Appendix B

MATLAB Tutorial

B.1 Introduction

MATLAB is a high-level technical computing environment suitable for solving scientific and engineering problems. When used with routines from its companion software, the Control System Toolbox, MATLAB can be used to analyze and design control systems problems such as those covered in this textbook. MATLAB and the Control System Toolbox are commercial software products available from MathWorks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2098. Phone: (508) 647-7000. Email: info@mathworks.com. URL: www.mathworks.com.

The MATLAB examples in this tutorial consist of solved problems that demonstrate the application of MATLAB to the analysis and design of control systems. Many problems were taken from examples in the text (identified with a **MATLAB** icon) that were solved without MATLAB. A Command Summary at the end of this appendix lists key MATLAB statements and their descriptions.

The code in this tutorial is also available in the Control Systems Engineering Toolbox folder. You should have MATLAB Version 9.3(R2017b) and the Control System Toolbox Version 10.3 installed on your machine to execute this appendix's code in the Control Systems Engineering Toolbox Version 8.

To run the M-files, first be sure the files are either added to the search path in **Set Path** under the **HOME** tab in the **ENVIRONMENT** section or appear in the **Current Folder** window, which is part of the **MATLAB** window. To see the computer responses after installing the M-files, run each problem by typing the M-file name, such as ch2p1, after the prompt (>>) in the **Command Window.** You may also run the files by right-clicking the file name, if it appears in the **Current Folder** window, and select **Run.**

To view all or part of the M-file in the **Command Window,** enter "type <file name>" or "help <file name>," respectively, after the prompt. You may also view and make changes to the M-file by double-clicking the file in the **Current Folder** window. This action brings up the editor. After editing, be sure to save the revised file before executing.

If you do not have the Control Systems Engineering Toolbox M-files, you can create your own M-files by typing the code for each problem in this appendix into a separate M-file (there is no need to type the final pause statement or comments), and naming each M-file with a .m extension, as in ch2p1.m. You can also type the code for more than one problem into an M-file, including the pause command, and name the M-file with the .m extension. You can then call the file from the **Command Window,** and continue past the pause statements to the next problem by pressing any key.

By its nature, this appendix cannot cover all the background and details necessary for a complete understanding of MATLAB. For further details, you are referred to other sources, including MATLAB reference manuals and instructions specific to your particular computer. The bibliography at the end of this appendix provides a partial listing of references. This appendix should give you enough information to be able to apply MATLAB to the analysis and design problems covered in this book.

The code will also run on workstations that support MATLAB. Consult the MATLAB *Installation Guide* for your platform for minimum system hardware requirements.

B.2 MATLAB Examples

Chapter 2: Modeling in the Frequency Domain

ch2apB1 Bit strings will be used to identify parts of this tutorial on the computer output. Bit strings are represented by the text enclosed in apostrophes, such as `'ab'`. Comments begin with `%` and are ignored by MATLAB. Numbers are entered without any other characters. Arithmetic can be performed using the proper arithmetic operator. Numbers can be assigned using a left-hand argument and an equals sign. Finally, we can find the magnitude and angle of a complex number, Q using `abs` (Q) and `angle` (Q), respectively.

```
'(ch2apB1)'                          % Display label.
' How are you?'                      % Display string.
-3.96                                % Display scalar number -3.96.
-4 + 7i                              % Display complex number -4+7i.
-5-6j                                % Display complex number -5-6 j.
(-4+7i)+(-5-6i)                      % Add two complex numbers and
                                     % display sum.
(-4+7j)*(-5-6j)                      % Multiply two complex numbers and
                                     % display product.
M=5                                  % Assign 5 to M and display.
N=6                                  % Assign 6 to N and display.
P=M+N                                % Assign M+N to P and display.
Q=3+4j                               % Define complex number, Q.
MagQ=abs(Q)                          % Find magnitude of Q.
ThetaQ=(180/pi)*angle(Q)             % Find the angle of Q in degrees.
pause
```

ch2apB2 Polynomials in s can be represented as row vectors containing the coefficients. Thus $P_1 = s^3 + 7s^2 - 3s + 23$ can be represented by the vector shown below with elements separated by a space or comma. Bit strings can be used to identify each section of this tutorial.

```
'(ch2apB2)'                          % Display label.
P1=[1 7 -3 23]                       % Store polynomial s^3 + 7s^2 -3s+
                                     % 23 as P1 and display.
pause
```

ch2apB3 Running the previous statements causes MATLAB to display the results. Ending the command with a semicolon suppresses the display. Typing an expression without a left-hand assignment and without a semicolon causes the expression to be evaluated and the result displayed. Enter `P2` in the **MATLAB Command Window** after execution.

```
'(ch2apB3)'                          % Display label.
P2=[3 5 7 8] ;                       % Assign 3s^3 + 5s^2 +7s + 8 to P2
                                     % without displaying.
3*5                                  % Evaluate 3*5 and display result.
pause
```

ch2apB4 An $F(s)$ in factored form can be represented in polynomial form. Thus $P_3 = (s+2)(s+5)(s+6)$ can be transformed into a polynomial using `poly (V)`, where `V` is a row vector containing the roots of the polynomial and `poly(V)` forms the coefficients of the polynomial.

```
'(ch2apB4)'                          % Display label.
P3=poly([-2 -5 -6])                  % Store polynomial
                                     % (s+2)(s+5)(s+6) as P3 and
                                     % display the coefficients.
pause
```

ch2apB5 We can find roots of polynomials using the `roots (V)` command. The roots are returned as a column vector. For example, find the roots of $5s^4 + 7s^3 + 9s^2 - 3s + 2 = 0$.

```
'(ch2apB5)'                          % Display label.
P4=[5 7 9 -3 2]                      % Form 5s^4+7s^3+9s^2-3s+2 and
                                     % display.
rootsP4=roots(P4)                    % Find roots of 5s^4+7s^3+9s^2
                                     %-3s+2,
                                     % assign to rootsP4, and display.
pause
```

ch2apB6 Polynomials can be multiplied together using the `conv(a,b)` command (standing for convolve). Thus, $P_5 = (s^3 + 7s^2 + 10s + 9)(s^4 - 3s^3 + 6s^2 + 2s + 1)$ is generated as follows:

```
'(ch2apB6)'                          % Display label.
P5=conv([1 7 10 9],[1 -3 6 2 1])     % Form (s^3+7s^2+10s+9)(s^4-
                                     % 3s^3+6s^2+2s+1), assign to P5,
                                     % and display.
pause
```

ch2apB7 The partial-fraction expansion for $F(s) = b(s)/a(s)$ can be found using the `[K, p, k ]= residue (b, a)` command (`K` = residue; `p` = roots of denominator; `k` = direct quotient, which is found by dividing polynomials prior to performing a partial-fraction expansion). We expand $F(s) = (7s^2 + 9s + 12)/[s(s + 7)(s^2 + 10s + 100)]$ as an example. Using the results from MATLAB yields: $F(s) = [(0.2554 - 0.3382i)/(s+5.0000-8.6603i)]+[(0.2554+0.3382i)/(s+5.0000+8.6603i)]-[0.5280/(s+7)]+[0.0171/s]$.

```
'(ch2apB7)'                          % Display label.
numf=[7 9 12];                       % Define numerator of F(s).
denf=conv(poly([0 -7]),[1 10 100]);  % Define denominator of F(s).
[K,p,k]=residue(numf,denf)           % Find residues and assign to K;
                                     % find roots of denominator and
                                     % assign to p; find
                                     % constant and assign to k.
pause
```

ch2apB8 (Example 2.3) Let us do Example 2.3 in the book using MATLAB.

```
'(ch2apB8) Example 2.3'              % Display label.
numy=32;                             % Define numerator.
deny=poly([0 -4 -8]);                % Define denominator.
[r,p,k]=residue(numy,deny)           % Calculate residues, poles, and
                                     % direct quotient.
pause
```

ch2apB9 Creating Transfer Functions

Vector Method, Polynomial Form A transfer function can be expressed as a numerator polynomial divided by a denominator polynomial, that is, $F(s) = N(s)/D(s)$. The numerator, $N(s)$, is represented by a row vector, `numf`, that contains the coefficients of $N(s)$. Similarly, the denominator, $D(s)$, is represented by a row vector, `denf`, that contains the coefficients of $D(s)$. We form $F(s)$ with the command, `F=tf(numf,denf)`. `F` is called a linear time-invariant (LTI) object. This object, or transfer function, can be used as an entity in other operations, such as addition or multiplication. We demonstrate with $F(s) = 150(s^2 + 2s + 7)/[s(s^2 + 5s + 4)]$. Notice after executing the `tf` command, MATLAB prints the transfer function.

Vector Method, Factored Form We also can create LTI transfer functions if the numerator and denominator are expressed in factored form. We do this by using row vectors containing the roots of the numerator and denominator. Thus $G(s) = K^*N(s)/D(s)$ can be expressed as an LTI object using the command, `G=zpk(numg,deng,K)`, where `numg` is a row vector containing the roots of $N(s)$ and `deng` is a row vector containing the roots of $D(s)$. The expression `zpk` stands for zeros (roots of the numerator), poles (roots of the denominator), and gain, K. We demonstrate with $G(s) = 20(s + 2)(s + 4)/[(s + 7)(s + 8)(s + 9)]$. Notice after executing the `zpk` command, MATLAB prints the transfer function.

Rational Expression in s Method, Polynomial Form (Requires Control System Toolbox 8.4) This method allows you to type the transfer function as you normally would write it. The statement `s=tf('s')` must precede the transfer function if you wish to create an LTI transfer function in polynomial form equivalent to using `F=tf(numf, denf)`.

Rational Expression in s Method, Factored Form (Requires Control System Toolbox 8.4) This method allows you to type the transfer function as you normally would write it. The statement `s=zpk('s')` must precede the transfer function if you wish to create an LTI transfer function in factored form equivalent to using `G=zpk(numg,deng,K)`.

For both rational expression methods the transfer function can be typed in any form regardless of whether `s=tf('s')` or `s=zpk('s')` is used. The difference is in the created LTI transfer function. We use the same examples above to demonstrate the rational expression in s methods.

```
'(ch2apB9)'                          % Display label.
' Vector Method, Polynomial Form'   % Display label.
numf=150*[1 2 7]                     % Store 150(s^2+2s+7) in numf and
                                     % display.
denf=[1 5 4 0]                       % Store s(s+1)(s+4) in denf and
                                     % display.
' F(s)'                              % Display label.
F=tf(numf,denf)                      % Form F(s) and display.
clear                                % Clear previous variables from
                                     % workspace.
```

```
' Vector Method, Factored Form'      % Display label.
numg=[-2 -4]                        % Store (s+2)(s+4) in numg and
                                    % display.
deng=[-7 -8 -9]                     % Store (s+7)(s+8)(s+9) in deng
                                    % and display.
K=20                                % Define K.
' G(s)'                             % Display label.
G=zpk(numg,deng,K)                  % Form G(s) and display.
clear                               % Clear previous variables from
                                    % workspace.
' Rational Expression Method, Polynomial Form'
                                    % Display label.
s=tf('s')                           % Define 's' as an LTI object in
                                    % polynomial form.
F=150*(s^2+2*s+7)/[s*(s^2+...       % Form F (s) as an LTI transfer
 5*s+4)]                            % function in polynomial form.
G=20*(s+2)*(s+4)/[(s+7)*...         % Form G(s) as an LTI transfer
(s+8)*(s+9)]                        % function in polynomial form.
clear                               % Clear previous variables from
                                    % workspace.
' Rational Expression Method, Factored Form'
                                    % Display label.
s=zpk('s')                          % Define 's' as an LTI object in
                                    % factored form.
F=150*(s^2+2 * s+7)/[s *(s^2+5*s+4)]
                                    % Form F(s) as an LTI transfer
                                    % function in factored form.
G=20*(s+2)*(s+4)/[(s+7)*(s+8)*(s+9)]
                                    % Form G(s) as an LTI transfer
                                    % function in factored form.
pause
```

ch2apB10 Transfer function numerator and denominator vectors can be converted between polynomial form containing the coefficients and factored form containing the roots. The MATLAB function, `tf2zp(numtf,dentf)`, converts the numerator and denominator from coefficients to roots. The results are in the form of column vectors. We demonstrate this with $F(s) = (10s^2 + 40s + 60)/(s^3 + 4s^2 + 5s + 7)$. The MATLAB function, `zp2tf(numzp,denzp,K)`, converts the numerator and denominator from roots to coefficients. The arguments `numzp` and `denzp` must be column vectors. In the demonstration that follows, apostrophes signify transpose. We demonstrate the conversion from roots to coefficients with $G(s) = 10(s + 2)\ (s + 4)/[s(s + 3)(s + 5)]$.

```
' (ch2apB10)'                       % Display label.
' Coefficients for F(s)'            % Display label.
numftf=[10 40 60]                   % Form numerator of F(s)=
                                    % (10s^2+40s+60)/(s^3+4s^2+5s
                                    % +7).
denftf=[1 4 5 7]                    % Form denominator of F(s)=
                                    % (10s^2+40s+60)/(s^3+4s^2+5s
                                    % +7).
' Roots for F(s)'                   % Display label.
[numfzp,denfzp]=tf2zp(numftf,denftf)
                                    % Convert F(s) to factored form.
' Roots for G(s)'                   % Display label.
numgzp=[-2 -4]                      % Form numerator of
```

```
K=10                                    % G(s)=10(s+2)(s+4)/[s(s + 3)
                                        % (s+5)].
dengzp=[0 -3 -5]                        % Form denominator of
                                        % G(s)=10(s+2)(s+4)/[s(s+3)(s+5)].
'Coefficients for G(s)'                 % Display label.
[numgtf,dengtf]=zp2tf(numgzp',dengzp',K)
                                        % Convert G(s) to polynomial form.
pause
```

ch2apB11 LTI models can also be converted between polynomial and factored forms. MATLAB commands `tf` and `zpk` are also used for the conversion between LTI models. If a transfer function, *Fzpk*(*s*), is expressed as factors in the numerator and denominator, then `tf(Fzpk)` converts *Fzpk*(*s*) to a transfer function expressed as coefficients in the numerator and denominator. Similarly, if a transfer function, *Ftf*(*s*), is expressed as coefficients in the numerator and denominator, then `zpk(Ftf)` converts *Ftf*(*s*) to a transfer function expressed as factors in the numerator and denominator. The following example demonstrates the concepts.

```
'(ch2apB11)'                            % Display label.
'Fzpk1(s)'                              % Display label.
Fzpk1=zpk([-2 -4],[0 -3 -5],10)         % Form Fzpk1(s)=
                                        % 10(s+2)(s+4)/[s(s+3)(s+5)].
'Ftf1'                                  % Display label.
Ftf1=tf(Fzpk1)                          % Convert Fzpk1(s) to
                                        % coefficients form.
'Ftf2'                                  % Display label.
Ftf2=tf([10 40 60],[1 4 5 7])           % Form Ftf2(s)=
                                        % (10s^2+40s+60)/(s^3+4s^2+5s
                                        % +7).
'Fzpk2'                                 % Display label.
Fzpk2=zpk(Ftf2)                         % Convert Ftf2(s) to
                                        % factored form.
pause
```

ch2apB12 Functions of time can be easily plotted using MATLAB's `plot(X,Y,S)`, where `X` is the independent variable, `Y` is the dependent variable, and `S` is a character string describing the plot's color, marker, and line characteristic. Type `HELP PLOT` in the **Command Window** to see a list of choices for `S`. Multiple plots also can be obtained using `plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,. . .)`. In the following example we plot on the same graph sin(5*t*) in red and cos(5*t*) in green for $t = 0$ to 10 seconds in 0.01 second increments. Time is specified as `t=start: increment: final`.

```
'(ch2apB12)'                            % Display label.
t=0:0.01:10;                            % Specify time range and increment.
f1=cos(5*t);                            % Specify f1 to be cos(5t).
f2=sin(5*t);                            % Specify f2 to be sin(5t).
plot(t,f1,'r',t,f2,'g')                 % Plot f1 in red and f2 in green.
pause
```

