

CHAPTER 3

MODELING, DESIGN, AND SIMULATION TOOLS

3.1 INTRODUCTION

In this chapter we will look more closely at continuous-time state-space models, their properties, and how they are derived from physical systems. This will lead to numerical methods and algorithms for computer software that can be applied to the many tasks associated with the simulation of an aerospace vehicle and design of its control systems. The software tools will provide the capability to trim aircraft models for steady-state flight, perform digital flight simulation, extract linear state-space and transfer function descriptions of aircraft models, and perform linear control system design. These operations are illustrated in Figure 3.1-1.

In the figure the nonlinear state and output equations are, respectively,

$$\dot{X} = f(X, U), X(n \times 1), U(m \times 1) \quad (3.1-1a)$$

$$Y = g(X, U), Y(p \times 1), \quad (3.1-1b)$$

where f and g represent arrays of continuous, single-valued functions. The linear versions of these equations are

$$\dot{x} = Ax + Bu \quad (3.1-2a)$$

$$y = Cx + Du \quad (3.1-2b)$$

An output equation is required because the state variables may not all be physical variables or directly accessible. Hence there is a need to represent physically measurable quantities by the output variables Y_i or y_i .

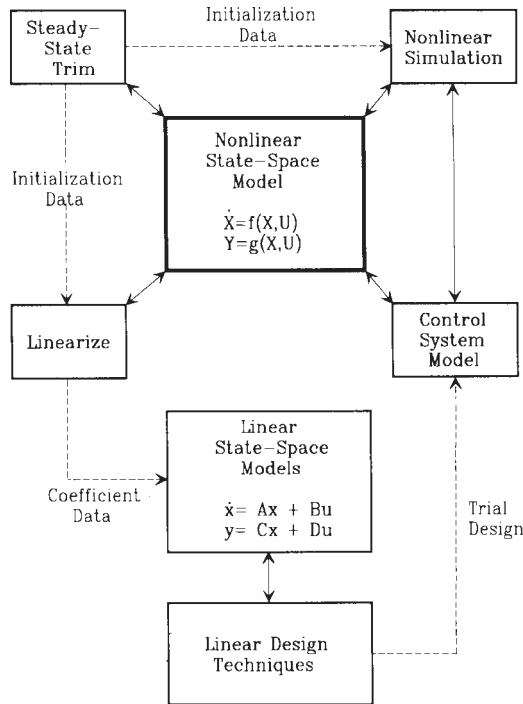


Figure 3.1-1 State-space operations.

Referring to Figure 3.1-1, the behavior of a real system can be simulated by solving the nonlinear model equations. The mathematical theory for numerical solution of ordinary differential equations (ODEs) is mature and is cast in the state-space canonical form. Therefore, a state-space model is a prerequisite for simulation. Second, real systems operate around design points or equilibrium conditions and are nonlinear to varying degrees as they deviate from equilibrium. By linearizing the nonlinear model around these design points the powerful tools of linear systems theory can be used to analyze the system and perform control systems design. The state-space formulation lends itself to linearization around equilibrium points. Third, the matrix formulation of the linear state equations readily handles multiple-input, multiple-output (MIMO) systems with single-input, single-output (SISO) systems as a special case. We will see that the MIMO description is essential for some aspects of aircraft dynamics. Lastly, if additional equations are to be coupled to the model, for example, to simulate an automatic control system, this is easily done when the controller equations are also in state-space form.

Also in this chapter, we have provided source code for state-space models of two different aircraft. These models will be used to illustrate aircraft dynamic behavior and for control system design examples in Chapter 4. The final topic of the chapter is a review of the classical control theory and design techniques that will be used in Chapter 4.

3.2 STATE-SPACE MODELS

ODEs are our most powerful method for modeling continuous-time lumped-parameter dynamic systems. Continuous time implies that the variables are uniquely defined at all moments in time within a specified interval, except possibly at a countable set of points. *Lumped parameter* implies that each of the interconnected elements that make up the model responds immediately to its excitation. This is in contrast to distributed-parameter systems, in which disturbances propagate through the system as waves. Distributed-parameter systems are described by partial differential equations. Real dynamic systems can behave as lumped- or distributed-parameter systems, depending on the frequency spectrum of their excitation. For example, an aircraft will respond partly as a distributed-parameter system to a sudden wind gust that excites the flexible bending modes of the wings and fuselage. For wind disturbances that are less abrupt, it will respond according to our rigid-body equations of motion. For large, flexible aircraft such as passenger jets, the flexible modes can be low enough in frequency to approach or overlap with the rigid-body modes. We will restrict ourselves to lumped-parameter models described by ODEs, and these ODEs allow us to describe a wide range of nonlinear and time-varying systems.

The continuous-time state-space equations are a “canonical form” of the ODE description of continuous-time lumped-parameter dynamic systems. That is, they represent a form to which all the members of the class can be reduced by appropriate transformations. Reduction to this canonical form allows a wide range of modeling, simulation, and design problems to be handled within a common framework of algorithms and software tools, as shown in Figure 3.1-1.

Models of Mechanical and Electrical Systems

There are many ways of deriving state equations for these systems. Here we give two examples in which we choose the state variables according to the energy storage elements and then use an appropriate technique for finding the state equations.

Example 3.2-1: State Equations for a Mechanical System Figure 3.2-1 shows a simple mechanical system to illustrate the derivation of state equations. The input to the system is the spring displacement $u(t)$, and the output is the displacement $y(t)$ of the mass m . The mass slides with negligible friction, and the springs are linear with stiffness k_i and zero mass. The dampers also have no mass and only dissipate energy. They produce a reaction force equal to the viscous constant, d_i , times the rate at which they are extending or contracting. The auxiliary variable $w(t)$ is needed so that equations can be written for the associated spring and damper. There are three independent energy storage elements, and state variables can be assigned accordingly:

$$\begin{aligned} x_1 &= (u - y), && \text{compression of spring } k_1 \\ x_2 &= (y - w), && \text{compression of spring } k_2 \\ x_3 &= \dot{y}, && \text{translational rate of } m \end{aligned}$$

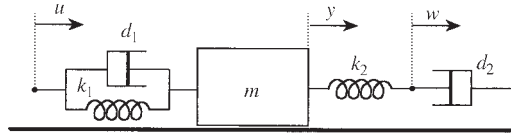


Figure 3.2-1 A mechanical system.

Equations involving the state variables can now be written by inspection:

$$\begin{aligned}\dot{x}_1 &= \dot{u} - x_3 \\ d_2(x_3 - \dot{x}_2) &= k_2 x_2 \\ m\dot{x}_3 &= k_1 x_1 + d_1(\dot{u} - x_3) - k_2 x_2\end{aligned}$$

These equations can be put into the form

$$\dot{x} = Ax + B_1 \dot{u}$$

To reduce this to state-space form, let

$$z = x - B_1 u$$

Then

$$\dot{z} = Ax = Az + (AB_1)u$$

and these are state equations in standard form. This approach will work for most simple mechanical systems. The technique used to remove \dot{u} from the first set of equations will also work for the equation:

$$\dot{x} = Ax + B_1 \dot{u} + B_0 u \quad \blacksquare$$

In any physical system, the stored energy cannot be changed instantaneously because this would require infinite power. This provides a way of directly finding the coefficient matrices of the linear state equations, which is most easily illustrated with an electrical circuit.

In an electrical circuit, energy is stored in the magnetic fields of inductors and in the electric fields of capacitors. Inductor currents and capacitor voltages can be chosen as state variables because these quantities determine the stored energy. Therefore, the current through an inductor, or voltage across a capacitor, cannot change instantaneously.

For any given unexcited circuit, if we could place an ideal unit-step voltage generator in series with each capacitor in turn, inside its terminals, its state variable would jump to 1.0 volts at $t = 0$. Instantaneously, all of the capacitors would act like short circuits and all of the inductors like open circuits. The capacitor currents and inductor voltages are proportional to the derivatives of their state variables ($i = C(dv/dt)$),

$v = L(di/dt)$). Therefore, all of the state derivatives can be found, when all but one of the state variables are zero, by analyzing a much simpler circuit in which capacitors have been replaced by short circuits and inductors by open circuits. This gives one column of the A -matrix. Similarly, placing a unit-step current generator in parallel with each inductor in turn gives the columns of the A -matrix corresponding to the inductor current state variables. This procedure is illustrated in our second example.

Example 3.2-2: State Equations for an Electrical System Here we will find the A - and B -matrix elements a_{ij} , b_i for the “bridged-T” circuit shown in Figure 3.2-2. In the figure the state variables x_1 , x_2 , and x_3 have been assigned to the capacitor voltages and inductor current. Imagine a unit-step voltage generator placed in series with C_1 , as indicated by the dashed circle, and let $t = 0^+$ indicate that the generator has switched from zero to 1 volt. Then, at $t = 0^+$,

$$x_1(0^+) = 1.0, \quad x_2(0^+) = 0.0, \quad x_3(0^+) = 0.0$$

The defining equation for a capacitor and the linear state equation then give

$$i_1(0^+) \equiv C_1 \dot{x}_1(0^+) = C_1 a_{11} x_1(0^+) \quad \text{or} \quad a_{11} = \frac{1}{C_1} \frac{i_1(0^+)}{x_1(0^+)}$$

The conductance $i_1(0^+)/x_1(0^+)$ is easily found from the resistive circuit obtained when C_2 is replaced by a short circuit and L by an open circuit. The result is

$$a_{11} = \frac{-1}{C_1 \left(R_1 + \frac{R_2 R_4}{R_2 + R_4} \right)}$$

The same voltage generator also gives

$$v_2(0^+) \equiv L \dot{x}_2(0^+) = L a_{21} x_1(0^+) \quad \text{or} \quad a_{21} = \frac{1}{L} \frac{v_2(0^+)}{x_1(0^+)} = \frac{1}{L}$$

and

$$a_{31} = \frac{i_2(0^+)}{C_2 x_1(0^+)} = \frac{-1}{C_2 (R_1 + R_4 + (R_1 R_4)/R_2)}$$

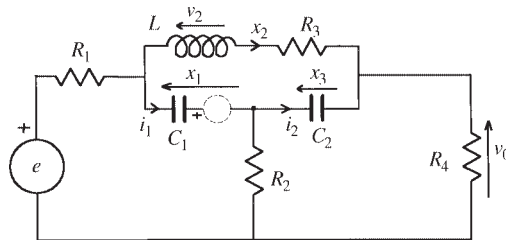


Figure 3.2-2 State variables for an electrical circuit.

The second and third columns of the A -matrix can be found, respectively, by putting a unit-step current generator in parallel with L and a unit-step voltage generator in series with C_2 . In the same manner, the B -matrix can be found by letting the input voltage $e(t)$ come from a unit-step voltage generator. The C - and D -matrices can be found in an ad hoc way; in this case an expression for the output voltage is

$$v_o = (x_2 + C_2 \dot{x}_3) R_4$$

Substitution of the state equation for \dot{x}_3 into this expression yields C and D . ■

This technique will always work for linear time-invariant electric circuits, though any mutual-induction coupling causes complications. Alternative techniques include setting up the Kirchhoff loop or nodal equations and reducing these to state equations (Nise, 1995).

Reduction of Differential Equations to State-Space Form

Consider the following nonlinear, scalar ODE:

$$\ddot{y} + f(\dot{y}) + g(y) = h(u) \quad (3.2-1)$$

Here $u(t)$ is a known input and $y(t)$ is the response of the system described by this ODE. The functions f , g , h are arbitrary, known nonlinear functions and may be the result of manipulating a preceding nonlinear equation to reduce the coefficient of the highest derivative to unity. Suppose that we convert this differential equation to two simultaneous integral equations by writing

$$\begin{aligned} y &= \int \dot{y} dt \\ \dot{y} &= \int [h(u(t)) - f(\dot{y}) - g(y)] dt \end{aligned} \quad (3.2-2)$$

Now consider how these equations might be solved. The variable y may be, for example, a position variable, but the functional form of the solution is the same if we use any other quantity as an *analog* of position. In an *analog computer* a device is available to perform integration of voltages with respect to time, and voltage is the analog of whatever physical variable we wish to simulate. We can make a diagram of the analog computer connections if we draw a box representing the integration operation, symbols representing summation and multiplication of variables, and lines showing which variables are subjected to each operation. Thus, Figure 3.2-3 shows a *simulation diagram* (or simply a *block diagram*) of Equations (3.2-2). If an analog computer is connected in this way and switched on with the correct initial conditions on the integrators, it will effectively solve Equations (3.2-1) or (3.2-2).

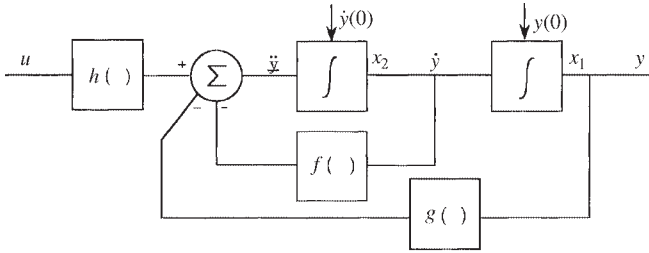


Figure 3.2-3 A SISO simulation diagram.

Referring to Figure 3.2-3, if we simultaneously break the input connections of all of the integrators (in an analog computer the inputs would be connected to “signal ground”), the outputs of the integrators will remain constant at the values they had at the instant of breaking the connections. If the continuation of the input signal $u(t)$ is available, the analog computer simulation can be restarted at any time and all of the signals will assume the values they had when the connections were broken, and the simulation can continue with no information lost. Therefore, the integrator output variables satisfy our earlier definition of state variables. Knowledge of their values at any time instant, together with the input signal, completely defines the state of the system.

Now starting from the right-hand side of Figure 3.2-3, let state variables x_1 and x_2 be defined as the integrator outputs. Therefore, by inspection of the diagram, or Equations (3.2-2), the state equations for the second-order ODE (3.2-1) are

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= h(u) - f(x_2) - g(x_1)\end{aligned}\tag{3.2-3}$$

This technique can easily be extended to find n state variables for an n th-order differential equation. It is not necessary to draw the block diagram, because the technique simply amounts to defining $y(t)$ and its derivatives up to the $(n - 1)$ th as state variables and then writing the equation for the n th derivative by inspection. State variables chosen in this way are often called *phase variables*.

Next consider a more difficult example where the block diagram will be helpful. This time the ODE model will involve derivatives of the input variable but will be linear with constant coefficients, as in the following second-order example:

$$\ddot{y} + a_1 \dot{y} + a_0 y = b_2 \ddot{u} + b_1 \dot{u} + b_0 u\tag{3.2-4}$$

The coefficient of the highest derivative of y can always be made equal to unity by dividing all of the other coefficients. Let the operator p^n indicate differentiation n times with respect to time and write this equation as

$$p^2(y - b_2 u) + p(a_1 y - b_1 u) + (a_0 y - b_0 u) = 0$$

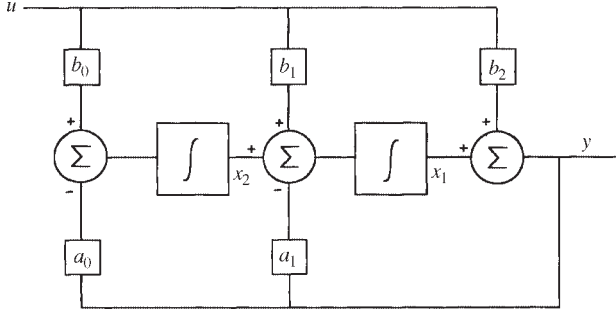


Figure 3.2-4 A general second-order SISO simulation diagram.

Now turn this into an integral equation for y ,

$$y = b_2 u + \int (b_1 u - a_1 y) dt + \int \int (b_0 u - a_0 y) d\tau dt$$

A simulation diagram can be drawn by inspection of this equation and is shown in Figure 3.2-4. A set of state equations can again be found by assigning state variables to the outputs of the integrators.

This example can be extended to the general case of an n th-order differential equation, with all derivatives of the input present. The differential equation for the general case is

$$p^n y + \sum_{i=0}^{n-1} a_i p^i y = \sum_{i=0}^n b_i p^i u \quad (3.2-5)$$

and the state equations are

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} -a_{n-1} & 1 & 0 & \dots & 0 \\ -a_{n-2} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_1 & 0 & 0 & \dots & 1 \\ -a_0 & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} (b_{n-1} - a_{n-1}b_n) \\ (b_{n-2} - a_{n-2}b_n) \\ \vdots \\ \vdots \\ (b_0 - a_0b_n) \end{bmatrix} u(t) \quad (3.2-6)$$

If the highest derivative on the right-hand side of (3.2-5) is the m th, then for real systems $m \leq n$. In the cases where $m = n$ this is a practical approximation in situations when very-high-frequency effects can be neglected. Therefore, $b_i \equiv 0$ for $i > m$, and when $m < n$ a group of n terms disappears from the right-hand side of (3.2-6).

This form of the linear state equations is known as the *observer canonical form*. The A -matrix has a structure called a *companion form*, which is known to be “ill conditioned” for numerical computation but is useful for theoretical results. The technique used here to obtain the simulation and state equations can be extended to include time-varying ODE coefficients (Laning and Battin, 1956), but there is no general method for deriving the simulation diagram when the ODE is nonlinear.

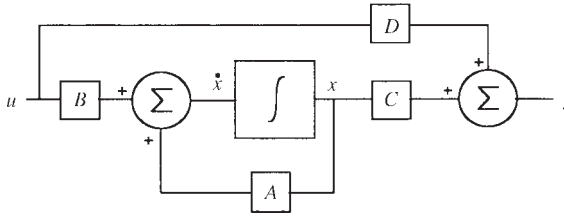


Figure 3.2-5 State equation simulation diagram.

Finally, let us take a look at a simulation diagram representation of the linear state equations. Figure 3.2-5 shows a diagram that represents the linear state and output equations, as given by Equations (3.1-2). In this diagram the lines, or signal paths, carry several variables simultaneously, the coefficient boxes represent matrix operations, and the integrator box represents multiple integrators simultaneously processing all of the input signals individually. This also represents a valid way to wire up an analog computer; the advantages of this form of system model will become apparent in this chapter.

Time-Domain Solution of LTI State Equations

When the state equations are linear and time invariant (LTI), they can be solved analytically. Elementary differential equation texts show that a linear first-order ODE can be solved by using an exponential “integrating factor” to yield an exact derivative. When the equation is also time invariant, the integrating factor reduces to a simple exponential function. An analogous method can be used to solve the set of n first-order LTI state equations.

The matrix exponential e^{At} , where A is a square constant matrix, is defined by the matrix series

$$e^{At} \equiv I + At + \frac{A^2 t^2}{2!} + \frac{A^3 t^3}{3!} + \dots \quad (3.2-7)$$

The series is uniformly convergent and can be differentiated or integrated term by term, resulting in properties analogous to the scalar exponential function

$$\frac{d}{dt}(e^{At}) = Ae^{At}$$

and

$$A \int e^{At} dt = e^{At} - I$$

It is also evident from (3.2-7) that $\exp(At)$ is commutative with A , that is,

$$Ae^{At} = e^{At}A$$

Now using e^{At} as an integrating factor, the state equation (3.1-2a) can be written as

$$\frac{d}{dt} (e^{-At}x(t)) = e^{At}Bu(t)$$

If this equation is integrated, with the constant of integration determined from initial conditions at $t = t_0$, we obtain

$$x(t) = e^{A(t-t_0)}x(t_0) + \int_{t_0}^t e^{A(t-\tau)}Bu(\tau) d\tau \quad (3.2-8)$$

The first component of this solution is the *homogeneous component*, which is the response to the initial conditions $x(t_0)$, and $e^{A(t-t_0)}$ takes the state from time t_0 to t and is called the *transition matrix*. The second component is the *forced component*, which is the response to the input $u(t)$. The integral on the right-hand side of (3.2-8) is a *convolution integral*, the time-domain equivalent of transform multiplication in the frequency domain.

The solution (3.2-8) is of little computational value to us because of the difficulties of finding analytical expressions for the transition matrix for systems of all but the lowest order. Also the convolution integral is inconvenient to evaluate for any but the simplest input functions. However, this solution does lead to a discrete-time recursion formula that is useful. A recursion formula becomes practical when we consider a short time interval T , over which the input can be approximated by a simple function. Therefore, we look for a discrete-time formula by considering a time interval from $t = kT$ to $t = (k+1)T$, where k is a positive integer. In Equation (3.2-8), let $t_0 = kT$ and $t = (k+1)T$ and make a change of variable $\lambda = (\tau - kT)$; then

$$x(k+1) = e^{AT}x(k) + e^{AT} \int_0^T e^{-A\lambda}Bu(\lambda + kT)d\lambda, \quad (3.2-9)$$

where T is implied in the argument of x .

The integral in (3.2-9) can be evaluated by a variety of methods, for example, the trapezoidal rule or Simpson's rule. We will take a simple stepped approximation to $u(t)$ such that $u(\lambda + kT) \approx u(kT)$ for $0 \leq \lambda < T$ [this is called a *zero-order hold (ZOH) approximation*]; $u(kT)$ can then be taken out of the integrand. The remaining integral can be evaluated by considering term-by-term integration of the matrix exponential, and the result is

$$x(k+1) = e^{AT}x(k) + Q(T)Bu(k), \quad (3.2-10)$$

where $Q(T)$ is given by

$$Q(T) = T \left[I + \frac{AT}{2!} + \frac{A^2T^2}{3!} + \frac{A^3T^3}{4!} + \cdots \right] \quad (3.2-11a)$$

or, when A^{-1} exists,

$$Q(T) = A^{-1} (e^{AT} - I) \quad (3.2-11b)$$

Equation (3.2-10) is a discrete-time recursion formula. It can be used as an alternative to numerical integration of the state equations when the equations are linear and e^{AT} has been found analytically or numerically. The matrix exponential e^{AT} is called the *discrete-time transition matrix*. Methods of computing the transition matrix are described in the literature (Healey, 1973; Moler and Van Loan, 1978; Zakian, 1970), and methods of computing integrals involving the matrix exponential (e.g., $Q(T)$) are described by Van Loan (1978). Commercial software is available to compute the transition matrix (e.g., MATLAB “expm”).

Modal Decomposition

In Section 1.3 the modal coordinates were introduced to show the connection between eigenvalues and the natural modes of a dynamic system. It is possible to use the additional information contained in the eigenvectors to determine which variables are involved in a given mode and what inputs will excite the mode. The time-domain solution (3.2-8) of the LTI state equation can be used for this purpose.

The continuous-time transition matrix can be expressed in terms of eigenvalues and eigenvectors in the following way. The similarity between the A matrix and (in general) a Jordan form matrix can be used to express an arbitrary power of A as

$$A^k = (MJM^{-1}) (MJM^{-1}) \dots = MJ^k M^{-1}$$

When this is done for every term in the matrix exponential series and when J is assumed to be diagonal (distinct eigenvalues), the result is

$$e^{At} = Me^{Jt}M^{-1} = [v_1 \ v_2 \ \dots v_n] \begin{bmatrix} e^{\lambda_1 t} & 0 & 0 & 0 & \dots \\ 0 & e^{\lambda_2 t} & 0 & 0 & \dots \\ 0 & 0 & e^{\lambda_3 t} & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & 0 & e^{\lambda_n t} \end{bmatrix} \begin{bmatrix} w_1^T \\ w_2^T \\ w_3^T \\ \dots \\ w_n^T \end{bmatrix}, \quad (3.2-12)$$

where v_i is the i th column of M (the i th eigenvector) and w_i^T is the i th row of M^{-1} . Then, by definition, “vectors” w_i are orthonormal with the eigenvectors, that is,

$$w_i^T v_j = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases} \quad (3.2-13)$$

It is also easy to show that the vectors w_i are actually the *left eigenvectors* of A , that is, the right eigenvectors of A^T . If (3.2-12) is postmultiplied by the initial-condition

vector x_0 , the homogeneous part of the solution of the continuous-time state equation is obtained:

$$e^{At}x_0 = [v_1 \ v_2 \ \cdots \ v_n] \begin{bmatrix} e^{\lambda_1 t} (w_1^T x_0) \\ e^{\lambda_2 t} (w_2^T x_0) \\ \dots\dots\dots \\ e^{\lambda_n t} (w_n^T x_0) \end{bmatrix}, \quad (3.2-14)$$

where the terms $(w_i^T x_0)$ are scalar products. Equation (3.2-14) can be rewritten as

$$e^{At}x_0 = \sum_{i=1}^n (w_i^T x_0) e^{\lambda_i t} v_i$$

If this same procedure is followed but with (3.2-12) postmultiplied by $Bu(\tau)$, the forced component of the response is obtained. The complete response is therefore given by

$$x(t) = \sum_{i=1}^n v_i e^{\lambda_i t} (w_i^T x_0) + \sum_{i=1}^n v_i \int_0^t e^{\lambda_i(t-\tau)} (w_i^T Bu(\tau)) d\tau \quad (3.2-15)$$

In effect, Equation (3.2-15) uses the n linearly independent eigenvectors as a basis for the n -dimensional space, associates a characteristic mode with each basis vector, and shows the fixed component of $x(t)$ in each direction. If, for example, the initial-condition vector lies in a direction such that a scalar product $(w_i^T x_0)$ is zero, the mode $e^{\lambda_i t}$ will not appear in the homogeneous response. According to (3.2-13), this will occur if the initial-condition vector lies along any eigenvector other than the i th. Similarly, if the scalar product $(w_i^T Bu(\tau))$ is zero, the mode $e^{\lambda_i t}$ will not be excited by the input. In Chapters 5 and 6 we discuss the related idea of *controllability* of the modes and show how it is determined by the A - and B -matrices. If we form the output vector by premultiplying (3.2-15) by the C -matrix, we see that whether or not a mode appears in the output depends on the C - and A -matrices. This is the concept of *observability*, also described in Chapters 5 and 6.

Equation (3.2-15) also shows that if we examine the i th eigenvector, its nonzero elements will indicate to what extent each state variable participates in the i th mode. The relative involvement of the different variables is complicated by the fact that the eigenvector elements can, in general, each have different units.

Laplace Transform Solution of LTI State Equations

The Laplace transform (LT) maps real functions of time into functions of the complex variable s , which is written in terms of its real and imaginary parts as $s = \sigma + j\omega$ and has the dimensions of a complex frequency variable. In the complex frequency domain (or s -domain) the functions of s can be manipulated algebraically into recognizable, known transforms and then mapped back into the time domain. Laplace

transform theory is thoroughly covered in many undergraduate texts (Ogata, 1998), and here we will only review two important points concerning applicability.

Two different ways of applying the LT will now be described. First, in general, analysis of a dynamic system will produce a set of simultaneous integro-differential equations. These equations should be transformed immediately, so that the initial-condition terms that appear from applying the LT differentiation and integration theorems represent the initial stored energy. If the integral terms are removed by differentiation, derivatives of the system input may appear. These give rise to extra initial-condition terms when the differential equations are transformed and can make it difficult to solve for all of the required initial conditions on the dependent variable. Therefore, we avoid transforming the general ODE in Equation (3.2-5) if there are nonzero initial conditions and derivatives on the right-hand side.

The second method of using the LT applies to initially unexcited systems; no initial-condition terms appear after transforming. If the equations have been differentiated so that derivatives of the input appear, the input initial-condition terms must cancel with the initial conditions on the output. Therefore, if the system is described by the differential equation (3.2-15), this equation can be transformed with zero initial conditions. For circuits there is actually no need to write the differential equations because, with no initial stored energy, the system elements can be represented by transform impedances (or admittances). Circuit analysis rules will then yield s -domain equations that can be solved for the output transform. This method will be addressed in the next section.

We will denote Laplace transforms by uppercase symbols, thus

$$X(s) = \mathcal{L}[x(t)], \quad U(s) = \mathcal{L}[u(t)]$$

The LTI state equations have no derivatives of $u(t)$ and can be solved by Laplace transforming (3.1-2a):

$$sX(s) - x(0^+) = AX(s) + BU(s) \quad (3.2-16)$$

$$\therefore X(s) = (sI - A)^{-1}[x(0^+) + BU(s)]$$

$$Y(s) = C(sI - A)^{-1}[x(0^+) + BU(s)] + DU(s) \quad (3.2-17)$$

Because there are no input derivatives, this solution requires n initial conditions on $x(t)$ only, and these would specify the initial stored energy in our earlier examples. The symbol 0^+ indicates the limit when the time origin is approached from the right-hand side.

If we compare the transform solution for $X(s)$ with the time-domain solution for $x(t)$, Equation (3.2-8), we see that the transition matrix is given by

$$e^{At} = \mathcal{L}^{-1}[(sI - A)^{-1}] \quad (3.2-18)$$

The LT solutions (3.2-16) and (3.2-17) are not well suited to machine computation, and hand computation involves a prohibitive amount of labor for other than low-order dynamic systems. Therefore, the LT solutions are mainly of interest as a complex number description of system properties, as we will now see.

3.3 TRANSFER FUNCTION MODELS

Derivation of Transfer Functions; Poles and Zeros

Consider the system described by the n th-order ODE (3.2-5) and transform with zero initial conditions. Solving algebraically for $Y(s)$,

$$Y(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0} U(s), m \leq n \quad (3.3-1)$$

The polynomial rational function relating $Y(s)$ to $U(s)$ is the *transfer function* of this SISO system. If we have obtained the transform $U(s)$, then, using the partial fraction technique (see below), the right-hand side of (3.3-1) can be broken down into a sum of transforms corresponding to known time functions, and hence $y(t)$ can be found as a sum of time functions.

From (3.2-17) we see that the transfer function obtained from the LTI state equations is a matrix expression and, for a MIMO system with p outputs and m inputs, the $(p \times m)$ transfer function matrix is given by

$$G(s) = C(sI - A)^{-1}B + D \quad (3.3-2)$$

It is easy to show that a transfer function matrix is unchanged by a nonsingular transformation of the state variables. Equation (1.3-13b) represents such a transformation, and if the coefficient matrices from that equation are substituted into (3.3-2), the result is

$$\begin{aligned} G(s) &= CL^{-1}(sI - LAL^{-1})^{-1}LB + D \\ &= C[L^{-1}(sI - LAL^{-1})L]^{-1}B + D \\ &= C(sI - A)^{-1}B + D, \end{aligned}$$

which is (3.3-2) again. Therefore, we can choose a new set of state variables for a system, and the transfer function will be unchanged.

We will now review some other important properties of transfer functions. A matrix inverse can be expressed in terms of the adjoint matrix with its elements divided by its determinant, and so (3.3-2) can be written as

$$G(s) = \frac{C \operatorname{adj} (sI - A)B + D|sI - A|}{|sI - A|} \quad (3.3-3)$$

The transfer function from the j th input to the i th output is the (i, j) th element of $G(s)$, and this is the SISO transfer function $G_{ij}(s)$. A SISO transfer function can therefore be written as

$$G_{ij}(s) = \frac{c_i \operatorname{adj} (sI - A)b_j + d_{ij}|sI - A|}{|sI - A|}, \quad (3.3-4)$$

where c_i and b_j are, respectively, the i th row of C and the j th column of B . This transfer function is a rational function of two polynomials. The elements of the adjoint $\text{adj}(sI - A)$ are, by definition, cofactors of $|sI - A|$ and are therefore polynomials in s of degree $n - 1$ or lower. The determinant $|sI - A|$ is a polynomial of degree n . When (3.3-4) is written out as the ratio of two polynomials, it will correspond exactly to the SISO transfer function in (3.3-1) that we obtained from the n th-order ODE.

In (3.3-4), when $d_{ij} = 0$, the *relative degree* (denominator degree minus numerator degree) of this transfer function is unity or higher. When $d_{ij} \neq 0$, the relative degree is zero and, referring to the simulation diagram in Figure 3.2-5, we see that d_{ij} forms a “direct-feed” path from input to output. This means that the system output immediately begins to follow an input, and then the modes of the system respond and begin to modify the output.

If the polynomials in the transfer function (3.3-4) are factored, we obtain

$$G_{ij}(s) = \frac{k(s + z_1)(s + z_2) \dots (s + z_m)}{(s + p_1)(s + p_2) \dots (s + p_n)} \quad (3.3-5a)$$

Or, equivalently,

$$G_{ij}(s) = \frac{a_1}{s + p_1} + \frac{a_2}{s + p_2} + \dots + \frac{a_n}{s + p_n} \quad (3.3-5b)$$

The denominator factors are the factors of $|sI - A|$, and it is evident from (3.3-3) and (3.3-4) that all of the individual SISO transfer functions have the same denominator factors, given by the roots of the n th-degree polynomial equation

$$|sI - A| = 0 \quad (3.3-6)$$

The roots $\{-p_i\}$ are called the *poles* of the transfer function and, at these values of s , the transfer function becomes infinite in magnitude.

Equation (3.3-6) is also the defining equation for the eigenvalues of the A -matrix. Therefore, the system poles are given by the eigenvalues of A . We know from Chapter 1 that the eigenvalues of a real system are real or occur in complex conjugate pairs and, according to Equation (3.2-15), determine the natural modes of a system. The position of the poles in the complex s -plane will determine the time constant of a real mode or the frequency of oscillation and exponential damping factor of a complex mode. Also, poles in the right-half s -plane will correspond to exponentially growing functions (unstable behavior). For this reason graphical operations in the s -plane are important to us.

Equation (3.3-5b) is the *partial fraction expansion* of the transfer function, and a coefficient a_i is the *residue* in the pole at $-p_i$ (Ogata, 1998). In the case of complex poles the partial fractions combine as conjugate pairs. Poles of multiplicity k require a numerator of degree $(k - 1)$ and can be further broken down into a finite expansion in inverse powers of $(s + p_i)$.

The *zeros* of the individual SISO transfer functions are the positions in the s -plane where their magnitudes become zero, that is, the roots $\{-z_i\}$ of the numerator polynomial of (3.3-4). The number of zeros of each SISO transfer function will

range from zero to n , depending on the relative degree of the transfer function. Equations (3.3-3) and (3.3-4) show that the transfer function zeros depend on the B , C , and D matrices, and Equation (3.2-15) shows how the B and C matrices, respectively, play a role in the excitation of a mode and its appearance in the system output. When a response transform is expanded in partial fractions, we see that the partial fraction coefficients depend on the numerator polynomial and hence on the zeros. The partial fraction terms correspond to the modes, and the zeros determine how strongly the modes are represented in the response. If all of the poles of the transform of the system input coincide with zeros of the SISO transfer function, there will be no forced response at the output of the system.

It is known that the values of polynomial roots are very sensitive to changes in the polynomial coefficients, and Equations (3.3-3) and (3.3-4) are the starting points of algorithms used to change the computation of zeros into a much more numerically stable eigenvalue problem (Emami-Naeini and Van Dooren, 1982). Transfer function-related analysis and design tools are based on poles and zeros, and we will have little use for the polynomial form of the transfer function.

