# Appendix C

# Basics of State Space Modeling

## C.1 Introductory Concepts

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

$$\dot{\vec{x}}(t) = A\vec{x}(t) + B\vec{u}(t) \tag{C.1a}$$

$$\vec{y}(t) = C\vec{x}(t) + D\vec{u}(t) \tag{C.1b}$$

Equation (C.1a) is a system of first-order differential equations and is known as the state equation of the system. The vector $\vec{x}(t)$ is the state vector, and $\vec{u}(t)$ is the input vector. Equation (C.1b) is referred to as the output equation. $A$ is called the state matrix, $B$ the input matrix, $C$ the output matrix, and $D$ is the direct transition matrix.

One advantage of the state space method is that the form lends itself easily to the digital and analog computation methods of solution. Further, the state space method can be easily extended to the analysis of nonlinear systems.

State equations may be obtained from an $n^{th}$ order differential equation or directly from the system model by identifying appropriate state variables. To illustrate the first method, consider an $n^{th}$ order linear plant model described by the differential equation

$$\frac{d^n y}{dt^n} + a_{n-1}\frac{d^{n-1}y}{dt^{n-1}} + \cdots + a_1\frac{dy}{dt} + a_0 y = u(t) \tag{C.2}$$

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

$$x_1 = y, \quad x_2 = \dot{y}, \quad x_3 = \ddot{y}, \ldots, x_n = \frac{d^{n-1}y}{dt^{n-1}} \tag{C.3}$$

Taking derivatives of the first $n-1$ state variables, we have

$$\dot{x}_1 = x_2, \quad \dot{x}_2 = x_3, \quad \ldots, \quad \dot{x}_{n-1} = x_n \tag{C.4}$$

In addition, $\dot{x}_n$ comes from rearranging Eq. (C.2) and substituting from Eq. (C.3):

$$\dot{x}_n = -a_0 x_1 - a_1 x_2 - \cdots - a_{n-1}x_n + u(t) \tag{C.5}$$

In matrix form, this looks like

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u(t) \tag{C.6}$$

Thus the output equation is simply

$$y = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \vec{x} \tag{C.7}$$

*Example C.1*

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

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

*Solution*:
The differential equation is third order, and thus there are three state variables: $x_1 = y$, $x_2 = \dot{y}$, and $x_3 = \ddot{y}$. The first derivatives are:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ \dot{x}_3 &= -4x_1 - 3x_2 - 2x_3 + 5u(t) \end{aligned}$$

Or, in matrix form:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -4 & -3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} u(t)$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

The m-file `ode2phv.m` was developed to convert an $n^{th}$ order ordinary differential equation to the state space phase variable form. The syntax is `[A, B, C] = ode2phv(ai,k)`, and returns the typical three matrices. The input `ai` is a row vector containing the coefficients of the equation in descending order, and `k` is the coefficient on the right hand side. Using the ODE from Example C.1, we would enter:

*Example Code:*

```
>> ai = [2 4 6 8];
>> k = 10;
>> [A, B, C] = ode2phv(ai,k)
A =
    0   1   0
    0   0   1
    -4  -3  -2
B =
```

```
     0
     0
     5
C =
     1     0     0
>>
```

**Equations of Electrical Networks**

The state variables are directly related to the energy storage elements of a system. It would seem, therefore, that the number of independent initial conditions is equal to the number of energy storing elements. This is true—provided that there is no loop containing only capacitors and voltage sources, and there is no cut set containing only inductive and current sources. In general, if there are $n_C$ loops of all capacitors and voltages sources, and $n_L$ cut sets of all inductors and current sources, the number of state variables is

$$n = e_L + e_C - n_C - n_L \tag{C.8}$$

where $e_L$ and $e_C$ are the numbers of inductors and capacitors, respectively.

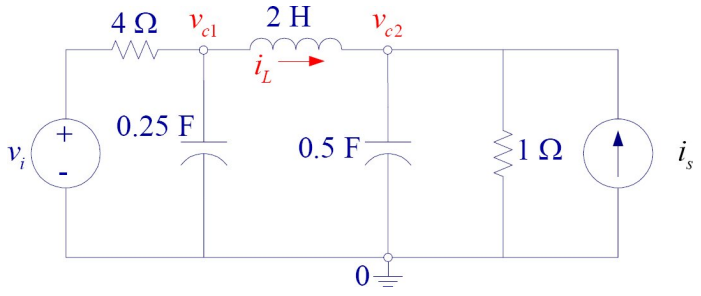



Figure C.1: Circuit of Example C.2

*Example C.2*

Write the state equation for the network shown in Figure C.1.
*Solution*:
Define the state variables as current through the inductor and voltage across the capacitors. Write two node equations containing capacitors and a loop equation containing the inductor. The state variables will be $v_{c1}$, $v_{c2}$, and $i_L$.
Node equations:

$$0.25\frac{dv_{c1}}{dt} + i_L + \frac{v_{c1} - v_i}{4} = 0 \Rightarrow \dot{v}_{c1} = -v_{c1} - 4i_L + v_i$$
$$0.5\frac{dv_{c2}}{dt} - i_L + v_{c2} - i_s = 0 \Rightarrow \dot{v}_{c2} = -2i_L + 2v_{c2} + 2i_s$$