Chapter 3: Modeling in the Time Domain

ch3apB1 The square system matrix, $\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -9 & -8 & -7 \end{bmatrix}$ is written with a space or

comma separating the elements of each row. The next row is indicated with a semicolon or carriage return. The entire matrix is then enclosed in a pair of square brackets.

```
'(ch3apB1)'                          % Display label.
A=[0 1 0;0 0 1; -9 -8 -7]            % Represent A.
'or'
A=[0 1 0                             % Represent A.
0 0 1
-9 -8 -7]
pause
```

ch3apB2 A row vector, such as the output matrix **C**, can be represented with elements separated by spaces or commas and enclosed in square brackets. A column vector, such as input matrix **B**, can be written as elements separated by semicolons or carriage returns, or as the transpose (') of a row vector.

```
'(ch3apB2)'                          % Display label.
C=[2 3 4]                            % Represent row vector C.
B=[7;8;9]                            % Represent column vector B.
'or'
B=[7                                 % Represent column vector B.
8
9]
'or'
B=[7 8 9]'                           % Represent column vector B.
pause
```

ch3apB3 The state-space representation consists of specifying the **A**, **B**, **C**, and **D** matrices followed by the creation of an LTI state-space object using the MATLAB command, `ss(A,B,C,D)`. Hence, for the matrices in (ch3p1) and (ch3p2), the state-space representation would be:

```
'(ch3apB3)'                          % Display label.
A=[0 1 0;0 0 1;-9 -8 -7];            % Represent A.
B=[7;8;9] ;                          % Represent column vector B.
C=[2 3 4] ;                          % Represent row vector C.
D=0;                                 % Represent D.
F=ss(A,B,C,D)                        % Create an LTI object and display.
```

ch3apB4 (Example 3.4) Transfer functions represented either by numerator and denominator or an LTI object can be converted to state space. For numerator and denominator representation, the conversion can be implemented using `[A, B, C, D]=tf2ss(num, den)`. The **A** matrix is returned in a form called the controller canonical form, which will be explained in Chapter 5 in the text. To obtain the phase-variable form, `[Ap,Bp,Cp,Dp]`, we perform the following operations: `Ap=inv(P)*A*P;Bp=inv(P)*B;Cp=C*P,Dp=D`, where **P** is a matrix with 1's along the anti-diagonal and 0's elsewhere. These transformations will be explained in Chapter 5. The command `inv (X)` finds the inverse of a square matrix. The symbol * signifies multiplication. For systems represented as LTI objects, the command `ss(F)`, where `F` is an LTI transfer-function object, can be used to convert `F` to a state-space object. Let us look at Example 3.4 in the text. For the numerator–denominator representation, notice that the MATLAB response associates the gain, 24, with the vector **C** rather than the vector **B** as in the example in the text. Both representations are equivalent. For the LTI transfer-function object, the conversion to state space does not yield the phase-variable form. The result

is a balanced model that improves the accuracy of calculating eigenvalues, which are covered in Chapter 4. Since `ss(F)` does not yield familiar forms of the state equations (nor is it possible to easily convert to familiar forms), we will have limited use for that transformation at this time.

```
'(ch3apB4) Example 3.4'              % Display label.
'Numerator-denominator representation conversion'
                                     % Display label.
'Controller canonical form'          % Display label.
num=24;                              % Define numerator of
                                     % G(s)=C(s)/R(s).
den=[1 9 26 24];                     % Define denominator of G(s).
[A,B,C,D]=tf2ss(num,den)             % Convert G(s) to controller
                                     % canonical form, store matrices
                                     % A, B, C, D, and display.
'Phase-variable form'                % Display label.
P=[0 0 1;0 1 0;1 0 0];               % Form transformation matrix.
Ap=inv(P)*A*P                        % Form A matrix, phase-variable
                                     % form.
Bp=inv(P)*B                          % Form B vector, phase-variable
                                     % form.
Cp=C*P                               % Form C vector, phase-variable
                                     % form.
Dp=D                                 % Form D phase-variable form.
'LTI object representation'          % Display label.
T=tf(num,den)                        % Represent T(s)=24/(s^3+9s^2 +
                                     % 26s+24) as an LTI transfer-
                                     % function object.
Tss=ss(T)                            % Convert T(s) to state space.
pause
```

ch3apB5 State-space representations can be converted to transfer functions represented by a numerator and a denominator using `[num,den]=ss2tf(A,B,C,D,iu)`, where `iu` is the input number for multiple-input systems. For single-input, single-output systems `iu=1`. For an LTI state-space system, `Tss`, the conversion can be implemented using `Ttf=tf(Tss)` to yield the transfer function in polynomial form or `Tzpk=zpk(Tss)` to yield the transfer function in factored form. For example, the transfer function represented by the matrices described in (ch3p3) can be found as follows:

```
'(ch3apB5)'                          % Display label.
       'Non LTI'                     % Display label.
A=[0 1 0;0 0 1;-9 -8 -7];            % Represent A.
B=[7;8;9];                           % Represent B.
C=[2 3 4];                           % Represent C.
D=0;                                 % Represent D.
'Ttf(s)'                             % Display label.
[num,den]=ss2tf(A,B,C,D,1)           % Convert state-space
                                     % representation to a
                                     % transfer function represented as
                                     % a numerator and denominator in
                                     % polynomial form, G(s)=num/den,
                                     % and display num and den.
       'LTI'                         % Display label.
Tss=ss(A,B,C,D)                      % Form LTI state-space model.
'Polynomial form, Ttf(s)'            % Display label.
```

```
Ttf=tf(Tss)                          % Transform from state space to
                                     % transfer function in polynomial
                                     % form.
'Factored form, Tzpk(s)'             % Display label.
Tzpk=zpk(Tss)                        % Transform from state space to
                                     % transfer function in factored
                                     % form.
pause
```

Chapter 4: Time Response

ch4apB1 (Example 4.6) We can use MATLAB to calculate characteristics of a second-order system, such as damping ratio, ζ; natural frequency, ω_n; percent overshoot, $\%OS$ (pos); settling time, T_s; and peak time, T_p. Let us look at Example 4.6 in the text.

```
'(ch4apB1) Example 4.6'              % Display label.
p1=[1 3+7*i];                        % Define polynomial containing
                                     % first pole.
p2=[1 3-7*i];                        % Define polynomial containing
                                     % second pole.
deng=conv(p1,p2);                    % Multiply the two polynomials to
                                     % find the 2nd order polynomial,
                                     % as^2+bs+c.
omegan=sqrt(deng(3)/deng(1))         % Calculate the natural frequency,
                                     % sqrt(c/a).
zeta=(deng(2)/deng(1))/(2*omegan)
                                     % Calculate damping ratio,
                                     % ((b/a)/2*wn).
Ts=4/(zeta*omegan)                   % Calculate settling time,
                                     % (4/z*wn).
Tp=pi/(omegan*sqrt(1-zeta^2))        % Calculate peak time,
                                     % pi/wn*sqrt(1-z^2).
pos=100*exp(-zeta*pi/sqrt(1-zeta^2))
                                     % Calculate percent overshoot
                                     % (100*e^(-z*pi/sqrt(1-z^2)).
pause
```

ch4apB2 (Example 4.8) We can use MATLAB to obtain system step responses. These responses are particularly valuable when the system is not a pure two-pole system and has additional poles or zeros. We can obtain a plot of the step response of a transfer function, $T(s)$ =`num/den`, using the command `step(T)`, where `T` is an LTI transfer-function object. Multiple plots also can be obtained using `step(T1,T2,...)`.

Information about the plots obtained with `step(T)` can be found by left-clicking the mouse on the curve. You can find the curve's label as well as the coordinates of the point on which you clicked. Right-clicking away from a curve brings up a menu. From this menu you can select (1) system responses to be displayed and (2) response characteristics to be displayed, such as peak response. When selected, a dot appears on the curve at the appropriate point. Let your mouse rest on the point to read the value of the characteristic. You may also select (3) choice for grid on or off, (4) choice to normalize the curve, and (5) properties, such as labels, limits, units, style, and characteristics.

If we add the left-hand side, `[y,t]=step(T)`, we create vectors containing the plot's points, where `y` is the output vector and `t` is the time vector. For this case, a plot is not made until the `plot(t,y)` command is given, where we assume we want to plot the output `(y)` versus time `(t)`. We can label the plot, the x-axis, and the y-axis with `title('ab')`, `xlabel('ab')`, and `ylabel('ab')`, respectively. The command `clf` clears the graph

prior to plotting. Finally, text can be placed anywhere on the graph using the command `text(X,Y,'text')`, where `(X,Y)` are the graph coordinates where `'text'` will be displayed. Let us look at Example 4.8 in the text.

```
'(ch4apB2) Example 4.8'              % Display label.
' Test Run'                          % Display label.
clf                                  % Clear graph.
numt1=[24.542];                      % Define numerator of T1.
dent1=[1 4 24.542];                  % Define denominator of T1.
' T1(s)'                             % Display label.
T1=tf(numt1,dent1)                   % Create and display T1(s).
step(T1)                             % Run a demonstration step response
                                     % plot
title (' Test Run of T1(s)' )        % Add title to graph.
pause
' Complete Run'                      % Display label.
[y1,t1]=step(T1);                    % Run step response of T1 and
                                     % collect points.
numt2=[245.42];                      % Define numerator of T2.
p1=[1 10];                           % Define (s+10) in denominator
                                     % of T2.
p2=[1 4 24.542];                     % Define (s^2+4s+24.542) in
                                     % denominator of T2.
dent2=conv(p1,p2);                   % Multiply (s + 10)(s^2+4s+24.542)
                                     % for denominator of T2.
' T2(s)'                             % Display label.
T2=tf(numt2,dent2)                   % Create and display T2.
[y2,t2]=step(T2);                    % Run step response of T2 and
                                     % collect points.
numt3=[73.626];                      % Define numerator of T3.
p3=[1 3];                            % Define (s+3) in denominator
                                     % of T3.
dent3=conv(p3,p2);                   % Multiply (s+3)(s^2+4s+24.542)
                                     % for denominator of T3.
' T3(s)'                             % Display label.
T3=tf(numt3,dent3)                   % Create and display T3.
[y3,t3]=step(T3);                    % Run step response of T3 and
                                     % collect points.
clf                                  % Clear graph.
plot(t1,y1,t2,y2,t3,y3)              % Plot acquired points with all
                                     % three plots on one graph.
title (' Step Responses of T1(s),T2 (s), and T3(s)')
                                     % Add title to graph.
xlabel(' Time(seconds)')             % Add time axis label.
ylabel (' Normalized Response')      % Add response axis label.
text(0.7,0.7,' c3(t)')               % Label step response of T1.
text(0.7,1.1,' c2(t)')               % Label step response of T2.
text(0.5,1.3,' c1(t)')               % Label step response of T3.
pause
step(T1,T2,T3)                       % Use alternate method of plotting
                                     % step responses.
title (' Step Responses of T1(s), T2(s), and T3(s)')
                                     % Add title to graph.
pause
```

ch4apB3 We also can plot the step response of systems represented in state space using the `step(T,t)` command. Here `T` is any LTI object and `t=a:b:c` is the range for the time axis,

where a is the initial time, b is the time step size, and c is the final time. For example, `t=0:1:10` means time from 0 to 10 seconds in steps of 1 second. The `t` field is optional. Finally, in this example we introduce the command `grid on`, which superimposes a grid over the step response. Place the `grid on` command after the `step(T,t)` command.

```
'(ch4apB3)'                          % Display label.
clf                                  % Clear graph.
A=[0 1 0;0 0 1;-24 -26 -9];          % Generate A matrix.
B=[0;0;1];                           % Generate B vector.
C=[2 7 1];                           % Generate C vector.
D=0;                                 % Generate D.
T=ss(A,B,C,D)                        % Generate LTI object, T, in state
                                     % space and display.
t=0:0.1:10;                          % Define range of time for plot.
step(T,t)                            % Plot step response for given
                                     % range of time.
grid on                              % Turn grid on for plot.
pause
```

ch4apB4 (Antenna Control Case Study) We now use MATLAB to plot the step response requested in the Antenna Control Case Study.

```
' (ch4apB4) Antenna Control Case Study'
                                     % Display label.
clf                                  % Clear graph.
numg=20.83;                          % Define numerator of G(s).
deng=[1 101.71 171];                 % Define denominator of G(s).
'G(s)'                               % Display label.
G=tf(numg,deng)                      % Form and display transfer
                                     % Function G(s).
step(G);                             % Generate step response.
title('Angular Velocity Response')
                                     % Add title.
pause
```

ch4apB5 (UFSS Case Study) As a final example, let us use MATLAB to do the UFSS Case Study in the text (*Johnson*, *1980*). We introduce table lookup to find the rise time. Using the `interp1(y,t,y1)` command, we set up a table of values of amplitude, `y`, and time, `t`, from the step response and look for the value of time for which the amplitude is `y1`= 0.1 and 0.9. We also generate time response data over a defined range of time using `t=a:b:c` followed by `[y,t]=step(G,t)`. Here `G` is an LTI transfer-function object and `t` is the range for the time axis, where a is the initial time, b is the time step size, and c is the final time; `y` is the output.

```
' (ch4apB5) UFSS Case Study'         % Display label.
clf                                  % Clear graph.
'(a)'                                % Display label.
numg=0.0169;                         % Define numerator of 2nd order
                                     % approximation of G(s).
deng=[1 0.226 0.0169];               % Define 2nd order term of
                                     % denominator of G(s).
'G(s)'                               % Display label.
G=tf(numg,deng)                      % Create and display G(s).
omegan=sqrt(deng(3))                 % Find natural frequency.
zeta=deng(2)/(2*omegan)              % Find damping ratio.
Ts=4/(zeta*omegan)                   % Find settling time.
Tp=pi/(omegan*sqrt(1-zeta^2))        % Find peak time.
```

```
pos=exp(-zeta*pi/sqrt(1-zeta^2))*100
                                    % Find percent overshoot.
t=0:0.1:35;                         % Limit time to find rise time. t=0
                                    % to 35 in steps of 0.1.
[y,t]=step(G,t);                    % Generate and save points of step
                                    % response over defined range of t.
Tlow=interp1(y,t,0.1);              % Search table for time when
                                    % y=0.1*finalvalue.
Thi=interp1(y,t,0.9);               % Search table for
                                    % time=0.9*finalvalue.
Tr=Thi-Tlow                         % Calculate rise time.
'(b)'                               % Display label.
numc=0.125*[1 0.435];               % Define numerator of C(s).
denc=conv(poly([0 -1.23]),[1 0.226 0.0169]);
                                    % Define denominator of C(s).
[K,p,k]=residue(numc,denc)          % Find partial-fraction expansion.
'(d)'                               % Display label.
numg=0.125*[1 0.435];               % Define numerator of G(s).
deng=conv([1 1.23],[1 0.226 0.0169]);
                                    % Define denominator of G(s).
'G(s)'                              % Display label.
G=tf(numg,deng)                     % Create and display G(s).
[y,t]=step(G);                      % Generate complete step response
                                    % and collect points.
plot(t,y)                           % Plot points.
title('Pitch Angle Response')       % Add title.
xlabel('Time(seconds)')             % label time axis.
ylabel('Pitch Angle (radians)')     % Label y-axis.
pause
```

Chapter 5: Reduction of Multiple Subsystems

ch5apB1 (UFSS Pitch Control System) MATLAB can be used for block diagram reduction. Three methods are available: (1) Solution via Series, Parallel, and Feedback Commands, (2) Solution via Algebraic Operations, and (3) Solution via Append and Connect Commands. Let us look at each of these methods.

1. Solution via Series, Parallel, and Feedback Commands

The closed-loop transfer function is obtained using the following commands successively, where the arguments are LTI objects: `series(G1,G2)` for a cascade connection of $G_1(s)$; and $G_2(s)$; `parallel(G1,G2)` for a parallel connection of $G_1(s)$ and $G_2(s)$; `feedback(G,H,sign)` for a closed-loop connection with $G(s)$ as the forward path, $H(s)$ as the feedback, and `sign` is –1 for negative-feedback systems or +1 for positive-feedback systems. The sign is optional for negative-feedback systems.

2. Solution via Algebraic Operations

Another approach is to use arithmetic operations successively on LTI transfer functions as follows: `G2*G1` for a cascade connection of $G_1(s)$ and $G_2(s)$; `G1+G2` for a parallel connection of $G_1(s)$ and $G_2(s)$; `G/(1+G*H)` for a closed-loop negative-feedback connection with $G(s)$ as the forward path and $H(s)$ as the feedback; `G/(1-G*H)` for positive-feedback systems. When using division we follow with the function `minreal(sys)` to cancel common terms in the numerator and denominator.

