NASA Technical Paper 3397

August 1993

Dynamic Forms Part I: Functions

George Meyer
and G. Allan Smith

NASA
National Aeronautics and Space Administration

NASA Technical Paper 3397

1993

Dynamic Forms Part I: Functions

George Meyer
and G. Allan Smith
Ames Research Center, Moffett Field, California

National Aeronautics and Space Administration

Ames Research Center
Moffett Field, California 94035-1000

CONTENTS

NOMENCLATURE

$\overleftrightarrow{a}$	right-handed orthonormal axis system a
$axis(M)$	axis operator; see equation (4-5)
C_{ab}	3×3 right-handed direction cosine matrix, rotation matrix from $\overleftrightarrow{b}$ to $\overleftrightarrow{a}$
C_i^k	binomial coefficient, namely $k!/[i!(k-i)!]$
$E_q(\alpha)$	elementary (Euler) rotation about axis q through angle α
f_x	partial derivative of f with respect to x
$poly'_n$	polynomial of degree n with constant coefficients
$poly_n$	polynomial of degree n with possibly variable coefficients
R^n	n-tuples of real numbers
$S(x)$	cross-product operator; see equation (3-3)
$sat(x,a)$	saturation at a for $\lvert x\rvert > a$
X^T	transpose of matrix X
X^{-1}	inverse of matrix X
$\dot{x}$	time derivative of x
$\hat{x}$	estimate of x
$\vec{x}$	vector x
$x^{(m)}$	m^{th} time derivative of x
x_a	column matrix of coordinates of $\vec{x}$ with respect to $\overleftrightarrow{a}$
α_{ab}	Euler angles of axis $\overleftrightarrow{a}$ relative to axis $\overleftrightarrow{b}$
δ_i	unit column matrix with 1 in location i
ω_{abc}	$\overleftrightarrow{c}$-coordinates of angular velocity of $\overleftrightarrow{a}$ relative to $\overleftrightarrow{b}$

Scalar Forms

$SF^m(x)$	scalar form of x to order m, namely $\left(x, \dot{x}, x^{(2)}, \ldots, x^{(m)}\right)$
$SF^m(0)$	zero scalar form, $(0, \ldots, 0)$
$SF^m(1)$	unit scalar form, $(1, 0, \ldots, 0)$
(x, a)	abbreviation for $(x, a, 0, \ldots, 0)$
$\{SF^m(x)\}_k$	value in the k^{th} location, namely $x^{(k)}$

Vector Forms

$VF^m(v_a)$	vector form of $\vec{v}$ in $\overleftrightarrow{a}$ to order m, namely $\left(v_a, \dot{v}_a, \ldots, v_a^{(m)}\right)$
$VF^m(0)$	zero vector form, namely $(0, \ldots, 0)$
$VF^m(\delta_i)$	unit vector form, namely $(\delta_i, 0, \ldots, 0)$
$VF^m(v_a)_i$	scalar form of the i^{th} component, namely $\left(v_{ai}, \dot{v}_{ai}, \ldots, v_{ai}^{(m)}\right)$

Matrix Forms

$MF^m(X)$	matrix form of X to order m, namely $\left(X, \dot{X}, \ldots, X^{(m)}\right)$
$MF^m(0)$	zero matrix form, namely $(0, \ldots, 0)$
$MF^m(I)$	unit matrix form, namely $(I, 0, \ldots, 0)$

MF_{ab}^m	matrix form of rotation matrix C_{ab}, namely $\left(C_{ab}, \dot{C}_{ab}, \ldots, C_{ab}^{(m)}\right)$

Rotational Forms

RF_{ab}^m	rotational form of $\overleftrightarrow{a}$ relative to $\overleftrightarrow{b}$ to order m, namely $\left(C_{ab}, \omega_{aba}, \dot{\omega}_{aba}, \ldots, \omega_{aba}^{(m-1)}\right)$
RF_{aa}^m	identity rotation form, namely $(I, 0, \ldots, 0)$
$AF_q^m(\alpha_{ab})$	Euler angle form with sequence q of $\overleftrightarrow{a}$ relative to $\overleftrightarrow{b}$ to order m
PF_{ab}^m	Euler parameter form of $\overleftrightarrow{a}$ relative to $\overleftrightarrow{b}$ to order m

Functions of Dynamic Forms

$\star$	product operator
C_f	control algorithm for pure feedback systems
$f[SF^m(x)]$	scalar form of $f(x)$, namely $\left(f(x), \dot{f}(x), \ldots, f^{(m)}(x)\right)$
$f^{-1}[SF^m(x)]$	scalar form of the inverse function $f^{-1}(x)$ of $f(x)$
T_{1f}	algorithm computing the output of a dynamic system
T_{1f}^{-1}	algorithm computing the control of a dynamic system
$tay[SF^m(x), \tau]$	first $m+1$ terms of the Taylor series of $x(t+\tau)$
$MF \succ RF$	transformation taking matrix forms to rotational forms
$MF \prec RF$	transformation taking rotational forms to matrix forms
$MF \succ AF_q$	transformation taking matrix forms to Euler angle forms in sequence q
$MF \prec AF_q$	transformation taking Euler angle forms in sequence q to matrix forms
$MF \succ PF$	transformation taking matrix forms to Euler parameter forms
$MF \prec PF$	transformation taking Euler parameter forms to matrix forms
$XY \succ CY$	transformation from Cartesian to cylindrical coordinates
$XY \prec CY$	transformation from cylindrical to Cartesian coordinates
$XY \succ SP$	transformation from Cartesian to spherical coordinates
$XY \prec SP$	transformation from spherical to Cartesian coordinates

SUMMARY

The formalism of dynamic forms is developed as a means for organizing and systematizing the design of control systems. The formalism allows the designer to easily compute derivatives to various orders of large composite functions that occur in flight-control design. Such functions involve many function-of-a-function calls that may be nested to many levels. The component functions may be multiaxis, nonlinear, and they may include rotation transformations.

A dynamic form is defined as a variable together with its time derivatives up to some fixed but arbitrary order. The variable may be a scalar, a vector, a matrix, a direction cosine matrix, Euler angles, or Euler parameters. Algorithms for standard elementary functions and operations of scalar dynamic forms are developed first. Then vector and matrix operations and transformations between parameterization of rotations are developed in the next level in the hierarchy. Commonly occurring algorithms in control-system design, including inversion of pure feedback systems, are developed in the third level.

A large-angle, three-axis attitude servo and other examples are included to illustrate the effectiveness of the developed formalism. All algorithms have been implemented in FORTRAN code. Practical experience shows that the proposed formalism may significantly improve the productivity of the design and coding process.

1 INTRODUCTION

This report presents a new procedure and a collection of algorithms for the solution of several problems associated with the design of automatic control systems. Our paradigm will be aircraft flight control, but the methods apply in other domains such as spacecraft attitude control, robotics, and process control. For flight-control design purposes, an aircraft may often be adequately modeled as a rigid body with force and moment generation that depends on the state of the motion of the rigid body, the controls, and wind. The design of the corresponding fully automatic flight-control system with large operating envelopes may be difficult for several reasons:

For all but very restricted small-angle maneuvers, both the fact that nonlinearities are associated with rigid rotation and the fact that the space of rotations is not flat become significant. Rigid body attitude is typically represented by direction cosine matrices or Euler angles in some sequence, or, in the case of spacecraft, Euler parameters (ref. 1). The link between the time derivatives of these attitude variables and the angular velocity and its derivatives is nonlinear and may even become singular. Thus, rotations introduce nonlinearities and singularity into the state equation. In order to avoid singularities it may be desirable to change from one representation to another at points along a flight maneuver. For example, the usual yaw-pitch-roll sequence becomes singular (gimbal lock) for 90 degrees of pitch, and in the vicinity of this condition it may be desirable to change to the yaw-pitch-yaw sequence. Each such change in representation entails a corresponding change in the state equations. The control system must be designed to operate in the various state space representations (coordinatizations), and the switch from one coordinate system to another must be made smoothly. Smooth patching of coordinates requires various-order derivatives of the right-hand side of the state equation (system function).

In many cases the nonlinearity of the force and moment generators may not be ignored, especially for powered-lift aircraft that have strong nonlinear and rather complex interaction between power and aerodynamics. A typical algorithm for the computation of the total force and moment acting on the aircraft may contain more than 4,000 lines of FORTRAN code (ref. 2). The input to the algorithm is at least 19-dimensional, consisting of the state, which is at least 12-dimensional, plus the controls, which are at least 4-dimensional, plus wind; the output is 6-dimensional, consisting of the 3-dimensional force and moment vectors. Inside the algorithm, the input flows through many successive functions so that the analytic form of the multivariable function represented by the algorithm is a deep nesting of elementary functions and table interpolations. The depth of nesting (e.g., square of sine of square root of sum of squares of ... etc.) easily exceeds dozens of levels. Consequently the analytic computation of such a simple object as the overall 6×19 input–output Jacobian matrix may be a formidable task. But such mathematical objects are needed if the designer is to improve system performance by taking advantage of the information contained in the force and moment model.

The number of controls often exceeds the basic four. In addition to the three moment and one throttle controls, there may be controls for directing thrust (such as a one- or two-degree-of-freedom nozzle), direct lift (such as a spoiler), side force, and flaps. The set of all the controls may be redundant in the sense that many combinations of controls produce the same total force and moment on the aircraft. This redundancy may be resolved advantageously by partitioning the set into two sets: the nonredundant set of active controls and the set of parametric controls. The active controls are manipulated by the regulator; the parametric controls are manipulated by a configuration-management system so as to maintain selected control margins for the active controls as well as to maintain certain functions of the state (such as the angle of attack) within the assigned limits (ref. 3). Each such partition represents a particular control mode. Typically there are several control modes, and there may also be several tracking modes. Each such mode is associated with a particular functional dependence of the output on the state. For example, near hover, the output may be the three Cartesian coordinates of the velocity vector; at a higher speed, the output may be defined as cylindrical coordinates of the velocity, namely, horizontal speed, heading, and vertical speed; at a still higher speed, the output selected for tracking may be the spherical coordinates of the velocity, namely, airspeed, glide-path angle, and heading angle. Each combination of control and tracking modes defines an operating mode. Thus, near hover, the nozzle may be an active control, and the pitch angle may be programmed independently. At high speed, the nozzle may be fixed, that is, programmed independently, and regulation is then achieved indirectly through the pitch angle. Each operating mode is a separate control problem with its particular control variables, output variables, and, possibly, state variables and state equation. The control-system design must incorporate many such operating modes and provide smooth intermode transitions. Smooth patching of modes requires various-order derivatives of the system function.