When $G_{ij}(s)$ is in the factored form (3.3-5a), with all coefficients of s equal to unity, or expressed as the ratio of two monic polynomials, the coefficient k is known as the *static loop sensitivity*. Note that if there are no poles or zeros at the s -plane origin, then, when $s = 0$, the magnitude of the transfer function (the dc gain) is finite and is determined by k and the zero and pole positions. If the relative degree is zero, k is the value of the transfer function at large values of s (the high-frequency gain).

From this point on we will drop the subscripts on $G(s)$, and it will be obvious from the context whether G represents a matrix or a scalar transfer function.

Interpretation of the SISO Transfer Function

The complex exponential function e^{st} , $s = \sigma + j\omega$, possesses time derivatives of all orders, all of the same form, and if we could apply it as an input to the SISO system described by the ODE (3.2-5), the particular solution of the ODE would be a time function of the same form. Furthermore, the solution would be given by an expression exactly like the transfer function (3.3-1), and we could use the response to e^{st} as a definition of a transfer function. The effect of the system on a specific exponential function $e^{s_1 t}$ could be found by evaluating $G(s_1)$, given by

$$G(s_1) = \frac{k(s_1 + z_1)(s_1 + z_2) \dots (s_1 + z_m)}{(s_1 + p_1)(s_1 + p_2) \dots (s_1 + p_n)} \quad (3.3-7)$$

The numerator and denominator factors in this transfer function can be represented in magnitude and phase by vectors in the s -plane, drawn from the zeros and poles, respectively, to the point s_1 . This is illustrated by the example shown in Figure 3.3-1 and, in general,

$$|G(s_1)| = k \frac{\text{product of lengths of vectors from zeros to } s_1}{\text{product of lengths of vectors from poles to } s_1} \quad (3.3-8a)$$

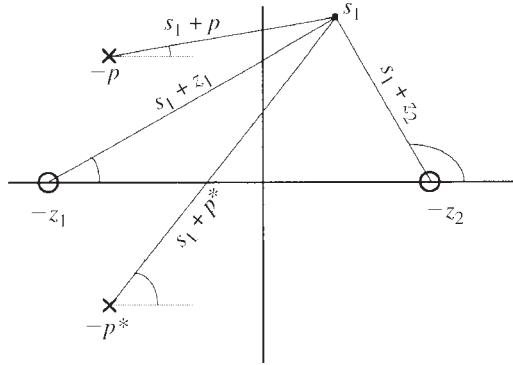


Figure 3.3-1 s -Plane vectors representing pole-zero factors.

and

$$\begin{aligned} \angle G(s_1) = & \text{sum of angles of vectors from zeros to } s_1 \\ & - \text{sum of angles of vectors from poles to } s_1 \end{aligned} \quad (3.3-8b)$$

Because complex poles and zeros occur in conjugate pairs,

$$G(s_1^*) = G^*(s_1), \quad (3.3-9)$$

where “*” denotes the conjugate, and this is clearly illustrated by drawing the appropriate vectors. This interpretation of the transfer function is particularly useful when s_1 is a point on the s -plane $j\omega$ axis. A real sinusoid can be represented as

$$\cos(\omega_1 t) = \frac{1}{2}(e^{j\omega_1 t} + e^{-j\omega_1 t}) \quad (3.3-10)$$

The particular solution of the ODE, with this input, is given by

$$\begin{aligned} y(t) &= \frac{1}{2} \left[G(j\omega_1) e^{j\omega_1 t} + G(-j\omega_1) e^{-j\omega_1 t} \right] \\ \therefore y(t) &= \text{Re}\{G(j\omega_1) e^{j\omega_1 t}\} = |G(j\omega_1)| \cos(\omega_1 t + \angle G(j\omega_1)), \end{aligned} \quad (3.3-11)$$

where Re is the real-part operator, and we have made use of (3.3-9). The sinusoidal input (3.3-10) was not switched on at some particular time; mathematically it has existed for all time, and the solution (3.3-11) represents the *steady-state response* to a sinusoidal input of the system whose transfer function is $G(s)$.

The plots of $|G(j\omega)|$ and $\angle G(j\omega)$, as ω is varied from low to high frequency, are called the *magnitude* and *phase* of the system *frequency response*. The vector interpretation (3.3-8) shows that if there is a pair of complex poles near the imaginary axis, the vectors drawn from these poles will become very short in length over some range of frequencies, and there will be a peak in the magnitude of $G(j\omega)$ and rapid changes in its phase. This is the phenomenon of *resonance*, in which a natural mode

of a system is excited by the input to the system. Conversely, if there is a pair of complex zeros close to the $j\omega$ axis, the magnitude will pass through a minimum and the phase will again change rapidly. These effects are discussed more thoroughly in the section on frequency response.

A transfer function carries some very basic information about the way in which an aircraft (or any other system) will respond that is usually not obvious to the student. Two theorems that are fundamental in interpreting the transfer function are the Laplace transform initial- and final-value theorems:

$$\text{initial value : } f(0^+) \equiv \lim_{t \rightarrow 0} f(t) = \lim_{s \rightarrow \infty} sF(s) \quad (3.3-12a)$$

$$\text{final value : } f(\infty) \equiv \lim_{t \rightarrow \infty} f(t) = \lim_{s \rightarrow 0} sF(s) \quad (3.3-12b)$$

As an example of these theorems, consider the response of a system to a unit-step function. A useful notation for functions that start at $t=0$ has been given by (DeRusso et al., 1965), and includes the unit step function

$$U_{-1}(t) \equiv \begin{cases} 0, & t < 0 \\ 1, & t > 0 \end{cases}$$

$U_{-1}(t)$ is undefined at $t = 0$ and has a Laplace transform $1/s$ (Ogata, 1998). The symbols U_{-2} , U_{-3} denote, respectively, a unit ramp and unit parabola, and U_0 denotes a unit impulse function. Now let the unit step, occurring at $t = 0$, be the input to the transfer function:

$$G(s) = \frac{-(s - \alpha)}{s^2 + s + 1}, \quad \alpha > 0 \quad (3.3-13)$$

The transfer function has a relative degree $r = 1$, and this makes the initial value of the step response zero:

$$f(0^+) = \lim_{s \rightarrow \infty} \left[s \frac{-(s - \alpha)}{s^2 + s + 1} \frac{1}{s} \right] = 0$$

The transform of the derivative is :

$$\mathcal{L}[\dot{f}(t)] = sF(s) - f(0^+) \Rightarrow sF(s)$$

and the initial value of the derivative is

$$\dot{f}(0^+) = \lim_{s \rightarrow \infty} [s^2 F(s)] = -1$$

The final value of the step response is

$$f(\infty) = \lim_{s \rightarrow 0} [sF(s)] = \alpha$$

The transfer function (3.3-13) has a sign difference between its behavior at small and large s ; $G(0) = \alpha$ (positive dc gain) and $G(\infty) = -1/s$. From the above analysis, the consequences are that the step response starts out in the negative direction but finishes with a positive value. The transfer function numerator factor $(s - \alpha)$, $\alpha > 0$, corresponds to a zero in the right-half s -plane, and this is the cause of the above behavior.

If a transfer function contains right-half-plane zeros, it is called *non-minimum phase* (NMP), and the initial response to a step input may have the opposite sign to the final response (depending on the number of NMP zeros). This is an undesirable type of response from the point of view of a human operator. NMP zeros are also undesirable in feedback controller design since, as we will see later from “root-locus” plots, a right-half-plane zero tends to attract the closed-loop poles to the right-half s -plane. These types of zeros occur when there are two or more different paths to the system output, or two or more different physical mechanisms, producing competing output components.

When there are left-half-plane zeros near the origin, these tend to promote an overshoot in the response to a step input, which is again undesirable. Problem 3.3-2 illustrates the effects of both NMP zeros and left-half-plane zeros close to the origin.

By writing the simple differential equations for an ideal integrator or differentiator, and transforming them, we can derive their transfer functions. Thus, the transfer function of an integrator consists of a single pole at the origin, and a differentiator corresponds to a single zero at the origin. In a block diagram using transform-domain quantities, we will represent integrators and differentiators by boxes containing, respectively, $1/s$ and s .

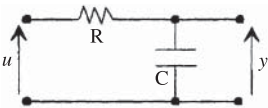
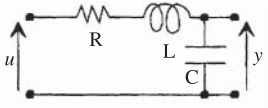
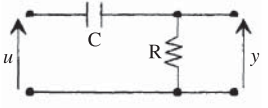
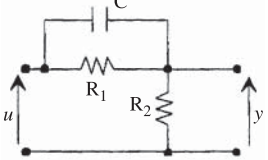
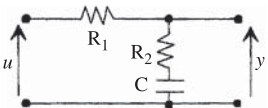
Finally, when transfer function poles and zeros are close together in the s -plane, the residue in the poles tends to be small (i.e., the coefficients of the corresponding partial fraction terms are small). A pole can be effectively canceled out of the transfer function by a nearby zero in this way.

Transfer Function Examples and Standard Forms

Table 3.3-1 shows a number of standard transfer functions, and their state equations, that are used as either models or compensating networks in control systems design (see Chapter 4). Simple electrical networks that can be used to realize these transfer functions are also shown. The voltage transfer functions (assuming no source and output loading effects) can be derived from the networks by representing the network elements by their Laplace transform impedances (i.e., $1/sC$ for a capacitor, sL for an inductor). They can then be analyzed in the same way as dc circuits. The transfer functions are written here in *standard form*. This requires all numerator and denominator factors to be written as either $(s\tau)$, $(s\tau + 1)$, where τ is a time constant, or the second-order standard form given below. The state equations can be derived from the transfer functions by the methods given earlier.

Four of the networks in Table 3.3-1 have only a single energy storage element, are modeled with a single state variable, and hence have only a single real pole in their transfer functions. The standard form for a transfer function factor corresponding to

TABLE 3.3-1 Network Transfer Functions and State Equations

Network	Transfer Function	State Equations
 <p style="text-align: center;">SIMPLE LAG</p>	$\frac{1}{1 + s\tau}, \quad \tau = CR$	$\dot{x} = \frac{u - x}{\tau}$ $y = x$
 <p style="text-align: center;">QUADRATIC LAG</p>	$\frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$ $\omega_n^2 = \frac{1}{LC}, \quad \zeta = \frac{R}{2} \sqrt{\frac{C}{L}}$	$\dot{x}_1 = x_2$ $\dot{x}_2 = -\omega_n^2 x_1 - 2\zeta\omega_n x_2 + \omega_n^2 u$ $y = x_1$
 <p style="text-align: center;">SIMPLE LEAD</p>	$\frac{s\tau}{1 + s\tau}, \quad \tau = CR$	$\dot{x} = \frac{u - x}{\tau}$ $y = u - x$
 <p style="text-align: center;">LEAD COMPENSATOR</p>	$\frac{s + z}{s + p}, \quad z = \frac{1}{CR_1},$ $\frac{z}{p} = \frac{R_2}{R_1 + R_2}$	$\dot{x} = u - px$ $y = u + (z - p)x$
 <p style="text-align: center;">LAG COMPENSATOR</p>	$\frac{p}{z} \frac{s + z}{s + p}, \quad z = \frac{1}{CR_2},$ $\frac{p}{z} = \frac{R_2}{R_1 + R_2}$	$\dot{x} = u - px$ $y = \frac{p}{z}[u + (z - p)x]$

a single real pole or zero is the dimensionless factor $(\tau s + 1)$, and the pole or zero is at $s = -1/\tau$. As an example, we will derive the transfer function of the network identified as a “simple lead.”

With the restriction that any load connected to the output of the network must draw negligible current, the same current flows in the series (connecting input and output) branch as in the shunt (across the output terminals) branch. The voltage transfer function $G(s)$ is then simply the impedance of the shunt branch divided by the sum of the shunt and series impedances:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{R}{R + 1/sC} = \frac{s\tau}{s\tau + 1}, \quad \text{where } \tau = CR \quad (3.3-14)$$

This transfer function has a zero at the s -plane origin and a pole at $s = -1/\tau$. We could immediately write down the differential equation relating input and output voltages and recognize that the derivative of the input is present (the transfer function relative degree is zero). We will therefore find the state equations by a method that is similar to that used for Equation (3.2-6).

Rewrite (3.3-14) with an auxiliary variable $Z(s)$ as

$$\frac{Y(s)}{s\tau} = \frac{U(s)}{s\tau + 1} \equiv Z(s) \quad (3.3-15)$$

Now draw a simulation diagram with, in general, a chain of integrators whose outputs, starting from the last one, are Z , sZ , s^2Z , and so on. Here we need only a single integrator. The $U(s)$ equation in (3.3-15) gives

$$s\tau Z(s) = U(s) - Z(s),$$

which allows the input connections of the integrator to be established, as shown in Figure 3.3-2. Similarly, the $Y(s)$ equation in (3.3-15) allows the simulation diagram output connections to be established. The final step is to assign state variables to the outputs of the integrators and write the state equations by inspection of the simulation diagram. In this case, Figure 3.3-2 gives the result shown in Table 3.3-1,

$$\dot{x} = (u - x)/\tau, \quad y = (u - x)$$

This method of finding state equations from transfer functions or ODEs extends readily to higher-order systems; it leads to an A -matrix in companion form. Therefore, for practical purposes we restrict it to low-order systems.

Next consider the *quadratic-lag* circuit in Table 3.3-1. This has two energy storage elements and requires two state variables. Because there is again only a single loop, the voltage transfer function can again be found from the branch impedances:

$$G(s) = \frac{1/sC}{sL + R + 1/sC} = \frac{1/(LC)}{s^2 + s(R/L) + 1/(LC)}$$

When a transfer function has the possibility of a pair of complex poles or zeros, it is usually convenient to represent these by a real second-order factor rather than a pair of complex first-order factors. A second-order transfer function factor is written

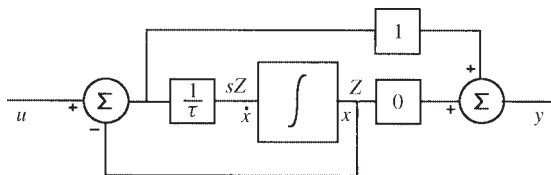


Figure 3.3-2 Simulation diagram for a simple lead.

as $(s^2 + 2\zeta\omega_n s + \omega_n^2)$, where ω_n is called the *natural frequency* and ζ is called the *damping ratio*. Using this form, the above transfer function becomes

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \quad (3.3-16)$$

where

$$\omega_n = 1/\sqrt{LC}, \quad \zeta = \frac{1}{2}R\sqrt{C/L}$$

Equation (3.3-16) is the standard second-order form for a complex pole pair; for complex zeros this form is inverted. Note that it has a dc gain of unity. Transfer functions can always be written in terms of the standard forms, and in the next sections we explore the properties of some standard forms rather than specific systems.

Frequency Response

Frequency response was defined in connection with Equations (3.3-10) and (3.3-11). Here we look at the frequency response of some standard-form transfer functions. An example of a first-order transfer function is

$$G(s) = \frac{s\tau_1}{s\tau_2 + 1} \quad \text{or} \quad G(s) = \frac{s\tau_1/\tau_2}{s + 1/\tau_2} \quad (3.3-17)$$

The first transfer function is in standard form for plotting frequency response, and the second matches the vector representation described earlier. Visualizing the vectors, we can see immediately that the frequency response starts from zero magnitude and 90° leading phase and, at high frequencies, it becomes constant in magnitude with zero phase angle. The transfer function itself shows that the high-frequency value of the magnitude is equal to τ_1/τ_2 .

Using the first form of the transfer function, with $s = j\omega$, the magnitude and phase are given by

$$|G(j\omega)| = \frac{\omega\tau_1}{\sqrt{(1 + \omega^2\tau_2^2)}}, \quad \angle G(j\omega) = \pi/2 - \tan^{-1}(\omega\tau_2) \quad (3.3-18)$$

An octave is a two-to-one frequency interval, and a decade is a ten-to-one interval; experience shows that the extent of the frequency range of interest for practical systems is usually a few decades. If the frequency-response plots are made with a logarithmically spaced frequency scale, each decade occupies the same width, and features that would be lost on a linear scale are visible.

In the case of the magnitude plot, it is found that plotting the logarithm of the magnitude is very convenient for engineering purposes. This is because overall gain can be found by adding log-magnitudes, but also because very often mechanical, electrical, or physiological effects are more nearly linearly related to the logarithm of a power ratio than to the direct power ratio. An example is the Weber-Fechner law of psychology, which states that the human ear responds logarithmically. The

logarithmic units most commonly used are the bel and the decibel (1 bel = 10 dB); the decibel is given by 10 times the common logarithm of the relevant power ratio, or 20 times the corresponding amplitude ratio. In engineering measurements, 0.1 dB represents good resolution, and a 60- to 80-dB range is roughly the limit of linear operation for many systems. Plots of decibel magnitude and linear phase, plotted against logarithmically spaced frequency, are known as *Bode plots*.

Taking the log-magnitude in Equation (3.3-18) gives

$$20 \log_{10}(|G(j\omega)|) = 20 \log_{10}(\omega\tau_1) - 10 \log_{10}(1 + \omega^2\tau_2^2)$$

The first term on the right increases by 20 dB for every tenfold increase in ω ; k -fold increases in frequency all occupy the same width on a logarithmic-spaced frequency scale. Therefore, this term has a straight-line Bode plot with a slope of 20 dB/decade (6 dB/octave). The second term on the right can be approximated as follows:

$$10 \log_{10}(1 + \omega^2\tau_2^2) \begin{cases} \approx 20 \log_{10}(\omega\tau_2), & \omega^2\tau_2^2 \gg 1 \\ = 3.01 \text{ dB}, & \omega\tau_2 = 1 \\ \approx 0 \text{ dB} & \omega^2\tau_2^2 \ll 1 \end{cases}$$

These results show that this term has asymptotes given by a 0-dB line at low frequency and a line with a slope of 20 dB/decade at high frequency. At the “corner frequency” (or break frequency), $\omega = 1/\tau_2$, the term is 3 dB from the 0-dB asymptote, and at an octave above and below the corner frequency, it is 1 dB from its asymptotes. The phase plot asymptotically approaches 90° at low frequencies and zero at high frequencies and passes through 45° at the corner frequency. It is much more spread out, being about 6° from its asymptotic values at a decade above and below the corner frequency. These decibel values and phase values are to be subtracted because this term came from the denominator of the transfer function. Exact Bode plots of the transfer function (3.3-17), with $\tau_1 = 10$ and $\tau_2 = 2$, are shown in Figure 3.3-3.

Consider next a quadratic transfer function factor:

$$s^2 + 2\zeta\omega_n s + \omega_n^2 \quad (3.3-19a)$$

$$= (s + \zeta\omega_n)^2 + \omega_n^2(1 - \zeta^2) \quad (3.3-19b)$$

The quadratic formula shows that this factor represents complex conjugate roots when $\zeta^2 < 1$, and (3.3-19b) shows that the roots are given by

$$s = -\zeta\omega_n \pm j\omega_n \sqrt{1 - \zeta^2} \equiv -\zeta\omega_n \pm j\omega_d, \quad (3.3-20)$$

where $\omega_d \equiv \omega_n(1 - \zeta^2)^{1/2}$ is the *damped frequency*. Figure 3.3-4 shows the s -plane vectors that could be used to evaluate the frequency response of a quadratic factor with complex roots. Complex poles are shown in the figure, but the following

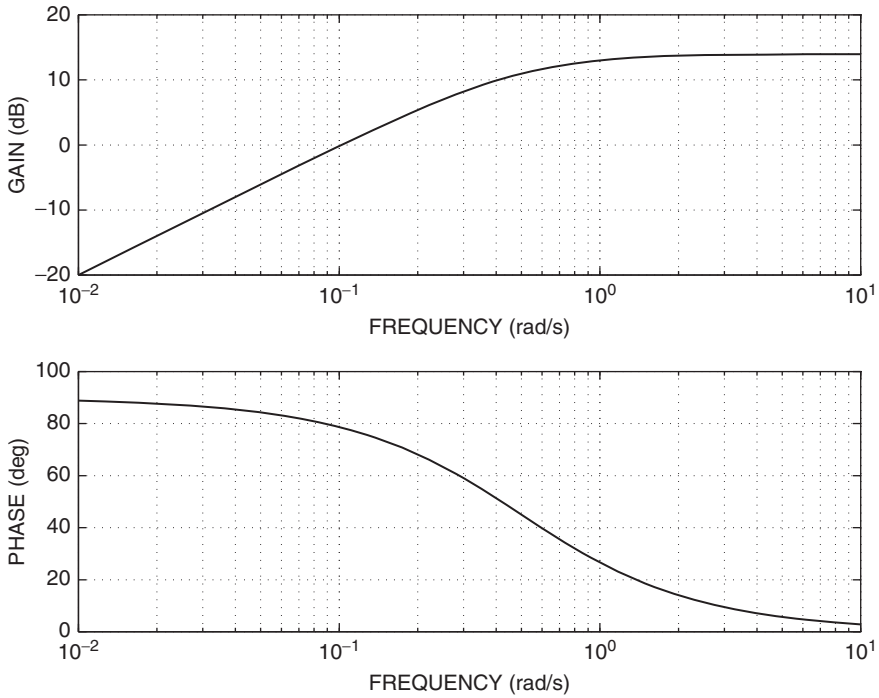


Figure 3.3-3 Bode plots for a simple lead.

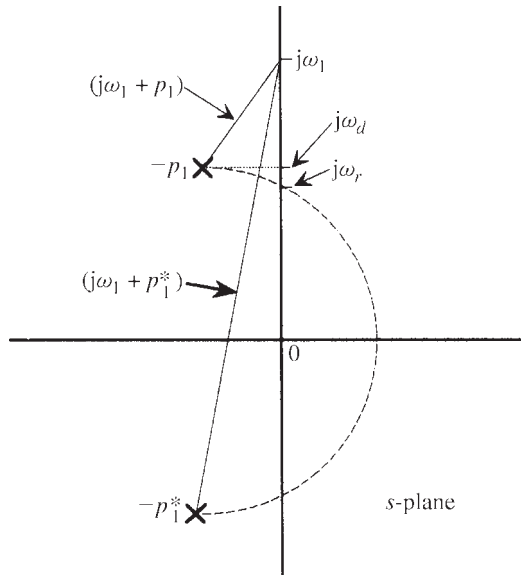


Figure 3.3-4 Geometrical properties of a quadratic lag.

results also apply to complex zeros. The *resonant frequency*, ω_r , is the frequency at which the product of the lengths of the vectors is at a minimum and is given by the imaginary-axis intersection of the semicircle whose diameter is the line joining the poles. This is because the vectors drawn to an imaginary-axis point are the sides of a constant-area triangle (constant base, constant height). At the point $j\omega_r$, the angle at the apex of the triangle reaches a maximum of 90° , so the product of its two sides is at a minimum (area = product of two sides and sine of included angle). By constructing another right triangle whose hypotenuse is a line from $-\zeta\omega_n$ on the real axis to $j\omega_r$ on the imaginary axis, we find that ω_r is given by

$$\omega_r = \omega_n \sqrt{1 - 2\zeta^2} \quad (3.3-21)$$

and so the resonant frequency approaches the natural frequency as the damping ratio becomes small. There is no peak or dip in the frequency response of a complex pair of poles or zeros when $\zeta > 1/\sqrt{2}$.

We can apply these results to the quadratic-lag standard form, (3.3-16). Its magnitude and phase are given by

$$|G(j\omega)| = \frac{\omega_n^2}{\sqrt{[(\omega_n^2 - \omega^2)^2 + 4\zeta^2\omega^2\omega_n^2]}} \quad (3.3-22a)$$

$$\angle G(j\omega) = -\tan^{-1}(2\zeta\omega/\omega_n, 1 - \omega^2/\omega_n^2) \quad (3.3-22b)$$

The four-quadrant inverse-tangent function is necessary because the phase angle, by which the output lags behind the input, lies between zero and 180° . At resonance, the magnitude of the quadratic-lag standard form is found by substituting (3.3-21) into (3.3-22a):

$$|G(j\omega_r)| = \frac{1}{2\zeta\sqrt{1 - \zeta^2}}, \quad \zeta < 1/\sqrt{2} \quad (3.3-23)$$

Figure 3.3-5 shows the Bode magnitude and phase plots for a quadratic lag, with $\omega_n = 1.0$. The asymptotes of the magnitude plot are now found to be a 0-dB line and a line with a slope of -40 dB/decade, intersecting the 0-dB line at ω_n . When the damping ratio is small, there is a large deviation from the asymptotes near ω_n .

Finally, consider a transfer function $(s + z)/(s + p)$. This has corner frequencies at $\omega = z$ and $\omega = p$, and the s -plane vectors show that its gain varies from z/p at zero frequency to unity at infinite frequency. If $z < p$, the gain will rise to unity and, if $z > p$, the gain will fall to unity. Rising gain is accompanied by a leading phase angle, and vice versa. On a logarithmic frequency scale, the maximum or minimum of the phase shift occurs midway between the pole and zero frequencies, and this is the geometric mean \sqrt{pz} . Other properties of this transfer function are derived in Section 3.9, where it is used for control system *compensation*. Figure 3.3-6 shows Bode plots for the leading-phase case, $z < p$.

Various systems, including control systems, audio amplifiers, and sensors and measurement devices, can have their performance specified in terms of frequency response. The usual criteria are the *bandwidth*, *peak magnification*, and amount of

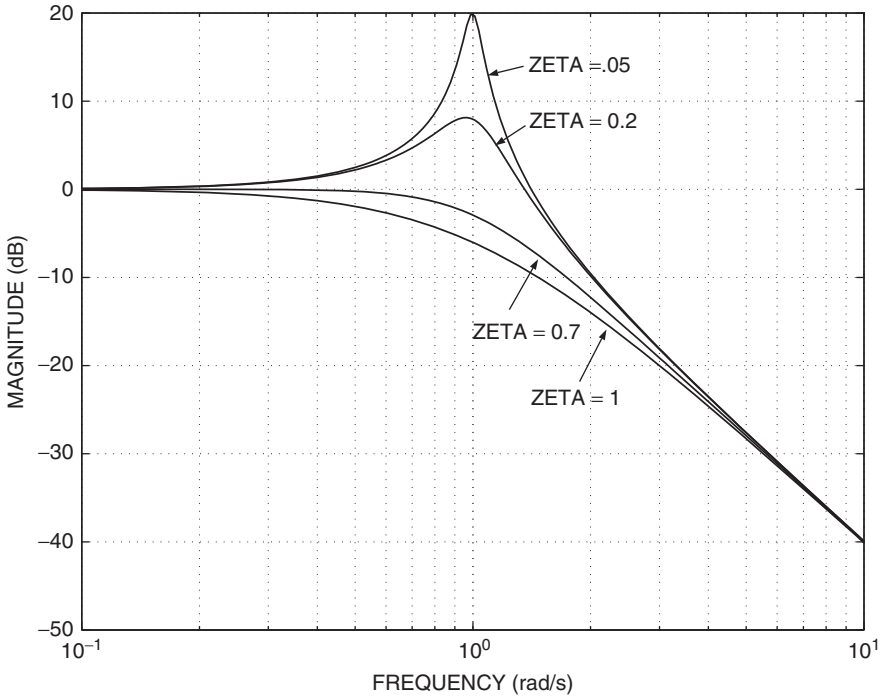


Figure 3.3-5a Bode gain plot for a quadratic lag.

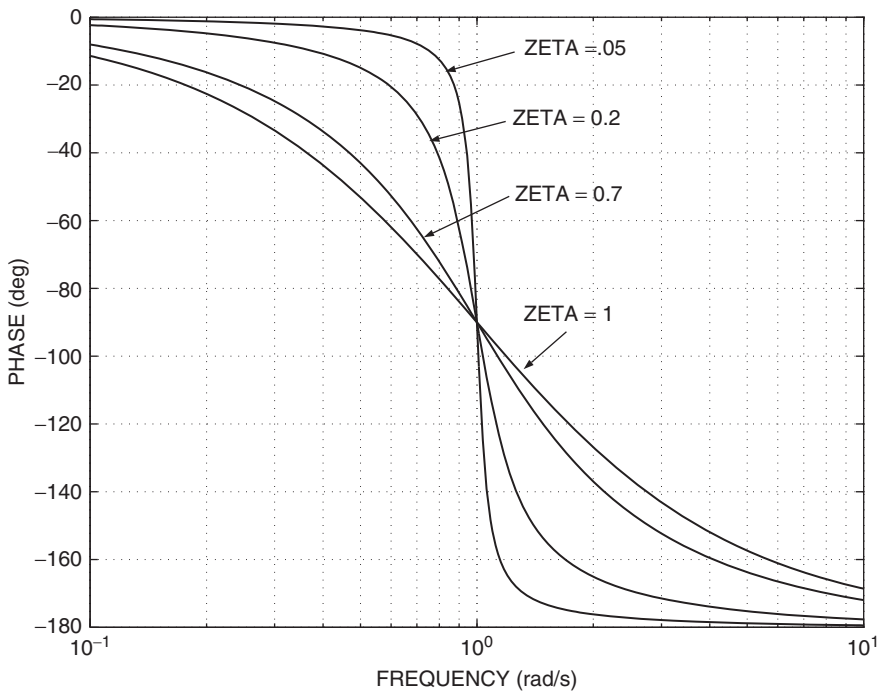


Figure 3.3-5b Bode phase plot for a quadratic lag.

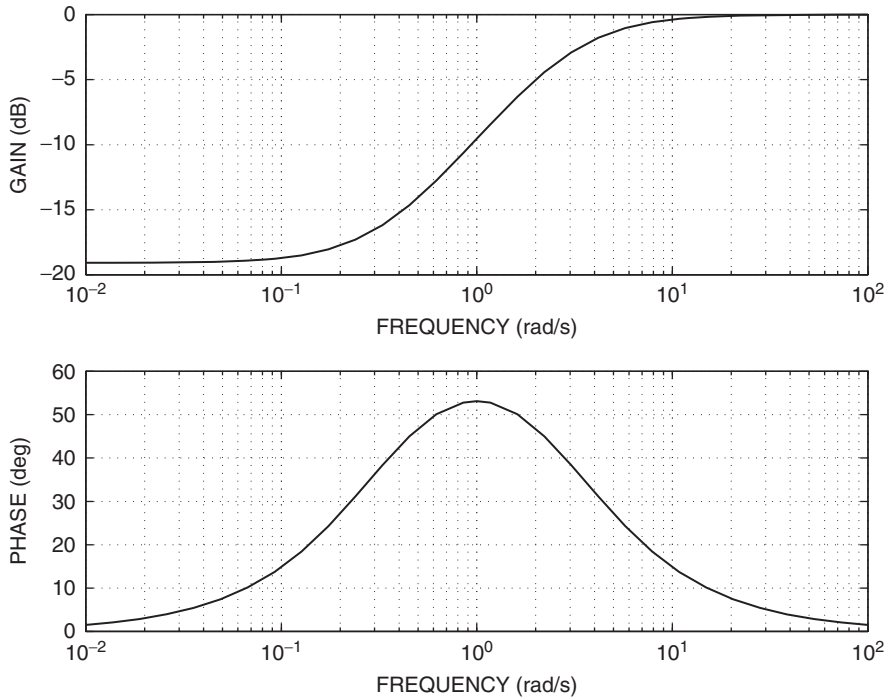


Figure 3.3-6 Bode plots for a lead compensator.

phase shift at some frequency. A system whose frequency response extends down to zero frequency and “rolls off” with increasing frequency (e.g., an integrator or a simple lag) is called a *low-pass* system. Most control systems behave in this way. Similar definitions apply to *high-pass* and *band-pass* systems. If a low-pass system has a level frequency response at low frequency, we define the bandwidth to be the frequency at which the gain has fallen by 3 dB from its low-frequency value. As an example, the quadratic lag is a low-pass transfer function, it may have a resonant peak before it rolls off, and it can be shown to be “3 dB down” at the frequency

$$\omega_B = \omega_n \sqrt{[(1 - 2\zeta^2) + \sqrt{(4\zeta^4 - 4\zeta^2 + 2)}]} \quad (3.3-24)$$

Time Response

Here we will look briefly at the step response of the simple-lag and quadratic-lag transfer functions. The transfer function of the simple lag is given in Table 3.3-1; the transform of a unit step input, occurring at $t = 0$, is $U(s) = 1/s$, and so the output transform is

$$Y(s) = \frac{1/\tau}{s + 1/\tau} \frac{1}{s} \equiv \frac{1}{s} + \frac{-1}{s + 1/\tau}$$

The partial fraction terms on the right correspond to known time functions, and so $y(t)$ can be written down directly:

$$y(t) = (1 - e^{-t/\tau})U_{-1}(t), \quad (3.3-25)$$

where the unit step $U_{-1}(t)$ serves to define the answer to be zero for $t < 0$. Equation (3.3-25) shows that the response of a simple real-pole transfer function to a step input is an exponential growth from zero to a final value given by the dc gain times the magnitude of the step.

The unit-step response of the quadratic lag (3.3-16) is given by

$$Y(s) = \frac{\omega_n^2}{(s + \zeta\omega_n)^2 + \omega_n^2(1 - \zeta^2)} \frac{1}{s} \equiv \frac{1}{s} - \frac{s + 2\zeta\omega_n}{(s + \zeta\omega_n)^2 + \omega_n^2(1 - \zeta^2)},$$

where the partial fraction coefficients were determined by the method of “comparing coefficients.” The solution can now be written down from a knowledge of the Laplace transforms of sine and cosine functions of time and the complex domain shifting theorem. Using a trigonometric identity to combine the sine and cosine terms gives

$$y(t) = \left[1 - \frac{e^{-\zeta\omega_n t}}{\sqrt{1 - \zeta^2}} \sin(\omega_d t + \phi) \right] U_{-1}(t), \quad (3.3-26)$$

where

$$\phi = \cos^{-1}(\zeta)$$

Plots of this answer are shown in Figure 3.3-7 for several values of ζ . The graphs were plotted with $\omega_n = 1$; they apply to any natural frequency if the horizontal scale

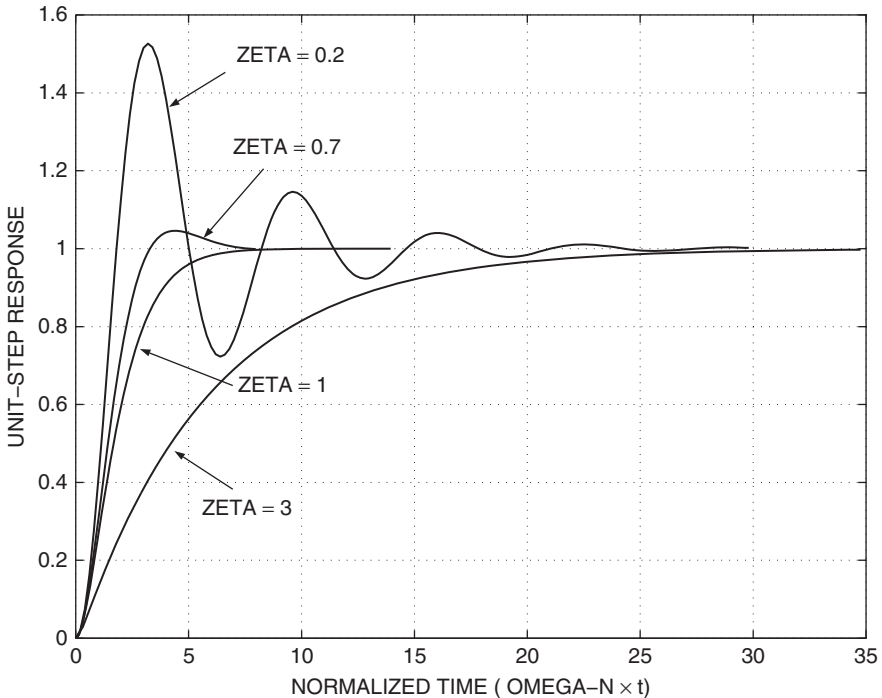


Figure 3.3-7 Step response of a quadratic lag.

is treated as “normalized time” $\omega_n t$. The case $\zeta = 1$ is defined to be *critically damped*; when the damping ratio is less than 1, the step response has an overshoot and the poles are complex.