Loop equation:

$$2\frac{di_L}{dt} + v_{c2} - v_{c1} = 0 \Rightarrow \dot{i}_L = 0.5v_{c1} - 0.5v_{c2}$$

Equivalently, in matrix form:

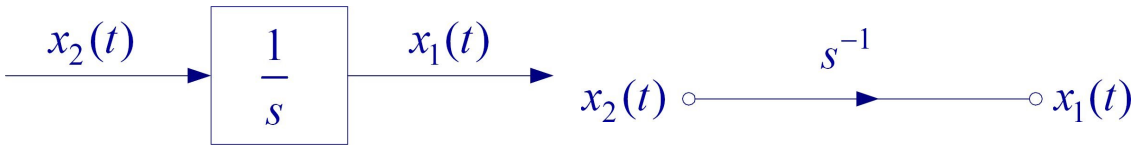
$$\begin{bmatrix} \dot{v}_{c1} \\ \dot{v}_{c2} \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} -1 & 0 & -4 \\ 0 & -2 & 2 \\ 0.5 & -0.5 & 0 \end{bmatrix} \begin{bmatrix} v_{c1} \\ v_{c2} \\ i_L \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_i \\ i_s \end{bmatrix}$$

### C.1.1 Simulation Diagrams

Equations (C.4) and (C.5) indicate that state variables are determined by integrating the corresponding state equation. A diagram known as the simulation diagram can be constructed to model the given differential equations. The basic element of the simulation diagram is the integrator. The first equation in (C.4) is $\dot{x}_1 = x_2$. Integrating we have:

$$x_1 = \int x_2 dx$$

The above integral is represented by the time-domain block diagram shown in Figure C.2a and by the signal flow graph in Figure C.2b.

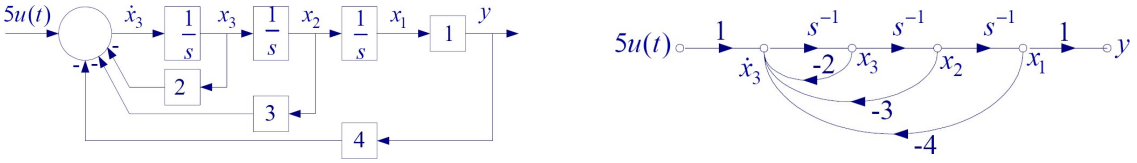



(a) Block diagram (b) Signal flow graph

Figure C.2: Simulation diagrams: Graphical representations of state integrators in the time domain.

It is important to know that that although the symbol $\frac{1}{s}$ is used for integration, the simulation diagram is still a time-domain representation. The number of integrators is equal to the number of state variables. For example, for the state equation in Example C.1 we have three integrators in cascade, the three state variables are assigned to the output of each integrator as shown in Figure C.3. The final state equation—seen in (C.5)—is represented via a summing point and

feedback paths. Completing the output equation, we obtain the simulation diagram known as phase-variable control canonical form (See Fig. C.3b).




(a) Block diagram (b) Signal flow

Figure C.3: Simulation diagrams for Example C.1 in phase-variable control canonical form.

### C.1.2 Transfer Function to State Space Conversion

Consider the transfer function of a third-order system where the numerator degree is lower than that of the denominator.

$$\frac{Y(s)}{U(s)} = \frac{b_2 s^2 + b_1 s + b_0}{s^3 + a_2 s^2 + a_1 s + a_0} \tag{C.9}$$

The above transfer function is decomposed into two (frequency domain) blocks in Figure C.4.


Figure C.4: The transfer function of Eq. (C.9) arranged in cascade form

Denoting the output of the first block as $W(s)$, we have the following input/output relationships:

$$W(s) = \frac{U(s)}{s^3 + a_2 s^2 + a_1 s + a_0} \tag{C.10a}$$

$$Y(s) = b_2 s^2 W(s) + b_1 s W(s) + b_0 W(s) \tag{C.10b}$$

Rearranging Eq. (C.10a), we get

$$s^3 W(s) = -a_2 s^2 W(s) - a_1 s W(s) - a_0 W(s) + U(s) \tag{C.11}$$

Using properties of frequency-domain tranforms, we see that Equations (C.11) and (C.10b) are the frequency-domain representations of the following time-domain differential equations:

$$\dddot{w} = -a_2 \ddot{w} - a_1 \dot{w} - a_0 w + u(t) \tag{C.12a}$$

$$y(t) = b_2 \ddot{w} + b_1 \dot{w} + b_0 w \tag{C.12b}$$

From the above expressions, we see that $\dddot{w}$ has to go through three integrators to get $w$ (as shown in Figure C.5). Completing the above equations results in the phase-variable control canonical simulation diagram.

Figure C.5 is a block diagram suitable for Simulink analysis. You may find it easier to construct the simulation diagram similar to the signal flow graph as shown in Figure C.6.