3. Solution via Append and Connect Commands

The last method, which defines the topology of the system, may be used effectively for complicated systems. First, the subsystems are defined. Second, the subsystems are

appended, or gathered, into a multiple-input/multiple-output system. Think of this system as a single system with an input for each of the subsystems and an output for each of the subsystems. Next, the external inputs and outputs are specified. Finally, the subsystems are interconnected. Let us elaborate on each of these steps.

The subsystems are defined by creating LTI transfer functions for each. The subsystems are appended using the command `G=append(G1,G2,G3,G4,.....,Gn)`, where the `Gi` are the LTI transfer functions of the subsystems and `G` is the appended system. Each subsystem is now identified by a number based upon its position in the append argument. For example, `G3` is 3, based on the fact that it is the third subsystem in the append argument (not the fact that we write it as `G3`).

Now that we have created an appended system, we form the arguments required to interconnect their inputs and outputs to form our system. The first step identifies which subsystems have the external input signal and which subsystems have the external output signal. For example, we use `inputs=[1 5 6]` and `outputs=[3 4]` to define the external inputs to be the inputs of subsystems 1,5, and 6 and the external outputs to be the outputs of subsystems 3 and 4. For single-input/single-output systems, these definitions use scalar quantities. Thus `inputs=5,outputs=8` define the input to subsystem 5 as the external input and the output of subsystem 8 as the external output.

At this point we tell the program how all of the subsystems are interconnected. We form a Q matrix that has a row for each subsystem whose input comes from another subsystem's output. The first column contains the subsystem's number. Subsequent columns contain the numbers of the subsystems from which the inputs come. Thus, a typical row might be as follows: `[3 6 −7]`, or subsystem 3's input is formed from the sum of the output of subsystem 6 and the negative of the output of subsystem 7.

Finally, all of the interconnection arguments are used in the `connect(G Q inputs, outputs)` command, where all of the arguments have been previously defined.

Let us demonstrate the three methods for finding the total transfer function by looking at the back endpapers and finding the closed-loop transfer function of the pitch control loop for the UFSS with $K_1 = K_2 = 1$ (*Johnson, 1980*). The last method using append and connect requires that all subsystems be proper (the order of the numerator cannot be greater than the order of the denominator). The pitch rate sensor violates this requirement. Thus, for the third method, we perform some block diagram maneuvers by pushing the pitch rate sensor to the left past the summing junction and combining the resulting blocks with the pitch gain and the elevator actuator. These changes are reflected in the program. You should verify all computer results with hand calculations.

```
'(ch5apB1) UFSS Pitch Control System'
' Solution via Series, Parallel, & Feedback Commands'
                                    % Display labels.
numg1=[-1];                         % Define numerator of G1(s).
deng1=[1];                          % Define denominator of G1(s).
numg2=[0 2];                        % Define numerator of G2(s).
deng2=[1 2];                        % Define denominator of G2 (s).
numg3=-0.125*[1 0.435];             % Define numerator of G3(s).
deng3=conv([1 1.23],[1 0.226 0.0169]);
                                    % Define denominator of G3(s).
numh1=[-1 0];                       % Define numerator of H1(s).
denh1=[0 1];                        % Define denominator of H1(s).
G1=tf(numg1,deng1);                 % Create LTI transfer function,
                                    % G1(s).
G2=tf(numg2,deng2);                 % Create LTI transfer function,
                                    % G2(s).
G3=tf(numg3,deng3);                 % Create LTI transfer function,
                                    % G3(s).
```

```
H1=tf (numh1,denh1);                % Create LTI transfer function,
                                    % H1(s).
G4=series(G2,G3);                   % Calculate product of elevator
                                    % and vehicle dynamics.
G5=feedback(G4,H1);                 % Calculate closed-loop transfer
                                    % function of inner loop.
Ge=series(G1,G5);                   % Multiply inner-loop transfer
                                    % function and pitch gain.
'T(s) via Series, Parallel, & Feedback Commands'
                                    % Display label.
T=feedback(Ge,1)                    % Find closed-loop transfer
                                    % function.
'Solution via Algebraic Operations'
                                    % Display label.
clear                               % Clear session.
numg1=[-1];                         % Define numerator of G1(s).
deng1=[1];                          % Define denominator of G1(s).
numg2=[0 2];                        % Define numerator of G2(s).
deng2=[1 2];                        % Define denominator of G2(s).
numg3=-0.125*[1 0.435];             % Define numerator of G3(s).
deng3=conv([1 1.23],[1 0.226 0.0169]);
                                    % Define denominator of G3(s).
numh1=[-1 0];                       % Define numerator of H1(s).
denh1=[0 1];                        % Define denominator of H1(s).
G1=tf (numg1,deng1);                % Create LTI transfer function, G1(s).
G2=tf (numg2,deng2);                % Create LTI transfer function, G2(s).
G3=tf (numg3,deng3);                % Create LTI transfer function, G3(s).
H1=tf (numh1,denh1);                % Create LTI transfer function, H1(s).
G4=G3*G2;                           % Calculate product of elevator and
                                    % vehicle dynamics.
G5=G4/(1+G4*H1);                    % Calculate closed-loop transfer
                                    % function of inner loop.
G5=minreal(G5);                     % Cancel common terms.
Ge=G5*G1;                           % Multiply inner-loop transfer
                                    % functions.
'T(s) via Algebraic Operations'     % Display label.
T=Ge/(1+Ge);                        % Find closed-loop transfer function.
T=minreal(T)                        % Cancel common terms.
'Solution via Append & Connect Commands'
                                    % Display label.
'G1(s)=(-K1)*(1/(-K2s))=1/s'        % Display label.
numg1=[1];                          % Define numerator of G1(s).
deng1=[1 0];                        % Define denominator of G1(s).
G1=tf(numg1,deng1)                  % Create LTI transfer function,
                                    % G1(s)=pitch gain*
                                    % 1 (1/Pitch rate sensor).
'G2(s)=(-K2s)*(2/(s+2)'             % Display label.
numg2=[-2 0];                       % Define numerator of G2(s).
deng2=[1 2];                        % Define denominator of G2(s).
G2=tf(numg2,deng2)                  % Create LTI transfer function,
                                    % G2(s)=pitch rate sensor*vehicle
                                    % dynamics.
'G3(s)=-0.125(s +0.435)/((s+1.23)(s^2+0.226s+0.0169))'
                                    % Display label.
numg3=-0.125*[1 0.435];             % Define numerator of G3(s).
deng3=conv([1 1.23],[1 0.226 0.0169]);
                                    % Define denominator of G3(s).
```

```
G3=tf(numg3,deng3);                 % Create LTI transfer function,
                                    % G3(s)=vehicle dynamics.
System=append(G1,G2,G3)             % Gather all subsystems.
input=1;                            % Input is at first subsystem,
                                    % G1(s).
output=3;                           % Output is output of third
                                    % subsystem, G3(s).
Q=[1 -3 0                           % Subsystem 1, G1(s), gets its
                                    % input from the negative of the
                                    % output of subsystem 3, G3(s).
2 1 -3                              % Subsystem 2, G2(s), gets its
                                    % input from subsystem 1, G1(s),
                                    % and the negative of the output
                                    % of subsystem 3, G3(s).
3 2 0];                             % Subsystem 3, G3(s), gets its
                                    % input from subsystem 2, G2(s).
T=connect(System,Q,input,output);
                                    % Connect the subsystems.
'T(s) via Append & Connect Commands'
                                    % Display label.
T=tf(T);                            % Create LTI closed-loop transfer
                                    % function.
T=minreal(T)                        % Cancel common terms.
pause
```

ch5apB2 (Example 5.3) We can use MATLAB to calculate the closed-loop characteristics of a second-order system, such as damping ratio, ζ; natural frequency, ω_n; percent overshoot, *%OS* (pos); settling time, T_s; and peak time, T_p. The command `[numt,dent]=tfdata(T,'v')` extracts the numerator and denominator of $T(s)$ for a single-input/single-output system from which the calculations are based. The argument `'v'` returns the numerator and denominator as simple row vectors. Omitting `'v'` would return the numerator and denominator as cell arrays requiring more steps to obtain the row vectors. We end by generating a plot of the closed-loop step response. Let us look at Example 5.3 in the text.

```
'(ch5apB2) Example 5.3'             % Display label.
numg=[25];                          % Define numerator of G(s).
deng=poly([0 -5]);                  % Define denominator of G(s).
'G(s)'                              % Display label.
G=tf(numg,deng)                     % Create and display G(s).
'T(s)'                              % Display label.
T=feedback(G,1)                     % Find T(s).
[numt,dent]=tfdata(T,'v');          % Extract numerator & denominator
                                    % of T(s).
wn=sqrt(dent(3))                    % Find natural frequency.
z=dent(2)/(2*wn)                    % Find damping ratio.
Ts=4/(z*wn)                         % Find settling time.
Tp=pi/(wn*sqrt(1-z^2))              % Find peak time.
pos=exp(-z*pi/sqrt(1-z^2))*100      % Find percent overshoot.
step(T)                             % Generate step response.
pause
```

ch5apB3 MATLAB can be used to convert transfer functions to state space in a specified form. The command `[Acc Bcc Ccc Dcc]=tf2ss(num,den)` can be used to convert $T(s)$=`num/den` into controller canonical form with matrices and vectors `Acc`, `Bcc`, `Ccc`, and `Dcc`. We can then form an LTI state-space object using `Scc=ss(Acc,Bcc,Ccc,Dcc)`. This object can then be converted into parallel form

using `Sp=canon(Scc,'type')`, where `type=modal` yields the parallel form. Another choice, not used here, is `type=companion`, which yields a right companion system matrix. Transformation matrices can be used to convert to other representations. As an example, let us convert $C(s)/R(s) = 24/[(s+2)(s+3)(s+4)]$ into a parallel representation in state space, as is done in Section 5.7—Parallel Form. Notice that the product of values in the **B** and **C** vectors yields the same product as the results in Eqs. (5.49) and (5.50). Thus, the two solutions are the same, but the state variables are ordered differently, and the gains are split between the **B** and **C** vectors. We can also extract the system matrices from the LTI object using `[A,B,C,D]=ssdata(S)`, where `S` is a state-space LTI object and `A, B, C, D,` are its associated matrices and vectors.

```
'(ch5apB3)'                          % Display label.
numt=24;                             % Define numerator of T(s).
dent=poly([-2 -3 -4]);               % Define denominator of T(s).
'T(s)'                               % Display label.
T=tf(numt,dent)                      % Create and display T(s).
[Acc Bcc Ccc Dcc]=tf2ss(numt,dent);
                                     % Convert T(s) to controller
                                     % canonical form.
Scc=ss(Acc,Bcc,Ccc,Dcc);             % Create LTI controller canonical
                                     % state-space object.
Sp=canon(Scc,'modal');               % Convert controller canonical form
                                     % to parallel form.
'Controller Canonical Form'          % Display label.
[Acc,Bcc,Ccc,Dcc]=ssdata(Scc)        % Extract and display controller
                                     % canonical form matrices.
'Parallel Form'                      % Display label.
[Ap,Bp,Cp,Dp]=ssdata(Sp)             % Extract and display parallel form
                                     % matrices.
pause
```

ch5apB4 (Example 5.9) We can use MATLAB to perform similarity transformations to obtain other forms. Let us look at Example 5.9 in the text.

```
'(ch5apB4) Example 5.9'              % Display label.
Pinv=[2 0 0; 3 2 0; 1 4 5];          % Define P inverse.
P=inv(Pinv)                          % Calculate P.
'Original'                           % Display label.
Ax=[0 1 0;0 0 1;-2 -5 -7]            % Define original A.
Bx=[0 0 1]                           % Define original B.
Cx=[1 0 0]                           % Define original C.
'Transformed'                        % Display label.
Az=Pinv*Ax*P                         % Calculate new A.
Bz=Pinv*Bx                           % Calculate new B.
Cz=Cx*P                              % Calculate new C.
pause
```

ch5apB5 Using MATLAB's `[P,d]=eig(A)` command, where the columns of `P` are the eigenvectors of `A` and the diagonal elements of `d` are the eigenvalues of `A`, we can find the eigenvectors of the system matrix and then proceed to diagonalize the system. We can also use `canon (S,'modal')` to diagonalize an LTI object, `S`, represented in state space.

```
'(ch5apB5)'                          % Display label.
A=[3 1 5;4 -2 7;2 3 1];              % Define original A.
B=[1;2;3];                           % Define original B.
```

```
C=[2 4 6];                          % Define original C.
[P,d]=eig(A)                        % Generate transformation matrix,
                                    % P, and eigenvalues, d.
'Via Transformation'                % Display label.
Adt=inv(P)*A*P                      % Calculate diagonal system A.
Bdt=inv(P)*B                        % Calculate diagonal system B.
Cdt=C*P                             % Calculate diagonal system C.
'Via Canon Command'                 % Display label.
S=ss(A,B,C,0)                       % Create state-space LTI object
                                    % for original system.
Sp=canon(S,'modal')                 % Calculate diagonal system via
                                    % canon command.
pause
```

Chapter 6: Stability

ch6apB1 (Example 6.7) MATLAB can solve for the poles of a transfer function in order to determine stability. To solve for the poles of $T(s)$ use the `pole(T)` command. Let us look at Example 6.7 in the text.

```
'(ch6apB1) Example 6.7'             % Display label.
numg=1;                             % Define numerator of G(s).
deng=conv ([1 0],[2 3 2 3 2]);      % Define denominator of G(s).
G=tf(numg,deng);                    % Create G(s) object.
'T(s)'                              % Display label.
T=feedback(G,1)                     % Calculate closed-loop T(s)
                                    % object.
                                    % Negative feedback is default
                                    % when there is no sign parameter.
poles=pole(T)                       % Find poles of T(s).
pause
```

ch6apB2 (Example 6.9) We can use MATLAB to find the range of gain for stability by generating a loop, changing gain, and finding at what gain we obtain right–half-plane poles.

```
'(ch6apB2) Example 6.9'             % Display label.
K=[1:1:2000];                       % Define range of K from 1 to 2000
                                    % in steps of 1.
for n=1:length(K);                  % Set up length of DO LOOP to equal
                                    % number of K values to be tested.
          dent=[1 18 77 K(n)];      % Define the denominator of T(s)
                                    % for the nth value of K.
          poles=roots(dent);        % Find the poles for the nth value
                                    % of K.
          r=real(poles);            % Form a vector containing the real
                                    % parts of the poles for K(n).
               if max(r) >=0,       % Test poles found for the nth
                                    % value of K for a real value ≥ 0.
                  poles             % Display first pole values where
                                    % there is a real part ≥ 0.
                  K=K(n)            % Display corresponding value of K.
                  break             % Stop loop if rhp poles are found.
          end                       % End if.
end                                 % End for.
pause
```

ch6apB3 (Example 6.11) We can use MATLAB to determine the stability of a system represented in state space by using the command `eig(A)` to find the eigenvalues of the system matrix, `A`. Let us apply the concept to Example 6.11 in the text.

```
'(ch6apB3) Example 6.11'          % Display label.
A=[0 3 1;2 8 1;-10 -5 -2]         % Define system matrix, A.
eigenvalues=eig(A)                % Find eigenvalues.
pause
```