These results are useful because very often a dynamic system has a *dominant pair* of complex poles (Nise, 1995), which essentially determine its behavior. The damping of a system can be specified by the *maximum overshoot*, or *settling time*, of its step response, while system speed of response can be specified by the *rise time*, or the *peak time*, of its step response. These performance figures can be related to the damping ratio and natural frequency of a dominant pair (Dorf and Bishop, 2001).

3.4 NUMERICAL SOLUTION OF THE STATE EQUATIONS

Introduction

The aircraft state equations are nonlinear, depend on experimentally determined data, and are subjected to arbitrary input signals. An analytical solution is out of the question, and numerical methods must be used to compute an aircraft trajectory. In general, the state vector of a physical system will move in a smooth, continuous manner in the n -dimensional state space because the state variables describe the energy stored in a physical system, and an instantaneous change in energy would require infinite power. Therefore, derivatives of the state variables will exist, and a Taylor series expansion can be used to predict the motion.

Numerical evaluation of the continuous trajectory implies that, given the initial condition $X(t_0)$ and control input $U(t)$, we must calculate discrete sequential values of the state:

$$X(t_0 + kT), \quad k = 1, 2 \dots \quad (3.4-1a)$$

that satisfy the state equations

$$\dot{X}(t) = f(X(t), U(t)) \quad (3.4-1b)$$

This is called the *initial-value problem*, and the *time step* T is usually chosen to be a fixed size. The state equations are not autonomous since the control input is an external input, and the time step must be made small enough that the control input can be approximated by a constant value during any interval kT to $(k + 1)T$. There are two classes of numerical solution methods for the initial-value problem, Runge-Kutta (RK) methods and linear multistep methods (LMMs), and these will now be described.

Runge-Kutta Methods

Consider the simplest ODE initial-value problem: a single first-order autonomous differential equation with a specified boundary condition,

$$\frac{dx}{dt} = f(x, t), \quad x(t_0) = x_0 \quad (3.4-2)$$

The problem of finding the discrete-solution values for (3.4-2) has an obvious connection to the Taylor series:

$$x(t_0 + T) = x(t_0) + T\dot{x}(t_0) + \frac{T^2}{2!}\ddot{x}(t_0) + \dots \quad (3.4-3)$$

The simplest RK method is *Euler integration*, which merely truncates the Taylor series after the first derivative. The Euler formula applied to (3.4-2) is

$$x_E(t_0 + T) \approx x(t_0) + Tf(x(t_0), t_0) \quad (3.4-4)$$

This formula is not very accurate unless very small time steps are used, and furthermore it can easily be improved upon, as follows.

In *trapezoidal integration* an estimate of the function derivative at the end of the time step is obtained from the Euler formula; then the average of the derivatives at the beginning and end of the time step is used to make a more accurate Euler step. The equations for a step forward from time t to $t + T$ are

$$\begin{aligned} x_E(t + T) &= x(t) + Tf(x(t), t) \\ \dot{x}_E(t + T) &= f(x_E(t + T), t + T) \\ x_T(t + T) &= x(t) + \frac{T}{2} [\dot{x}(t) + \dot{x}_E(t + T)], \end{aligned} \quad (3.4-5)$$

where subscripts E and T indicate, respectively, Euler and trapezoidal steps. For reasons that will soon become clear, these equations are commonly written as

$$\begin{aligned} k_1 &= Tf(x, t) \\ k_2 &= Tf(x + k_1, t + T) \\ x_T(t + T) &= x(t) + \frac{1}{2}(k_1 + k_2) \end{aligned} \quad (3.4-6)$$

This algorithm can be shown to agree with the first three Taylor series terms, that is, up to and including the second derivative term. Therefore, this trapezoidal integration formula is said to be of order 2, and it gives an improvement in accuracy over the Euler first-order method. RK algorithms are an extension of (3.4-6) to higher orders, and the general form is

$$\begin{aligned} k_1 &= Tf(x, t) \\ k_2 &= Tf(x + \beta_1 k_1, t + \alpha_1 T) \\ k_3 &= Tf(x + \beta_2 k_1 + \beta_3 k_2, t + \alpha_2 T) \\ k_4 &= Tf(x + \beta_4 k_1 + \beta_5 k_2 + \beta_6 k_3, t + \alpha_3 T) \\ &\vdots \\ x_{RK}(t + T) &= x(t) + \gamma_1 k_1 + \gamma_2 k_2 + \gamma_3 k_3 + \dots \end{aligned} \quad (3.4-7)$$

Implicit RK algorithms also exist, wherein a coefficient k_i occurs on both sides of one of the equations above. The constants α_i , β_i , and γ_i are chosen so that a particular RK scheme agrees with the Taylor series to as high an order as possible. A great deal of algebraic effort is needed to derive higher-order (greater than 4) RK algorithms, and the constants are not unique for a given order. An algorithm that dates from the end of the nineteenth century and is still popular is Runge's fourth-order rule, which uses the constants

$$\begin{aligned}\alpha_1 &= \alpha_2 = \beta_1 = \beta_3 = \frac{1}{2} \\ \alpha_3 &= \beta_6 = 1 \\ \beta_2 &= \beta_4 = \beta_5 = 0 \\ \gamma_1 &= \gamma_4 = 1/6, \quad \gamma_2 = \gamma_3 = 1/3\end{aligned}\tag{3.4-8}$$

In this case only one previous k value appears in each of the k equations in (3.4-7), thus making a simpler algorithm. This algorithm has been used for most of our examples, and computer code for the general case of n simultaneous nonlinear state equations is given in the example below.

An important feature of the RK methods is that the only value of the state vector that is needed is the value at the beginning of the time step; this makes them well suited to the ODE initial-value problem. The amount of computation involved is governed by the number of derivative evaluations using the state equations performed during each time step. The number of derivative evaluations depends on the order chosen. For example, a fourth-order RK algorithm cannot be achieved with fewer than four derivative evaluations. For a given overall accuracy in a time-response calculation, there is a trade-off between many small steps with a low-order method and fewer steps but more derivative evaluations with a higher-order method. This led mathematicians to consider the problem of estimating the error in the computed solution function at each time step. Such an error estimate can be used to control the step size automatically in order to meet a specified accuracy. Algorithms that combine RK integration with error estimation include Runge-Kutta-Merson (RKM), Runge-Kutta-England, and Runge-Kutta-Gill; computer codes are commonly available. In terms of (3.4-7) the coefficients for the RKM scheme, for example, are

$$\begin{aligned}\alpha_1 &= \beta_1 = 1/3 \\ \alpha_2 &= 1/3, \quad \beta_2 = \beta_3 = 1/6 \\ \alpha_3 &= 1/2, \quad \beta_4 = 1/8, \quad \beta_5 = 0, \quad \beta_6 = 3/8 \\ \alpha_4 &= 1, \quad \beta_7 = 1/2, \quad \beta_8 = 0, \quad \beta_9 = -3/2, \quad \beta_{10} = 2 \\ \gamma_1 &= 1/6, \quad \gamma_2 = \gamma_3 = 0, \quad \gamma_4 = 2/3, \quad \gamma_5 = 1/6\end{aligned}\tag{3.4-9}$$

and the estimated error is

$$E \approx \frac{1}{30}[2k_1 - 9k_3 + 8k_4 - k_5]$$

Linear Multistep Methods

In the LMMs the solution function is a linear combination of past values of the function and its derivatives, as described by the linear difference equation

$$x(n+1) = \sum_{r=0}^n \alpha_r x(n-r) + T \sum_{r=-1}^n \beta_r \dot{x}(n-r), \quad (3.4-10)$$

where $x(i)$ indicates the value of x at time iT , with i an integer. If β_{-1} is nonzero, the algorithm is an implicit algorithm because the solution $x(n+1)$ is needed to evaluate $\dot{x}(n+1)$ on the right-hand side. Otherwise the algorithm is explicit. The implicit equation must be solved at each time step. LMMs can be designed to require less computation than RK methods because a number of past values can be kept in storage as the computation proceeds. Because of the requirements for past values, the LMMs are not self-starting, and an RK method, for example, could be used to generate the starting values.

The LMM algorithms can be created in a number of different ways. For instance, if the scalar state equation (3.4-2) is written as an integral equation over the time interval nT to $(n+k)T$, the result is

$$x(n+k) = x(n) + \int_{nT}^{(n+k)T} f(x, t) dt \quad (3.4-11)$$

There are many finite-difference formulae for evaluating a definite integral, and this approach leads to the *Newton-Coates integration formulae* (Isaacson and Keller, 1966; Ralston, 1965). Two examples are

$$x(n+1) = x(n-1) + 2T\dot{x}(n) \quad (3.4-12a)$$

$$x(n+1) = x(n-1) + \frac{T}{3}[\dot{x}(n+1) + 4\dot{x}(n) + \dot{x}(n-1)] \quad (3.4-12b)$$

The first formula uses the *midpoint rule* for the area represented by the integral and is explicit, while the second uses *Simpson's rule* and is implicit. Implicit and explicit formulae can be used together in a *predictor-corrector algorithm* (Hamming, 1962). The explicit formula is the predictor, used to obtain an approximate value of the solution, and the implicit formula is the corrector equation, which is solved (by iteration) to obtain a more accurate solution.

LMMs of any order can be derived directly from (3.4-10). When $\alpha_r \equiv 0$ for $r > 0$, the *Adams-Bashforth-Moulton (ABM) formulae* are obtained.

We now give two examples. Assume that Equation (3.4-10) has the terms

$$x(n+1) = \alpha_0 x(n) + T[\beta_0 \dot{x}(n) + \beta_1 \dot{x}(n-1)] \quad (3.4-13)$$

Now write Taylor series expansions for the terms that are not taken at time nT :

$$\begin{aligned} x(n+1) &= x(n) + T\dot{x}(n) + \frac{T^2}{2!}\ddot{x}(n) + \dots \\ \dot{x}(n-1) &= \dot{x}(n) - T\ddot{x}(n) + \frac{T^2}{2!}\ddot{\ddot{x}}(n) \dots \end{aligned}$$

Substitute these expressions in (3.4-13) and equate powers of T on both sides of the resulting equation; this gives

$$\begin{aligned} T^0 : \quad 1 &= \alpha_0 \\ T^1 : \quad 1 &= \beta_0 + \beta_1 \\ T^2 : \quad \frac{1}{2} &= -\beta_1 \end{aligned}$$

Therefore, (3.4-13) yields the second-order ABM formula

$$x(n+1) = x(n) + \frac{T}{2}[3\dot{x}(n) - \dot{x}(n-1)] \quad (3.4-14)$$

This requires only one state equation evaluation per time step, and it has often been used for simulation. The higher-order methods also require only one derivative evaluation per time step, and the third-order ABM is

$$x(n+1) = x(n) + \frac{T}{12}[23\dot{x}(n) - 16\dot{x}(n-1) + 5\dot{x}(n-2)] \quad (3.4-15)$$

The implicit formulae may be derived in the same way; they give improved accuracy and can also provide an error estimate. They are commonly used in the predictor-corrector form, and this requires two derivative evaluations per step.

Stability, Accuracy, and Stiff Systems

In developing numerical algorithms it is always necessary to consider how computational errors are magnified. If, in pursuit of greater accuracy, one blindly attempts to create higher-order LMM formulae, it is quite possible that the algorithm will be unstable and errors will grow with time. Stability can be determined by analyzing a finite-difference equation associated with the integration algorithm. This analysis (Shampine and Gordon, 1975) is beyond the scope of this chapter and we simply note that the specific algorithms described above are stable.

The RK stability properties are different from those of the LMMs. In the case of the RK algorithms, a reduction in time-step size will eventually eliminate an instability, although the required step size may be unreasonably small. Example 3.6-5 is an example of a reduction in step size eliminating an instability. When a set of state equations is being integrated, the required step size will be determined by the smallest time constant (i.e., the fastest component) of the solution function. A system with a very wide spread of time constants is known as a *stiff system*, and a very large number of RK steps may be necessary to yield only a small part of the complete solution. Other techniques are required for stiff systems (see below).

Choice of Integration Algorithm

The most important feature of the RK methods is that they directly solve the initial-value problem. That is, no past values are needed to start the integration.

This, of course, exactly matches the philosophy of the state-space formulation in which all of the information describing the “state” of the system is contained in the state vector at any given time instant. The full significance of these facts can only be appreciated when a simulation containing discrete events is considered. This is a common practical engineering situation. For instance, at a given time a new subsystem may be activated or at a certain value of some variable the equations of motion may change because limiting or saturation behavior occurs. This means that previous states are less relevant; the information they carry may now apply to only a part of the complete system. This fact favors the RK methods over the multistep methods, and we will return to these points later. The disadvantages of the RK methods are that the error expressions are complex, they are inefficient when dealing with stiff systems, and more derivative evaluations are required for a given order than is the case with LMMs. The tremendous increases in computing power in recent years have made these disadvantages much less significant for small- to medium-sized simulations. Such simulations are commonly run with a fixed time step that has been found (by trial and error) to be adequate for the required accuracy and is also determined by other discrete-event considerations.

The important features of LMMs are that higher-order methods are obtained for a given number of derivative evaluations, and an accurate expression for the integration error can usually be obtained. These methods come into their own on very large systems of equations and large stiff systems and when there is no hard-limiting behavior or topological changes due to switching. The software package ODE-PACK (Hindmarsh, 1982) is available for large and stiff problems, and it handles equations in standard explicit form or in linearly implicit form. For nonstiff problems it uses the implicit ABM methods, and for stiff problems it uses a backward-difference formula and improves on the Gear algorithms (Gear, 1971) that have long been used for stiff systems. These algorithms have been used on atmosphere models with more than 10,000 simultaneous ODEs; the spread of time constants in the problem ranged from milliseconds to years, thus making the equations extremely stiff.

Time-History Simulation

Here we will show how the integration techniques can be used to determine a *state trajectory*, that is, the motion of the tip of the state vector as a function of time in the n -dimensional space. This is usually called *time-history simulation*. Our state-space dynamic equations are already in the best form for simulation, either non-real-time simulation or real-time simulation (e.g., in a flight simulator); it is only necessary to couple them with the integration algorithm.

In general, a simulation will also need to process discrete-time calculations, that is, calculations in which the signals are only defined at the “sample instants.” Such signals may arise from simulating a digital computer or sampling external signals. The numerical integration algorithms are based on the assumption that external inputs to the state equations will remain constant during an integration step. Therefore, the integration routines effectively impose a “zero-order data-hold” (ZOH) on the sampled signals. The ZOH is described in more detail in Chapter 7.

Figure 3.4-1 shows how a non-real-time simulation program may be organized. Two separate functions or subroutines are needed for the dynamic models. One

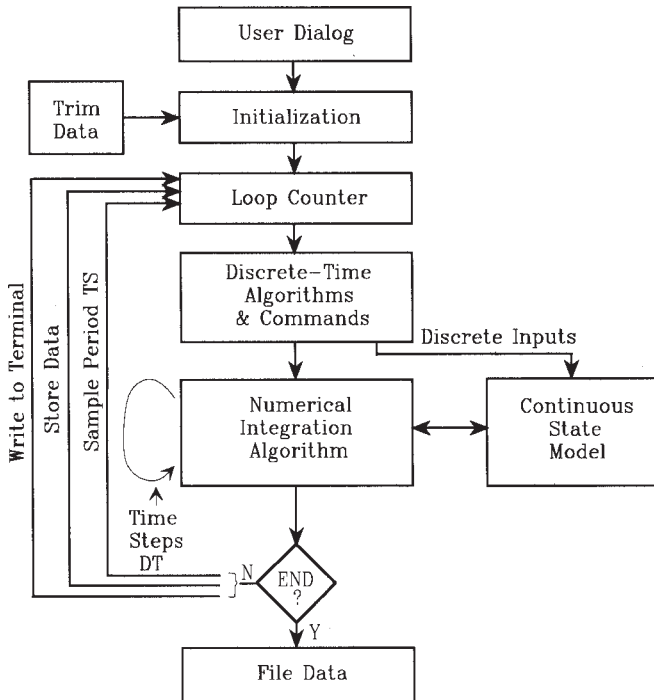


Figure 3.4-1 Time-history simulation.

function contains the continuous-time state equations, and another function contains preprogrammed discrete-time commands and any discrete-time algorithms used for digital control. Simulation time is controlled by a for-loop, and the basic increment of time is the integration time step DT . The *sample period* for the discrete dynamics TS can conveniently be chosen to be an integer multiple of DT . Alternatively, TS may be the basic time increment, and the integration algorithm may integrate over TS while adaptively adjusting its step size to attain a specified integration accuracy. Periodic sampling is not essential, and the adaptive integration may continue until a discrete input occurs. The following example shows that a simple time-history simulation is very easy to perform.

Example 3.4-1: Integration of the Van der Pol Equation A simple time-history program “NLSIM” written in MATLAB code is shown below; it prompts the user for the name of the m-file containing the state equations and the name of an initial-condition file. The convention used is that the state equation function will always have the arguments “time,” X , and U , in that order. The initial condition file will be a text file with a “.dat” extension, which can also be read by other programming languages; it should have a different name from the state-equations file. The “.dat” extension must be entered at the MATLAB prompt because otherwise MATLAB assumes an “.m” extension.

```
% NLSIM.M Nonlinear Simulation
clear all
```

```

% global % add variables as needed
name= input('Enter Name of State Equations m-file : ','s');
icfile= input('Enter Name of i.c. File : ','s');
tmp= dlmread(icfile,',' );
n=tmp(1); m=tmp(2);
x=tmp(3:n+2); u=tmp(n+3:n+m+2);
stat=fclose('all');
runtime= input('Enter Run-Time : ');
dt = input('Enter Integration Time-step : ');
N=runtime/dt; k=0; NP= fix ( max(1,N/500) ); time=0.;
xd= feval(name,time,x,u); % Set variables in state equations
save=u(2); % For Example 3.6-3 only
for i=0:N
    time=i*dt;
    if rem(i,NP)==0
        k=k+1;
        y(k,1)= x(1); % record data as needed
        y(k,2)= x(2);
        %y(k,3)=
    end
    %if time>=2 % For Example 3.6-3
    % u(2)=save;
    %elseif time>=1.5
    % u(2)=save-2;
    %elseif time>=1.0
    % u(2)=save+2;
    %else
    % u(2)=save;
    %end
    [x]= RK4(name,time,dt,x,u);
end
t= NP*dt*[0:k-1];
figure(1)
plot(y(:,1), y(:,2)) % For Van der Pol
grid on
axis([-3,3,-4,5])
xlabel('X(1)')
ylabel('X(2)')
text(-1.8,3.2,'(-2,3)')

```

The fourth-order RK algorithm, with the constants given in (3.4-8) is

```

function [xnew]= RK4(f,time,dt,xx,u)
xd=feval(f,time,xx,u);
xa=xd*dt;
x =xx + 0.5*xa;
t =time + 0.5*dt;
xd=feval(f,t,x,u);
q = xd*dt;
x = xx + 0.5*q;
xa= xa + 2.0*q;
xd= feval(f,t,x,u);
q = xd*dt;
x = xx + q;
xa= xa + 2.0*q;

```

```

time= time + dt;
xd= feval(f,time,x,u);
xnew= xx + (xa + xd*dt)/6.0;

```

The state equations used as an example are those of the Van der Pol oscillator, which exhibits interesting nonlinear behavior,

```

% File VDPOL.m
function [xd]= vdpol(time,x,u)
xd= [x(2)  -u(1)*(x(1)^2-1)*x(2)-x(1)];

```

The control input $u = 0.8$ was used as the parameter that controls the dynamic behavior of the Van der Pol oscillator. For this example the initial condition file VDP.dat contained the number of states and controls, the initial state, and the control input as follows:

2, 1, .1, .1, .8

Figure 3.4-2 shows state x_2 plotted against state x_1 and is called a *phase portrait*. Two different sets of initial conditions are shown in the figure, and in both cases the state trajectories approach the same closed contour. The resulting constant-amplitude oscillation is called a *limit cycle*. This example is studied further in Problem 3.4-1, and NLSIM.m is used for aircraft simulation in Section 3.6.

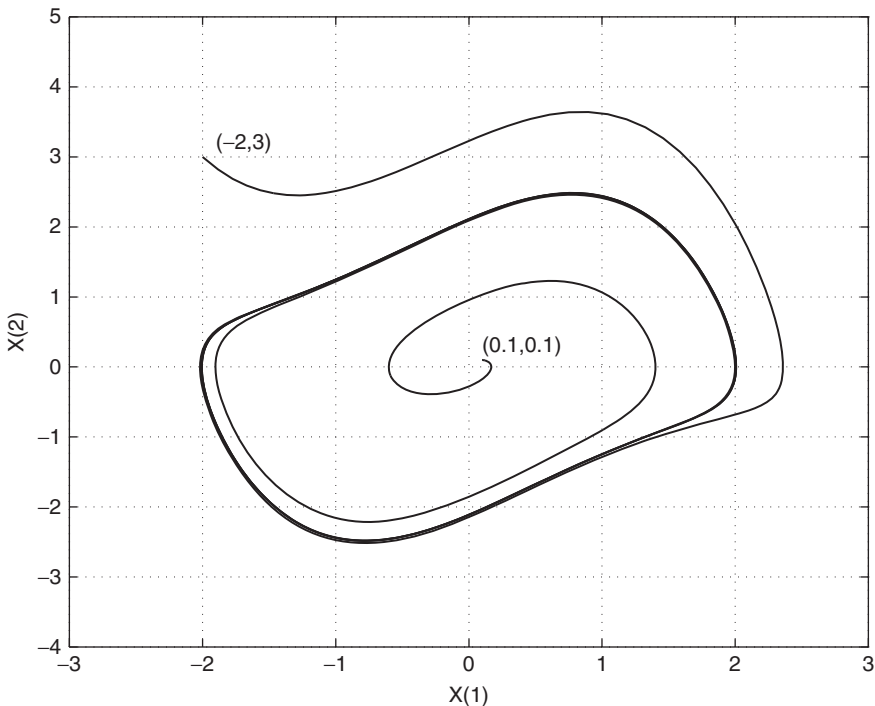


Figure 3.4-2 A Van der Pol limit cycle.

3.5 AIRCRAFT MODELS FOR SIMULATION

Simulation Issues

In Section 3.4 we used MATLAB to illustrate simple nonlinear simulation, but this *interpreted* code executes one to two orders of magnitude more slowly than *compiled* code. In time-history simulation we wish to use a fixed sample period of 5 to 50 ms for the purposes of adequately sampling external inputs, generating random inputs, and interfacing with discrete-time controllers. The integration step size must be less than or equal to the discrete-time sample period and depends on the accuracy required and the stiffness of the dynamics. With this constraint, any improvement in speed of execution must come from linking MATLAB with compiled code. Rather than use this approach we have chosen to present a simple aircraft model in MATLAB code and a more complicated model in Fortran code. The choice of Fortran produces readable code. The reader has the option of converting the Fortran code to MATLAB (which is relatively easy to do), using MATLAB with compiled code (free compilers can be found), or running the Fortran code. A Fortran simulation program, TRESP, was written in the same form as the MATLAB program in Example 3.4-1 and using the RK4 integrator but using subroutines for the continuous dynamics and the discrete dynamics and with more comprehensive interactive capabilities. This program was used for the F-16 flight simulation examples in Section 3.6 and Chapter 4.

A Simple Longitudinal Model

This model has only three degrees of freedom (i.e., translation and pitching motion in the vertical plane): it has fixed aerodynamic coefficients and is representative of a medium-size transport aircraft at a low-speed flight condition. Data are also provided for the effects of extending landing gear and flaps. The aircraft weighs 162,000 lb (one-half fuel, partial cargo), and it has two turboprop engines, each developing 30,000 lb of static thrust at sea level. The wing area is 2170 ft², wing span 140 ft, length 90 ft, and pitch-axis inertia 4.1×10^6 slug-ft². The model is illustrated in Figure 3.5-1, programmed as a MATLAB function.

This MATLAB function calculates the state derivative vector \dot{x} from the state vector x and the control vector u ; the formal argument “time” is unused. It is compatible with the time-history program given in Example 3.4-1. The control inputs $u(3)$ and $u(4)$ are used, respectively, to set the x -axis position of the cg and the landing configuration switch. For the aircraft models we will use the customary term *center of gravity* (cg) synonymously with cm, although technically a cg does not exist if a body does not have a spherically symmetrical distribution of mass.

Miscellaneous model outputs can be made available by setting up global variables. The aerodynamic derivatives are in stability axes and have “per degree” units except for the pitch damping coefficients (C_{m_q} , $C_{m_{\dot{\alpha}}}$), which are per radian per second. There is provision for a $C_{L_{\dot{\alpha}}}$ derivative, but it is zero in this case. Lift is calculated from a linear lift curve and the stall is not modeled, while drag is calculated from the nonlinear drag polar. The elevator deflection is in degrees, and the throttle input is in the zero to

```

function [xd]= transp(time,x,u)
% Medium-sized transport aircraft, longitudinal dynamics.
%
S=2170.0; CBAR=17.5; MASS=5.0E3; IYY= 4.1E6;
TSTAT=6.0E4; DTDV =-38.0; ZE = 2.0; CDCLS= .042;
CLA = .085; CMA =-.022; CMDE =-.016; % per degree
CMQ =-16.0; CMADOT= -6.0; CLADOT= 0.0; % per radian
RTOD = 57.29578; GD=32.17;
THTL =u(1);
ELEV =u(2);
XCG = u(3);
LAND = u(4);
VT = x(1); % TAS in fps
ALPHA= RTOD*x(2); % A.O.A.
THETA= x(3); % PITCH ATTITUDE
Q = x(4); % PITCH RATE
H = x(5); % ALTITUDE
%
[MACH,QBAR]= ADC(VT,H);
QS = QBAR*S;
SALP= sin (x(2)); CALP= cos(x(2));
GAM = THETA - x(2); SGAM= sin (GAM); CGAM= cos(GAM);
if LAND == 0 % CLEAN
CLO= .20; CDO= .016;
CMO= .05; DCDG= 0.0; DCMG= 0.0;
elseif LAND == 1 % LANDING FLAPS & GEAR
CLO= 1.0; CDO= .08;
CMO= -.20; DCDG= .02; DCMG= -.05;
else
disp('Landing Gear & Flaps ?')
end
THR= (TSTAT+VT*DTDV) * max(THTL,0); % THRUST
CL=CLO+CLA*ALPHA; % NONDIM. LIFT
CM=DCMG+CMO+CMA*ALPHA+CMDE*ELEV+CL* (XCG-.25); % MOMENT
CD=DCDG+CDO+CDCLS*CL*CL; % DRAG POLAR
%
% STATE EQUATIONS NEXT
xd(1) = (THR*CALP-QS*CD)/MASS - GD*SGAM;
xd(2)=(-THR*SALP-QS*CL+MASS*(VT*Q+GD*CGAM))/(MASS*VT+QS*CLADOT);
Xd(3) = Q;
D = .5*CBAR*(CMQ*Q+CMADOT*xd(2))/VT; % PITCH DAMPING
Xd(4) = (QS*CBAR*(CM + D) + THR*ZE)/IYY; % Q-DOT
Xd(5) = VT*SGAM; % VERTICAL SPEED
Xd(6) = VT*CGAM; % HORIZNTL. SPEED

```

Figure 3.5-1 Transport aircraft model.

unity range. Atmospheric density (and hence dynamic pressure) is calculated in the function ADC (air data computer, see Appendix A) from the temperature variation of the standard atmosphere (Yuan, 1967). The engine thrust is modeled as decreasing linearly with airspeed, to approximate the characteristics of a propeller-driven aircraft. The thrust vector does not pass through the cg (the perpendicular distance

from this vector to the cg is Z_E), and therefore throttle changes will tend to cause pitching motion of the aircraft. Other parts of the model are either self-evident or can be understood by referring to the descriptions of aerodynamic effects in Chapter 2. This model will be used later for illustrative examples.

A Six-Degree-of-Freedom Nonlinear Aircraft Model

The mathematical model given here uses the wind tunnel data from NASA-Langley wind tunnel tests on a subscale model of an F-16 airplane (Nguyen et al., 1979). The data apply to the speed range up to about $M = 0.6$ and were used in a NASA-piloted simulation to study the maneuvering and stall/poststall characteristics of a relaxed static stability airplane. Because of the application and the ease of automated data collection, the data cover a very wide range of angle of attack (-20° to 90°) and sideslip angle (-30° to 30°). However, the present state of the art does not allow accurate dynamic modeling in the poststall region, and in addition the aircraft has insufficient pitching moment control for maneuvering at angles of attack beyond about 25° . Therefore, for use here, we have reduced the range of the data to $-10^\circ \leq \alpha \leq 45^\circ$ and approximated the beta dependence in some cases.

The F-16 has a leading-edge flap that is automatically controlled as a function of alpha and Mach and responds rapidly as alpha changes during maneuvering. In the speed range for which the data are valid, the Mach-dependent variation of the flap is small, and so we have eliminated this dependence. Then, neglecting the dynamics of the flap actuator and assuming that the flap is dependent on alpha only, we have merged all of the independent flap data tables into the rest of the tabular aerodynamic data. The effect of the flap deflection limits (but not the rate limits) is still present in the reduced data. These steps have greatly reduced the size of the database and made it feasible to present the data here (Appendix A). The approximate model constructed from these data exhibits steady-state flight trim conditions, and corresponding dynamic modes, that are close to those of the full (50-lookup-table) model.

The F-16 model has been programmed as a Fortran subroutine in a form similar to the MATLAB model. The code is shown in Figure 3.5-2; all subroutines and functions called by the model are included in Appendix A. Note that English units have been used here, rather than SI units. State variables V_T , α , and β have been used instead of the velocity components U , V , and W for ease of comparison with the linear small-perturbation equations. For serious simulation purposes it would be preferable to change to states U , V , and W . The quantities XCGR and HX are, respectively, x -coordinates of the reference cg position and engine angular momentum (assumed constant at 160 slug-ft²/s).

The aerodynamic force and moment component buildup follows the outline presented in Section 2.3 except that body axes are used. For example, CX(alpha, EL) is a function subprogram that computes the nondimensional force coefficient for the body x -axis and is a function of angle of attack and elevator deflection. The total force coefficients for the three axes are CXT, CYT, and CZT. As shown in the appendix, the component functions typically contain a two-dimensional data lookup table


```

SUBROUTINE F (TIME,X,XD)
REAL X(*), XD(*), D(9), MASS
COMMON/PARAM/XCG
COMMON/CONTROLS/THTL,EL,AIL,RDR
COMMON/OUTPUT/AN,ALAT,AX,QBAR,AMACH,Q,ALPHA
PARAMETER (AXX=9496.0, AYY= 55814.0, AZZ=63100.0, AXZ= 982.0)
PARAMETER (AXZS=AXZ**2, XPQ=AXZ*(AXX-AYY+AZZ),
& GAM=AXX*AZZ-AXZ**2)
PARAMETER (XQR= AZZ*(AZZ-AYY)+AXZS, ZPQ=(AXX-AYY)*AXX+AXZS)
PARAMETER ( YPR= AZZ - AXX )
PARAMETER (WEIGHT= 25000.0, GD= 32.17, MASS= weight/gd)

DATA S,B,CBAR,XCGR,HX/300,30,11.32,0.35,160.0/
DATA RTOD / 57.29578/

C
C Assign state & control variables
C
VT= X(1); ALPHA= X(2)*RTOD; BETA= X(3)*RTOD
PHI=X(4); THETA= X(5); PSI= X(6)
P= X(7); Q= X(8); R= X(9); ALT= X(12); POW= X(13)

C
C Air data computer and engine model
C
CALL ADC(VT,ALT,AMACH,QBAR); CPOW= TGEAR(THTL)
XD(13) = PDOT(POW,CPOW); T= THRUST(POW,ALT,AMACH)

C
C Look-up tables and component buildup
C
CXT = CX (ALPHA,EL)
CYT = CY (BETA,AIL,RDR)
CZT = CZ (ALPHA,BETA,EL)
DAIL= AIL/20.0; DRDR= RDR/30.0
CLT = CL(ALPHA,BETA) + DLDA(ALPHA,BETA)*DAIL
& + DLDR(ALPHA,BETA)*DRDR
CMT = CM(ALPHA,EL)
CNT = CN(ALPHA,BETA) + DNDA(ALPHA,BETA)*DAIL
& + DNDR(ALPHA,BETA)*DRDR

C
C Add damping derivatives :
C
TVT= 0.5/VT; B2V= B*TVT; CQ= CBAR*Q*TVT
CALL DAMP(ALPHA,D)
CXT= CXT + CQ * D(1)
CYT= CYT + B2V * ( D(2)*R + D(3)*P )
CZT= CZT + CQ * D(4)
CLT= CLT + B2V * ( D(5)*R + D(6)*P )
CMT= CMT + CQ * D(7) + CZT * (XCGR-XCG)
CNT= CNT + B2V*(D(8)*R + D(9)*P) - CYT*(XCGR-XCG) * CBAR/B

C
C Get ready for state equations
C
CBTA = COS(X(3)); U=VT*COS(X(2))*CBTA
V= VT * SIN(X(3)); W=VT*SIN(X(2))*CBTA

```

Figure 3.5-2 Model of the F-16 aircraft.

```

      STH  = SIN(THETA); CTH= COS(THETA);   SPH= SIN(PHI)
      CPH  = COS(PHI) ; SPSI= SIN(PSI);     CPSI= COS(PSI)
      QS   = QBAR * S ; QSB= QS * B;        RMQS= QS/MASS
      GCTH = GD * CTH ; QSPH= Q * SPH
      AY   = RMQS*CYT ; AZ= RMQS * CZT

C
C Force equations
C
      UDOT  = R*V - Q*W - GD*STH + (QS * CXT + T)/MASS
      VDOT  = P*W - R*U + GCTH * SPH + AY
      WDOT  = Q*U - P*V + GCTH * CPH + AZ
      DUM   = (U*U + W*W)
      xd(1) = (U*UDOT + V*VDOT + W*WDOT)/VT
      xd(2) = (U*WDOT - W*UDOT) / DUM
      xd(3) = (VT*VDOT- V*XD(1)) * CBTA / DUM

C
C Kinematics
C
      xd(4) = P + (STH/CTH)*(QSPH + R*CPH)
      xd(5) = Q*CPH - R*SPH
      xd(6) = (QSPH + R*CPH)/CTH

C
C Moments
C
      ROLL  = QSB*CLT
      PITCH = QS *CBAR*CMT
      YAW   = QSB*CNT
      PQ    = p*Q
      QR    = Q*R
      QHX   = Q*HX
      xd(7) = ( XPQ*PQ - XQR*QR + AZZ*ROLL + AXZ*(YAW + QHX) )/GAM
      xd(8) = ( YPR*P*R - AXZ*(P**2 - R**2) + PITCH - R*HX )/AYY
      xd(9) = ( ZPQ*PQ - XPQ*QR + AXZ*ROLL + AXX*(YAW + QHX) )/GAM

C
C Navigation
C
      T1= SPH * CPSI; T2= CPH * STH; T3= SPH * SPSI
      S1= CTH * CPSI; S2= CTH * SPSI; S3= T1 * STH - CPH * SPSI
      S4= T3 * STH + CPH * CPSI; S5= SPH * CTH; S6= T2*CPSI + T3
      S7= T2 * SPSI - T1; S8= CPH * CTH

C
      xd(10) = U * S1 + V * S3 + W * S6      ! North speed
      xd(11) = U * S2 + V * S4 + W * S7      ! East speed
      xd(12) = U * STH -V * S5 - W * S8      ! Vertical speed

C
C Outputs
C
      AN -AZ/GD; ALAT= AY/GD
      RETURN
      END

```

Figure 3.5-2 (Continued)

and a linear interpolation routine. We have used as much commonality as possible in the data tables and interpolation routines and have provided an interpolator that will also extrapolate beyond the limits of the tables. Therefore, a simulation may recover without loss of all data despite temporarily exceeding the limits of a lookup table.