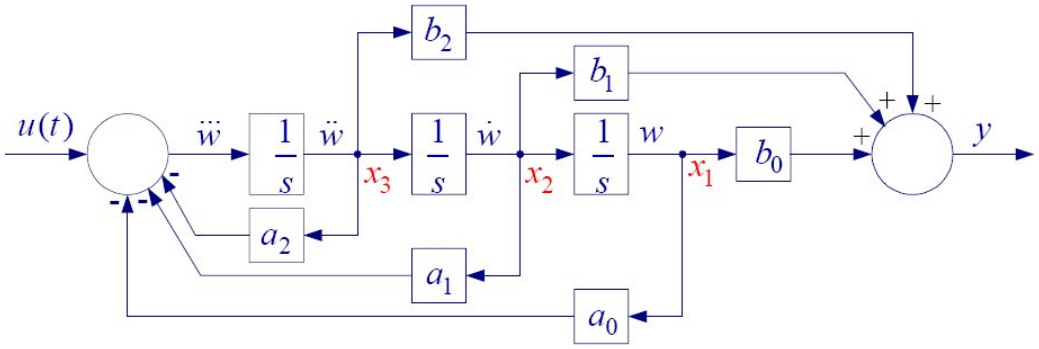



Figure C.5: Phase variable control canonical simulation block diagram for the transfer function in Eq. (C.9)

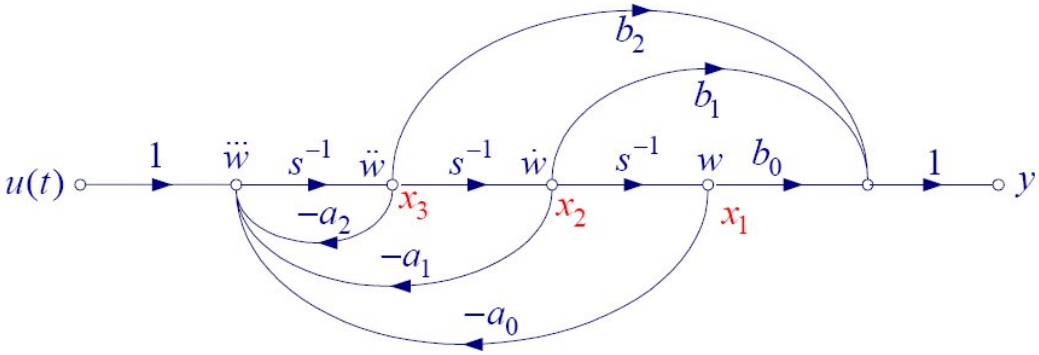



Figure C.6: Phase variable control canonical signal flow diagram for the transfer function in Eq. (C.9)

In order to write the state equation, the state variables $x_1(t)$, $x_2(t)$, and $x_3(t)$ are assigned to the output of each integrator from right to left in Figs. C.5 and C.6. Next, an equation is written for the input of each integrator:

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ \dot{x}_3 &= -a_0x_1 - a_1x_2 - a_2x_3 + u(t)\end{aligned}$$

and the output equation is $y = b_0x_1 + b_1x_2 + b_2x_3$. Once again, lumping it into matrix form, we get

$$\begin{aligned}\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(t) \\ y &= \begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\end{aligned} \tag{C.13}$$

It is important to note that Mason's gain formula can be applied to the simulation diagram in Fig. C.6 to obtain the original transfer function. Indeed, the determinant of the matrix $sI - A$ from Eq. (C.13) yields the characteristic equation (often denoted $\Delta$) for Mason's rule. See Section 2.7 of [1] for more on Mason's rule.

In conclusion, it is important to remember that there is no unique state space representation for a given transfer function. The state space often depends on the application involved, the complexity of the model, and the individual engineer!

The Control System Toolbox in MATLAB contains a set of functions for model conversion. Specifically, `[A, B, C, D] = tf2ss(num,den)` converts the transfer function fraction to state space phase-variable control canonical form.

*Example C.3*

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^2 + 7s + 2}{s^3 + 9s^2 + 26s + 24}$$

For the above transfer function, do the following:

1. Draw the simulation diagram and find a state space representation.
2. Use the MATLAB Control System Toolbox function `tf2ss` to find a state model.

*Solution*:
1. The transfer function in block diagram cascade form looks like Figure C.7a. From this we have

$$\begin{aligned} s^3 W(s) &= -9Ws^2 W(s) - 26sW(s) - 24W(s) + U(s) \\ Y(s) &= s^2 W(s) + 7sW(s) + 2W(s) \end{aligned}$$

Converting these to the time domain:

$$\begin{aligned} \dddot{w} &= -9\ddot{w} - 26\dot{w} - 24w + u \\ y(t) &= \ddot{w} + 7\dot{w} + 2w \end{aligned}$$

The time-domain equations above yield the simulation diagram in Figure C.7b. To obtain the state equation, the state variables $x_1(t)$, $x_2(t)$, and $x_3(t)$ are assigned to the output of each integrator from right to left. The equations corresponding to the input of each integrator are:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ \dot{x}_3 &= -24x_1 - 26x_2 - 9x_3 + u(t) \end{aligned}$$