The flight-control system as considered in this report includes the functions of configuration management, in which operating modes and reconfiguration commands are computed; guidance, in which flyable reference trajectories linking way points given by, say, air traffic control, are generated; and regulation, which ensures tracking of reference trajectories in spite of unavoidable uncertainties and approximations in system modeling. If the operating envelope is small enough relative to system nonlinearity, then linear design methods based on a single Jacobian matrix (perturbation model) of the system function, evaluated at a single operating point (trim point), are adequate for the purposes of regulator design. In such a case, the Jacobian matrix may be computed numerically by perturbing each

input variable (ref. 4). For larger envelopes, robust single-point linear designs (ref. 5) based on one representative value of the Jacobian matrix may be adequate in spite of actual variations of the matrix. Since the operating point is now a variable, feed-forward signals (particular solutions) may have to be provided to reduce tracking error and unload the feedback (ref. 6). Nonlinear methods become essential for the design of flight-control systems with large operating envelopes. Two situations arise: If the inner-loop dynamics (attitude control) can be made sufficiently fast relative to the outer-loop dynamics (trajectory control), then the nonlinearity of the force and moment function may be removed by means of numeric inversion (refs. 7–13). The nonlinear control theory based on differential geometry (refs. 14 and 15) provides design techniques when such a separation is impossible. One fruitful technique is based on a coordinate change of state and control in order to simplify the form of the state equation. In certain practical cases, a coordinate change may suppress the nonlinearity to the extent that linear design techniques become applicable in the new coordinates (ref. 16). Techniques are also available for the generation of the nonlinear analogs of the particular solution (refs. 17 and 18). The practical drawback of such techniques, for the case of flight control, is that they have a voracious appetite for various-order derivatives of the force and moment functions.

Thus, the design of large-envelope flight-control systems is difficult because the state space is not flat, the force and moment function is big and complicated, and many operating modes must be considered. The difficulty can be further traced to the need for high-order differentiation of the system function. There are three differentiation techniques: hand, symbolic, and automatic. Hand differentiation and coding is very tedious and highly unreliable for the size of problems being considered. The other two alternatives are much more appealing. The symbolic approach would be to machine translate the, say, FORTRAN code for the system function into the appropriate language (such as MACSYMA, MATHEMATICA, or MAPLE) within which differentiation is defined, and then proceed with the nonlinear design techniques employing the derivatives. Applications of this approach to relatively small systems have been successful (ref. 19). However, for larger systems (4,000 lines of FORTRAN) involving deeply nested functions, symbolic methods may be slow and may often produce large, unmanageable expressions (ref. 20).

The remaining choice for the computation of derivatives, automatic differentiation, is based on the fact (known since Leibnitz) that Taylor series, which carry derivatives as coefficients, can be propagated through an arbitrary sequence of elementary functions without any truncation error. Thus, automatic differentiation does not suffer from the rapid-chain-rule fanout of terms that plagues the symbolic differentiation. Furthermore, machine translation into an automatic-differentiation language is as practical as it is for symbolic languages (ref. 21).

The theory of dynamic forms (ref. 22) described in the present report may be considered to be a particular example of automatic differentiation and a basis for a formal language for the computer-aided design of automatic control systems.

A dynamic form is defined as a variable together with its time derivatives up to some fixed, but arbitrary, order. The variable may be a scalar, a vector, a matrix, a direction cosine matrix, Euler angles, or Euler parameters. Most of the report is devoted to the translation of a set of elementary functions and operations into corresponding functions and operations on dynamic forms. The set is rich enough so that typical system functions occurring in flight control may be assembled from the members of this set. Whereas many examples are provided to demonstrate the application of the methodology to automatic

control, the emphasis in the present report is on the formalism of dynamic forms. The emphasis in future reports will be on the application of dynamic forms and on the reformulation of control problems in terms of dynamic forms.

2 SCALAR FORMS

Mathematical models of practical dynamic processes such as an aircraft frequently contain functions of functions to many levels. A small example of a typical case is shown in figure 2.1. In the figure the output y is related to the input $x = (x_1, x_2, x_3)$ by a function f, which is built up from the elementary functions such as trig functions, powers, roots, exponentials, and, of course, addition, subtraction, multiplication, and division. The function f so constructed then becomes a block in a higher level function and so on until the final level, say $w = F(u)$, is reached. Implementation of such a function F as an algorithm on the computer is routine, even though in practice the algorithm may easily take 4,000 lines of FORTRAN. On the other hand, the symbolic expression of F becomes unwieldy. Even the simple example fragment f of the complete algorithm F is beginning to look complicated.

$$y = f(x_1, x_2, x_3) = f_5\{f_3(x_3) * f_4[f_1(x_1) + f_2(x_2)]\}/f_3(x_3) \tag{2-1}$$

The function is six levels deep in the sense that six function calls $(/, f_5, *, f_4, +, f_2)$ are needed to get from x_2 to y. Now, whereas the analytic form is not needed for simulation, it is often used during design and analysis for the computation of, for example, time derivatives, gradients, partial derivatives, and Jacobian matrices. Suppose that we wish to compute the first five time derivatives of the output y in figure 2.1 given the input (x_1, x_2, x_3) and its first five time derivatives. Repeated use of the chain rule would produce rather long expressions. In general the length grows rapidly with the depth of the nesting. The same considerations apply to partial and other derivatives since they can be expressed in terms of time derivatives. Indeed, an effective procedure for computing time derivatives can be easily adapted for computing other derivatives. This procedure will be discussed later in the report, so we focus on the computation of time derivatives. The scalar dynamic forms discussed next greatly simplify the computation of time derivatives.

Suppose that a scalar variable x is a function of time. Imagine an array with x in location 0, time derivatives running to the right, and time integrals to the left:

$$\left(\ldots,\quad x^{(-3)},\quad \int\int x,\quad \int x,\quad x,\quad \dot{x},\quad \ddot{x},\quad x^{(3)},\quad \ldots\right) \tag{2-2}$$

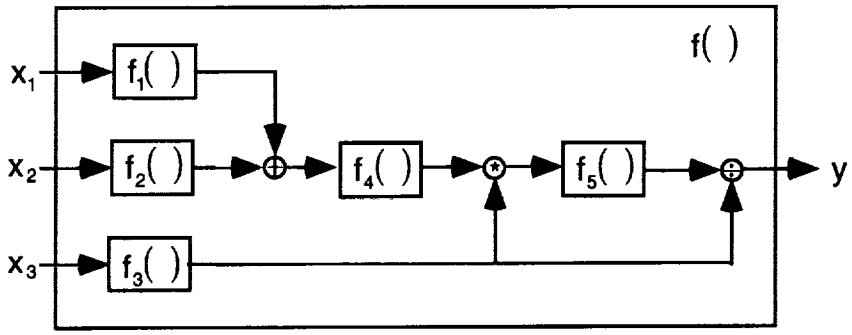

Figure 2.1. Typical nested fragment f.

The scalar form $SF^m(x)$ of order m is a glimpse at this array through a window at locations 0 through m. That is,

$$SF^m(x) = \left(x, \dot{x}, \ldots, x^{(m)}\right) \tag{2-3}$$

where $x \in R^1$ is a function of time and $x^{(m)}$ in location m denotes the m^{th} time derivative of x. Location 0 contains $x = x^{(0)}$. Scalar forms will be denoted by the symbol SF to distinguish them from other dynamic forms to be introduced later in the report.

A time derivative of a scalar form will be defined as a right shift of the window:

$$d/dtSF^m(x) = SF^m(\dot{x}) = \left(\dot{x}, \ddot{x}, \ldots, x^{(m+1)}\right) \tag{2-4}$$

The integral of a scalar form is given by a left shift of the window:

$$\int SF^m(x) = SF^m\left(\int x\right) = \left(\int x, x, \dot{x}, \ldots, x^{(m-1)}\right) \tag{2-5}$$

We use the following notation for extracting a time derivative from a form:

$$\{SF^m(x)\}_k = \begin{cases} x^{(k)} & \text{if } 0 \le k \le m \\ 0 & \text{else} \end{cases} \tag{2-6}$$

There is a close relation between a dynamic form $SF^m(x)$ and the first $m+1$ terms of a Taylor series expansion of $x(t)$ at t. Thus,

$$x(t+\tau) \approx x + \dot{x}\tau + \frac{1}{2}\ddot{x}\tau^2 + \ldots + \frac{1}{m!}x^{(m)}\tau^m$$

where the time derivatives $x^{(k)}$ are contained in $SF^m(x)$. We use the notation

$$x(t+\tau) \approx tay[SF^m(x), \tau] = \sum_{k=0}^{m} \{SF^m(x)\}_k \tau^k/k! \tag{2-7}$$

Algorithms such as equation (2-1) may be considered to be of order zero. The primary objective of this report is to develop a formalism for easy conversion of such zero-order algorithms into corresponding algorithms of order $m > 0$. Thus the algorithm in (2-1) will translate into

$$SF^m(y) = f_5\left(f_3[SF^m(x_3)] * f_4\{f_1[SF^m(x_1)] + f_2[SF^m(x_2)]\}\right)/f_3[SF^m(x_3)] \tag{2-8}$$

where the input is given by the three scalar forms (variables and their time derivatives up to order m)

$$SF^m(x_1) = (x_1, \dot{x}_1, \ldots, x_1^{(m)})$$

$$SF^m(x_2) = (x_2, \dot{x}_2, \ldots, x_2^{(m)})$$

$$SF^m(x_3) = (x_3, \dot{x}_3, \ldots, x_3^{(m)})$$

the output is given by the scalar form (variable and its time derivatives up to order m)

$$SF^m(y) = (y, \dot{y}, \ldots, y^{(m)})$$

and the meaning of f_i and the operators in equation (2-8) will be developed next. The formalism will allow us automatically to translate large zero-order algorithms such as an aircraft total-force-and-moment subroutine, which, as noted earlier, may contain 4,000 lines of FORTRAN, into an m^{th} order algorithm. That in turn may be used for the design of control algorithms such as the generation of reference trajectories and model inversion. Examples of such designs will be given later in the report.

