Example 8.13 illustrates that the closed-loop compensation of digital systems in the z-domain can be carried out similarly to that of analog systems in the s-domain, which was discussed in Section 2.12. You can extend the remaining techniques presented in Section 2.12, namely, *lead compensation*, *lag compensation*, and *lead-lag compensation* for the z-domain design of digital systems.

8.8 State-Space Modeling of Multivariable Digital Systems

Consider a single-input, single-output digital system described by the pulse transfer function, $G(z)$. Instead of having to find the partial fraction expansion of $Y(z) = G(z)U(z)$ for calculating a system's response, an alternative approach would be to express the pulse transfer function, $G(z)$, as a *rational function* in z, given by $G(z) = num(z)/den(z)$, and then obtaining the inverse z-transforms of both sides of the equation, $Y(z)den(z) = U(z)num(z)$. Let us write a general expression for a rational pulse transfer function, $G(z)$, as follows:

$$G(z) = (b_m z^m + b_{m-1} z^{m-1} + \cdots + b_1 z + b_0)/(z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0) \quad (8.58)$$

We can write $Y(z)den(z) = U(z)num(z)$ as follows:

$$Y(z)(z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0) = U(z)(b_m z^m + b_{m-1} z^{m-1} + \cdots + b_1 z + b_0) \quad (8.59)$$

or

$$\begin{aligned} &z^n Y(z) + a_{n-1} z^{n-1} Y(z) + \cdots + a_1 z Y(z) + a_0 Y(z) \\ &\quad = b_m z^m U(z) + b_{m-1} z^{m-1} U(z) + \cdots + b_1 z U(z) + b_0 U(z) \end{aligned} \quad (8.60)$$

Recall the time-shift property of the z-transform given by Eq. (8.13). For a single translation in time, $z\{y(kT + T)\} = zY(z) - zy(0^-)$, where $Y(z)$ is the z-transform of $y(kT)$. Taking another step forward in time, we can similarly write $z\{y(kT + 2T)\} = z\{y(kT + T)\} - zy(T^-) = z^2 Y(z) - z^2 y(0^-) - zy(T^-)$, and for n time steps the corresponding result would be $z\{y(kT + nT)\} = z^n Y(z) - z^{n-1} y(0^-) - z^{n-2} y(T^-) - \cdots - zy\{n - 1)T^-\}$. Similarly, for the input we could write $z\{u(kT + mT)\} = z^m U(z) - z^{m-1} u(0^-) - z^{m-2} u(T^-) - \cdots - zu\{(m - 1)T^-\}$. The values of the output at n previous instants, $y(0^-), y(T^-), \ldots, y\{(n - 1)T^-\}$ are analogous to the *initial conditions* that must be specified for solving an nth order continuous-time differential equation. In addition, the input, $u(kT)$, is assumed to be known at the m previous time instants, $u(0^-), u(T^-), \ldots, u\{(m - 1)T^-\}$. For simplicity, let us assume that both input and output are zero at the n and m previous time instants, respectively, i.e. $y(0^-) = y(T^-) = \cdots = y\{(n - 1)T^-\} = u(0^-) = u(T^-) = \cdots = u\{(m - 1)T^-\} = 0$. Then, it follows that $zY(z) = z\{y(kT + T)\}, \ldots, z^n Y(z) = z\{y(kT + nT)\}$, and $zU(z) = z\{u(kT + T)\}, \ldots, z^m U(z) = z\{u(kT + mT)\}$. Hence, we can easily take the inverse z-transform of Eq. (8.60) and write

$$\begin{aligned} y(kT + nT) = &-a_{n-1} y(kT + nT - T) - \cdots - a_1 y(kT + T) - a_0 y(kT) \\ &+ b_0 u(kT) + b_1 u(kT + T) + \cdots + b_m u(kT + mT) \end{aligned} \quad (8.61)$$