The output equation is the summation of the feedforward links: $y = 2x_1 + 7x_2 + x_3$. Finally, in matrix form we have

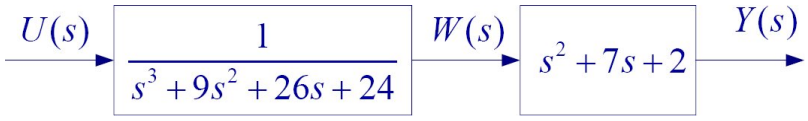
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -24 & -26 & -9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(t)$$

$$y = \begin{bmatrix} 2 & 7 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

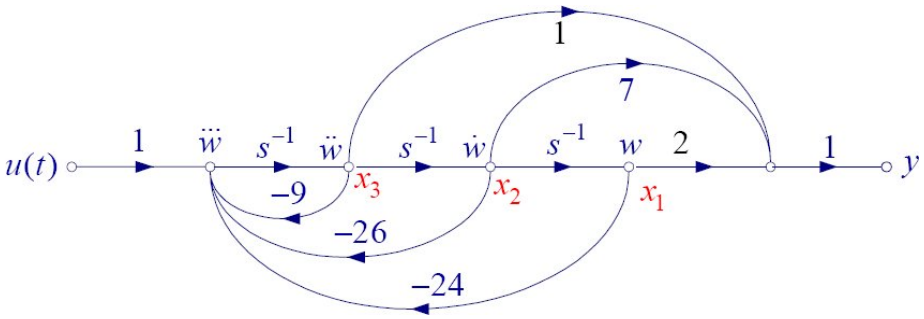
2. We write the following statements to check our results in MATLAB:

```
>> num = [1 7 2]; den = [1 9 26 24];
>> [A, B, C, D] = tf2ss(num,den)
A =                  B =          C =              D =
    -9  -26 -24          1            1   7   2        0
    1   0   0            0
    0   1   0            0
>>
```

It is important to realize that parts 1 and 2 of Example C.3 are equivalent. MATLAB simply numbers the state variables in a different order. To confirm this, we need only perform a simple substitution for one set of state variables.




(a) Block diagram of the transfer function in cascade




(b) Signal flow simulation digram

Figure C.7: Diagrams for Example C.3

### C.1.3 State Space to Transfer Function Conversion

Consider the state equation (C.1a). We may take its Laplace transform and rearrange it as follows:

$$sX(s) = AX(s) + BU(s) \Rightarrow (sI - A)X(s) = BU(s)$$

If we combine this with the transform of the output equation: $Y(s) = CX(s) + DU(s)$, we get

$$Y(s) = C(sI - A)^{-1}BU(s) + DU(s)$$

or, equivalently

$$\frac{Y(s)}{U(s)} = C(sI - A)^{-1}B + D \tag{C.14}$$

In the Control Systems Toolbox, the command `[num, den] = ss2tf(A,B,C,D,i)` converts the state equation to a transfer function for the $i^{th}$ input.

*Example C.4*

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$
$$y = \begin{bmatrix} 8 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Obtain the transfer function for the system described in the above state space model.

*Solution*:

Use the formula in Eq. (C.14).

$$sI - A = \begin{bmatrix} s & -1 \\ 6 & s+5 \end{bmatrix}$$
$$\Rightarrow \Phi(s) := (sI - A)^{-1} = \frac{\begin{bmatrix} s+5 & 1 \\ -6 & s \end{bmatrix}}{s^2 + 5s + 6}$$
$$\Rightarrow G(s) := C(sI - A)^{-1}B = \begin{bmatrix} 8 & 1 \end{bmatrix} \frac{\begin{bmatrix} s+5 & 1 \\ -6 & s \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}}{s^2 + 5s + 6}$$
$$= \frac{\begin{bmatrix} 8 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ s \end{bmatrix}}{s^2 + 5s + 6}$$

Therefore the transfer function is

$$G(s) = \frac{s+8}{s^2 + 5s + 6}$$

*Example C.5*

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

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Use MATLAB to find the transfer function corresponding to the above state space model.

*Solution*:

```
>> A = [0 1 0; 0 0 1; -1 -2 -3]; B = [10; 0; 0];
>> C = [1 0 0]; D = [0];
>> [num, den] = ss2tf(A,B,C,D,1);
>> G = tf(num,den)
G =
    10 s^2 + 30 s + 20
   ---------------------
   s^3 + 3 s^2 + 2 s + 1
>>
```

Also, `[z, p, k] = ss2zp(A,B,C,D,i)` converts the state equations to the transfer function in factored form.

