C H A P T E R 15

COMPUTER ANALYSIS

We have presented methods for analyzing linear models of dynamic systems by using classical differential equations, Laplace transforms, and transfer functions. For nonlinear systems, we have shown how one can develop a linear approximation that is valid when the variables remain within some region.

For most linear models of third and higher order, and for nearly all nonlinear models, computers are used to obtain solutions for the time responses. They can also carry out other types of analyses for linear models. In this chapter we will describe two commercially available software packages: Pro-MATLAB[1] for linear models and ACSL[2] for nonlinear models. Other packages exist for this type of analysis and simulation, but these two will serve to illustrate the capabilities that are available. Both of these packages can be run on a large variety of machines, including personal computers (PCs), and both are widely used in educational and industrial settings.

We will illustrate the use of MATLAB for building and combining linear models in several different forms. Then we will show how these models can be analyzed. Finally, we will use ACSL to simulate the response of nonlinear models to arbitrary inputs.

[1]Pro-MATLAB is a product of The MathWorks, Inc. (Natick, Massachusetts) that runs on engineering workstations and a variety of other high-performance platforms. PC-MATLAB is the personal computer version.

[2]ACSL stands for Advanced Continuous Simulation Language and is a product of MGA, Inc. of Concord, Massachusetts. It runs on a wide range of machines, including engineering workstations and personal computers.

15.1 BUILDING LINEAR MODELS

MATLAB allows four different types of models for linear, continuous-time systems and the same four types for linear, discrete-time systems. We will deal only with continuous-time systems and will restrict our attention to three of the four model types: transfer-function, zero-pole, and state-space. The fourth form, which corresponds to the partial-fraction expansion of the system's transfer function, will not be covered. For each of the three types we will illustrate how the model can be created. Then we will show how to convert a model from one form to either of the remaining two and how to combine the models for subsystems in series, parallel, and feedback configurations. Methods for analyzing these models, such as computing and plotting step and impulse responses, are considered in Section 15.2.

Most of the commands we will discuss belong to the Control System Toolbox[3] that has been developed for use with Pro-MATLAB and PC-MATLAB. This toolbox consists of a set of files, all having extension `.M`, that contain MATLAB commands for implementing the model-building and analytical operations that one commonly encounters when working with control systems. Other toolboxes have been developed for areas such as signal processing and system identification. The interested reader is referred to the *MATLAB User's Guide* for details on MATLAB itself and to the *Control System Toolbox User's Guide* for more information on most of the commands we shall use here. MATLAB and its toolboxes have an online help facility that can be used to obtain detailed information about commands while the program is running.

We will introduce some of the notation and capabilities of MATLAB but will not attempt to be comprehensive. All the examples will be for single-input, single-output (SISO) systems. However, the procedures can readily be extended to include systems with multiple inputs and/or multiple outputs.

Transfer-Function Form

The usual form for the transfer function of a single-input, single-output linear model is a ratio of polynomials—that is, a rational function of the variable s. To create the model in MATLAB, we represent the numerator and denominator polynomials by row vectors whose elements are the coefficients of the corresponding powers of s in the polynomials. For example, if the numerator of $G(s)$ is the polynomial $N(s) = 3s^2+4s+5$, it can be expressed by entering the command `numG = [3 4 5]`, where the symbol `numG` is the name we have assigned to the numerator polynomial.

If the denominator of $G(s)$ is the polynomial $D(s) = s^3+2s^2+4s+8$, we can give it the name `denG` and express it by entering `denG = [1 2`

[3]A product of The MathWorks, Inc., Natick, Massachusetts.

4 8]. If we enter these two commands exactly as shown here, we will have created the transfer function

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

without any further keystrokes.

If we want to perform some operation on $G(s)$, such as computing the step response, the quantities numG and denG are used as arguments, *in this order*, to the command. Most any other names with fewer than 20 characters can be used for the numerator and denominator vectors, in place of numG and denG, as long as they are listed in the proper order in any subsequent commands. One should keep in mind that MATLAB is case-sensitive, so numG and numg are two distinct entities. MATLAB will display the names and values of the arguments on the screen, which facilitates verifying that they have been entered correctly. This echoing by MATLAB can, however, be suppressed by appending a semicolon to the line.

Zero-Pole Form

We know from Chapter 8 that a transfer function $G(s) = N(s)/D(s)$ can be expressed in terms of a gain constant, its zeros [the solutions of $N(s) = 0$], and its poles [the roots of $D(s) = 0$]. To create a model for a single-input, single-output linear system in this form, we enter the gain as a scalar and the zeros and poles as column vectors.

For example, to build a MATLAB model for a SISO system whose transfer function has a gain of 5.5, zeros at $s = -3$ and -5, and poles at $s = -1$, $-2 + j3$, and $-2 - j3$, we could enter the lines

```
k = 5.5
zro = [-3; -5]
pol = [-1; -2+3*i; -2-3*i]
```

Again, different names could have been selected in place of k, zro, and pol. The symbol i is predefined by MATLAB to represent $\sqrt{-1}$, which we have denoted by j throughout this book. The semicolons within the brackets indicate to MATLAB that the element that follows is on a new row. Thus the quantities zro and pol are column vectors. An alternative method of specifying them is to omit the semicolons and transpose the resulting row vector by appending the symbol ', as in

```
zro = [-3   -5]'
pol = [-1   -2+3*i   -2-3*i]'
```

In either case, the transfer function defined by the foregoing commands is

$$G(s) = \frac{5.5(s + 3)(s + 5)}{(s + 1)(s + 2 - j3)(s + 2 + j3)}$$

The numerical values for the gain, zeros, and poles can be entered in any order. However, when using any MATLAB command to operate on $G(s)$, we must list the *arguments* in the command in the following order: zeros, poles, and gain.

State-Space Form

For a fixed linear system, the matrix form of the state-variable model was given in (3.33) as

$$\begin{aligned} \dot{\mathbf{q}} &= \mathbf{A}\mathbf{q} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{q} + \mathbf{D}\mathbf{u} \end{aligned} \tag{1}$$

We enter this model form in MATLAB by defining the four matrices **A**, **B**, **C**, and **D**. To analyze a model that exists in state-variable form within MATLAB, we need only specify the names that have been assigned to the four coefficient matrices for that particular model as arguments of the appropriate command.

For example, the four matrices corresponding to the third-order system described by

$$\begin{aligned} \dot{q}_1 &= q_2 \\ \dot{q}_2 &= q_3 \\ \dot{q}_3 &= -q_1 - 4q_2 - 2q_3 + 2u \\ y &= q_1 - 3q_2 + 2q_3 + 0.5u \end{aligned}$$

are

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -4 & -2 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & -3 & 2 \end{bmatrix} \qquad \mathbf{D} = 0.5$$

We define this model in MATLAB by entering the lines

```
A = [0 1 0; 0 0 1; -1 -4 -2]
B = [0; 0; 2]
C = [1 -3 2]
D = 0.5
```

Changing Forms

The commands that will transform a model among the three commonly used forms of the Control System Toolbox are shown in Table 15.1. In each case, the names of the quantities to be determined are entered in square brackets

TABLE 15.1 MATLAB Commands for Changing the Form of a Model

`ss2tf`	State-space form to transfer-function form
`ss2zp`	State-space form to zero-pole form
`tf2ss`	Transfer-function form to state-space form
`tf2zp`	Transfer-function form to zero-pole form
`zp2ss`	Zero-pole form to state-space form
`zp2tf`	Zero-pole form to transfer-function form

on the left side of the equals sign, and the names of the known quantities appear as arguments of the command. For the two commands that start with the state-variable form (`ss2tf` and `ss2zp`), it is necessary to include an additional integer argument that designates the particular input involved. For single-input, single-output systems, this integer is always 1.

The syntax for using these six commands follows. We have used the names `num` and `den` to represent the numerator and denominator coefficient vectors of the transfer-function form; `k`, `zro`, and `pol` to represent the gain and the column vectors of the zero-pole form; and `A`, `B`, `C`, and `D` to represent the matrices of the state-space form. These symbols would be replaced with the user-assigned names in an actual application. The commands are

```
[num,den] = ss2tf(A,B,C,D,1)
[zro,pol,k] = ss2zp(A,B,C,D,1)
[A,B,C,D] = tf2ss(num,den)
[zro,pol,k] = tf2zp(num,den)
[A,B,C,D] = zp2ss(zro,pol,k)
[num,den] = zp2tf(zro,pol,k)
```

We will illustrate the use of these commands in the following example by entering a system model in zero-pole form, transforming it to the two other forms, and finally returning it to the original form.

▶ *EXAMPLE 15.1*

Enter the model of a system whose transfer function is

$$G(s) = \frac{4(s+1)(s+4)(s+12)}{s(s+5)(s+3-j6)(s+3+j6)}$$

and use the appropriate MATLAB commands to obtain the model that has $G(s)$ as a rational function and also the state-space model.

Solution

We begin by entering the values for the gain, zeros, and poles.


```
k = 4
z = [-1; -4; -12]
p = [0; -5; -3+6*i; -3-6*i]
```

Because there are no semicolons at the ends of the lines, MATLAB echoes the values and yields the following output.

```
k =   4
z =  -1
     -4
    -12
p =   0
     -5.0000
     -3.0000 + 6.0000i
     -3.0000 - 6.0000i
```

To obtain the coefficients of the numerator and denominator polynomials of $G(s)$, we enter the command

```
[num,den] = zp2tf(z,p,k)
```

which results in the output

```
num =     0     4    68   256   192
den =     1    11    75   225     0
```

From these two row vectors of polynomial coefficients, we can write the transfer function as

$$G(s) = \frac{4s^3 + 68s^2 + 256s + 192}{s^4 + 11s^3 + 75s^2 + 225s}$$

To obtain the state-space form of the model from the numerator and denominator polynomials, we enter the line

```
[A,B,C,D] = tf2ss(num,den)
```

which yields the following list of the four matrices.

```
A =   -11   -75  -225     0
        1     0     0     0
        0     1     0     0
        0     0     1     0
B =     1
        0
        0
        0
C =     4    68   256   192
D =     0
```

We see that A has four rows and four columns, corresponding to four state variables. This also implies that B will have four rows and that C will have four columns. Because there is one input, B and D have one column, and because there is one output, C and D have one row.