Engine Model The NASA data include a model of the F-16 afterburning turbofan engine in which the thrust response is modeled with a first-order lag and the lag time constant is a function of the actual engine power level (POW) and the commanded power (CPOW). This time constant is calculated in the function PDOT, whose value is the rate of change of power, while the state variable X_{13} represents the actual power level. The function TGEAR (throttle gearing) relates the commanded power level to the throttle position (0 to 1.0) and is a linear relationship apart from a change of slope when the military power level is reached at 0.77 throttle setting. The variation of engine thrust with power level, altitude, and Mach number is contained in the function THRUST.

Sign Convention for Control Surfaces The sign conventions used in the model follow a common industry convention and are given in Table 3.5-1.

Testing the Model When constructing this model, a simple program should be written to exercise each of the aerodynamic lookup tables individually and plot the data before the tables are used with the model. The range of the independent variables should be chosen to ensure that both extrapolation and interpolation are performed correctly. A simple check on the complete model can be obtained by writing another program to set the parameter, input, and state vectors to the arbitrarily chosen values given in Table 3.5-2. The resulting values of the derivative vector should then agree with those given in the table.

Next we must bring this model under control by finding a combination of values of the state and control variables that correspond to a steady-state flight condition. Unlike a real pilot who is constantly receiving visual and other cues, this is quite difficult for us and will be the subject of the next section. In the next section steady-state trim data will be given for both wings-level, non-sideslipping flight and turning flight. Therefore, the longitudinal equations can be tested alone before all the equations are brought into play.

TABLE 3.5-1 Aircraft Control Surface Sign Conventions

	Deflection	Sense	Primary Effect
Elevator	trailing edge down	positive	negative pitching moment
Rudder	trailing edge left	positive	negative yawing moment
Ailerons	right-wing trailing edge down	positive	negative rolling moment

TABLE 3.5-2 F-16 Model Test Case

Index (i)	PARAM	$U(i)$	$X(i)$	$\dot{X}(i)$
1	.4	0.9	500	-75.23724
2		20	0.5	-0.8813491
3		-15	-0.2	-0.4759990
4		-20	-1	2.505734
5			1	0.3250820
6			-1	2.145926
7			0.7	12.62679
8			-0.8	0.9649671
9			0.9	0.5809759
10			1000	342.4439
11			900	-266.7707
12			10000	248.1241
13			90	-58.68999

3.6 STEADY-STATE FLIGHT

Steady-state flight is important because it provides an initial condition for flight simulation and a flight condition in which we can linearize the aircraft dynamics (see Section Numerical Linearization). Figure 3.1-1 shows how a steady-state “trim” program fits into the state-space context. A generic trim program links to any nonlinear model and produces a file containing the steady-state values of the control and state vectors for use by the time-history and linearization programs. Steady-state flight was defined in Section 2.6 and was shown to require the solution of a set of nonlinear simultaneous equations derived from the state model. Now we are faced with the problem of actually calculating the values of the state and control vectors that satisfy these equations. This cannot be done analytically because of the very complex functional dependence of the aerodynamic data. Instead, it must be done with a numerical algorithm that iteratively adjusts the independent variables until some solution criterion is met. The solution will be approximate but can be made arbitrarily close to the exact solution by tightening up the criterion. Also, the solution may not be unique—for example, steady-state level flight at a given engine power level can in general correspond to two different airspeeds and angles of attack. Our knowledge of aircraft behavior will allow us to specify the required steady-state condition so that the trim algorithm converges on an appropriate, if not unique, solution.

One of the first things that must be decided is how to specify the steady-state condition, how many of the state and control variables can be chosen independently, and what constraints exist on the remaining variables. A computer program can then be written so that the specification variables are entered from the keyboard, and the independent variables are adjusted by the numerical algorithm that solves the nonlinear equations, while the remaining variables are determined from the constraint equations.

For steady-state flight we expect to be able to specify the altitude and the velocity vector (i.e., speed and flight-path angle) within the limits imposed by engine power. Then, assuming that the aircraft configuration (i.e., flap settings, landing gear up or down, speed brake deployed, etc.) is prespecified, for a conventional aircraft we expect that a unique combination of the control inputs and the remaining state variables will exist. All of the control variables (throttle, elevator, aileron, and rudder) enter the model only through tabular aerodynamic data, and we cannot, in general, determine any analytical constraints on these control inputs. Therefore, these four control inputs must be adjusted by our numerical algorithm. This is not the case for the state variables.

Since only the NED altitude component of the tangent-plane position vector is relevant and can be prespecified, we can temporarily eliminate the three position states from consideration. Consider first steady translational flight. The state variables ϕ , P , Q , R are all identically zero, and the orientation ψ can be specified freely; this only leaves V_T , α , β (or U , V , W), and θ to be considered. The sideslip angle cannot be specified freely; it must be adjusted by our trim algorithm to zero out any sideforce. This leaves the variables V_T , α , and θ ; the first two are interrelated through the amount of lift needed to support the weight of the aircraft; therefore, only two may be specified independently (θ and either V_T or α). We usually wish to impose a flight-path angle (γ) constraint on the steady-state condition, so we will finally choose to specify V_T and γ .

Because the atmospheric density changes with altitude, a steady-state flight condition does not strictly include a nonzero flight-path angle. Nevertheless, it is useful to be able to determine a trimmed condition for a nonzero flight-path angle at any given altitude, since rate of climb (ROC) can then be determined and linearized dynamic models can be obtained for nonzero flight-path angles. We will therefore derive a general ROC constraint; this constraint will allow a nonzero roll angle so that it can also be applied to steady-state turning flight.

Steady-state turning flight must now be considered; the variables ϕ , P , Q , and R will no longer be set to zero. The turn can be specified by the Euler angle rate $\dot{\psi}$; this is the rate at which the aircraft's heading changes (the initial heading can still be freely specified). Then, given values of the attitude angles ϕ and θ , the state variables P , Q , and R can be determined from the kinematic equation (1.4-4). The required value of θ can be obtained from the ROC constraint if the value of ϕ is known, and we next consider the determination of ϕ .

The roll angle (ϕ) for the steady-state turn can be freely specified, but then, in general, there will be a significant sideslip angle and the turn will be a "skidding" turn. The pilot will feel a force pushing him or her against the side of the cockpit, the passengers' drinks will spill, and the radius of the turn will be unnecessarily large. In a "coordinated" turn the aircraft is rolled at an angle such that there is no component of aerodynamic force along the body y -axis. This condition is used as the basis of the turn coordination constraint derived below. The turn coordination constraint will be found to involve both θ and ϕ ; therefore, it must be solved simultaneously with the ROC constraint.

Chapters 1 and 2 have shown, via the flat-Earth equations, that the dynamic behavior of the aircraft is determined by the relative wind ($-\mathbf{v}_{rel}$) and height in the

atmosphere and hence by the variables V_T , α , β , and h . The behavior is essentially independent of the wind velocity $\mathbf{v}_{W/e}$. Therefore, when we wish to determine a steady-state flight condition for studying the dynamics, we will set the wind velocity to zero.

The Rate-of-Climb Constraint

With no wind, the velocity relative to the atmosphere is just the velocity over Earth, and

$$\mathbf{v}_{cm/e}^{tp} = C_{tp/bf} C_{bf/w} \mathbf{v}_{rel}^w$$

In the flat-Earth equations the rate of climb is simply $V_T \sin \gamma$, and this is the $-z$ -component of the velocity in tangent-plane coordinates. Therefore, the above equation yields

$$V_T \begin{bmatrix} * \\ * \\ -\sin \gamma \end{bmatrix} = C_{tp/bf} C_{bf/w} \begin{bmatrix} V_T \\ 0 \\ 0 \end{bmatrix} \quad (3.6-1)$$

The asterisks indicate “don’t care” components, and if this equation is expanded and then arranged to solve for θ (Problem 3.6-3), the results are

$$\sin \gamma = a \sin \theta - b \cos \theta, \quad (3.6-2)$$

where

$$a = \cos \alpha \cos \beta, \quad b = \sin \phi \sin \beta + \cos \phi \sin \alpha \cos \beta$$

Now, solving for θ , we find

$$\tan \theta = \frac{ab + \sin \gamma \sqrt{a^2 - \sin^2 \gamma + b^2}}{a^2 - \sin^2 \gamma}, \quad \theta \neq \pm \pi/2 \quad (3.6-3)$$

As a check, in wings-level, non-sideslipping flight this equation reduces to $\theta = \alpha + \gamma$.

The Turn Coordination Constraint

In a perfectly coordinated turn the components of force along the aircraft body-fixed y -axis sum to zero, and in addition we have the steady-state condition $\dot{V} = 0$. Then, from Table 2.5-1,

$$0 = -RU + PW + g_D \sin \phi \cos \theta$$

Now use Equation (1.4-4), with $\dot{\phi} = \dot{\theta} = 0$, to introduce the *turn rate* $\dot{\psi}$ in place of P and R . Also use Equation (2.3-6a) to introduce V_T in place of U and W ; the result is

$$-V_T \dot{\psi} \cos \beta [\cos \alpha \cos \theta \cos \phi + \sin \alpha \sin \theta] = g_D \sin \phi \cos \theta$$

If we define

$$\mathcal{G} \equiv \dot{\psi} V_T / g_D,$$

which is the centripetal acceleration (in g 's), then the constraint can be written as

$$\sin \phi = \mathcal{G} \cos \beta (\sin \alpha \tan \theta + \cos \alpha \cos \phi) \quad (3.6-4)$$

This is the required coordination constraint; it can be used in conjunction with (3.6-3) to trim the aircraft for turning flight with a specified rate of climb. If we can now solve (3.6-3) and (3.6-4) simultaneously for the state variables ϕ and θ , our numerical trim algorithm need only vary α and β (in addition to the four controls). The simultaneous solution is quite cumbersome but can be shown to be

$$\tan \phi = \mathcal{G} \frac{\cos \beta}{\cos \alpha} \frac{(a - b^2) + b \tan \alpha [c(1 - b^2) + \mathcal{G}^2 \sin^2 \beta]^{\frac{1}{2}}}{a^2 - b^2(1 + c \tan^2 \alpha)}, \quad (3.6-5)$$

where

$$a = 1 - \mathcal{G} \tan \alpha \sin \beta, \quad b = \sin \gamma / \cos \beta, \quad c = 1 + \mathcal{G}^2 \cos^2 \beta$$

The value of ϕ given by (3.6-5) can now be used to solve (3.6-3) for θ . Note that when the flight-path angle γ is zero, (3.6-5) reduces to

$$\tan \phi = \frac{\mathcal{G} \cos \beta}{\cos \alpha - \mathcal{G} \sin \alpha \sin \beta}, \quad (3.6-6)$$

and when β is small, this reduces to

$$\tan \phi = \frac{\mathcal{G}}{\cos \alpha} = \frac{\dot{\psi}}{g_D} \frac{V_T}{\cos \alpha} = \frac{\dot{\psi}}{g_D} \frac{V_T}{\cos \theta} \quad (3.6-7)$$

Equation (3.6-7) applies to a level, non-sideslipping turn and can be found from a simplified analysis given in standard texts. This completes the description of the flight-path constraints; we next show how a trim program may be constructed and provide examples of trimming the aircraft models.

The Steady-State Trim Algorithm

The steady-state flight conditions are determined by solving the nonlinear state equations for the state and control vectors that make the state derivatives \dot{U} , \dot{V} , \dot{W} (or \dot{V}_T , $\dot{\alpha}$, $\dot{\beta}$) and \dot{P} , \dot{Q} , \dot{R} identically zero. A convenient way to do this, with a readily available numerical algorithm, is to form a scalar *cost function* from the sum of the squares of the derivatives above. A function minimization algorithm can then be used to adjust the control variables and the appropriate state variables to minimize this scalar cost. Examples of suitable algorithms are the IMSL routine “ZXMW”

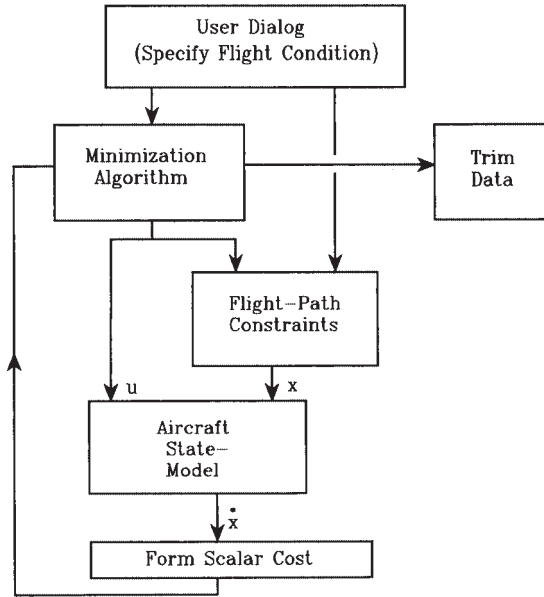


Figure 3.6-1 Steady-state trim flowchart.

(IMSL, 1980), and the SIMPLEX algorithm (Press et al., 1986; Nelder and Mead, 1964). Figure 3.6-1 illustrates how the complete trim algorithm may be organized. Only the cost function is tailored to a specific aircraft or set of state equations. We now give a simple trim example using the transport aircraft model.

Example 3.6-1: Steady-State Trim for a 3-DoF Aircraft Model In this example we will construct a simple 3-DoF trim program and use it on the transport aircraft model in Figure 3.5-1. It is only necessary to choose the speed and altitude, set the pitch rate state to zero, and adjust the throttle and elevator controls and the angle-of-attack state. Instead of the ROC constraint, we can specify the flight-path angle and constrain the pitch-attitude state to be equal to the angle of attack plus the flight-path angle. A simple MATLAB program is as follows:

```

% TRIM.m
clear all
global x u gamma
x(1)=input('Enter Vt : ');
x(5)= input('Enter h : ');
gamma=input('Enter Gamma (deg.) : ')/57.29578;
name= input('Name of Cost function file ? : ',' s');
cg= 0.25; land=1; % 0=clean 1=gear+flaps
u=[0.1 -10 cg land];
x(2)=.1; % Alpha, initial guess
x(3)=x(2) +gamma; % Theta

```



```
x(4)=0;    % Pitch rate
x(6)=0;
s0=[u(1) u(2) x(2)];
% Now initialize any other states and get initial cost
disp(['Initial cost = ',num2str( feval(name,s0) ) ])
[s,fval]=fminsearch(name,s0) ;
x(2)=s(3); x(3)=s(3)+gamma;
u(1)=s(1); u(2)=s(2) ;
disp(['minimum cost = ',num2str(fval)])
disp(['minimizing vector= ',num2str(s)])
temp=[length(x),length(u),x,u];
name= input('Name of output file ? : ',' s') ;
dlmwrite(name,temp);
```

and a cost function for the transport aircraft model is

```
% Cost Function for 3-DOF Aircraft
function [f]=cost(s);
global x u gamma
u(1)= s(1);
u(2)= s(2);
x(2)= s(3);
x(3)= x(2)+ gamma;
time= 0.0;
[xd]=transp(time,x,u);
f= xd(1)^2 + 100*xd(2)^2 + 10*xd(4)^2;
```

The MATLAB function “fminsearch” performs the minimization and is actually a Nelder and Mead Simplex algorithm. The results obtained for level flight with $c_g = 0.25\bar{c}$ and flaps and landing gear retracted are shown in Table 3.6-1. The cost function can be reduced to less than 1E-30, but anything below about 1E-12 causes negligible changes in the states and controls. The weighting on the derivatives in the cost function was experimental and makes little difference to the results.

The trim program for Example 3.6-1 can easily be modified for other experiments, such as trimming for a specific alpha by varying the airspeed (Problem 3.6-4). We next consider the slightly more difficult problem of trimming a 6-DoF model, with additional dynamics such as an engine model that must also be put into a steady-state condition. This will be illustrated with the F-16 model using the Fortran code in Appendix B.

TABLE 3.6-1 Trim Data for the Transport Aircraft Model

Altitude (ft)	speed (ft/s)	initial cost	final cost	throttle	elevator (deg)	alpha (deg)
0	170	28.9	< 1E-20	0.297	−25.7	22.1
0	500	3.54	< 1E-20	0.293	2.46	0.580
30k	500	10.8	< 1E-20	0.204	−4.10	5.43



Example 3.6-2: Steady-State Trim for a 6-DoF Model The following *cost function* subprogram has been specifically tailored to the F-16 model but is representative of the 6-DoF case in general:

```
function cost(s)
parameter (nn=20)
real s(*)
common/state/x(nn),xd(nn)
common/controls/tht1,el,ail,rdr
tht1 = s(1)
el = s(2)
x(2) = s(3)
ail = s(4)
rdr = s(5)
x(3) = s(6)
x(13)= tgear(tht1)
call constr(x)
call f(time,x,xd)
cost = xd(1)**2 + 100*(xd(2)**2 + xd(3)**2) + 10*(xd(7)**2
& + xd(8)**2 + xd(9)**2)
return
end
```

This cost function is specific to the F-16 model because of the assignment statement for X_{13} . An examination of the F-16 model will show that this statement sets the derivative \dot{X}_{13} to zero and hence puts the engine dynamics into the steady state. Any other dynamics in the aircraft model besides the rigid-body dynamics must be put into the steady-state condition in this way. In our original large F-16 model, this was done for the leading-edge flap actuator and its phase-lead network.

In this cost function, unlike the previous case, the state variables X_4 through X_9 (excluding X_6) are continually assigned new values in the constraint routine CONSTR. This routine implements the rate-of-climb and turn coordination constraints that were derived earlier. In the cost the aerodynamic angle rates $\dot{\alpha}$ and $\dot{\beta}$ have been weighted the most heavily, the angular rate derivatives \dot{P} , \dot{Q} , \dot{R} have medium weights, and the derivative \dot{V}_T has the least weight. Again, the weights are uncritical.

We will now use this cost function to determine the steady-state conditions in a coordinated turn performed by the F-16 model. The cg location of the model is at $0.35\bar{7}$, and the aircraft dynamics are unstable in pitch in the chosen flight condition. The turn would stress a pilot since it involves a sustained normal acceleration of 4.5 gs.

The trim program dialog and keyboard inputs are shown in Figure 3.6-2 as they would appear on a terminal display. Note that entering a “r” in response to a Fortran read statement causes the program to use the last values assigned to the variable. This allows the minimization to be picked up from where it was stopped if the final cost function was not low enough. In the run shown, the cost function was reduced by almost 10 orders of magnitude after 1000 function calls. Execution is very fast, and this is a reasonable number of calls.

The cost function can always be reduced to 1×10^{-10} or less; lower values are useful simply for checking consistency of results. The most effective way to use the simplex algorithm is to perform 500 to 1000 iterations and, if the cost is not

```

? Altitude (ft) : 0
? Air Speed (ft/s) and Climb Angle (deg) : 502,0
? Roll, Pull-Up, and Turn rates (rad/s) : 0,0,.3

Turn Radius (ft)= 1.6733E+03   Approx Roll Angle (deg)= 77.94
? Coordinated Turn (def. = Y) : /
? Required No. of Trim Iterations (def. =1000) : /

? Guess : Throttle, Elevator, Ailerons, Rudder : /
Computed : 8.35E-01 -1.48E+00 9.54E-02 -4.11E-01

    Angle of Attack 1.37E+01    Sideslip Angle 2.92E-02
    Pitch Angle 2.87E+00    Roll Angle 7.83E+01
Normal Acceleration 4.65E+00    Lateral Accn. -5.02E-06
Dynamic Pressure 3.00E+02    Mach Number 4.50E-01

Initial Cost Function 1.85E+01,    Final Cost Fn. 3.98E-09
? More Iterations (def = Y) : N

? Enter "M" to modify this trim
    "R" to restart
    "/" to file data/quit : /

? Name of Output File (def= None) : /

? Enter "M" for Menu
    "/" to quit : /

```

Figure 3.6-2 Terminal display for trim.

acceptable, to reinitialize the step size of the minimization algorithm before each new set of iterations. More trim iterations were later performed on this example and the cost function reached a lower limit of $5.52\text{E-}13$ (the trim program and model use only single-precision arithmetic); no significant changes occurred in the numerical values given above. The final state and control vectors placed in the output file were as follows:

$$\begin{aligned}
 X_1 &= 5.020000\text{E} + 02, & X_2 &= 2.392628\text{E} - 01, & X_3 &= 5.061803\text{E} - 04, \\
 X_4 &= 1.366289\text{E} + 00, & X_5 &= 5.000808\text{E} - 02, & X_6 &= 2.340769\text{E} - 01, \\
 X_7 &= -1.499617\text{E} - 02, & X_8 &= 2.933811\text{E} - 01, & X_9 &= 6.084932\text{E} - 02, \\
 X_{10} &= 0.000000\text{E} + 00, & X_{11} &= 0.000000\text{E} + 00, & X_{12} &= 0.000000\text{E} + 00, \\
 X_{13} &= 6.412363\text{E} + 01, \\
 U_1 &= 8.349601\text{E} - 01, & U_2 &= -1.481766\text{E} + 00, & U_3 &= 9.553108\text{E} - 02, \\
 U_4 &= -4.118124\text{E} - 01
 \end{aligned}$$

This trim will be used for a flight simulation example in a following subsection and in Section 3.7 to illustrate coupling effects in the aircraft dynamics. ■

Trimmed Conditions for Studying Aircraft Dynamics

The steady-state performance of an airplane can be investigated very thoroughly from a set of trimmed flight conditions. The specific fuel consumption, rate of climb, various critical speeds for takeoff and landing, radius of turn, and so on, can all be determined for a number of different flight conditions. We have not provided enough modeling detail for all of these investigations, but the model and the trim program could be further developed if required.

Here we will examine the trimmed level-flight conditions over a range of speed. The F-16 is balanced to minimize trim drag, and for straight and level flight across the speed range of our model, the change in the trimmed elevator deflection is very small and varies erratically. At very low speeds, and therefore low dynamic pressure, a high value of the lift coefficient is needed to support the aircraft weight. This causes high induced drag, and because of the large angle of attack, the engine thrust must support a large component of the aircraft weight. Therefore, the throttle setting must increase at low speeds. The throttle setting also increases as transonic speeds are approached because of the increasing drag, and thus the throttle-setting v. speed curve must pass through a minimum.

Data for trimmed level flight at sea level, with the nominal cg position, are given in Table 3.6-2. As the speed is lowered, the angle of attack increases, the leading-edge flap reaches its limit (at about $\alpha = 18^\circ$, although no longer visible in the data), and the trimmed throttle setting begins to increase from its very low value. The model can be trimmed until alpha reaches about 45° , when a rapid increase in trimmed elevator deflection occurs, quickly reaching the deflection limit.

Figure 3.6-3 shows throttle setting plotted against airspeed. This curve is not the same as the airplane “power-required” curve because the engine characteristics are also included in it. Nevertheless, we shall loosely refer to it as the *power curve*. It shows clearly the minimum throttle setting. For a propeller-driven plane this is the condition for *best endurance* (but not best range) at the given altitude. For a jet plane the fuel consumption is more strongly related to thrust than power, so this is no longer

TABLE 3.6-2 Trim Data for the F-16 Model

Speed	130	140	150	170	200	260	300	350	400
Throttle	0.816	0.736	0.619	0.464	0.287	0.148	0.122	0.107	0.108
AOA	45.6	40.3	34.6	27.2	19.7	11.6	8.49	5.87	4.16
Elevation	20.1	−1.36	0.173	0.621	0.723	−0.090	−0.591	−0.539	−0.591
Speed	440	500	540	600	640	700	800	ft/s	
Throttle	0.113	0.137	0.160	0.200	0.230	0.282	0.378	Per unit	
AOA	3.19	2.14	1.63	1.04	0.742	0.382	−0.045	degrees	
Elevation	−0.671	−0.756	−0.798	−0.846	−0.871	−0.900	−0.943	degrees	

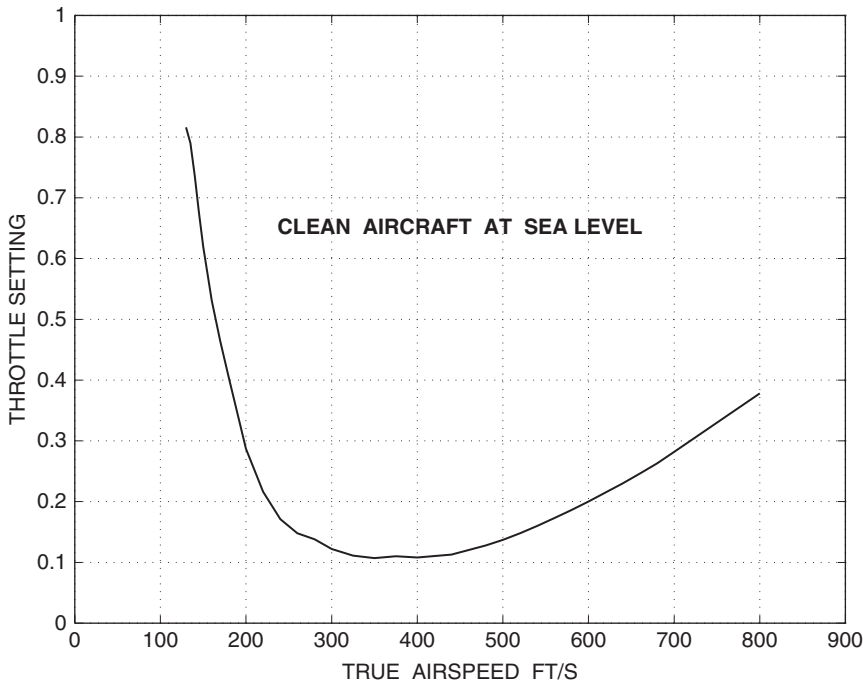


Figure 3.6-3 F-16 model, trimmed power curve.

true. For more details on the static performance information that can be derived from a power-available curve, see Dommasch et al. (1967).

The region to the left of the minimum of the power-required curve is known as the *back side of the power curve*. If the aircraft is operating on the back side of the power curve, opening the throttle produces an increase in altitude, *not* an increase in speed. The speed is then controlled by the elevator. This region of operation may be encountered in the landing phase of flight (e.g., carrier landings).

Table 3.6-3 presents another set of trimmed conditions for the F-16 model; these will be used for the simulation examples in this chapter and for controller design in subsequent chapters. The F-16 model aerodynamic data were referenced to the $0.35\bar{x}$ -position, and this is the “nominal” position for the cg. The nominal speed and altitude were chosen to give a representative flight condition suitable for later examples and designs. The table contains data for the nominal condition, a forward-cg condition, an aft-cg condition, and steady-state turn and pull-up conditions with a forward cg. The forward- and aft-cg cases have been included for a later demonstration of the effect of cg position on stability. A forward-cg location has been used for the two maneuvering cases so that the effects of the maneuver can be illustrated without the additional complication of unstable dynamics.

TABLE 3.6-3 Trimmed Flight Conditions for the F-16Nominal Condition: $h = 0$ ft, $\bar{q} = 300$ psf, $X_{cg} = .35\bar{c}$, $\dot{\phi} = \dot{\theta} = \dot{\psi} = \gamma = 0$

variable	CONDITION				
	Nominal	$X_{cg} = 0.3\bar{c}$	$X_{cg} = +0.38\bar{c}$	$X_{cg} = +0.3\bar{c}$ $\dot{\psi} = 0.3$ r/s	$X_{cg} = -0.3\bar{c}$ $\dot{\theta} = 0.3$ r/s
V_T (ft/s)	502.0	502.0	502.0	502.0	502.0
α (rad)	0.03691	0.03936	0.03544	0.2485	0.3006
β (rad)	$-4.0E-9$	$4.1E-9$	$3.1E-8$	$4.8E-4$	$4.1E-5$
ϕ (rad)	0	0	0	1.367	0
θ (rad)	0.03691	0.03936	0.03544	0.05185	0.3006
P (r/s)	0	0	0	-0.01555	0
Q (r/s)	0	0	0	0.2934	0.3000
R (r/s)	0	0	0	0.06071	0
THTL(0-1)	0.1385	0.1485	0.1325	0.8499	1.023
EL(deg)	-0.7588	-1.931	-0.05590	-6.256	-7.082
AIL(deg)	$-1.2E-7$	$-7.0E-8$	$-5.1E-7$	0.09891	$-6.2E-4$
RDR(deg)	$6.2E-7$	$8.3E-7$	$4.3E-6$	-0.4218	0.01655

Flight Simulation Examples

Here we give two flight simulation examples using the MATLAB simulation program from Section 3.4 with the transport aircraft model and one example using the F-16 model with a Fortran version of the simulation program.

Example 3.6-3: Simulated Response to an Elevator Pulse The transport aircraft model was trimmed for level flight in the “clean” condition at sea level, with $x_{cg} = 0.25$ and a true airspeed of 250 ft/s, using the trim program given in this section. The state and control vectors were

$$U^T = [0.1845 \quad -9.2184]; \quad X^T = [250 \quad 0.16192 \quad 0.16192 \quad 0 \quad 0]$$

A time-history simulation was performed using the program NLSIM.m, as given in Example 3.4-1, and with the above initial conditions. RK4 integration with a step size of 20 ms was used. An elevator-*doublet* pulse of 2° from 1 to 1.5 s and -2° from 1.5 to 2 s was superimposed on the trimmed elevator deflection using the code that was shown disabled in Example 3.4-1. A doublet is bidirectional with a mean value of zero and is intended to restore the original flight conditions when it ends.

Figure 3.6-4 shows the pitch-attitude and angle-of-attack responses to the elevator doublet. The initial pitch responses do not match (in shape or duration) the elevator disturbance that caused them. Instead, the responses are characteristic of the aircraft and represent a *natural mode* of the aircraft dynamics, in which alpha and theta vary

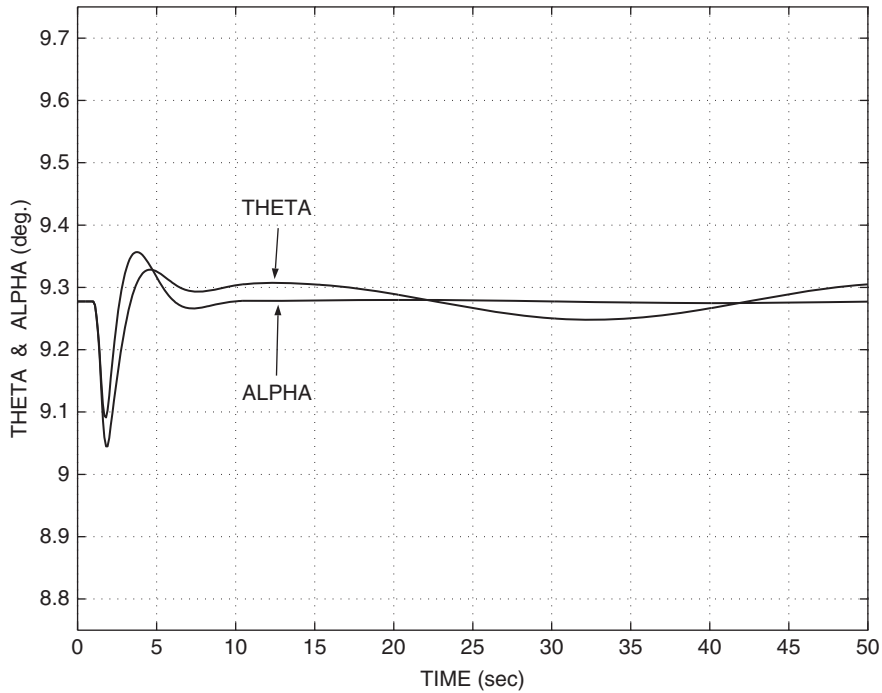


Figure 3.6-4 Transport aircraft, elevator-doublet response.

together, thus causing very little change in the flight-path angle. This mode is known as the *short-period mode*. If we inspect the other longitudinal variables, we will find that airspeed and altitude are almost constant, and only alpha, theta, and pitch rate vary. When the short-period response dies out, at about 10 s, alpha becomes constant and pitch rate becomes zero. There remains a small-amplitude, very lightly damped oscillation in which the aircraft gains altitude, with increasing pitch attitude and a positive flight-path angle and decreasing speed, and then reverses this motion. This is the *phugoid mode* of an aircraft. The short-duration elevator doublet may cause very little excitation of the phugoid mode if that mode is better damped than is the case here. ■

Example 3.6-4: Simulated Response to a Throttle Pulse In this example we will use the transport aircraft with the same trim conditions as Example 3.6-3 and superimpose a doublet pulse on the steady-state throttle setting. The doublet will have the value 0.1 from 1 to 4 s and -0.1 from 4 to 7 s. Figure 3.6-5 shows the response. The angle of attack is barely affected, but the pitch attitude exhibits the phugoid oscillation that was observed in Example 3.6-3. An examination of the speed, altitude, and flight-path

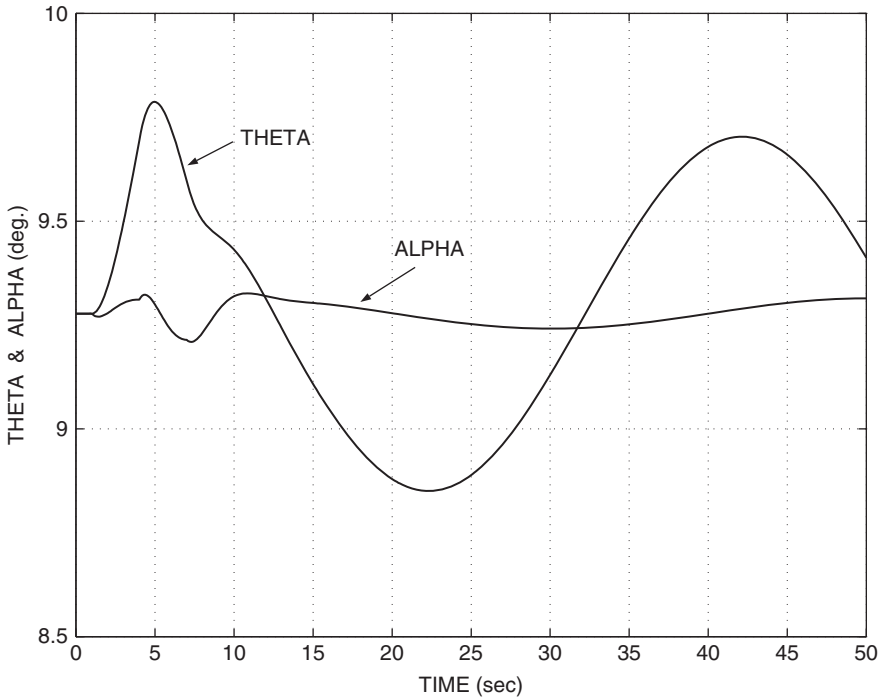


Figure 3.6-5 Transport aircraft, throttle-doublet response.

angle variables shows that they vary in unison with theta. Therefore, we conclude that the thrust disturbance has excited the phugoid mode, with very little effect on the short-period mode.