Let us now proceed with the development of the formalism. We shall first convert elementary functions to the corresponding functions of scalar forms. Then, with these functions as basic building blocks, we will assemble a hierarchy of composite functions that are particularly useful for the design of control systems.

Algebra

Zero and unit scalar forms are defined, respectively, by

$$SF^m(0) = (0, \ldots, 0) \tag{2-9}$$

$$SF^m(1) = (1, 0, \ldots, 0) \tag{2-10}$$

The scalar form of time is

$$SF^m(t) = (t, \quad 1, \quad 0, \ldots, \quad 0) \tag{2-11}$$

To simplify notation, we assume padding with zeros:

$$(x, a) = (x, a, 0, \ldots, 0) \tag{2-12}$$

Thus, we may write

$$SF^m(0) = (0)$$
$$SF^m(1) = (1)$$
$$SF^m(t) = (t, 1)$$

It is possible to define sum, product, inverse, and division for scalar forms. If $z = ax + by$ and a and b are constant in time, then for $0 \leq k \leq m$

$$z^{(k)} = ax^{(k)} + by^{(k)} \tag{2-13}$$

An outline of the algorithm for scalar-form sum is shown next, where scalar forms are treated as scalar arrays.

ALGORITHM: $SF^m(z) = SF^m(x) + SF^m(y)$
do $k = 0, m$
$z^{(k)} = a * x^{(k)} + b * y^{(k)}$
end do

The effect of the algorithm will be denoted as

$$SF^m(z) = aSF^m(x) + bSF^m(y) \tag{2-14}$$

Note that

$$SF^m(x) - SF^m(x) = SF^m(0)$$

If $z = xy$, then the k^{th} derivative, $0 \leq k \leq m$, of z is given by the Leibnitz product rule (convolution, ref. 23):

$$z^{(k)} = \sum_{i=0}^{k} C_i^k x^{(k-i)} y^{(i)} = \sum_{i=0}^{k} C_i^k x^{(i)} y^{(k-i)} \tag{2-15}$$

where the binomial coefficient $C_i^k = k!/[i!(k-i)!]$ may be computed by means of the Pascal triangle:

$$\begin{cases} C_0^k = C_k^k = 1 & \text{for } k \geq 0 \\ C_i^k = C_{i-1}^{k-1} + C_i^{k-1} & \text{for } 2 \leq k \text{ and } 1 \leq i \leq k-1 \end{cases} \tag{2-16}$$

An outline of the product algorithm is shown next, where as before scalar forms are treated as scalar arrays.

ALGORITHM: $SF^m(z) = SF^m(x) \star SF^m(y)$
do $k = 0, m$
$z^{(k)} = 0$
do $i = 0, k$
$z^{(k)} = z^{(k)} + C_i^k * x^{(k-i)} * y^{(i)}$
end do
end do

The effect of the algorithm will be denoted as

$$SF^m(z) = SF^m(x) \star SF^m(y) \tag{2-17}$$

The scalar form product commutes since it commutes for real numbers:

$$SF^m(x) \star SF^m(y) = SF^m(y) \star SF^m(x)$$

It may be noted that the Leibnitz rule holds also for objects other than scalars. It matters only that the product is defined for objects x and y so that its time derivative

$$(xy)^{(1)} = x^{(1)}y + xy^{(1)} \tag{2-18}$$

We will take advantage of this fact later when we consider vectors and matrices. For now, we return to scalars.

Powers and Polynomials

The conversion of a zero-order algorithm to the corresponding algorithm of order m is illustrated by the following very simple case.

If $z = x^n$ for an integer $n \geq 1$, then an obvious algorithm for computing z is given by

```
ALGORITHM: z = x^n
z = 1
do i = 1, n
  z = z * x
end do
```

The rule for conversion to order m is simple: replace any variable of time by its scalar form. Thus, the algorithm for computing

$$SF^m(z) = [SF^m(x)]^n \tag{2-19}$$

is given by

```
ALGORITHM: SF^m(z) = [SF^m(x)]^n
SF^m(z) = SF^m(1)
do i = 1, n
  SF^m(z) = SF^m(z) * SF^m(x)
end do
```

where the algorithm (subroutine) for $SF^m(z) \star SF^m(x)$ has been already constructed (see eq. (2-17)). The algorithm works for any x but requires n scalar-form products. For $x \neq 0$ an algorithm requiring essentially one product will be constructed when fractional powers are considered later in the report.

Polynomials occur frequently enough in practice to deserve consideration. Let $poly'_n(a, x)$ denote a polynomial of degree n in x with constant coefficients $a = (a_0, \ldots, a_n)$, that is,

$$poly'_n(a, x) = a_0 + a_1 x + \ldots + a_n x^n = a_0 + (a_1 + \ldots + (a_{n-1} + a_n x) \ldots x) \tag{2-20}$$

If $z = poly'_n(a, x)$, then the following algorithm is a possible realization of $poly'_n$:

```
ALGORITHM: z = poly'_n(a, x)
z = a_n x
do i = 1, n - 1
  z = (a_{n-i} + z) * x
end do
z = a_0 + z
```

The order of the algorithm is raised from zero to m simply by replacing z and x by $SF^m(z)$ and $SF^m(x)$, respectively:

ALGORITHM: $SF^m(z) = poly'_n[a, SF^m(x)]$
$SF^m(z) = a_n SF^m(x)$
do $i = 1, n-1$
  $SF^m(z) = [a_{n-i} SF^m(1) + SF^m(z)] \star SF^m(x)$
end do
$SF^m(z) = a_0 SF^m(1) + SF^m(z)$

This algorithm will be denoted as

$$SF^m(z) = poly'_n[a, SF^m(x)] \tag{2-21}$$

Example 2.1. Suppose that we need to generate the following polynomial function of time:

$$x = poly'_4(a, t) = 1 + t - t^2/2 + t^3/3! + t^4/4!$$

and that we need derivatives to order six. Then we need to compute

$$SF^6(x) = poly'_4[a, SF^6(t)]$$

Thus, for example, at $t = 1$, the scalar form of time is

$$SF^6(t) = (\,1.00,\quad 1.00,\quad 0.00,\quad 0.00,\quad 0.00,\quad 0.00,\quad 0.00\,)$$

and a call to the *poly* algorithm with degree= 4 and coefficients

$$a = \left(1, 1, -\frac{1}{2}, \frac{1}{3!}, \frac{1}{4!}\right)$$

produces the scalar form of x,

$$SF^6(x) = poly'_4[a, SF^6(t)] = (\,1.71,\quad 0.67,\quad 0.50,\quad 2.00,\quad 1.00,\quad 0.00,\quad 0.00\,)$$

This result may be checked by hand: $x(1) = 1.708\ldots$, $\dot{x}(1) = (1 - t + t^2/2 + t^3/3!)_{t=1} = 0.666\ldots$, etc. Next suppose that we must pass x through a nonlinear block represented by a polynomial of degree three in x:

$$z = poly'_3(b, x) = 1 - 0.50x - 0.50x^2 + 0.1x^3$$

Then another call to the *poly* algorithm of degree three and coefficients

$$b = (1, -0.5, -0.5, 0.1)$$

gives the scalar form of z,

$$SF^6(z) = poly'_3[b, SF^6(x)] = (\,-0.82,\quad -0.89,\quad -0.66,\quad -2.46,\quad -0.38,\quad 7.17,\quad 30.31\,)$$

Thus, for example, the sixth time derivative of the output of the nonlinear block z, namely $z^{(6)}$, has the value of 30.31 at $t = 1$. The algorithm is shown as a block diagram in figure 2.2. Clearly the process could be continued if there were additional polynomial nonlinearities in the sequence, so multiple nesting of polynomials is easily handled by means of dynamic forms.

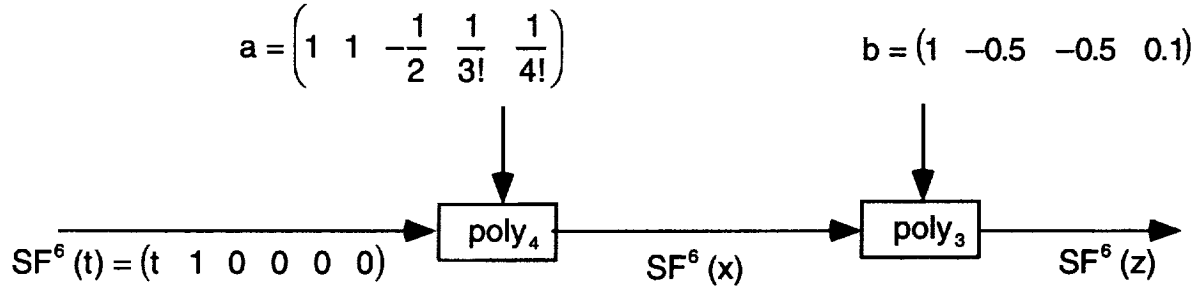

Figure 2.2. Scalar forms for the example of nested polynomials.