As a final step in this example, we compute the zeros, poles, and gain of the model from the state-space matrices. To distinguish their values from the original set, we refer to them as zz, pp, and kk. We enter the command

```
[zz,pp,kk] = ss2zp(A,B,C,D,1)
```

which results in the output

```
zz =   -1.0000
       -4.0000
      -12.0000
pp =    0
       -3.0000 + 6.0000i
       -3.0000 - 6.0000i
       -5.0000
kk =    4.0000
```

These values agree with the original values of z, p, and k. The fact that there are only three zeros in the list indicates that the degree of the numerator polynomial is one less than that of the denominator. Hence $G(s)$ approaches zero as s approaches infinity.

Series Connection

Figure 15.1(a) shows two single-input, single-output subsystems that have been defined in transfer-function form and are to be connected in series, resulting in the equivalent system shown in part (b) of the figure. We know from Chapter 8 that the transfer function of the resulting system will be the product of the individual transfer functions, namely

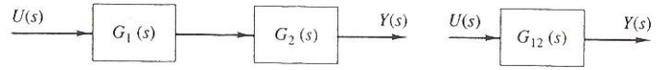
$$G_{12}(s) = G_1(s)G_2(s)$$

Assume that the polynomials numG1 and denG1 represent the first subsystem, and numG2 and denG2 the second subsystem. MATLAB can create the polynomials numG12 and denG12 via the command

```
[numG12,denG12] = series(numG1,denG1,numG2,denG2)
```

A series connection of two subsystems can also be done if both systems are expressed in state-space form in terms of their four matrices, **A**, **B**, **C**, and **D**. We assume that the outputs of the first subsystem are the same as the inputs of the second. In this case the appropriate form of the MATLAB command is

```
[A12,B12,C12,D12] = series(A1,B1,C1,D1,A2,B2,C2,D2)
```

(a) (b)

FIGURE 15.1 (a) Series connection of two subsystems in transfer-function form. (b) Equivalent system.

▶ *EXAMPLE 15.2*

Using MATLAB, enter the subsystems that have the transfer functions

$$G_1(s) = \frac{2(s+1)}{s(s+4)}$$

and

$$G_2(s) = \frac{3}{s^2+2s+5}$$

and connect them in series to form a new system that has the transfer function

$$G_{12}(s) = G_1(s)G_2(s)$$

Then obtain the zero-pole and state-space forms of $G_{12}(s)$.

Solution

Because we know the zeros, poles, and gain of $G_1(s)$, we enter it with the commands kG1 = 2, zerG1 = -1, and polG1 = [0; -4] and transform the model to transfer-function form with the command

```
[numG1,denG1] = zp2tf(zerG1,polG1,kG1)
```

The resulting matrices for the transfer-function model of subsystem 1 are

```
numG1 =     0     2     2
denG1 =     1     4     0
```

Because $G_2(s)$ is given in polynomial form, we enter the numerator and denominator polynomials as

```
numG2 = 3
denG2 = [1 2 5]
```

If we want to know the poles of $G_2(s)$, we can convert the transfer function to zero-pole form by entering

```
[zerG2,polG2,kG2] = tf2zp(numG2,denG2)
```

Doing so yields

```
zerG2 =   [ ]
polG2 =   -1.0000 + 2.0000i
          -1.0000 - 2.0000i
kG2 =     3
```

We see that $G_2(s)$ has a pair of complex poles at $s = -1 + j2$ and $s = -1 - j2$ and a gain of 3. The symbol [ ] that is shown for `zerG2` denotes an empty matrix. This means that the symbol has been defined but at present has no value. Because the numerator of $G_2(s)$ is just the constant 3, the transfer function has no zeros in the finite s-plane. In other words, the set of finite zeros of $G_2(s)$ is empty, which is what MATLAB is expressing with this notation.

Now that both subsystems have been defined in transfer-function form, we can make the series connection by giving the command

```
[num_ser,den_ser] = series(numG1,denG1,numG2,denG2)
```

which results in

```
num_ser =     0     0     0     6     6
den_ser =     1     6    13    20     0
```

This is MATLAB's representation of the rational function

$$G_{12}(s) = \frac{6s + 6}{s^4 + 6s^3 + 13s^2 + 20s}$$

To verify that the poles and zeros of the series combination are the union of those of the two subsystems, we enter the command

```
[zer_ser,pol_ser,k_ser] = tf2zp(num_ser,den_ser)
```

and obtain the result

```
zer_ser =  -1
pol_ser =   0
           -4.0000
           -1.0000 + 2.0000i
           -1.0000 - 2.0000i
k_ser =     6
```

which agrees with the form

$$G_{12}(s) = \frac{6(s + 1)}{s(s + 4)(s^2 + 2s + 5)}$$

where the complex poles correspond to the quadratic term in the denominator.

To obtain a state-space representation of the series connection, we enter

```
[A_ser,B_ser,C_ser,D_ser] = tf2ss(num_ser,den_ser)
```

and obtain

```
A_ser =   -6   -13   -20     0
           1     0     0     0
           0     1     0     0
           0     0     1     0

B_ser =    1
           0
           0
           0

C_ser =    0     0     6     6

D_ser =    0
```

From the dimensions of these matrices, we can see that the resulting system has four state variables, one input, and one output.

Feedback Connection

Figure 15.2(a) shows two subsystems that are connected in the feedback configuration introduced in Section 13.5 and shown in Figure 13.22(a). When each subsystem has a single input and a single output, and when the feedback signal has a negative sign where it enters the summing junction, the MATLAB command

```
[num_fbk,den_fbk]=feedback(num_G,den_G,num_H,den_H)
```

will create the numerator and denominator polynomials for the feedback interconnection.

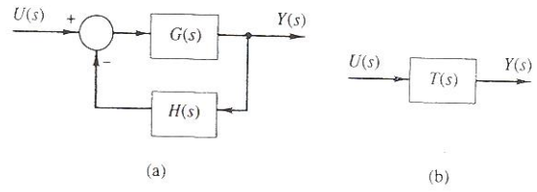


FIGURE 15.2 (a) Feedback connection of two subsystems. (b) Equivalent system.

Alternatively, we can handle the feedback interconnection of two multi-input, multi-output subsystems via the state-space representations, provided that we take care to ensure that the numbers of inputs and outputs are consistent at each connection point. The default condition is for a negative sign on the feedback signal where it enters the summing junction, although it is possible to obtain a positive sign by using an additional argument in the `feedback` command.

EXAMPLE 15.3

The two subsystems used in the last example appear in the feedback configuration shown in Figure 15.3. Find the closed-loop transfer function and also its gain, zeros, and poles.

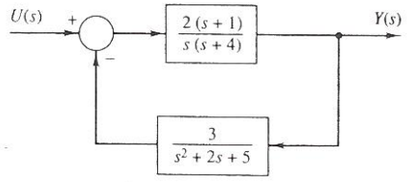


FIGURE 15.3 Feedback connection for Example 15.3.

Solution

The two blocks that were created in the previous example can be connected in a feedback configuration (with a negative sign where the feedback signal enters the summing junction) by issuing the command

```
[num_fbk,den_fbk]=feedback(numG1,denG1,numG2,denG2)
```

which will create the following coefficients for the numerator and denominator polynomials in the closed-loop transfer function.

```
num_fbk =     0     2     6    14    10
den_fbk =     1     6    13    26     6
```

Thus

$$T(s) = \frac{2s^3 + 6s^2 + 14s + 10}{s^4 + 6s^3 + 13s^2 + 26s + 6} \tag{2}$$

To determine the closed-loop poles and zeros, we use the `tf2zp` command

```
[zer_fbk,pol_fbk,k_fbk] = tf2zp(num_fbk,den_fbk)
```


and obtain

```
zer_fbk =  -1.0000 + 2.0000i
           -1.0000 - 2.0000i
           -1.0000

pol_fbk =  -4.3083
           -0.7154 + 2.1969i
           -0.7154 - 2.1969i
           -0.2609

k_fbk =     2
```

Note that all four of the closed-loop poles differ from those of $G_1(s)$ and $G_2(s)$. This is usually the case when two blocks are joined in a feedback configuration, whereas the poles and zeros for a series connection are the same as those of the individual blocks. We can see that the closed-loop zeros are $s = -1$, which is the zero of $G_1(s)$, and $s = -1 + j2$ and $s = -1 - j2$, which are the poles of $G_2(s)$. It is a general property of feedback systems that the closed-loop zeros are the *zeros* of the *forward*-path transfer function and the *poles* of the *feedback*-path transfer function.

15.2 ANALYSIS WITH MATLAB

Having illustrated how the model of a fixed, linear system can be expressed in a variety of forms, we now turn our attention to using MATLAB to analyze such models. Complete coverage of MATLAB's varied capabilities is well beyond our scope here, but a number of the most basic and most commonly used commands will be presented. We will show how to compute and plot the responses to step functions and impulses, and then the responses to arbitrary inputs. Next we will illustrate generation of the frequency response in the form of a Bode diagram. The section concludes with a demonstration of how root-locus plots such as those shown in Chapter 14 are created.