■

Example 3.6-5: Simulation of a Coordinated Turn This example is a time-history simulation of a steady-state coordinated turn using the F-16 model with the trim data from Example 3.6-2. The simulation data from the TRESP program are presented in Figure 3.6-6. The aircraft is turning at 0.3 rad/s and therefore turns through 54 rad or about 8.6 revolutions in the 180-s simulation.

Figure 3.6-7 shows the ground track of the aircraft and shows that the eight circles fall exactly over each other. In Section 3.8 we will see that the aircraft dynamics have quite a wide spread of time constants, and in this flight condition, there is an unstable mode with a time constant of about 1.7 s. Unless the integration time step is reduced below about 0.02 s, the fourth-order Runge-Kutta routine eventually diverges when integrating this example.


```
Initial condition filename ? (def.= none): EX362

Number of states and outputs to be recorded ?
# States (def.= 0) : 2
Which ones ? : 10 11
# Outputs (def.= 0) : /
Variable (V) or fixed (F) step integration ? (def= F) : /
Length of run (sec) ? : 180
Print time-interval (on screen) ? : 10
Plotting time-interval ? : .5
Sample period (integration step) ? : .01

      TIME      X-10      X-11
0.00E+00  0.00E+00  0.00E+00
1.00E+01  2.36E+02  3.33E+03
2.00E+01 -4.68E+02  6.65E+01
3.00E+01  6.90E+02  3.20E+03
4.00E+01 -8.97E+02  2.61E+02
5.00E+01  1.09E+03  2.94E+03
6.00E+01 -1.26E+03  5.68E+02
7.00E+01  1.40E+03  2.59E+03
8.00E+01 -1.51E+03  9.62E+02
9.00E+01  1.60E+03  2.16E+03
1.00E+02 -1.65E+03  1.41E+03
1.10E+02  1.67E+03  1.70E+03
1.20E+02 -1.66E+03  1.89E+03
1.30E+02  1.61E+03  1.22E+03
1.40E+02 -1.53E+03  2.35E+03
1.50E+02  1.42E+03  7.87E+02
1.60E+02 -1.28E+03  2.76E+03
1.70E+02  1.11E+03  4.22E+02
1.80E+02 -9.21E+02  3.07E+03

Enter  0 to file data      1 to quit
       2 to restart        3 to pick other states
       4 to change integration  5 to change run time :
```

Figure 3.6-6 Simulation results for F-16 model.

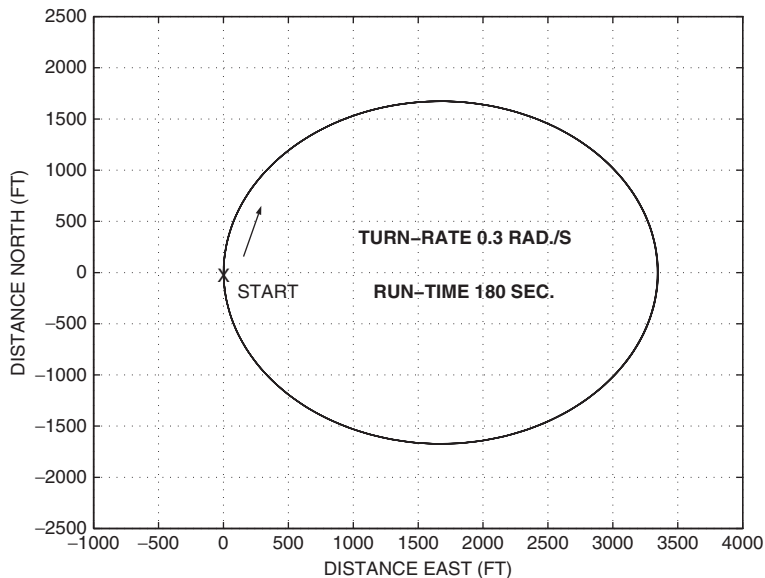


Figure 3.6-7 The ground track of a coordinated turn.

The foregoing examples have illustrated digital simulation using nonlinear continuous-time dynamic equations, with control inputs applied in discrete time (i.e., changing only at the sampling instants). In the next section we will derive linear dynamic equations; these offer no advantages for simulation but do allow a variety of analytical tools to be applied to the dynamics.

3.7 NUMERICAL LINEARIZATION

Theory of Linearization

In Section 2.6 we linearized the aircraft implicit nonlinear state equations algebraically and obtained LTI state equations corresponding to a given flight condition. This linearization was specific to aircraft equations and was only tractable under the restrictions of wings-level, non-sideslipping steady-state flight. Now we will introduce a numerical linearization algorithm that can be applied to any nonlinear model in the same explicit state-space form that was used with numerical integration.

A multivariate Taylor series expansion of the explicit state equations (3.4-1b) around a point (X_e, U_e) gives

$$\dot{X} + \delta\dot{X} = f(X_e, U_e) + \frac{\partial f}{\partial X}\delta X + \frac{\partial f}{\partial U}\delta U + \text{h.o.t.},$$

where the partial derivative terms denote Jacobian matrices (as in Section 2.6) and the perturbations

$$\delta X \equiv (X - X_e), \quad \delta U \equiv (U - U_e)$$

are “small.” In the series “h.o.t.” denotes higher-order terms, which will be neglected. If X_e and U_e are equilibrium solutions obtained from the trim program, then

$$0 = \dot{X} = f(X_e, U_e)$$

and so

$$\delta\dot{X} = \frac{\partial f}{\partial X}\delta X + \frac{\partial f}{\partial U}\delta U \quad (3.7-1)$$

This equation is in the form of the LTI state equation,

$$\dot{x} = Ax + Bu, \quad (3.7-2)$$

where the lowercase symbols denote perturbations from the equilibrium, but \dot{x} is the actual value of the derivative vector.

The method of estimating the first partial derivatives, which make up the Jacobian matrices, will be illustrated with a function of a single variable $z = g(v)$. Using Taylor series expansions of g around $v = v_e$, we obtain

$$\begin{aligned} z_1 \equiv g(v_e + h) &= g(v_e) + h \frac{\partial g}{\partial v}(v_e) + \frac{h^2}{2!} \frac{\partial^2 g}{\partial v^2}(v_e) + \text{h.o.t.} \\ z_{-1} \equiv g(v_e - h) &= g(v_e) - h \frac{\partial g}{\partial v}(v_e) + \dots \end{aligned}$$

Then it is easy to see that

$$\left. \frac{\partial g}{\partial v} \right|_{v=v_e} = \frac{z_1 - z_{-1}}{2h} - \frac{h^2}{3!} \frac{\partial^3 g}{\partial v^3}(v_e) - \text{h.o.t.} \quad (3.7-3)$$

and neglecting terms of order h^2 and higher leaves a very simple approximation for the first partial derivative.

A higher-order approximation can be found by writing the Taylor series for

$$z_2 = g(v_e + 2h)$$

and

$$z_{-2} = g(v_e - 2h)$$

It can then be shown that

$$\left. \frac{\partial g}{\partial v} \right|_{v=v_e} = \frac{8(z_1 - z_{-1}) - (z_2 - z_{-2})}{12h} + O(h^4) \quad (3.7-4)$$

Therefore, by using four values of the function g , we can obtain an estimate of the first partial derivative that includes Taylor series terms through h^3 .

Algorithm and Examples

When turning the formulae for the partial derivatives into a numerical algorithm, one must determine what size of perturbation can be considered “small” in Equation (3.7-1). The perturbations may often be around an equilibrium value of zero, so it is not always possible to choose some fraction of the equilibrium value. Instead, one can start with a fairly arbitrary initial perturbation and progressively reduce it until the algorithm obtained from (3.7-3) or (3.7-4) converges on some value for the derivative. Figure 3.7-1 shows a flowchart for numerical linearization, and a simple MATLAB program is given below.

```
% File LINZE.m
clear all
name = input('Enter Name of State Eqns. File : ','s');
tfile= input('Enter Name of Trim File : ','s');
tmp= dlmread(tfile,',' );
n=tmp(1); m=tmp(2); x=tmp(3:n+2);
u=tmp(n+3:m+n+2); tol=1e-6; time=0.;
mm= input('Number of control inputs to be used ? : ');
dx=0.1*x;
for i=1:n % Set Perturbations
    if dx(i)==0.0;
        dx(i)=0.1;
    end
end
last=zeros(n,1); a=zeros(n,n);
for j=1:n
```

```

xt=x;
for i=1:10
    xt(j)=x(j)+dx(j);
    xd1= feval (name,time,xt,u);
    xt(j)=x(j)-dx(j);
    xd2= feval (name,time,xt,u);
    a(:,j)= (xd1-xd2)'/(2*dx(j));
    if max( abs(a(:,j))-last)./abs( a(:,j) + 1e-12 )<tol;
        break
    end
    dx(j)= 0.5*dx(j);
    last = a(:,j);
end
%column=j
iteration=i;
if iteration==10
    disp('not converged on A, column',num2str(j))
end
end
end
dlmwrite('A.dat',a);

```

This program computes one column of A at a time; a fractional error is computed for each element of the column array, and the largest fractional error must satisfy a convergence tolerance. If the algorithm has not converged after dividing the initial perturbation by 2^9 , the user is informed and must deal with the problem by increasing the tolerance or linearizing at a slightly different flight condition. To save space, the

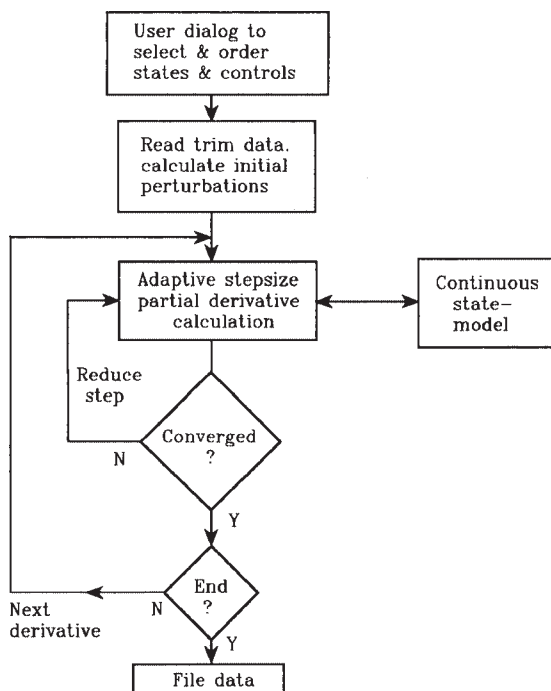


Figure 3.7-1 Flowchart for linearization.

calculation of the B -matrix is not shown; note that not all of the control variables should be perturbed (e.g., the landing gear switch) so the user must enter a value for the variable mm .

For a differentiable function, perturbations that are too large produce a (Taylor series) truncation error, while perturbations that are too small cause a round-off error due to the finite-precision computer arithmetic. Many simulations are written with single-precision arithmetic, and round-off error will then be much more significant than with MATLAB double-precision arithmetic. Difficulties are also caused by discontinuities in the function or its derivative. For example, if a simulation variable has reached a “hard” limit, this will cause a discontinuity in its derivative. A more sophisticated linearization algorithm may be designed to find an optimal-size perturbation and to detect discontinuities (see, for example, Taylor and Antonioti, 1993). The student should add the calculations for the B -, C -, and D -matrices to the above program (see Problem 3.7-2). A linearization program “Jacob” for the Fortran model is given in Appendix B.

Example 3.7-1: Comparison of Algebraic and Numerical Linearization This example uses the transport aircraft longitudinal model of Section 3.5. The model contains an alpha-dot contribution to the pitching moment, the thrust vector is offset from the cg by the amount ZE , and the engine thrust varies with speed. It therefore provides a good check on the results of the algebraic linearization in Section 2.6.

A short program was written to evaluate the longitudinal state equation coefficient matrices (2.6-29) using the formulae in Tables 2.6-1 to 2.6-3 (see Problem 3.7-3). The program contains the dimensionless stability derivatives given in the transport aircraft model and reads the steady-state trim data from a data file. It calculates the A - and B -matrices in (2.6-30) and then premultiplies them by the inverse of the E -matrix. The new A - and B -matrices were printed out for comparison with numerical linearization results.

The model was trimmed in the clean condition at a large angle of attack and in climbing flight, so that $\sin \alpha$ and $\sin \gamma$ terms contributed significantly to the results. The trim condition was $cg = 0.25\bar{c}$, $h = 0$ ft, $V_T = 200$ ft/s, and $\gamma = 15^\circ$. This condition required an angle of attack of 13.9° and a throttle setting of 1.01 (i.e., slightly beyond maximum power!). The algebraic linearization program gave

$$E^{-1}A = \begin{array}{c} \begin{array}{cccc} v_T & \alpha & \theta & q \end{array} \\ \begin{bmatrix} -2.7337E-02 & 1.6853E+01 & -3.1074E+01 & 0.0000E+00 \\ -1.4167E-03 & -5.1234E-01 & -4.1631E-02 & 1.0000E+00 \\ 0.0000E+00 & 0.0000E+00 & 0.0000E+00 & 1.0000E+00 \\ -1.1415E-04 & -4.9581E-01 & 4.8119E-03 & -4.2381E-01 \end{bmatrix} \end{array}$$

$$E^{-1}B = \begin{array}{c} \begin{array}{cc} \delta_i & \delta_e \end{array} \\ \begin{bmatrix} 1.0173E+01 & 0.0000E+00 \\ -1.2596E-02 & 0.0000E+00 \\ 0.0000E+00 & 0.0000E+00 \\ 2.7017E-02 & -7.0452E-03 \end{bmatrix} \end{array}$$

The algebraic linearization did not account for an altitude state and the consequent coupling of the equations through the atmosphere model. Therefore, only the first four states were selected when the numerical linearization was performed. The numerical linearization produced the following results:

$$A = \begin{bmatrix} -2.7337E-02 & 1.6852E+01 & -3.1073E+01 & 0.0000E+00 \\ -1.4168E-03 & -5.1232E-01 & -4.1630E-02 & 1.0000E+00 \\ 0.0000E+00 & 0.0000E+00 & 0.0000E+00 & 1.0000E+00 \\ -1.1415E-04 & -4.9583E-01 & 4.8118E-03 & -4.2381E-01 \end{bmatrix}$$

$$B = \begin{bmatrix} 1.0173E+01 & 0.0000E+00 \\ -1.2596E-02 & 0.0000E+00 \\ 0.0000E+00 & 0.0000E+00 \\ 2.7017E-02 & -7.0452E-03 \end{bmatrix}$$

These results are in very close agreement with the algebraic linearization results; the largest discrepancy is a difference of 2 in the fifth digit. ■

The stability derivatives and numerical linearization play complementary roles. The stability derivatives are useful for preliminary design; they can be estimated from a geometrical description of an aircraft and can be used to calculate the modes and stability of the aircraft. When an aircraft configuration has been chosen and enough data have been gathered to build a mathematical model, numerical linearization can be used to perform control system design and to obtain a dynamic description of the aircraft in other than wings-level non-sideslipping flight.

Example 3.7-2: Linearization of the F-16 Model In Chapter 2 it was shown that under the conditions of small perturbations from steady-state, wings-level, non-sideslipping flight, the rigid-aircraft equations of motion could be split into two uncoupled sets. These were the longitudinal equations that involve the variables speed, alpha, pitch attitude, and pitch rate and the lateral-directional equations that involve beta, roll angle, and roll and yaw rates. The Jacob program makes it easy to demonstrate this decoupling and to show that coupling occurs when the sideslip and roll angles are nonzero. A good example is provided by two steady-state flight conditions that differ only in terms of roll angle. In Table 3.6-3, the wings-level pull-up with $\dot{\theta} = 0.3$ rad/s and the coordinated turn at 0.3 rad/s are both at 300 psf dynamic pressure with zero sideslip and similar angles of attack (and almost identical normal acceleration). The Jacobian matrices for these two steady-state conditions will now be compared.

The Fortran linearization program makes provision for reordering states and for choosing subsets of states. The north and east geographic position states and the geographic heading state ψ have no effect on the dynamic behavior; their derivatives are a function of the other states, but these states themselves are not coupled back into the state equations. Also, the altitude state only enters the aircraft equations through the atmosphere model and dynamic pressure, and in this case it has negligible coupling to the other states. Therefore, we will not select these state variables for the LTI model.

To illustrate decoupling, Jacob was used to reorder the remaining nine states into longitudinal states v_T , α , θ , q and the engine power state (POW), followed by lateral directional states β , ϕ , p , r . The inputs were ordered as δ_t , δ_e , δ_a , δ_r and the outputs as a_n , q , α . The results for the steady-state pull-up are shown in Figure 3.7-2.

These results are rounded to three significant digits, except for numbers less than 0.001, which are rounded to only one significant digit. The A - and B -matrices have been partitioned to separate the longitudinal and lateral states and controls, and it is evident that the expected decoupling does indeed exist. In the A -matrix, the exact relationship $\dot{\theta} = q$, when $\phi = 0$, is evident on the third row. There is a small amount of coupling of \dot{V}_T to β and ϕ and of \dot{q} to r , but the other terms are one to two orders of magnitude smaller. The B -matrix shows that the aileron and rudder have essentially no effect on the longitudinal states and the throttle and elevator have no effect on the lateral-directional states.

The D -matrix has a nonzero entry, corresponding to the elevator to normal acceleration transfer function, because accelerations are directly coupled to forces and therefore to control surface deflection. Other expected features are the reciprocal of the engine time constant (at full power) as the only nonzero entry on the fifth row

$$\begin{aligned}
 A = & \begin{array}{c} \begin{array}{ccccccccc} v_T & \alpha & \theta & q & \text{pow} & \beta & \phi & p & r \end{array} \\ \left[\begin{array}{ccccccccc} -.127 & -.235 & -32.2 & -9.51 & .314 & : & -.0028 & .00126 & 5E-5 & 2E-4 \\ -7E-4 & -.969 & 0 & .908 & -2E-4 & : & 1.5E-5 & 0 & -4E-5 & -1E-5 \\ 0 & 0 & 0 & 1 & 0 & : & 0 & 0 & 0 & 0 \\ 9E-4 & -4.56 & 0 & -1.58 & 0 & : & 9.2E-5 & 0 & 0 & -.00287 \\ 0 & 0 & 0 & 0 & -5.00 & : & 0 & 0 & 0 & 0 \end{array} \right. \\ \hline & \left[\begin{array}{ccccccccc} 1E-8 & 2E-5 & 3E-6 & 8E-7 & -3E-8 & : & -.322 & .0612 & .298 & -.948 \\ 0 & 0 & 0 & 0 & 0 & : & 0 & .0930 & 1.00 & .310 \\ -3E-7 & -.00248 & 0 & 3E-4 & 0 & : & -62.5 & 0 & -3.00 & 1.99 \\ -3E-6 & -.00188 & 0 & .00254 & 0 & : & 7.67 & 0 & -.262 & -.629 \end{array} \right] \end{array} \\
B = & \begin{array}{c} \begin{array}{cccc} \delta_t & \delta_e & \delta_a & \delta_r \end{array} \\ \left[\begin{array}{cccc} 0 & -.244 & : & 6E-6 & 2E-5 \\ 0 & -.00209 & : & 0 & 0 \\ 0 & 0 & : & 0 & 0 \\ 0 & -.199 & : & 0 & 0 \\ 1087 & 0 & : & 0 & 0 \end{array} \right. \\ \hline & \left[\begin{array}{cccc} 0 & 2E-8 & : & 3E-4 & 8E-4 \\ 0 & 0 & : & 0 & 0 \\ 0 & 0 & : & -.645 & .126 \\ 0 & 0 & : & -.0180 & -.0657 \end{array} \right] \end{array} \\
D = & \begin{bmatrix} 0 & .0333 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
C = & \begin{bmatrix} .0208 & 15.2 & 0 & 1.45 & 0 & -4.5E-4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.00 & 0 & 0 & 0 & 0 & 0 \\ 0 & 57.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{array}{c} a_n \\ q \\ \alpha \end{array}
\end{aligned}$$

Figure 3.7-2 Jacobian matrices for a pull-up.

$$A = \begin{bmatrix} -.090 & -.169 & -.31.2 & -.7.75 & .318 & : & 31.4 & -.7.73 & 5E-4 & 2E-3 \\ -5E-4 & -.1.05 & .0151 & .903 & -2E-4 & : & 3E-4 & -.0607 & -5E-4 & -1E-4 \\ 0 & 0 & 0 & .203 & 0 & : & 0 & -.300 & 0 & -.979 \\ 1E-3 & 1.26 & 0 & -1.66 & 0 & : & 1E-3 & 0 & .0589 & -.0157 \\ 0 & 0 & 0 & 0 & -5.00 & : & 0 & 0 & 0 & 0 \\ \hline -1E-4 & 1.4E-4 & -.0032 & 7E-6 & -3E-7 & : & -.322 & .0130 & .248 & -.961 \\ 0 & 0 & .300 & .0508 & 0 & : & 0 & 0 & 1.00 & .0105 \\ -3E-4 & .0578 & 0 & -.0469 & 0 & : & -59.4 & 0 & -3.19 & 1.64 \\ 5E-5 & -.0617 & 0 & .0123 & 0 & : & 8.88 & 0 & -.299 & -.564 \end{bmatrix}$$

Figure 3.7-3 Jacobian matrix for 4.5 g turn.

of the A -matrix and the value of g appearing in the (1, 3) position. The reader should compare these results with the state equations derived in Chapter 2 and determine the significance of the numerical values.

If we next use the Jacobian program to determine the A -matrix for the 4.5 g coordinated turn, we obtain the matrix shown in Figure 3.7-3. A comparison of this matrix with the previous A -matrix shows that several strong coupling terms have now appeared in the upper right block. Less pronounced coupling has appeared in the lower left block. These couplings can be understood by referring to the nonlinear equations (2.5-19), the nonlinear moment equations, and the Euler angle equations (1.4-5), with ϕ nonzero. In the next section we develop the tools to determine how the dynamic behavior is changed by this coupling, and in Chapter 4 we consider the implications of these changes in the dynamics. ■

3.8 AIRCRAFT DYNAMIC BEHAVIOR

In Examples 3.6-3 and 3.6-4 two different characteristic modes of the aircraft dynamics were excited separately by applying different inputs. One of the variables (θ) observed in the simulation was found to be involved in both modes; the other observed variable (α) was essentially involved in only one. In this section we will study the dynamic behavior analytically through modal decomposition and analysis of the transfer functions.

Modal Decomposition Applied to Aircraft Dynamics

In Section 3.6 the classical phugoid and short-period aircraft modes were illustrated by nonlinear simulation. The complete set of modes of a conventional aircraft will now be illustrated by modal decomposition using a linear F-16 model. The second set of trim conditions in Table 3.6-3 will be used, that is, straight and level flight with stable dynamics. A Jacobian A -matrix must first be found for this flight condition, and not all of the thirteen states in the full A -matrix will be needed.

Once again we will drop the north and east geographic position states and the geographic heading state, which do not affect the dynamics. These states

correspond to the integrals of linear combinations of other states and, if retained in the A -matrix, will produce zero eigenvalues (poles at the origin). The engine power state couples into the dynamics (through V_T) but is not influenced by any other states. Left in the A -matrix, it will produce an eigenvalue of -1.0 , corresponding to the reciprocal of the 1-s engine time constant, and again does not affect the aircraft modes.

There is also clear decoupling of the lateral and longitudinal dynamics in this flight condition. Therefore, the modal decomposition will be demonstrated using two separate reduced Jacobian matrices. Note that the method of deriving the A -matrix by perturbing the state variables assumes that the control inputs are constant. Therefore, the modes derived in the analysis are “stick-fixed” modes, that is, the control surfaces are implicitly assumed to be locked in position. This assumption will hold most accurately for fully powered (as opposed to power-boosted or unpowered) control surfaces; these control systems are called *irreversible*.

Example 3.8-1: F-16 Longitudinal Modes The IMSL eigenvalue subroutine EIGRF (IMSL, 1980), with double precision, was used to produce the following results. Other sources of eigenvalue/eigenvector routines are readily available (Press et al., 1986; MATLAB, 1990). A simple driver program was written, and each pair of eigenvectors was normalized by dividing all elements by the complex number corresponding to the element of greatest magnitude. The longitudinal dynamics Jacobian matrix for the F-16 model in straight and level flight at 502.0 ft/s with a cg position of $0.3\bar{c}$ is given by

$$A = \begin{matrix} & \begin{matrix} v_T & \alpha & \theta & q \end{matrix} \\ \begin{bmatrix} -2.0244E - 02 & 7.8763E + 00 & -3.2170E + 01 & -6.5020E - 01 \\ -2.5372E - 04 & -1.0190E + 00 & 0.0000E + 00 & 9.0484E - 01 \\ 0.0000E + 00 & 0.0000E + 00 & 0.0000E + 00 & 1.0000E + 00 \\ 7.9472E - 11 & -2.4982E + 00 & 0.0000E + 00 & -1.3861E + 00 \end{bmatrix} \end{matrix}$$

The four states give rise to two complex conjugate pairs of eigenvalues, which correspond to two stable oscillatory modes. The eigenvalues are

$$\begin{aligned} & -1.2039 \pm j1.4922 && (\text{short-period mode, } T = 4.21\text{s, } \zeta = 0.628) \\ & -0.0087297 \pm j0.073966 && (\text{phugoid mode, } T = 84.9\text{s, } \zeta = 0.117) \end{aligned}$$

The periods of these modes are separated by more than an order of magnitude, so they are easily identifiable as the short-period and phugoid modes. The phugoid mode is very lightly damped ($\zeta = 0.117$), but its period is so long that a pilot would have no difficulty in damping out a phugoid oscillation. The short-period mode is reasonably well damped in this particular flight condition, and the aircraft response to elevator commands would be acceptable to the pilot.

The corresponding eigenvectors are two complex conjugate pairs given by

<i>short-period</i>	<i>phugoid</i>
$\begin{bmatrix} 1.0 \\ 0.090 \\ 0.059 \\ 0.0092 \end{bmatrix} \pm j \begin{bmatrix} 0 \\ 0.017 \\ 0.054 \\ 0.15 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ -9.6E-5 \\ -3.8E-4 \\ 1.7E-4 \end{bmatrix} \pm j \begin{bmatrix} 0 \\ 5.0E-7 \\ 2.3E-3 \\ 8.4E-6 \end{bmatrix} \begin{matrix} v_T \\ \alpha \\ \theta \\ q \end{matrix}$

Both pairs of eigenvectors are dominated by the element corresponding to airspeed, and the relative involvement of the other variables is difficult to assess. Nevertheless, the results show that the variables α and q are involved relatively weakly in the phugoid mode as compared to the short period, and the phugoid mode involves mostly V_T and θ . This agrees with the conclusions drawn from the nonlinear simulation examples.

The relative involvement of different variables in the dynamic modes can be determined more precisely if the dynamic equations are made dimensionless, so that the eigenvectors are also dimensionless. This requires the introduction of time scaling (Etkin, 1972). Additional information can be extracted from the eigenvectors if they are plotted in the complex plane so that their phase relationships can be observed (Etkin, 1972). ■

Example 3.8-2: F-16 Lateral-Directional Modes The Jacobian matrix for the lateral-directional dynamics of the F-16 model in straight and level flight at 502.0 ft/s with a cg position of $0.3\bar{c}$ is given by

$$A = \begin{matrix} & \beta & \phi & p & r \\ \begin{bmatrix} -3.2200E-01 & 6.4032E-02 & 3.8904E-02 & -9.9156E-01 \\ 0.0000E+00 & 0.0000E+00 & 1.0000E+00 & 3.9385E-02 \\ -3.0919E+01 & 0.0000E+00 & -3.6730E+00 & 6.7425E-01 \\ 9.4724E+00 & 0.0000E+00 & -2.6358E-02 & -4.9849E-01 \end{bmatrix} \end{matrix}$$

This time there are two real eigenvalues and a complex conjugate pair. They are

$-0.4399 \pm j3.220$	(dutch roll mode, $T = 1.95s, \zeta = 0.135$)
-3.601	(roll subsidence mode, $\tau = 0.28 \text{ s}$)
-0.0128	(spiral mode, $\tau = 77.9 \text{ s}$)

The eigenvectors are

<i>Dutch Roll Mode</i>	<i>Roll Mode</i>	<i>Spiral Mode</i>
$\begin{bmatrix} -0.11 \\ -0.037 \\ 1.0 \\ -0.29 \end{bmatrix} \pm j \begin{bmatrix} -0.097 \\ -0.30 \\ 0 \\ 0.33 \end{bmatrix}$	$\begin{bmatrix} -0.0020 \\ -0.28 \\ 1 \\ 0.015 \end{bmatrix}$	$\begin{bmatrix} 0.0032 \\ 1 \\ -0.015 \\ 0.063 \end{bmatrix} \begin{matrix} \beta \\ \phi \\ p \\ r \end{matrix}$

The oscillatory mode involves all of the variables and is a rolling and yawing motion with some sideslipping. This motion has been likened to the motion of a drunken skater and is called the *dutch roll mode*. The aircraft rudder produces both rolling and yawing moments, and a rudder pulse will excite this mode. The eigenvalues show that the dutch roll period is quite short ($T = 1.95$ s) and the oscillation is very lightly damped ($\zeta = 0.135$). This would make landing in gusty wind conditions difficult for the pilot, and in a passenger aircraft, passengers sitting near the tail would be very uncomfortable in turbulent conditions.

The second mode is simply a stable exponential mode and clearly involves mostly roll rate and a corresponding roll angle; it is known as the *roll subsidence mode*. The aircraft roll angle response to lateral control inputs is an important part of the handling qualities requirements. This mode, derived from the linear model, will not allow the aircraft maximum roll rate to be calculated but does give a good idea of how quickly the aircraft will start to roll. In this case the time constant of 0.28 s indicates a fast roll response.

The third mode is also a stable exponential mode but is distinguished by a much longer time constant (78 s). It involves more roll angle and yaw rate than the roll mode and is known as the *spiral mode*. The small amount of sideslip shows that the spiral mode can be a coordinated motion. In some aircraft the spiral mode may be unstable, and stability can be built into a design by using wing dihedral (see Chapter 2). An unstable spiral mode can cause an aircraft to get into an ever-steeper, but coordinated, spiral dive. ■

Interpretation of Aircraft Transfer Functions

We now look at the aircraft dynamic behavior through various transfer functions. More specifically, we will look for pole-zero cancellations to determine what modes remain involved in a particular response. We will also look at the frequency response of a transfer function in order to improve our understanding of the correlation between the frequency and time domains.

Example 3.8-3: F-16 Elevator-to-Pitch-Rate Transfer Function For this example a full thirteen-state Jacobian A -matrix was obtained for the straight and level flight conditions used in Examples 3.8-1 and 3.8-2. The B and C Jacobian matrices were also obtained, and from these matrices the elevator-to-pitch-rate transfer function was selected. The D -matrix is 1×1 and is null, meaning that the transfer function has a relative degree of unity or greater.

Table 3.8-1 shows the static loop sensitivity and poles and zeros resulting from double-precision computations rounded to seven digits. The poles and zeros have been ordered to suit the purposes of this example. Because of the straight and level flight condition of the symmetrical aircraft, all of the lateral-directional poles will be canceled by zeros. There will be three poles at the origin, corresponding to integration of north and east velocity components, and integration of heading rate. These will also be canceled by zeros since those states are not involved in the dynamics. The throttle input is not being used and so the 1-s engine lag will be canceled by a zero.