In the algorithm $poly_n^l$ the coefficients are fixed. We denote polynomials with time-variable coefficients by $poly_n$. Thus

$$z = poly_n(a, x) = a_0 + a_1 x + \ldots + a_n x^n \tag{2-22}$$

is a polynomial with possibly variable coefficients. The corresponding algorithm for dynamic forms is obtained from the $poly_n^l(a, x)$ algorithm in equation (2-20) by replacing not only z and x by their forms but also a_i by $SF^m(a_i)$ for $i = 0, \ldots, n$:

ALGORITHM: $SF^m(z) = poly_n[SF^m(a), SF^m(x)]$
$SF^m(z) = SF^m(a_n) \star SF^m(x)$
do $i = 1, n-1$
  $SF^m(z) = [SF^m(a_{n-i}) + SF^m(z)] \star SF^m(x)$
end do
$SF^m(z) = SF^m(a_0) + SF^m(z)$

Nonlinear functions of several variables are frequently given in practice in tabular form. Thus, for example, the subroutine generating the aircraft total force and moment may contain a table and interpolating routine representing the functional dependence of the moment coefficient on the angle of attack and the Mach number, $C_m = f(\alpha, Mach)$. Sometimes such tables may be represented by nested polynomials, which for two variables may take the following form:

$$z = f(x, y) = (b_{00} + \ldots + b_{0n}y^n) + (b_{10} + \ldots + b_{1n}y^n)x + \ldots + (b_{k0} + \ldots + b_{kn}y^n)x^k$$

The algorithm for computing m time derivatives of f is easily constructed:

ALGORITHM: $SF^m(z) = f[SF^m(x), SF^m(y)]$
do $i = 0, k$
$SF^m(a_i) = poly_n[b_i, SF^m(y)]$
end do
$SF^m(z) = poly_k[SF^m(a), SF^m(x)]$

where $b_i = (b_{i0}, \ldots, b_{in})$.

Thus we are beginning to compute easily time derivatives of fairly complicated functions. In the preceding example an algorithm was constructed from simpler algorithms that were built up from still simpler functions. At the bottom of such a hierarchy are functions from a basic set that already includes sum and product. Next, more functions are added to this set.

Inverse, Division, Fractional Powers, Logarithms, and Exponentials

Many standard functions may be converted to functions of scalar forms by means of the Leibnitz product rule. In this section we derive the algorithms for x^{-1}, x/y, $x^{a/b}$, e^x, and $\ln x$. Trigonometric functions $\cos x$, $\sin x$, and $\arctan(y, x)$ will be derived in the next section. For convenience, we shall denote, say, $SF^k[f(x, y, \ldots)]$ by $f[SF^k(x), SF^k(y), \ldots]$. Thus, for example,

$$\ln SF^m(x) = SF^m(\ln x) \triangleq \left(\ln x, (\ln x)^{(1)}, \ldots, (\ln x)^{(m)} \right) \tag{2-23}$$

The notation will allow such easy visual and machine translation as, for example,

$$z = e^{-ax^2 - by^2} \Rightarrow SF^m(z) = e^{-a[SF^m(x)]^2 - b[SF^m(y)]^2}$$

In the following discussion it is understood that for any x, $f[SF^m(x)]$ is defined only if $f(x)$ is defined.

If $z = x/y$ and $y \neq 0$, then, since $x = zy$, the time derivatives $z^{(k)}$ are given iteratively by the following algorithm:

$$\begin{cases} z^{(0)} = xy^{-1} \\ z^{(k)} = \left(x^{(k)} - \sum_{i=1}^{k} C_i^k z^{(k-i)} y^{(i)} \right) y^{-1} \quad \text{if } 0 < k \leq m \end{cases} \tag{2-24}$$

This division algorithm will be denoted as

$$SF^m(z) = SF^m(x)/SF^m(y) \tag{2-25}$$

It may be noted that the cancellation law holds for forms:

$$[SF^m(w) \star SF^m(x)]/[SF^m(w) \star SF^m(y)] = SF^m(x)/SF^m(y) \tag{2-26}$$

In particular, if $z = y^{-1}$, then, since $z = 1/y$, the division algorithm produces the inverse.

$$\begin{cases} z^{(0)} = y^{-1} \\ z^{(k)} = -\left(\sum_{i=1}^{k} C_i^k z^{(k-i)} y^{(i)} \right) y^{-1} \quad \text{if } 0 < k \leq m \end{cases} \tag{2-27}$$

The effect of this algorithm will be denoted as

$$SF^m(z) = [SF^m(x)]^{-1} \tag{2-28}$$

and referred to as the scalar form inverse. Note that

$$SF^m(x) \star [SF^m(x)]^{-1} = [SF^m(x)]^{-1} \star SF^m(x) = SF^m(1)$$

If $z = x^{a/b}$ for constant integers a, b with $b > 0$ and $x \neq 0$, then $bx\dot{z} = az\dot{x}$; hence, on application of the Leibnitz rule to both sides,

$$\sum_{i=0}^{k} C_i^k b x^{(i)} \dot{z}^{(k-i)} = \sum_{i=0}^{k} C_i^k a z^{(i)} \dot{x}^{(k-i)} \tag{2-29}$$

But since $\dot{z}^{(k-i)} = z^{(k-i+1)}$, and similarly for x,

$$\sum_{i=0}^{k} C_i^k b x^{(i)} z^{(k-i+1)} = \sum_{i=0}^{k} C_i^k a z^{(i)} x^{(k-i+1)} \tag{2-30}$$

Or, peeling off the first ($i = 0$) term, for $1 \leq k$,

$$bxz^{(k+1)} + \sum_{i=1}^{k} C_i^k b x^{(i)} z^{(k-i+1)} = az^{(0)} x^{(k+1)} + \sum_{i=1}^{k} C_i^k a z^{(i)} x^{(k-i+1)} \tag{2-31}$$

Consequently, we obtain the following basic algorithm

$$\begin{cases} z^{(0)} = x^{a/b} \\ z^{(1)} = (bx)^{-1} azx^{(1)} \\ z^{(k+1)} = (bx)^{-1} \{ az^{(0)} x^{(k+1)} + \sum_{i=1}^{k} C_i^k [az^{(i)} x^{(k-i+1)} - bx^{(i)} z^{(k-i+1)}] \} \quad \text{for } 1 \leq k \end{cases} \tag{2-32}$$

The effect of this algorithm will be denoted as

$$SF^m(z) = [SF^m(x)]^{a/b} \tag{2-33}$$

We note that

$$[SF^m(x)]^{a/b} \star [SF^m(x)]^{c/d} = [SF^m(x)]^{(a/b)+(c/d)} \tag{2-34}$$

If $z = x^n$, with $x \neq 0$ and $n > 0$, then

$$SF^m(z) = [SF^m(x)]^{n/1} \tag{2-35}$$

which will be simplified to

$$SF^m(z) = [SF^m(x)]^n \tag{2-36}$$

It may be noted that this algorithm involves only one Leibnitz convolution instead of n as in the simple algorithm discused previously.

If $z = |x|$ then, of course,

$$|SF^m(x)| = \begin{cases} SF^m(x) & \text{if } x \geq 0 \\ -SF^m(x) & \text{if } x < 0 \end{cases} \tag{2-37}$$

If $z = e^x$, then $\dot{z} = z\dot{x}$ and so, for $0 \leq k \leq m-1$, the Leibnitz product rule gives

$$z^{(k+1)} = \sum_{i=0}^{k} C_i^k z^{(k-i)} x^{(i+1)} \tag{2-38}$$

We denote this algorithm as

$$SF^m(z) = e^{SF^m(x)} \tag{2-39}$$

Conversely, if $z = \ln x$ for $x > 0$, then $z^{(0)} = \ln x$ and since $x\dot{z} = \dot{x}$, $z^{(1)} = x^{-1}x^{(1)}$ and for $1 \leq k \leq m-1$ the product formula leads to

$$z^{(k+1)} = x^{-1}\left(x^{(k+1)} - \sum_{i=1}^{k} C_i^k x^{(i)} z^{(k-i+1)}\right) \tag{2-40}$$

This algorithm will be denoted as

$$SF^m(z) = \ln SF^m(x) \tag{2-41}$$

It may be noted that

$$e^{\ln[SF^m(x)]} = SF^m(x) \tag{2-42}$$

Other functions may now be built up. For example, the scalar form of a Gaussian, $z = be^{-x^2}$, is computed easily by first calling the dynamic-form squarer, then the sign change, and then the exponential:

$$SF^m(z) = be^{-[SF^m(x)]^2} \tag{2-43}$$

Note how the notation keeps track of the calling sequence. In effect the result of each call has an independent existence. On the other hand, the algorithm $SF^m(z) = SF^m(be^{-x^2})$, while true overall, is not parsed as a calling sequence of independent subroutines.