MATLAB's Control System Toolbox contains many functions for model creation and inversion, data extraction, and system interconnections. A few of these functions for continuous-time control systems are listed in Table C.1. For a complete list of the toolbox's functions, type `help/control/control` at the command prompt.

| *Command* | *Description* |
|---|---|
| `tf` | Create transfer function models |
| `zpk` | Create zero/pole/gain models |
| `ss` | Create state space models |
| `tfdata` | Extract numerators and denominators |
| `zpkdata` | Extract zero/pole/gain data |
| `ssdata` | Extract state space matrices |
| `append` | Group LTI systems by appending inputs and outputs |
| `parallel` | Generalized parallel connection |
| `series` | Generalized series connection |
| `feedback` | Feedback connection of two systems |
| `connect` | Derive state space model from block diagram description |
| `blkbuild` | Builds a model from a block diagram |

Table C.1: Important continuous-time control system commands

The Control System Toolbox supports three commonly used representations of linear time-invariant (LTI) systems: `tf`, `zpk`, and `ss` objects. To create an LTI model or object, use the corresponding constructor. For example, `sys = tf(1,[1 0])` creates the transfer function $H(s) = 1/s$. The resulting variable `sys` is a `tf` object containing the numerator and denominator data.

You can now treat the entire model as a single MATLAB variable. For more details and examples on how to specify the various types of LTI models, type `ltimodels` followed by the construct type at the MATLAB command prompt.

The functions `tfdata`, `zpkdate`, and `ssdata` are provided for extracting the parameters of their corresponding objects. For example, the command `[num, den] = tfdata(T,'v')` returns the numerator and denominator of the `tf` object `T`. The argument `'v'` formats the outputs as row vectors rather than cell arrays.

The Control System Toolbox contains many more commands that allow the construction of a system out of its components. The lower entries in Table C.1 are useful for this purpose.

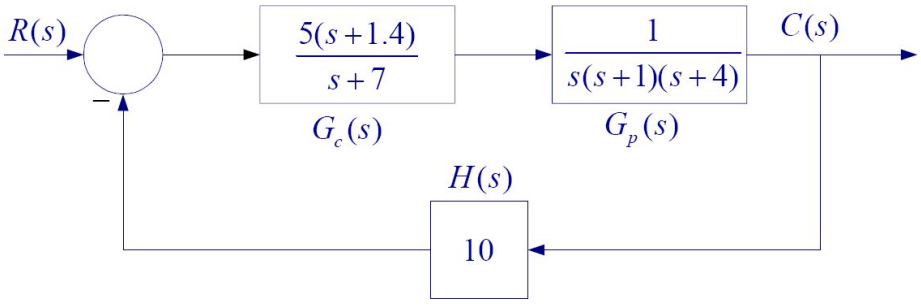



Figure C.8: Block diagram in the frequency domain for the system in Example C.6.

*Example C.6*

Use the `feedback` function to obtain the closed-loop transfer function and the `tf2ss` function to obtain the closed-loop state space model of the system in Figure C.8.

*Solution*:

The following commands should produce the desired result:

```
Gc = tf(5*[1 1.4],[1 7]);        % transfer function Gc
Gp = tf([1],[1 5 4 0]);          % transfer function Gp
H = 10;
G = series(Gc,Gp)                % connect Gc and Gp in cascade
T = feedback(G,H)                % close feedback loop
[num, den] = tfdata(T,'v')       % return num and den as row vectors
[A, B, C, D] = tf2ss(num,den)    % converts to state space model
```

The transfer function should look like

```
                5 s + 7
     ---------------------------------
      s^4 12 s^3 + 39 s^2 + 78 s + 70
```

And the matrices are

```
A =                     B =     C =                 D =
    -12 -39 -78 -70         1       0   0   5   7       0
    1   0   0   0           0
    0   1   0   0           0
    0   0   1   0           0
```

## C.2 MATLAB Functions for Modeling and Analysis

Once a system is described by a certain model—be it in state space, by a transfer function, or otherwise—it is often important to perform some form of analysis on it. How does it respond to different initial conditions? Which inputs correspond to which outputs? Is this thing stable or can we make it stable? These are all questions we may ask of a system that comes presented to us as a "black box."

The MATLAB Control System Toolbox contains the functions in Table C.2 for analysis of time-domain response.

| *Command* | *Description* |
|---|---|
| `step` | Step response |
| `impulse` | Impulse response |
| `initial` | Response of a state space system to the given initial state |
| `lsim` | Response to arbitrary inputs |
| `gensig` | Generates input signal for `lsim` |
| `damp` | Natural frequency and damping of system poles |
| `ltiview` | Response analysis GUI (LTI System Viewer) |

Table C.2: Time-domain system analysis functions

Given a transfer function of a closed-loop control system, the function `step(num,den)` produces the step response plot with the time vector automatically determined. If the closed-loop system is defined in state space instead, we use `step(A,B,C,D)` with or without subsequent optional arguments. If output variables are specified for the `step` function, say `[y, t, x]`, then the output will be saved to `y` for the time vector `t`. The array `x` contains the trajectories of all of the state variables along the same time vector. This syntax also applies to `impulse`, `initial`, and `lsim`.

*Example C.7*

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

Obtain the unit step response of the system with the closed-loop transfer function above. Also, use the `damp` function to obtain the roots of the characteristic equation, the corresponding damping factors, and the natural frequencies.