TABLE 3.8-1 F-16 Model, Elevator-to-Pitch-Rate Transfer Function

Static Loop Sensitivity = -10.453 (deg. units)				
ZEROS		POLES		
Real Part	Imaginary Part	Real Part	Imaginary Part	
0.0000E + 00	0.0000E + 00	0.0000E + 00	0.0000E + 00	N
0.0000E + 00	0.0000E + 00	0.0000E + 00	0.0000E + 00	E
0.0000E + 00	0.0000E + 00	0.0000E + 00	0.0000E + 00	ψ
-4.3987E-01	3.2200E + 00	-4.3987E-01	3.2200E + 00	dutch
-4.3987E-01	-3.2200E + 00	-4.3987E-01	-3.2200E + 00	dutch
-3.6009E + 00	0.0000E + 00	-3.6009E + 00	0.0000E + 00	roll
-1.2835E-02	0.0000E + 00	-1.2835E-02	0.0000E + 00	spiral
-8.8010E-04	0.0000E + 00	-2.0874E-03	0.0000E + 00	Alt.
-1.0000E + 00	0.0000E + 00	-1.0000E + 00	0.0000E + 00	Engine
0.0000E + 00	0.0000E + 00	-1.2040E + 00	1.4923E + 00	SP
-2.1785E-02	0.0000E + 00	-1.2040E + 00	-1.4923E + 00	SP
-9.8713E-01	0.0000E + 00	-7.6538E-03	7.8119E-02	PHUG
		-7.6538E-03	-7.8119E-02	PHUG

The table shows that all of these cancellations are exact within the precision of the calculations. The remaining poles correspond to the phugoid and short-period modes and the altitude pole. The altitude pole is a very slow pole and is only weakly involved in the transfer function because of the nearby zero, so that the elevator-to-pitch-rate transfer function is approximately given by

$$\frac{q}{\delta_e} = \frac{-10.45s(s + 0.9871)(s + 0.02179)}{(s + 1.204 \pm j1.492)(s + 0.007654 \pm j0.07812)} \frac{\text{deg/s}}{\text{deg}}$$

The phugoid mode has a natural frequency of 0.079 rad/s and a damping ratio of 0.10; the corresponding figures for the short-period mode are 1.9 rad/s and 0.63. This transfer function has a dc gain of zero (because of the zero at the origin), indicating that a constant elevator deflection will not sustain a steady pitch rate. If the phugoid poles are canceled with the zero at the origin and the zero at $s = -0.02$, a *short-period approximation* transfer function is obtained:

$$\frac{q}{\delta_e} = \frac{-10.45(s + 0.9871)}{(s + 1.204 \pm j1.492)} \frac{\text{deg/s}}{\text{deg}}$$

This transfer function has a finite dc gain and shows that constant elevator deflection tends to produce constant pitch rate over an interval of time that is short compared to the phugoid period. The short-period approximation will be used in controller designs in Chapter 4, and its validity will be demonstrated. In the next example the short-period approximation will be examined in the frequency domain. ■

Example 3.8-4: F-16 Elevator-to-Pitch-Rate Frequency Response The poles and zeros of the elevator-to-pitch-rate transfer function, given in Example 3.8-3, will now

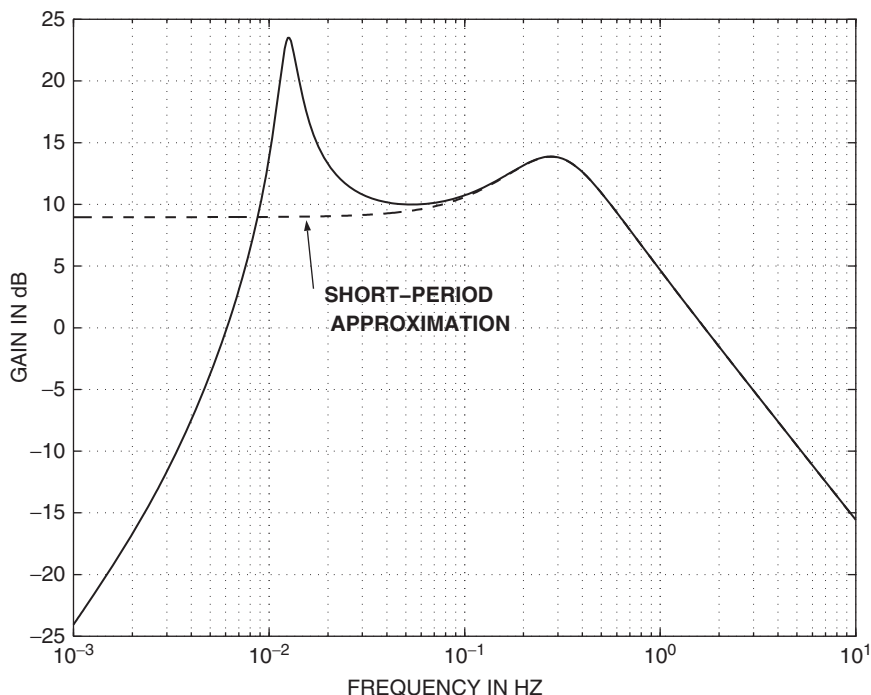


Figure 3.8-1a Bode gain plot, elevator to pitch rate.

be used to generate the corresponding frequency-response plots. Figure 3.8-1a shows a Bode plot of the magnitude response of both the complete transfer function and the short-period approximation. The phase plots are shown in Figure 3.8-1b. The magnitude plot shows a large peak in the response at a frequency close to the natural frequency of the lightly damped phugoid mode and a smaller peak due to the more heavily damped short-period mode.

Both the magnitude and phase plots show that the short-period approximation is a good approximation to the pitch-rate transfer function at frequencies above about 0.03 Hz. The upper *cutoff* or *corner* frequency of the short-period transfer function is about 0.8 Hz, and this gives some feel for the speed of response in pitch when different aircraft are compared. Note that the exact phase plot starts at $+90^\circ$ due to the zero at the origin, rises toward 180° because of the additional phase lead of the zero at $s = -0.02$, and then falls back rapidly because of the 180° lag effect of the phugoid poles. The zero at $s = -0.99$ causes another small lead effect before the lag of the short-period poles takes over; the high-frequency asymptotic phase shift is -90° because the relative degree of the transfer function is unity.

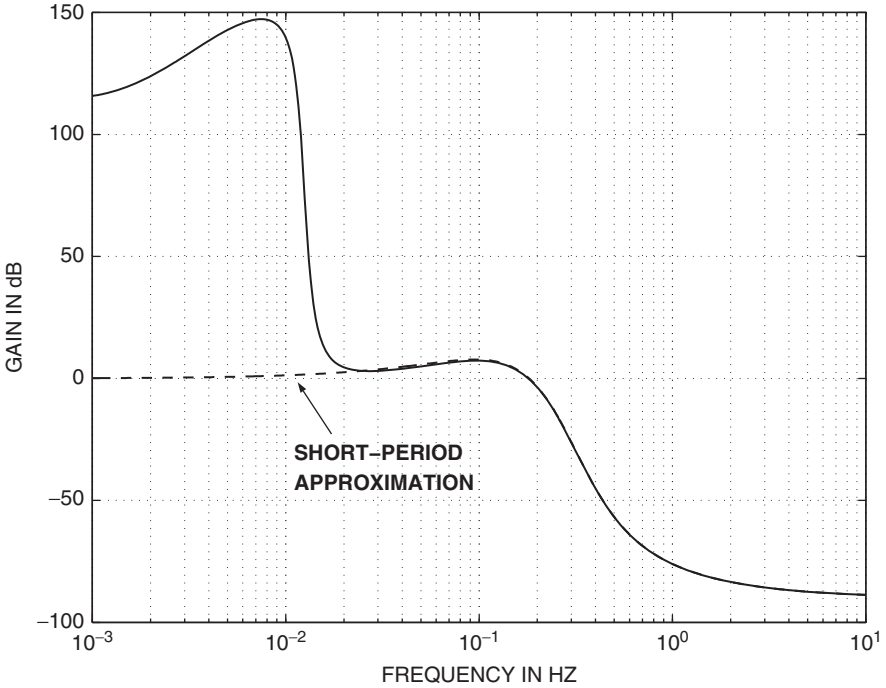


Figure 3.8-1b Bode phase plot of the pitch-rate transfer function. ■

Example 3.8-5: Transport Aircraft Throttle Response In this example we examine the throttle-to-speed transfer function for the transport aircraft model in Section 3.5 using the same flight condition as Examples 3.6-3 and 3.6-4. The model was trimmed for level flight at sea level in the clean configuration, with $x_{cg} = 0.25\bar{c}$ and a true airspeed of 250 ft/s, and the following Jacobian matrices were determined:

$$A = \begin{bmatrix} v_T & \alpha & \theta & q & h \\ -1.6096E-02 & 1.8832E+01 & -3.2170E+01 & 0.0000E+00 & 5.4000E-05 \\ -1.0189E-03 & -6.3537E-01 & 0.0000E+00 & 1.0000E+00 & 3.7000E-06 \\ 0.0000E+00 & 0.0000E+00 & 0.0000E+00 & 1.0000E+00 & 0.0000E+00 \\ 1.0744E-04 & -7.7544E-01 & 0.0000E+00 & -5.2977E-01 & -4.1000E-07 \\ 0.0000E+00 & -2.5000E+02 & 2.5000E+02 & 0.0000E+00 & 0.0000E+00 \end{bmatrix}$$

$$B = \begin{bmatrix} 9.9679E+00 \\ -6.5130E-03 \\ 0.0000E+00 \\ 2.5575E-02 \\ 0.0000E+00 \end{bmatrix}, \quad C = [1 \ 0 \ 0 \ 0 \ 0], \quad D = 0$$

The altitude state h has very small coupling to the other states and was initially neglected. The throttle-to-speed transfer function (with the elevator fixed), as determined from the states v_T , α , θ , q , was found to be

$$\frac{V_T}{\delta_t} = \frac{9.968(s - 0.0601)(s + 0.6065 \pm j0.8811)}{(s + 2.277E - 4 \pm j0.1567)(s + 0.5904 \pm j0.8811)} \quad (1)$$

As expected, this transfer function essentially involves only the phugoid mode, and when the short-period poles are canceled with the nearby complex zeros, we are left with the approximation

$$\frac{V_T}{\delta_t} = \frac{9.968(s - 0.0601)}{(s + 2.277E - 4 \pm j0.1567)} \quad (2)$$

The poles and zeros of (2) are quite close to the origin and the relative degree is unity, so throttle inputs are initially integrated. However, the phugoid mode will soon take over and hide this effect under a very lightly damped oscillation in speed. In addition, the NMP zero indicates that there are competing physical mechanisms at work. It may be remembered that the engine thrust line is offset below the cg, and this will cause the aircraft to tend to pitch up and consequently slow down in response to a sudden increase in throttle. Furthermore, at this relatively low speed the aircraft is trimmed with a large amount of “up elevator,” so that any initial increase in speed tends to create an increase in the nose-up pitching moment and again counteract the increase in speed. These facts can be confirmed by changing the engine offset and by trimming the model at higher speeds where less elevator is required. The NMP zero can thus be made to move to the origin and into the left-half plane.

In general, when the throttle is opened, the extra thrust may produce an increase in speed and/or a gain in altitude, and the phugoid mode is associated with the subsequent interchange of potential and kinetic energy. In this case we see that the positive static loop sensitivity and single NMP zero correspond to a negative dc gain. Therefore, when the throttle is opened, a very lightly damped phugoid oscillation will be initiated, starting with an increase in speed but with a mean value corresponding to a lower speed. The increased thrust will therefore be converted to an increase in altitude. This can be confirmed with a time-history simulation by applying a step throttle input to the linear model from which transfer function (1) was obtained.

Now we consider a more accurate transfer function model of the aircraft. If the aircraft altitude state is included in the A-matrix, it is found that because of the atmosphere model, there are small coupling terms from altitude to several other states. The transfer function corresponding to (1) then becomes

$$\frac{V_T}{\delta_t} = \frac{9.968(s - 0.01506)(s - 0.04528)(s + 0.6066 \pm j0.8814)}{(s + 3.305E - 5)(s + 0.5905 \pm j0.8813)(s + 6.788E - 5 \pm j0.1588)} \quad (3)$$

The very slow altitude pole (at $s = -3E - 5$) has now appeared in the transfer function. An additional NMP zero is also present, and the dc gain of the transfer function is now positive. The physical explanation is that since the decrease in atmospheric

density with altitude is now modeled, the tendency to gain altitude is reduced and the speed will now increase in response to a throttle increase.

Simulation results (see Problem 3.8-2) show that for the linear model without the altitude state the average airspeed (averaged over the phugoid period) decreases in response to a throttle step. When the altitude state is included, the average airspeed decreases at first and then increases. The altitude increases in either case. The response of the nonlinear model with a relatively small throttle step increment (10% increase) agrees closely with the linear model with altitude included. ■

3.9 FEEDBACK CONTROL

Introduction

A major portion of this book is concerned with performing feedback control design on aircraft dynamics, and in this section we review the relevant classical control theory and design techniques.

In the context of controlling dynamic systems, feedback is defined as returning to the input of a system a signal obtained from its output, as shown in Figure 3.9-1. If the signal is fed back with the intention of canceling the effect of an input that produced it, we have negative feedback, as indicated by the minus sign at the *summing junction* in the figure. The system to be controlled is called the *plant*, denoted by G_p , and the feedback connection forms a *closed loop* around the plant. Components that are added to make the feedback control work effectively are represented by the *compensator* G_c . The block labeled H may represent additional compensation and/or a measurement transducer; thus the output of H may be the electrical analog of the physical *output variable* y . The *command input* r may be a different physical variable (e.g., r may be the pilot's control stick force or deflection and y may be gs) and the *command prefilter* H_r will have a conversion factor such that the same kinds of physical quantity are compared at the summing junction.

When r and y are the same physical quantities, they can still have different scale factors, for example, a system may be designed to make $y = 10r$. A general definition of *control error*, e , is

$$e = r - y \quad (3.9-1)$$

Very often H_r and H will have different dynamics but the same low-frequency gain. Then the control error is independent of input amplitude and is a more practical

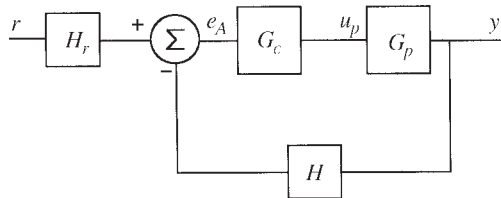


Figure 3.9-1 Feedback control: single-loop configuration.

measure of performance. In addition, when the dynamic effects in H and H_r lie outside the frequency range of interest for the plant and compensator, they can be replaced by identity transformations. If H is replaced by the identity operation, the system is said to have *unity feedback*. The output of the summing junction is the “actuating” signal e_A , and in a well-designed control system it will normally be very small compared to the summed inputs. When H and H_r can be replaced by identity operations, e_A is identical to the control error.

One of the reasons for using negative feedback is to “regulate” the output of the plant, that is, to hold the output constant at a “set point,” as in an aircraft “altitude-hold” autopilot. Another reason is to make the output “track” (i.e., follow) a changing command, as in making an aircraft track the pilot’s pitch-rate command. In these *regulator* and *tracker* applications, negative feedback can change the nature of the plant behavior, for example, the plant may exhibit integrating behavior (e.g., as in steering a vehicle) whereas the closed-loop system simply follows its commands. Also, negative feedback can stabilize an unstable plant (e.g., the X-29 and F-117 aircraft), improve an unsatisfactory system response, allow us to trade gain for increased bandwidth and better linearity, and reduce the effects of extraneous inputs (i.e., “disturbances”). The closed-loop transient response can be made to have dominant modes that are known to provide acceptable *handling qualities* for a human operator (from back-hoe operator to aircraft pilot) or even make one aircraft simulate another (e.g., Gulfstream trainer for space shuttle pilots). To achieve these benefits, it is usually necessary to include some additional compensator dynamics within the feedback loop. In the rest of this section we will see how to choose a suitable type of compensator, how to perform a design with the compensator in place, and how to evaluate the control system design.

Feedback Configurations and Closed-Loop Equations

We will analyze feedback configurations of the form shown in Figure 3.9-1 and with additional loops as in Figure 3.9-2. In Figure 3.9-2 the dynamics G_c , in the *forward path*, represent a *cascade compensator* (i.e., in “series” with the plant). The “inner-loop” feedback H_i represents *feedback compensation* and may correspond to *rate feedback* from a tachometer or a rate gyro when y is an angular position.

In Figure 3.9-2, the individual dynamics may be described by transfer functions or state equations, and from these we need to be able to construct a dynamic description

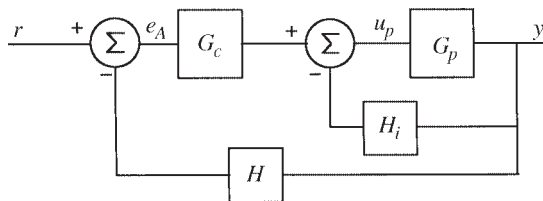


Figure 3.9-2 Feedback control with an inner loop.

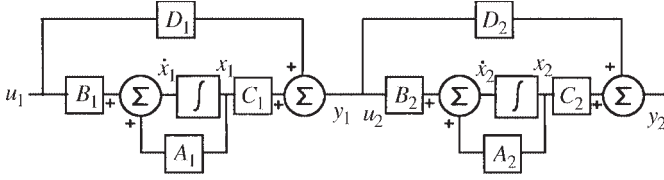


Figure 3.9-3 Cascaded state-space systems.

of the complete system, with input r and output y . To achieve this, we must first reduce the inner loop to a single transfer function or state-space description, cascade the result with the dynamics G_c , and then reduce the outer loop by the same method as the inner loop.

First, consider the basic problem of cascading two dynamic systems. We already know that systems described by transfer functions may be cascaded by multiplying together the transfer functions and, in matrix form, we must ensure compatible dimensions in the matrix multiplication. We will now derive a formula for cascading state-space dynamic equations. In Figure 3.9-3, state-space system “1” is followed by system “2” in cascade. If we include both sets of state variables in one “augmented” state vector, an inspection of the block diagram gives the following equations:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} A_1 & 0 \\ B_2 C_1 & A_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 D_1 \end{bmatrix} u_1 \quad (3.9-2a)$$

$$y_2 = [D_2 C_1 \quad C_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [D_2 D_1] u_1 \quad (3.9-2b)$$

These operations are numerically stable and easy to perform or incorporate in software design tools (e.g., MATLAB “Series” command).

Next, consider a single feedback loop with forward path dynamics G and feedback-path dynamics H (Figure 3.9-1 with $H_r = I$ and $G_c G_p$ combined as G). Let G represent a $(p \times m)$ transfer function matrix and H a $(m \times p)$ transfer function matrix. $H(s)$ may have some null rows or columns, depending on the choice of feedback connections. Simple matrix algebra gives the relationships

$$E_A(s) = R - HGE_A \quad (3.9-3a)$$

$$Y(s) = GE_A = G(I + HG)^{-1}R(s) \quad (3.9-3b)$$

so the closed-loop transfer function matrix is

$$G_{CL}(s) = G(I + GH)^{-1} \quad (3.9-4a)$$

and in the SISO case,

$$G_{CL}(s) = \frac{G}{1 + GH}, \quad (3.9-4b)$$

where $G(s)H(s)$ is called the *loop transfer function*. For frequency ranges over which the SISO loop transfer function has large magnitude, the closed-loop transfer function is given by

$$G_{CL}(s) \approx \frac{1}{H} \quad (3.9-4c)$$

and this property is used to provide a precisely defined transfer function when G is large in magnitude but not well defined (e.g., operational amplifiers).

In the state-space case, if we draw diagrams of the form used in Figure 3.9-3 for both G and H , it is easy to derive the following closed-loop state equations:

$$\begin{bmatrix} \dot{x}_G \\ \dot{x}_H \end{bmatrix} = \begin{bmatrix} A_G - B_G D_H C_G & -B_G C_H \\ B_H C_G & A_H \end{bmatrix} \begin{bmatrix} x_G \\ x_H \end{bmatrix} + \begin{bmatrix} B_G \\ 0 \end{bmatrix} r \quad (3.9-5a)$$

$$y = [C_G \quad 0] \begin{bmatrix} x_G \\ x_H \end{bmatrix} + [0] u \quad (3.9-5b)$$

Here the subscripts G and H indicate, respectively, the forward-path and feedback-path dynamics. We have also taken the special case of $D_G \equiv 0$ since the plant dynamics are normally low pass. Again, these operations are very simple numerically and are preferable to working with polynomials (the MATLAB command “feedback” will perform these operations).

The forward-path dynamics may include a cascade compensator and a plant and, in the special case where the feedback path contains only a gain matrix K , the above equations reduce to

$$\dot{x} = (A_G - B_G K C_G)x + B_G r \quad (3.9-6a)$$

$$y = C_G x \quad (3.9-6b)$$

The closed-loop A -matrix A_{CL} is given by

$$A_{CL} = (A_G - B_G K C_G), \quad (3.9-6c)$$

and the other matrices are unchanged.

Design software is usually arranged to produce a value for the feedback gain K . If we wish to obtain a unity-feedback design, we must scale the command r by the same gain as the feedback (i.e., K). The closed-loop system is then equivalent to a unity-feedback system with a gain K in the error path; this is illustrated in Figure 3.9-4. The closed-loop equations, with $D \equiv 0$, are easily seen to be

$$\dot{x} = (A - BKC)x + BKr \quad (3.9-7a)$$

$$y = Cx \quad (3.9-7b)$$

Therefore, we can design unity-feedback systems with the same software as long as we finally postmultiply the original B -matrix by K .

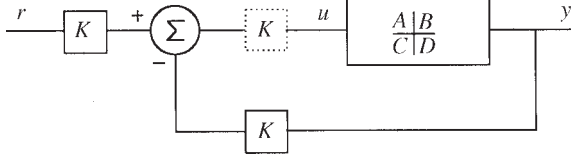


Figure 3.9-4 Transforming to unity feedback.

In Chapter 4 we will perform a classical control design on a MIMO plant, namely, the aircraft lateral-directional dynamics. These have two inputs, aileron and rudder deflections, and two or more outputs. The classical design approach is iterative, closing one loop at a time and then repeating the process until the design is satisfactory. Here we will note how the loops can interact. The closed-loop MIMO transfer function is given in Equation (3.9-4a) and, closing any individual SISO loop through GH , in general, changes both the poles and zeros of any other SISO transfer function in G_{CL} . Various types of zeros are defined for a transfer function matrix (MacFarlane and Karcianias, 1976; Desoer and Schulman, 1974), and different subsets of these zeros will appear in the various SISO transfer functions. The zeros of a particular element of the transfer function matrix will not all appear in the corresponding SISO transfer function because of pole-zero cancellations. The actual SISO zeros can be found by the method of “coupling numerators” (McRuer et al., 1973).

Now let us return to the SISO case, with the closed-loop transfer function (3.9-4b). Also, let G and H be represented by monic polynomials as

$$G(s) = k_G \frac{N_G(s)}{D_G(s)}, \quad H(s) = k_H \frac{N_H(s)}{D_H(s)}$$

Then

$$\frac{Y(s)}{R(s)} = \frac{G}{1 + GH} = \frac{k_G N_G D_H}{D_G D_H + k_G k_H N_G N_H} \quad (3.9-8)$$

and $K = k_G k_H$ is the static loop sensitivity of the loop transfer function $G(s)H(s)$. We will often simply refer to K as the loop gain. Equation (3.9-8) shows that the closed-loop zeros are the combined zeros of G and poles of H . The closed-loop poles are given by the zeros of the *characteristic equation*

$$1 + G(s)H(s) = 0, \quad (3.9-9a)$$

that is, the roots of the *characteristic polynomial*

$$D_G D_H + K N_G N_H \quad (3.9-9b)$$

Therefore, the positions of the closed-loop poles vary with K . The frequency response of the loop transfer function GH and the behavior, with K , of the roots of the characteristic polynomial are the basis of the two common design techniques of classical control theory. These will be described shortly.

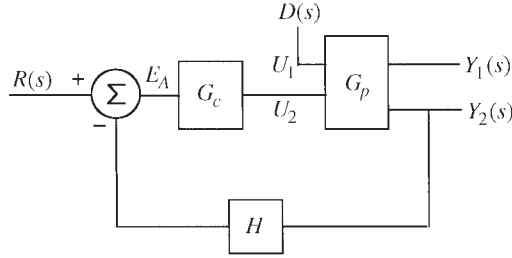


Figure 3.9-5 Disturbance input, transfer function description.

This subsection has examined the closed-loop equations in various forms. Our last development will be the closed-loop equations for signals, other than the command, injected into the loop. An example is an aircraft model with control inputs applied to the control surfaces and wind gusts entering into the aerodynamic calculations as disturbances. In general, a MIMO plant model will be required with inputs for both disturbances and controls. Figure 3.9-5 shows an example where the plant G_p is treated as a two-input, two-output system. One input-output pair is used by the feedback loop (e.g., for feedback of angle of attack to the elevator), and the other pair has been created to determine the effect of the disturbance on some other output variable (e.g., effect of wind gusts on normal acceleration).

Referring to Figure 3.9-5, the transfer function equations are

$$Y(s) = G_p \begin{bmatrix} 1 \\ 0 \end{bmatrix} D(s) + G_p \begin{bmatrix} 0 \\ 1 \end{bmatrix} (G_c R(s) - [0 \quad G_c H] Y(s))$$

or, solving for $Y(s)$,

$$Y(s) = \left(I + G_p \begin{bmatrix} 0 & 0 \\ 0 & G_c H \end{bmatrix} \right)^{-1} G_p \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} D + \begin{bmatrix} 1 \\ G_c \end{bmatrix} R \right) \quad (3.9-10)$$

Control inputs are multiplied by G_c , and higher gain in G_c will give better rejection of disturbance inputs. Equation (3.9-10) requires an inversion of a polynomial matrix and is inconvenient for numerical computations.

Figure 3.9-6 shows a state-space description of a compensator cascaded with a plant and a disturbance input to the plant. An analysis of the diagram is greatly simplified by the absence of a direct feedthrough path in the compensator, and this matches the aircraft situation if the control surface actuators are included with the compensator model. By inspection, the state equations are

$$\begin{bmatrix} \dot{x}_p \\ \dot{x}_c \end{bmatrix} = \begin{bmatrix} A_p & B_p C_c \\ -B_c C_p & A_c - B_c D_p C_c \end{bmatrix} \begin{bmatrix} x_p \\ x_c \end{bmatrix} + \begin{bmatrix} B_p \\ -B_c D_p \end{bmatrix} d + \begin{bmatrix} 0 \\ B_c \end{bmatrix} r \quad (3.9-11a)$$

$$y = [C_p \quad D_p C_c] \begin{bmatrix} x_p \\ x_c \end{bmatrix} + [D_p] d \quad (3.9-11b)$$

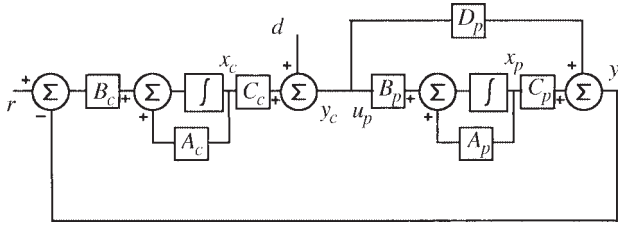


Figure 3.9-6 Disturbance input, state-space description.

These equations can be used to determine the transfer function (or poles and zeros or frequency response) from the disturbance input to the response variable of interest. Aircraft in flight are subjected to random disturbances from discrete wind gusts and continuous turbulence, and these types of disturbances are modeled with a *power spectral density function* (PSDF) (see MIL-F-8785C). A measure of a stationary (statistics independent of time) random signal is its *root-mean-square* (rms) value, and this can be calculated from the area under the PSDF (Brown and Hwang, 1997). The aircraft response PSDF must be calculated from the squared magnitude of the appropriate transfer function multiplied by the PSDF of the disturbance. Alternatively, a model of the disturbance can be included in the nonlinear state equations, and the response to the disturbance can be found by simulation.

Steady-State Error and System Type

An analysis of Figure 3.9-1 with SISO transfer functions G and H in the forward and feedback paths (with $H_r = I$ and $G = G_c G_p$) using the error definition (3.9-1) gives the error transfer function

$$\frac{E(s)}{R(s)} = \frac{1 + G(H - 1)}{1 + GH} \quad (3.9-12a)$$

With unity feedback, this reduces to

$$\frac{E(s)}{R(s)} = \frac{1}{1 + G} \quad (3.9-12b)$$

First consider the unity-feedback case and think of a regulator where “steady state” implies a constant output. If the forward-path transfer function exhibits integrating action, then a steady-state error cannot exist. Electronic operational amplifier circuits can provide almost perfect integration and so, if necessary, we may decide to include an integrator in the error path. In the case of a tracker, the steady-state error will depend on the type of command input and so we must analyze this situation.

In (3.9-12b) let there be q pure integrations in G , after any cancellations with zeros at the origin, and let $G = s^{-q} G'$. Then,

$$\frac{E(s)}{R(s)} = \frac{1}{1 + G} = \frac{s^q}{s^q + G'}, \quad (3.9-13)$$

where $G'(0)$ is finite. Let the system be stable and have a polynomial input $(t^n/n!)$ $U_{-1}(t)$ with Laplace transform $1/s^{n+1}$. Then the final-value theorem gives the following expression for the steady-state control error:

$$e_{ss}(t) = \lim_{s \rightarrow 0} sE(s) = \lim_{s \rightarrow 0} \frac{s^{q-n}}{(s^q + G')} \quad (3.9-14)$$

Therefore, in order to track a polynomial input of degree n , with finite steady-state error, the control system must have n pure integrations in the forward path. Such a system is called a *type- n* control system. An additional integration will reduce the steady-state error to zero, while one less integration will cause the steady-state error to grow without bound. Each integration adds 90° of phase lag and makes the control system progressively more difficult to stabilize, so that systems are restricted to type 2 or lower in practice. The value of the finite steady-state error is

$$\begin{aligned} \text{when } q = n = 0 \text{ (step input),} \quad e_{ss} &= \frac{1}{(1 + G(0))} \\ \text{when } q = n \geq 1 \text{ (ramp, parabola, etc.),} \quad e_{ss} &= 1/G'(0) \end{aligned} \quad (3.9-15)$$

The step, ramp, and parabolic *error coefficients* are defined as

$$\begin{aligned} K_p &\equiv \lim_{s \rightarrow 0} G(s) \text{ and error to a unit step} = \frac{1}{(1 + k_p)} \\ K_v &\equiv \lim_{s \rightarrow 0} sG(s) \text{ and error to a unit ramp} = \frac{1}{K_v} \\ K_a &\equiv \lim_{s \rightarrow 0} s^2 G(s) \text{ and error to a unit parabola} = \frac{1}{K_a} \end{aligned} \quad (3.9-16)$$

and are used to specify performance requirements. Now consider the nonunity-feedback case and, as discussed in connection with (3.9-1), let $H(0) = 1$. The formula corresponding to (3.9-14) is

$$e_{ss}(t) = \lim_{s \rightarrow 0} \frac{s^{q-n} + G'(H-1)/s^n}{s^q + G'H} \quad (3.9-17)$$

and for comparison with the unity-feedback case, let $q = n$. Then, if $q = n = 0$,

$$e_{ss} = \left[\frac{1 + G(H-1)}{(1 + GH)} \right]_{s=0} = \frac{1}{1 + G(0)} \quad (3.9-18a)$$

and if $q = n > 0$,

$$e_{ss} = \lim_{s \rightarrow 0} \left[\frac{1 + G'(H-1)/s^n}{G'H} \right] \quad (3.9-18b)$$

Because of the condition $H(0) = 1$, the steady-state error with a constant input, Equation (3.9-18a), is the same as the unity-feedback case. With polynomial

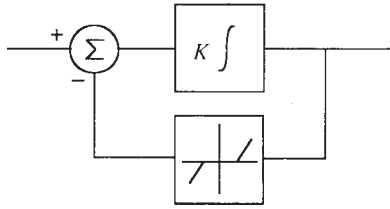


Figure 3.9-7 Integrator windup protection.

inputs, Equation (3.9-18b) shows that the steady-state error depends on the limit of $(H - 1)/s^n$ as s becomes zero. If $H(s)$ is written as a ratio of polynomials, then $H(0) = 1$ guarantees that $H - 1$ has at least one free s to cancel with s^n . Therefore, unlike the unity-feedback case, the error can become infinite with a parabolic input ($n = 2$). This is illustrated in Example 3.9-6.

Practical command inputs may contain derivatives of all orders for short periods of time, so that the tracking error may grow and then decrease again. System-type requirements and error coefficients are preliminary design considerations, and these ideas are used later in the design examples and in Chapter 4.

System type can be misleading in nonlinear situations. If, for example, the plant includes an electric motor, the integration of motor speed to angular position is a “kinematic” integration. If there are no other integrations in the forward path, the error signal must become large enough to overcome the static frictional torques of the motor and load before the motor will begin to turn. Therefore, this system will not behave like a type-1 system.

Another problem encountered with integral control of real systems is *integrator windup*. An electronic integrator *saturates* when its output gets close to the circuit positive or negative supply voltages. If the plant becomes temporarily nonlinear (e.g., “rate saturation”) before saturation occurs in the integrator, then depending on the command signal, the integrator may begin to integrate a large error signal that takes its output farther beyond the plant saturation level. When the plant comes out of saturation or the command reverses, it may take some significant time before this excessive output is removed and linear control is regained. Figure 3.9-7 shows an anti-windup arrangement. When the output of the integrator reaches the plant saturation value, it exceeds the threshold of the dead-zone device. The resulting feedback turns the integrator into a fast lag transfer function. Anti-windup arrangements are used in both analog and digital aircraft flight control systems. A related problem can occur when switching between different control system modes. All energy storage elements must be initialized so that unwanted sudden movements of the control surfaces do not occur.

Stability

A familiar example of feedback causing instability is provided by the public address (PA) system of an auditorium. When an acoustical signal from the loudspeakers is

received at the microphone and the gain and phase around the acoustical path are such that the signal reinforces itself, the loudspeakers produce a loud whistle. This is probably the most natural intuitive way to understand feedback stability. Thus, we might examine the frequency response of the loop transfer function GH to determine if the gain is greater than unity when the phase lag has reached 180° ; that is, we must look for the condition $GH(j\omega) = -1$. This corresponds to finding a root of $(1 + GH) = 0$ on the s -plane $j\omega$ -axis, which is the stability boundary.

In 1932 H. Nyquist used the *principle of the argument* from complex variable theory (Phillips, 1961), applied to $F(s) = 1 + GH(s)$, to develop a test for stability. A semicircular “test” contour of “infinite” radius is used to enclose the right-half s -plane. According to the principle of argument, as s traverses the closed test contour in a clockwise direction, the increment in the argument of $F(s)$ is $N \times (2\pi)$, where $N = (P - Z)$, and P and Z are, respectively, the number of poles and zeros of $F(s)$ inside the test contour. We see that poles and zeros of $F(s) = 1 + GH(s)$ are, respectively, the open-loop and closed-loop poles, and N is the number of counterclockwise encirclements of the s -plane origin.

Rather than count the encirclements of the origin by $1 + GH$, we can, more conveniently, count the encirclements of the *critical point* $(-1 + j0)$ by $GH(s)$. In general, P and Z are both greater than or equal to zero, and so N may be a positive or a negative integer. Since the test contour encloses the whole right-half s -plane, we have a closed-loop stability test by finding Z , given by

$$Z = P - N \quad (3.9-19)$$

or

$$\# \text{ unstable } CL \text{ poles} = \# \text{ unstable } OL \text{ poles} - \# CCW \text{ encirclements}$$

The test contour, known as the *Nyquist D-contour*, can be indented with infinitesimal semicircles to exclude open-loop poles on the $j\omega$ -axis. Note that some authors define N to be the number of clockwise encirclements, and they reverse the sign of N in (3.9-19).