The commands we will discuss can be used either with or without arguments on the left side of an equals sign. When one or more arguments are present, as in the command `[y,x,t] = step(num,den)`, MATLAB supplies the numerical values of the variables listed inside the square brackets to the left of the equals sign. If a plot is to be drawn, such as `y` versus `t`, it is up to the user to enter the appropriate plotting command(s). Alternatively, when a command is given without the equals sign and any arguments to the left of it, as in `step(num,den)`, MATLAB generates the plot automatically but does not supply any numerical values.

Each of the commands we will illustrate can be done with the model expressed in either transfer-function or state-space form. For the most part,

we will use the transfer-function form where the model is given in terms of its numerator and denominator coefficients. For systems of relatively low order, say 10 or less, the choice is not critical. For systems of higher order, the state-space form, which uses the four coefficient matrices, is generally less susceptible than the transfer-function form to numerical errors.

Step and Impulse Responses

MATLAB computes time responses for specified inputs and initial conditions by using the system's state-transition matrix (see Section 6.6). The user generally selects the length of time for which the plot is desired and also the time interval between adjacent points. There are commands for computing the step and impulse responses, and they will yield the values of both the outputs and the state variables, if desired. A more general command called `lsim` exists for obtaining the responses to rather arbitrary inputs; it will be demonstrated shortly. In the following example, we illustrate the use of the `step` and `impulse` commands.

▶ *EXAMPLE 15.4*

Compute and plot the unit step response for the feedback system constructed in Example 15.3, for the interval $0 \leq t \leq 20$.

Solution

Because, from the solution of Example 15.3, the model exists as the arrays `num_fbk` and `den_fbk`, we need only define the time vector before computing the time responses. MATLAB requires uniformly spaced time points, defined by either a row vector or a column vector. To use an interval of 0.10 seconds and to define the time variable as a row vector, we enter

```
time = [0:0.1:20];
```

where the semicolon suppresses the printing of the 201 values. The step-response values of the output variable are generated and plotted in response to the command

```
step(num_fbk,den_fbk,time)
```

Grid lines and a title can be added to the plot via the line

```
grid;title('step response of fdbk configuration')
```

The resulting plot appears as Figure 15.4. In order to see the response to a unit impulse, we can issue the same commands, with `step` replaced by `impulse`.

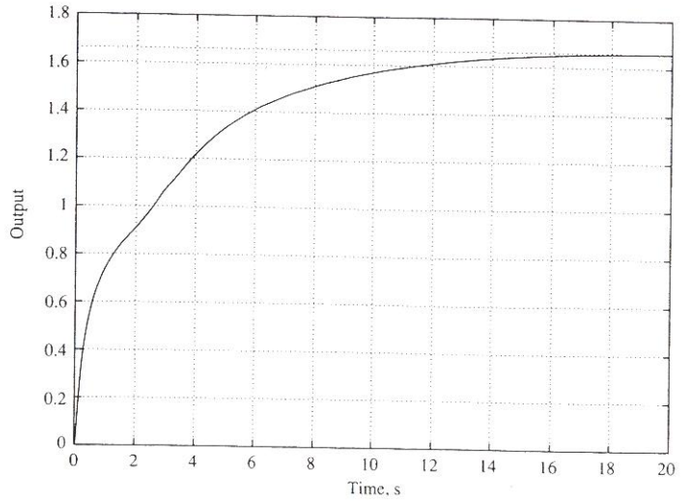


FIGURE 15.4 Unit step response of the closed-loop system for Examples 15.3 and 15.4.

Responses to Arbitrary Inputs

In addition to finding the step and impulse responses of a linear model, MATLAB can also compute the responses to arbitrary inputs, both with and without initial conditions on the state variables. The command that accomplishes this is called `lsim`, which stands for *linear simulation.*

The model can be in either state-space or transfer-function form, and the user must define both the time vector and the input. For single-input models, the input variable must be a column vector that has one element per time point. For multi-input models, it must be a matrix that has one column for each of the model's inputs. The time vector is defined as for the step and impulse responses—that is, as a row or column vector of uniformly spaced time points. If the initial state is not zero, the state-space form of the model must be used.

▶ *EXAMPLE 15.5*

The feedback system shown in Figure 15.5 is the same as that in Figure 15.3, except that an adjustable gain K has been included in the forward path. Obtain the zero-state response over the interval $0 \leq t \leq 50$ seconds with $K = 2.5$. The input is the signal shown in Figure 15.6, where $A = 1$, $t_1 = 1$ s, $t_2 = 16$ s, and $t_3 = 31$ s.

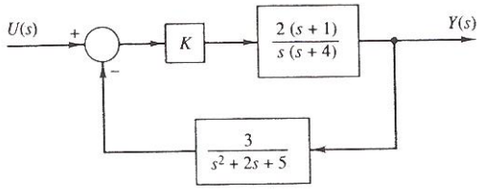


FIGURE 15.5 Feedback system from Example 15.4 with adjustable gain K added to the forward path.

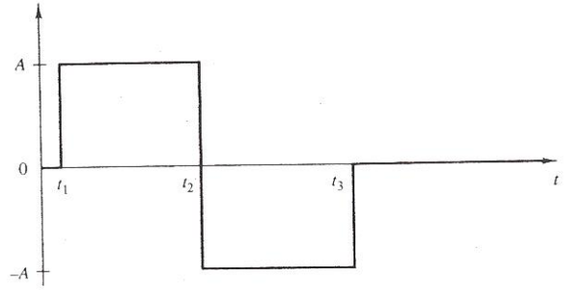


FIGURE 15.6 Input signal for Example 15.5.

Solution

To build the MATLAB model of the closed-loop system with the gain K set to 2.5, we start with the transfer functions $G_1(s)$ and $G_2(s)$ that were created in Example 15.2, create the transfer function of the forward path by multiplying $G_1(s)$ by the constant 2.5, and use the `feedback` command.

```
K = 2.5;
[num_K,den_K] = feedback(K*numG1,denG1,numG2,denG2)
```

Because there is no semicolon at the end of the `feedback` statement, the results are displayed.

```
num_K =     0     5    15    35    25
den_K =     1     6    13    35    15
```

We now have the model of the closed-loop system in transfer-function form and are ready to define the time and input vectors. To create a column vector for the time, with the points separated by 0.1 s, we enter

```
time = [0:0.1:50]';
```

which will give us 501 time points. The apostrophe after the closing bracket denotes the transpose of the matrix.

The length of the input vector must always be the same as the length of the time vector. We shall first initialize the input vector `in` to be a column of 501 zeros. Then we shall change the input values to 1 for $1 < t \leq 16$ and to -1 for $16 < t \leq 31$, as required by Figure 15.6.

```
in = 0*time;
for i=11:160, in(i) = 1.0;end;
for i=161:310, in(i) = -1.0;end;
```

Because the initial states are to be zero, we are ready to compute the response of the output y and plot it, along with the input, via the commands

```
y = lsim(num_K,den_K,in,time);
plot(time,y,time,in);grid
title('Response of feedback system with K=2.5')
```

The resulting plot is shown in Figure 15.7, which indicates that the closed-loop system is stable but lightly damped. The labels for the input and output curves have been added by hand. To find the values of the system's closed-loop poles, we need only enter the commands

```
[zcl,pcl,kcl] = tf2zp(num_K,den_K);
pcl
```

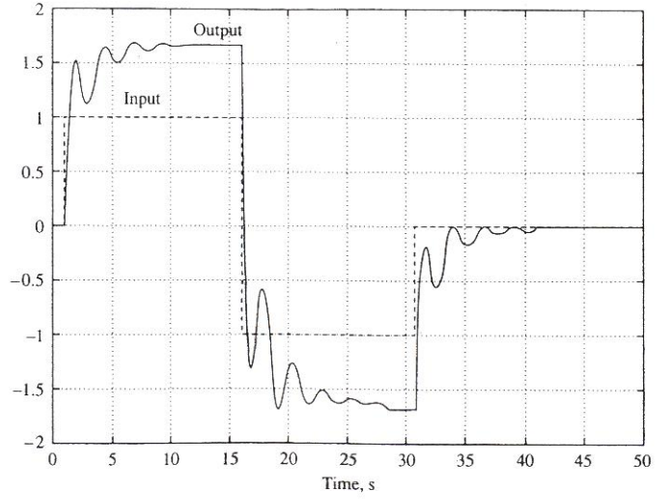


FIGURE 15.7 Response to pulse input of feedback system for Example 15.5 with $K = 2.5$.

which result in

```
pcl =  -4.6739
       -0.4119 + 2.4937i
       -0.4119 - 2.4937i
       -0.5024
```

As a check on the nature of the output plot, we note that the pair of complex poles has a damping ratio $\zeta = \cos[\tan^{-1}(2.4937/0.4119)] = 0.163$ and a time constant $\tau = 1/0.4119 = 2.43$ s. The steady-state value of the unit step response must be $T(0) = 25/15 = 1.60$, which is again consistent with the figure.

Bode Diagrams

As described in Section 14.3, Bode plots of a system's open-loop frequency response are often used in the analysis and design of feedback systems. MATLAB has a `bode` command that can produce magnitude and phase angle curves for models in either state-space or transfer-function form. The models may be either open-loop (no feedback active) or closed-loop. Also, the user has the option of either specifying the frequency values for which the calculations are to be done or letting MATLAB make the selection on the basis of the poles and zeros of the system's transfer function.