*Solution*:

The following code produces the response seen in Figure C.9.

```
>> num = 25*[0.4 1];
>> den = conv([0.16 1], [1 6 25])   % Multiplies two polynomials.
>> T = tf(num,den)                  % Create TF object.
>> step(T), grid                    % Produces step response plot.
>> damp(T)                          % Produces data below:
        Eigenvalue              Damping         Freq. (rad/s)
        -3 + 4i                 6.00e-001           5.00
        -3 - 4i                 6.00e-001           5.00
        -6.25                   1.00e+000           6.25
>>
```

The damping factors and natural frequencies are displayed in the right two columns. The eigenvalues are the roots of the characteristic equation.

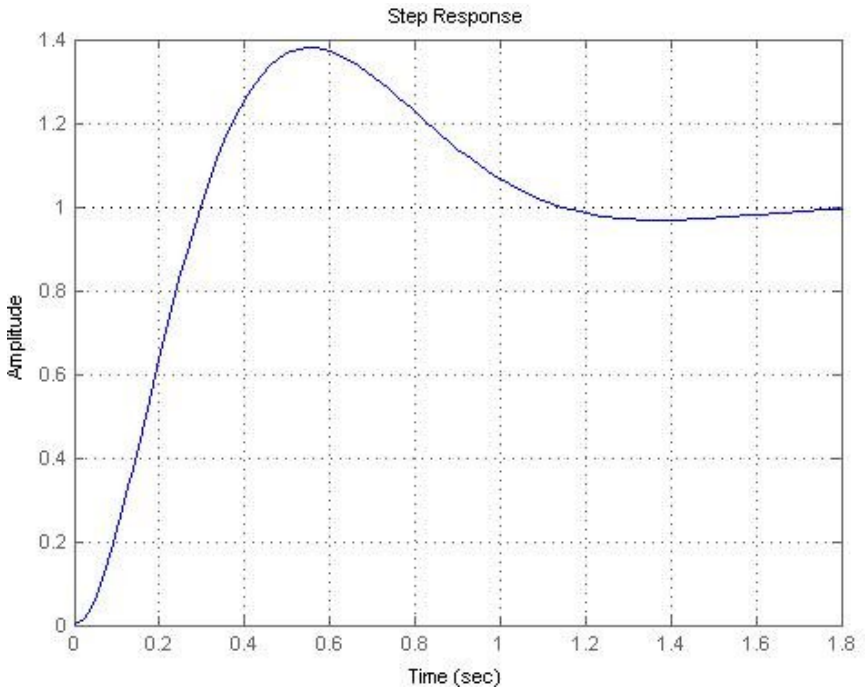



Figure C.9: Unit step response for system in Example C.7

*Example C.8*

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

The closed-loop transfer function of a control system is described by the third-order transfer function $T(s)$. Do the following:

1. Find the dominant poles of the system.
2. Find a reduced-order model.
3. Obtain the step response of the third-order system and the reduced-order system on the same figure plot.

*Solution*:
1. The poles are the roots of the denominator polynomial. To find this, we use `roots([1 36 205 750])`. This gives us roots at $-30$ and $-3 \pm 4i$. To determine dominance, we look at the time constants (negative inverse of real part) associated with these poles. We see the pole at $-30$ has time constant $\tau_1 = 1/30$ where as the other two have time constant $\tau_2 = 1/3$. The order of magnitude difference tells us that the poles at $3 \pm 4i$ are dominant. For more on time constants, see [1], Section 2.5.
2. To reduce the model, we factor out the negligible pole and divide the numerator by its magnitude:

$$T(s) = \frac{750}{(s+30)(s^2+6s+25)} \Rightarrow \tilde{T}(s) = \frac{25}{s^2+6s+25}$$

*Example Code:*

```
3. The following script produces the desired plot (Fig.\ \ref{fig.ex.dompoles}).
\begin{verbatim}
num1 = 750;
den1 = [1 36 205 750];
T = tf(num1,den1);              % Third-order system tf object.
num2 = 25;
den2 = [1 6 25];
Ttilde = tf(num2,den2);         % Reduced system tf object.
[y1 t] = step(T);               % Response and time vectors of T.
y2 = step(Ttilde,t);            % Response using the same time
                                    % vector of the reduced system.
plot(t,y1,'r-',t,y2,'b:')       % Put responses on the same plot.
grid;
xlabel('Time (s)');
ylabel('Displacement');
legend('Third-order','Reduced-order');
\end{verbatim}
```

