \\ \dot{x}_n &= -a_0 x_1 - a_1 x_2 - \ldots - a_{n-1}\, x_n + u(t) \end{aligned} \tag{A.3}$$

or in matrix form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ -a_0 & -a_1 & -a_2 & \ldots & -a_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u(t) \tag{A.4}$$

and the output equation is

$$y = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \end{bmatrix} \mathbf{x} \tag{A.5}$$

The M-file **ode2phv.m** is developed which converts an nth-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.

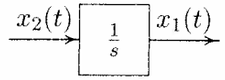
Equation (A.3) indicates that state variables are determined by integrating the corresponding state equation. A diagram known as the simulation diagram can be constructed to model the given differential equations. The basic element of the simulation diagram is the integrator. The first equation in (A.3) 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 (A.3) via a summing point and feedback paths, a simulation diagram is obtained.



A.18 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.

A.18.1 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 that 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.

A.18.2 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.

A.18.3 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 A.25 (simexa25.mdl)

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.

$$T_F = \frac{9}{5}T_C + 32 \tag{A.6}$$

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 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 A.16.

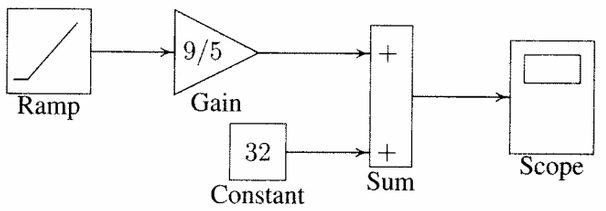


FIGURE A.16
Simulink diagram for the system of Example A.25.

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 **simexa25** Start the simulation. Double click on the Scope, click on the **Auto Scale**, the result is displayed as shown in Figure A.17.

Example A.26 (simexa26.mdl)

Construct a simulation diagram for the state equation described by

$$\frac{dx_1}{dt} = x_2$$
$$\frac{dx_2}{dt} = \frac{1}{M}[f(t) - Bx_2 - kx_1]$$

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 A.18.

To create a *SIMULINK* block diagram presentation select **new...** from the **File** menu. This provides an untitled blank window for designing and simulating a

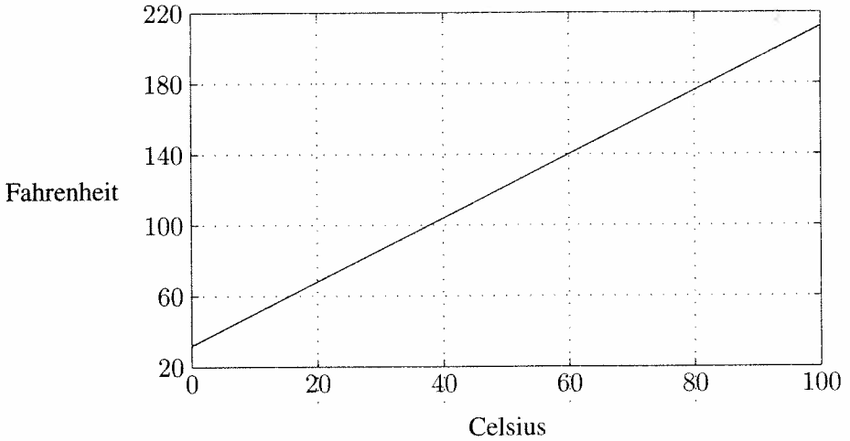


FIGURE A.17
Fahrenheit-Celsius temperature graph for Example A.25.

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 A.18. 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

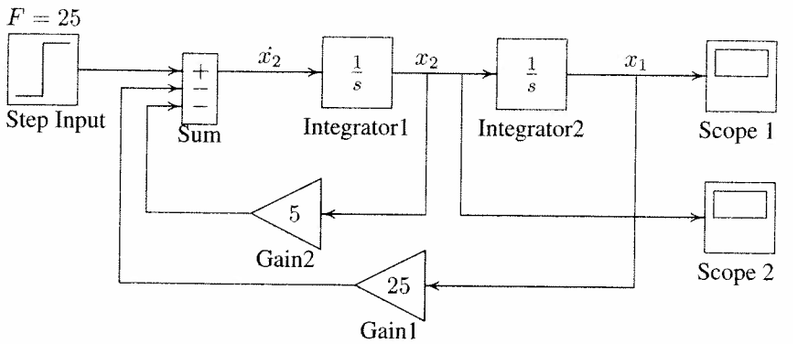


FIGURE A.18
Simulink diagram for the system of Example A.26.

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 A.18. Pull down the **File** menu and use **Save as** to save the model under a file name **simexa26**. 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 A.19 and A.20.

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.

Example A.27 (simexa27.mdl)

Consider the system defined by

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

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

$$x_1 = y$$
$$x_2 = \dot{y}$$
$$x_3 = \ddot{y}$$

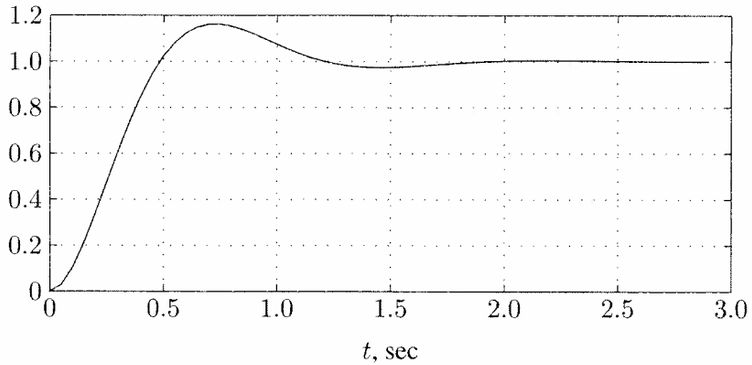


FIGURE A.19
Displacement response of the system described in Example A.26.

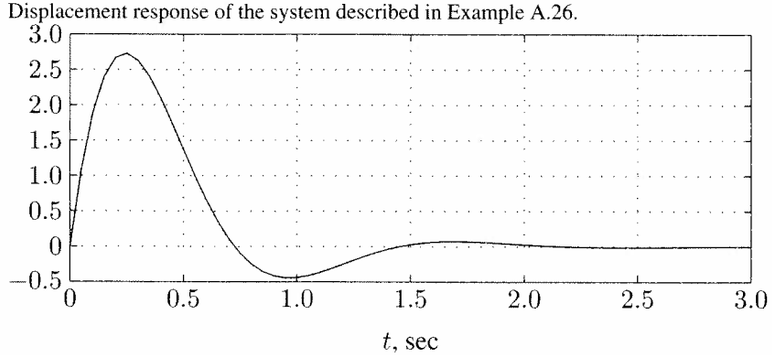


FIGURE A.20
Velocity response of the system described in Example A.26.

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 $\dddot{y}$ 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

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

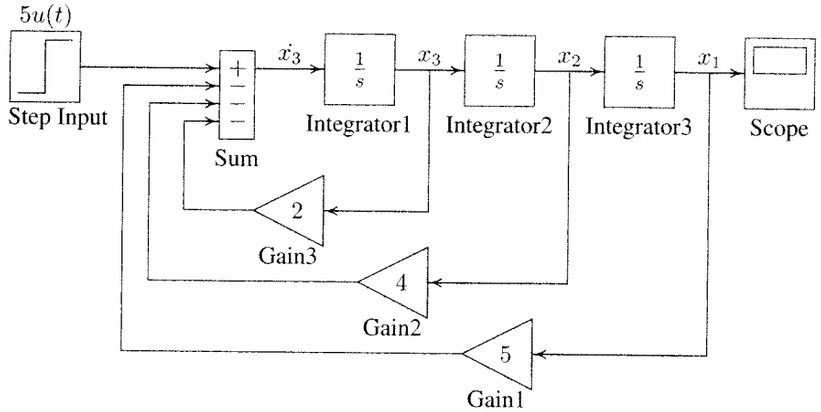


FIGURE A.21
Simulation diagram for the system of Example A.27.

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 A.21. A *SIMULINK* Block diagram is constructed and saved as **simexa27**. The simulation response is shown in Figure A.22.

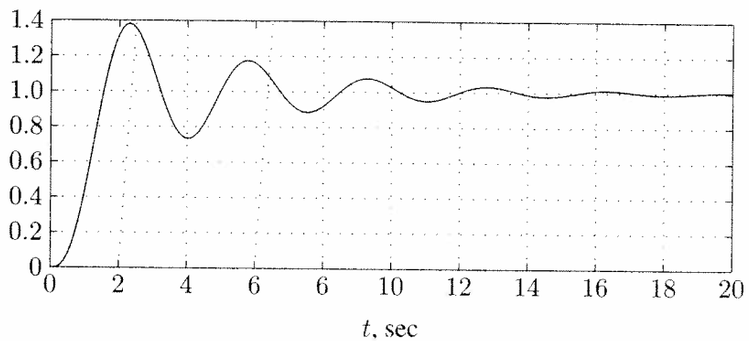


FIGURE A.22
Simulation result for the system in Example A.27.

Example A.28 (simexa28.mdl)

Use the **state-space** model to simulate the state and output equations described in Example A.27.

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 A.23, and saved as **simexa28**.

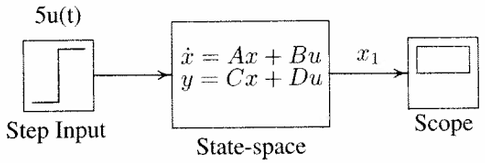


FIGURE A.23
State-space model for system in Example A.28.

Note that in this example, the output is given by $y = x_1$, and we define C as $C = [1 \quad 0 \quad 0]$. 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.

A.18.4 USING THE TO WORKSPACE BLOCK

The To Workspace block can be used to return output trajectories to the *MATLAB* Workspace. Example A.29 illustrates this use.

Example A.29 (simexa29.mdl)

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 **simexa29** as shown in Figure A.24.

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.

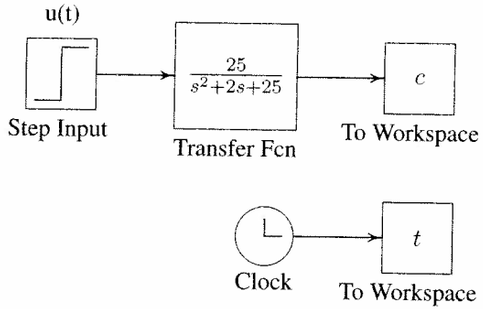


FIGURE A.24
Simulink model for system in Example A.29.

A.18.5 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

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{A.7}$$
$$y(t) = Cx(t) + Du(t) \tag{A.8}$$

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 A.30 (simexa30.mdl)

Obtain the state-space model for the system represented by the block diagram shown in Figure A.25. The model is saved with a filename **simexa30**. 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('simexa30')
```

The result is

```
A =                          B =
     0     0     0    20          0
    -1    -1     0     0          1
```

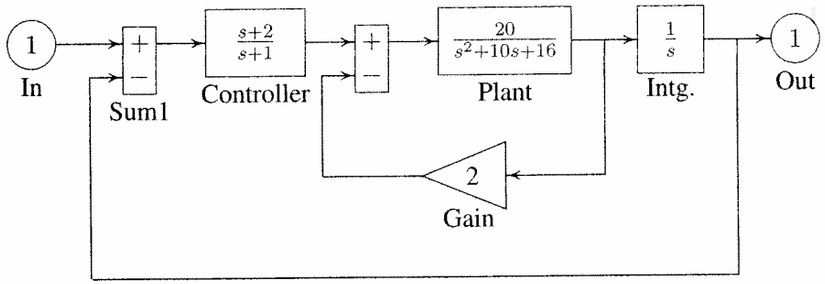


FIGURE A.25
Simulink model for system in Example A.30.

```
    -1     1   -10   -56          1
     0     0     1     0          0
C =

     1     0     0     0
D =
     0
```

In order to obtains 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)
```

A.18.6 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 A.31 (simexa31.mdl)

To encapsulate a portion of an existing *SIMULINK* model into a subsystem, consider the *SIMULINK* model of Example A.27 shown in Figure A.26, and proceed as follows:

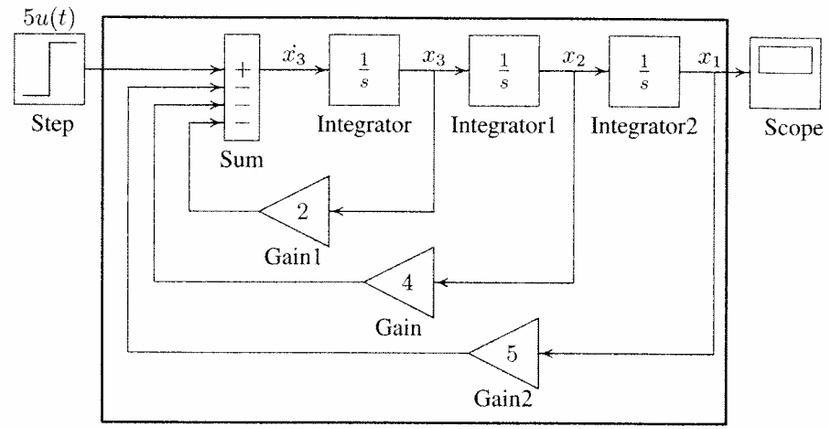


FIGURE A.26
Simulation diagram for the system of Example A.27.

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.

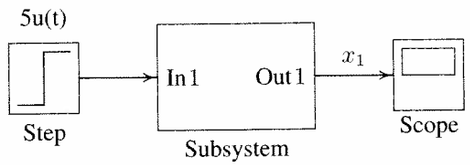


FIGURE A.27
Simulation diagram for the system of Example A.27.

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.

- 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 **simexa32**. Additional *SIMULINK* examples are found in Chapter 12. Also, many interesting examples are available in *SIMULINK* **demo**.

APPENDIX B

REVIEW OF FEEDBACK CONTROL SYSTEMS

B.1 THE CONTROL PROBLEM

The first step in the analysis and design of control systems is mathematical modeling of the system. The two most common methods are the transfer function approach and the state equation approach. The state equations can be applied to portray linear as well as nonlinear systems

All physical systems are nonlinear to some extent. In order to use the transfer function and linear state equations, the system must first be linearized. Thus, proper assumptions and approximations are made so that the system can be characterized by a linear mathematical model. The model may be validated by analyzing its performance for realistic input conditions and then by comparing with field test data taken from the dynamic system in its operating environment. Further analysis of the simulated model is usually necessary to obtain the model response for different feedback configurations and parameters settings. Once an acceptable controller has been designed and tested on the model, the feedback control strategy is then applied to the actual system to be controlled.