Example 3.9-1: An Example of Nyquist’s Stability Criterion Let the open-loop transfer function be given by

$$G(s)H(s) = \frac{K(s+2)(s+4)}{s(s^2 - 4s + 13)}, \quad k > 0$$

Figure 3.9-8a shows the Nyquist D-contour, indented with a semicircle to avoid the pole at the origin. Figure 3.9-8b shows the Nyquist plot; letters have been used to mark corresponding points on the two plots. The indentation can be represented by the equation $s = re^{j\theta}$, with $r \rightarrow 0$ and $-\pi/2 \leq \theta \leq \pi/2$, as an aid to establishing the corresponding points. Imagine $G(s)H(s)$ represented by vectors drawn from the poles and zeros to a starting point at a on the D-contour. When $s = a$, the net angle of the vectors is zero and the magnitude of $GH(a)$ approaches infinity as r becomes zero; this gives the corresponding point a' on the GH plot. When $s = b$, the angle of the

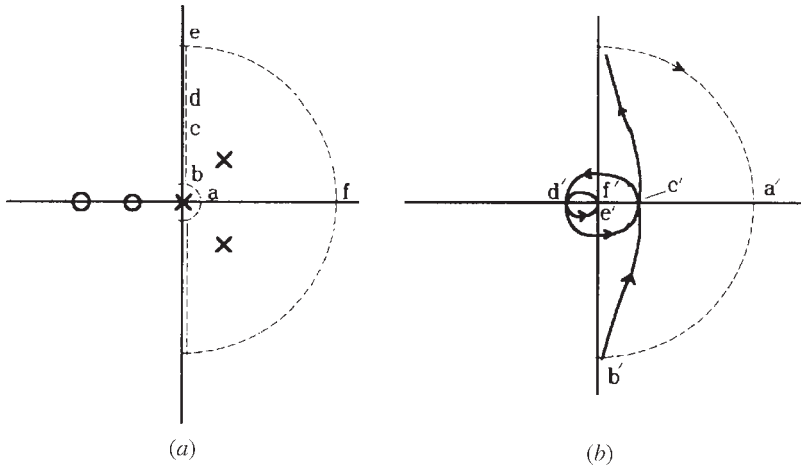


Figure 3.9-8 (a) A Nyquist D-contour; (b) a Nyquist plot.

vector from the pole at the origin has become 90° , but the net angle of the other vectors is close to zero; this gives the point b' . The part of the D-contour from b to e corresponds to real frequencies, and the frequency response $GH(j\omega)$ could be measured with test equipment if the system were not unstable. The relative degree is unity so, as ω increases, $GH(j\omega)$ approaches zero magnitude with a phase angle of -90° (or $+270^\circ$). Let $s = c$ and d be the points where the phase of GH passes through zero and 180° , respectively. From $s = e$ to $s = f$, the phase of $GH(s)$ returns to zero, while the magnitude remains infinitesimal. The remainder of the D-contour uses conjugate values of s , and the remaining half of the Nyquist plot is the conjugate of the part already drawn.

The D-contour shows that $P = 2$, and the Nyquist plot shows that the number of counterclockwise encirclements of the critical point is $N = 0$ or $N = 2$, depending on the magnitude of $GH(j\omega)$ at d' . This, in turn, depends on the loop gain K , and so

$$\text{small } K \rightarrow Z = 2 - 0 = 2 \quad (\text{closed-loop unstable})$$

$$\text{large } K \rightarrow Z = 2 - 2 = 0 \quad (\text{closed-loop stable})$$

This is the opposite of the common behavior, in which a system becomes unstable when the gain is increased too much, and this behavior is known as *conditional stability*. At $s = c$ and d' , $\text{Im}[GH(j\omega)] = 0$; then solving $\text{Re}[GH(j\omega)] = -1$, evaluated at the higher value of ω , gives the value of K at the stability boundary. A Nyquist plot can be obtained with the following MATLAB code:

```
num=[1 6 8]; den=[1 -4 13 0];
w= logspace(-1,1,400);
k= 6;
nyquist(k*num,den,2*pi*w)
```

% 0.1 Hz to 10 Hz, 400 points
% Stable k

Most practical control systems are open-loop stable, so that $Z = -N$, and therefore we require $N = 0$ for stability. Also, we need only consider the positive $j\omega$ -axis of the D-contour, since the negative $j\omega$ -axis gives a conjugate locus in the GH -plane, and the infinite semicircle maps to the origin of the GH -plane (because the relative degree of the transfer function of a real compensator and plant is greater than zero). A few rough sketches will show that, under these conditions, *if the locus of GH is plotted as the frequency is varied from $\omega = 0$ to $\omega = \infty$, the closed-loop system is unstable if the critical point lies to the right of the locus*. An example of the Nyquist plot of a stable type-1, relative-degree-3 system is shown in Figure 3.9-13. These restricted conditions for stability agree with the intuitive criterion that the magnitude of the loop transfer function should be less than unity when its phase lag is 180° .

Stability criteria, other than Nyquist's test, mostly involve testing the characteristic equation (CE) directly for roots in the right-half s -plane. A necessary but not sufficient condition for stability is that all of the coefficients of the CE should have the same sign and be nonzero [see also Descartes' rule of signs (D'Azzo and Houpis, 1988)]. Routh's test (Dorf and Bishop, 2001) uses the coefficients of the CE and provides more information, in that the number of right-half-plane roots and the stability boundary can be determined. In the state-space context the roots of the CE are the eigenvalues of the A -matrix, but then we must go to the trouble of solving the characteristic equation.

Types of Compensation

The discussion of stability and some Nyquist sketches for simple systems that are open-loop stable should lead to some ideas about the frequency-domain properties required of a compensator. Alternatively, we might look at a compensator as a means of adding extra terms to the system characteristic equation so that the roots can be moved to desirable locations in the left-half s -plane. In frequency-domain terms, a compensator should produce phase lead in a frequency range where the lag of the plant is approaching 180° and the gain is near unity or it should cut the gain when the phase lag is approaching 180° . For a minimum-phase transfer function phase lead is associated with rising gain, and this approximates the characteristics of a differentiator. Differentiation accentuates the noise on a signal, and so practical compensators should be designed to produce phase lead and rising gain only over a limited frequency range.

Let us now examine a compensator with a single differentiation, a proportional-plus-derivative (PD) compensator, which can be approximated in real systems. The transfer function is

$$G_c(s) = K_P + K_D s \quad (3.9-20)$$

Equation (3.9-4b) gives the unity-feedback closed-loop transfer function as

$$\frac{Y}{R} = \frac{(K_P + K_D s)G_p}{1 + (K_P + K_D s)G_p} \quad (3.9-21)$$

The characteristic equation now contains the proportional and derivative terms K_p and K_D , and it may be possible to achieve satisfactory closed-loop poles. However, in addition to the noise problem, there is now a closed-loop zero at $s = -K_p/K_D$, and this zero can cause a large overshoot in the step response unless the plant poles are heavily damped.

As an alternative to PD compensation, consider Figure 3.9-2 with unity feedback, simple proportional control, and inner-loop rate feedback:

$$G_c = K_p; \quad H_i = K_r s$$

Then, using (3.9-4b) to close the inner loop first, the overall closed-loop transfer function is found to be

$$\frac{Y}{R} = \frac{K_p G_p}{1 + (K_p + K_r s) G_p} \quad (3.9-22)$$

Therefore, with rate feedback, we can achieve the same closed-loop poles as PD control, but without accentuating noise in the error channel and without the troublesome closed-loop zero.

A practical cascade compensator that only approximates PD control and satisfies the practical requirement of relative degree greater than or equal to zero is the simple “phase-lead” compensator shown in Table 3.3-1. The numerator $(s + z)$, on its own, represents a derivative term plus a proportional term, which is equivalent to a zero at $s = -z$. The pole is at $s = -p$, with $p > z$, and if we were to compare the Bode plots of $(s/z + 1)$ and $(s/z + 1)/(s/p + 1)$ we would see that the derivative action begins to disappear as the second corner frequency $\omega = p$ is approached. The practical limit $(p/z) < 10$ is usually observed to avoid greatly accentuating noise.

Phase-lead compensation is effective and inexpensive; inner-loop rate feedback incurs the cost of a rate sensor and may not be physically appropriate for a particular plant. A practical rate sensor also has limited bandwidth, and its transfer function pole(s) will appear as closed-loop zeros. However, these zeros are likely to be much farther from the s -plane origin than the lead compensator zero and therefore less troublesome in terms of causing overshoot.

The subsection on steady-state error and system type explained the need for “integral control.” Unfortunately, “pure” integral control has some detrimental effects on closed-loop transient response. First, a pole at the s -plane origin is destabilizing because it adds a constant 90° phase lag to the loop transfer function. Second, an open-loop pole at the origin may become a slow closed-loop pole (see the root-locus section). To overcome the phase-lag problem we use “proportional plus integral” (PI) control in the cascade compensator. The compensator transfer function is $(k_p + k_i/s)$ or, equivalently, $k_p(s + k_i/k_p)/s$. The Bode plot of this transfer function shows that the phase lag disappears at high frequency.

If we use Figure 3.9-1 with unity feedback, $H_r = 1$, and a cascaded PI compensator, the closed-loop transfer function is

$$\frac{Y}{R} = \frac{(sK_p + K_i)G_p}{s + (sK_p + K_i)G_p} \quad (3.9-23)$$

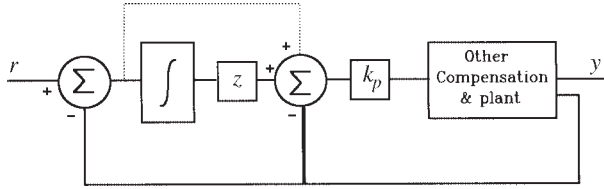


Figure 3.9-9 PI compensation with no closed-loop zero.

The PI control has introduced a closed-loop zero at $s = -K_i/K_p$, and again this may cause an excessive overshoot in the step response. To see what inner-loop feedback can do for us, in Figure 3.9-2 let

$$G_c = K_i/s; \quad H_i = K_f; \quad H = 1$$

so that the closed-loop transfer function becomes

$$\frac{Y}{R} = \frac{K_i G_p}{s + (sK_f + K_i)G_p} \quad (3.9-24)$$

The inner-loop proportional feedback, combined with pure integral control, has the same characteristic equation as PI control but has eliminated the closed-loop PI zero. Another way of looking at this is shown in Figure 3.9-9. The signal fed back to the plant input is unchanged if the PI proportional path (lightly dotted line) is removed and the feedback path shown with the heavy line is added. The overall closed-loop transfer function has changed because the input signal, r , no longer sees a proportional path. We can also see that the inner-loop feedback would remove the effect of an integration in the plant, and so this modification may reduce the system type. Proportional-plus-derivative control can be interpreted in a similar manner.

A lag compensator is also shown in Table 3.3-1; it has a pole and a zero, with the pole closer to the origin. If the pole is placed very close to the origin, it can be thought of as an approximation to PI compensation, although we usually choose the zero position in a different way from PI compensation. By using the s -plane vector interpretation of the lag compensator or drawing its Bode plots, we see that it provides a reduction in gain at high frequency (hf) without the 90° asymptotic phase lag. It can be thought of as a way of alleviating stability problems caused by phase lag at hf or of boosting low-frequency (lf) gain relative to hf gain in order to improve the position error coefficient in a type-0 system.

The compensators described above may be used in combination and, for example, two stages of phase-lead compensation can provide more lead than a single stage for the same increase in gain.

SISO Root-Locus Design

In this subsection we introduce our first classical design technique: *root-locus* design, devised by W. R. Evans in 1948. The root-locus technique provides a

graphical method of plotting the locii of the roots of a polynomial, in the complex plane, when a coefficient in the polynomial is varied. It can be applied directly to the characteristic equation of a closed-loop control system to determine when any poles become lightly damped or unstable and to determine the effects of adding compensator poles and zeros.

Consider the following polynomial equation in the complex variable s ,

$$s^n + a_{n-1}s^{n-1} + \cdots + a_js^j + \cdots + a_1s + a_0 = 0$$

Suppose that we wish to examine the movement of the roots when the coefficient a_j is varied. The root-locus rules of construction can be applied by writing the equation as

$$1 + \frac{a_js}{(s^n + \cdots + a_0) - a_js} = 0 \quad (3.9-25)$$

The characteristic equation (3.9-9a) can be written in this form as

$$1 + \frac{K N(s)}{D(s)} = 0, \quad (3.9-26)$$

where the monic polynomials $N(s)$ and $D(s)$ contain, respectively, the known open-loop zeros and poles (n poles and m zeros) and the static loop sensitivity K is to be varied.

Equation (3.9-26) is the equation that is satisfied on the locii of the closed-loop poles, that is, on the “branches” of a *root-locus plot*. It can be rewritten as

$$\frac{K N(s)}{D(s)} = -1 \quad (3.9-27)$$

from which we get the “angle condition”

$$\angle N(s) - \angle D(s) = \begin{cases} (2r+1)\pi, & K > 0 \\ r(2\pi), & K < 0 \end{cases} \quad r = 0, \pm 1, \pm 2, \dots \quad (3.9-28)$$

and the “magnitude condition”

$$|K| = \frac{|D(s)|}{|N(s)|} = \frac{\Pi (\text{lengths of vectors from poles})}{\Pi (\text{lengths of vectors from zeros})} \quad (3.9-29)$$

When there are no zeros, the denominator of (3.9-29) is unity. These two conditions are the basis of most of the root-locus rules, which are now enumerated:

1. Number of branches = number of open-loop poles (n).
2. The root-locus plot is symmetrical about the s -plane real axis.

3. For $K > 0$, sections of the real axis to the left of an odd number of poles and zeros are part of the locus. When K is negative, we have the so-called *zero-angle root locus*, which is on the axis to the left of an even number of poles and zeros.
4. The n branches start (when $K = 0$) at the open-loop poles and end (when $K = \infty$) on the m open-loop zeros, or at infinity (if $n > m$).
5. Branches that go to infinity approach asymptotes given by

$$\angle \text{asymptotes} = \pm \frac{(2r+1)\pi}{(n-m)}, \quad r = 0, \pm 1, \pm 2, \dots$$

$$\text{real-axis intersection of asymptotes} = \frac{\Sigma(\text{finite poles}) - \Sigma(\text{finite zeros})}{(n-m)}$$

6. If two real-axis branches meet as K is increased, they will break away to form a complex pair of poles. Similarly, two complex branches may arrive at the same real-axis point and become a real pair. Break-away and arrival points can be found by solving (3.9-27) for K and then finding the values of s that satisfy $\partial K / \partial s = 0$ with s treated as a real variable.
7. Root-locus branches meet or leave the real axis at 90° .
8. If a “test point” is very close to a complex pole or zero, all of the vectors from the other poles and zeros can be approximated by drawing them to that pole or zero. The angle of the remaining vector, found from the angle condition (3.9-28), gives the angle of departure or arrival of the root-locus branch for the pole or zero in question.
9. Imaginary-axis crossing points can be found by replacing s by $j\omega$ in the characteristic equation and solving the separate real and imaginary conditions that result. Alternatively, the root-locus angle condition can be applied or a standard test for stability (e.g., Routh-Hurwitz) can be used.
10. Constant net damping: When the relative degree $(n-m)$ of the loop transfer function is greater than unity, then, if some branches are moving left, others must be moving right.

Software is available to construct root-locus plots (e.g., MATLAB “rlocus” and “rltool”), but the above rules allow us to anticipate the effects of proposed compensators. We will now illustrate root-locus design by means of some examples.

Example 3.9-2: Root-Locus Design Using a Lead Compensator In this example we will show how a phase-lead compensator can stabilize an unstable system, but the compensator will be chosen to illustrate the root-locus rules rather than to produce the “best” control system design. This example can be done more easily using transfer functions, but we wish to develop familiarity with the state-space approach, for later applications.

Let the plant be type 2 with transfer function

$$G(s) = \frac{100}{s^2(s+10)}$$

In yet another technique for obtaining state equations, the transfer function was expanded as a sum of partial fraction terms, and state variables were chosen to be the integrator outputs in the simulation diagram representation of each partial fraction term, as in Section 3.2. The plant A -, B -, C -, and D -matrices are

$$ap = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -10 \end{bmatrix} \quad bp = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \quad cp = [10 \quad -1 \quad 1] \quad dp = [0]$$

The compensator state-space description is given in Table 3.3-1. Equations (3.9-2) can easily be used to cascade the compensator with the plant, but here we will illustrate the use of the “series” command in a MATLAB program:

```
ap= [ 0 1 0; 0 0 0; 0 0 -10];
bp= [0; 1; 1]; cp= [10 -1 1]; dp= [0];           % Plant
z=.6; p= 9;                                       % Compr. Zero & pole
ac= [-p]; bc= [1]; cc= [z-p]; dc= [1];          % Lead comp.
[a b c d] = series(ac,bc,cc,dc,ap,bp,cp,0);      % Comp. + plant
k= linspace(0,10,2000);
r= rlocus(a,b,c,d,k); plot(r)
grid on
```

The root-locus plot is shown in Figure 3.9-10, with the compensator pole at $s = -9$ and the zero at $s = -0.6$. Without the compensator the two branches from the double

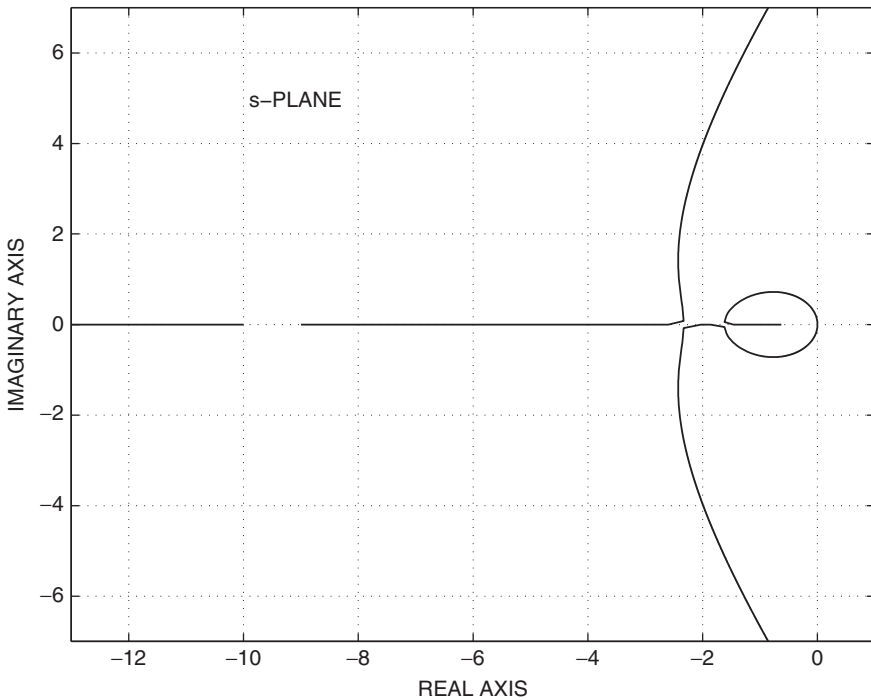


Figure 3.9-10 Lead compensation on the root-locus plot.

pole at the origin would immediately move into the right-half s -plane, while the real pole at $s = -10$ moves left (i.e., constant net damping). The effect of placing the compensator zero near the origin, with its pole well to the left, is strong enough to pull the two branches from the origin into the real axis. The branch that approaches the compensator zero represents a closed-loop pole close to the origin and hence a slow closed-loop mode. The “strength” of this mode (i.e., residue in the pole) will depend on how close the pole gets to the zero, but in a practical design the compensator zero would be placed farther to the left. The other branch from the origin moves left and meets the compensator pole. They break away from the real axis and move toward the right-half plane (i.e., constant net damping again) and approach 60° asymptotes. It is worthwhile to check the root-locus rules, one by one, against this example. All of the rules are illustrated except the “angle-of-departure” rule. ■

This phase-lead example will be repeated as Example 3.9-5, done in the frequency domain, and with more emphasis on practical design considerations. In general, possible root-locus design techniques include placing the compensator zero on or to the left of the second real plant pole from the origin or placing it at the real part of a desired complex pair. The compensator pole position may then be adjusted to give a closed-loop dominant pair a desired frequency or damping. The closed-loop step response should be checked and the design may be modified by moving the pole position or by moving both the pole and zero keeping the ratio p/z constant.

PI compensator design will be illustrated next by the following root-locus example.

Example 3.9-3: Root-Locus Design of a PI Compensator Let the plant and PI compensator transfer functions be

$$G_p = \frac{1}{(s+3)(s+6)} \quad G_c = \frac{K(s+z)}{s}$$

The design goals will be to obtain a dominant complex pole pair with damping ratio of $1/\sqrt{2}$ together with the highest possible ramp error coefficient. The root-locus plot will show the trade-offs in the design, and a simulation will be used to check that the closed-loop step response is like that of a quadratic lag with $\zeta = 1/\sqrt{2}$. A MATLAB program is:

```

z = 2; num= [1 z]; den= [1 9 18 0]; % Choose z
[a,b,c,d]= tf2ss(num,den); % Compr. + Plant
k= linspace(0,50,2000);
r= rlocus(a,b,c,d,k); plot(r), grid on % Root locus

sgrid(.707,0)
axis([-8,1,-8,8])
rlocfind(a,b,c,d) % Find K for zeta=.707

sys1= ss(a,18*b,c,d); % K=18
sys2= feedback(sys1,1,-1); % Close loop
step(sys2,3) % Step response
grid on

```

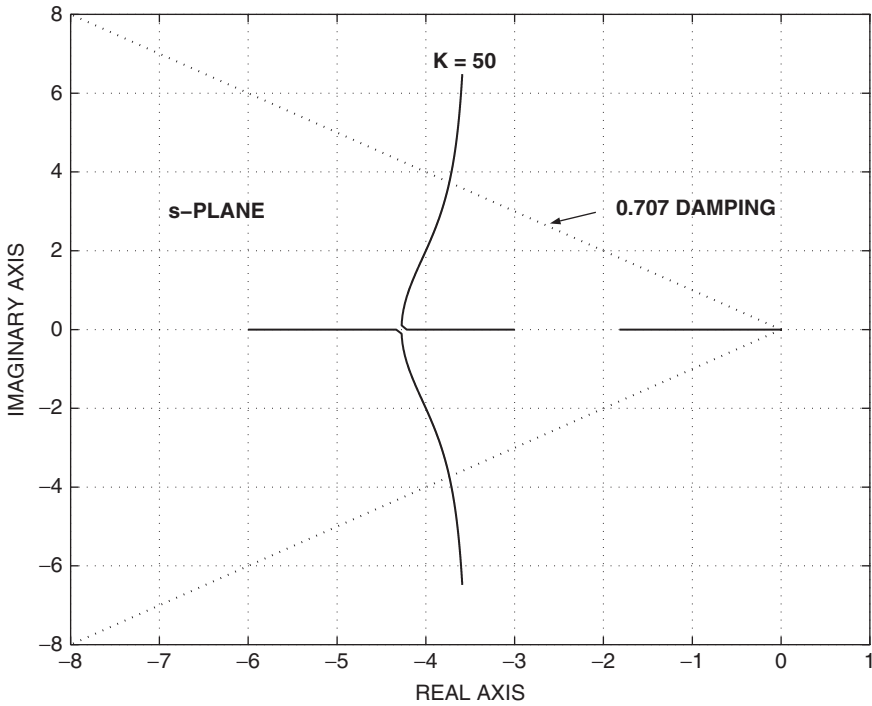


Figure 3.9-11 PI compensation on the root-locus plot.

Figure 3.9-11 is the root-locus plot with $z = 2$. The relative degree of the loop transfer function is 2, and so the asymptotes are at 90° to the real axis. The damping of the complex poles can become very small, but the system can never become unstable. The ramp error coefficient is

$$K_v = \lim_{s \rightarrow 0} sG(s) = \frac{Kz}{(3)(6)}$$

If we make z small, the error coefficient will be small. In addition, the root-locus plot shows that there will be a slow closed-loop pole trapped near the origin. If we place the PI zero to the left of the plant pole at $s = -3$, the complex poles will break away from the axis between $s = 0$ and $s = -3$. This could produce a dominant pair of poles, but they may be too close to the origin for a fast, well-damped response. Therefore, we might try $1 < z < 4$ while adjusting K to give a damping ratio $\zeta = 0.707$ and checking K_v . When this is done, K_v is found to peak when $z = 3$ and $K = 18$. The zero then cancels the slowest plant pole and the closed-loop dynamics are second order with the desired damping ratio. The step response is shown in Figure 3.9-12. In general, the best position for the zero should be determined on a case-by-case basis using considerations similar to those above.

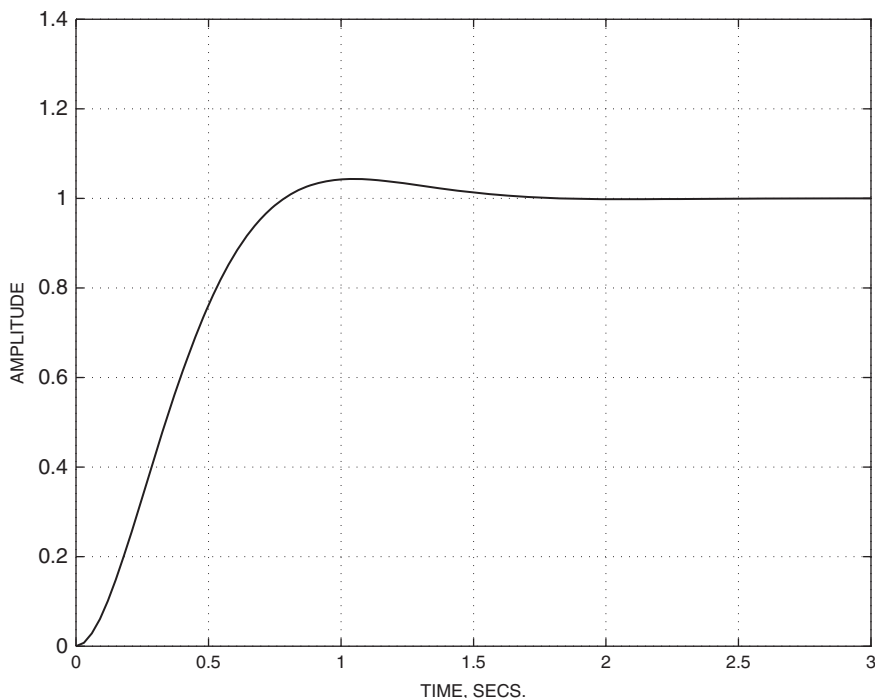


Figure 3.9-12 Step response with PI compensation. ■

A lag compensator (see Table 3.3-1) can be used to increase the value of a control system error coefficient (K_p, K_v, K_a, \dots), without appreciably affecting stability. The lag compensator pole can be placed close to the origin and the lag compensator zero placed not far to the left of the pole. At low frequencies, the compensator gain is given by the length of the zero vector divided by that of the pole vector. Time constants up to about 100 s are practicable, so the pole could be placed at $s = -0.01$. Then, placing the zero at $s = -0.1$ will give a low-frequency gain of 10.0. At high frequency these two pole and zero vectors are close together and have little effect on the dynamics. This technique traps a slow pole near the origin, as was noted in Example 3.9-2. Note that to get greater than unity low-frequency compensator gain, an amplifier is required. An alternative approach to lag compensation is to increase the loop gain as much as possible and solve problems of high-frequency instability by using the lag compensator to cut the high-frequency gain without adding much phase lag. The term *lag compensator* is unfortunate in that, unlike a simple lag transfer function, its ultimate phase lag is zero. This technique is useful in situations where “unmodeled” high-frequency dynamics are causing instability, and trial-and-error compensation is used. Lag compensation is better illustrated in the frequency domain than on the root-locus plot.

In summary, the root-locus technique works well with low-order dynamics and is especially useful as a “back-of-the-envelope” analysis or design technique.

With a large number of poles and zeros it becomes necessary to switch to the frequency-domain techniques illustrated in the next subsection.

Frequency-Domain Design

In frequency-domain design we plot the frequency response of the loop transfer function and use ideas related to Nyquist, Bode, and Nichols plots (Franklin et al., 2002) to arrive at appropriate parameters for one of the standard compensator transfer functions.

The Nyquist stability test leads to useful analysis and design ideas. Some control loops contain pure delay effects, for example, signal propagation delays in a transmission medium or “transport delays” due to piping, belt-feed devices, and so on. In the aircraft case, we have computational delays in a flight control computer and decision and reaction time delays in the human pilot. A pure delay T has the transcendental transfer function ke^{-sT} . This function can easily be plotted in a graphical frequency-response design format but, for root-locus design, it can only be approximated as a rational polynomial function (Franklin et al., 2002). In the case of nonlinear plants, the *describing function* technique allows us to analyze stability and *limit cycle* oscillations by using a movable critical point on the Nyquist plot (West, 1960). Other important Nyquist-related design tools are the *gain* and *phase margins*; these will be illustrated here and applied in Chapter 4.

If the open-loop frequency-response locus passes close to the point $(-1 + j0)$, the stability boundary is being approached and the system transient response is likely to be underdamped. The gain margin of a feedback loop is the increase in gain that can be allowed before the loop becomes unstable. It can be calculated by finding the gain at the phase crossover frequency, as illustrated in Figure 3.9-13. The phase margin is the number of degrees by which the phase angle of GH exceeds -180° when $|GH| = 1.0$. It can be calculated from the gain crossover shown in Figure 3.9-13. As a rule of thumb a phase margin of 30° to 60° will be required to obtain a good closed-loop transient response, and this should be accompanied by a gain margin of 6 to 15 dB. When closing a feedback loop produces only an underdamped complex pair of poles, the closed-loop damping ratio is related to the phase margin by $\zeta \approx PM^0/100$ for phase margins up to about 70° (Franklin et al., 2002). This relationship also holds approximately if the closed dynamics are dominated by a complex pair.

In classical frequency-domain design, lead, lag, and PI cascade compensators are used, in conjunction with gain and phase margins, to achieve satisfactory closed-loop designs. We will first review the frequency-domain properties of these compensators. Table 3.3-1 shows passive networks that implement lead and lag compensation (see also Section 3.3), and the lead and lag transfer functions can both be written (apart from a gain constant) as

$$G_C(s) = \frac{s+z}{s+p} \quad \begin{array}{l} p > z \equiv \text{lead} \\ p < z \equiv \text{lag} \end{array} \quad (3.9-30)$$

Inspection of this transfer function shows that the hf gain is 1.0 and the lf gain is z/p , with a phase angle of zero in both cases. The polar plot of $G_C(j\omega)$ is a semicircle

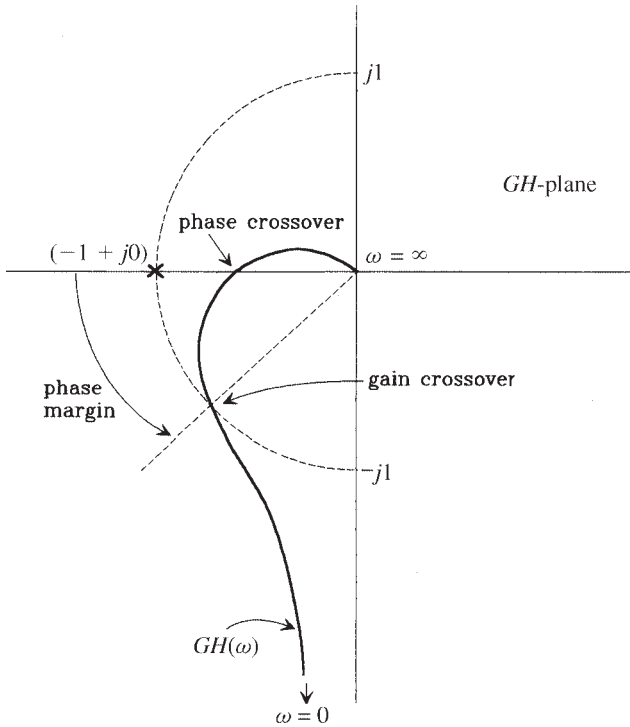


Figure 3.9-13 Stability margins on the Nyquist plot.

above (lead) or below (lag) the positive real axis, with its diameter determined by the lf and hf gains (Problem 3.9-9). This is shown in Figure 3.9-14 for the lead transfer function. The figure shows that the maximum lead angle ϕ_M and the corresponding gain are given by

$$\sin \phi_M = \frac{1 - z/p}{1 + z/p} \quad (3.9-31a)$$

$$|G(\phi = \phi_M)| = \sqrt{(z/p)} \quad (3.9-31b)$$

For the passive lag compensator this gain must be multiplied by p/z , giving

$$|G(\phi = \phi_M)| = \sqrt{(p/z)} \quad (\text{lag comp.}) \quad (3.9-31c)$$

The Bode plot shows that the frequency of maximum lead or lag is the geometrical mean of the corner frequencies:

$$\omega_{\phi_M} = \sqrt{(pz)} \quad (3.9-31d)$$

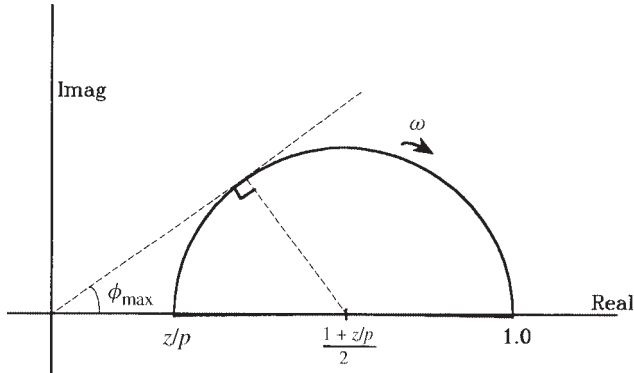


Figure 3.9-14 Lead compensator polar plot.

The design techniques with these compensators are illustrated in the following examples.

Example 3.9-4: Design of a Passive Lag Compensator This system has unity feedback, with a loop transfer function

$$G_c(s)G_p(s) = \frac{p_c (s + z_c)}{z_c (s + p_c)} \frac{K}{s(s+1)(s+15)(s+20)},$$

and the closed-loop requirements will be a velocity error coefficient of $K_v \geq 13$ and a phase margin of 25° .

A loop gain of $K = 4000$ meets the K_v requirement. Also, the Bode plot of G_p shows that the phase angle is -150° at a frequency $\omega_1 = 1.25$ rad/s. If the compensated loop transfer function has unit magnitude at this frequency and the compensator produces only about 5° lag, then the phase margin requirement will be satisfied. A passive lag compensator has a gain close to p_c/z_c and about 5° lag at one decade above the upper corner frequency. Therefore, we now choose the compensator zero to be $Z_c = 0.1\omega_1 = 0.125$ rad/s. At ω_1 the magnitude of the plant transfer function is 6.5, and so we require $p_c/z_c = 1/6.5$. This gives the compensator pole frequency as $p_c = 0.125/6.5$, or about 0.02.

Parts of the following MATLAB code were used to produce the Bode and Nyquist plots shown in Figures 3.9-15a and b and also a step response.

```
den= conv([1 1 0],[1 35 300]); num=[4000];           % Plant
nc= conv([1.125], num);
dc=conv(6.5*[1.02],den);                             % Plant + Compr.
margin(num,den); hold on margin(nc,dc)                % Margins

w= 2*pi*logspace(-.5,1,400);                          % Code for Nyquist Plots
[re,im]=nyquist(num,den,w);                          % Uncompensated
plot(re,im)
```

```

grid on
axis([-4,.5,-1,.4])
hold on
w=2*pi*logspace(-.8,1,400);
[re,im]=nyquist(nc,dc,w);           % Compensated
plot(re,im)

sys=tf(nc,dc);                     % Code for closed loop step
sys2=feedback(sys,1,-1);           % close loop
step(sys2)

```

The Bode plots in Figure 3.9-15a show that, above about 0.1 rad/s, the lag compensator has cut the gain by a constant amount and, at the gain crossover frequency (1.27 rad/s), it adds negligible phase lag. This stabilizes the system, and the compensated phase margin is almost exactly equal to the design value of 25° . The phase lag of the compensator can be seen to be concentrated in the range 0.01 to 0.5 rad/s. The Nyquist plots in Figure 3.9-15b show the unstable uncompensated system and the stable compensated system. Because of the small phase margin, the closed-loop step response is lightly damped (overshoot $> 50\%$), and the design could easily be repeated to increase the phase margin.