Equation (8.61) is called a *difference equation* (digital equivalent of a *differential equation* for analog systems) of order n. It denotes how the output is *propagated* in time by n sampling instants, given the values of the output at all *previous* instants, as well as the values of the input at all instants up to $t = kT + mT$. Note that for a proper system, $m \leq n$, a simplified notation for Eq. (8.61) is obtained by dropping T from the brackets, and writing the difference equation as follows:

$$y(k+n) = -a_{n-1}y(k+n-1) - \cdots - a_1 y(k+1) - a_0 y(k) \\ + b_0 u(k) + b_1 u(k+1) + \cdots + b_m u(k+m) \tag{8.62}$$

where $y(k+n)$ is understood to denote $y(kT+nT)$, and so on. Solving the difference equation, Eq. (8.62), for the discrete time output, $y(k)$, is the digital equivalent of solving an nth order differential equation for the output, $y(t)$, of an analog system. The most convenient way of solving an nth order difference equation is by expressing it as a set of n first order difference equations, called the *digital state-equations* (analogous to the state-equations of the continuous time systems).

For simplicity, let us assume that $n = m$ in Eq. (8.62), which can then be written as follows:

$$y(k+n) = -a_{n-1}y(k+n-1) - \cdots - a_1 y(k+1) - a_0 y(k) \\ + b_0 u(k) + b_1 u(k+1) + \cdots + b_n u(k+n) \tag{8.63}$$

To derive a digital state-space representation for the digital system described by Eq. (8.63), let us draw a *schematic diagram* for the system – shown in Figure 8.9–in a manner similar to Figure 3.4 for an nth order analog system. The drawing of the schematic diagram is made simple by the introduction of a *dummy variable*, $q(k)$, in a manner similar to the analog system of Example 3.6 (since the right-hand side of Eq. (8.63) contains time delays of the input), such that

$$q(k+n) + a_{n-1}q(k+n-1) + \cdots + a_1 q(k+1) + a_0 q(k) = u(k) \tag{8.64}$$

and

$$b_n q(k+n) + b_{n-1}q(k+n-1) + \cdots + b_1 q(k+1) + b_0 q(k) = y(k) \tag{8.65}$$

In Figure 8.9, the *integrator* of Figure 3.4 is replaced by a *delay element* (denoted by a rectangle inscribed with $\mathcal{D}$). While the output, $x(t)$, of an integrator is the time-integral of its input, $x^{(1)}(t)$, the output, $x(k)$, of a delay element is the input, $x(k+1)$, *delayed* by one sampling interval. Note from Eq. (8.13) that the pulse transfer function of a delay element is $1/z$, while the transfer function of an integrator is $1/s$. Thus, a delay element is the digital equivalent of an integrator. We can arbitrarily select the state variables, $x_1(k), x_2(k), \ldots, x_n(k)$, to be the *outputs* of the delay elements, *numbered from the right*, as shown in Figure 8.9. Therefore, $x_1(k) = q(k), x_2(k) = q(k+1), \ldots, x_n(k) = q(k+n-1)$, and the digital state-equations are the following:

$$x_1(k+1) = x_2(k) \tag{8.66a}$$

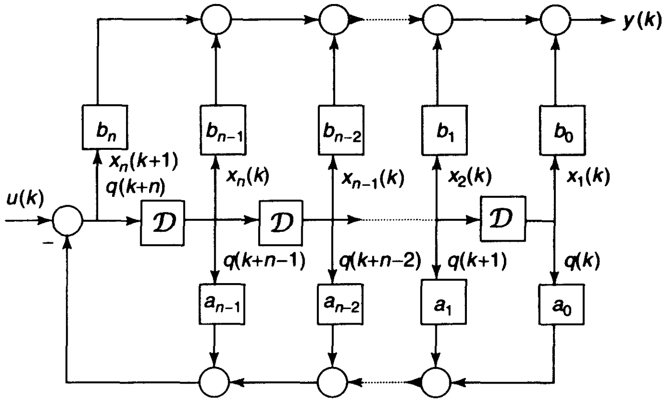

Figure 8.9 Schematic diagram for an nth order digital system

$$x_2(k+1) = x_3(k) \tag{8.66b}$$

$$\vdots$$

$$x_{n-1}(k+1) = x_n(k) \tag{8.66c}$$

$$x_n(k+1) = -[a_{n-1}x_n(k) + a_{n-2}x_{n-1}(k) + \cdots + a_0x_1(k) - u(k)] \tag{8.66d}$$