Trigonometric Functions

To obtain trigonometric functions, it is convenient to group cos and sin as follows:

$$z = \begin{pmatrix} z_1 \\ \\ z_2 \end{pmatrix} = \begin{pmatrix} \cos x \\ \\ \sin x \end{pmatrix} \tag{2-44}$$

Then, since

$$\dot{z} = \dot{x} Q z \tag{2-45}$$

where

$$Q = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{2-46}$$

the derivatives of $\cos x$ and $\sin x$ are given by

$$z^{(k+1)} = \sum_{i=0}^{k} C_i^k x^{(k-i+1)} Q z^{(i)} \tag{2-47}$$

The algorithm (2-47) for the scalar form of the cos and sin functions will be denoted as

$$\begin{pmatrix} SF^m(z_1) \\ SF^m(z_2) \end{pmatrix} = \begin{pmatrix} \cos SF^m(x) \\ \sin SF^m(x) \end{pmatrix} \tag{2-48}$$

Conversely, suppose that z is redefined as

$$z = r \begin{pmatrix} \cos x \\ \sin x \end{pmatrix} \tag{2-49}$$

with $r > 0$ so that

$$x = \arctan(z_2, z_1) \tag{2-50}$$

Then the derivatives of x may be computed without assuming that the norm of z is 1, as follows: From equation (2-49), $r^2 = z^T z$ and, upon differentiating equation (2-49), $\dot{z} = \dot{r} r^{-1} z + \dot{x} Q z$. Premultiplying the last equation by $z^T Q$ and noting that $z^T Q z = 0$ and $QQ = -I$, we obtain

$$\dot{x} z^T z = \dot{z}^T Q z \tag{2-51}$$

Let

$$\begin{cases} u = z^T z = z_1 z_1 + z_2 z_2 \\ w = \dot{z}^T Q z = -\dot{z}_1 z_2 + \dot{z}_2 z_1 \end{cases} \tag{2-52}$$

Then

$$\dot{x} = w/u \tag{2-53}$$

Now we raise the order,

$$\begin{cases} SF^m(u) = SF^m(z_1) * SF^m(z_1) + SF^m(z_2) * SF^m(z_2) \\ SF^{m-1}(w) = -SF^{m-1}(\dot{z}_1) * SF^{m-1}(z_2) + SF^{m-1}(\dot{z}_2) * SF^{m-1}(z_1) \end{cases} \tag{2-54}$$

and

$$SF^{m-1}(\dot{x}) = SF^{m-1}(w)/SF^{m-1}(u) \tag{2-55}$$

Equations (2-50, 2-54, and 2-55) define the arctangent algorithm, which will be denoted as

$$SF^m(x) = \arctan[SF^m(z_2), SF^m(z_1)] \tag{2-56}$$

Of course, if it is known that $r = 1$ identically, that is $SF^m(u) = (1, 0)$, then

$$\begin{cases} x^{(0)} = \arctan(z_2, z_1) \\ x^{(k+1)} = w^{(k)}, \qquad 0 \le k \le m-1 \end{cases}$$

If a nonlinearity is modeled by a few terms of its Fourier series (here k is the fundamental wave number),

$$z = fourier_n(k, a, b, x) = a_0 + \sum_{i=1}^{n} a_i \cos(ikx) + b_i \sin(ikx) \tag{2-57}$$

then $SF^m(z)$ is given by

$$fourier_n[k, a, b, SF^m(x)] = a_0 SF^m(1) + \sum_{i=1}^{n} a_i \cos[ikSF^m(x)] + b_i \sin[ikSF^m(x)] \tag{2-58}$$

Functions of several variables can be treated similarly. Multivariable polynomials find use in modeling aircraft force and moment generators (ref. 24). Multidimensional Fourier series and other functions may be used for describing winds, terrain, or other fields that influence the aircraft. For example, if the generating function is a polynomial in p variables each of maximum degree n,

$$z = \sum_{i_1,\ldots,i_p}^{n} a(i_1, \ldots, i_p) x_1^{i_1} \ldots x_p^{i_p} \tag{2-59}$$

then $SF^m(z)$ may be evaluated (assuming all $x_i > 0$) by the following algorithm:

do $i = 1, p$
  $SF^m(y_i) = \ln SF^m(x_i)$
end do
$SF^m(z) = \sum_{i_1,\ldots,i_p}^{n} a(i_1, \ldots, i_p) e^{\sum_{j=1}^{p} i_j SF^m(y_j)}$

The following example illustrates the application of dynamic forms.

Example 2.2. As an example of the application of scalar forms, consider a nap-of-the-Earth (NOE) problem (ref. 25). A simple version of this problem may be stated as follows: Suppose that a plan view of the trajectory to be flown by the helicopter is given. The altitude must be maintained at a fixed, usually minimal, distance above the terrain. The problem is to determine whether the resulting trajectory is within the helicopter performance limits. The intermediate problem is to determine altitude and its time derivatives. Suppose that the horizontal velocity of a helicopter is given in polar coordinates by horizontal speed $\dot{s}(t)$ and heading $\psi(t)$. Then the plan view of the path evolves according to

$$\begin{pmatrix} \dot{x} \\ \\ \dot{y} \end{pmatrix} = \begin{pmatrix} \dot{s}\cos\psi \\ \\ \dot{s}\sin\psi \end{pmatrix}$$

and

$$\begin{pmatrix} x \\ \\ y \end{pmatrix} = \begin{pmatrix} x(0) + \int_0^t \dot{x} \\ \\ y(0) + \int_0^t \dot{y} \end{pmatrix}$$

where the initial condition $x(0)$, $y(0)$ is also given. Suppose, in addition, that there are some irregular vertical obstructions that are collectively covered by a Gaussian cap:

$$h = h_{max} e^{-(x/\sigma_x)^2 - (y/\sigma_y)^2}$$

where h is the altitude. The helicopter is required to stay on that cap while flying the given horizontal trajectory. The problem is to determine the vertical speed and the next four of its derivatives. It is easily done; simply replace the variables by their forms, as follows:

$$\begin{pmatrix} SF^4(\dot{x}) \\ \\ SF^4(\dot{y}) \end{pmatrix} = \begin{pmatrix} SF^4(\dot{s}) \star \cos SF^4(\psi) \\ \\ SF^4(\dot{s}) \star \sin SF^4(\psi) \end{pmatrix}$$

$$\begin{pmatrix} SF^5(x) \\ \\ SF^5(y) \end{pmatrix} = \left[\begin{pmatrix} x(0) + \int_0^t SF^0(\dot{x}) \\ \\ y(0) + \int_0^t SF^0(\dot{y}) \end{pmatrix}, \begin{pmatrix} SF^4(\dot{x}) \\ \\ SF^4(\dot{y}) \end{pmatrix} \right]$$

$$SF^5(h) = h_{max} e^{-(SF^5(x)/\sigma_x)^2 - (SF^5(y)/\sigma_y)^2}$$

The scalar form $SF^5(h)$ contains the altitude h and five of its time derivatives. These higher derivatives of h are needed for testing constraint satisfaction, such as acceleration $\ddot{h}$, which affects pitch angle and power requirements, $h^{(4)}$, which has a direct bearing on moment requirements, and $h^{(5)}$, which affects the usually limited control rates. Details of the force and moment model would be needed to compute the actual values of controls. Of course it is possible to compute other functions of scalar forms. For example, the total translational kinetic and potential energy divided by weight is

$$e = \frac{\dot{s}^2 + \dot{h}^2}{2g} + h$$

where g is the acceleration of gravity. The algorithm for computing translational power and power rate is obtained by simply replacing variables by their forms:

$$SF^4(e) = \frac{[SF^4(\dot{s})]^2 + [SF^4(\dot{h})]^2}{2g} + SF^4(h)$$

A numerical example is given in appendix A.

Partial Derivatives

We have developed in the preceding sections several functions of dynamic forms. With these functions as building blocks many other practically useful functions can be assembled. For example, suppose that there is an algorithm for computing

$$z = f(x, u) \tag{2-60}$$

for scalar z, x, u, and that this zero-order algorithm has been converted to order m:

$$SF^m(z) = f[SF^m(x), SF^m(u)] \tag{2-61}$$

as shown in figure 2.3. The input consists of the scalar forms $SF^m(x)$ and $SF^m(u)$, and the output is the scalar form $SF^m(z) = f[SF^m(x), SF^m(u)]$. Now we proceed to compute, using only this algorithm, several useful objects associated with f.

Consider the partial derivatives of f. Since $z = f(x, u)$,

$$\dot{z} = f_x \dot{x} + f_u \dot{u} \tag{2-62}$$

where f_x denotes the partial derivative of f with respect to x. Choose an input to be

$$\begin{cases} SF^m(x) = (\, x, 1\,) \\ \\ SF^m(u) = (\, u, 0\,) \end{cases} \tag{2-63}$$

where zero padding is assumed: $(\, x, 1\,) = (\, x, 1, 0, \ldots, 0\,)$ and $(\, u, 0\,) = (\, u, , 0, \ldots, 0\,)$. Then the value of the x-partial derivative of f at x is given by $\dot{z}$, that is, by the content in the number one location of $SF^m(z)$,

$$f_x(x, u) = \dot{z} = \{SF^m(z)\}_1 = \{f[(x, 1), (u, 0)]\}_1 \tag{2-64}$$

where we use the notation

$$\{SF^m(y)\}_k = \begin{cases} y^{(k)} & \text{if } 0 \le k \le m \\ \\ 0 & \text{else} \end{cases}$$

The purpose of equation (2-64) is not to replace a familiar and convenient notation by one that is obscure and awkward; the purpose is to show that the useful object, the x-partial derivative of f at (x, u), may be computed by means of scalar forms as the value in location one of the output of the scalar form algorithm f whose input is $(x, 1)$ and $(u, 0)$.

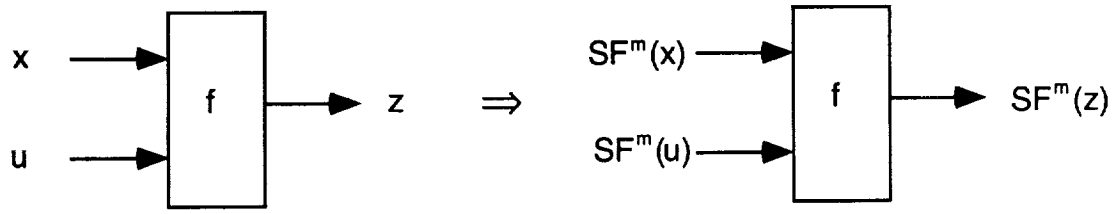

Figure 2.3. Algorithm for a composite function f.