Chapter 7: Steady-State Errors

ch7apB1 (Example 7.4, sys. b) Static error constants are found using $\lim s^n G(s)$ as $s \to 0$. Once the static error constant is found, we can evaluate the steady-state error. To evaluate the static error constant we can use the command `dcgain(G)`, which evaluates $G(s)$ at $s = 0$. Let us look at Example 7.4, system (b), in the text.

```
'(ch7apB1) Example 7.4, sys.b'    % Display label.
numg=500*poly([-2 -5 -6]);        % Define numerator of G(s).
deng=poly([0 -8 -10 -12]);        % Define denominator of G(s).
G=tf(numg,deng);                  % Form G(s).
' Check Stability'                % Display label.
T=feedback(G,1);                  % Form T(s).
poles=pole(T)                     % Display closed-loop poles.
' Step Input'                     % Display label.
Kp=dcgain(G)                      % Evaluate Kp=numg/deng for s=0.
ess=1/(1+Kp)                      % Evaluate ess for step input.
' Ramp Input'                     % Display label.
numsg=conv([1 0],numg);           % Define numerator of sG(s).
densg=poly([0 -8 -10 -12]);       % Define denominator of sG(s).
sG=tf(numsg,densg);               % Create sG(s).
sG=minreal(sG);                   % Cancel common 's' in
                                  % numerator(numsg) and
                                  % denominator(densg).
Kv=dcgain(sG)                     % Evaluate Kv=sG(s) for s=0.
ess=1/Kv                          % Evaluate steady-state error for
                                  % ramp input.
' Parabolic Input'                % Display label.
nums2g=conv([1 0 0],numg);        % Define numerator of s^2G(s).
dens2g=poly([0 -8 -10-12]);       % Define denominator of s^2G(s).
s2G=tf(nums2g,dens2g);            % Create s^2G(s).
s2G=minreal(s2G);                 % Cancel common 's' in
                                  % numerator(nums2g) and
                                  % denominator(dens2g).
Ka=dcgain(s2G)                    % Evaluate Ka=s^2G(s) for s=0.
ess=1/Ka                          % Evaluate steady-state error for
                                  % parabolic input.
pause
```

ch7apB2 (Example 7.6) We can use MATLAB to evaluate the gain, K, required to meet a steady-state error specification. Let us look at Example 7.6 in the text.

```
'(ch7apB2) Example 7.6'           % Display label.
numgdK=[1 5];                     % Define numerator of G(s)/K.
dengdK=poly([0 -6 -7 -8]);        % Define denominator of G(s)/K.
GdK=tf(numgdK,dengdK);            % Create G(s)/K.
numgkv=conv([1 0],numgdK);        % Define numerator of sG(s)/K.
```

```
dengkv=dengdK;                    % Define denominator of sG(s)/K.
GKv=tf(numgkv,dengkv);            % Create sG(s)/K.
GKv=minreal(GKv);                 % Cancel common 's' in numerator
                                  % and denominator of sG(s)/K.
KvdK=dcgain(GKv)                  % Evaluate (Kv/K)=(numgkv/dengkv)
                                  % for s=0.
ess=0.1                           % Enumerate steady-state error.
K=1/(ess*KvdK)                    % Solve for K.
' Check Stability'                % Display label.
T=feedback(K*GdK,1);              % Form T(s).
poles=pole(T)                     % Display closed-loop poles.
pause
```

Chapter 8: Root Locus Techniques

ch8apB1 (Example 8.7) MATLAB allows root loci to be plotted with the `rlocus(GH)` command, where $G(s)H(s)$=`numgh/dengh` and `GH` is an LTI transfer-function object. Points on the root locus can be selected interactively using the `[K,p]=rlocfind(GH)` command. MATLAB then yields the gain `(K)` at that point as well as all other poles (`p`) that have that gain. We can zoom in and out of the root locus by changing the range of axis values using the command `axis([xmin,xmax,ymin,ymax])`. The root locus can be drawn over a grid that shows constant damping ratio `(z)` and constant natural frequency `(wn)` curves using the `sgrid(z,wn)` command. To plot multiple ζ and ω_n curves, use `z=zmin:zstep:zmax` and `wn=wnmin:wn-step:wnmax` to specify ranges of values.

```
'(ch8apB1) Example 8.7'           % Display label.
clf                               % Clear graph.
numgh=[1 -4 20];                  % Define numerator of G(s)H(s).
dengh=poly([-2 -4]);              % Define denominator of G(s)H(s).
' G(s)H(s)'                       % Display label.
GH=tf(numgh,dengh)                % Create G(s)H(s) and display.
rlocus(GH)                        % Draw root locus.
z=0.2:0.05:0.5;                   % Define damping ratio values : 0.2
                                  % to 0.5 in steps of 0.05.
wn=0:1:10;                        % Define natural frequency values:
                                  % 0 to 10 in steps of 1.
sgrid(z,wn)                       % Generate damping ratio and
                                  % natural frequency grid lines for
                                  % root locus.
title ('Root Locus')              % Define title for root locus.
pause
rlocus(GH)                        % Draw close-up root locus.
axis([-3 1 -4 4])                 % Define range on axes for root
                                  % locus close-up view.
title('Close-up')                 % Define title for close-up root
                                  % locus.
z=0.45;                           % Define damping ratio line for
                                  % overlay on close-up root locus.
wn=0;                             % Suppress natural frequency
                                  % overlay curves.
sgrid(z,wn)                       % Overlay damping ratio curve on
                                  % close-up root locus.
for k=1:3                         % Loop allows 3 points to be
                                  % selected as per Example 8.7,
                                  %(z=0.45, jwcrossing,breakaway).
```

```
[K,p]=rlocfind(GH)                  % Generate gain, K, and closed-loop
                                    % poles, p, for point selected
                                    % interactively on the root locus.
end                                 % End loop.
pause
```

ch8apB2 (Example 8.8) We can couple the design of gain on the root locus with a step-response simulation for the gain selected. We introduce the command `rlocus(G,K)`, which allows us to specify the range of gain, `K`, for plotting the root locus. This command will help us smooth the usual root locus plot by equivalently specifying more points via the argument, `K`. Notice that the first root locus plotted without the argument `K` is not smooth. We also introduce the command `x=input('prompt')`, which allows keyboard entry of a value for `x` in response to a prompt. We apply this command to enter the desired percent overshoot. We also add a variable's value to the title of the root locus and step-response plots by inserting another field in the title command and use `num2str(value)` to convert value from a number to a character string for display. Let us apply the concepts to Example 8.8 in the text.

```
'(ch8apB2) Example 8.8'             % Display label.
clear                               % Clear variables from workspace.
clf                                 % Clear graph.
numg=[1 1.5];                       % Define numerator of G(s).
deng=poly([0 -1 -10]);              % Define denominator of G(s).
'G(s)'                              % Display label.
G=tf(numg,deng)                     % Create and display G(s).
rlocus(G)                           % Draw root locus (H(s)=1).
title('Original Root Locus')        % Add title.
pause
K=0:0.005:50;                       % Specify range of gain to smooth
                                    % root locus.
rlocus(G,K)                         % Draw smoothed root locus
                                    % (H(s)=1).
title('Smoothed Root Locus')        % Add title.
pos=input('Type %OS ');             % Input desired percent overshoot
                                    % from the keyboard.
z=-log(pos/100)/sqrt(pi^2+[log(pos/100)]^2)
                                    % Calculate damping ratio.
sgrid(z,0)                          % Overlay desired damping ratio
                                    % line on root locus.
title(['Root Locus with',num2str(pos),'% overshoot line'])
                                    % Define title for root locus
                                    % showing percent overshoot used.
[K,p]=rlocfind(G)                   % Generate gain, K, and closed-loop
                                    % poles, p, for point selected
                                    % interactively on the root locus.
pause
'T(s)'                              % Display label.
T=feedback(K*G,1)                   % Find closed-loop transfer
                                    % function
                                    % with selected K and display.
step(T)                             % Generate closed-loop step
                                    % response for point select on
                                    % root locus.
title(['Step Response for K=',num2str(K)])
                                    % Give step response a title which
                                    % includes the value of K.
pause
```

Chapter 9: Design via Root Locus

ch9apB1 (Example 9.3) We can use MATLAB to design PD controllers. The program allows us to input a desired percent overshoot via the keyboard. MATLAB then produces a root locus for the uncompensated system with an overlay of the percent overshoot line. We interactively select the intersection of the root locus and the desired percent overshoot line to set the gain. MATLAB outputs an estimate of the uncompensated system's performance specifications and a step response of the uncompensated system for us to determine the required settling time. After we input the settling time through the keyboard, MATLAB designs the PD controller and produces a root locus of the PD compensated system from which we can interactively select the gain. Finally, MATLAB produces an estimate of the PD compensated system's performance specifications and a step response of the PD compensated system.

```
'(ch9apB1) Example 9.3'              % Display label.
clf                                  % Clear graph.
'Uncompensated System'               % Display label.
numg=1;                              % Generate numerator of G(s).
deng=poly([0 -4 -6]);                % Generate denominator of G(s).
'G(s)'                               % Display label.
G=tf(numg,deng)                      % Create and display G(s).
pos=input('Type desired percent overshoot ');
                                     % Input desired percent overshoot.
z=log(pos/100)/sqrt(pi^2+[log(pos/100)]^2);
                                     % Calculate damping ratio.
rlocus(G)                            % Plot uncompensated root locus.
sgrid(z,0)                           % Overlay desired percent
                                     % overshoot line.
title(['Uncompensated Root Locus with ',num2str(pos),...
'% Overshoot Line'])                 % Title uncompensated root locus.
[K,p]=rlocfind(G);                   % Generate gain, K, and closed-loop
                                     % poles, p, for point selected
                                     % interactively on the root locus.
'Closed-loop poles='                 % Display label.
p                                    % Display closed-loop poles.
f=input('Give pole number that is operating point ');
                                     % Choose uncompensated system
                                     % dominant pole.
'Summary of estimated specifications for selected point on'
'uncompensated root locus'           % Display label.
operatingpoint=p(f)                  % Display uncompensated dominant
                                     % pole.
gain=K                               % Display uncompensated gain.
estimated_settling_time=4/abs(real(p(f)))
                                     % Display uncompensated settling
                                     % time.
estimated_peak_time=pi/abs(imag(p(f)))
                                     % Display uncompensated peak time.
estimated_percent_overshoot=pos
                                     % Display uncompensated percent
                                     % overshoot.
estimated_damping_ratio=z            % Display uncompensated damping
                                     % ratio.
estimated_natural_frequency=sqrt(real(p(f))^2+imag(p(f))^2)
                                     % Display uncompensated natural
                                     % frequency.
numkv=conv([1 0],numg);              % Set up numerator to evaluate Kv.
```

```
denkv=deng;                              % Set up denominator to evaluate Kv.
sG=tf(numkv,denkv);                      % Create sG(s).
sG=minreal(sG);                          % Cancel common poles and zeros.
Kv=dcgain(K*sG)                          % Display uncompensated Kv.
ess=1/Kv                                 % Display uncompensated
                                         % steady-state
                                         % error for unit ramp input.
'T(s)'                                   % Display label.
T=feedback(K*G,1)                        % Find uncompensated T(s).
step(T)                                  % Plot step response of
                                         % uncompensated system.
title(['Uncompensated System Step Response with ',num2str(pos),...
'% Overshoot'])                          % Add title to uncompensated step
                                         % response.
' Press any key to go to PD compensation'
                                         % Display label.
pause
' Compensated system'                    % Display label.
Ts=input('Type Desired Settling Time ');
                                         % Input desired settling time from
                                         % the keyboard.
wn=4/(Ts*z);                             % Calculate natural frequency.
desired_pole=(-z*wn)+(wn*sqrt(1-z^2)*i);
                                         % Calculate desired dominant pole
                                         % location.
angle_at_desired_pole=(180/pi)*...
angle(polyval(numg,desired_pole)/polyval(deng,desired_pole));
                                         % Calculate angular contribution
                                         % to desired pole without PD
                                         % compensator.
PD_angle=180-angle_at_desired_pole;
                                         % Calculate required angular
                                         % contribution from PD
                                         % compensator.
zc=((imag(desired_pole)/tan(PD_angle*pi/180))...
-real (desired_pole));                   % Calculate PD zero location.
' PD Compensator'                        % Display label.
numc=[1 zc];                             % Calculate numerator of Gc(s).
denc=[0 1];                              % Calculate denominator of Gc(s).
'Gc(s)'                                  % Display label.
Gc=tf (numc,denc)                        % Create and display Gc(s).
'G(s)Gc(s)'                              % Display label.
Ge=G*Gc                                  % Cascade G(s) and Gc(s).
rlocus(Ge,0:0.005:100)                   % Plot root locus of PD compensated
                                         % system.
sgrid(z,0)                               % Overlay desired percent
                                         % overshoot line.
title (['PD Compensated Root Locus with ', num2str(pos),...
'% Overshoot Line'])                     % Add title to PD compensated root
                                         % locus.
[K,p]=rlocfind(Ge);                      % Generate gain, K, and closed-loop
                                         % poles, p, for point selected
                                         % interactively on the root locus.
'Closed-loop poles='                     % Display label.
p                                        % Display PD compensated systems'
                                         % closed-loop poles.
f=input('Give pole number that is operating point ');
                                         % Choose PD compensated system
                                         % dominant pole.
```

```
' Summary of estimated specifications for selected point on PD'
' compensated root locus'              % Display label.
operatingpoint=p(f)                    % Display PD compensated dominant
                                       % pole.
gain=K                                 % Display PD compensated gain.
estimated_settling_time=4/abs(real(p(f)))
                                       % Display PD compensated settling
                                       % time.
estimated_peak_time=pi/abs(imag(p(f)))
                                       % Display PD compensated peak time.
estimated_percent_overshoot=pos        % Display PD compensated percent
                                       % overshoot.
estimated_damping_ratio=z              % Display PD compensated damping
                                       % ratio.
estimated_natural_frequency=sqrt(real(p(f))^2+imag(p(f))^2)
                                       % Display PD compensated natural
                                       % frequency.
s=tf([1 0],1);                         % Create transfer function,'s'.
sGe=s*Ge;                              % Create sGe(s).
sGe=minreal(sGe);                      % Cancel common poles and zeros.
Kv=dcgain(K*sGe)                       % Display compensated Kv.
ess=1/Kv                               % Display compensated
                                       % steady-state error for
                                       % unit ramp input.
' T(s)'                                % Display label.
T=feedback(K*Ge,1)                     % Create and display PD compensated
                                       %T(s).
' Press any key to continue and obtain the PD compensated step'
' response'                            % Display label.
pause
step(T)                                % Plot step response for PD
                                       % compensated system.
title(['PD Compensated System Step Response with '...
num2str(pos),'% Overshoot'])           % Add title to step response
                                       % of PD compensated system.
pause
```