When we wish to develop a feedback control system for a specific purpose, the general procedure may be summarized as follows:

1. Choose a way to adjust the variable to be controlled; e.g., the mechanical load will be positioned with an electric motor or the temperature will be controlled by an electrical resistance heater.

2. Select suitable sensors, power supplies, amplifiers, etc., to complete the loop.

3. Determine what is required for the system to operate with the specified accuracy in steady-state and for the desired response time.

4. Analyze the resulting system to determine its stability.

5. Modify the system to provide stability and other desired operating conditions by redesigning the amplifier/controller, or by introducing additional control loops.

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

B.2 STABILITY

Consider the block diagram of a simple closed-loop control system as shown in Figure B.1 where $R(s)$ is the s-domain reference input, and $C(s)$ is the s-domain controlled output. $G(s)$ is the plant transfer function, K is a simple gain controller,

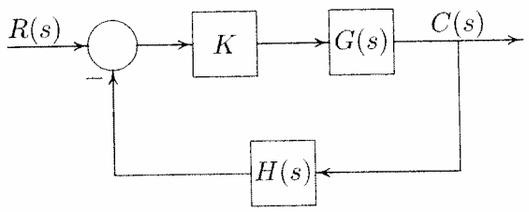


FIGURE B.1
A simple closed-loop control system.

and the feedback elements $H(s)$ represent the sensor transfer function. The closed-loop transfer function is

$$\frac{C(s)}{R(s)} = T(s) = \frac{KG(s)}{1 + KG(s)H(s)} \tag{B.1}$$

or the s-domain response is

$$C(s) = T(s)R(s) \tag{B.2}$$

The gain $KG(s)H(s)$ is commonly referred to as the *open-loop transfer function*. 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 denominator polynomial of the closed-loop transfer function set equal to zero is the system characteristic equation. That is, the characteristic equation is given by

$$1 + KG(s)H(s) = 0 \tag{B.3}$$

The roots of the characteristic equation are known as the poles of the closed-loop transfer function. 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 **impulse** 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 nonlinear 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*. 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*.

B.2.1 THE ROUTH-HURWITZ STABILITY CRITERION

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

$$a_n s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0 = 0 \tag{B.4}$$

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

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

$a_n, a_{n-1}, \ldots, a_0$ 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 (B.4) lie in the left half of the s-plane is that the elements of the first column of the Routh-Hurwitz 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-Hurwitz array and determines if any roots have positive real parts. **a** is a row vector containing the coefficients of the characteristic equation.

If the first element in a row is zero, it is replaced by a very small positive number ϵ, and the calculation of the array is completed. 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.

B.2.2 ROOT-LOCUS METHOD

The *root-locus method*, developed by W. R. Evans, 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.

Consider the feedback control system given in Figure B.1. In general, the open-loop transfer function is given by

$$KG(s)H(s) = \frac{K(s+z_1)(s+z_2)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)} \tag{B.5}$$

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 given by (B.3); therefore

$$\frac{(s+p_1)(s+p_2)\cdots(s+p_n)}{(s+z_1)(s+z_2)\cdots(s+z_m)} = -K \tag{B.6}$$

From (B.6) 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.

$$K = \frac{\text{product of vector lengths from finite poles}}{\text{product of vector lengths from finite zeros}} \tag{B.7}$$

and

$$\sum \text{angles of zeros of } GH(s) - \sum \text{angles of poles of } GH(s) = r(180)^\circ \quad \text{where} \quad r = \pm 1, \pm 3, \pm 5, \cdots \tag{B.8}$$

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, ' . ')** to obtain the plot. **rlocus** function is accurate, and we use it to obtain the root-locus. 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. For the basic construction rules for sketching the root locus, refer to any text on feedback control systems.

B.3 STEADY-STATE ERROR

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 has decayed, 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 B.1. The closed-loop transfer function is given by (B.1). The error of the closed-loop system is

$$E(s) = R(s) - H(s)C(s) = \frac{1}{1 + KG(s)H(s)}R(s) \tag{B.9}$$

Using the final-value theorem, we have

$$e_{ss} = \lim_{s \to 0} \frac{sR(s)}{1 + KG(s)H(s)} \tag{B.10}$$

For the polynomial inputs, such as step, ramp, and parabolas, the steady-state error from the above equation will be:

Unit step input

$$e_{ss} = \frac{1}{1 + \lim_{s \to 0} KG(s)H(s)} = \frac{1}{1 + K_p} \tag{B.11}$$

Unit ramp input

$$e_{ss} = \frac{1}{\lim_{s \to 0} sKG(s)H(s)} = \frac{1}{K_v} \tag{B.12}$$

Unit parabolic input

$$e_{ss} = \frac{1}{\lim_{s \to 0} s^2KG(s)H(s)} = \frac{1}{K_a} \tag{B.13}$$

In order to define the *system type*, the general open-loop transfer function is written in the following form.

$$KG(s)H(s) = \frac{K(1 + T_1S)(1 + T_2s)\dots(1 + T_ms)}{s^j(1 + T_as)(1 + T_bs)\dots(1 + T_ns)} \tag{B.14}$$

The *type* of feedback control system refers to the *order* of the pole of $G(s)H(s)$ at $s = 0$.

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.

B.4 STEP RESPONSE

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. The step response for a second-order system is obtained. The standard form of the second-order transfer function is given by

$$G(s) = \frac{{\omega_n}^2}{s^2 + 2\zeta\omega_n s + {\omega_n}^2} \tag{B.15}$$

where ω_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. ζ 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

$$c(t) = 1 - \frac{1}{\beta}e^{-\zeta\omega_n t}\sin(\beta\omega_n t + \theta) \tag{B.16}$$

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

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

$$P.O. = \frac{\text{maximum value} - \text{final value}}{\text{final value}} \tag{B.17}$$

The peak time is obtained by setting the derivative of (B.16) to zero.

$$t_p = \frac{\pi}{\omega_n \sqrt{1-\zeta^2}} \tag{B.18}$$

The peak value of the step response occurs at this time, and evaluating the response in (B.16) at $t = t_p$ yields

$$C(t_p) = M_{pt} = 1 + e^{-\zeta\pi/\sqrt{1-\zeta^2}} \tag{B.19}$$

Therefore, from (B.17), the percent overshoot is

$$P.O. = e^{-\zeta\pi/\sqrt{1-\zeta^2}} \times 100 \tag{B.20}$$

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 ± 2 percent of the final value. For the second-order system, the response remains within 2 percent after 4 time constants, that is

$$t_s = 4\tau = \frac{4}{\zeta\omega_n} \tag{B.21}$$

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.

B.5 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 forward path, as shown in Figure B.2. $G_c(s)$ must be chosen so that the root locus will pass through the proper region of the s-plane.

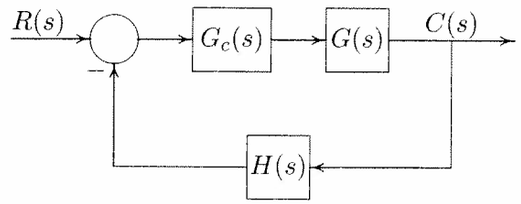


FIGURE B.2
A closed-loop control system with controller.

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).

$$G_c(s) = K_P + \frac{K_I}{s} + K_D s \tag{B.22}$$

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

$$G_c(s) = \frac{K_c(s + z_0)}{s + p_0} \tag{B.23}$$

B.5.1 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.

B.5.2 PHASE-LEAD DESIGN

In (B.23) 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 (B.8) 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 DC gain of the compensator is

$$a_0 = G_c(0) = \frac{K_c z_0}{p_0} \tag{B.24}$$

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 (B.8) to determine p_0. Then, the gain K_c is obtained by applying the magnitude criterion of (B.7). 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

$$s_1 = |s_1| \angle \beta \tag{B.25}$$

z_0 and p_0 are obtained such that the equation

$$1 + G_c(s_1)G_p(s_1) = 0 \tag{B.26}$$

is satisfied. It can be shown that the above parameters are found from the following equations.

$$z_0 = \frac{a_0}{a_1} \qquad p_0 = \frac{1}{b_1} \quad \text{and} \quad K_c = \frac{a_0 p_0}{z_0} \tag{B.27}$$

where

$$a_1 = \frac{\sin\beta + a_0 M \sin(\beta - \psi)}{|s_1|\, M \sin\psi}$$

$$b_1 = -\frac{\sin(\beta + \psi) + a_0 M \sin\beta}{|s_1| \sin\psi} \tag{B.28}$$

where M and ψ are the magnitude and phase angle of the open-loop plant transfer function evaluated at s_1, i.e.,

$$G_p(s_1) = M \angle \psi \tag{B.29}$$

For the case that ψ is either $0°$ or $180°$, (B.28) is given by

$$a_1 |s_1| \cos\beta \pm \frac{b_1 |s_1|}{M} \cos\beta \pm \frac{1}{M} + a_0 = 0 \tag{B.30}$$

where the plus sign applies for $\psi = 0°$ and the minus sign applies for $\psi = 180°$. For this case, the zero of the compensator must also be assigned.

B.5.3 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 **[numo, deno, denc] = 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.

$$a_0 = G_c(0) = \frac{K_c z_0}{p_0} = 1 \tag{B.31}$$

Therefore,

$$K_c = \frac{p_0}{z_0} \qquad \text{since } p_0 < z_0 \text{ then } K_c < 1 \tag{B.32}$$

If K_0 is the gain required for the desired closed-loop pole s_1, then from (B.3)

$$K_0 = -\frac{1}{G_p(s_1)} \tag{B.33}$$

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

$$G_c(s_1) = \frac{K_c(s + z_0)}{s + p_0} \simeq K_c \tag{B.34}$$

Now, the gain K required to place a closed-loop pole at approximately s_1 is given by

$$K = -\frac{1}{G_c(s_1)G_p(s_1)} \simeq -\frac{1}{K_c G_p(s_1)} \simeq \frac{K_0}{K_c} \tag{B.35}$$

Since $K_c < 1$, then $K > K_0$. Next, select the compensator zero z_0, arbitrarily small. Then from (B.31) the compensator pole is

$$p_0 = K_0 z_0 \tag{B.36}$$

The compensated system transfer function is then given by

$$KG_pG_c = KK_c\frac{s+z_0}{s+p_0}G_p \tag{B.37}$$

A lag-lead controller may be obtained by appropriately combining a lag and a lead network in series.

B.5.4 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 (B.25), the following equations are obtained to satisfy (B.26).

$$K_P = \frac{-\sin(\beta+\psi)}{M\sin\beta} - \frac{2K_I\cos\beta}{|s_1|}$$

$$K_D = \frac{\sin\psi}{|s_1|\,M\sin\beta} + \frac{K_I}{|s_1|^2} \tag{B.38}$$

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, (B.38) is applied only for the complex pole s_1.

B.5.5 PD CONTROLLER

Here, both the error and its derivative are used for control, and the compensator transfer function is

$$G_c(s) = K_P + K_D s = K_D\left(s + \frac{K_P}{K_D}\right) \tag{B.39}$$

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 **[numo, deno, denc] = rldesign(num, den,** s_1) with option 4 is used for the PD controller design.

B.5.6 PI CONTROLLER

The integral of the error as well as the error itself is used for control, and the compensator transfer function is

$$G_c(s) = K_P + \frac{K_I}{s} = \frac{K_P(s+K_I/K_P)}{s} \tag{B.40}$$

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 **[numo, deno, denc] = rldesign(num, den,** s_1) with option 5 is used for the PI controller design.

B.5.7 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 **[numo, deno, denc] = rldesign(num, den,** s_1) with option 6 is used for the PID controller design.

Based on the above equations, several functions are developed for the root-locus design. These are

Function	Controller
[numo, deno, denc] = pcomp(num, den, ζ**)**	Proportional
[numo, deno, denc] = phlead(num, den, s_1**)**	Phase-Lead
[numo, deno, denc] = phlag(num, den, ζ**)**	Phase-Lag
[numo, deno, denc] = pdcomp(num, den, s_1**)**	PD
[numo, deno, denc] = picomp(num, den, s_1**)**	PI
[numo, deno, denc] = pidcomp(num, den, s_1**)**	PID

Alternatively, the function **[numo, deno, denc] = rldesign(num, den,** s_1) displays a menu with six options that allow 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 ζ, 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 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 B.1 (exb1)

The block diagram of a control system is as shown in Figure B.3. $G_c(s)$ is a simple proportional controller of gain K.

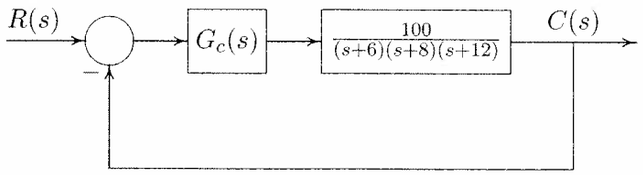


FIGURE B.3
Control system for Example B.1.

(a) Construct the Routh-Hurwitz array and determine the range of K for closed-loop stability.
(b) Find the value of K to yield a steady-state error of 0.15 for a unit step input.
(c) Use *MATLAB* **rlocus** function to obtain the root-locus plot.
(d) Use **rldesign** and option 1 to find the gain K_0 such that the dominant closed-loop poles damping ratio will be equal to 0.96. Obtain the step response curves for K_0, and the value of K found in (b).

(a) The closed-loop transfer function of the control system shown in Figure B.3 is

$$\frac{C(s)}{R(s)} = \frac{100K}{s^3 + 26s^2 + 216s + 576 + 100K}$$

The Routh-Hurwitz array for this polynomial is then (see Appendix B.2.1)

s^3	1	216
s^2	26	$576 + 100K$
s^1	$193.846 - 3.846K$	0
s^0	$576 + 10K$	0

From the s^1 row we see that, for control system stability, K must be less than 50.4, also from the s^0 row, K must be greater than -5.76. Thus, with positive values of K, for control system stability, the gain must be

$$K < 50.4$$

For $K = 50.4$, the auxiliary equation from the s^2 row is

$$26s^2 + 5616 = 0$$

or $s = \pm j14.7$. That is, for $K = 50.4$, we have a pair of conjugate poles on the $j - \omega$ axis, and the control system is marginally stable.