The value of the k^{th} partial of f at x with respect to x is given by

$$f_{x\ldots x} = \{f[(x,1),(u,0)]\}_k, \quad 1 \le k \le m \tag{2-65}$$

Similarly,

$$f_{u\ldots u} = \{f[(x,0),(u,1)]\}_k, \quad 1 \le k \le m \tag{2-66}$$

Furthermore, since

$$\ddot{f} = f_{xx}\dot{x}\dot{x} + f_x\ddot{x} + 2f_{xu}\dot{x}\dot{u} + f_u\ddot{u} + f_{uu}\dot{u}\dot{u} \tag{2-67}$$

and

$$\begin{aligned} f_{xx} &= \{f[(x,1),(u,0)]\}_2 \\ f_{uu} &= \{f[(x,0),(u,1)]\}_2 \end{aligned} \tag{2-68}$$

it follows that

$$f_{xu} = \frac{1}{2}\{f[(x,1),(u,1)] - f[(x,1),(u,0)] - f[(x,0),(u,1)]\}_2 \tag{2-69}$$

Function Inverse

The algorithm for computing the dynamic form of the inverse of a function is often of practical interest. Suppose that an algorithm is given for computing a possibly time-varying function

$$z = f(x,u,t) \tag{2-70}$$

for scalar z, x, u and time t; that this zero-order algorithm has been raised to order m:

$$SF^m(z) = f[SF^m(x), SF^m(u), SF^m(t)] \tag{2-71}$$

and that we have the (partial) inverse f^{-1} of f

$$u = f^{-1}(x,z,t) \tag{2-72}$$

so that for all x, u, z, t

$$f[x, f^{-1}(x,z,t), t] = z \tag{2-73}$$

Often in practice f^{-1} is obtained numerically by an algorithm such as Newton–Raphson. The extension of $f^{-1}(x,z,t)$ to

$$SF^m(u) = f^{-1}[SF^m(x), SF^m(z), SF^m(t)] \tag{2-74}$$

producing not only u but also m of its time derivatives, may be computed as follows: The time derivative of equation (2-70) is

$$\dot{z} = f_x\dot{x} + f_u\dot{u} + f_t \tag{2-75}$$

But, according to equation (2-71),

$$\dot{z} = \{f[(x,\dot{x}),(u,\dot{u}),(t,1)]\}_1 = \{f[(x,\dot{x}),(u,0),(t,1)]\}_1 + f_u\dot{u} \tag{2-76}$$

Hence it follows that

$$\dot{u} = (f_u)^{-1}(\dot{z} - \{f[(x,\dot{x}),(u,0),(t,1)]\}_1) \tag{2-77}$$

where the inverse of the partial derivative f_u is

$$(f_u)^{-1} = \{f[(x,0),(u,1),(t,0)]\}_1^{-1} \tag{2-78}$$

Similarly,

$$\ddot{z} = \{f[(x,\dot{x},\ddot{x}),(u,\dot{u},0),(t,1,0)]\}_2 + f_u\ddot{u} \tag{2-79}$$

so that

$$\ddot{u} = (f_u)^{-1}\left(\ddot{z} - \{f[(x,\dot{x},\ddot{x}),(u,\dot{u},0),(t,1,0)]\}_2\right) \tag{2-80}$$

and for higher derivatives,

$$u^{(k)} = (f_u)^{-1}\left(z^{(k)} - \{f[(x,\ldots,x^{(k)}),(u,\ldots,u^{(k-1)},0),(t,1)]\}_k\right) \tag{2-81}$$

Thus, in addition to the base point $u = f^{-1}(x,z)$, only derivatives of f are needed for the construction of derivatives of f^{-1}. Hence the algorithm:

ALGORITHM: $SF^m(u) = f^{-1}[SF^m(x), SF^m(z)]$
$SF^m(u) = (f^{-1}(x,z,t),0,\ldots,0)$
$(f_u)^{-1} = \{f[(x,0),(u,1),(t,0)]\}_1^{-1}$
do $k = 1,m$
  $u^{(k)} = (f_u)^{-1}\left(z^{(k)} - \{f[SF^k(x), SF^k(u), SF^k(t)]\}_k\right)$
end do

That is, first $SF^m(u)$ is loaded with the base point $u = f^{-1}(x,z,t)$ and zero derivatives; then the inverse of f_u is computed as in equation (2-78); finally, equation (2-81) is iterated from 1 to m. The combined action of f^{-1} and f is shown in figure 2.4. The input is $SF^m(x)$, the desired evolution is $SF^m(z)$, and the required control is $SF^m(u)$. So far we have been considering static systems. Next we consider dynamic systems such as differential equations.

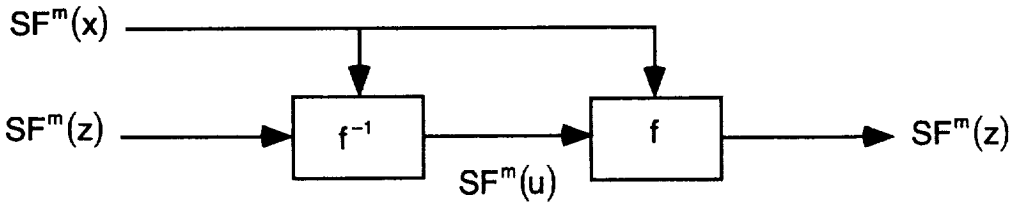

Figure 2.4. Useful factors of identity.

Differential Equations

Suppose that we are given a possibly time-varying, first-order scalar system

$$\dot{x} = f(x, u, t) \tag{2-82}$$

where x is the state and u the control, and we wish to compute derivatives of x up to order m given the derivatives of u. First, as described previously, we convert the zero-order algorithm

$$z = f(x, u, t)$$

to order m:

$$SF^m(z) = f[SF^m(x), SF^m(u), (t, 1)] \tag{2-83}$$

where, as before, $SF^m(t) = (t, 1, 0, \ldots, 0)$ is abbreviated as $(t, 1)$. Then

$$x^{(k+1)} = \{f[SF^k(x), SF^k(u), (t, 1)]\}_k \tag{2-84}$$

Consequently the time derivatives of x can be computed iteratively from the initial condition x and the control $SF^m(u)$:

ALGORITHM: $[SF^m(x)] = T_f[x, SF^m(u)]$
$SF^m(x) = (x, 0, \ldots, 0)$
do $k = 0, m$
  $x^{(k+1)} = \{f[SF^k(x), SF^k(u), (t, 1)]\}_k$
end do

That is, first $SF^m(x)$ is loaded with the base point x and zero derivatives; then the derivatives $x^{(k)}$ are obtained iteratively by means of equation (2-84). The effect of the algorithm will be denoted as

$$[SF^m(x)] = T_f[x, SF^m(u)] \tag{2-85}$$

where the subscript in T_f is a reminder that the function f raised to order m must be provided.

Now that we have $SF^m(x)$, we can approximate the solution $\phi(x, t+\tau)$ of equation (2-82) by the first $m+1$ terms of its Taylor series:

$$\phi(x, t+\tau) \approx tay[SF^m(x), \tau] = \sum_{k=0}^{m} x^{(k)} \tau^k / k! \tag{2-86}$$

The T_f algorithm may be easily generalized to scalar differential equations of higher order. Thus, suppose that in equation (2-82) $x \in R^n$, $u \in R$, and that we have zero-order algorithms

$$z_i = f_i(x_1, \ldots, x_n, u, t) \tag{2-87}$$

for $1 \leq i \leq n$ and these algorithms have been converted as discussed previously to

$$SF^m(z_i) = f_i[SF^m(x_1), \ldots, SF^m(x_n), SF^m(u), (t, 1)] \tag{2-88}$$

Then, since $x_i^{(k+1)} = z_i^{(k)}$ for $i = 1, \ldots, n$, the time derivatives of the n-dimensional state x may be constructed from the base point x and the control $SF^m(u)$ as follows:

ALGORITHM: $[SF^m(x)] = T_f[x, SF^{m-1}(u)]$
do $i = 1, n$
  $SF^m(x_i) = (x_i, 0, \ldots, 0)$
end do
do $k = 0, m$
  do $i = 1, n$
    $x_i^{(k+1)} = \{f_i[SF^m(x_1), \ldots, SF^m(x_n), SF^m(u), (t, 1)]\}_k$
  end do
end do

The input is the state x and the evolution of control $SF^m(u)$; the output is the evolution of the n state coordinates $SF^m(x_i)$. In the algorithm the base point x with zero derivatives is loaded into $SF^m(x)$; then the derivatives are computed by columns using f_i. It may be noted that, in general, the control u must be available to order $m-1$ for the computation of x_i to order m. An important special case occurs for pure feedback systems for which the function f above has the following triangular structure:

$$f_i(x_1, \ldots, x_n, u, t) = f_i(x_1, \ldots, x_{i+1}, t), \quad 1 \leq i < n \tag{2-89}$$

so that the state equation has the following form:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \ldots \\ \dot{x}_n \end{pmatrix} = \begin{pmatrix} f_1(x_1, x_2, t) \\ f_2(x_1, x_2, x_3, t) \\ f_3(x_1, x_2, x_3, x_4, t) \\ \ldots \\ f_n(x_1, x_2, \ldots, x_n, u, t) \end{pmatrix} \tag{2-90}$$

For such systems u to order $m-n$ is sufficient to compute x_i to order $m+1-i$. In particular, derivatives of u are not used for the computation of x_1 to order n, and u itself is not used in the computation of x_1 to order $n-1$. The computations in T_f for $n = m = 4$ may be represented as follows:

$\rightarrow$	$x_1^{(0)}$	$x_1^{(1)}$	$x_1^{(2)}$	$x_1^{(3)}$	$x_1^{(4)}$	$x_1^{(5)}$	$x_1^{(6)}$	...
x_1	•	*	*	*	*	o	o	...
x_2	•	*	*	*	o	o	o	...
x_3	•	*	*	o	o	o	o	...
x_4	•	*	o	o	o	o	o	...
u	•	o	o	o	o	o	o	...

where • denotes the initial data, and * and o denote significant and insignificant new entries, repectively. The computation flows from left to right starting with the supplied first column. The algorithm T_f

specialized to the case in which the output of the system is x_1 will be denoted as T_{1f}, that is,

$$SF^m(x_1) = T_{1f}[(x, SF^{m-n}(u)] \tag{2-91}$$

where, if $m < n$, then the entries in the scalar form of u are not used by the algorithm. The block diagram of T_{1f}, which we will refer to as the forward solution since $u(t)$ is the input and the Taylor series of $x_1(t)$ is the output, is shown in figure 2.5. The input consists of the base point x and the scalar form $SF^{m-n}(u)$ of control u. The output is the scalar form $SF^m(x_1)$ of the lowest state coordinate x_1. Only the first $m-n$ derivatives of u are used by the algorithm. If $m < n$, then u is not needed at all in the computation of $x^{(m)}$. This condition, in which no entries of a scalar form $SF^m(x)$ are used by the algorithm, will be denoted in diagrams by $SF^m(x) = \emptyset$.