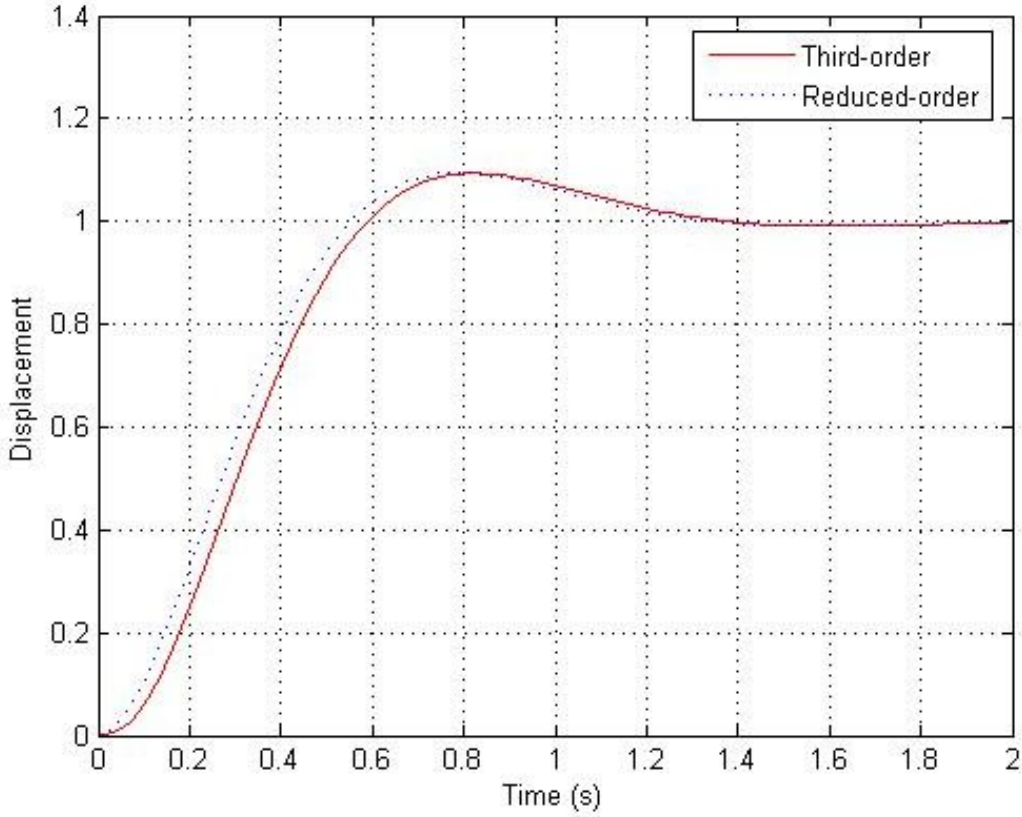



Figure C.10: Step responses of the original and reduced systems in Example C.8

### C.2.1 The LTI Viewer

The Control System Toolbox LTI Viewer is a GUI (graphical user interface) that visually demonstrates the analysis of linear time-invariant systems. We use the LTI Viewer to view and compare the response plots of several linear models at the same time. We can also generate time- and frequency-domain response plots to inspect key response parameters such as rise time, maximum overshoot, and stability margins. Using mouse-driven interactions, we can select input and output channels for multi-input, multi-output (MIMO) systems. The LTI Viewer can display up to six different plot types simultaneously, including step and impulse responses, Bode, Nyquist, Nichols, sigma, and pole/zero diagrams.

The command syntax is

```
ltiview('plot type',sys,extra)
```

where `sys` is the system object in question, and `'plot type'` is one of the following strings: `step`, `impulse`, `initial`, `lsim`, `bode`, `nyquist`, `nichols`, or `sigma`. The optional argument `extra` specifies the final time of the simulation.

Once an LTI Viewer is open, a right-click of the mouse allows us to change the response type and obtain the system time-domain and frequency-domain specifications, such as those in Table C.3.

| *Menu Title* | *Description* |
|---|---|
| Plot Type | Changes the plot type |
| Systems | Selects any of the models loaded in the LTI Viewer |
| Characteristics | Displays key response characteristics and parameters |
| Zoom | Zooms in and out of plot regions |
| Grid | Adds grids to the plots |
| Properties | Opens the Property Editor to customize plot attributes |

Table C.3: LTI Viewer context-click menus

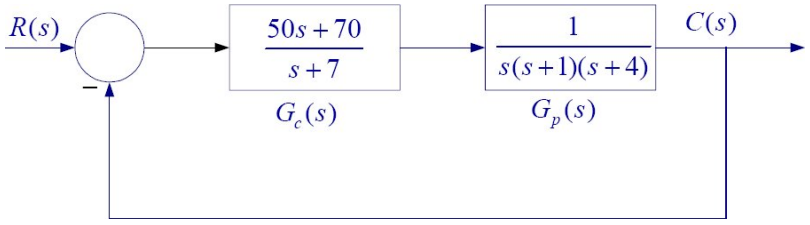



Figure C.11: Block diagram for Example C.9

*Example C.9*

Use the LTI Viewer to obtain the step response and the time-domain specifications for the control system shown in Fig. C.11.
*Solution*:
We use the following commands to generate the response plot in Figure C.12.

```
>> Gc = tf([50 70],[1 7]);      % Transfer function Gc.
>> Gp = tf([1],conv([1 1 0],[1 4]));    % Transfer function Gp,
                                % with expanded denominator.
>> H = 1;                       % Unity feedback.
>> G = series(Gc,Gp);           % Connect Gc and Gp in cascade.
>> T = feedback(G,H)            % Close unity feedback loop.

Transfer function:
            50 s + 70
---------------------------------
s^4 + 12 s^3 + 39 s^2 + 78 s + 70

>> ltiview('step',T)
>>
```

For time-domain analysis, it is often much easier to use the LTI Viewer because it is possible to obtain many system parameters with a simple right-click of the mouse. In addition, it allows us to select from a myriad of responses and data projections via the Plot Type.