The output equation is obtained from the summing junction at the top of Figure 8.9 as follows:

$$y(k) = b_0x_1(k) + b_1x_2(k) + \cdots + b_{n-1}x_n(k) + b_nx_n(k+1) \tag{8.67}$$

and substituting Eq. (8.66d) into Eq. (8.67) the output equation is expressed as follows:

$$y(k) = (b_0 - a_0b_n)x_1(k) + (b_1 - a_1b_n)x_2(k) + \cdots + (b_{n-1} - a_{n-1}b_n)x_n(k) + b_nu(k) \tag{8.68}$$

The matrix form of the digital state-equations is the following:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ \vdots \\ x_{n-1}(k+1) \\ x_n(k+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_2 & \dots & -a_{n-1} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_{n-1}(k) \\ x_n(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u(k) \tag{8.69}$$

and the output equation in matrix form is as follows:

$$y(k) = [(b_0 - a_0b_n)(b_1 - a_1b_n)\dots(b_{n-1} - a_{n-1}b_n)] \begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_n(k) \end{bmatrix} + b_nu(k) \tag{8.70}$$

On comparing Eqs. (8.69) and (8.70) with Eqs. (3.59) and (3.60), respectively, we find that the digital system has been expressed in the *controller companion form.* Alternatively, we can obtain the *observer companion form* and the *Jordan canonical form* using the same approach as given in Chapter 3 for analog systems.

Whereas we used a single-input, single-output system to illustrate the digital state-space representation, the main utility of digital state-space representations lie in modeling *multivariable* digital systems. A general multivariable, linear, time-invariant digital system can be expressed by the following state-space representation:

$$\mathbf{x}(k+1) = \mathbf{A_d}\mathbf{x}(k) + \mathbf{B_d}\mathbf{u}(k) \tag{8.71}$$

$$\mathbf{y}(k) = \mathbf{C_d}\mathbf{x}(k) + \mathbf{D_d}\mathbf{u}(k) \tag{8.72}$$

where $\mathbf{A_d}$, $\mathbf{B_d}$, $\mathbf{C_d}$, and $\mathbf{D_d}$ are the constant coefficient matrices, and $\mathbf{x}(k)$ and $\mathbf{u}(k)$ are the digital state-vector and the digital input vector, respectively, at the kth sampling instant. The subscript d on the coefficient matrices is used to indicate a digital state-space representation (as opposed to an analog state-space representation, which is denoted by **A**, **B**, **C**, **D**). All the properties of the state-space representation as discussed in Chapter 3 for analog systems – namely linear transformation, system characteristics, and block-building – apply for the digital state-space representations, *with the crucial difference* that the s-plane for analog systems is transformed into the z-plane for digital systems. For example, a *multivariable digital state-space* representation given by the coefficient matrices $\mathbf{A_d}$, $\mathbf{B_d}$, $\mathbf{C_d}$, and $\mathbf{D_d}$ has a *pulse transfer matrix*, $\mathbf{G}(z) = \mathbf{C_d}(z\mathbf{I} - \mathbf{A_d})^{-1}\mathbf{B_d} + \mathbf{D_d}$.

Example 8.14

Let us find a state-space representation for the digital system of Example 8.12. The pulse transfer function of the system is $G(z) = z/(z^2 - 0.7z + 0.3)$. We can employ the MATLAB (CST) function *ss* to get a state-space representation of the digital system as follows:

```
>>num = [1 0];den = [1 -0.7 0.3];sys=tf(num,den);sys=ss(sys) <enter>

a =
              x1       x2
       x1     0.7      -0.6
       x2     0.5        0

b =
              u1
       x1     1
       x2     0

c =
              x1       x2
       y1     1        0

d =
              u1
       y1     0
```

The eigenvalues of the digital state-dynamics matrix, $\mathbf{A}_d$, must be the *poles* of the system in the z-domain. We can determine the eigenvalues, natural frequencies, and damping ratios associated with $\mathbf{A_d}$ for a sampling interval, $T = 0.1$ second, by using the MATLAB (CST) function *ddamp* as follows (*ddamp* allows the use of the state-dynamics matrix as an input, instead of the denominator polynomial of $G(z)$):

```
>>[Ad,Bd,Cd,Dd]=ssdata(sys);ddamp(Ad,0.1) <enter>

Eigenvalue         Magnitude Equiv. Damping Equiv. Freq. (rad/sec)
 0.3500+0.4213i 0.5477     0.5657         10.6421
 0.3500-0.4213i 0.5477     0.5657         10.6421
```