(b) The position error constant given by (B.11) is

$$K_p = \lim_{s\to 0} G_c(s)G_p(s) = \lim_{s\to 0} \frac{100K}{(s+6)(s+8)(s+12)} = \frac{100K}{576}$$

For a unit step input

$$e_{ss} = \frac{1}{1 + K_p} = 0.15$$

Thus

$$K_p = 5.667 = \frac{100K}{576}$$

or

$$K = 32.64$$

The closed-loop transfer function for this gain is

$$\frac{C(s)}{R(s)} = \frac{(100)(32.64)}{s^3 + 26s^2 + 216s + 3840}$$

(c) The *MATLAB Control Toolbox* function **rlocus** is used to obtain the root-locus plot.

(d) To find the gain for the step response damping ratio of $\zeta = 0.96$, and the step response plots, we use the following commands.

```
num = 100;
den = [1  26  216  576];
figure(1), rlocus(num, den), grid,  axis([-20 0 -15 15]);
zeta = 0.96;                                  % damping ratio
[numo, deno, denc]=rldesign(num,den,zeta);% Gain controller
```

```
t = 0:.005:4;
c1 = step(numo, denc, t); % Step response for zeta = 0.96
num2 = 100*32.64;  den2 = [1 26 216 3840];
c2 = step(num2, den2, t);   % Step response for K = 32.64
figure(2), plot(t, c1,  t, c2) , grid
xlabel('t, sec'), ylabel('c(t)')
text(3.1, 0.75, 'K = 32.64'), text(3.1, 0.1, 'K = 0.28')
timespec(num2, den2)    % Time-domain spec. for K = 32.64
```

The result is

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:          K0 = 0.28

Row vectors of polynomial coefficients of the compensated
system:

   Open-loop num.     28
   Open-loop den.      1   26   216   576
   Closed-loop den.    1   26   216   604

   Roots of the compensated characteristic equation:
   -12.8445
   -6.5778 + 1.9383i
   -6.5778 - 1.9383i

   Peak time = 0.289          Percent overshoot = 65.9
   Rise time = 0.096
   Settling time = 3.3
```

The root-locus plot is shown in Figure B.4, and the step response is shown in Figure B.5. The step response damping ratio of 0.96 resulted in a controller gain of $K_0 = 0.28$.

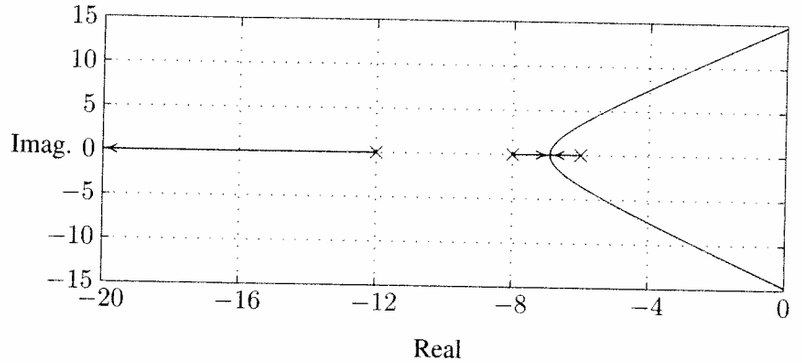


FIGURE B.4
Root-locus plot for Example B.1.

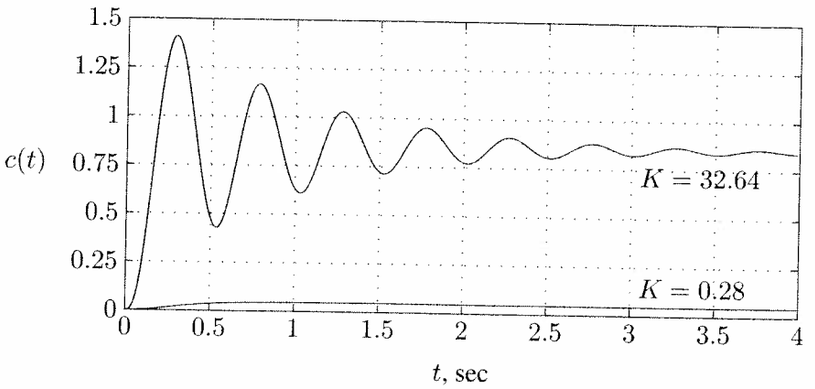


FIGURE B.5
Step response with proportional controller for Example B.1.

From Figure B.5, we see that the transient response is satisfactory, but the steady-state error given by

$$e_{ss} = \frac{1}{1 + \frac{28}{576}} = 0.9536$$

is very large, and the steady-state response is $1 - 0.9536 = 0.0464$. In order to reduce the steady-state error, the gain must be increased. The gain for a steady-state error of 0.15 was found to be 32.64, but the step response is highly oscillatory with an overshoot of 65.9 percent, which is not satisfactory.

```
t = 0:.005:4;
c1 = step(numo, denc, t); % Step response for zeta = 0.96
num2 = 100*32.64;  den2 = [1 26 216 3840];
c2 = step(num2, den2, t);   % Step response for K = 32.64
figure(2), plot(t, c1,  t, c2) , grid
xlabel('t, sec'), ylabel('c(t)')
text(3.1, 0.75, 'K = 32.64'), text(3.1, 0.1, 'K = 0.28')
timespec(num2, den2)    % Time-domain spec. for K = 32.64
```

The result is

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:          K0 = 0.28

Row vectors of polynomial coefficients of the compensated
system:

   Open-loop num.     28
   Open-loop den.      1   26   216   576
   Closed-loop den.    1   26   216   604

   Roots of the compensated characteristic equation:
   -12.8445
   -6.5778 + 1.9383i
   -6.5778 - 1.9383i

   Peak time = 0.289          Percent overshoot = 65.9
   Rise time = 0.096
   Settling time = 3.3
```

The root-locus plot is shown in Figure B.4, and the step response is shown in Figure B.5. The step response damping ratio of 0.96 resulted in a controller gain of $K_0 = 0.28$.

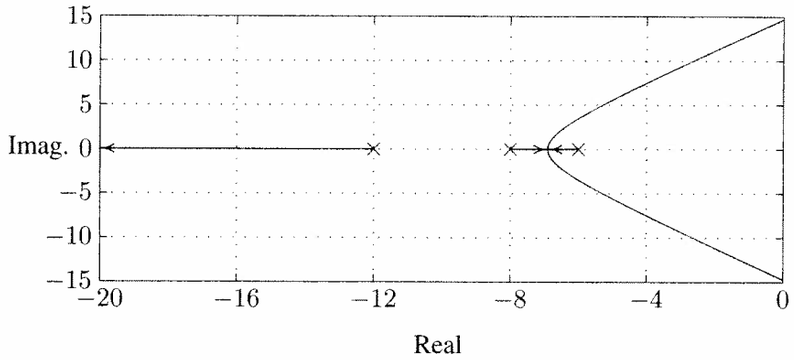


FIGURE B.4
Root-locus plot for Example B.1.

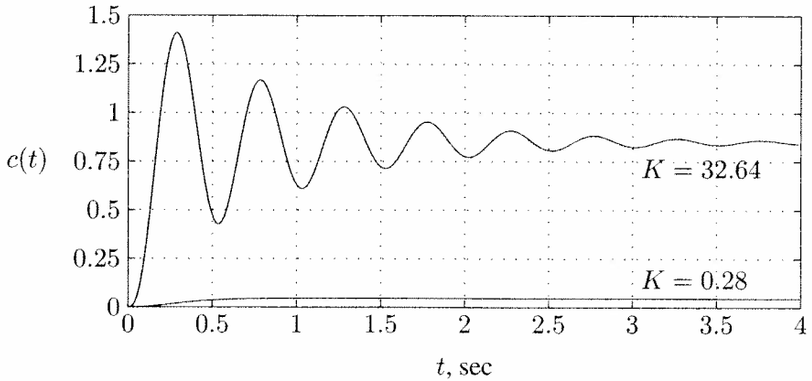


FIGURE B.5
Step response with proportional controller for Example B.1.

From Figure B.5, we see that the transient response is satisfactory, but the steady-state error given by

$$e_{ss} = \frac{1}{1 + \frac{28}{576}} = 0.9536$$

is very large, and the steady-state response is $1 - 0.9536 = 0.0464$. In order to reduce the steady-state error, the gain must be increased. The gain for a steady-state error of 0.15 was found to be 32.64, but the step response is highly oscillatory with an overshoot of 65.9 percent, which is not satisfactory.

Example B.2 (exb2)

For the control system in Example B.1, design a controller to meet the following specifications.

- Zero steady-state error for a step input
- Step response dominant poles damping ratio $\zeta = 0.995$
- Step response dominant pole time constant $\tau = 0.1$ second

The plant transfer function of the control system in Example B.1 is type zero. To reduce the steady-state error to zero, we must increase the system type by one. Thus, we select a PID controller, i.e.,

$$G_c(s) = K_P + \frac{K_I}{s} + K_D s$$

From the last two specifications, $\zeta\omega_n = \frac{1}{\tau} = 10$, and $\theta = \cos^{-1} 0.995 = 5.73°$. Thus, the required complex closed-loop poles are $-10 \pm j1$. The function **rldesign** with option 6 is used for a PID controller design. The user is prompted to enter a value for the integral gain K_I, and the program determines K_P and K_D. The process may be repeated for different values of K_I until a satisfactory response is obtained. For this example, use a value of 9.09 for K_I. The following commands

```
num = 100; den = [1  26  216  576];
s1= -10+j*1;        % Desired location of closed-loop poles
[numo, deno, denc]=rldesign(num, den, s1);     %PID design
t = 0:.01:4;
step(numo, denc, t), grid
xlabel('Time -sec.'), ylabel('c(t)')
```

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 → 9.09
```



```
Gc = 2.1 + 9.09/s + 0.14s

Row vectors of polynomial coefficients of the compensated
system:

  Open-loop num.    14   210   909
  Open-loop den.     1    26   230   576     0
  Closed-loop den    1    26   230   786   909

  Roots of the compensated characteristic equation:
  -10 + 1i
  -10 - 1i
  -3
  -3
```

Thus, the compensated open-loop transfer function is

$$G_c G_p = \frac{14s^2 + 210s + 909}{s(s^3 + 26s^2 + 216s + 576)}$$

and the compensated closed-loop transfer function is

$$\frac{C(s)}{R(s)} = \frac{14s^2 + 210s + 909}{s^4 + 26s^3 + 2230s^2 + 786s + 909}$$

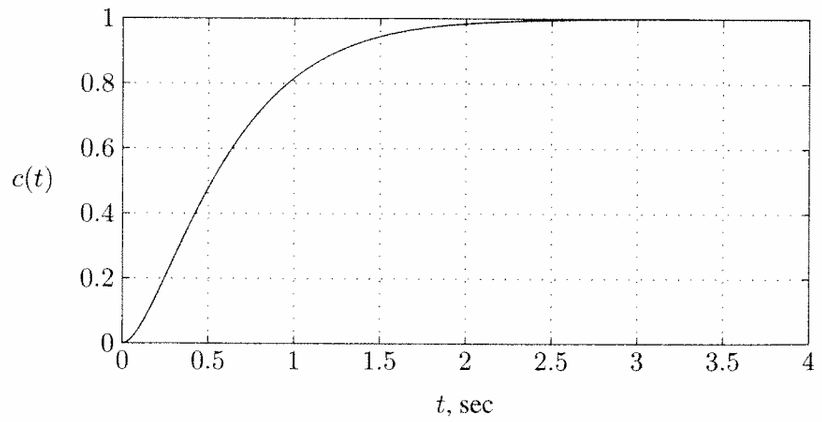


FIGURE B.6
Step response with PID controller for Example B.2.

The PID controller increases the system type by 1. That is, we have a type 1 system, and the steady-state error due to a unit step input is zero. The transient response is also improved as shown in Figure B.6.

B.6 FREQUENCY RESPONSE

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.

The response of a linear time-invariant system to sinusoidal input $r(t) = A\sin(\omega t)$ is given by

$$c(t) = A\,|G(j\omega)|\,\sin[\omega t + \theta(\omega)] \tag{B.41}$$

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

$$G(j\omega) = |G(j\omega)|\,\angle\theta(\omega) \tag{B.42}$$

Alternatively, the transfer function can be represented in rectangular complex form as

$$G(j\omega) = \Re\, G(j\omega) + j\Im\, G(j\omega) = R(j\omega) + jX(j\omega) \tag{B.43}$$

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*.

B.6.1 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 ω, the frequency. The second graph of a Bode plot shows the phase function $\theta(\omega)$ versus the logarithm of ω. 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,** ω**)** or **bode(A, B, C, D, iu,** ω**)** uses the user-supplied frequency vector ω. 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,** ω**]**, the frequency response of the system in the matrices **mag**, **phase**, and ω are returned, and we need to use **plot** or **semilogx** functions to obtain the plot.

B.6.2 POLAR PLOT

A *polar plot*, also called the *Nyquist plot*, is a graph of $\Im G(j\omega)$ versus $\Re G(j\omega)$ with ω 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.

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

B.6.3 RELATIVE STABILITY

The closed-loop transfer function of a control system is given by

$$T(s) = \frac{C(s)}{R(s)} = \frac{KG(s)}{1 + KGH(s)} \tag{B.44}$$

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. All points on the root locus satisfy the following conditions.

$$|\,KGH(s)\,| = 1 \qquad \text{and} \qquad \angle GH(s) = -180^\circ \tag{B.45}$$

The intersection of the polar plot with the negative real axis has a phase angle of -180°. The frequency ω_{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

$$| K_c GH(j\omega_{pc}) | = 1 \qquad \text{and} \qquad \angle GH(j\omega_{pc}) = -180^\circ \tag{B.46}$$

The closed-loop response becomes marginally stable when the frequency response magnitude is unity and its phase angle is -180°. 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

$$| KGH(j\omega) | < 1 \qquad \text{at} \qquad \angle GH(j\omega_{pc}) = -180^\circ \tag{B.47}$$

The proximity of the $KGH(j\omega)$ plot in the polar coordinates to the $(-1, 0)$ point gives an indication of the stability of the closed-loop system.

B.6.4 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 amount of gain 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

$$G.M. = \frac{K_c}{K} \tag{B.48}$$

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

$$G.M. = \frac{K_c \, |GH(j\omega_{pc})|}{K \, |GH(j\omega_{pc})|} = \frac{1}{K \, |GH(\omega_{pc})|} = \frac{1}{a} \tag{B.49}$$

In terms of decibels, the gain margin is

$$G.M._{dB} = 20 \log_{10}(G.M.) = -20 \log_{10} |KGH(j\omega_{pc})| = -20 \log_{10} a \tag{B.50}$$

The gain margin is simply the factor by which K must be changed in order to render the system 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 ω_{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

$$P.M. = \angle GH(j\omega_{gc}) - (-180^\circ) \tag{B.51}$$

For satisfactory performance, the phase margin should be between 30° and 60°, and the gain margin should be greater than 6 dB. The *MATLAB Control System Toolbox* function **[Gm, Pm,** ω_{pc} **,** ω_{gc} **] = margin(mag, phase** ω**)** can be used with **bode** function for evaluation of gain and phase margins, ω_{pc} and ω_{gc}.

B.6.5 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° point on the Bode plot. The Nyquist criterion is based upon a theorem of complex variable mathematics developed by 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

$$Z = N + P \tag{B.52}$$

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

$$N = -P \tag{B.53}$$

B.6.6 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 ω increasing from 0^+ to ∞ 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	stable	$\omega_{pc} > \omega_{gc}$	$\|GH(j\omega_{pc})\| < 1,\ GM_{dB} > 0,\ PM > 0^\circ$
On	-1	marg. stable	$\omega_{pc} = \omega_{gc}$	$\|GH(j\omega_{pc})\| = 1,\ GM_{dB} = 0,\ PM = 0^\circ$
Left of	-1	not stable	$\omega_{pc} < \omega_{gc}$	$\|GH(j\omega_{pc})\| > 1,\ GM_{dB} < 0,\ PM < 0^\circ$

If P is not zero, the closed-loop system is stable if and only if the number of counterclockwise encirclements of the Nyquist diagram about $(-1, 0)$ point is equal to P.

The *MATLAB Control System Toolbox* function **[re, im] = nyquist(num, den,** ω) can obtain the Nyquist diagram by mapping the Nyquist path. However, the argument ω 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 ω 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 $GH(s)$.

B.6.7 CLOSED-LOOP FREQUENCY RESPONSE

The *closed-loop frequency response* is the frequency response of the closed-loop transfer function $T(j\omega)$. The *Control System Toolbox* function **bode**, described in Section B.6.1, is used to obtained the closed-loop frequency.

The performance specifications in terms of closed-loop frequency response are the closed-loop system *bandwidth* ω_B and the closed-loop system *resonant peak magnitude* M_p. The bandwidth ω_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 fast 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 ω_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, ω_r, and the bandwidth ω_B from the frequency response data.

B.6.8 FREQUENCY RESPONSE DESIGN

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.

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.

$$G_c(s) = K_P + \frac{K_I}{s} + K_D s \tag{B.54}$$

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

$$G_c(s) = \frac{K_c(s + z_0)}{s + p_0} \tag{B.55}$$

Several functions have been developed for the selection of suitable controller parameters based on the satisfaction of frequency response criteria, such as gain margin and phase margin. These functions tabulated below.

Alternatively, the function **[numo, deno, denc] = 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.

Function	Controller
[numo, deno, denc] = frqp(num, den)	Proportional
[numo, deno, denc] = frqlead(num, den)	Phase-lead
[numo, deno, denc] = frqlag(num, den)	Phase-lag
[numo, deno, denc] = frqpd(num, den)	PD
[numo, deno, denc] = frqpi(num, den)	PI
[numo, deno, denc] = frqpid(num,den)	PID

Example B.3 (exb3)

Design a PID controller for the system of Example B.1 for a compensated system phase margin of 77.8°. Choose a value of 9.09 for K_I, and select the new phase crossover frequency of 1.53 rad/s. Also, obtain the Bode plot of the compensated open-loop transfer function. The following commands:

```
num = 100;   den = [1  26  216  576];
[numo, deno, denc]=frdesign(num, den); % PID design
w = .1:.1:20;
[mag, phase] =bode(numo, deno, w); dB = 20*log10(mag);
figure(1), plot(w, dB), grid
xlabel('w, rad/sec'), ylabel('dB')
figure(2), plot(w, phase), grid
xlabel('w, rad/sec'), ylabel('Degrees')
```

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 KI → 9.09
Enter desired Phase Margin → 77.8
Enter wgc → 1.53
Uncompensated control system
Gain Margin = 50.4 Gain crossover w = NaN
Phase Margin = Inf Phase crossover w = 14.7
Gc = 2.10655 + 9.09/s + 0.14074s
Row vectors of polynomial coefficients of the compensated
system:
 Open-loop num.    14.07  210.65     909
 Open-loop den.        1      26     216     576    0
 Closed-loop den       1      26  230.07  786.65  909
    Gain Margin = 30300  Gain crossover w = 1.53
    Phase Margin = 77.8  Phase crossover w = 653
    Bandwidth = 1.95
Roots of the compensated characteristic equation:
-9.9943 + 1.0097i
-9.9943 - 1.0097i
-3.1671
-2.8444
```

The PID controller increases the system type by 1. That is, we have a type 1 system, and the steady-state error due to a unit step input is zero, and the step response is similar to Figure B.6. The compensated open-loop Bode plot is shown in Figure B.7.

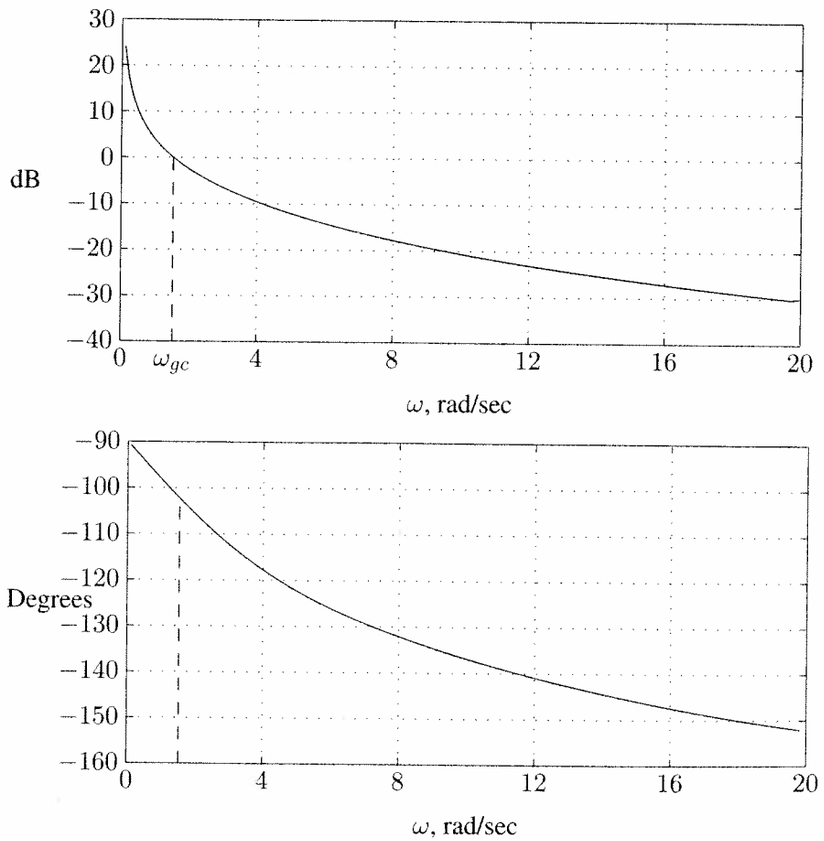


FIGURE B.7
The compensated open-loop Bode plot for Example B.3.

B.7 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 or higher.

B.7.1 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 B.4 (exb4)