ch9apB2 (Example 9.4) We can use MATLAB to design a lead compensator. The program allows us to input a desired percent overshoot via the keyboard. MATLAB then produces a root locus for the uncompensated system with an overlay of the percent overshoot line. We interactively select the intersection of the root locus and the desired percent overshoot line to set the gain. MATLAB outputs an estimate of the uncompensated system's performance specifications and a step response of the uncompensated system for us to determine the required settling time. Next we input the settling time and the lead compensator zero through the keyboard. At this point we take a different approach from that of the previous example. Rather than letting MATLAB calculate the lead compensator pole directly, MATLAB produces a root locus for every interactive guess of a lead compensator pole. Each root locus contains the desired damping ratio and natural frequency curves. When our guess is correct, the root locus, the damping ratio line, and the natural frequency curve will intersect. We then interactively select this point of intersection to input the gain. Finally, MATLAB produces an estimate of the lead-compensated system's performance specifications and a step response of the lead-compensated system.

```
'(ch9apB2) Example 9.4'                % Display label.
clf                                    % Clear graph.
```

```
' Uncompensated System'              % Display label.
numg=1;                              % Generate numerator of G(s).
deng=poly([0 -4 -6]);                % Generate denominator of G(s).
' G(s)'                              % Display label.
G=tf(numg,deng)                      % Create and display G(s).
pos=input('Type desired percent overshoot ');
                                     % Input desired percent overshoot.
z=-log(pos/100)/sqrt(pi^2+[log(pos/100)]^2);
                                     % Calculate damping ratio.
rlocus(G)                            % Plot uncompensated root locus.
sgrid(z,0)                           % Overlay desired percent
                                     % overshoot line.
title(['Uncompensated Root Locus with ', num2str(pos),...
'% Overshoot Line'])                 % Title uncompensated root locus.
[K,p]=rlocfind(G);                   % Generate gain, K, and closed-loop
                                     % poles, p, for point selected
                                     % interactively on the root locus.
' Closed-loop poles='                % Display label.
p                                    % Display closed-loop poles.
f=input('Give pole number that is operating point ');
                                     % Choose uncompensated system
                                     % dominant pole.
' Summary of estimated specifications for selected point on'
' uncompensated root locus'          % Display label.
operatingpoint=p(f)                  % Display uncompensated dominant
                                     % pole.
gain=K                               % Display uncompensated gain.
estimated_settling_time=4/abs(real(p(f)))
                                     % Display uncompensated settling
                                     % time.
estimated_peak_time=pi/abs(imag(p(f)))
                                     % Display uncompensated peak time.
estimated_percent_overshoot=pos      % Display uncompensated percent
                                     % overshoot.
estimated_damping_ratio=z            % Display uncompensated damping
                                     % ratio.
estimated_natural_frequency=sqrt(real(p(f))^2+imag(p(f))^2)
                                     % Display uncompensated natural
                                     % frequency.
numkv=conv([1 0],numg);              % Set up numerator to evaluate Kv.
denkv=deng;                          % Set up denominator to evaluate Kv.
sG=tf(numkv,denkv);                  % Create sG(s).
sG=minreal(sG);                      % Cancel common poles and zeros.
Kv=dcgain(K*sG)                      % Display uncompensated Kv.
ess=1/Kv                             % Display uncompensated
                                     % steady-state error for
                                     % unit ramp input.
' T(s)'                              % Display label.
T=feedback(K*G,1)                    % Create and display T(s).
step(T)                              % Plot step response of
                                     % uncompensated system.
title(['Uncompensated System Step Response with ',...
num2str(pos),'% Overshoot'])         % Add title to uncompensated step
                                     % response.
' Press any key to go to lead compensation'
                                     % Display label.
pause
```

```
Ts=input('Type Desired Settling Time');
                                    % Input desired settling time.
b=input('Type Lead Compensator Zero, (s+b). b= ');
                                    % Input lead compensator zero.
done=1;                             % Set loop flag.
while done==1                       % Start loop for trying lead
                                    % compensator pole.
a=input ('Enter a Test Lead Compensator Pole, (s+a). a= ');
                                    % Enter test lead compensator pole.
numge=conv(numg,[1 b]);             % Generate numerator of Gc(s)G(s).
denge=conv([1 a],deng);             % Generate denominator
                                    % of Gc(s)G(s).
Ge=tf(numge,denge);                 % Create Ge(s)=Gc(s)G(s).
wn=4/(Ts*z);                        % Evaluate desired natural
                                    % frequency.
clf                                 % Clear graph.
rlocus(Ge)                          % Plot compensated root locus with
                                    % test lead compensator pole.
axis([-10,10,-10,10])               % Change lead-compensated
                                    % root locus axes.
sgrid(z,wn)                         % Overlay grid on lead-compensated
                                    % root locus.
title(['Lead-Compensated Root Locus with ', num2str(pos),...
'% Overshoot Line, Lead Pole at ',...
num2str(-a),' and Required Wn'])    % Add title to lead-compensated
                                    % root locus.
done=input ('Are you done? (y=0,n=1)');
                                    % Set loop flag.
end                                 % End loop for trying compensator
                                    % pole.
[K,p]=rlocfind(Ge);                 % Generate gain, K, and closed-loop
                                    % poles, p, for point selected
                                    % interactively on the root locus.
'Gc(s)'                             % Display label.
Gc=tf([1 b],[1 a])                  % Display lead compensator.
'Gc(s)G(s)'                         % Display label.
Ge                                  % Display Gc(s)G(s).
'Closed-loop poles='                % Display label.
p                                   % Display lead-compensated
                                    % system's
                                    % closed-loop poles.
f=input('Give pole number that is operating point ');
                                    % Choose lead-compensated system
                                    % dominant pole.
'Summary of estimated specifications for selected point on lead'
'compensated root locus'            % Display label.
operatingpoint=p(f)                 % Display lead-compensated
                                    % dominant pole.
gain=K                              % Display lead-compensated gain.
estimated_settling_time=4/abs(real(p(f)))
                                    % Display lead-compensated
                                    % settling time.
estimated_peak_time=pi/abs(imag(p(f)))
                                    % Display lead-compensated
                                    % peak time.
estimated_percent_overshoot=pos     % Display lead-compensated
                                    % percent overshoot.
```

```
estimated_damping_ratio=z          % Display lead-compensated
                                   % damping ratio.
estimated_natural_frequency=sqrt(real(p(f))^2+imag(p(f))^2)
                                   % Display lead-compensated
                                   % natural frequency.
s=tf ([1 0],1);                    % Create transfer Function,'s'.
sGe=s*Ge;                          % Create sGe(s) to evaluate Kv.
sGe=minreal(sGe);                  % Cancel common poles and zeros.
Kv=dcgain(K*sGe)                   % Display lead-compensated Kv.
ess=1/Kv                           % Display lead-compensated steady-
                                   % state error for unit ramp input.
'T(s)'                             % Display label.
T=feedback(K*Ge,1)                 % Create and display lead-
                                   % compensated T(s).
'Press any key to continue and obtain the lead-compensated step'
'response'                         % Display label.
pause
step(T)                            % Plot step response for lead-
                                   % compensated system.
title (['Lead-Compensated System Step Response with ',...
num2str(pos),'% Overshoot'])       % Add title to step response
                                   % of lead-compensated system.
pause
```

Chapter 10: Frequency Response Techniques

ch10apB1 (Example 10.3) We can use MATLAB to make Bode plots using `bode(G)`, where `G/(s)=numg/deng` and `G` is an LTI transfer-function object. Information about the plots obtained with `bode(G)` can be found by left-clicking the mouse on the curve. You can find the curve's label, as well as the coordinates of the point on which you clicked. Right-clicking away from a curve brings up a menu if the icons on the menu bar are deselected. From this menu you can select (1) system responses to be displayed and (2) characteristics, such as peak response. When selected, a dot appears on the curve at the appropriate point. Let your mouse rest on the point to read the value of the characteristic. You may also select (3) which curves to view, (4) choice for grid on or off, (5) returning to full view after zooming, and (6) properties, such as labels, limits, units, style, and characteristics. We can obtain points on the plot using `[mag, phase, w]=bode(G)`, where magnitude, phase, and frequency are stored in `mag, phase,` and `w`, respectively. Magnitude and phase are stored as 3-D arrays. We use `mag(:,:)', phase(:,:)'` to convert the arrays to column vectors, where the apostrophe signifies matrix transpose. Let us look at Example 10.3 in the text.

```
'(ch10apB1) Example 10.3'          % Display label.
clf                                % Clear graph.
numg=[1 3];                        % Define numerator of G(s).
deng=conv([1 2],[1 2 25]);         % Define denominator of G(s).
'G(s)'                             % Display label.
G=tf(numg,deng)                    % Create and display G(s).
bode(G)                            % Make a Bode plot.
grid on                            % Turn on grid for Bode plot.
title('Open-Loop Frequency Response')
                                   % Add a title to the Bode plot.
[mag,phase,w]=bode(G);             % Store points on the Bode plot.
points=[20*log10(mag(:,:))',phase(:,:)',w]
                                   % List points on Bode plot with
                                   % magnitude in dB.
pause
```

ch10apB2 (Example 10.5) We can use MATLAB to make Nyquist diagrams using `nyquist(G)`, where `G(s)=numg/deng` and `G` is an LTI transfer-function object. Information about the plots obtained with `nyquist(G)` can be found by left-clicking the mouse on the curve. You can find the curve's label, as well as the coordinates of the point on which you clicked and the frequency. Right-clicking away from a curve brings up a menu if the icons on the menu bar are deselected. From this menu you can select (1) system responses to be displayed and (2) characteristics, such as peak response. When selected, a dot appears on the curve at the appropriate point. Let your mouse rest on the point to read the value of the characteristic. You may also select (3) whether or not to show negative frequencies, (4) choice for grid on or off, (5) choice for zooming to (–1,0), (6) returning to full view after zooming, and (7) properties, such as labels, limits, units, style, and characteristics. We can obtain points on the plot by using `[re,im,w]=nyquist(G)`, where the real part, imaginary part, and frequency are stored in `re`, `im`, and `w`, respectively, and `re` and `im` are 3-D arrays. We can specify a range of `w` by using `[re,im]=nyquist(G,w)`. We use `re(:,:)'`, and `im(:,:)'` to convert the arrays to column vectors. Let us look at Example 10.5 in the text.

```
'(ch10apB2) Example 10.5'          % Display label.
clf                                % Clear graph.
numg=[1 2];                        % Define numerator of G(s).
deng=[1 0 0];                      % Define denominator of G(s).
'G(s)'                             % Display label.
G=tf(numg,deng)                    % Create and display G(s).
nyquist(G)                         % Make a Nyquist diagram.
grid on                            % Turn on grid for Nyquist diagram.
title('Open-Loop Frequency Response')
                                   % Add a title to the Nyquist
                                   % diagram.
w=0:0.5:10;                        % Let 0 <w< 10 in steps of 0.5.
[re,im]=nyquist(G,w);              % Get Nyquist diagram points for a
                                   % range of w.
points=[re(:,:)',im(:,:)',w']      % List specified range of points
                                   % in Nyquist diagram.
pause
```

ch10apB3 (Example 10.8) We can use MATLAB to find gain margin (`Gm`), phase margin (`Pm`), the gain-margin frequency, where the phase plot goes through 180 degrees (`Wcg`), and the phase-margin frequency, where the magnitude plot goes through zero dB (`Wcp`). To find these quantities we use $[G_m, P_m, W_{cg}, W_{cp}]$`=margin(G)`, where $G(s)$`=numg/deng` and `G` is an LTI transfer-function object. Let us look at Example 10.8 in the text.

```
'(ch10apB3) Example 10.8'          % Display label.
clf                                % Clear graph.
numg=6;                            % Define numerator of G(s).
deng=conv([1 2],[1 2 2]);          % Define denominator of G(s).
'G(s)'                             % Display label.
G=tf(numg,deng)                    % Create and display G(s).
nyquist(G)                         % Make a Nyquist diagram.
grid on                            % Turn on grid for the Nyquist
                                   % diagram.
title ('Open-Loop Frequency Response')
                                   % Add a title to the Nyquist
                                   % diagram.
[Gm,Pm,Wcg,Wcp]=margin(G);         % Find margins and margin
                                   % frequencies.
```

```
'Gm(dB); Pm(deg.); 180 deg. freq.(r/s); 0 dB freq. (r/s)'
                                        % Display label.
margins=[20*log10(Gm),Pm,Wcg,Wcp]
                                        % Display margin data.
pause
```

ch10apB4 (Example 10.9) We can use MATLAB to determine the range of K for stability using frequency response methods. Let us look at Example 10.9 in the text.

```
'(ch10apB4) Example 10.9'               % Display label.
numg=1;                                 % Define numerator of G(s).
deng=poly([-2 -4 -5]);                  % Define denominator of G(s).
'G(s)'                                  % Display label.
G=tf(numg,deng)                         % Create and display G(s).
[Gm,Pm,Wcg,Wcp]=margin(G);              % Find margins and margin
                                        % frequencies.
K=Gm                                    % Display K for stability.
pause
```

ch10apB5 (Example 10.11) We can use MATLAB to find the closed-loop frequency response. Let us look at Example 10.11 in the text.

```
'(ch10apB5) Example 10.11'              % Display label.
clf                                     % Clear graph.
numg=50;                                % Define numerator of G(s).
deng=poly([0 -3 -6]);                   % Define denominator of G(s).
'G(s)'                                  % Display label.
G=tf(numg,deng)                         % Create and display G(s).
'T(s)'                                  % Display label.
T=feedback(G,1)                         % Find and display closed-loop
                                        % transfer function.
bode(T)                                 % Make a Bode plot.
grid on                                 % Turn on the grid for the plots.
title('Closed-Loop Frequency Response')
                                        % Add a title to the Bode plot.
pause
nyquist(T)                              % Make a Nyquist diagram.
title('Closed-Loop Frequency Response')
                                        % Add a title to the Nyquist
                                        % diagram.
pause
```

ch10apB6 We can use MATLAB to plot Nichols charts using `nichols(G)`, where *G(s)*=numg/deng and G is an LTI transfer-function object. The Nichols grid can be added using the `ngrid` command after the `nichols(G)` command. Information about the plots obtained with `nichols(G)` can be found by left-clicking the mouse on the curve. You can find the curve's label, as well as the coordinates of the point on which you clicked and the frequency. Right-clicking away from a curve brings up a menu if the icons on the menu bar are deselected. From this menu you can select (1) system responses to be displayed and (2) characteristics, such as peak response. When selected, a dot appears on the curve at the appropriate point. Let your mouse rest on the point to read the value of the characteristic. You may also select (3) choice for grid on or off, (4) returning to full view

after zooming, and (5) properties, such as labels, limits, units, style, and characteristics. Let us make a Nichols chart of $G(s) = 1/[s(s+1)(s+2)]$.

```
'(ch10apB6)'                          % Display label.
clf                                   % Clear graph.
numg=1;                               % Define numerator of G(s).
deng=poly([0 -1 -2]);                 % Define denominator of G(s).
'G(s)'                                % Display label.
G=tf(numg,deng)                       % Create and display G(s).
nichols(G)                            % Make a Nichols plot.
ngrid                                 % Add Nichols grid.
pause
```

ch10apB7 (Example 10.15) We can use MATLAB and frequency response methods to include time delay in the loop. Time delay is represented by `[numd,dend]=pade(T, n)`, where `T` is the delay time in seconds and `n` is the order. Larger values of `n` give better approximations to the delay, G_d(s)=`numd/dend`. Since we are plotting multiple plots, we first collect the data for the Bode plots by using `[mag,phase]=bode(G,w)`, where `w` is specified as a range of frequencies. We then use the generic plotting command. Also notice the commands used to label the axes and the plots on the Bode plot (see the MATLAB instruction manual for details). Let us look at Example 10.15 in the text.

```
'(ch10apB7) Example 10.15'            % Display label.
clf                                   % Clear graph.
hold off                              % Turn graph hold off.
numg=1;                               % Define numerator of G(s).
deng=poly([0 -1 -10]);                % Define denominator of G(s).
'G(s)'                                % Display label.
G=tf(numg,deng)                       % Create and display G(s).
w=0.01:0.1:10;                        % Let 0.01<w<10 in steps of 0.1.
[magg,phaseg]=bode (G,w);             % Collect Bode data for G(s).
[numd,dend]=pade(1,6);                % Represent the delay.
Gd=tf(numd,dend);                     % Create and display the delay,
                                      % Gd(s).
[magd,phased]=bode(Gd,w);             % Collect Bode data for Gd(s).
Ge=Gd*G;                              % Form Gd(s)G(s).
[mage,phasee]=bode(Ge,w);             % Collect Bode data for Gd(s)G(s).
subplot(2,1,1)                        % Subdivide plot area for plot 1.
semilogx(w,20*log10(mage(:,:)))       % Plot magnitude response.
grid on                               % Turn on grid for magnitude plot.
axis([0.01,10,-80,20]);               % Limit Bode plot axes.
title ('Magnitude Response with Delay')
                                      % Add title to magnitude response.
xlabel('Frequency (rad/s)')           % Label x-axis of magnitude
                                      % response.
ylabel(' 20log M')                    % Label y-axis of magnitude
                                      % response.
subplot(2,1,2)                        % Subdivide plot area for plot 2.
phased=phased-1080;                   % Adjust phase offset to compensate
                                      % for modulo 360.
phasee=phasee-1080;                   % Adjust phase offset to compensate
                                      % for modulo 360.
semilogx(w,phaseg(:,:),w,phased(:,:),w,phasee(:,:))
                                      % Plot phase response for G(s),
                                      % Gd(s), and G(s)Gd(s) on one
                                      % graph.
```

```
grid on                                  % Turn on grid for phase plot.
axis([0.01,10,-900,0]);                  % Limit Bode plot axes.
title ('Phase Response with Delay')
                                         % Add title to phase response.
xlabel('Frequency (rad/s)')              % Label x-axis of phase response.
ylabel('Phase (degrees)')                % Label y-axis of phase response.
text(1.5,-50,'Time Delay')               % Label time delay curve.
text(4,-150,'System')                    % Label system curve.
text(2.7,-300,' Total')                  % Label total curve.
pause
```

ch10apB8 (Example 10.18) We can use MATLAB and frequency response methods to determine experimentally a transfer function from frequency response data. By determining simple component transfer functions and then successively subtracting their frequency response, we can approximate the complete transfer function. Let us look at Example 10.18 in the text and use MATLAB for a portion of the problem. You can complete the program for practice. For this problem we generate the original frequency response plot via a transfer function. Normally, the data for the original frequency response plot would be tabular, and the program would begin at the step `[M0,P0]=bode(G0,w)` where the tabular data is generated. In other words, in a real application, the data would consist of column vectors `M0, P0, and w'`.

```
'(ch10apB8) Example 10.18'               % Display label.
clf                                      % Clear graph.
hold off                                 % Turn graph hold off.
% Generate the experimental Bode plots for G0(s)=numg0/deng0, that
% is, M0,P0.
numg0=70*[1 20];                         % Define numerator of G0(s).
deng0=conv([1 7],[1 2 25]);              % Partially define denominator of
                                         % G0(s).
deng0=conv(deng0,[1 70]);                % Complete the denominator of
                                         % G0(s).
G0=tf(numg0,deng0);                      % Create G0(s).
w=1:0.5:1000;                            % Let 1<w<1000 in steps of 0.5.
[M0, P0]=bode(G0,w);                     % Generate the tabular data.
[20*log10(M0(:,:))',P0(:,:)',w'];
                                         % Convert magnitude data to dB.
bode(G0,w)                               % Generate a Bode plot.
grid on                                  % Turn on grid for Bode plot.
title('Experimental')                    % Add title.
pause
clf                                      % Clear graph.
% Estimate a component part of the transfer function as
% G1 (s)=25/(s^2+2*0.22*5s+5^2) and subtract it from the experimental
% frequency response
numg1=5^2;                               % Define numerator of G1(s).
deng1=[1 2*0.22*5 5^2];                  % Define denominator of G1(s).
'First estimate'                         % Display label.
G1=tf(numg1,deng1)                       % Create and display G1(s).
[M1,P1]=bode(G1,w);                      % Generate Bode data for G1(s).
M2=20*log10(M0(:,:))-20*log10(M1(:,:));
                                         % Subtract Bode magnitude data of
                                         % G1 from original magnitude data.
P2=P0(:,:)-P1(:,:);                      % Subtract Bode phase data of G1
                                         % from original phase data.
subplot(2,1,1)                           % Divide plot area in two for
                                         % magnitude plot.
semilogx(w(:,:),M2)                      % Plot magnitude response after
                                         % subtracting.
```

```
grid on                                % Turn on grid for magnitude plot.
xlabel('Frequency (rad/sec)')          % Add x-axis label.
ylabel('Gain dB')                      % Add y-axis label.
subplot(2,1,2)                         % Divide plot area in two for phase
                                       % plot.
semilogx(w,P2)                         % Plot the phase response after
                                       % subtracting.
grid on                                % Turn on grid for phase plot.
title('Experimental Minus 25/(s^2+2*0.22*5s+5^2)')
                                       % Add title.
xlabel('Frequency (rad/sec)')          % Add x-axis label.
ylabel('Phase deg')                    % Add y-axis label.
'This completes a portion of Example 10.18.'
'The student should continue the program for practice.'
pause
```