Note that these values are the same as those obtained in Example 8.12. Let us find the *Jordan canonical form* of the digital system using the command *canon* as follows:

```
>>sysc = canon(sys,'modal') <enter>

a =
                x1         x2
     x1       0.35     0.42131
     x2   -0.42131        0.35

b =
              u1
     x1      1.354
     x2     1.1248

c =
               x1         x2
     y1    0.73855          0

d =
              u1
     y1      0
```

Note the appearance of the real and imaginary parts of the poles in z-domain as the elements of the Jordan form state-dynamics matrix.

While obtaining the pulse transfer functions of single-input, single-output, sampled-data analog systems, we employed a sampling and holding of continuous time input signal. This procedure can be extended to multivariable systems, where each input is sampled and held. Such a procedure leads to a *digital approximation* of multivariable analog system, which we studied in Chapter 4. Instead of the pulse transfer functions (or pulse transfer matrices), it makes a better sense to talk of *digital state-space representation* of a sampled-data, multivariable analog system. Chapter 4 presented analog-to-digital conversion of analog state-equations, to obtain their time domain solution on a digital computer. The same techniques of converting an analog system to an equivalent digital system can be employed for other purposes, such as controlling an analog plant by a digital controller. In Chapter 4, the MATLAB (CST) commands *c2d* and *c2dm* were introduced for approximating an analog system by an equivalent digital system. While *c2d* employs a zero-order hold (z.o.h) for holding the sampled signals, the command *c2dm* also allows a *first-order hold* (f.o.h), a *bilinear* (Tustin), or a *prewarp* interpolation for better accuracy when the continuous time input signals are smooth. The command c2dm is used as follows:

```
>>[Ad,Bd,Cd,Dd] = c2dm(A,B,C,D,T,'method') <enter>
```

where **A**, **B**, **C**, **D** are the continuous-time state-space coefficient matrices, **Ad**, **Bd**, **Cd**, **Dd** are the returned digital state-space coefficient matrices, T is the specified sampling interval, *method* is the method of interpolation for the input vector to be specified as *zoh* (for zero-order hold), *foh* (for first-order hold), *tustin* (for Tustin interpolation), or *prewarp* (for Tustin interpolation with *frequency prewarping* – a more accurate interpolation than plain Tustin). While the *first-order hold* was covered in Chapter 4, the *bilinear* (or Tustin) approximation refers to the use of the approximation, $s = \ln(z)/T \approx 2(z-1)/[T(z+1)]$ while mapping from the s-plane to the z-plane. Such an approximation is obtained by expanding $\ln(z) = 2(z-1)/(z+1) + (1/3)[(z-1)/(z+1)]^3 + \cdots$ and retaining only the first term of the series. While the imaginary axis on the s-plane is not *precisely* mapped as a unit circle in the z-plane, as it would be according to $z = e^{Ts}$ (or $s = \ln(z)/T$). A frequency response of the bilinear transformed digital system would be thus *warped* by this approximation. To remove the frequency warping of the bilinear transformation, a *pre-warping* technique is applied, by which the *critical frequencies* of the s-plane transfer function fall precisely on the z-plane at the points where they belong. You may refer to a textbook on digital control [2] for pre-warping techniques applied to bilinear transformation.

Example 8.15

Let us find a digital state-space representation of a sampled-data analog system with the following state coefficient matrices:

$$\mathbf{A} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -0.5 & 0.7 \\ 0 & -0.7 & -0.5 \end{bmatrix}; \quad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \\ 0 & -1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad \mathbf{D} = \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix} \tag{8.73}$$

if the sampling interval is 0.2 second. We begin by a digital conversion using a z.o.h with the command *c2dm* as follows:

```
>>A = [-1 0 0; 0 -0.5 0.7; 0 -0.7 -0.5]; B = [1 0; 0 -1; 0 -1]; C = [1 0
  0; 0 0 1]; <enter>

>>D = [0 0; 0 -2]; [Ad,Bd,Cd,Dd] = c2dm(A,B,C,D,0.1,'zoh')
  <enter>
```