Obtain the closed-loop transfer function for the control system shown in Figure B.8.
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:
     4 s^3 + 23 s^2 + 62 s + 64
---------------------------------
s^4 + 14 s^3 + 96 s^2 + 214 s + 320
```

B.7.2 The LTI Viewer

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

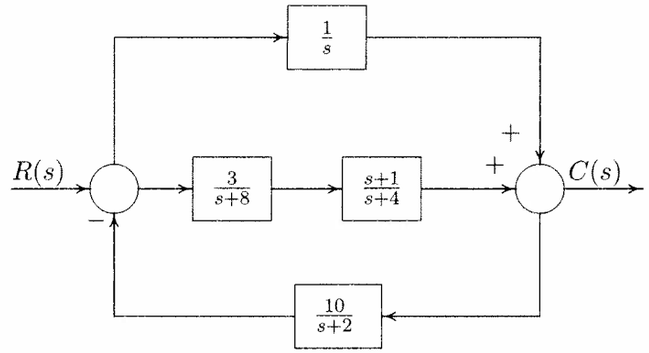


FIGURE B.8
Block diagram for Example B.4.

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 B.5 (exb5)

Use the LTI Viewer to obtain the step response and the closed-loop frequency response for the control system shown in Figure B.9.

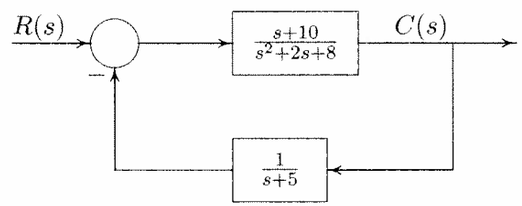


FIGURE B.9
Block diagram for Example B.5

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 B.10. 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 B.11. The mouse right-click is used to obtain the response peak amplitude.

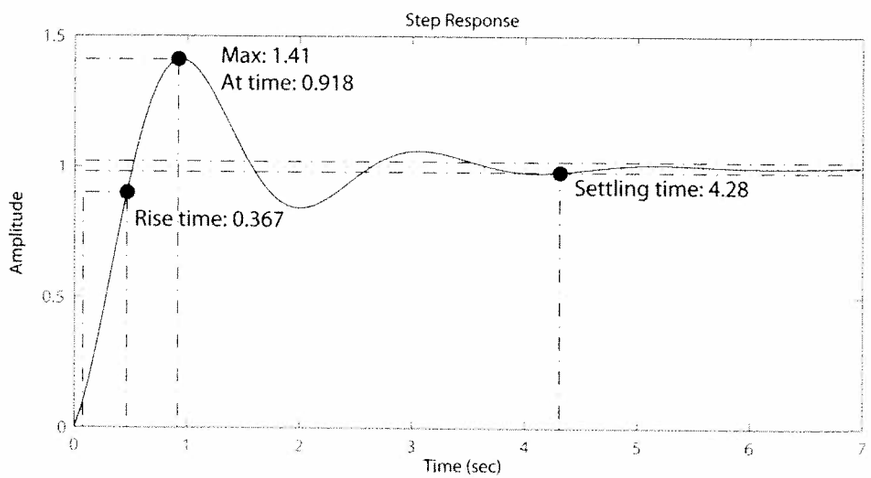


FIGURE B.10
Step response of Example B.5.

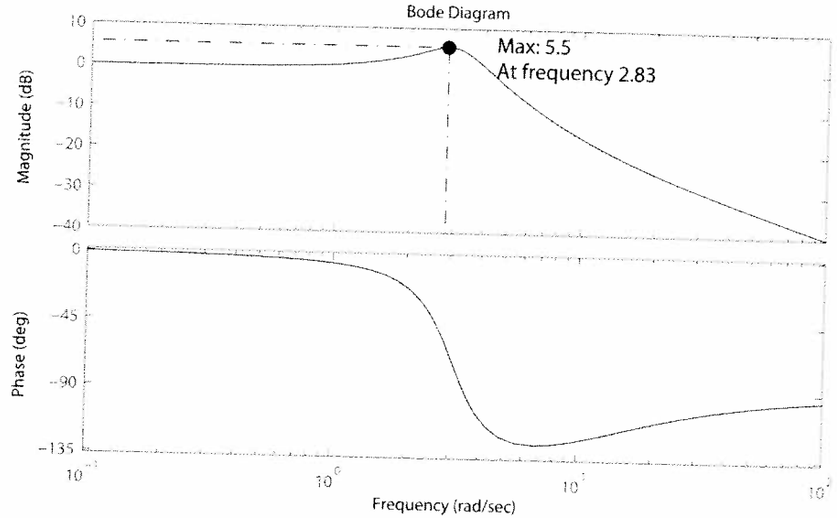


FIGURE B.11
Frequency response of Example B.5.

APPENDIX C

POWER SYSTEM TOOLBOX

The Power System Toolbox, containing a set of M-files, has been developed by the author to assist in typical power system analysis. Some of the programs, such as power flow, optimization, short-circuit and stability analysis, were originally developed by the author for a mainframe computer when working for power system consulting firms many years ago. These programs have been refined and modularized for interactive use with *MATLAB* for many problems related to the operation and analysis of power systems. The software modules are structured in such a way that the user may mix them for other power system analyses. The M-files for typical power system analyses are designed to work in synergy and communicate with each other through the use of some global variables.

The CD-Rom included with this book contains all the developed functions and chapter examples. Instructions for installing the Power System Toolbox can be found with the Installing the Text Toolbox described in Appendix A. This appendix contains a list of all functions and script files in the Power System Toolbox developed by the author. The file names for the chapter examples are also included.

LIST OF FUNCTIONS, SCRIPT FILES, AND EXAMPLES IN THE POWER SYSTEM TOOLBOX

Load Cycle	
barcycle(data)	Plot load cycle for a given load interval

Transmission Line Parameters	
[GMD, GMRL, GMRC] = gmd	Multicircuits GMD and GMR
[L, C] = gmd2lc	Multicircuit GMD, GMRL, L, and C
acsr	Displays the ACSR characteristics

Transmission Line Performance	
lineperf	Line performance program
[r, L, C, f] = abcd2rlc(ABCD)	ABCD to rLC conversion
[Z, Y, ABCD] = abcd2pi(A, B, C)	ABCD to π model conversion
[Z, Y, ABCD] = pi2abcd(Z, Y)	π model to ABCD conversion
[Z, Y, ABCD] = rlc2abcd(r, L, C, g, f, Ln)	rLC to ABCD conversion
[Z, Y, ABCD] = zy2abcd(z, y, Ln)	zy to ABCD conversion
listmenu	Displays 8 options for analysis
givensr(ABCD)	Sending end values from receiving end power
givenss(ABCD)	Receiving end values from sending end power
givenzl(ABCD)	Sending end values from load impedance
loadabil(L, C, f)	Line loadability curves
openline((ABCD)	Open line analysis and reactor compensation
shcktlin(ABCD)	Receiving end short circuit
compmenu	Displays 3 options for capacitive compensation
sercomp(ABCD)	Series capacitor compensation
shntcomp(ABCD)	Shunt capacitor compensation
srshcomp(ABCD)	Shunt and series capacitors compensation
profmenu	Displays two options for loadabil and vprofile
pwrcirc(ABCD)	Receiving end power circle diagram
vprofile(ABCD)	Voltage curves for various loading

Optimal Dispatch of Generation	
bloss	Returns loss coefficients when followed by power flow program
dispatch	Obtains optimum dispatch of generation
gencost	Computes the total generation cost $/hr

Transformer and Induction Motor	
trans	Transformer characteristics
tperf	This script file is called by trans
[Rc, Xm] = troct(Vo, Io, Po)	Shunt branch from OC test
[Ze] = trsct(Vsc, Isc, Psc)	Obtains the series branch from SC test
[Ze1, Ze2]= trsct(E1, E2, Z1, Z2)	Winding impedances to Eq. impedance
rotfield	Revolving field demonstration
im	Equivalent circuit analysis
imchar	Torque/speed curve (called by im)
imsol	Motor performance (called by im)

Power Flow Analysis	
ybus1	Obtains Y_{bus}, given R and X values
lfybus	Obtains Y_{bus}, given π model with specified *linedata* file
lfgauss	Power flow solution by the Gauss-Seidel method
lfnewton	Power flow solution by the Newton-Raphson method
decouple	Power flow solution by the Fast Decoupled method
busout	Returns the bus output result in tabular form
lineflow	Returns the line flow and losses in tabular form

Symmetrical Components	
sctm	Symmetrical Components Transformation Matrix
phasor(F)	Plots phasors expressed in rectangular or polar
F012 = abc2sc(Fabc)	Phasors to symmetrical components conversion
Fabc = sc2abc(F012)	Symmetrical components to phasors conversion
Z012 = zabc2sc(Zabc)	Impedance matrix to symmetrical components
Fr = pol2rec(Fp)	Polar phasor to rectangular phasor conversion
Fp = rec2pol(Fr)	Rectangular phasor to polar phasor conversion

Fault Analysis	
dlgfault(Z0, Zbus0, Z1, Zbus1, Z2, Zbus2, V)	Double line-to-ground fault
lgfault(Z0, Zbus0, Z1, Zbus1, Z2, Zbus2, V)	Line-to-ground fault
llfault(Z1, Zbus1, Z2, Zbus2, V)	Line-to-line fault
symfault(Z1, Zbus1, V)	Line-to-ground fault
Zbus = zbuild(zdata)	Builds the Bus Impedance Matrix
Zbus = zbuildpi(linedata, gendata, load)	Builds the Bus Impedance Matrix, compatible with load flow data

Synchronous Machine Transients	
lgshort(t,i)	Returns state derivatives of current for L-G short circuit
llshort(t, i)	Returns state derivatives of current for line-line short circuit
symshort(t, i)	Returns state derivatives of current for 3-phase short circuit

Power System Stability	
cctime	Obtains the critical clearing time for fault
eacfault(P0, E, V, X1, X2, X3)	Displays equal area criterion & finds critical clearing time of fault
eacpower(P0, E, V, X)	Displays equal area criterion & max. steady-state power
xdot = afpower(t, x)	One-machine system state derivative after fault
xdot = pfpower(t, x)	One-machine system state derivative during fault
swingmeu(Pm, E, V, X1, X2, X3, H, f, tc, tf)	One-machine swing curve, modified Euler
swingrk2(Pm, E, V, X1, X2, X3, H, f, tc, tf)	One-machine swing curve, *MATLAB* ode23
swingrk4(Pm, E, V, X1, X2, X3, H, f, tc, tf)	One-machine swing curve, *MATLAB* ode34
xot = afpek(t, x)	Multimachine system state derivative after fault
xdot = dfpek(t, x)	Multimachine system state derivative during fault
trstab	Stability analysis works in synergy with load flow
[Ybus, Ybf] = ybusbf(linedata, yload, nbus1, nbust)	Multimachine system reduced Y_{bus} before fault
Ypf = ybusbf(Ybus, nbus1, nbust, nf)	Multimachine system reduced Y_{bus} during fault
Yaf = ybusaf(linedata, yload, nbus1, nbust, nbrt)	Multimachine system reduced Y_{bus} after fault

Control System Functions	
electsys	Returns the state derivatives for Example A.19
errortf	Steady-state error, transfer function in polynomial form
errorzp	Steady-state error, transfer function in zero pole form
frcntrl	Frequency response design equations
frdesign	Frequency response design program
frqlag	Frequency response design phase-lag controller
frqlead	Frequency response design phase-lead controller
frqp	Frequency response design P controller
frqpd	Frequency response design PD controller
frqpi	Frequency response design PI controller
frqpid	Frequency response design PID controller
frqspec	Frequency response performance specifications
ghs	Returns magnitude and phase of a complex function $GH(s)$
ltstm	Laplace transform of state transition matrix
mechsys	Returns the state derivatives for Example A.18
pcomp	Root-locus design P controller
pdcomp	Root-locus design PD controller
pdlead	Root-locus design phase-lead controller
pendulum	Returns the state derivatives for Example A.20
phlag	Root-locus design phase-lag controller
picomp	Root-locus design PI controller
pidcomp	Root-locus design PID controller
placepol	Pole-placement design
pnetfdbk	Feedback compensation using passive elements
riccasim	Returns state derivative of Riccati equation
riccati	Optimal regulator design
rldesign	Root-locus design program
routh	Routh-Hurwitz array
ss2phv	Transformation to control canonical form
statesim	Returns state derivatives for use in Riccati equation
stm	Determines the state transition matrix $\phi(t)$
system	System matrices defined for use in Riccati equation
tachfdbk	Tachometer feedback control
timespec	Time-domain performance specifications

List of M-Files for Chapter Examples				
CHP1EX7	CHP5EX5	CHP7EX7	CHP10EX3	EXA3
CHP2EX1	CHP5EX6	CHP7EX8	CHP10EX4	EXA4
CHP2EX2	CHP5EX7	CHP7EX9	CHP10EX5	EXA5
CHP2EX3	CHP5EX8	CHP7EX10	CHP10EX6	EXA6
CHP2EX4	CHP5EX9	CHP7EX11	CHP10EX7	EXA7
CHP2EX5	CHP6EX1	CHP8EX1	CHP10EX8	EXA8
CHP2EX6	CHP6EX2	CHP8EX2	CHP11EX1	EXA9
CHP2EX7	CHP6EX3	CHP8EX3	CHP11EX2	EXA10
CHP2EX8	CHP6EX4	CHP8EX4	CHP11EX3	EXA11
CHP3EX1	CHP6EX5	CHP8EX5	CHP11EX4	EXA12
CHP3EX2	CHP6EX6	CHP8EX6	CHP11EX5	EXA13
CHP3EX3	CHP6EX7	CHP8EX7	CHP11EX6	EXA14
CHP3EX4	CHP6EX8	CHP8EX8	CHP11EX7	EXA15
CHP3EX5	CHP6EX9	CHP8EX9	CHP12EX1	EXA16
CHP3EX6	CHP6EX10	CHP8EX10	CHP12EX2	EXA17
CHP4EX1	CHP6EX11	CHP9EX1	CHP12EX3	EXA18
CHP4EX2	CHP6EX12	CHP9EX2	CHP12EX4	EXA19
CHP4EX3	CHP6EX13	CHP9EX3	CHP12EX5	EXA20
CHP4EX4	CHP6EX14	CHP9EX4	CHP12EX6	EXA21
CHP4EX5	CHP6EX15	CHP9EX5	CHP12EX7	EXA22
CHP4EX6	CHP7EX1	CHP9EX6	CHP12EX8	EXA23
CHP4EX7	CHP7EX2	CHP9EX7	CHP12EX9	EXA24
CHP5EX1	CHP7EX3	CHP9EX8	CHP12XX	EXB1
CHP5EX2	CHP7EX4	CHP9EX9	CHP12XX1	EXB2
CHP5EX3	CHP7EX5	CHP10EX1	EXA1	EXB3
CHP5EX4	CHP7EX6	CHP10EX2	EXA2	EXB4,5

List of SIMULINK-Files for Chapter Examples				
SIM11EX3	SIM11EX6	SIM12EX1	SIM12EX3	SIM12EX4
SIM12EX5	SIM12EX6	SIM12EX7	SIM12EX8	SIM12EX9
SIM12XXB	SIM12XXD	SIM12XX1	SIMEXA25	SIMEXA26
SIMEXA27	SIMEXA28	SIMEXA29	SIMEXA30	SIMEXA31
SIMEXA32				

List of Graphical User Interface Programs (GUI)	
acsrgui	This program displays the ASR code names and allows the user to select up to 15 code names for listing their specifications simultaneously.
chp2ex1gui	This allows you to see instantly the effect of changing load from inductive to resistive and capacitive on the instantaneous power $p(t)$, $p_R(t)$, and $p_X(t)$.
chp2ex6gui	This allows you to see instantly the effect of sweeping voltage phase angle δ_1 on the direction of real power flow, and the effect of sweeping voltage magnitude V_1 on the direction of the reactive power flow.
inductionmotor	In this GUI program the parameters of the equivalent circuit can be entered on the circuit diagram. The program draws the Thévenin's equivalent circuit. Performance characteristics are then obtained for the specified slip and the torque slip curve is plotted on the screen. The analysis can be repeated for different slips.