If the frequency values are to be specified, the user must first create a row vector of frequency values, expressed in radians per unit of time (typically, seconds). Because we use the logarithm of the frequency as the independent variable, it is customary to have the elements of the frequency list spaced logarithmically so that adjacent values in the list have the same ratio rather than the same difference. This is easily done in MATLAB by means of the `logspace` command, for which the first two arguments are the beginning and ending frequencies, expressed as powers of 10. The optional third argument is the number of points in the list, 50 being the default value.

For each of the frequencies in the list, the `bode` command calculates the magnitude and phase angle of the frequency response, expressed as a magnitude ratio and in degrees, respectively. To obtain a plot with the magnitude expressed in decibels, we must convert the magnitude ratios to decibels by taking $20 \log_{10}$ of each element. This is done by operating on the entire vector of magnitude ratios, as illustrated below. For plotting with a logarithmic horizontal scale and a linear vertical scale, MATLAB has the command `semilogx`, which is used in just the same way as the `plot` command.

There is also a `margin` command that can be used with *open-loop* models for finding the gain and phase margins, provided that they are defined for the specific model being analyzed. Recall that $|G(j\omega)H(j\omega)|$ must cross the zero-dB axis for the phase margin to exist, and $\arg[G(j\omega)H(j\omega)]$ must cross $-180°$ for the gain margin to exist.

EXAMPLE 15.6

Draw plots of the frequency response magnitude (in decibels) and phase angle (in degrees) for the closed-loop system created in Example 15.3. Use the state-space form of the model.

Solution

The steps to be taken are (1) create the list of frequencies, (2) compute the magnitude ratio and the phase angle at each frequency value, (3) convert the magnitude ratios to decibels, and (4) construct the magnitude and phase plots versus frequency. The commands that will accomplish these steps, along with some comments, follow.

```
%-- 100 freq values from 0.01 to 100 rad/s
freq = logspace(-2,2,100);
%-- magnitude in dB and phase in degrees
[mag_ratio,phase,w] ...
      = bode(A_fbk,B_fbk,C_fbk,D_fbk,1,freq);
mag_db = 20*log10(mag_ratio);
%-- plot magnitude in dB vs logarithmic frequency
subplot(211)
semilogx(freq,mag_db);
grid;xlabel('freq - rad/s');ylabel('mag - dB')
title('Closed-loop frequency-response')
%-- plot phase angle versus logarithmic frequency
subplot(212)
semilogx(freq,phase);grid
xlabel('freq - rad/s');ylabel('phase - deg')
subplot     %-- return to normal plotting format
```

The desired Bode diagram is shown in Figure 15.8.

From the upper part of Figure 15.8 we see that for low frequencies, $|T(j\omega)|$ is asymptotic to a value of just less than 5 dB. The actual value is 4.437 dB = 20 $\log_{10}(10/6)$, which can be obtained by taking the limit as $\omega \to 0$ of $T(s)$ as given by (2). The lower part of the figure shows that for low frequencies, the phase is asymptotic to zero. As ω increases, the magnitude of the frequency response of the closed-loop system remains flat until about $\omega = 0.1$ rad/s and then begins to fall off, with a slight peak near $\omega = 2$ rad/s. It continues to decrease at a slope of -20 dB/decade as the frequency increases. The phase plot exhibits a similar behavior as the frequency increases but becomes asymptotic to $-90°$ as $\omega \to \infty$.

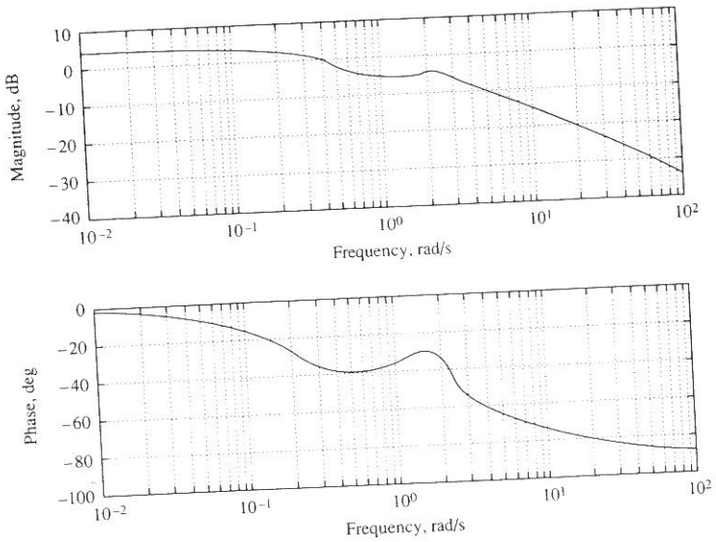


FIGURE 15.8 Bode diagram for Example 15.6.

Root-Locus Plots

In Section 14.2, we showed how a root-locus plot can be used to determine how the closed-loop poles of a feedback system will vary as the open-loop gain is varied. As with the other MATLAB commands that we are discussing, the user has several options available for the `rlocus` command. The model can be given in either transfer-function or state-space form. A list of gain values can be specified, or MATLAB will select a set that it considers appropriate. The user can have MATLAB either draw the plot without returning the numerical values of the roots or return the values without automatically drawing the plot. The region within the s-plane covered by the plot can also be specified or left to MATLAB. There is an additional command called `rlocfind` that allows the user to select any point on the locus once it has been drawn and to obtain both the value of the gain parameter corresponding to that point and the values of all n of the closed-loop characteristic roots for that value of gain.

The following example illustrates how root-locus plots can be created with MATLAB and how the gain can be determined for specific points on the locus.

▶ ***EXAMPLE 15.7***

Generate a root-locus plot for the feedback system shown in Figure 15.5. Use the plot to determine the value of the gain K for which the closed-loop system will become marginally stable.

Solution

Recognizing that the forward and feedback transfer functions, excluding the gain K, are the functions $G_1(s)$ and $G_2(s)$ that were built and connected in series in Example 15.2, we can specify the open-loop model in transfer-function form in terms of the existing polynomial coefficient vectors `num_ser` and `den_ser`. The gain K in the forward path is included automatically. Thus, to generate the root-locus plot with the region and gains determined by MATLAB, we enter the commands

```
rlocus(num_ser,den_ser)
grid;title('root-locus plot for Example 15.7')
```

The plot shown in Figure 15.9 will appear on the screen. Note that the symbol `x` has been placed at each of the four open-loop poles and the symbol `o` at the single open-loop zero.

To determine the value of K for which the complex branches cross into the right half of the s-plane, we enter

```
[k,clpoles] = rlocfind(num_ser,den_ser)
```

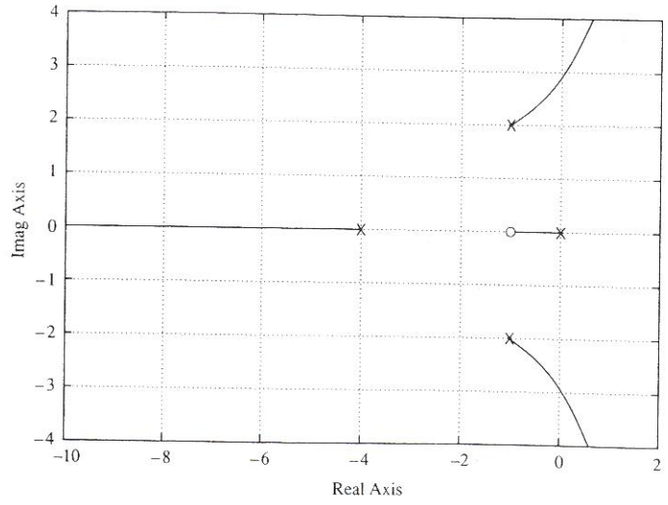


FIGURE 15.9 Root-locus plot for Example 15.7.

MATLAB will prompt the user to select the point for which the gain is to be found by clicking with the mouse directly on the root-locus plot that has just been drawn in the graphics window. We get $K^* = 5.85$.

15.3 BUILDING ACSL MODELS

In the previous two sections we showed how a digital computer can be used to obtain numerical solutions for fixed, linear models. Now we direct our attention to the equally important task of numerically solving for the responses of nonlinear models. It is not usually possible to find a closed-form solution for nonlinear dynamic systems. What we can do is numerically integrate the nonlinear differential equations that describe the system, for a specific set of inputs and initial conditions. By repeating this process for a number of inputs and initial conditions, we can develop a good understanding of the dynamic behavior of the nonlinear model. In this section we will describe the **nonlinear simulation** program ACSL, which is representative of a number of such programs.

The starting point for developing a simulation is to write the model in state-variable form. If the model has n state variables and m inputs, we write the n first-order differential equations

$$\begin{aligned} \dot{q}_1 &= f_1(q_1, \cdots, q_n, u_1, \cdots, u_m, t) \\ \dot{q}_2 &= f_2(q_1, \cdots, q_n, u_1, \cdots, u_m, t) \\ &\vdots \\ \dot{q}_n &= f_n(q_1, \cdots, q_n, u_1, \cdots, u_m, t) \end{aligned} \tag{3}$$

where the f_i can be nonlinear functions of the state variables, the inputs, and possibly the time t. If the model has p outputs, there will be p algebraic output equations expressing the outputs in terms of the state variables, inputs, and perhaps t. The key feature of (3) is that the derivatives of all of the state variables can be calculated for any instant at which the state variables and inputs are known. These derivatives can be used to determine, by numerical integration, the approximate values of the state variables at the next instant. This iterative process can be started at the initial time t_0 from a known set of initial conditions $q_i(t_0)$. At the end of the computer run we will have the solution—actually, an approximation to the true solution—at a set of discrete time points.