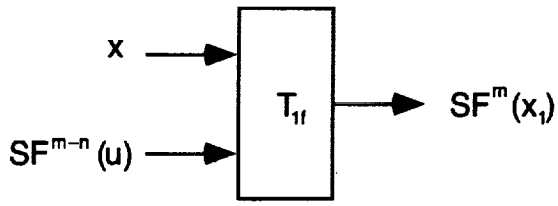

Figure 2.5. Forward solution.

Example 2.3. Suppose that $n = 4$, the state equation is

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} x_2 \\ a\sin(bx_1)x_2 + [2+\cos(ct)]x_3 \\ x_4 \\ u \end{pmatrix} \tag{2-92}$$

and a, b, c are constants. The first step is to raise f_i to order m as follows.

$$\begin{aligned} f_1 &= SF^m(x_2) \\ f_2 &= a\sin[bSF^m(x_1)] \star SF^m(x_2) + \{2+\cos[cSF^m(t)]\} \star SF^m(x_3) \\ f_3 &= SF^m(x_4) \\ f_4 &= SF^m(u) \end{aligned} \tag{2-93}$$

Then the forward algorithm T_{1f} is constructed.

ALGORITHM: $SF^m(x) = T_{1f}[x, SF^{m-4}(u)]$
do $i = 1, 4$
$SF^m(x_i) = (x_i, 0 \ldots, 0)$
end do
do $k = 0, m-1$
$x_1^{(k+1)} = \{SF^m(x_2)\}_k$
$x_2^{(k+1)} = \{a \sin[bSF^m(x_1)] \star SF^m(x_2) + \{2 + \cos[cSF^m(t)]\} \star SF^m(x_3)\}_k$
$x_3^{(k+1)} = \{SF^m(x_4)\}_k$
$x_4^{(k+1)} = \{SF^m(u)\}_k$
end do

This very simple example, for which even hand computation of derivatives is relatively easy, is intended only to illustrate the technique.

Dynamic Inverse

Consider again the first-order, scalar, dynamical system in equation (2-82). It is useful to be able to compute what the control signal must be so that the system output will track a given function of time. The evolution of control u that will produce the desired evolution of the output x may be obtained as follows: Construct the inverse $u = f^{-1}(x, z, t)$ of $z = f(x, u, t)$ so that

$$f[x, f^{-1}(x, z, t), t] = z$$

and raise its order to m by means of the function inverse algorithm as described previously.

$$SF^m(u) = f^{-1}[SF^m(x), SF^m(z), (t, 1)] \tag{2-94}$$

Then the control evolution $SF^m(u)$ that will produce the desired output $SF^m(x)$ is obtained by imposing the constraint $z = \dot{x}$ as given by the following algorithm:

ALGORITHM: $[x, SF^{m-n}(u)] = T_{1f}^{-1}[SF^m(x)]$
$SF^{m-1}(z) = SF^{m-1}(\dot{x})$
$SF^{m-1}(u) = f^{-1}[SF^{m-1}(x), SF^{m-1}(z), (t, 1)]$
$x = \{SF^m(x)\}_0$

That is, first $SF^m(x)$ is shifted to the right and loaded into $SF^{m-1}(z)$ to get

$$(z, \ldots, z^{(m-1)}) = (x^{(1)}, \ldots, x^{(m)})$$

Then a call to the f^{-1} algorithm produces $SF^{m-1}(u)$, and the state x is just the zero term of $SF^m(x)$.

This algorithm may be generalized to higher order differential equations, provided that they are of the pure feedback form as in equation (2-89), and that, for $1 \leq i < n$, each f_i is invertible with respect to x_{i+1} and f_n is invertible with respect to u (we denote these inverses by f_i^{-1}),

$$\begin{cases} x_{i+1} = f_i^{-1}(x_1, \ldots, z_i, t) & \text{for } 1 \leq i < n \\ u = f_i^{-1}(x_1, \ldots, z_{i+1}, t) & \text{for } i = n \end{cases} \tag{2-95}$$

Thus,

ALGORITHM: $[x, SF^{m-n}(u)] = T_{1f}^{-1}[SF^m(x_1)]$
do $k = 1, n-1$
$SF^{m-k}(z_k) = SF^{m-k}(\dot{x}_k)$
$SF^{m-k}(x_{k+1}) = f_k^{-1}[SF^{m-k}(x_1), \ldots, SF^{m-k}(x_k), SF^{m-k}(z_k), (t,1)]$
end do
$SF^{m-n}(z_{n+1}) = SF^{m-n}(\dot{x}_n)$
$SF^{m-n}(u) = f_n^{-1}[SF^{m-n}(x_1), \ldots, SF^{m-n}(x_n), SF^{m-n}(z_{n+1}), (t,1)]$
do $i = 1, n$
$x_i = \{SF^m(x_i)\}_0$
end do

Thus, the inversion is accomplished downward by rows until all n coordinates of x have been computed; then a call to f_n^{-1} produces the control and its derivatives; finally the base point x is assembled from the zero-order terms. The state must be at this value of x in order for the evolution $SF^m(x_1)$ to be possible. The flow in algorithm T_{1f}^{-1} for $n = m = 4$ may be represented as follows:

$\downarrow$	$x_1^{(0)}$	$x_1^{(1)}$	$x_1^{(2)}$	$x_1^{(3)}$	$x_1^{(4)}$	$x_1^{(5)}$	$x_1^{(6)}$	...
x_1	•	•	•	•	•	o	o	...
x_2	*	*	*	*	o	o	o	...
x_3	*	*	*	o	o	o	o	...
x_4	*	*	o	o	o	o	o	...
u	*	o	o	o	o	o	o	...

The computation proceeds downward from the initially supplied first row. In the usual application of linearization techniques to the control of pure feedback systems, $m = n$ so that only the control u is obtained. Cases with $m > n$ are of interest when derivatives of u are needed for path planning. The block diagram of this algorithm, which we will call the inverse solution since the input is $x_1(t)$ and the output is $u(t)$, is shown in figure 2.6. The input is the scalar form $SF^m(x_1)$ of the lowest state coordinate x_1. The output consists of the complete state base point $x = (x_1, x_2, \ldots, x_n)^T$ and the scalar form SF^{m-n} of the control variable u.

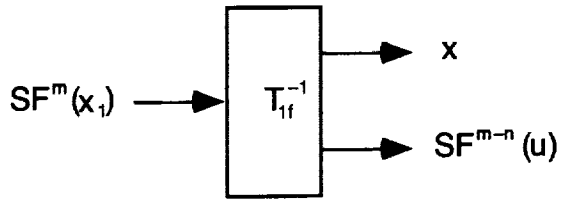

Figure 2.6. Inverse solution.

If $m < n$, then neither u nor any of its derivatives are computed.

Example 2.4. The inverse f_i^{-1} of f_i in example 2.3 is easily computed:

$$\begin{aligned} f_1^{-1} &= SF^m(z_1) \\ f_2^{-1} &= \{SF^m(z_2) - a\sin[bSF^m(x_1)] \star SF^m(x_2)\}/\{2 + \cos[cSF^m(t)]\} \\ f_3^{-1} &= SF^m(z_3) \\ f_4^{-1} &= SF^m(z_4) \end{aligned} \tag{2-96}$$

Equation (2-96) defines the inverse algorithm T_{1f}^{-1}:

ALGORITHM: $[x, SF^{m-4}(u)] = T_{1f}^{-1}[SF^m(x_1)]$
$SF^{m-1}(z_1) = SF^{m-1}(\dot{x}_1)$
$SF^{m-1}(x_2) = SF^{m-1}(z_1)$
$SF^{m-2}(z_2) = SF^{m-1}(\dot{x}_2)$
$SF^{m-2}(x_3) = \{SF^{m-2}(z_2) - a\sin[bSF^{m-2}(x_1)] \star SF^{m-2}(x_2)\}/\{2 + \cos[cSF^{m-2}(t)]\}$
$SF^{m-1}(z_3) = SF^{m-1}(\dot{x}_3)$
$SF^{m-3}(x_4) = SF^{m-3}(z_3)$
$SF^{m-1}(z_4) = SF^{m-1}(\dot{x}_4)$
$SF^{m-4}(u) = SF^{m-4}(z_4)$
do $i = 1, 4$
  $x_i = \{SF^m(x_i)\}_0$
end do

The example illustrates how easy it is by means of dynamic forms to propagate the derivatives backwards through the system.

It may be noted that T_{1f}^{-1} is the inverse of T_{1f} in the following sense. For any evolution $SF^m(x_1)$ of the output variable x_1,

$$T_{1f}\{T_{1f}^{-1}[SF^m(x_1)]\} = SF^m(x_1) \tag{2-97}$$

This significant relation is shown as a block diagram in figure 2.7 An application of this relation to automatic control is obtained by inserting the plant between T_{1f}^{-1} and T_{1f} as discussed next.