lcgui	This program is developed for the computation of transmission line parameters. This is a user-friendly program, which makes the data entry for various configurations very easy.
lineperfgui	This interactive easy-to-use GUI program provides maximum flexibility for entering different types of data. It includes eight options for transmission line performance and compensation.
rldesigngui	This GUI program has been developed for the design of a first-order controller in the forward path of a closed-loop control system for proportional, phase-lag, phase-lead, PD, PI, and PID controllers.
transformer	A This GUI program is developed for the transformer tests and analysis. This program obtains the transformer equivalent circuit from open-circuit and short-circuit tests. It also finds the transformer performance characteristics using the transformer parameters.

If you encounter any bugs or problems please contact me at the following e-mail addresses or visit my Web sites for updates and information on this product.

Web sites: http://www.psapublishing.com
http://myweb.msoe.edu/~ saadat

e-mails: info@psapublishing.com
saadat@msoe.edu

BIBLIOGRAPHY

1. Anderson, P. M., Analysis of Faulted Power Systems, IEEE Press, New York, 1973.
2. Anderson, P. M., and Fouad, A. A., Power System Control and Stability, The Iowa State University Press, Ames, Iowa, 1977.
3. Arrillaga, J., Arnold, C. P., and Harker, B. J., Computer Modeling of Electrical Power Systems, John Wiley & Sons, Inc., New York, 1986.
4. Bergen, A. R., Power Systems Analysis, Prentice-Hall, Englewood, Cliffs, New Jersey, 1970.
5. Bergseth, F. R., and Venkata, S. S., Introduction to Electric Energy Devices, Prentice-Hall, Englewood, Cliffs, New Jersey, 1987.
6. Billinton, R., Ringlee, R., and Wood, A., Power System Reliability Calculations, MIT Press, Cambridge, Massachusetts, 1973.
7. Billinton, R., Power System Reliability Evaluations, Gordon and Breach, New York, 1970.
8. Brosan, G. S., and Hayden, J. T., Advanced Electrical Power and Machines, Sir Isaac Pitman & Sons, Ltd., London, 1966.
9. Brown, H. E., Solution of Large Networks by Matrix Methods, John Wiley & Sons, Inc., New York, 1975.
10. Brown, H. E., Person, C. E., Kirchmayer, L. K., and Stagg, W. G., Digital Calculation of Three-Phase Short-Circuits by Matrix Method, AIEE Trans., Part 3, pp. 1277-1282, 1960.

11. Burchett, R. C., Happ, H. H., and Vierath, D. R., Quadratically Convergent Optimal Power Flow, IEEE Trans., PAS-103, No. 11, pp. 3267-3275, November 1984.

12. Burchett, R.C., Happ, H. H., and Wirgau, K. A., Large Scale Optimal Power Flow, IEEE Trans., PAS-101, No. 10, pp. 3722-3731, October 1982.

13. Byerly, R. T., and Kimbark, E. W., Stability of Large Electric Power Systems, IEEE Press, 1974.

14. Carson, J. R., Wave Propagation in Overhead Wires with Ground Return, Bell System Technical Journal, Vol. 5, pp. 539-554, October 1928.

15. Clarke, E., Circuit Analysis of AC Power Systems, John Wiley & Sons, Inc., New York, 1958.

16. Cohn, N., Control of Generation and Power Flow on Interconnected Systems, John Wiley & Sons, Inc., New York, 1966.

17. Concordia, C., Synchronous Machine—Theory and Performance, John Wiley & Sons, Inc., New York, 1951.

18. Crary, Power System Stability, Vol. II, John Wiley & Sons, Inc., New York, 1955.

19. Del Toro, V., Electric Power Systems, Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

20. Doherty, R. E., and Nickle, C. A., Synchronous Machines, AIEE Trans., Vol. 45, pp. 912-942, 1926.

21. Dommel, H. W., and Sato, N., Fast Transient Stability Solutions, IEEE Trans., Power Apparatus and Systems, PAS-91, pp. 1643-1650, October 1972.

22. Dommel, H. W., and Tinney, W. F., Optimal Power Flow Solutions, IEEE Trans., Power Apparatus and Systems, PAS-87, pp. 1866-1876, October 1968.

23. Dopaz, J. F., Klitin, O. A., Stagg, G. W., and Watson, M., An Optimization Technique for Real and Reactive Power Allocation, Proc. of the IEEE, Vol. 55, No. 11, pp. 1877-1885, 1967.

24. Elgerd, O. I., Electric Energy Systems Theory, Second Edition, McGraw-Hill Book Company, New York, 1982.

25. El-Abiad, A. H., Digital Calculation of Line-to-Ground Short-circuits by Matrix Method, AIEE Trans., Vol. 79, pp. 323-332, 1960.

26. El-Hawary, M. E., Electrical Power Systems Design and Analysis, Reston Publishing Company, Reston, Virginia, 1983.

27. EPRI, Transmission Line Reference Book, 345 KV and Above, Electric Power Research Institute, Palo Alto, California, 1982.

28. Fink, D. G., and Beaty, H. W., Standard Handbook for Electrical Engineers, McGraw-Hill Book Company, New York, 1987.

29. Fitzgerald, A. E., Kingsley, C., and Umans, S., Electric Machinery, Fourth Edition, McGraw-Hill, New York, 1982.

30. Fortescue, C. L., Method of Symmetrical Components Applied to the Solution of Polyphase Networks, AIEE Trans., Vol. 37, pp. 1027-1140, 1918.

31. Fouad, A. A. and Stanton, S.E., Transient Stability of Multimachine Power Systems, Part I and II, IEEE Trans., Vol. PAS-100, pp. 3408-3424, July 1981.

32. Fouad, A. A., Vijay, V., Power System Transient Stability Analysis Using the Transient Energy Function Method, Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

33. Gless, G. E., Direct Method of Lyapunov Applied to Transient Power System Stability, IEEE Trans., Vol. PAS-85, No. 2, pp. 159-168, February 1966.

34. Glover, J. D., Power System Analysis and Design, Second Edition, PWS Publishing Company, Boston, 1994.

35. Gönen, T., Electric Power Distribution Systems Engineering, McGraw-Hill Book Company, New York, 1986.

36. Greenwood, A., Electrical Transients in Power Systems, Wiley Interscience, New York, 1971.

37. Gross, C. A., Power System Analysis, Second Edition, John Wiley & Sons, New York, 1983.

38. Guile, A. E., and Paterson, W., Electrical Power Systems, Oliver & Boyd, Edinburgh, 1969

39. Gungor, B. R., Power Systems, Harcourt, Brace, Jovanovich, Inc., New York, 1988.

40. Happ, H., Optimal Power Dispatch—A Comprehensive Survey, IEEE Trans., Power Apparatus and Systems, PAS-96, pp. 841-854, June 1977.

41. Heydt, G. T., Computer Analysis Methods for Power Systems, Macmillan Publishing Company, New York, 1986.

42. Horton, J. S., and Grigsby, L. L., Voltage Optimization Using Combined Linear Programming and Gradient Techniques, IEEE Trans., PAS-103, No. 7, pp. 1637-1643, July 1984.

43. IEEE Standard 115, Test Procedure for Synchronous Machines.

44. Kimbark, E. W., Power System Stability, Vol. 1, Elements of Stability Calculations, John Wiley & Sons, Inc., New York, 1948.

45. Kimbark, E. W., Power System Stability, Vol. 2, Power Circuit Breakers and Protective Relays, John Wiley & Sons, Inc., New York, 1950.

46. Kimbark, E. W., Power System Stability, Vol. 3, Synchronous Machines, John Wiley & Sons, Inc., New York, 1956.

47. Kirchmayer, L. K., Economic Operation of Power Systems, John Wiley & Sons, Inc., New York, 1958.

48. Kirchmayer, L. K., Economic Control of Interconnected Systems, John Wiley & Sons, Inc., New York, 1959.

49. Knable, A., Electrical Power Systems Engineering, McGraw-Hill Book Company, New York, 1967.

50. Kron, G., Tensorial Analysis of Integrated Transmission Systems, Part I, The Six Basic Reference Frames, AIEE Trans., Vol. 70, pp. 1239-1248, 1951.

51. Kron, G., Tensorial Analysis of Integrated Transmission Systems, Part II, Off-Nominal Turn Ratios, AIEE Trans., Vol. 71, pp. 505-512, 1952.

52. Kundur, P., Power System Stability and Control, McGraw-Hill Book Company, New York, 1994.

53. Kusic, G. L., Computer Aided Power System Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

54. Lee, K. Y., Park, Y. M., and Ortiz, J. L., A United Approach to Optimal Real and Reactive Power Dispatch, IEEE Trans., PAS-104, No. 5, pp. 1147-1153, May 1985.

55. Lewis, W. A., The Principles of Synchronous Machines, IIT Press, Chicago, Illinois, 1949.

56. Lewis, W. A., A Basic Analysis of Synchronous Machines, AIEE Trans., PAS-77, pp. 436-455, 1958.

57. Lyapunov, A. M., Stability of Motion, English translation, Academic Press, Inc., New York, 1967.

58. Mamandur, K. R. C., and Chenoweth, R. D., Optimal Control of Reactive Power Flow for Improvements in Voltage Profile and Real Power Loss Minimization, IEEE Trans., PAS-100, No. 7, pp. 3185-33194, July 1981.

59. Nasar, S. A., Electric Energy Conversion and Transmission, Macmillan Publishing Company, New York, 1985.

60. Neuenswander, J. R., Modern Power Systems, International Textbook Company, Scranton, Pennsylvania, 1971.

61. Park, R. H., Two Reaction Theory of Synchronous Machines—Generalized Method of Analysis, AIEE Trans., Vol. 48, pp. 716-727, 1929.

62. Rustebakke, H. M., Electric Utility Systems and Practices, John Wiley & Sons, Inc., New York, 1983.

63. Saadat, H., Time Domain Simulation of Synchronous Machine Imbalance Faults Using PC-MATLAB, Proc. of 20th North American Power Symposium, pp. 285-291, October 1988.

64. Saadat, H., Microcomputer Applications in Electrical Power Engineering Education, Proc. of the 19th North American Power Symposium, pp. 307-314, October 1987.

65. Saadat H., Optimal Load Flow Solution by the Power Perturbation Technique, IEEE, PES Winter Meeting, A79028-2, 1979.

66. Saadat, H., Steady State Analysis of Power Systems Including the Effects of Control Devices, Vol. 2, Journal of Electric Power System Research, pp. 111-118, 1979.

67. Saadat, H., Application of Quasi-Newton Method to Load Flow Studies and Solution of Load Flow During Three-Phase Fault, Journal of Electric Power System Research, pp. 173-179, 1978.

68. Saadat, H., Computational Aids in Control Systems Using MATLAB, McGraw-Hill Book Company, New York, 1983.

69. Sauer, P. W., and Pai, M. A., Power System Dynamics and Stability, Prentice-Hall, Englewood Cliffs, New Jersey, 1998.

70. Singh, L. P., Advanced Power System Analysis and Design, Halsted Press, New York, 1983.

71. Shoults. R. R., and Sun, D. T., Optimal Power Flow Based upon P-Q Decomposition, IEEE Trans., PAS-101, No. 2, pp. 397-405, February 1982.

72. Shultz, R. D., and Smith, R. A., Introduction to Electric Power Engineering, Harper & Row Publishers, New York, 1985.

73. Stagg, G. W., and El-Abiad, A. H., Computer Methods in Power System Analysis, McGraw-Hill Book Company, New York, 1968.

74. Stevenson, W. D., and Grainger, J. J., Power System Analysis, McGraw-Hill Book Company, New York, 1994.

75. Stott, B., Decoupled Newton Load Flow, IEEE Trans. Power Apparatus and Systems, PAS-91, pp. 1955-1959, October 1972.

76. Stott, B., and Alsac, O., Fast Decoupled Load Flow, IEEE Trans. Power Apparatus and Systems, PAS-93, pp. 859-869, May-June 1974.

77. Sullivan, R., Power System Planning, McGraw-Hill Book Company, New York, 1977.

78. Sun, D. I., Ashely, B., Brewer, B., Hughes, A., and Tinney, W. F., Optimal Power Flow by Newton Approach, IEEE Trans., PAS-103, No. 10, pp. 2864-2879, October 1984.

79. Taylor, C. W., Power System Voltage Stability, McGraw-Hill Book Company, New York, 1993.