A variety of methods exist for doing numerical integration, and they involve different amounts of calculation and have different error characteristics. Perhaps the most commonly used is the fourth-order Runga-Kutta algorithm, which does four sets of evaluations of the state-variable derivatives for each step forward in time.

The flow chart in Figure 15.10 shows the steps that must be taken for a numerical simulation. There is a section that is done initially but does not have to be repeated. Then there is a section that is done repetitively, as time advances, until the stopping condition is satisfied. Output in numerical form can be displayed as the run progresses. At the conclusion of the calculations, plots are generated from values that have been stored during the run. As we shall see, ACSL is set up to implement exactly this scenario. The user is also able to make modifications to parameter values (such as constants and initial conditions) between runs, without having to change the computer code for the model.

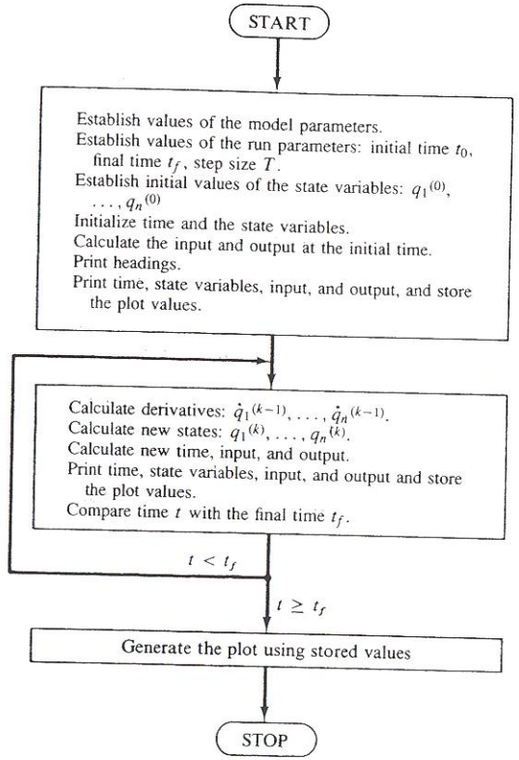


FIGURE 15.10 Operations required for the numerical solution of system models.

Two types of computer files are used for an ACSL simulation. These are the **model file**, which contains the equations that describe the system's

mathematical model, and the **command file**, which describes how the simulation is to be conducted. For example, the command file contains the names of the variables whose values are to be displayed at certain intervals as the solution progresses.

Once the model file has been prepared, it is automatically translated into Fortran code, which is then compiled and linked with the appropriate libraries to yield an executable module. When execution begins, the command file is read and its instructions are carried out. At the completion of the command file, the keyboard becomes active so that the user can enter new commands, change the values of constants, and make additional runs.

Structure of an ACSL Program

Figure 15.11 shows the structure of a simplified ACSL program that will simulate any continuous-time system we are likely to encounter. There may be other elements in a more complicated situation, but they need not concern us. Note that each keyword that begins the program or a section must be matched by an `end` statement. Any text on a line following an exclamation point is a comment and is ignored by ACSL. In the `initial` section, any calculations that need be done only once should be performed, such as calculating coefficients that are used in expressions for derivatives and calculating constants that depend on the initial conditions. Although it is not essential, it is good practice to declare all constants and assign them values here.

```
program
initial
 - declare constants and assign values
 - do initial calculations (parameters, initial values)
end ! of initial section
derivative
 - evaluate inputs
 - evaluate derivatives of state variables
 - integration statements
 - evaluate outputs
 - check termination condition
end ! of derivative section
end ! of program
```

FIGURE 15.11 Simplified structure of an ACSL program for continuous-time models.

The `derivative` section contains all the computations that must be performed at each time step. This includes testing for the stopping condition. This test usually invoives comparing the actual simulation time to an ending time. However, other types of stopping conditions can be used, such as the displacement of a mass reaching a specified value.

Integration

The heart of a simulation program is integration of the first-order differential equations of the state-variable model. To accomplish this integration, we must know the initial conditions and must evaluate the derivative of each state variable. Then we use the `integ` command with the derivative and initial condition as its arguments to obtain the values of the variable at subsequent time points.

For example, to solve the first-order nonlinear differential equation $\dot{x} + 2x^3 = 0$ with the initial condition $x(0) = 2$, we first write $\dot{x} = -2x^3$. If we then denote $\dot{x}$ by `xdot` and $x(0)$ by `xic`, we enter the statements

```
constant    xic = 2
xdot = -2*x*x*x
x = integ(xdot,xic)
```

At any point in time, the derivative `xdot` is evaluated and is used by the integration algorithm to determine the value of `x` at the next time point. The user need not be concerned with the details of these calculations for most simulation tasks, provided that the step size is kept sufficiently small. If we are using a fixed-step-size method, we can generally verify that the step size is satisfactory by running the simulation for several step sizes and comparing the results. Some of the nine built-in integration algorithms that ACSL has automatically vary the step size in order to maintain the integration error within bounds. The default is the fourth-order Runga-Kutta algorithm, which can be changed by specifying a different value for the ACSL parameter `ialg`. The variable `t` is automatically computed by ACSL as the independent time variable.

Functions

ACSL has the usual mathematical functions, including sine, cosine, tangent, square root, exponential, and logarithm. There are also functions for obtaining the absolute value, maximum, and minimum of a variable. Such common time functions as the step, ramp, and pulse are available. And a variety of nonlinear functions such as saturation, dead zone, and a switch exist.

Timing Parameters

The key timing parameter is the communication interval (`cint`), which determines how often the values of the variables are to be displayed and

stored for printing. The calculation interval—that is, the integration step size—can never exceed the communication interval. For fixed-step-size methods such as the Runga-Kutta algorithm, there are `nstp` integration steps per communication interval, and the default value of `nstp` is 10. Thus we can easily investigate the effects of integration step size by doing several runs with different values for `nstp`.

Usually a simulation run is terminated by the time variable's reaching a specified maximum value. We ensure this by using the `termt` statement in the `derivative` section as in

```
termt (t .ge. tstop,'reached final time')
```

where the value of `tstop` is set with a `constant` statement or entered by the user from the keyboard. When the time variable `t` reaches `tstop`, the run terminates and the message `reached final time` appears in the output display. More complicated stopping conditions, such as a variable's reaching a predetermined value, are also possible.

Run-Time Statements

Once the ACSL model has been entered and converted to executable code, run-time commands must be given to tell ACSL what is to be done. Typical commands are to define a title for the particular case being run, to specify some parameter values and/or initial conditions, to identify certain variables to be printed on the display as the run progresses, to identify variables whose values should be saved for plotting, and to indicate that a run should be started. Rather than discussing these statements individually, we will illustrate their use in the examples that follow.

There are several ways in which the run-time commands can be entered. First, they can be typed at the keyboard. Second, they can be read in from a file that has the same name as the model file but also includes the extension `.cmd`, which designates the command file. The third method involves the definition of blocks of run-time statements, called procedures, in the command file. When this is done, the entire block of commands can be executed merely by entering the name of the specific procedure. In fact, the procedures can have arguments that are specified by the user when invoked.

15.4 RUNNING ACSL MODELS

This section gives two examples of ACSL models and uses them to simulate the response of a dynamic system. We start with a simple linear system and then simulate a nonlinear mechanical system that has both translational and rotational motion with a nonlinear stiffness characteristic.

▶ ***EXAMPLE 15.8***

Create an ACSL model for the second-order linear system described by the state-variable equations

$$\dot{q}_1 = q_2$$
$$\dot{q}_2 = -aq_1 - bq_2 + x(t)$$
$$y = q_1 + q_2$$

where q_1 and q_2 are the state variables, y is the output, and $x(t)$ is the input. Simulate the zero-state response to a pulse of height 2.0 starting at $t = 0.5$ s and ending at $t = 4.5$ s. The simulation should be done for the parameter values $a = 1.5$ and $b = 2.0$ and should cover the time interval $0 \leq t \leq 10$ seconds.

Solution

An ACSL model file that implements this linear model is shown in Figure 15.12. Note that constants are defined in the `initial` section for the coefficients of the differential equation, the initial conditions, the time at which the input begins and ends, and the time at which the simulation terminates. Because these parameters are defined by `constant` statements, any of them can be changed at run time from the keyboard, from the command file, or with a procedure call.

The remainder of the model is contained in the `derivative` section: the expressions for the two state-variable derivatives, the input, the two

```
program example15-8
initial !-- initialize parameters and constants
constant   a = 1.5,  b = 2.0
constant   q1ic = 0.0,  q2ic = 0.0
constant   tstep1 = 0.5,  tstep2 = 4.5
constant   xamp = 2.0,  tstop = 10.0
end ! of initial section
derivative !-- create dynamic model
x = xamp*(step(tstep1)-step(tstep2))
q1dot = q2     ! derivatives of states
q2dot = x - (a*q1 + b*q2)
q1 = integ(q1dot,q1ic)     ! state variables
q2 = integ(q2dot,q2ic)
y = q1 + q2     ! output
termt(t .ge. tstop,'reached final time')
end  ! of derivative section
end  ! of program
```

FIGURE 15.12 ACSL model file for Example 15.8.

`integ` statements that provide the integration of the derivatives, the output expression, and the stopping condition. Note that there is nothing in the model file that says what is to be done with the model—the model file just defines the model itself. The reader must understand this important distinction in order to prepare simulations of more complicated systems properly.