Chapter 11: Design via Frequency Response

ch11apB1 (Example 11.1) We can design via gain adjustment on the Bode plot using MATLAB. You will input the desired percent overshoot from the keyboard. MATLAB will calculate the required phase margin and then search the Bode plot for that phase margin. The magnitude at the phase-margin frequency is the reciprocal of the required gain. MATLAB will then plot a step response for that gain. Let us look at Example 11.1 in the text.

```
'(ch11apB1) Example 11.1'              % Display label.
clf                                    % Clear graph.
numg=[100];                            % Define numerator of G(s).
deng=poly([0 -36 -100]);               % Define denominator of G(s).
G=tf(numg,deng)                        % Create and display G(s).
pos=input('Type %OS ');                % Input desired percent overshoot.
z=(-log(pos/100))/(sqrt(pi^2+log(pos/100)^2));
                                       % Calculate required damping ratio.
Pm=atan(2*z/(sqrt(-2*z^2+sqrt(1+4*z^4))))*(180/pi);
                                       % Calculate required phase margin.
w=0.01:0.01:1000;                      % Set range of frequency from 0.01
                                       % to 1000 in steps of 0.01.
[M,P]=bode(G,w);                       % Get Bode data.
Ph=-180+Pm;                            % Calculate required phase angle.
for k=1:1:length(P);                   % Search Bode data for required
                                       % phase angle.
if P(k)-Ph<=0;                         % If required phase angle is found,
                                       % find the value of
M=M(k);                                % magnitude at the same frequency.
'Required K'                           % Display label.
K=1/M                                  % Calculate the required gain.
break                                  % Stop the loop.
end                                    % End if.
end                                    % End for.
T=feedback(K*G,1);                     % Find T(s) using the calculated K.
step(T)                                % Generate a step response.
title(['Closed-Loop Step Response for K= ',num2str(K)])
                                       % Add title to step response.
pause
```

ch11apB2 (Example 11.2) Let us use MATLAB to design a lag compensator. The program solves Example 11.2 in the text and follows the same design technique

demonstrated in that example. You will input the value of gain to meet the steady-state error requirement followed by the desired percent overshoot. MATLAB then designs a lag compensator, evaluates K_v, and generates a closed-loop step response.

```
'(ch11apB2) Example 11.2'          % Display label.
clf                                % Clear graph.
K=input('Type value of K to meet steady-state error requirement ');
                                   % Input K.
pos=input ('Type %OS ');           % Input desired percent overshoot.
numg=[100*K];                      % Define numerator of G(s).
deng=poly([0 -36 -100]);           % Define denominator of G(s).
'G(s)'                             % Display label.
G=tf(numg,deng)                    % Create and display G(s).
z=(-log(pos/100))/(sqrt(pi^2+log(pos/100)^2));
                                   % Calculate required damping
                                   % ratio.
Pm=atan(2*z/(sqrt(-2*z^2+sqrt(1+4*z^4))))*(180/pi)+10;
                                   % Calculate required phase margin.
w=0.01:0.01:100;                   % Set range of frequency from 0.01
                                   % to 1000 in steps of 0.01.
[M,P]=bode(G,w);                   % Get Bode data.
Ph=-180+Pm;                        % Calculate required phase angle.
for k=1:1:length(P);               % Search Bode data for required
                                   % phase angle.
if P(k)-Ph <=0;                    % If required phase angle is found,
                                   % find the value of
M=M(k);                            % magnitude at the same frequency.
wf=w(k);                           % At this frequency the magnitude
                                   % plot must go through 0 dB.
break                              % Stop the loop.
end                                % End if.
end                                % End for.
wh=wf/10;                          % Calculate the high-frequency
                                   % break of the lag compensator.
wl=(wh/M);                         % Calculate the low-frequency
                                   % break of the lag compensator;
                                   % found from lag compensator,
                                   % Gc(s)=Kc(s+wh)/(s+wl), high & low
                                   % frequency gain requirements.
                                   % At low w, gain=1. Thus,
                                   % Kc*wh/wl=1. At high w, gain=1/M.
                                   % Thus Kc=1/M. Hence
                                   % Kc=wl/wh=1/M, or wl=wh/M.
numc=[1 wh];                       % Generate numerator of lag
                                   % compensator, Gc(s).
denc=[1 wl];                       % Generate denominator of lag
                                   % compensator, Gc(s).
Kc=wl/wh;                          % Generate K for Gc(s).
'Lag compensator'                  % Display label.
Kc                                 % Display lag compensator K.
'Gc(s)'                            % Display label.
Gc=tf(Kc*numc,denc)                % Create and display Gc(s).
'Gc(s)G(s)'                        % Display label.
GcG=Gc*G                           % Create and display Gc(s)G(s).
s=tf([1 0],1);                     % Create transfer function,'s'.
sGcG=s*GcG;                        % Create sGc(s)G(s).
sGcG=minreal(sGcG);                % Cancel common terms.
```

```
Kv=dcgain(sGcG)                    % Evaluate Kv.
T=feedback(GcG,1);                 % Create T(s).
step(T)                            % Generate a closed-loop, lag-
                                   % compensated step response.
title ('Closed-Loop Step Response for Lag-Compensated System')
                                   % Add title to step response.
pause
```

ch11apB3 (Example 11.3) Let us use MATLAB to design a lead compensator. The program solves Example 11.3 in the text and follows the same design technique demonstrated in that example. You will enter desired percent overshoot, peak time, and K_v. MATLAB then designs the lead compensator using Bode plots, calculates K_v, and plots a closed-loop step response.

```
'(ch11apB3) Example 11.3'          % Display label.
clf                                % Clear graph.
pos=input ('Type %OS ');           % Input desired percent overshoot.
Tp=input(' Type peak time ');      % Input desired peak time.
Kv=input(' Type value of Kv ');    % Input Kv.
numg=[100];                        % Define numerator of G(s).
deng=poly ([0 -36 -100]);          % Define denominator of G(s).
G=tf(numg,deng);                   % Create G(s).
s=tf([1 0],1);                     % Create transfer function,'s'.
sG=s*G;                            % Create sG(s).
sG=minreal(sG);                    % Cancel common factors.
K=dcgain(Kv/sG);                   % Solve for K.
'G(s)'                             % Display label.
G=zpk(K*G)                         % Put K into G (s), convert to
                                   % factored form, and display.
z=(-log(pos/100))/(sqrt(pi^2+log(pos/100)^2));
                                   % Calculate required damping
                                   % ratio.
Pm=atan(2*z/(sqrt(-2*z^2+sqrt(1+4*z^4))))*(180/pi);
                                   % Calculate required phase margin.
wn=pi/(Tp*sqrt(1-z^2));            % Calculate required natural
                                   % frequency.
wBW=wn*sqrt((1-2*z^2)+sqrt(4*z^4-4*z^2+2));
                                   % Determine required bandwidth.
w=0.01:0.5:1000;                   % Set range of frequency from 0.01
                                   % to 1000 in steps of 0.5
[M,P]=bode(G,w);                   % Get Bode data.
[Gm,Pm,Wcg,Wcp]=margin(G);         % Find current phase margin.
Pmreq=atan(2*z/(sqrt(-2*z^2+sqrt(1+4*z^4))))*(180/pi);
                                   % Calculate required phase margin.
Pmreqc=Pmreq+10;                   % Add a correction factor of 10
                                   % degrees.
Pc=Pmreqc-Pm;                      % Calculate phase contribution
                                   % required from lead compensator.
% Design lead compensator
beta=(1-sin(Pc*pi/180))/(1+sin(Pc*pi/180));
                                   % Find compensator beta.
magpc=1/sqrt(beta);                % Find compensator peak magnitude.
for k=1:1:length(M);               % Find frequency at which
                                   % uncompensated system has a
                                   % magnitude of 1/magpc.
                                   % This frequency will be the new
                                   % phase margin frequency.
if M(k)-(1/magpc) <=0;             % Look for peak magnitude.
```

```
wmax=w(k);                          % This is the frequency at the
                                    % peak magnitude.
break                               % Stop the loop,
end                                 % End if.
end                                 % End for.
% Calculate lead compensator zero, pole, and gain.
zc=wmax*sqrt(beta);                 % Calculate the lead compensators'
                                    % low break frequency.
pc=zc/beta;                         % Calculate the lead compensators'
                                    % high break frequency.
Kc=1/beta;                          % Calculate the lead compensators'
                                    % gain.
'Gc(s)'                             % Display label.
Gc=tf(Kc*[1 zc],[1 pc]);            % Create Gc(s).
Gc=zpk(Gc)                          % Convert Gc (s) to factored form
                                    % and display.
'Ge(s)=G(s)Gc(s)'                   % Display label.
Ge=G*Gc                             % Form Ge(s)=Gc(s)G(s).
sGe=s*Ge;                           % Create sGe(s).
sGe=minreal(sGe);                   % Cancel common factors.
Kv=dcgain(sGe)                      % Calculate Kv.
T=feedback(Ge,1);                   % Find T(s).
step(T)                             % Generate closed-loop, lead-
                                    % compensated step response.
title('Lead-Compensated Step Response')
                                    % Add title to lead-compensated
                                    % step response.
pause
```

ch11apB4 (Example 11.4) Let us use MATLAB to design a lag–lead compensator. The program solves Example 11.4 in the text and follows the same design technique demonstrated in that example. You will enter desired percent overshoot, peak time, and K_v. MATLAB then designs the lag–lead compensator using Bode plots, calculates K_v, and plots a closed-loop step response.

```
'(ch11apB4) Example 11.4'           % Display label.
clf                                 % Clear graph.
pos=input('Type %OS ');             % Input desired percent overshoot.
Tp=input('Type peak time ');        % Input desired peak time.
Kv=input('Type value of Kv ');      % Input desired Kv.
numg=[1];                           % Define numerator of G(s).
deng=poly([0 -1 -4]);               % Define denominator of G(s).
G=tf(numg,deng);                    % Create G(s) without K.
s=tf([1 0],1);                      % Create transfer function,'s'.
sG=s*G;                             % Create sG(s).
sG=minreal(sG);                     % Cancel common factors.
K=dcgain(Kv/sG);                    % Solve for K.
'G(s)'                              % Display label.
G=tf(K*numg,deng);                  % Put K into G(s).
G=zpk(G)                            % Convert G(s) to factored form and
                                    % display.
z=(-log(pos/100))/(sqrt(pi^2+log(pos/100)^2));
                                    % Calculate required damping ratio.
Pmreq=atan(2*z/(sqrt(-2*z^2+sqrt(1+4*z^4))))*(180/pi);
                                    % Calculate required phase margin.
wn=pi/(Tp*sqrt(1-z^2));             % Calculate required natural
                                    % frequency.
```

```
wBW=wn*sqrt((1-2*z^2)+sqrt(4*z^4-4*z^2+2));
                                    % Determine required bandwidth.
wpm=0.8*wBW;                        % Choose new phase-margin
                                    % frequency.
[M,P]=bode(G,wpm);                  % Get Bode data.
Pmreqc=Pmreq-(180+P)+5;             % Find phase contribution required
                                    % from lead compensator
                                    % with additional 5 degrees.
beta=(1-sin(Pmreqc*pi/180))/(1+sin(Pmreqc*pi/180));
                                    % Find beta.
                                    % Design lag compensator zero, pole,
                                    % and gain.
zclag=wpm/10;                       % Calculate zero of lag compensator.
pclag=zclag*beta;                   % Calculate pole of lag compensator.
Kclag=beta;                         % Calculate gain of lag compensator.
'Lag compensator, Glag(s)'          % Display label.
Glag=tf(Kclag*[1 zclag],[1 pclag]); % Create lag compensator.
Glag=zpk(Glag)                      % Convert Glag(s) to factored form
                                    % and display.
                                    % Design lead compensator zero,
                                    % pole, and gain.
zclead=wpm*sqrt(beta);              % Calculate zero of lead
                                    % compensator.
pclead=zclead/beta;                 % Calculate pole of lead
                                    % compensator.
Kclead=1/beta;                      % Calculate gain of lead
                                    % compensator.
'Lead compensator'                  % Display label.
Glead=tf(Kclead*[1 zclead],[1 pclead]);
                                    % Create lead compensator.
Glead=zpk(Glead)                    % Convert Glead(s) to factored form
                                    % and display.
'Lag-Lead Compensated Ge(s)'        % Display label.
Ge=G*Glag*Glead                     % Create compensated system,
                                    % Ge(s)=G(s) Glag(s) Glead(s).
sGe=s*Ge;                           % Create sGe(s).
sGe=minreal(sGe);                   % Cancel common factors.
Kv=dcgain(sGe)                      % Calculate Kv.
T=feedback(Ge,1);                   % Find T(s).
step(T)                             % Generate closed-loop, lag-lead-
                                    % compensated step response.
title('Lag-Lead-Compensated Step Response')
                                    % Add title to lag-lead-
                                    % compensated
                                    % step response.
pause
```