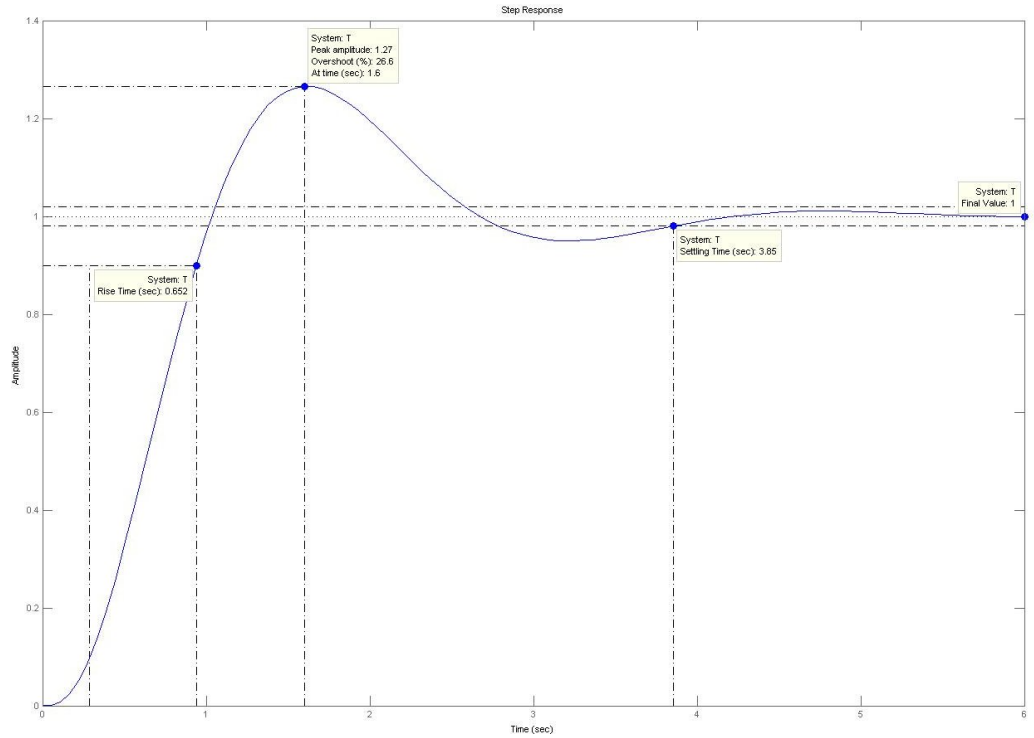



Figure C.12: Step response with metadata tags for Example C.9

### C.2.2 Numerical Solutions of Differential Equations

There are many powerful techniques in MATLAB for the numerical solution of nonlinear equations. A popular technique is the Runge-Kutta method, which is based on formulas derived by using an approximation to replace the truncated Taylor series expansion. The interested reader should see [3] or an equivalent text on numerical methods.

MATLAB provides several powerful functions for the numerical solution of differential equations: See Table C.4. Two of the functions employing the Runge-Kutta-Fehlberg methods are `ode23` and `ode45`; they are based on the Fehlberg second- and third-order pair of formulas for medium accuracy and the fourth- and fifth-order pair for higher accuracy.

| *Command* | *Description* |
|---|---|
| `ode23` | Solve non-stiff differential equations, low order method |
| `ode45` | Solve non-stiff differential equations, medium order method |

Table C.4: Numerical solvers for ordinary differential equations

The $n^{th}$-order differential equation must be transformed first into $n$ first-order differential equations and must be placed in an m-file that returns the derivative of the state equations. This file's name is represented as the string `'xprime'` in the general syntax below:

```
[t, x] = ode23('xprime',tspan,x0,option)
```

Also, `tspan` is a vector of times covering the interval of integration, and `x0` is a column vector of initial state conditions. Commonly used options are scalar relative error tolerance (`RelTol = 1e-3` by default) and vector absolute error tolerances (all elements of `AbsTol = 1e-6` by default).

*Example C.10*

$$\frac{d^2\theta}{dt^2} + \frac{B}{m}\frac{d\theta}{dt} + \frac{g}{l}\sin\theta = 0$$

The above equation describes the motion of the simple pendulum derived in Lab Session 3, Case Study 3.2.2. Using the MATLAB function `ode23`, obtain the numerical solution for the following values:

- $m = 0.5$kg and $l = 0.613$m
- $B = 0.05$kg-s/m and $g = 9.81$m/s/s
- $\theta(t = 0) = \dot{\theta}(t = 0) = 0$

# Bibliography

[1] Richard C. Dorf and Robert H. Bishop. Modern Control Systems, 11th edition. Prentice Hall, Upper Saddle River, NJ. 2008.

[2] EE351L Laboratory Manual, 1st edition.

[3] John C. Butcher. Numerical Methods for Ordinary Differential Equations, 2nd edition. John Wiley. 2003.