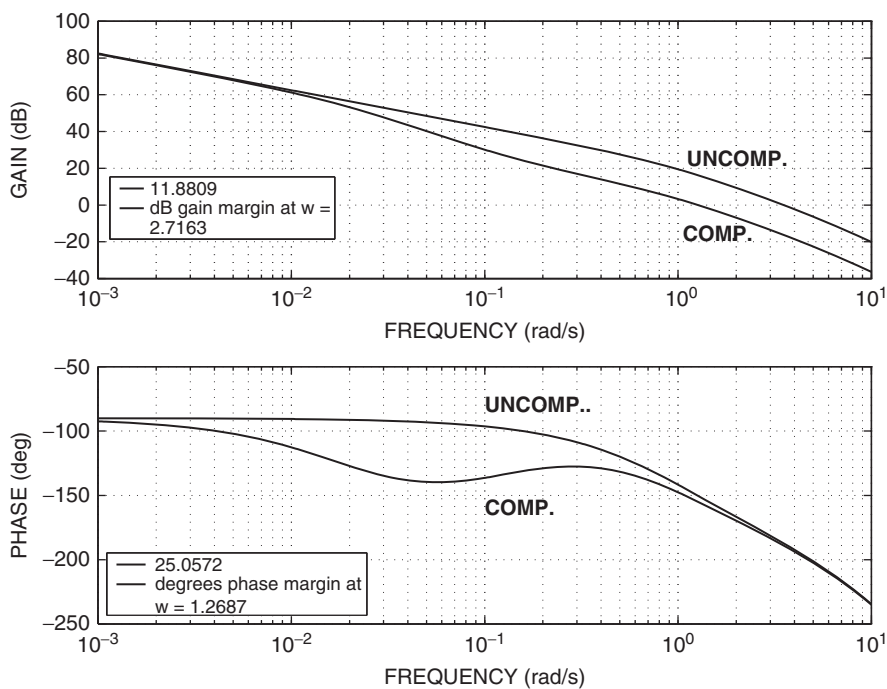


Figure 3.9-15a Lag-compensated Bode plots.

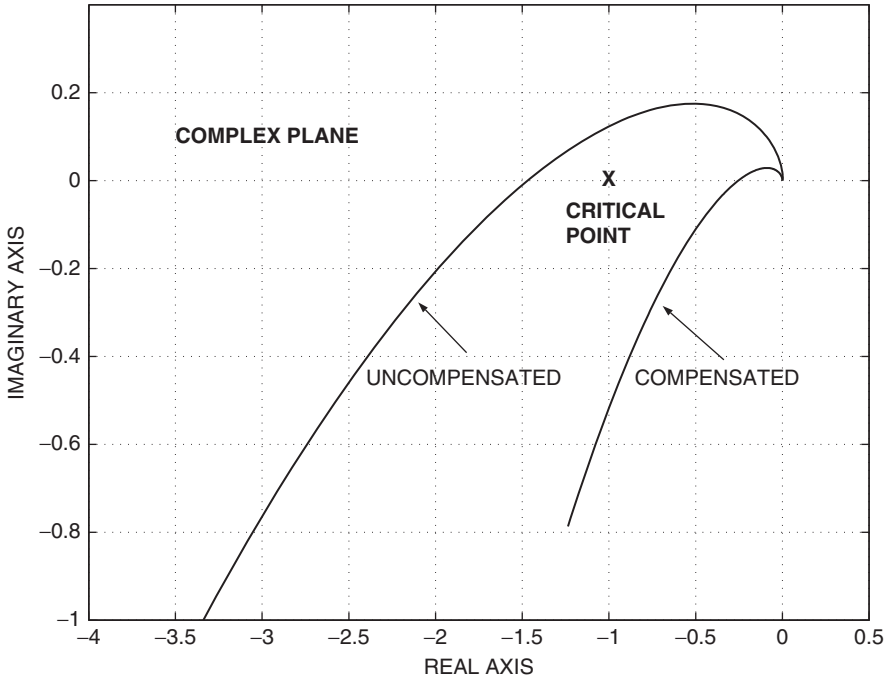


Figure 3.9-15b Lag-compensated Nyquist plot. ■

Example 3.9-5: Design of a Passive Lead Compensator In this example we will use a passive lead compensator to stabilize an unstable type-2 system. The design specifications will be to achieve a phase margin of 45° with the highest possible acceleration error coefficient K_a and a compensator pole-zero ratio not greater than 10. Using the passive lead transfer function from Table 3.3-1, the loop transfer function will be

$$G_c(s)G_p(s) = \frac{(s + z_c)}{(s + p_c)} \frac{K}{s^2(s + 5)}$$

The usual starting point for lead compensator design is to choose the frequency of maximum lead to be equal to the phase margin frequency of the plant. A Bode plot of the plant only, with $K = 1$, shows that the phase margin frequency and the phase margin are, respectively,

$$\omega_\phi = 0.45 \text{ rad/s}$$

$$\phi_M = -5.1^\circ \text{ (unstable)}$$

The required compensator phase lead is obtained from the design specification, with an allowance of an extra 5° :

$$\phi_{MAX} = (45^\circ - (-5.1^\circ)) + 5^\circ \approx 55^\circ$$

Therefore, from (3.9-31a), the compensator zero-pole ratio is

$$\frac{z_c}{p_c} = \frac{1 - \sin(\phi_{MAX})}{1 + \sin(\phi_{MAX})} = 10.05$$

By setting

$$\omega_{\phi M} = \omega_{\phi}$$

as noted above, the compensator equation (3.9-31d) gives

$$z_c = \omega_{\phi} \sqrt{z_c/p_c} = 0.142 \text{ rad/s}$$

$$p_c = 10z_c = 1.42 \text{ rad/s}$$

The compensated phase margin can now be checked, and Figure 3.9-16 shows the compensated and uncompensated Bode plots as well as the gain and phase margins. The phase margin is only 42.3° but, if we adjust the compensator to move the peak of the phase curve to the left, it will coincide with the gain crossover and the phase margin will be adequate. It is also evident that the gain margin is bigger than required and, if we raise the loop gain, the phase margin will improve without changing the compensator. For the next design iteration $K = 3$ was used, and the peak of the phase curve occurred exactly at the gain crossover, with a phase margin of 50° and gain margin of 22.4 dB.

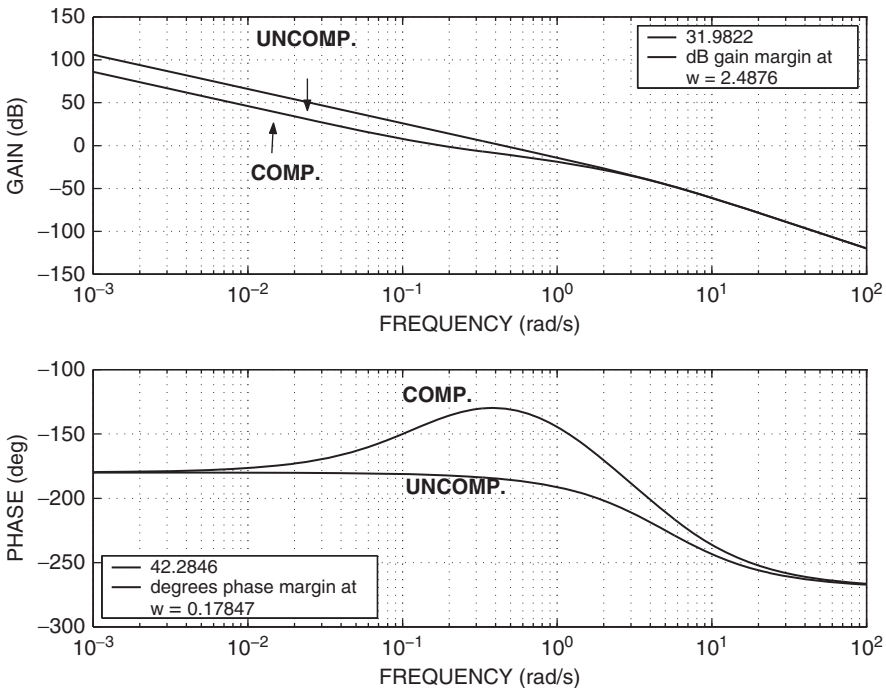


Figure 3.9-16 Lead-compensated Bode plots.

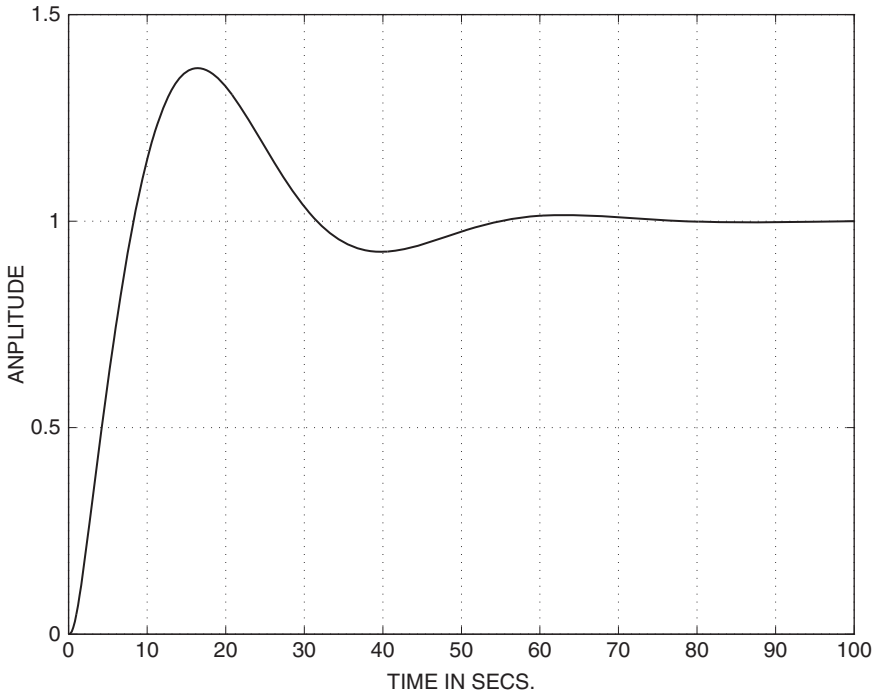


Figure 3.9-17 Lead-compensated step response.

The phase margin of 50° is greater than required by our specification, and we could either retain $K = 3$ and retune the compensator using a reduced p/z ratio or increase K and retune the compensator using the same p/z ratio. For example, we can achieve $\phi_M = 45^\circ$ with $K = 13$ if we keep $(p_c/z_c) = 10$ and use $z_c = 0.31$. Another consideration is that the acceleration error coefficient is given by

$$K_a = \frac{z_c}{p_c} \frac{K}{5}$$

and should be checked as we trade gain K with compensator pole-zero ratio. However, even with the generous margins of 50° in phase and 22.4 dB in gain, the step response, plotted in Figure 3.9-17, has a large overshoot. The next example illustrates a way to overcome this problem. ■

In summary, a phase-lead cascade compensator has the effect of increasing closed-loop bandwidth, thereby producing a faster system. It usually provides a moderate increase in error coefficient and an overshoot in the closed-loop step response.

The frequency-domain design techniques that have been illustrated above do not distinguish between transfer functions in the forward path or in the feedback path. If the cascade compensator is in the forward path, its zeros appear as zeros of the closed-loop transfer function. If it is in the feedback path, its poles will appear as

closed-loop zeros. In the above example, the lead compensator zero, close to the origin, caused a large overshoot in the step response. If the same compensator is moved to the feedback path, the resulting closed-loop zero will be much farther to the left and the overshoot will be reduced. This technique has been used in aircraft and missile control systems.

Another technique that can be used to overcome the effect of the closed-loop zero is to cancel it with a pole of the prefilter H_r . Alternatively, instead of forward-path lead compensation, another compensation technique that similarly speeds up the system response can be used, for example, inner-loop rate feedback.

Example 3.9-6: Feedback Compensation with a Phase-Lead Network Here we will use the results of Example 3.9-5 with the lead compensator in the feedback path. To demonstrate the effectiveness of this technique the loop gain has been increased to $K = 106$ when the phase margin is only 30° (the optimum compensator is now $z_c = 0.93$, $p_c = 9.3$). The compensator dc gain has been increased to unity by multiplying the B - and D -matrices by p/z and, to maintain $K = 106$, the plant gain has been reduced by z/p . The following MATLAB code will generate step, ramp, and parabolic responses:

```
ap=[0 1 0; 0 0 1; 0 0 -5];      k=106;                % Plant
bp=[0; 0; k*z/p];  cp=[1 0 0];  dp=[0];
p=9.3;  z=.93;                % Compensator
ac=[-p];  bc=[p/z];  cc=[z-p];  dc=[p/z];
[a,b,c,d]= feedback(ap,bp,cp,dp,ac,bc,cc,dc,-1);  % Close loop
t=[0:.005:6];
u=ones(length(t),1);          % Step Input
%u=t';                        % Ramp Input
%u=[0.5*t.^2]';               % Parabolic Input
[y,x]=lsim(a,b,c,d,u,t);      % Time history
plot(t,y,t,u)
```

Figure 3.9-18 shows the unit-step response. The overshoot is about 5%, compared to 55% when the compensator is in the forward path, and the speed of response is about the same in each case.

When Equation (3.9-17) is applied, with $q = 2$, we find

$$\begin{aligned} \text{step input, } n = 0 : \quad & e_{ss} = 0 \\ \text{ramp input, } n = 1 : \quad & e_{ss} = (p/z - 1)/p = 0.968 \\ \text{parabolic input, } n = 2 : \quad & e_{ss} = \infty \end{aligned}$$

These results can be confirmed by simulation using the code given above. Therefore the system has effectively been reduced to a type-1 system. Depending on the design specifications, this may be perfectly acceptable.

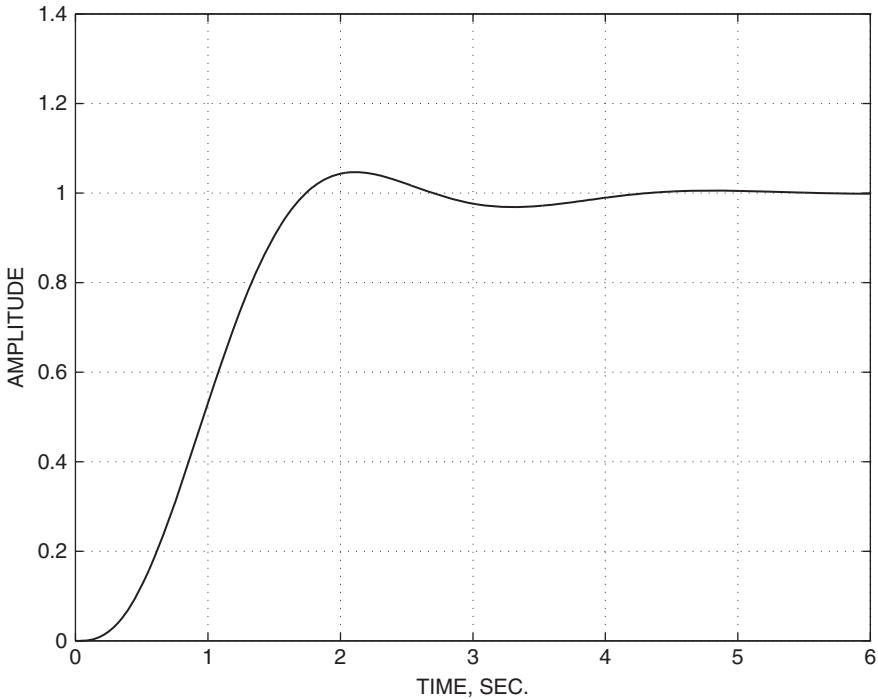


Figure 3.9-18 Step response with feedback lead compensation. ■

3.10 SUMMARY

In this chapter we have developed all of the components shown in Figure 3.1-1. Two nonlinear state-space aircraft models have been provided in the form of source code. Programs for trimming, linearization, and time-response simulation have been described, and some source code is given in Appendix B. All of the development has been illustrated with applications to aircraft, so that the reader should be well prepared for aircraft control system design in Chapter 4. Our review of linear systems and feedback control has been limited to theory and techniques that we use in the text. For additional background material, the reader should consult some of the current control theory texts (Kailath, 1980; Kuo, 1987; D’Azzo and Houpis, 1988; Brogan, 1991; Nise, 1995; Dorf and Bishop, 2001; Ogata, 2002; Franklin et al., 2002).

REFERENCES

- Brogan, W. L. *Modern Control Theory*. 3d ed. Englewood Cliffs, N.J.: Prentice Hall, 1991.
- Brown, R. G., and P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. 3d ed. New York: Wiley, 1997.

- D'Azzo, J. J., and C. H. Houpis. *Linear Control System Analysis and Design*. 3d ed. New York: McGraw-Hill, 1988.
- Desoer, C. A., and J. D. Schulman. "Zeros and Poles of Matrix Transfer Functions and Their Dynamical Interpretation." *IEEE Transactions on Circuits and Systems* CAS-21 (1974): 3–8.
- DeRusso, P. M., R. J. Roy, and C. M. Close. *State Variables for Engineers*. New York: Wiley, 1965, p. 397.
- Dommasch, D. O., S. S. Sherby, and T. F. Connolly. *Airplane Aerodynamics*. 4th ed. Belmont, Calif.: Pitman, 1967.
- Dorf, R. C., and R. H. Bishop. *Modern Control Systems*. 9th ed. Upper Saddle River, N.J.: Prentice Hall, 2001.
- Emami-Naeini, A., and P. Van Dooren. "Computation of Zeros of Linear Multivariable Systems." *Automatica* 18, no. 4 (1982): 415–430.
- Etkin, B. *Dynamics of Atmospheric Flight*. New York: Wiley, 1972.
- Franklin, G. F., J. D. Powell, and A. Emami Naeini. *Feedback Control of Dynamic Systems*. 4th ed. Upper Saddle River, N.J.: Prentice Hall, 2002.
- Gear, C. W. *Numerical Initial Value Problems in Ordinary Differential Equations*. Englewood Cliffs, N.J.: Prentice Hall, 1971.
- Hamming, R. W. *Numerical Methods for Scientists and Engineers*. New York: McGraw-Hill, 1962.
- Healey, M. "Study of Methods of Computing Transition Matrices." *Proceedings of the IEE* 120, no. 8 (August 1973): 905–912.
- Hindmarsh, A. C. "Large Ordinary Differential Equation Systems and Software." *IEEE Control Systems Magazine* (December 1982): 24–30.
- IMSL. *Library Contents Document*. 8th ed. Houston, Tx.: International Mathematical and Statistical Libraries, 1980.
- Isaacson, E., and H. B. Keller. *Analysis of Numerical Methods*. New York: Wiley, 1966.
- Kailath, T. *Linear Systems*. Englewood Cliffs, N.J.: Prentice Hall, 1980.
- Kuo, B. C. *Automatic Control Systems*. Englewood Cliffs, N.J.: Prentice Hall, 1987.
- Laning, J. H., and R. H. Battin. *Random Processes in Automatic Control*. New York: McGraw-Hill, 1956, Appendix C.
- MacFarlane, A. G. J., and N. Karcianas. "Poles and Zeros of Linear Multivariable Systems, A Survey of the Algebraic, Geometric, and Complex-variable Theory." *International Journal of Control* 24 (1976): 33–74.
- MATLAB User's Guide*. Natick, Mass.: MathWorks, Inc., 1990.
- McRuer, D., I. Ashkenas, and D. Graham. *Aircraft Dynamics and Automatic Control*. Princeton, N.J.: Princeton University Press, 1973.
- MIL-F-8785C*. "U.S. Dept. of Defense Military Specification: Flying Qualities of Piloted Airplanes," November 5, 1980.
- Moler, C., and C. Van Loan. "Nineteen Dubious Ways to Compute the Exponential of a Matrix." *SIAM Review* 20, no. 4 (October 1978): 801–836.
- Nelder, J. A., and R. Mead. "A Simplex Method for Function Minimization." *Computer Journal* 7 (1964): 308–313.
- Nguyen, L. T., et al. "Simulator Study of Stall/Post-Stall Characteristics of a Fighter Airplane with Relaxed Longitudinal Static Stability." *NASA Technical Paper 1538*. Washington, D.C.: NASA, December 1979.

- Nise, N. S. *Control Systems Engineering*. 2d ed. Menlo Park, Calif.: Addison-Wesley, 1995.
- Ogata, K. *System Dynamics*. 3d ed. Upper Saddle River, N.J.: Prentice Hall, 1998.
- *Modern Control Engineering*. 4th ed. Upper Saddle River, N.J.: Prentice Hall, 2002.
- Phillips, E. G. *Functions of a Complex Variable*. 8th ed. Edinburgh: Oliver and Boyd, 1961.
- Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. New York: Cambridge University Press, 1986.
- Ralston, A. *A First Course in Numerical Analysis*. New York: McGraw-Hill, 1965.
- Shampine, L. F., and M. K. Gordon. *Solution of Ordinary Differential Equations: The Initial Value Problem*. San Francisco: W. H. Freeman, 1975.
- Taylor, J. H., and A. J. Antoniotti. "Linearization Algorithm for Computer-Aided Control Engineering." *IEEE Control Systems Magazine* 13, no. 2 (April 1993): 58–64.
- Van Loan, C. F. "Computing Integrals Involving the Matrix Exponential." *IEEE Transactions on Automatic Control* AC-23, no. 3 (June 1978): 395–404.
- West, J. C. *Analytical Techniques for Nonlinear Control Systems*. London: The English Universities Press, 1960.
- Yuan, S. W. *Foundations of Fluid Mechanics*. Englewood Cliffs, N.J.: Prentice Hall, 1967.
- Zakian, V. "Rational Approximants to the Matrix Exponential." *Electronics Letters* 6, no. 5 (December 10, 1970): 814–815.

PROBLEMS

Section 3.2

- 3.2-1** Given the mechanical system in Figure 3.2-1, add another mass, m_2 , at the junction of k_2 and d_2 . Let the mass have negligible friction to ground (other than d_2). Find a set of state equations for this system and write out the A , B , C , D coefficient matrices. The input is u , and there are two outputs: y and w .
- 3.2-2** Repeat Problem 3.2-1 with an additional spring, k_3 , connected from m_1 to ground.
- 3.2-3** For the bridged-T circuit shown in Figure 3.2-2, follow the method given in Example 3.2-2 and find expressions for the rest of the elements of the A -, B -, C -, and D -matrices.
- 3.2-4** Use the technique from Example 3.2-2 to find a set of state and output equations for the quadratic-lag circuit in Table 3.3-1.
- 3.2-5** Given the differential equation

$$2\ddot{y} + 3\dot{y} + 4y = 4\ddot{u} + 6\dot{u} + u$$

turn it into an integral equation for y , draw a simulation diagram, assign state variables to the outputs of the integrators, and find a set of coefficient matrices A , B , C , D for the state equations. Show that your A - and B -matrices agree with Equation (3.2-6).

3.2-6 Apply Equation (3.2-9) to two sample periods and use Simpson's rule to obtain an approximation to the integral. Then obtain a recursion formula for $x(k)$.

3.2-7 Given the A -matrix

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -4 & -3 \end{bmatrix}$$

find, by hand, the eigenvalues, eigenvectors, and a modal matrix. Use (3.2-12) to find the matrix e^{At} .

3.2-8 Use the formula (3.2-18) to find e^{At} , in its simplest form, for the A -matrix

$$A = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix}$$

3.2-9 Use the Laplace transform to solve the following ODE, with α as a parameter and zero initial conditions. Reduce the solution to its simplest form (i.e., one trigonometric function, not two).

$$\dot{y} + y = \sin(10t + \alpha)U_{-1}(t), \quad 0 \leq \alpha \leq \pi/2$$

Plot a few graphs (e.g., in MATLAB) of the solution, for $0 \leq t \leq 5$ s, and use these to explain the effect of different values of α . Suggest a practical situation that this model describes.

3.2-10 (a) Put the following ODE into state-space form and solve the state equations by Laplace transform:

$$\ddot{y} + 2\dot{y} + 25y = 10 \sin(\omega_1 t)U_{-1}(t)$$

(b) Construct a plot of the amplitude of the particular solution, y_p , as ω_1 is varied from 1 to 20 rad/s.

Section 3.3

3.3-1 Given the following A - and B -matrices, use Cramer's rule to find the transfer function $X_2(s)/U(s)$:

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 2 \\ -1 & -3 & -2 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

3.3-2 Use the Laplace transform to find the step response of the transfer function

$$\frac{s + \alpha}{s^2 + s + 1}$$

with α as a parameter. Program the answer and obtain plots of the step response for positive and negative α . Describe the effect of the zero on the system step response.

- 3.3-3** Use the Laplace transform to find the step response of the simple-lead transfer function $s\tau / (s\tau + 1)$.
- 3.3-4** Use the Laplace transform to find the unit impulse response of a standard form quadratic-lag transfer function.
- 3.3-5** (a) Show that the Laplace transform of a periodic function $f(t) \equiv f(t + kT)$, $k = \text{integer}$, is given by

$$F(s) = \frac{\mathcal{L}[f_1(t)]}{(1 - e^{-Ts})} \quad \text{where } f_1(t) \equiv \begin{cases} f(t) & 0 \leq t \leq T \\ 0, & \text{elsewhere} \end{cases}$$

(b) Sketch the poles and zeros of $F(s)$, assuming a set for $F_1(s)$.

- 3.3-6** (a) The transfer function of a zero-order hold is given by

$$G(s) = \frac{1 - e^{-Ts}}{s}$$

Explain the effect of the factor $(1 - e^{-Ts})$ and contrast it with the same factor in Problem 3.3-5.

(b) Sketch all of the poles and zeros of $G(s)$.

Section 3.4

- 3.4-1** Program the second-order ABM formula, Equation (3.4-14), as an M-file. Use it to integrate the Van der Pol equation (Example 3.4-1) and perform an execution speed versus accuracy comparison with the RK4 integration.
- 3.4-2** Simulate the Lorenz equations

$$\begin{aligned}\dot{x} &= 10(y - x) \\ \dot{y} &= (r - z)x - y \\ \dot{z} &= xy - 8z/3\end{aligned}$$

using the format of Example 3.4-1. Choose a set of initial conditions, ($X^T(0) = [-0.1, 0.1, -0.1], [-1, -1, 100], [0, 5, 75]$ is suggested) and investigate the dynamic behavior for $0 < r < 170$. Plot waveforms (time histories) and 2D and 3D phase portraits. Make provision to view only the last several seconds of a phase portrait so that asymptotically stable periodic orbits can be seen.

Section 3.5

- 3.5-1** (a) Program the transport aircraft model in Section 3.5.
 (b) Check your model using the data in Table 3.6-1; calculate the weighted sum of squares of the derivatives for each test case.
 (c) Find, in the textbook, the source of each of the state equations. Show source equation numbers and give any algebraic derivation.
- 3.5-2** (a) Program the F-16 model given in Section 3.5.
 (b) Make a plot of $CM(\alpha, \epsilon)$ with “ ϵ ” as a parameter.
 (c) Plot $CZ(\alpha, \beta, \epsilon)$ to best display its 3D nature.
 (d) Make a driver program for your model in part (a) and obtain the test case results given in Table 3.5-2.

Section 3.6

- 3.6-1** (a) With the transport aircraft model from Problem 3.5-1, use the TRIM.m program to reproduce the steady-state trim conditions given in Table 3.6-1.
 (b) Use the trim program to find out how steeply the aircraft (in clean configuration, with $x_{cg} = 0.25\bar{c}$) can climb for a range of speeds from 200 to 500 ft/s at sea level. Compute the rate of climb (ROC) for each speed and determine the speed at which the ROC is a maximum.
- 3.6-2** Devise a trim algorithm or use the program in the Appendix B1 to trim the F-16 model. Duplicate some longitudinal trims from Tables 3.6-2 and 3.6-3.
- 3.6-3** (a) Derive Equation (3.6-3) for the pitch attitude in terms of the flight-path angle.
 (b) Derive Equation (3.6-5).
- 3.6-4** Modify the trim program used in Problem 3.6-1 to trim the transport aircraft for a prescribed angle of attack by varying V_T . Derive a trim condition for $\alpha = 15^\circ$ at 10,000 ft.
- 3.6-5** Use the trim and time-history programs to duplicate the results of Examples 3.6-3 and 3.6-4.

Section 3.7

- 3.7-1** Given the nonlinear state equations

$$\dot{x}_1 = x_1^3 - x_2^2 + 8$$

$$\dot{x}_2 = x_1 x_2$$

- (a) Find all of the singular (equilibrium) points.
 (b) Linearize the equations and find the algebraic “A-matrix.”

- (c) Find the numerical A -matrix and its eigenvalues (by hand) at each singular point and describe the type of perturbed behavior that you would expect near each point.
- 3.7-2** Program the MATLAB linearization algorithm given in Section 3.7 and add a calculation of the B -matrix. Use this to confirm the results of Example 3.7-1.
- 3.7-3** Write a program to compute the matrices $E^{-1}A$ and $E^{-1}B$ for the decoupled longitudinal equations given in Section 2.6 from the stability derivatives. Test it on the transport aircraft model and compare the results with those given in Example 3.7-1.
- 3.7-4** (a) Derive the result given in Equation (3.7-4).
 (b) Incorporate Equation (3.7-4) in the linearization program and compare its performance with the original algorithm on the transport aircraft model.

Section 3.8

- 3.8-1** Find and identify the eigenvalues and eigenvectors of the transport aircraft A -matrix given in Example 3.8-5. Use the modal expansion [Equation (3.2-15)] to determine what variables are chiefly involved in each mode.
- 3.8-2** Run linear and nonlinear time-history simulations of a step throttle input to the transport aircraft model using the data of Example 3.8-5. Compare the various speed and altitude responses and confirm the points made in Example 3.8-5 about the transfer functions.
- 3.8-3** (a) Obtain magnitude and phase Bode plots for the transport aircraft throttle-to-speed transfer function using the data of Example 3.8-5.
 (b) Repeat part (a) for the throttle-to-altitude transfer function. Explain how the features of the plots match the transfer function factors and identify all asymptotes.
- 3.8-4** (a) Use the transport aircraft dynamics in Example 3.8-5 to find the Bode magnitude and phase plots of the elevator-to-pitch-rate transfer function.
 (b) Determine a short-period approximation and show it on the same plots as in part (a).
 (c) Repeat part (a) using the elevator-to-pitch-attitude transfer function and explain the difference between the two sets of graphs.

Section 3.9

- 3.9-1** (a) A unity-feedback control system has the forward-path transfer function $G(s) = 18(s + 2) / [s(s + 3)(s + 6)]$. Calculate, by hand, the steady-state error when the reference input is a unit-ramp function.
 (b) Confirm the answer to part (a) by means of a simulation and plot.

- 3.9-2** (a) A feedback control system has forward-path SISO transfer functions G_1 followed by G_2 in cascade and a feedback transfer function H . An additive disturbance $D(s)$ is injected between G_1 and G_2 . Find the transfer functions from D to the output Y and to the error $(R - Y)$.
- (b) If $G_1 = 10/s$, $G_2 = 1/[s(s + 5)]$, and $H = 10(s + .9)/(s + 9.0)$, find the error as a function of time when the disturbance is a unit step.
- (c) If we are free to redistribute the gain in the forward path, how can the error be reduced?
- 3.9-3** (a) A unity-feedback control system has $G_c = k(s + z)/s$ and $G_p = 10/[s(s + 10)]$. Determine the compensator parameters k and z to achieve closed-loop poles with $\zeta = 1/\sqrt{5}$ and the highest possible error constant.
- (b) This system controls the azimuth rotation of a radar antenna. The antenna is tracking a target with a velocity vector $[2000, 0, 0]^T$ starting from an initial position of $[-10000, 2000, 0]^T$ at $t = 0$ (coordinate origin at the radar, right handed with x and y in the horizontal plane, and y pointing to the closest approach point of the target). Use MATLAB “Isim” to obtain a plot of the tracking error as the target goes past the radar. Use a state-space model and calculate an initial-condition vector to avoid a large transient.
- 3.9-4** Repeat Example 3.9-1 and obtain your own Nyquist plot. Calculate, by hand, the value of K at the stability boundary.
- 3.9-5** (a) A feedback control system has the loop transfer function

$$G(s)H(s) = \frac{K}{(s + 1)(s + 2)(s + 4)}, \quad K > 0$$

Sketch the Nyquist D-contour and the Nyquist plot and label all of the significant corresponding points.

- (b) Solve, by hand, the equation $\text{Im}[GH(j\omega)] = 0$ to find the value of K that gives neutral stability.
- 3.9-6** A feedback control system has the loop transfer function

$$G(s)H(s) = \frac{K(s + 6)}{s(s + 4)(s^2 + 4s + 8)}, \quad K > 0$$

- (a) Make a rough sketch of the expected root-locus plot.
- (b) Calculate, by hand, any real-axis breakaway and entry points and the angles and real-axis intersection points of any asymptotes.
- (c) Use any available commercial software to get an accurate root-locus plot.
- 3.9-7** Redesign Example 3.9-2 with root locus to try to get dominant poles with $\zeta = 1/\sqrt{2}$ and p/z not greater than 10.

- 3.9-8** Given the loop transfer function $G(s)H(s) = K(s + 1)/[s(s - 1)]$, with $K > 0$:
- (a) Draw the D-contour and Nyquist plot and identify corresponding points on each.
 - (b) Calculate the value of K that gives marginal stability.
 - (c) Find the gain and phase margins when $K = 2$.
- 3.9-9** (a) Show that the polar plot of the transfer function $G(s) = (s + z)/(s + p)$ is a semicircle with its diameter on the real axis.
- (b) Derive the expression for the maximum phase lead or phase lag, the frequency at which this occurs, and the gain at this frequency.
 - (c) Sketch the polar plot for the phase-lag case.
- 3.9-10** Design a lead compensator for the unity-feedback control system in Example 3.9-2 [forward-path transfer function $100/(s^2(s + 10))$]. Use a lead compensator with a pole-to-zero ratio of 10. Design for the largest possible loop gain consistent with a gain margin of at least 12 dB and (a) a 30° phase margin and (b) a 45° phase margin. Derive the state equations with the compensator included (as in Example 3.9-2), close the loop with the appropriate gain matrix, and compare the step responses of these two designs.