Chapter 12: Design via State Space

ch12apB1 (Example 12.1) We can use MATLAB to design controller gains using pole placement. You will enter the desired percent overshoot and settling time. We introduce the following commands: `[num,den]=ord2(wn,z)`, which produces a second-order system, given the natural frequency `(wn)` and the damping ratio `(z)`. Then we use the denominator `(den)` to specify the dominant poles; and `K=acker (A,B,-poles)`, which calculates controller gains from the system matrix `(A)`, the input matrix `(B)`, the desired poles `(poles)`. Let us look at Example 12.1 in the text.

```
'(ch12apB1) Example 12.1'           % Display label.
clf                                 % Clear graph.
numg=20*[1 5];                      % Define numerator of G(s).
deng=poly([0 -1 -4]);               % Define denominator of G(s).
'Uncompensated G(s)'                % Display label.
G=tf(numg,deng)                     % Create and display G(s).
pos=input('Type desired %OS ');     % Input desired percent overshoot.
Ts=input('Type desired settling time ');
                                    % Input desired settling time.
z=(-log(pos/100))/(sqrt(pi^2+log(pos/100)^2));
                                    % Calculate required damping ratio.
wn=4/(z*Ts);                        % Calculate required natural
                                    % frequency.
[num,den]=ord2(wn,z);               % Produce a second-order system
                                    % that meets the transient response
                                    % requirements.
r=roots(den);                       % Use denominator to specify
                                    % dominant poles.
poles=[r(1) r(2) -5.1];             % Specify pole placement for all
                                    % poles.
characteristiceqdesired=poly(poles)
                                    % Form desired characteristic
                                    % polynomial for display.
[Ac Bc Cc Dc]=tf2ss(numg,deng);     % Find controller canonical form
                                    % of state-space representation
                                    % of G(s).
P=[0 0 1;0 1 0;1 0 0];              % Transformation matrix for
                                    % controller canonical to phase-
                                    % variable form.
Ap=inv(P)*Ac*P;                     % Transform Ac to phase-variable
                                    % form.
Bp=inv(P)*Bc;                       % Transform Bc to phase-variable
                                    % form.
Cp=Cc*P;                            % Transform Cc to phase-variable
                                    % form.
Dp=Dc;                              % Transform Dc to phase-variable
                                    % form.
Kp=acker(Ap,Bp,poles)               % Calculate controller gains in
                                    % phase-variable form.
Apnew=Ap-Bp*Kp;                     % Form compensated A matrix.
Bpnew=Bp;                           % Form compensated B matrix.
Cpnew=Cp;                           % Form compensated C matrix.
Dpnew=Dp;                           % Form compensated D matrix.
[numt,dent]=ss2tf(Apnew,Bpnew,Cpnew,Dpnew);
                                    % Form T(s) numerator and
                                    % denominator.
'T(s)'                              % Display label.
T=tf(numt,dent)                     % Create and display T(s).
poles=roots(dent)                   % Display poles of T(s).
Tss=ss(Apnew,Bpnew,Cpnew,Dpnew)     % Create and display Tss, an LTI
                                    % state-space object.
step(Tss)                           % Produce compensated step
                                    % response.
title('Compensated Step Response')
                                    % Add title to compensated step
                                    % response.
pause
```

ch12apB2 (Example 12.2) We can test controllability by using the MATLAB command `Cm=ctrb(A,B)` to find the controllability matrix given the system matrix `(A)` and the input matrix `(B)`. This command is followed by `rank(Cm)` to test the rank of the controllability matrix `(Cm)`. Let us apply the commands to Example 12.2.

```
'(ch12apB2) Example 12.2'          % Display label.
A=[-1 1 0;0 -1 0;0 0 -2]           % Define compensated A matrix.
B=[0;1;1]                          % Define compensated B matrix.
Cm=ctrb(A,B)                       % Calculate controllability
                                   % matrix.
Rank=rank(Cm)                      % Find rank of controllability
                                   % matrix.
pause
```

ch12apB3 (Example 12.4) If we design controller gains using MATLAB, we do not have to convert to phase-variable form. MATLAB will give us the controller gains for any state-space representation we input. Let us look at Example 12.4 in the text.

```
'(ch12apB3) Example 12.4'          % Display label.
clf                                % Clear graph.
A=[-5 1 0;0 -2 1;0 0 -1];          % Define system matrix A.
B=[0;0;1];                         % Define input matrix B.
C=[-1 1 0];                        % Define output matrix C.
D=0;                               % Define matrix D.
pos=input('Type desired %OS ');    % Input desired percent overshoot.
Ts=input('Type desired settling time ')
                                   % Input desired settling time.
z=(-log(pos/100))/(sqrt(pi^2+log(pos/100)^2));
                                   % Calculate required damping ratio.
wn=4/(z*Ts);                       % Calculate required natural
                                   % frequency.
[num,den]=ord2(wn,z);              % Produce a second-order system
                                   % that meets the transient
                                   % requirements.
r=roots(den);                      % Use denominator to specify
                                   % dominant poles.
poles=[r(1) r(2) -4];              % Specify pole placement for all
                                   % poles.
K=acker(A,B,poles)                 % Calculate controller gains.
Anew=A-B*K;                        % Form compensated A matrix.
Bnew=B;                            % Form compensated B matrix.
Cnew=C;                            % Form compensated C matrix.
Dnew=D;                            % Form compensated D matrix.
Tss=ss(Anew,Bnew,Cnew,Dnew);       % Form LTI state-space object.
'T(s)'                             % Display label.
T=tf(Tss);                         % Create T(s).
T=minreal(T)                       % Cancel common terms and display
                                   % T(s).
poles=pole(T)                      % Display poles of T(s).
step(Tss)                          % Produce compensated step
                                   % response.
title('Compensated Step Response')
                                   % Add title to compensated step
                                   % response.
pause
```

ch12apB4 (Example 12.5) We can design observer gains by using the command `l=acker(A',C',poles)'`. Notice we use the transpose of the system matrix `(A)` and output matrix `(C)` along with the desired poles `(poles)`. Let us look at Example 12.5 in the text.

```
'(ch12apB4) Example 12.5'          % Display label.
numg=[1 4];                        % Define numerator of G(s).
deng=poly([-1 -2 -5]);             % Define denominator of G(s).
'G(s)'                             % Display label.
G=tf(numg,deng)                    % Create and display G(s).
[Ac,Bc,Cc,Dc]=tf2ss(numg,deng);    % Transform G(s) to controller
                                   % canonical form in state space.
Ao=Ac';                            % Transform Ac to observer
                                   % canonical form.
Bo=Cc';                            % Transform Bc to observer
                                   % canonical form.
Co=Bc';                            % Transform Cc to observer
                                   % canonical form.
Do=Dc;                             % Transform Dc to observer
                                   % canonical form.
r=roots([1 2 5])                   % Find the controller-compensated
                                   % system poles.
poles=10*[r' 10*real(r(1))]        % Make observer poles 10x bigger.
lp=acker(Ao',Co',poles)'           % Find the observer gains in
                                   % observer canonical form.
pause
```

ch12apB5 (Example 12.6) We can test observability using the MATLAB command `Om=obsv(A,C)` to find the observability matrix given the system matrix `(A)` and the output matrix `(C)`. This command is followed by `rank(Om)` to test the rank of the observability `matrix (Om)`. Let us apply the commands to Example 12.6.

```
'(ch12apB5) Example 12.6'          % Display label.
A=[0 1 0;0 0 1;-4 -3 -2]           % Define compensated A matrix.
C=[0 5 1]                          % Define compensated C matrix.
Om=obsv(A,C)                       % Form observability matrix.
Rank=rank(Om)                      % Find rank of observability
                                   % matrix.
pause
```

ch12apB6 (Example 12.8) We can design observer gains using the command `l=acker (A',C',poles)'` without transforming to observer canonical form. Let us look at Example 12.8 in the text.

```
'(ch12apB6) Example 12.8'          % Display label.
A=[-5 1 0;0 -2 1;0 0 -1];          % Define system matrix A.
B=[0;0;1];                         % Define input matrix B.
C=[1 0 0];                         % Define output matrix C.
D=0;                               % Define matrix D.
poles=roots([1 120 2500 50000])    % Specify pole placement for all
                                   % poles.
l=acker(A',C',poles)'              % Calculate observer gains.
pause
```

Chapter 13: Digital Control Systems

ch13apB1 (Example 13.4) We can convert $G_1(s)$ in cascade with a zero-order hold (z.o.h.) to $G(z)$ using MATLAB's `G=c2d (G1,T,' zoh')` command, where `G1` is an LTI continuous-system object and `G` is an LTI sampled-system object. `T` is the sampling interval and `' zoh'` is a method of transformation that assumes $G_1(s)$ in cascade with a z.o.h. We simply put $G_1(s)$ into the command (the z.o.h. is automatically taken care of) and the command returns $G(z)$. Let us apply the concept to Example 13.4. You will enter `T` through the keyboard.

```
'(ch13apB1) Example 13.4'          % Display label.
T=input('Type T ');                % Input sampling interval.
numg1s=[1 2];                      % Define numerator of G1(s).
deng1s=[1 1];                      % Define denominator of G1(s).
'G1(s)'                            % Display label.
G1=tf(numg1s,deng1s)               % Create G1(s) and display.
'G(z)'                             % Display label.
G=c2d(G1,T,'zoh')                  % Convert G1(s) in cascade with
                                   % z.o.h. to G(z) and display.
pause
```

ch13apB2 We also can use MATLAB to convert $G(s)$ to $G(z)$ when $G(s)$ is not in cascade with a z.o.h. The command `H=c2d(F,T,'zoh')` transforms $F(s)$ in cascade with a z.o.h. to $H(z)$, where $H(z) = ((z-1)/z)^*z\{F(s)/s\}$. If we let $F(s) = sG(s)$, the command solves for $H(z)$, where $H(z) = ((z-1)/z)^*z\{G(s)\}$. Hence, $z\{G(s)\} = (z/[z-1])^*H(z)$. In summary, input $F(s) = sG(s)$, and multiply the result of `H=c2d(F,T,'zoh')` by $(z/[z-1])$. This process is equivalent to finding the z-transform. We convert $G(s) = (s+3)/(s^2 + 6s + 13)$ into $G(z)$. You will enter `T`, the sampling interval, through the keyboard. `T` is used to form $H(z)$. We use an unspecified sampling interval, `T=[ ]`, to form $z/(z-1)$.

```
'(ch13apB2)'                       % Display label.
T=input('Type T ');                % Input sampling interval.
numgs=[1 3];                       % Define numerator of G(s).
dengs=[1 6 13];                    % Define denominator of G(s).
'G(s)'                             % Display label.
Gs=tf(numgs,dengs)                 % Create and display G(s).
Fs=Gs*tf([1 0],1);                 % Create F(s)=sG(s).
Fs=minreal(Fs);                    % Cancel common poles and zeros.
Hz=c2d(Fs,T,'zoh');                % Convert F(s) to H(z) assuming
                                   % z.o.h.
Gz=Hz*tf([1 0],[1 -1],[ ]);        % Form G(z)=H(z)*z/(z-1).
'G(z)'                             % Display label.
Gz=minreal(Gz)                     % Cancel common poles and zeros.
pause
```

ch13apB3 Creating Digital Transfer Functions Directly

Vector Method, Polynomial Form

A digital transfer function can be expressed as a numerator polynomial divided by a denominator polynomial, that is, $F(z) = N(z)/D(z)$. The numerator, $N(z)$, is represented by a vector, `numf`, that contains the coefficients of $N(z)$. Similarly, the denominator, $D(z)$, is represented by a vector, `denf`, that contains the coefficients of $D(z)$. We form $F(z)$ with the command, `F=tf(numf,denf,T)`, where `T` is the sampling interval. `F` is called a linear time-invariant (LTI) object. This object, or transfer function, can be used as an entity in other operations, such as addition or multiplication. We demonstrate with

$F(z) = 150(z^2 + 2z + 7)/(z^2 - 0.3z + 0.02)$. We use an unspecified sampling interval, `T=[]`. Notice after executing the `tf` command, MATLAB prints the transfer function.

Vector Method, Factored Form

We also can create digital LTI transfer functions if the numerator and denominator are expressed in factored form. We do this by using vectors containing the roots of the numerator and denominator. Thus, $G(s) = K*N(z)/D(z)$ can be expressed as an LTI object using the command, `G=zpk(numg,deng,K,T)`, where `numg` is a vector containing the roots of $N(z)$, `deng` is a vector containing the roots of $D(z)$, `K` is the gain, and `T` is the sampling interval. The expression `zpk` stands for zeros (roots of the numerator), poles (roots of the denominator), and gain, `K`. We demonstrate with $G(z) = 20(z+2)(z+4)/[(z-0.5)(z-0.7)(z-0.8)]$ and an unspecified sampling interval. Notice after executing the `zpk` command, MATLAB prints the transfer function.

Rational Expression in *z* Method, Polynomial Form (Requires Control System Toolbox 9.7)

This method allows you to type the transfer function as you normally would write it. The statement `z=tf('z')` must precede the transfer function if you wish to create a digital LTI transfer function in polynomial form equivalent to using `G=tf(numg, deng,T)`.

Rational Expression in *z* Method, Factored Form (Requires Control System Toolbox 9.7)

This method allows you to type the transfer function as you normally would write it. The statement `z=zpk('z')` must precede the transfer function if you wish to create a digital LTI transfer function in factored form equivalent to using `G=zpk(numg,−deng,K,T)`.

For both rational expression methods the transfer function can be typed in any form regardless of whether `z=tf('z')` or `z=zpk('z')` is used. The difference is in the created digital LTI transfer function. We use the same examples above to demonstrate the rational expression in z methods.

```
'(ch13apB3)'                          % Display label.
'Vector Method, Polynomial Form'      % Display label.
numf=150*[1 2 7]                      % Store 150(z^2+2z+7) in numf and
                                      % display.
denf=[1 -0.3 0.02]                    % Store(z^2-0.3z+0.02) in denf and
                                      % display.
'F(z)'                                % Display label.
F=tf(numf,denf,[])                    % Form F(z) and display.
clear                                 % Clear previous variables from
                                      % workspace.
'Vector Method, Factored Form'        % Display label.
numg=[-2 -4]                          % Store (s+2)(s+4) in numg and
                                      % display.
deng=[0.5 0.7 0.8]                    % Store (s-0.5)(s-0.7)(s-0.8) in
                                      % deng and display.
K=20                                  % Define K.
'G(z)'                                % Display label.
G=zpk(numg,deng,K,[])                 % Form G(z) and display,
clear                                 % Clear previous variables from
                                      % workspace.
'Rational Expression Method, Polynomial Form'
                                      % Display label.
z=tf('z')                             % Define z as an LTI object in
                                      % polynomial form.
```

```
F=150*(z^2+2*z+7)/(z^2-0.3*z+0.02)
                                    % Form F(z) as an LTI transfer
                                    % function in polynomial form.
G=20*(z+2)*(z+4)/[(z-0.5)*(z-0.7)*(z-0.8)]
                                    % Form G(z) as an LTI transfer
                                    % function in polynomial form.
clear                               % Clear previous variables from
                                    % workspace.
'Rational Expression Method, Factored Form'
                                    % Display label.
z=zpk('z')                          % Define z as an LTI object in
                                    % factored form.
F=150*(z^2+2*z+7)/(z^2-0.3*z+0.02)
                                    % Form F(z) as an LTI transfer
                                    % function in factored form.
G=20*(z+2)*(z+4)/[(z-0.5)*(z-0.7)*(z-0.8)]
                                    % Form G(z) as an LTI transfer
                                    % function in factored form.
pause
```

ch13apB4 We also can use MATLAB to convert $G(z)$ to $G(s)$ when $G(s)$ is not in cascade with a z.o.h. First, we create a sampled LTI transfer function, as discussed in ch13p3. The command `F=d2c(H,'zoh')` transforms $H(z)$ to $F(s)$ in cascade with a z.o.h., where $H(z) = ((z-1)/z)z\{F(s)/s\}$. If we consider $F(s) = sG(s)$, the command solves for $sG(s)$ given $H(z)$. Finally, $sG(s)/s = G(s)$ yields the final result. In summary, form $H(z)$, where $H(z) = ((z-1)/z)G(z)$. Use `F=d2c(H,'zoh')` to find $F(s) = sG(s)$. Divide the result by s and obtain $G(s)$. We convert $G(z) = z/(z-0.3)$ into $G(s)$. You will enter `T`, the sampling interval, through the keyboard.

```
'(ch13apB4)'                        % Display label.
T=input('Type T ');                 % Input sampling interval.
numgz=[1 0];                        % Define numerator of G(z).
dengz=[1 -.3];                      % Define denominator of G(z).
'G(z)'                              % Display label.
Gz=tf(numgz,dengz,T)                % Create and display G(z).
Hz=Gz*tf([1 -1],[1 0],T);           % Create H(z)=((z-1)/z)*G(z).
Hz=minreal(Hz);                     % Cancel common poles and zeros.
Fs=d2c(Hz,'zoh');                   % Convert from H(z) to F(s)=sG(s).
Gs=Fs*tf(1,[1 0]);                  % Create G(s)=F(s)(1/s).
'G(s)'                              % Display label.
Gs=minreal(Gs)                      % Cancel common poles and zeros.
pause
```

ch13apB5 (Example 13.6) We can use MATLAB to find the gain for stability. Let us look at Example 13.6 in the text.

```
'(ch13apB5) Example 13.6'           % Display label.
numgas=27;                          % Define numerator of Ga(s).
dengas=[1 27 0];                    % Define denominator of Ga(s).
'Ga(s)'                             % Display label.
Ga=tf(numgas,dengas)                % Create and display Ga(s).
'G(z)'                              % Display label.
Gz=c2d(Ga,0.1,'zoh')                % Find G(z) assuming Ga(s) in
                                    % cascade with z.o.h. and display.
```

```
for K=1:0.1:50;                    % Set range of K to look for
                                   % stability.
Tz=feedback(K*Gz,1);               % Find T(z).
r=pole(Tz);                        % Get poles for this value of K.
rm=max(abs(r));                    % Find pole with maximum absolute
                                   % value for this value of K.
if rm>=1,                          % See if pole is outside unit
                                   % circle.
break;                             % Stop if pole is found outside
                                   % unit circle.
end;                               % End if.
end;                               % End for.
K                                  % Display K value.
r                                  % Display closed-loop poles for
                                   % this value of K.
rm                                 % Display absolute value of pole.
pause
```