To run the simulation, we will use the command file shown in Figure 15.13. The command `set title` makes the message between the apostrophes start at the beginning of the first line at the top of the output plot; `set title(41)` shifts the corresponding message by 41 spaces (which in this case is at the start of the second line). The fourth line in Figure 15.13 sets the plotting device to be the standard one for the particular computer system in use. Some of the other options for the variable `devplt` are 3 for Tektronix format, 5 for Postscript format, and 6 for an X-windows driver. Setting the ACSL variable `nciout` to 10 causes the printed output to appear for only every tenth communication interval, thereby reducing the volume of the output. The `output` command tells ACSL which of the variables and parameters are to be printed at time intervals of `nciout*cint`. The `prepar` command indicates which variables are to have their values saved every `cint` units of time so that they can be plotted if desired. Note that the variable `t` appears in both the `output` and `prepar` lists. The simulation time `t` is the *only* variable that does not have to be defined separately.

Unless instructed otherwise, ACSL draws solid grid lines, which can be confused with the response curves. By setting the value of the ACSL plotting parameter `gltplt` to 200, we obtain dashed grid lines.

```
! Command file for running example 15-8
set title = 'Ex 15.8: linear example with 2 states'
set title(41) = 'a=1.5, b=2.0'
set devplt = 1 ! use default plot output device
set nciout = 10 ! print every 10th comm interval
output  t,x,y,q1,q2,a,b,xamp ! print these
prepare t,x,y,q1,q2 ! store for later plotting
set gltplt=200      ! dashed grid lines
start
!--- plot output and input
plot/xaxis=t/xtag=', sec' y,x/hi=2.5/lo=-0.5
pause
!--- plot the states
plot  q1,q2
!--- keyboard becomes active here
```

FIGURE 15.13 Command file for Example 15.8.

The `start` command tells ACSL to begin a simulation that will terminate when the logical expression that is the first argument of the `termt` statement in the `derivative` section of the model file is satisfied. At the conclusion of the run, the command file tells ACSL to draw two plots, with a pause between them. The first plot will show the output `y` and the input `x` versus time; it appears in Figure 15.14(a). The axes are scaled automatically unless the user specifies the scale. The command for the first plot makes the scale for `x` extend from −0.5 to 2.5.

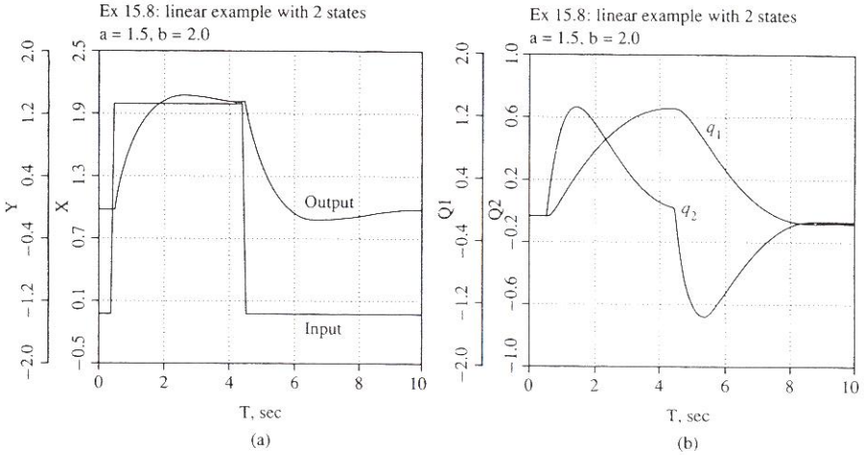


FIGURE 15.14 Responses to pulse input computed by the ACSL simulation of Example 15.8. (a) Output and input. (b) State variables.

After the user has viewed the first plot, a carriage return initiates the second plot, which shows the two state variables (Figure 15.14b). To differentiate between two or more curves in the same plot, ACSL has commands to add symbols or to use different types of lines. The labels on the individual curves in Figure 15.14 were added by hand.

When the last command in the file has been completed, the keyboard will be active, and the user can enter further commands manually. For example, the coefficients `a` and `b` might be changed and the simulation repeated. To do this and to view just the plot of the output and input, the user would enter

```
set a=3.0, b=1.0
start
plot y,x
```

It is not necessary to enter the other run-time commands that were in the file, because their settings are retained until modified by the user.

Having illustrated the main features of ACSL in the context of a simple linear system, we now show how it can be used to obtain responses of a nonlinear system that cannot be handled analytically.

EXAMPLE 15.9

Develop an ACSL simulation of the mechanical system shown in Figure 15.15(a) and use it to compute the zero-input response and the zero-state response to $\tau_a(t)$. Include the force due to gravity on the mass M in all parts of the problem. The front end of the shaft attached to J_2 is fixed in the wall, and the shaft obeys the nonlinear torque-displacement relationship $\tau_K = \alpha|\theta_2|\theta_2$. The gear ratio is defined to be $N = R_2/R_1$. The outputs are x, the displacement of the mass, and f_K, the tensile force exerted by the translational spring. Because of the presence of the flexible cable, the force f_K is zero whenever the spring is not in tension. The parameter values are

$J_1 = 0.25$ kg·m^2	$J_2 = 12$ kg·m^2	$M = 50$ kg
$R_1 = 0.075$ m	$R_2 = 0.30$ m	$K = 1500$ N/m
$B = 25$ N·m·s/rad	$\alpha = 215$ N·m/rad^2	$A = 50$ N·m

The gravitational constant is 9.807 m/s^2. Angles are measured in radians, and translational displacements are in meters. When $\theta_2 = x = 0$, both the torsional shaft and the translational spring are undeflected. The numerical values of the input torque and the state variables should be printed. The translational displacement x and the applied torque $\tau_a(t)$ should be plotted.

The following specific tasks are to be accomplished:

1. Write the model as a set of first-order differential equations (not necessarily in state-variable form) and use them to solve analytically for the equilibrium values $\bar{\theta}_2$ and $\bar{x}$ when the applied torque $\bar{\tau}_a$ is zero.
2. Develop an ACSL program to simulate the response to an applied torque that is a step function of height A commencing at 0.5 s, where the initial conditions for θ_2 and x are specified relative to their equilibrium values and where the initial velocities are always zero.
3. Simulate the zero-input response when the mass is released from the position where $\theta_2 = x = 0$. Plot the displacement x versus time.
4. Simulate the response when the amplitude of the applied torque is $A = 50$ N·m and the initial conditions are $\theta_2(0) = \bar{\theta}_2$ and $x(0) = \bar{x}$. Plot the input torque $\tau_a(t)$ and the displacement x versus time.
5. Repeat the previous step when the damping coefficient B is increased to 200 N·m·s/rad.

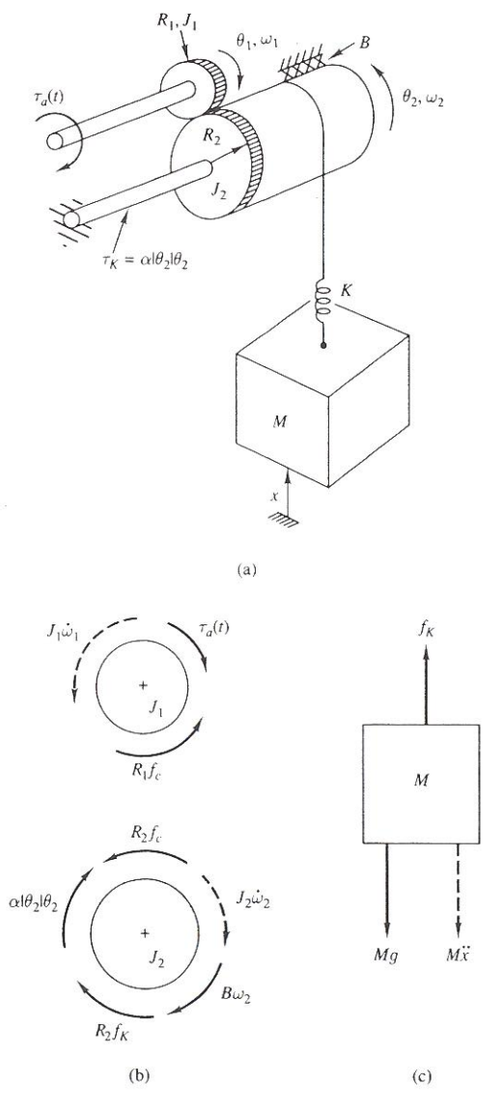


FIGURE 15.15 (a) Nonlinear mechanical system for Example 15.9. (b), (c) Free-body diagrams.

Solution

To develop the modeling equations, we first draw the free-body diagrams shown in parts (b) and (c) of Figure 15.15. Summing moments on the two rotational bodies and summing forces on the mass yield the following equations:

$$J_1\dot{\omega}_1 + R_1 f_c - \tau_a(t) = 0 \tag{4a}$$

$$J_2\dot{\omega}_2 + B\omega_2 + \alpha|\theta_2|\theta_2 - R_2 f_c + R_2 f_K = 0 \tag{4b}$$

$$M\dot{v} - f_K + Mg = 0 \tag{4c}$$

where f_c denotes the contact force on the gears and $\omega_1 = N\omega_2$, where the gear ratio has been defined as $N = R_2/R_1$. Because the translational spring K is attached to a flexible cable, it is unable to exert a compressive force. Hence the force f_K obeys the nonlinear relationship

$$f_K = \begin{cases} K(R_2\theta_2 - x) & \text{for } R_2\theta_2 \geq x \\ 0 & \text{for } R_2\theta_2 < x \end{cases} \tag{5}$$

To obtain the model in a form suitable for ACSL, we select a set of state variables and write an equation for each of their derivatives. However, because of the complexity of the expression for the force f_K, we do not attempt to express these derivatives as explicit functions of only the state variables and the input. Rather, we use (5) to compute f_K and then use it where needed in the expressions for the derivatives. The steps to be taken are as follows:

1. Select x, v, θ_2, and ω_2 as state variables.
2. Use the relationship $\omega_1 = N\omega_2$ to eliminate ω_1 in (4a).
3. Combine (4a) and (4b) to eliminate the contact force f_c and define the equivalent moment of inertia $J_{eq} = J_2 + N^2 J_1$.
4. Rearrange the terms to obtain expressions for the derivative of each of the four state variables.