80. Tinney, W. F., Compensation Methods for Network Solutions by Optimally Ordered Triangular Factorization, IEEE Trans., PAS-91, pp. 123-127, January 1972.

81. Wadhwa, C. L., Electrical Power Systems, John Wiley & Sons, Inc., New York, 1983.

82. Wagner, C. F., and Evens, R. D., Symmetrical Components, McGraw-Hill Book Company, New York, 1933.

83. Wallach, Y., Calculations and Programs for Power System Networks, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

84. Ward, J., and Hale, H., Digital Computer Solution of Power Flow Problems, AIEE Trans., Vol. 75, pt. III, pp. 398-404, 1956.

85. Weedy, B. M., Electric Power Systems, Third Edition, John Wiley & Sons, Inc., New York, 1979.

86. Weeks, W. L., Transmission and Distribution of Electrical Energy, Harper & Row Publishers, New York, 1981.

87. Westinghouse Electric Corporation, Electric Transmission and Distribution Reference Book, East Pittsburgh, Pennsylvania, 1964.

88. Wood, A. J., and Wollenberg, B. F., Power Generation Operation and Control, John Wiley & Sons, Inc., New York, 1974.

89. Yamayee, Z. A., Electromechanical Energy Devices and Power Systems, John Wiley & Sons, Inc., New York, 1994.

90. Yu, Yao-nan, Electric Power Systems Dynamics, Academic Press, New York, 1983.

ANSWERS TO PROBLEMS

Chapter 1

1.1. 15 MW

1.2. 79.94%

1.3. (a) 342.71 kW, 3×10^6 kWh/year, (b) 3.6 cents/kWh

1.4. (a) 120.10 MW, (b) 315.62×10^6 kWh

1.5. 799.34 GW

1.6. 6.93%

1.7. 8 MW, 50%

Chapter 2

2.1. $S_1 = 2000$ W $+ j3464.1$ var, $S_2 = 2165.1$ W $- j1250$ var, $S_3 = 2000$ W $+ j0$ var, $S = 6165.1$ W $+ j2214.1$ var

2.2. (a) 800 W + $j600$ var, (b) $10\cos(377t - 36.87°)$ A, $7.071\angle -36.87°$ A, (c) $20\angle 36.87°$ Ω

2.3. 12.8 Ω, 9.6 Ω

2.4. 20Ω, 26.67Ω

2.5. 280 kW + $j335$ kvar

2.6. (a) 60 Ω, 80 Ω, (b) $1250\angle 16.26°$ V

2.7. (a) $S_1 = 1$ kW + $j7$ kvar, $S_2 = 1$ kW − $j2$ kvar, $S_3 = 4$ kW + $j3$ kvar, (b) $S = 6$ kW + $j8$ Kvar, $50\angle -53.13°$ A, 0.6 lagging, (c) 8 kvar, 530.5 μF, 30 A

2.8. Source 1 delivers 28 kW and receives 21 kvar, Source 2 receives 24.57 kW and delivers 32.76 kvar, 3.43 kW, 11.76 kvar

2.10. (b) 30 kW

2.11. (a) $50\angle -36.87°$ A, $50\angle -156.87°$ A, $50\angle -276.87°$ A, (b) 288 kW, 216 kvar

2.12. (a) $150\angle -66.87°$ A, $150\angle -186.87°$ A, $50\angle 53.13°$ A, (b) 864 kW, 648 kvar

2.13. (a) $12\angle -53.13°$ A, (b) 2592 W + j 3456 var, (c) 162.33 V

2.14. (a) 18 kW, 0 kvar, unity power factor, 50 A, (b) $66.9\angle -41.63°$ A,0.7474 lagging

2.15. (a) 360 kW + $j480$ kvar, 0.6 lagging, $27.78\angle -53.13°$ A, (b) 210 kvar, 3.58 μF, $20.835\angle -36.87°$ A

2.16. (a) $40\angle -36.87°$ A, 19.2 kW + $j14.4$ kvar, (b) 160 V, 277.1 V

Chapter 3

3.1. (a) 2440.80 V, 6.12 %, (b) 2200.4 V, −4.33%

3.2. (a) 44.9 kV, 7.675°, (b) 288 MW, (c) 547.47 A, 0.7306 lagging

3.3. (a) 12806 V, (b) 80.4 MW, (c) $3344\angle 36.73°$

3.4. (b) 16.26°, 30 kV, (c) 138.712 MW at 75°

3.5. (a) $0.4 + j0.9$ Ω, 1000 Ω, $j1500$Ω, (b) 2453.9 V, 2.247%, (c) 2387 V, −0.541%

3.6. (a) $28 + j96$ Ω, 6666.67 Ω, $j5000$ Ω, (b) 21.839%, 85.97%, (c) 53.237 kVA, 86.057%, (d) 85.88%

3.7. (a) 21 kVA, (b) 96%

3.8. 13.346 kV

3.9. (a) 247.69 kV, (b) 249.72 kV

3.10. (a) 1.03205 pu, 247.69 kV, (b) 1.0405 pu, 249.72 kV

3.11. 0.926 pu, 1.0 pu

3.12. $0.122 + j0.252$ pu

3.13. $X_{G_1} = j0.1$, $X_{T_1} = j0.2$, $X_{T_2} = j0.25$, $X_{G_2} = j0.081$, $X_{Line} = j0.3$, $X_{Load} = 0.75 + j1.0$,

3.14. $X_G = j0.3$, $X_{T_1} = j0.2$, $X_{T_2} = j0.15$, $X_{T_3} = j0.16$ $X_{Line_1} = j0.25$, $X_{Line_2} = j0.35$ $X_M = j0.27$, $X_{Load} = -j10$, $Z_P = j0.06$, $Z_S = j0.18$, $Z_T = j0.12$

3.15. (a) $X_{G_1} = j0.15$, $X_{T_1} = j0.2$, $X_{T_2} = j0.2$, $X_{Line} = 0.3 + j.5$ $X_M = j0.15$, (b) 26.359 kV, 27.5 kV

3.16. 440 kV , 480 kV

3.17. 126.5 kV, 27.6 kV

Chapter 4

4.1. (a) 4.1 Ω, (b) 4.6 Ω

4.2. 0.3774 Ω/km

4.3. 1.894 cm, 556000 cmil

4.4. 0.35 cm, 46 mH

4.5. (a) $1.46r$, (b) $1.723r$

4.6. 1.486 mH/km

4.7. 10 m

4.8. (a) 1.3 mH/km, (b) 1.15 cm

4.9. 27.5% decrease, 35.25% increase

4.10. 0.88929 mH/km, 0.012658 μF/km

4.11. 0.4752 mH/km, 0.0240035 μF/km, 0.517453 mH/km, 0.0219974 μF/km

4.12. $\frac{4\pi\varepsilon}{\ln 0.866D/r}$

4.13. 5 V/km

4.14. 90 V

Chapter 5

5.1. (a) 70.508 kV, 10.17%, 58.39 MW $+j50.37$ Mvar, 95.90%,
(b) 69.0 kV, 7.83%, 127 MW $+j24.61$ Mvar, 94.465%

5.2. (a) 14.117 Mvar, 9.14μF, (b) 61.24 Mvar, 39.66μF

5.3. (a) $0.9951 + j0.000544$, $4 + j36$ Ω, $j0.0002713$ S
(b) 242.67 kV, $502.38\angle -33.69°$ A, 10.847%, 163.18 MW $+j134.02$ Mvar, 98.052%
(c) 230.03 kV, $799.86\angle 2.5°$ A, 5.073%, 313.74 MW $+j55.9$ Mvar, 97.53%

5.4. 141.123 Mvar, 7.734μF

5.5. $0.98182 + j0.0012447$, $4.035 + j58.947$, $j0.00061137$

5.6. 387.025 kV, $592.29\angle -27.325°$ A, 324.87 MW, $+j228.25$ Mvar, 14.259%, 98.5%

5.7. 345 kV, $672.54\angle -9.633°$ A, 401.884 MW $+j0.0$ Mvar, 2.73%, 98.743%

5.8. (a) 264.702 Ω, 29°, 4965.2 km, 2210.88 MW, 0.8746, $j128.34$ Ω, $j0.0018316$ S
(b) 896.982 kV, $1100.23\angle -2.456°$ A, 1600 MW $+j601.508$ Mvar, 39.536%
(c) 653.33 kV, $1748.78\angle -43.556°$ A, 1920 MW $+j479.33$ Mvar, 33.88%
(d) 735.13 kV, $1604.07\angle 28.98°$ A, 2042.44 MW $+j1.32$ Mvar, 14.358%

5.9. 874.68 kV, (b) 772.13 Ω, 699.658 Mvar

5.10. $3441.47\angle -90°$ A, $3009.92\angle -90°$ A

5.11. 802.95 Mvar, 3.943 μF, $1209.46\angle 24.653°$ A, 1600 MW $-j90.38$ Mvar, 19%

5.12. 822.677 kV, $1164.59\angle -3.625°$ A, 1600 MW $+j440.16$ Mvar, 21.035%

5.13. 81.464 Mvar, 51.65 μF, 563.25 Mvar, 2.765 μF, 765 kV, $1209.72\angle 16.1°$ A, 1600 MW $-j96.32$ Mvar, 12.55%

5.14. Use **lineperf** to obtain the transmission line performance. Present a summary of the calculations along with your recommendations.

5.15. (a) 622.153 kV, $794.649\angle -1.33°$ A, 800 MW $+j305.408$ Mvar, 44.687%
(b) 0.96, $j39.2$, $j0.002$
(c) 530.759 kV, $891.142\angle -5.65°$ A, 800 MW $+j176.448$ Mvar, 10.575%

5.16. (a) 0.002 Rad/km, 500 Ω, (b) 1000 Ω, 176.4 Mvar

5.17. 400 kV

Chapter 6

6.1. $$Y_{bus} = \begin{bmatrix} 0.0 - j20.25 & 0.0 + j4.00 & 0.0 + j10.00 & 0.0 + j2.50 \\ 0.0 + j4.00 & 0.0 - j15.00 & 0.0 + j0.00 & 0.0 + j6.25 \\ 0.0 + j10.00 & 0.0 + j0.0 & 1.0 - j15.00 & 0.0 + j5.00 \\ 0.0 + j2.50 & 0.0 + j6.25 & 0.0 + j5.00 & 2.0 - j14.00 \end{bmatrix}$$

6.2. $$V_{bus} = \begin{bmatrix} 1.0293\angle 1.46° \\ 1.0217\angle 0.99° \\ 1.0001\angle -0.015° \end{bmatrix}$$

6.3. (a) $x_1 = 5.0000 \quad x_2 = 1.0000$, $x_1 = 2.0006 \quad x_2 = 3.9994$

6.4. (a) 1, (b) 1, 4, 7, 9

6.5.
$$X^{(1)} = \begin{bmatrix} 6 \\ 5 \end{bmatrix} \quad X^{(2)} = \begin{bmatrix} 4.3929 \\ 4.9286 \end{bmatrix} \quad X^{(3)} = \begin{bmatrix} 4.0222 \\ 4.9964 \end{bmatrix} \quad X^{(4)} = \begin{bmatrix} 4.0001 \\ 5.0000 \end{bmatrix}$$

6.6. (a) $V_2^{(1)} = 0.9200 - j0.1000 \quad V_2^{(2)} = 0.9024 - j0.0981$
$V_2^{(3)} = 0.9005 - j0.1000 \quad V_2^{(4)} = 0.9001 - j0.1000$
(b) $S_{12} = 300$ MW $+ j100$ Mvar
$S_{21} = -280$ MW $- j60$ Mvar
$S_L = 20$ MW $+ j40$ Mvar

6.7. (a) $V_2^{(1)} = 0.9360 - j0.0800 \quad V_3^{(1)} = 0.9602 - j0.0460$
$V_2^{(2)} = 0.9089 - j0.0974 \quad V_3^{(2)} = 0.9522 - j0.0493$
(b) $S_{12} = 300$ MW $+ j300$ Mvar
$S_{21} = -300$ MW $- j240$ Mvar
$S_{L_{12}} = 0$ MW $+ j60$ Mvar
$S_{13} = 400$ MW $+ j400$ Mvar
$S_{31} = -400$ MW $- j360$ Mvar
$S_{L_{13}} = 0$ MW $+ j40$ Mvar
$S_{23} = -100$ MW $- j80$ Mvar
$S_{32} = 100$ MW $+ j90$ Mvar
$S_{L_{23}} = 0$ MW $+ j10$ Mvar
$S_1 = 700$ MW $+ j700$ Mvar

6.8. (a) $V_2^{(1)} = 1.0025 - j0.0500 \quad Q_3^{(1)} = 1.2360$
$V_3^{(1)} = 1.0299 + j0.0152$
$V_2^{(2)} = 1.0001 - j0.0409 \quad Q_3^{(2)} = 1.3671$
$V_3^{(2)} = 1.0298 + j0.0216$
(b) $S_{12} = 150.428$ MW $+ j100.159$ Mvar
$S_{21} = -150.428$ MW $- j92.387$ Mvar
$S_{L_{12}} = 0$ MW $+ j7.772$ Mvar
$S_{13} = -50.428$ MW $- j9.648$ Mvar
$S_{31} = 50.428$ MW $+ j10.902$ Mvar
$S_{L_{13}} = 0$ MW $+ j1.255$ Mvar
$S_{23} = -249.572$ MW $- j107.613$ Mvar
$S_{32} = 249.572$ MW $+ j126.034$ Mvar
$S_{L_{23}} = 0$ MW $+ j18.421$ Mvar
$S_1 = 100$ MW $+ j90.51$ Mvar

6.9. $Y_{bus} = \begin{bmatrix} -j125 & 0 & j100 & 0 \\ 0 & -j6.25 & 0 & j5 \\ j100 & 0 & -j89 & j9 \\ 0 & j5 & j9 & -j13 \end{bmatrix}$

6.10. $|V_2^{(1)}| = 0.9100 \quad \delta_2^{(1)} = -0.1300$ rad
$|V_2^{(2)}| = 0.8886 \quad \delta_2^{(2)} = -0.1464$ rad

6.11. $|V_2^{(1)}| = 0.8000 \quad \delta_2^{(1)} = -0.1000$ rad
$|V_2^{(2)}| = 0.7227 \quad \delta_2^{(2)} = -0.1350$ rad

6.12. The bus admittance matrix in polar form is

$$Y_{bus} = \begin{bmatrix} 60\angle -\frac{\pi}{2} & 40\angle \frac{\pi}{2} & 20\angle \frac{\pi}{2} \\ 40\angle \frac{\pi}{2} & 60\angle -\frac{\pi}{2} & 20\angle \frac{\pi}{2} \\ 20\angle \frac{\pi}{2} & 20\angle \frac{\pi}{2} & 40\angle -\frac{\pi}{2} \end{bmatrix}$$

(a) Substituting for the elements of the bus admittance matrix in (6.52) and (6.53) result in the given equations.
(b)

$\delta_2^{(1)} = 0.0275$ radian $= 1.5782^\circ \quad \delta_2^{(2)} = 0.0285$ radian $= 1.6327^\circ$
$\delta_3^{(1)} = -0.1078$ radian $= -6.179^\circ \quad \delta_3^{(2)} = -0.1189$ radian $= -6.816^\circ$
$|V_3^{(1)}| = 0.9231$ pu $\quad |V_3^{(2)}| = 0.9072$ pu
(c) The power flow program **lfnewton** is used to obtain the solution (See Example 6.9).

6.13. (a)
$\delta_2^{(1)} = 0.0262$ radian $= 1.5006^\circ \quad \delta_2^{(2)} = 0.0277$ radian $= 1.5863^\circ$
$\delta_3^{(1)} = -0.1119$ radian $= -6.412^\circ \quad \delta_3^{(2)} = -0.1182$ radian $= -6.772^\circ$
$|V_2^{(1)}| = 0.9250$ pu $\quad |V_3^{(2)}| = 0.9088$ pu
(b) The power flow program **decouple** is used to obtain the solution (See Example 6.11).

6.14. Follow the Instruction for Data Preparation (Section 6.9) and Example 6.9.

Chapter 7

7.1. A square of side length = 1.4142, perimeter = 5.6568

For $\lambda = -2.828$, $\frac{\partial^2 L}{\partial x^2} = \frac{\partial^2 L}{\partial y^2} = -5.6568$. Second derivatives are negative. Thus, objective function is maximized.

7.2. $x = y = -\frac{4}{3}, \lambda = \frac{8}{3}$

$\frac{\partial^2 L}{\partial x^2} = 2, \frac{\partial^2 L}{\partial y^2} = 4$. Second derivatives are positive. Thus, objective function is minimized.

7.3. Base = 1.732, Height = 1.5, Area = 1.299

7.4. $\zeta = 0.5, \omega_n = 10,000$ rad/sec, $M_{p\omega} = 1.1547$

7.5. Minimum value of the function = 12.5, at $x = 1.5, y = 3.2, \lambda = 1$

7.6. Minimum value of the function = 17, at $x = 1, y = 4, \lambda = \frac{14}{13}$

7.7. (a) $P_1 = 250$ MW, $P_2 = 300$ MW
(b) $P_1 = 500$ MW, $P_2 = 800$ MW
(c) $\beta = 6.8$, $\gamma = 0.002$

7.8. (i) $C_t = 4,849.75$ \$/h (ii) $C_t = 7,310.46$ \$/h (iii) $C_t = 12,783.04$ \$/h

7.9. (i) $P_1 = 100$ MW, $P_2 = 140$ MW, $P_3 = 210$ MW, $\lambda = 8.0$
$C_t = 4,828.70$ \$/h
(ii) $P_1 = 175$ MW, $P_2 = 260$ MW, $P_3 = 310$ MW, $\lambda = 8.6$
$C_t = 7,277.20$ \$/h
(iii) $P_1 = 325$ MW, $P_2 = 500$ MW, $P_3 = 510$ MW, $\lambda = 9.8$
$C_t = 12,705.20$ \$/h
(c) Savings: (i) 21.05 \$/h (ii) 33.26 \$/h (iii) 77.84 \$/h