ch13apB6 (Example 13.9) We can use MATLAB's command `dcgain(Gz)` to find steady-state errors. The command evaluates the dc gain of `Gz`, a digital LTI transfer function object, by evaluating `Gz` at $z = 1$. We use the dc gain to evaluate, K_p, K_v, and K_a. Let us look at Example 13.9 in the text. You will input `T`, the sampling interval, through the keyboard to test stability.

```
'(ch13apB6) Example 13.9'          % Display label.
T=input('Type T ');                % Input sampling interval.
numg1s=[10];                       % Define numerator of G1(s).
deng1s=poly([0 -1]);               % Define denominator of G1(s).
'G1(s)'                            % Display label.
G1s=tf(numg1s,deng1s)              % Create and display G1(s).
'G(z)'                             % Display label.
Gz=c2d(G1s,T,'zoh')                % Convert G1(s) and z.o.h. to G(z)
                                   % and display.
'T(z)'                             % Display label.
Tz=feedback(Gz,1)                  % Create and display T(z).
'Closed-Loop z-Plane Poles'        % Display label.
r=pole(Tz)                         % Check stability.
M=abs(r)                           % Display magnitude of roots.
pause
Kp=dcgain(Gz)                      % Calculate Kp.
GzKv=Gz*(1/T)*tf([1 -1],[1 0],T);  % Multiply G(z) by (1/T)*(z-1).
                                   % Also, divide G(z) by z , which
                                   % makes transfer function proper
                                   % and yields same Kv.
GzKv=minreal(GzKv,0.00001);        % Cancel common poles and zeros.
Kv=dcgain(GzKv)                    % Calculate Kv.
GzKa=Gz*(1/T^2)*tf([1 -2 1],[1 0 0],T); % Multiply G(z) by (1/T^2)(z-1)^2.
                                   % Also, divide G(z)by z^2 , which
                                   % makes the transfer function
                                   % proper and yields the same Ka.
GzKa=minreal(GzKa,0.00001);        % Cancel common poles and zeros.
Ka=dcgain(GzKa)                    % Calculate Ka.
pause
```

ch13apB7 (Example 13.10) We now use the root locus to find the gain for stability. First, we create a digital LTI transfer-function object for $G(z) = N(z)/D(z)$, with an unspecified sampling interval. The LTI object is created using `tf(numgz,dengz,[])`, where `numgz` represents

$N(z)$, `dengz` represents $D(z)$, and `[]` indicates an unspecified sampling interval. MATLAB produces a z-plane root locus along with the unit circle superimposed using the command, `zgrid ([],[])`. We then interactively select the intersection of the root locus and the unit circle. MATLAB responds with the value of gain and the closed-loop poles. Let us look at Example 13.10.

```
'(ch13apB7) Example 13.10'          % Display label.
clf                                 % Clear graph.
numgz=[1 1];                        % Define numerator of G(z).
dengz=poly([1 0.5]);                % Define denominator of G(z).
'G(z)'                              % Display label.
Gz=tf(numgz,dengz,[])               % Create and display G(z).
rlocus(Gz)                          % Plot root locus.
zgrid([],[])                        % Add unit circle to root locus.
title (['z-Plane Root Locus'])      % Add title to root locus.
[K,p]=rlocfind(Gz)                  % Allows input of K by selecting
                                    % point on graphic.
pause
```

ch13apB8 (Example 13.11) We now use the root locus to find the gain to meet a transient response requirement. After MATLAB produces a z-plane root locus, along with damping ratio curves superimposed using the command `zgrid`, we interactively select the desired operating point at a damping ratio of 0.7, thus determining the gain. MATLAB responds with a gain value as well as the step response of the closed-loop sampled system using `step(Tz)`, where `Tz` is a digital LTI transfer-function object. Let us look at Example 13.11.

```
'(ch13apB8) Example 13.11'          % Display label.
clf                                 % Clear graph.
numgz=[1 1];                        % Define numerator of G(z).
dengz=poly([1 0.5]);                % Define denominator of G(z).
'G(z)'                              % Display label.
Gz=tf(numgz,dengz,[])               % Create and display G(z).
rlocus(Gz)                          % Plot root locus.
axis([0,1,-1,1])                    % Create close-up view.
zgrid                               % Add damping ratio curves to root
                                    % locus.
title(['z-Plane Root Locus'])       % Add title to root locus.
[K,p]=rlocfind(Gz)                  % Allows input of K by selecting
                                    % point on graphic.
'T(z)'                              % Display label.
Tz=feedback(K*Gz,1)                 % Find T(z).
step(Tz)                            % Find step response of gain-
                                    % compensated system.
title (['Gain Compensated Step Response'])
                                    % Add title to step response of
                                    % gain-compensated system.
pause
```

ch13apB9 (Example 13.12) Let us now use MATLAB to design a digital lead compensator. The s-plane design was performed in Example 9.6. Here we convert the design to the z-plane and run a digital simulation of the step response. Conversion of the s-plane lead compensator, $Gc(s)$=`numgcs/dengcs`, to the z-plane compensator, $Gc(z)$=`numgcz/dengcz`, is accomplished using the `Gcz=c2d(numgcs,dengcs,T,'tustin')` command to perform a Tustin transformation, where `T`=sampling interval, which for this example is 1/300. This exercise solves Example 13.12 using MATLAB.

```
'(ch13apB9) Example 13.12'          % Display label.
clf                                 % Clear graph.
T=0.01                              % Define sampling interval.
numgcs=1977*[1 6];                  % Define numerator of Gc(s).
dengcs=[1 29.1];                    % Define denominator of Gc(s).
'Gc(s) in polynomial form'          % Print label.
Gcs=tf(numgcs,dengcs)               % Create Gc(s) in polynomial form
                                    % and display.
'Gc(s) in polynomial form'          % Display label.
Gcszpk=zpk(Gcs)                     % Create Gc(s) in factored form
                                    % and display.
'Gc(z) in polynomial form via Tustin Transformation'
                                    % Display label.
Gcz=c2d(Gcs,T,'tustin')             % Form Gc(z) via Tustin
                                    % transformation.
'Gc(z) in factored form via Tustin Transformation'
                                    % Display label.
Gczzpk=zpk(Gcz)                     % Show Gc(z) in factored form.
numgps=1;                           % Define numerator of Gp(s).
dengps=poly([0 -6 -10]);            % Define denominator of Gp(s).
'Gp(s) in polynomial form'          % Display label.
Gps=tf(numgps,dengps)               % Create Gp(s) in polynomial form
                                    % and display.
'Gp(s) in factored form'            % Display label.
Gpszpk=zpk(Gps)                     % Create Gp(s) in factored form
                                    % and display.
'Gp(z) in polynomial form'          % Display label.
Gpz=c2d(Gps,T,'zoh')                % Form Gp(z) via zoh transformation.
'Gp(z) in factored form'            % Display label.
Gpzzpk=zpk(Gpz)                     % Form Gp(z) in factored form.
Gez=Gcz*Gpz;                        % Form Ge(z) = Gc(z)Gp(z).
'Ge(z)=Gc(z)Gp(z) in factored form' % Display label.
Gezzpk=zpk(Gez)                     % Form Ge(z) in factored form
                                    % and display.
'z-1'                               % Display label.
zm1=tf([1 -1],1,T)                  % Form z-1.
zm1Gez=minreal(zm1*Gez,0.00001);    % Cancel common factors.
'(z-1)Ge(z) for finding steady-state error'
                                    % Display label.
zm1Gezzpk=zpk(zm1Gez)               %Form & display(z-1)Ge(z) in
                                    % factored form.
Kv=(1/T)*dcgain(zm1Gez)             % Find Kv.
'T(z)=Ge(z)/(1+Ge(z))'              % Display label.
Tz=feedback(Gez,1)                  % Find closed-loop
                                    % transfer function, T(z)
step(Tz,0:T:2)                      % Find step response.
title('Closed-Loop Digital Lead Compensated Step Response')
                                    % Add title to step response.
```

B.3 Command Summary

`abs(x)`	Obtain absolute value of *x*.
`acker(A,B,poles)`	Find gains for pole placement.
`angle(x)`	Compute the angle of *x* in radians.
`atan(x)`	Compute arctan(*x*).
`axis([xmin,xmax,ymin,ymax])`	Define range on axes of a plot.

`bode(G,w)`	Make a Bode plot of transfer function $G(s)$ over a range of frequencies, ω. Field ω is optional.
`break`	Exit loop.
`c2d(G,T,'tustin')`	Convert $G(s)$ to $G(z)$ using the Tustin transformation. T is the sampling interval.
`c2d(G,T,'zoh')`	Convert $G(s)$ in cascade with a zero-order hold to $G(z)$. T is the sampling interval.
`canon(S,'modal')`	Convert an LTI state-space object, *S,* to parallel form.
`clear`	Clear variables from workspace.
`clf`	Clear current figure.
`conv([a b c d],[e f g h])`	Multiply $(as^3 + bs^2 + cs + d)$ by $(es^3 + fs^2 + gs + h)$.
`ctrb(A,B)`	Find controllability matrix.
`d2c(G,'zoh')`	Convert $G(z)$ to $G(s)$ in cascade with a zero-order hold.
`dcgain(G)`	Find dc gain for $G(s)$ (i.e., $s = 0$), or $G(z)$ (i.e., $z = 1$).
`eig(A)`	Find eigenvalues of matrix **A**.
`end`	End the loop.
`exp(a)`	Obtain e^a.
`feedback(G,H,sign)`	Find $T(s) = G(s)/[1 \pm G(s)H(s)]$. Sign $= -1$ or is optional for negative feedback systems. Sign $= +1$ for positive feedback systems.
`grid on`	Put grid lines on a graph.
`hold off`	Turn off graph hold; start new graph.
`imag(P)`	Form a matrix of the imaginary parts of the components of matrix **P**.
`input('str')`	Permit variable values to be entered from the keyboard with prompt `str`.
`interp1 (x,y,x1)`	Perform table lookup by finding the value of y at the value of $x = x_1$.
`inv(P)`	Find the inverse of matrix **P**.
`length(P)`	Obtain dimension of vector **P**.
`log(x)`	Compute natural log of *x.*
`log10(x)`	Compute log to the base 10 of *x.*
`margin(G)`	Find gain and phase margins, and gain and phase margin frequencies of transfer function, $G(s)$. Return [Gain margin, Phase margin, 180° frequency, 0 dB frequency].
`max(P)`	Find the maximum component of **P**.
`minreal(G,tol)`	Cancel common factors from transfer function $G(s)$ within tolerance, tol. If `'tol'` field is blank, a default value is used.
`ngrid`	Superimpose grid over a Nichols plot.
`nichols(G,w)`	Make a Nichols plot of transfer function $G(s)$ over a range of frequencies, ω. Field ω is optional.

`nyquist(G,w)`	Make a Nyquist diagram of transfer function $G(s)$ over a range of frequencies, ω. Field ω is optional.
`obsv(A,C)`	Find observability matrix.
`ord2(wn,z)`	Create a second-order system, $G(s) = 1/[s^2 + 2\zeta\omega_n s + \omega_n^2]$.
`pade(T,n)`	Obtain nth order Padé approximation for delay, T.
`pause`	Pause program until any key is pressed.
`plot(t1,y1,t2,y2,t3,y3)`	Plot $y1$ versus $t1$, $y2$ versus $t2$, and $y3$ versus $t3$ on the same graph.
`pole(G)`	Find poles of LTI transfer function object, $G(s)$.
`poly([-a -b -c])`	Form polynomial $(s + a)(s + b)(s + c)$.
`polyval(P,a)`	Find polynomial $P(s)$ evaluated at a, that is, $P(a)$.
`rank(A)`	Find rank of matrix **A**.
`real(P)`	Form a matrix of the real parts of the components of matrix **P**.
`residue(numf,denf)`	Find residues of $F(s)$ = numf/denf.
`rlocfind(GH)`	Allow interactive selection of points on a root locus plot for loop gain, $G(s)H(s)$. Return value for K and all closed-loop poles at that K.
`rlocus(GH,K)`	Plot root locus for loop gain, $G(s)H(s)$, over a range of gain, K. The K field is optional.
`roots(P)`	Find roots of polynomial, P.
`semilogx(w,P1)`	Make a semilog plot of P_1 versus $\log_{10}(\omega)$.
`series(G1,G2)`	Find $G_1(s)G_2(s)$.
`sgrid(z,wn)`	Overlay $z(\zeta)$ and wn(ω_n) grid lines on a root locus.
`sin(x)`	Find sin(x).
`sqrt(a)`	Compute $\sqrt{a}$.
`ss2tf(A,B,C,D,1)`	Convert a state-space representation to a transfer function. Return `[num,den]`.
`ss(A,B,C,D)`	Create an LTI state-space object, S.
`ss(G)`	Convert an LTI transfer function object, $G(s)$, to an LTI state-space object.
`ssdata(S)`	Extract **A**, **B**, **C**, and **D** matrices from LTI state-space object, S.
`step(G1,G2,.. Gn,t)`	Plot step responses of $G_1(s)$ through $G_n(s)$ on one graph over a range of time, t. Field t is optional as are fields G_2 through G_n.
`subplot(xyz)`	Divide plotting area into an x by y grid with z as the window number for the current plot.
`tan(x)`	Find tangent of x radians.
`text(a,b,'str')`	Put `str` on graph at graph coordinates, $x = a$, $y = b$.
`tf2ss(numg,deng)`	Convert $G(s) =$ `numg/deng` to state space in controller canonical form. Return `[A,B,C,D]`.
`vpa(a,D)`	Calculate a with D digits and convert to a symbolic with D digits.

`tf2zp(numg,deng)`	Convert $G(s) =$ `numg/deng` in polynomial form to factored form. Return `[zeros, poles, gains]`.
`tf(numg,deng,T)`	Create an LTI transfer function, $G(s) =$ `numg/deng`, in polynomial form. T is the sampling interval and should be used only if G is a sampled transfer function.
`tf(G)`	Convert an LTI transfer function, $G(s)$, to polynomial form.
`tfdata(G,'v')`	Extract numerator and denominator of an LTI transfer function, $G(s)$, and convert values to a vector. Return `[num, den]`.
`title('str')`	Put title `str` on graph.
`xlabel('str')`	Put label `str` on x-axis of graph.
`ylabel('str')`	Put label `str` on y-axis of graph.
`zgrid`	Superimpose `z(ζ)` and `wn` (ω_n) grid curves on a z-plane root locus.
`zgrid([],[])`	Superimpose the unit circle on a z-plane root locus.
`zp2tf ([−a −b]',[−c −d]', K)`	Convert $F(s) = K(s+a)(s+b)/(s+c)(s+d)$ to polynomial form. Return `[num,den]`.
`zpk(numg,deng,K,T)`	Create an LTI transfer function, $G(s) =$ `numg/deng`, in factored form. T is the sampling interval and should be used only if G is a sampled transfer function.
`zpk(G)`	Convert an LTI transfer function, $G(s)$, to factored form.

Bibliography

Johnson, H. et al. *Unmanned Free-Swimming Submersible (UFFS) System Description*. NRL Memorandum Report 4393. Naval Research Laboratory, Washington, D.C., 1980.

MathWorks. *Control System Toolbox™ Getting Started Guide R2018b*. MathWorks, Natick, MA, 2000–2018.

MathWorks. *Control System Toolbox™ User's Guide R2018b*. MathWorks, Natick, MA, 2001–2018.

AQ1 MathWorks. *MATLAB® Primer R2018b*. MathWorks, Natick, MA, 1984–2018.

MathWorks. *MATLAB® Graphics R2018b*. MathWorks, Natick, MA, 1984–2018.

MathWorks. *MATLAB® Mathematics R2018b*. MathWorks, Natick, MA, 1984–2018.

MathWorks. *MATLAB® Programming Fundamentals R2018b*. MathWorks, Natick, MA, 1984–2018.

MathWorks. *Simulink® Getting Started Guide R2018b*. MathWorks, Natick, MA, 1990–2018.

MathWorks. *Simulink® User's Guide R2018b*. MathWorks, Natick, MA, 1990–2018.

MathWorks *MATLAB® & Simulink® Installation Guide R2018b*. MathWorks, Natick, MA, 1996–2018.