We obtain the following set of first-order differential equations for the derivatives of the state-variables.

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= (f_K - Mg)/M \\ \dot{\theta}_2 &= \omega_2 \\ \dot{\omega}_2 &= -[\alpha|\theta_2|\theta_2 + B\omega_2 + R_2 f_K - N\tau_a(t)]/J_{eq} \end{aligned} \tag{6}$$

where f_K is given by (5).

At any equilibrium position, the translational spring will be in tension and all derivatives will be zero. Then (5) and (6) become

$$\begin{aligned} \bar{f}_K &= K(R_2\bar{\theta}_2 - \bar{x}) \\ \bar{f}_K - Mg &= 0 \\ \alpha|\bar{\theta}_2|\bar{\theta}_2 + R_2\bar{f}_K - N\bar{\tau}_a &= 0 \end{aligned}$$

from which

$$\begin{aligned} |\bar{\theta}_2|\bar{\theta}_2 &= (N\bar{\tau}_a - R_2 Mg)/\alpha \\ \bar{x} &= R_2\bar{\theta}_2 - Mg/K \end{aligned} \tag{7}$$

An ACSL model file that can be used with a command file or with user-entered run-time commands to obtain the required responses is given in Figure 15.16. In the `initial` section, the values of the physical parameters are specified, and a number of quantities are computed for use in the `derivative` section. Note that the equilibrium values $\bar{\theta}_2$ and $\bar{x}$ for $\bar{\tau}_a = 0$ are computed from (7). The initial conditions for θ_2 and x are then defined relative to this equilbrium condition. Various constants that appear in the differential equations, including the gear ratio N and the equivalent moment of inertia J_{eq}, are evaluated once in this section. Finally, the communication interval is set to 0.05 s so that the plots will be smooth. This also forces the default integration step size to be 0.005 s, which is a satisfactory value for this problem.

In the `derivative` section, all of the time-varying quantities are evaluated and the terminal condition is tested. Equation (5), which switches between two different expressions for f_K when $R_2\theta_2 - x$ changes sign, is implemented by a real switch designated by `rsw` in the file. Note that each of the derivatives is evaluated explicitly, although it is permissible to use the expression for a derivative as the first argument in the `integ` statement. However, the approach taken here is preferable because the values of the derivatives can be printed or plotted. Although we generally do not need to see a derivative of a state variable explicitly, this information can be very helpful in debugging a simulation that is not correct. Likewise, the initial conditions of the states are defined explicitly, and their symbols, rather than the values themselves, are used as the arguments of the `integ` statements.

A command file to run the model is shown in Figure 15.17. The first portion of the file contains commands that set the title, the frequency at which the output is printed, the variables and constants whose values are to be printed each `cint*nciout` seconds, and the variables whose values will be available for plotting at the end of the run. The last portion of the file establishes a `procedure` called `task3` that will simulate and plot the zero-input response (the third task in the example statement) when the user enters the command `task3`. By examining the command file, we see that the plot title is modified to reflect the specific conditions in effect, the proper initial conditions are set, the amplitude of the input torque is set to zero, the solution is started, the minimum and maximum values of the spring force

```
program ex15-9 ! trans + rot mechanical system
initial
constant J1=0.25, J2=12.0      ! kg-m^2
constant M=50.     ! kg
constant R1=0.075, R2=0.30       ! m
constant K=1500.     ! N/m
constant B=25.      ! N-m-s/rad
constant alpha=215.    ! N-m/rad^2
constant G=9.807      ! m/s^2
constant delx=0, delth2=0     ! ICs rel to equil
constant vic=0, w2ic=0
constant Ataua=0, tauabar=0, tstop=10.0, tstep=0.5
!--- compute nominal values, ICs, constants
delta = N*tauabar - R2*M*G
th2bar = sign(sqrt(abs(delta/alpha)),delta)
xbar = R2*th2bar-M*G/K
xic = xbar + delx
th2ic = th2bar + delth2
N = R2/R1
Jeq = J2+(N**2)*J1    ! equiv moment of inertia
cinterval cint=0.05 !-- communication interval
end    ! of initial section
derivative
taua = Ataua*step(tstep)
xdot = v
vdot = (fK-M*G)/M
th2dot = w2
w2dot = (-alpha*th2*abs(th2) - R2*fK &
             - B*w2 + N*taua)/Jeq
x = integ(xdot,xic)
v = integ(vdot,vic)
th2 = integ(th2dot,th2ic)
w2 = integ(w2dot,w2ic)
fK = rsw(R2*th2.ge.x, K*(R2*th2 - x), 0)
termt (t.ge.tstop,'reached final time')
end    ! of derivative section
end    ! of program
```

FIGURE 15.16 ACSL model file for Example 15.9.

f_K are displayed by the `range` command, and the displacement x is plotted versus time. The command file could be expanded to define procedures for carrying out the simulations defined in the fourth and fifth tasks of the list, but no such instructions are included in the figure.


```
set title = 'Example 15.9'
set ialg=4 ! second-order Runga-Kutta algorithm
set symcpl=.t.,npccpl=100    ! symbols on plots
set nciout=20 ! output every 20th comm interval
output t,x,v,th2,w2,taua,fK,alpha,K,B,Ataua
prepar t,x,th2,taua,fK ! store for plotting
plot/xaxis=t
procedure task3    ! zero-input response
  set title(13) = ', Task 3'
  set title(41) = 'alpha=220, K=1500, B=25'
  set title(81) = &
          'delx=-xbar, delth2=-th2bar, Ataua=0'
  set delx=0.5751,delth2=0.8272 ! x(0)=th2(0)=0
  set Ataua=0
  start
  range fK ! make sure fK stays nonnegative
  plot x/hi=0.2/lo=-0.8
end    ! of task3
```

FIGURE 15.17 ACSL command file for Example 15.9.

Figure 15.18 shows the zero-input response to an initial displacement of the mass that starts at $x = 0$ and settles to its equilibrium (for $\bar{\tau}_a = 0$) value of -0.5751 m. Note that the oscillations take over 10 s to damp out.

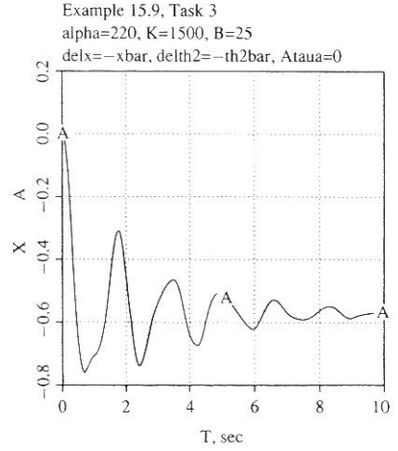


FIGURE 15.18 Zero-input response of the mechanical system in Example 15.9.

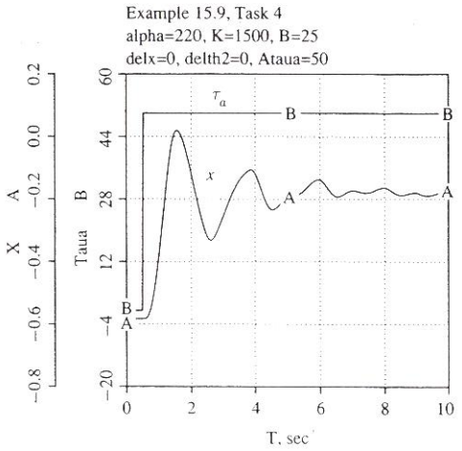


FIGURE 15.19 Zero-state response to a step input for the mechanical system in Example 15.9 with $B = 25$.

Figure 15.19 shows the zero-state response (where zero-state refers to the equilibrium condition with gravity but with no applied torque acting) when $\tau_a(t)$ undergoes a step change of 50 N·m at $t = 0.5$ s and when the rotational damping coefficient is $B = 25$ N·m·s/rad. The displacement of

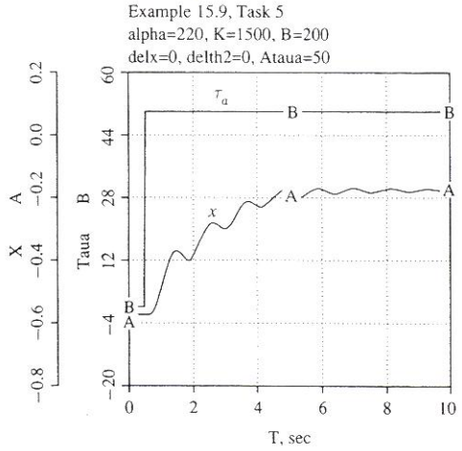


FIGURE 15.20 Zero-state response to a step input for the mechanical system in Example 15.9 with $B = 200$.

the mass starts at its unforced equilibrium value of −0.5751 m and settles out to a value of −0.1777 m, following a series of decaying oscillations. This behavior is consistent with the arrows in Figure 15.15(a) that show the positive senses of these variables.