7.10. (i) $P_1 = 122$ MW, $P_2 = 260$ MW, $P_3 = 68$ MW, $\lambda = 7.148$
$C_t = 4,927.13$ \$/h
(ii) $P_1 = 175$ MW, $P_2 = 260$ MW, $P_3 = 310$ MW, $\lambda = 8.6$
$C_t = 7,277.20$ \$/h
(iii) $P_1 = 350$ MW, $P_2 = 540$ MW, $P_3 = 445$ MW, $\lambda = 10$
$C_t = 12,724.38$ \$/h

7.11. $P_1 = 161.1765$ MW, $P_2 = 258.6003$ MW, $\lambda = 7.8038$ $C_t = 3,375.43$ \$/h

7.12. $P_1 = 70.360$ MW, $P_2 = 181.557$ MW, $P_7 = 97.111$ MW, $\lambda = 8.1513$
$C_t = 3,194.85$ \$/h

Chapter 8

8.1. (a) $\alpha = 75.75°$, $i(t) = 3\sin 315t$
(b) $\alpha = -14.25°$, $i(t) = 3\sin(315t - \pi/2) + 3e^{-80t}$
(c) In *MATLAB* using [Imax, k] = max(i), tmax= t(k) result in
imax = 4.37 A, tmax = 0.0096 sec.

8.2. In the file chp8ex2.m set $d = 30°$, rename the file and run the program.

8.3. In the file chp8ex3.m set $d = 30°$, rename the file and run the program.

8.5. In the file chp8ex4.m set $d = 30°$, rename the file and run the program.

8.6. (a) 0.6667 pu, 2.2222 pu, 4.0 pu
(b) $i_{ac}(t) = (2.5142e^{-25t} + 2.2e^{-0.7143t} + 0.9428)\sin\omega t$

8.7. $i_{asy}(t) = (2.5142e^{-25t} + 2.2e^{-0.7143t} + 0.9428)\sin(\omega t + \pi/2) + 5.6568e^{-3.3333t}$

8.8. $X'_d = 0.449$, $\tau'_d = 1.382$ sec, $X''_d = 0.2498$, $\tau''_d = 1.0397$ sec,

8.9. $i_{asy}(t) = (2.357e^{-25t} + 3.5355e^{-t} + 1.1785)\sin(\omega t + \pi/2) + 7.071e^{-4t}$
$I_{ac} = 5.0$ rms, $I_{dcmax} = 7.071$, $Iasy = 8.66$ rms

8.10. (a) $I''_d = 2.5$ pu, 7,216.88 A, 360.84 A
$I'_d = 2.0$ pu, 5,773.50 A, 288.68 A
$I_d = 0.6667$ pu, 1,924.5 A, 96.23 A
(b) $I_{asy} = 4.3333$ pu, 12,500 A, 625 A
(c) $i_{asy}(t) = (0.7071e^{-28.57t} + 1.8856e^{-2t} + 0.9428)\sin(\omega t + \pi/2) + 3.5355e^{-3.333t}$

8.11. $I'_g = 2.56\angle{-75.53°}$ pu, or $7393.69\angle{-75.53°}$ A

8.12. $I'_g = 3.545\angle{-78.6°}$ pu, $I'_m = 3.599\angle{-95.3°}$ pu, $I'_f = 7.068\angle{-87.03°}$ pu

Chapter 9

9.1. $2.0\angle{-90°}$ pu = $288.675\angle{-90°}$ A, 200 MVA

9.2. 1.8 Ω

9.3. (a) $j0.2$ pu, $5.0\angle{-90°}$ pu, (b) $V_1 = 0.4$ pu, $V_2 = 0.8$ pu, $V_3 = 0.7$ pu

9.4. (a) $j0.4$ pu, $2.5\angle{-90°}$ pu
(b) $V_1 = 0.925$ pu, $V_2 = 0.925$ pu, $V_3 = 0.475$ pu
$I_{12} = 0$ pu, $I_{13} = 1.5\angle{-90°}$ pu, $I_{23} = 1.0\angle{-90°}$ pu

9.5. (a) $j0.5$ pu, $2.0\angle{-90°}$ pu
(b) $V_1 = 0.60$ pu, $V_2 = 0.65$ pu, $V_3 = 0.38$ pu, $V_4 = 0$ pu
$I_{13} = 1.1\angle{-90°}$ pu, $I_{21} = 0.1\angle{-90°}$ pu, $I_{23} = 0.9\angle{-90°}$ pu
$I_{34} = 2.0\angle{-90°}$ pu
(c): (a) $j0.25$ pu, $4.0\angle{-90°}$ pu
(b) $V_1 = 0.44$ pu, $V_2 = 0.09$ pu, $V_3 = 0.3$ pu, $V_4 = 0.3$ pu
$I_{12} = 0.7\angle{-90°}$ pu, $I_{13} = 0.7\angle{-90°}$ pu, $I_{32} = 0.7\angle{-90°}$ pu

9.6. $Z_{bus} = \begin{bmatrix} j0.16 & j0.06 & j0.16 \\ j0.06 & j0.21 & j0.06 \\ j0.16 & j0.06 & j0.31 \end{bmatrix}$

9.7. $Z_{bus} = \begin{bmatrix} j0.12 & j0.04 & j0.06 \\ j0.04 & j0.08 & j0.02 \\ j0.06 & j0.02 & j0.08 \end{bmatrix}$

9.8. $Z_{bus} = \begin{bmatrix} j0.0450 & j0.00750 & j0.0300 \\ j0.0075 & j0.06375 & j0.0300 \\ j0.0300 & j0.03000 & j0.2100 \end{bmatrix}$

9.9. $Z_{bus} = \begin{bmatrix} j0.32 & j0.16 & j0.28 \\ j0.16 & j0.48 & j0.24 \\ j0.28 & j0.24 & j0.42 \end{bmatrix}$

9.10. Same as Problem 9.4

9.11. Same as Problem 9.5

9.12. $4.0\angle -90°$ pu
$V_1 = 0.46$ pu, $V_2 = 0.61$ pu, $V_3 = 0.16$ pu, $V_4 = 0.01$ pu
$I_{13} = 1.5\angle -90°$ pu, $I_{14} = 1.5\angle -90°$ pu, $I_{21} = 0.3\angle -90°$ pu,
$I_{24} = 1.0\angle -90°$ pu, $I_{34} = 1.5\angle -90°$ pu

9.13. Run chp9ex7 for a bolted fault at bus 9.

9.14. Run chp9ex8 for a bolted fault at bus 9.

9.15. Run chp9ex9 for a bolted fault at bus 9.

9.16–9.18. Make data similar to Examples 9.7–9.9.

Chapter 10

10.1. $V_a^0 = 42.265\angle -120°$, $V_a^1 = 193.185\angle -135°$, $V_a^2 = 86.947\angle -84.896°$

10.2. $I_a = 8.185\angle 42.216°$, $I_b = 4.0\angle -30°$, $I_c = 8.185\angle -102.216°$

10.4.

$$V_L^{012} = \begin{bmatrix} 0 \\ 763.763\angle -10.93° \\ 288.675\angle 30° \end{bmatrix} \quad V_{an}^{012} = \begin{bmatrix} 0 \\ 440.958\angle -40.89° \\ 166.667\angle 60° \end{bmatrix}$$

$$V^{abc} = \begin{bmatrix} 440.958\angle -19.106° \\ 600.925\angle -166.102° \\ 333.333\angle 60° \end{bmatrix}$$

10.5. $I_a^{012} = \begin{bmatrix} 20\angle 90° \\ 60\angle -90° \\ 40\angle 90° \end{bmatrix}$

10.6. $I^{abc} = \begin{bmatrix} 20\angle -90° \\ 20\angle 150° \\ 20\angle 30° \end{bmatrix}$

10.7. (a)

$$Z^{012} = \begin{bmatrix} 10 + j50 & 0 & 0 \\ 0 & 10 + j35 & 0 \end{bmatrix} \quad (b) \quad V_a^{012} = \begin{bmatrix} 42.265\angle -120° \\ 193.185\angle -135° \\ 86.947\angle -84.896° \end{bmatrix}$$

$$(c) \quad I_a^{012} = \begin{bmatrix} 0.829\angle 161.31° \\ 5.307\angle 150.95° \\ 2.388\angle -158.95° \end{bmatrix} \quad (d) \quad I^{abc} = \begin{bmatrix} 7.907\angle 165.46° \\ 5.819\angle 14.867° \\ 2.701\angle -96.93° \end{bmatrix}$$

(e) $S_{3\phi} = 1,036.8 + j3,659.6$

(f) Same as (e)

10.8.

$$V_{an}^{012} = \begin{bmatrix} 0 \\ 136.879\angle 139.933° \\ 451.105\angle 54.603° \end{bmatrix} \quad V^{abc} = \begin{bmatrix} 480.754\angle 70.560° \\ 333.338\angle 163.741° \\ 569.611\angle -73.685° \end{bmatrix}$$

$$I^{abc} = \begin{bmatrix} 12.993\angle 70.561° \\ 900.9\angle 163.741° \\ 15.395\angle -73.686° \end{bmatrix}$$

10.9. 1.8Ω

10.10. 0.825Ω

10.11. $I_a = 12\angle -90° pu$

10.12. $I_b = -9.116$ pu

10.13. $I_f = I_c + I_b = 12.5\angle 90°$ pu

10.14. (a) $5\angle -90°$ pu, (b) $6\angle -90°$ pu, (c) $-4.33\angle -90°$ pu, (d) $7.5\angle 90°$ pu

10.15. (a) $4.395\angle -90°$ pu, (b) $4.669\angle -90°$ pu, (c) $3.807\angle 180°$ pu, (d) $4.979\angle 90°$ pu

10.16. $I_f = 4.6693\angle -90°$ pu

Bus	Voltage Magnitude		
No.	Phase a	Phase b	Phase c
1	0.0000	0.9704	0.9704
2	0.5214	0.9567	0.9567
3	0.7977	0.9535	0.9535
4	0.8911	0.9739	0.9739

From Bus	To Bus	Line current magnitude		
		Phase a	Phase b	Phase c
1	F	4.6693	0.0000	0.0000
2	1	1.4786	0.1556	0.1556
3	1	2.0234	1.0117	1.0117
4	2	1.0895	0.5447	0.5447

10.17. $I_f = -3.8067$ pu

Bus	Voltage Magnitude		
No.	Phase a	Phase b	Phase c
1	1.0000	0.5000	0.5000
2	1.0000	0.6401	0.6401
3	1.0000	0.7954	0.7954
4	1.0000	0.8871	0.8871

From Bus	To Bus	Line current magnitude		
		Phase a	Phase b	Phase c
1	F	0.0000	3.8067	3.8067
2	1	0.0000	1.3323	1.3323
3	1	0.0000	2.4744	2.4744
4	2	0.0000	1.3323	1.3323

10.18. $I_f = 4.9793\angle 90^\circ$ pu

Bus No.	Voltage Magnitude		
	Phase a	Phase b	Phase c
1	0.9336	0.0000	0.0000
2	0.9004	0.4965	0.4965
3	0.8921	0.7626	0.7626
4	0.9419	0.8711	0.8711

From Bus	To Bus	Line current magnitude		
		Phase a	Phase b	Phase c
1	F	0.0000	4.5486	4.5485
2	1	0.1660	1.5076	1.5076
3	1	1.0788	2.5325	2.5325
4	2	0.5809	1.3636	1.3636

10.19.

$$Z_{bus}^{(1)} = \begin{bmatrix} 0.120 & 0.040 & 0.030 & 0.020 \\ 0.040 & 0.080 & 0.010 & 0.040 \\ 0.030 & 0.010 & 0.045 & 0.005 \\ 0.020 & 0.04070.005 & 0.045 & \end{bmatrix}$$

Total fault current = 10.8253 per unit

Bus No.	Voltage Magnitude		
	Phase a	Phase b	Phase c
1	1.0000	0.6614	0.6614
2	1.0000	0.5000	0.5000
3	1.0000	0.9079	0.9079
4	1.0000	0.6614	0.6614

From Bus	To Bus	Line current magnitude		
		Phase a	Phase b	Phase c
1	2	0.0000	0.7217	0.7217
1	2	0.0000	1.4434	1.4434
2	F	0.0000	10.8253	10.8253
3	1	0.0000	2.1651	2.1651
4	2	0.0000	8.6603	8.6603

10.20 Run chp10ex8 for a bolted fault at bus 9.

10.21. Make data similar to Example 10.8.

Chapter 11

11.1. $M = 8.5$ MJ.rad/sec, $H = 4.0$ MJ/MVA

11.2. $600^\circ/\text{sec}^2 = 100\text{rpm/sec}$, 20°, 3620 rpm

11.3. $376^\circ/\text{sec}^2 = 62.667\text{rpm/sec}$, 4.23°, 3609.4 rpm

11.5. $H = 2.4$ MJ/MVA, $P_m = 0.5$ pu, $P_{max} = 2$,
$\frac{d^2\delta}{dt^2} = 4500(0.5 - 2\sin\delta)$, (δ is in degrees)

11.6. $E' = 1.25\angle 27.819^\circ$, $0.03\frac{d^2\delta}{dt^2} = 0.77 - 1.65\sin\delta)$, (δ is in radians)

11.7. $0.03\frac{d^2\delta}{dt^2} = 0.77 - 0.5\sin\delta)$, (δ is in radians)

11.8. $\frac{H}{\pi f_0}\frac{d^2\Delta\delta}{dt^2} + P_s\Delta\delta = 0$, where $P_s = \frac{dP}{d\delta}\Big|_{\delta_0} = P_{max}\cos\delta_0 + 2P_k\cos 2\delta_0$

11.9. $\zeta = 0.6$, $\omega_d = 4.0$ rad/sec

11.10. $\delta = 27.835^\circ + 16.0675e^{-2.4977t}\sin(6.5059t + 69^\circ)$
$f = 60 - 0.311e^{-2.4977t}\sin 6.5059t$ Hz

11.11.

$$A = \begin{bmatrix} 0 & 1 \\ -48.5649 & -4.9955 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

11.12. $\delta = 27.835^\circ + 5.8935[1 - 1.0712e^{-2.4977t}\sin(6.5059t + 69^\circ)$
$f = 60 + 0.1222e^{-2.4977t}\sin 6.5059t$ Hz

11.13. A, B, D are the same as in Problem 11.11, $B = \begin{bmatrix} 0 \\ 4.9955 \end{bmatrix}$

11.14. (a) 0.649 pu, (b) 1.195 pu

11.15. (a) $\delta_c = 82.593^\circ$, $t_c = 0.273$ sec (b) $\delta_c = 77.82^\circ$

11.16. $t_c = 0.37$ sec

11.17. (a) Stable (b) Unstable (c) $t_c = 0.29$ sec

11.18. (a) Stable (b) Unstable (c) $t_c = 0.72$ sec

Chapter 12

12.1. 25 MW

12.2. $P_1 = 200$ MW, $P_2 = 300$ MW

12.3. (a) $\Delta f = -0.3$ Hz, $P_1 = 250$ MW, $P_2 = 580$ MW

(b) $\Delta f = -0.291$ Hz, $P_1 = 248.5002$ MW, $P_2 = 577.6004$ MW, $\Delta P_L = -3.8994$

12.4. (a) $R > 0.009678$

(b) Use **rlocus** for $KG(s)H(s) = \frac{k}{2s^3+12.2s^2+17.2s+1.6}$, where $K = \frac{1}{R}$

12.5. (a) -0.5563 Hz

(b) $T(s) = \frac{0.0625s^2+0.375s+0.5}{s^3+6.1s^2+8.6s+13.3}$, use **step** function with value of -0.25 pu.

(c) Simulation response same as the response in (b)

12.6. (b) $T(s) = \frac{0.0625s^3+0.375s^2+0.5s}{s^4+6.1s^3+8.6s^2+13.3s+0.5K_I}$, use **step** function with value of -0.25

12.7. Modify sim12ex4.mdl

12.8. Modify sim12ex5.mdl

12.9. (a) $K_A < 43.3125$

(b) b) Use **rlocus** for $KG(s)H(s) = \frac{32k_A}{s^3+23s^2+62s+1320}$

(c) $T(s) = \frac{1280}{s^3+23s^2+62s+1320}$, use **step** to obtain the response

(d) Simulation response same as the response in (c)

12.10. (a) $T(s) = \frac{1280(s+25)}{s^4+48s^32+637s^2+6870s+37000}$, use **step** to obtain the response

(b) Simulation response same as the response in (a)

12.12. (b) 0, 0, ± 4.5388, (c) Unstable

(d) $K = \begin{bmatrix} -170.3666 & -38.0540 & -17.3293 & -24.1081 \end{bmatrix}$

(d) $A_f = \begin{bmatrix} 0 & 1.0000 & 0 & 0 \\ -64.5823 & -19.0270 & -8.6646 & -12.0540 \\ 0 & 0 & 0 & 1.0000 \\ 42.1012 & 9.5135 & 4.3323 & 6.0270 \end{bmatrix}$

12.14. (a) $K = \begin{bmatrix} -125.0988 & -28.6369 & -5.0000 & -10.5129 \end{bmatrix}$

(b) $A_f = \begin{bmatrix} 0 & 1.0000 & 0 & 0 \\ -41.9484 & -14.3185 & -2.5000 & -5.2565 \\ 0 & 0 & 0 & 1.0000 \\ 30.7842 & 7.1592 & 1.2500 & 2.6282 \end{bmatrix}$

12.16. (a) $A = \begin{bmatrix} -4 & 0 & -100 \\ 2 & -2 & 0 \\ 0 & 0.0625 & -0.1 \end{bmatrix}$

$B = \begin{bmatrix} 0 & 0 & -0.0625 \end{bmatrix}, BPL = B * PL, C = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}, D = 0$

(c) $K = \begin{bmatrix} 6.4 & 5.4 & -94.4 \end{bmatrix}$ $A-BK = \begin{bmatrix} -4.0 & 0.0 & -100 \\ 2.0 & -2.0 & 0 \\ 0.4 & 0.4 & -6 \end{bmatrix}$

12.17. (a) $K = \begin{bmatrix} 8.3477 & 1.4637 & -162.0007 \end{bmatrix}$

$A_f = \begin{bmatrix} -4.0000 & 0 & -100.0000 \\ 2.0000 & -2.0000 & 0 \\ 0.5217 & 0.1540 & -10.2250 \end{bmatrix}$

INDEX