Figure 15.20 shows the response under the same conditions as before, except that the rotational viscous damping coefficient B has been increased from 25 to 200 N·m·s/rad. Comparing Figure 15.20 with 15.19, we see that the starting and ending values of the displacement remain the same. The rise of the mass is substantially slower, and the amplitude of the oscillations has been reduced. However, the oscillations take about the same amount of time to damp out, because the increased damping directly affects only the drum J_2. The mass M has no damping force acting directly on it.

SUMMARY

Computer methods are required at some stage in the design of all but the simplest systems. In this chapter, we have illustrated key features of MATLAB and ACSL, two widely available software packages. ACSL can be introduced and applied as early as Chapter 3. Because important aspects of MATLAB are based on transfer functions, its use would normally be deferred until Chapter 8. Both programs are important tools for the analysis and design of dynamic systems.

For linear systems, MATLAB models can be created from the transfer functions or from the matrix state-variable equations. We can combine the models for subsystems that are connected in various configurations. We can then ask for the time responses to step functions, impulses, and other inputs. We can also obtain Bode diagrams for the frequency-response function of any part of the system. In the design of feedback systems, two important graphical aids based on the open-loop transfer function are Bode diagrams and root-locus plots. In Chapter 14 we used MATLAB to produce many such plots.

ACSL is used to simulate nonlinear systems. We first build a model file, usually from the state-variable equations describing the system. We also need a command file (or some alternative way to enter run-time commands) to tell ACSL how the simulation is to be done and how the results are to be presented. Even for a nonlinear system, it is easy to plot the responses to a variety of inputs and initial conditions, thereby gaining a good understanding of the system's dynamic behavior under a wide range of conditions.

PROBLEMS

In Problems 15.1 through 15.16, use MATLAB or another suitable software package. You should produce a diary file that shows the input, the numerical results, and (where appropriate) computer-generated plots.

15.1 The transfer function of a linear system has zeros at $s = -1$, $-4+j2$, $-4-j2$, poles at $s = -0.6$, $-1+j8$, $-1-j8$, -15, and a gain of 2.

a) Obtain the transfer function as a ratio of polynomials.

b) Find the four matrices that describe the state-space model.

c) Compute the zeros, poles, and gain of the transfer function from the state-space matrices.

15.2 The transfer function of a linear system is

$$T(s) = \frac{2s^2 + 4s + 1}{3s^4 + 8s^3 + 9s^2 + 10s}$$

a) Determine the zeros, poles, and gain of the transfer function.

b) Find the four matrices that describe the state-space model.

c) Compute the numerator and denominator polynomials of the transfer function from the state-space matrices.

*** 15.3** The state-space matrices for a linear system are

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

a) Obtain the transfer function as a ratio of polynomials.

b) Determine the zeros, poles, and gain of the transfer function.

15.4 **a)** Use (13.37) to develop a state-space model of the system that was studied in Example 13.7, whose block diagram is shown in Figure 13.18(b).

b) Convert this model to transfer-function form and compare the result with the differential equation given in the example statement.

*** 15.5** **a)** Use (13.41a), (13.41b), (13.41c), and (13.42) to develop a state-space model of the system that was studied in Example 13.8, whose block diagram is shown in Figure 13.19(c).

b) Convert this model to transfer-function form and compare the result with the differential equation given in the example statement.

15.6 Obtain state-space models for the individual subsystems defined in Example 15.2. Then, for the series combination, obtain both the zero-pole and transfer-function forms of the overall transfer function $G_{12}(s)$. Verify that the results agree with those found in the example.

*** 15.7** For the series connection shown in Figure P15.7, obtain the zero-pole model and the transfer-function model.

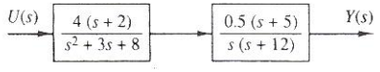


FIGURE P15.7

15.8 **a)** Use the series and parallel commands of MATLAB to build the model represented in Figure 13.6 in transfer-function form.

b) Determine the zeros, poles, and gain of $T(s) = Z(s)/U(s)$.

15.9 Repeat Problem 15.8 for the block diagram shown in Figure P15.9.

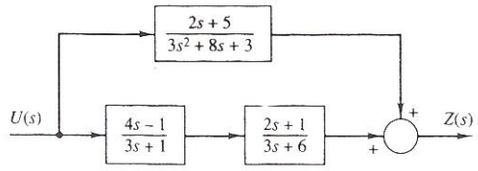


FIGURE P15.9

15.10 For the feedback connection shown in Figure P15.10, obtain the model for the closed-loop system in zero-pole form, transfer-function form, and state-space form.

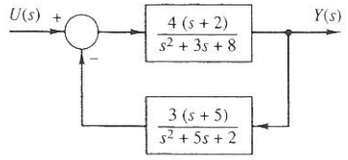


FIGURE P15.10

15.11 Repeat Problem 15.10 when the negative sign at the summing junction is changed to a plus sign—that is, when the feedback is positive rather than negative.

*** 15.12** Repeat Problem 15.10 for the feedback system shown in Figure P15.12.

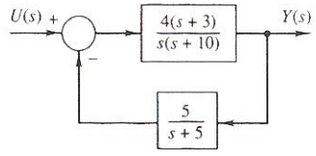


FIGURE P15.12

15.13 Obtain plots of the unit step and unit impulse responses for the linear system given in Problem 15.2.

15.14 Obtain plots of the unit step and unit impulse responses for the linear system given in Problem 15.3.

*** 15.15** Obtain the zero-state response of the system described in Problem 15.2 to the input shown in Figure P15.15 over the interval $0 \leq t \leq 50$ s. Plot the input and the output on the same axes.

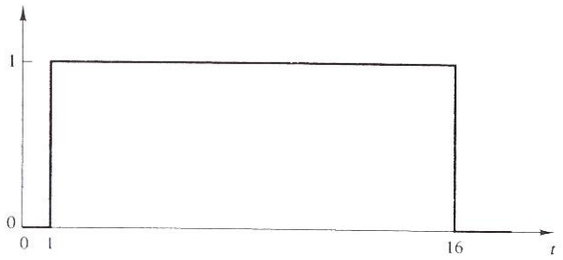


FIGURE P15.15

15.16 Obtain the zero-state response of the system described in Problem 15.3 to the input shown in Figure P15.16 over the interval $0 \leq t \leq 20$ s. Plot the input and the output on the same axes.

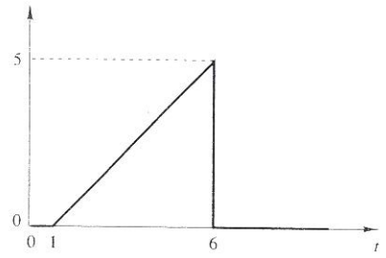


FIGURE P15.16

15.17 Obtain the Bode plots shown in Figure 14.20.

15.18 **a)** Obtain the Bode plot for the system that was considered in Examples 14.3, 14.4, and 14.16 when $K = 10$.
b) Determine the gain and phase margins and the frequencies at which they are defined.
c) Use the gain margin to determine the maximum value of K for which the closed-loop system is stable.

* **15.19** Repeat Problem 15.18 for the system that was studied in Example 14.5 when $K = 150$.

15.20 Repeat Problem 15.18 for the system that was studied in Example 14.7 when $K = 40$.

15.21 **a)** Obtain a root-locus plot similar to the one shown in Figure 14.9 for the feedback system studied in Example 14.3.
b) Determine the value of K that will result in a complex closed-loop pole having an imaginary part of +2. Find the other closed-loop poles that correspond to this value of gain.

15.22 **a)** Obtain a root-locus plot similar to the one shown in Figure 14.10 for the feedback system studied in Example 14.5.
b) Determine the value of K that will result in a complex closed-loop pole having a real part of -2. Find the other closed-loop poles that correspond to this value of gain.

15.23 Repeat Example 14.6 when $\alpha = 20$. Determine the maximum value of K for which the closed-loop system is stable.

* **15.24** **a)** Obtain a root-locus plot similar to the one shown in Figure 14.12 for the feedback system studied in Example 14.7.
b) Determine the value of K that will result in a complex closed-loop pole having an imaginary part of +2. Find the other closed-loop poles that correspond to this value of gain.

In Problems 15.25 through 15.32, use the nonlinear simulation language ACSL or any equivalent software package.

* **15.25** Consider the linear system that was discussed in Example 13.7, whose block diagram is shown in Figure 13.18(b). Simulate the zero-state response of the system to the inputs described below. In each case, generate two plots. The first plot should show the input and the output, and the second should show each of the state variables.
a) A unit step function starting at $t = 1$ s.
b) The function shown in Figure P15.16.

15.26 Repeat Problem 15.25 for the linear system that was discussed in Example 13.8, whose block diagram is shown in Figure 13.19(c). Use the following inputs.
a) A unit ramp function starting at $t = 1$ s.
b) The function shown in Figure P15.26.

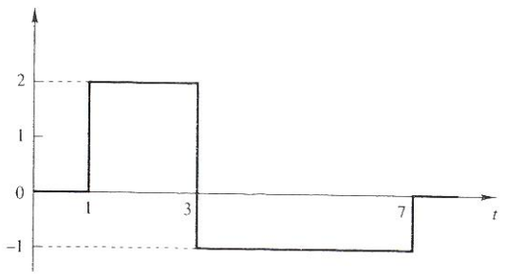


FIGURE P15.26

15.27 Repeat Problem 15.25 for the linear system whose transfer function is

$$T(s) = \frac{3s^2 + 4s + 6}{2s^3 + 2s^2 + 5s + 4}$$

* **15.28** Simulate the response of the second-order nonlinear system considered in Example 9.7 and described by (9.23) over the interval $0 \leq t \leq 10$ s. Verify by