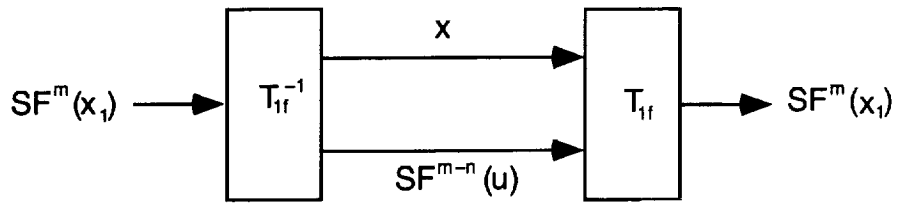

Figure 2.7. Useful factors of identity.

Consider the problem of tracking a given reference input in the presence of disturbances. Let the plant be described as follows: The state $x_p \in R^n$, the control $u_p \in R$, the output $y = x_{1p}$, and the state equation is possibly nonlinear and time-varying but of maximal relative degree (pure feedback form):

$$\dot{x}_p = f(x_p, u_p, t) = \begin{pmatrix} f_1(x_{1p}, x_{2p}, t) \\ f_2(x_{1p}, x_{2p}, x_{3p}, t) \\ \cdots \\ f_n(x_{1p}, x_{2p}, \ldots, x_{np}, u_p, t) \end{pmatrix} \tag{2-98}$$

The structure of a possible control-system design (ref. 26) is shown in figure 2.8. There are three subsystems in addition to the plant, namely transformer, regulator, and command generator. The command generator provides at every t not only the required value of the commanded output x_{1c} but also at least n of its derivatives, i.e., $SF^n(x_{1c})$. The plant state x_p is passed through T_{1f} to obtain $SF^{n-1}(x_{1p})$. Note that this step does not involve any differentiation of signals that are corrupted by noise. Furthermore, no element of $SF(u)$ is needed in this step. Next the error form is computed to order $n-1$ in the regulator by comparing the output form with the input form:

$$SF^{n-1}(e) = SF^{n-1}(x_{1p}) - SF^{n-1}(x_{1c}) \tag{2-99}$$

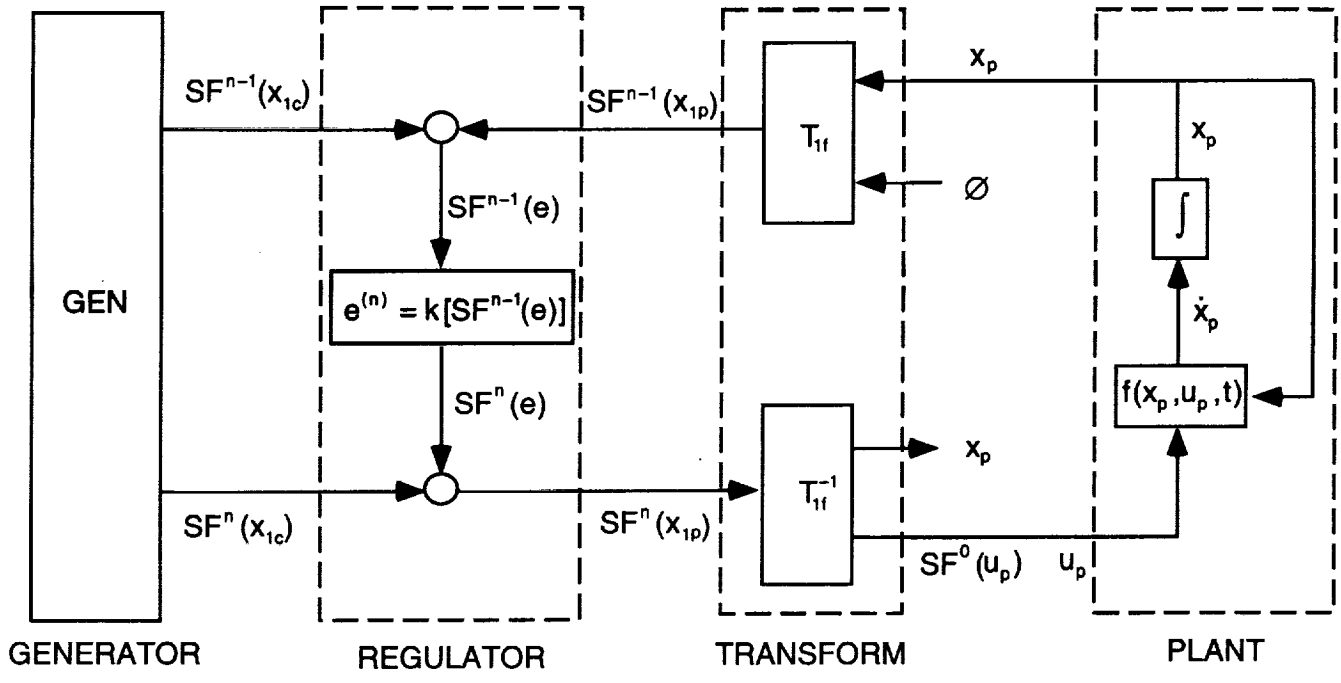

Figure 2.8. A structure for asymptotic trackers with pure feedback plant dynamics.

Then the order of the error form is raised by one by the regulator law k, and the error form is added to the reference, thereby raising the order by one of the desired output form:

$$SF^n(x_{1p}) = SF^n(x_{1c}) + SF^n(e) \tag{2-100}$$

Finally, the desired evolution of the output is passed through the inverter T_{1f}^{-1}, whose output $SF^0(u)$ drives the control u_p.

It may be noted that the transformations T_{1f}^{-1} and T_{1f} make the plant look like an (invariant) string of integrators, n integrators long. Thus, the plant has been transformed into a Brunowsky canonical form (ref. 14) with the Kronecker index equal to n, where $(x_{1p}, \ldots, x_{1p}^{(n-1)})$ is the transformed state and $x_{1p}^{(n)}$ is the transformed control. Consequently, the system may be regulated by means of a simple linear law

$$e^{(n)} = -\sum_{i=0}^{n-1} k_i e^{(i)} \tag{2-101}$$

despite changes in the perturbation model of f. In effect, the transformations T_{1f} and T_{1f}^{-1} provide automatically the necessary gain scheduling and therefore the system output x_{1p} will track asymptotically any input x_{1c} that is differentiable n times. The derivatives of the input must be available, and the plant dynamics must be of the pure feedback type. It should be noted, however, that only the state is needed and none of its derivatives. Noise is not an issue. The control algorithm denoted as C_f may be summarized as follows: It is assumed that C_f is called at the implementation sampling rate, which is sufficiently fast relative to the dynamics of the plant.

ALGORITHM: $u_p = C_f[x_p, SF^n(x_{1c})]$
$SF^{n-1}(x_{1p}) = T_{1f}(x_p, \emptyset)$
$SF^{n-1}(e) = SF^{n-1}(x_{1p}) - SF^{n-1}(x_{1c})$
$e^{(n)} = k[SF^{n-1}(e)]$
$SF^n(x_{1p}) = SF^n(x_{1c}) + SF^n(e)$
$[SF^0(u_p), x_p] = T_{1f}^{-1}[SF^n(x_{1p})]$
$u_p = \{SF^0(u_p)\}_0$

The input to the algorithm consists of the (estimated) plant state x_p and the generator command $SF^n(x_{1c})$. The output is the plant control u_p. In the algorithm the symbol $\emptyset$ indicates that the content of $SF^{n-1}(u)$ does not matter for this call of the subroutine T_{1f}. The output x_p (next to last line in the algorithm) is the plant state.

It may be noted that C_f is a general algorithm in the sense that it applies to any plant defined by f such that 1) f is of the pure feedback type and 2) f is a composite of the functions that have been elevated to order m. In such a case the zero-order function f can be automatically raised to order m, at which point all the steps in C_f become fixed. A numerical example illustrating the operation of control systems with the structure shown in figure 2.8 is given in appendix B.

In summary, we have shown that scalar forms may be used to organize effectively the design of automatic control systems for the class of plants having pure feedback dynamics f. The key step is to

raise the order of f from zero to n, but that step is made routine by the formalism of dynamic forms in which elementary functions and operations have been translated into their dynamic-form equivalents. In the remainder of the report we translate additional functions frequently occurring in the design of control systems. Further discussion of control algorithms, including plant dynamics with transmission zeros, will be presented in a future report.

3 VECTOR AND MATRIX FORMS

We are concerned mainly with three-dimensional vectors. Vectors and dot and cross products will be denoted as usual. Thus, $\vec{x}$ and $\vec{y}$ are vectors, and $\vec{x} \cdot \vec{y}$ and $\vec{x} \times \vec{y}$ are their dot and cross products. Right-handed orthonormal axis systems will be denoted by a double arrow. Thus $\overleftrightarrow{a}$ is an axis system $(\vec{a}_1 \quad \vec{a}_2 \quad \vec{a}_3)$ where, for $i, j = 1, 2, 3$, $\vec{a}_i \cdot \vec{a}_j = \delta_{ij}$ and $\vec{a}_3 = \vec{a}_1 \times \vec{a}_2$. The a-coordinates of $\vec{x}$ will be denoted by the column matrix x_a. Thus,

$$\vec{x} = \sum_{j=1}^{3} x_{aj} \vec{a}_j \tag{3-1}$$

the dot product $\vec{x} \cdot \vec{y} = x_a^T y_a$, and the cross product $\vec{z} = \vec{x} \times \vec{y}$

$$z_a = -S(x_a) y_a \tag{3-2}$$

where, for any $x \in R^3$, the skew symmetric matrix

$$S(x) = \begin{pmatrix} 0 & x_3 & -x_2 \\ -x_3 & 0 & x_1 \\ x_2 & -x_1 & 0 \end{pmatrix} \tag{3-3}$$

The transpose is denoted by $(\cdot)^T$ and δ_i denotes a column with 1 in row i and zeros elsewhere.

Next, consider vector forms. Let $\overleftrightarrow{a}$ be an axis system. The vector form of a vector $\vec{v}$ in $\overleftrightarrow{a}$ is defined by

$$VF^m(v_a) = \left(v_a, \dot{v}_a, \ldots, v_a^{(m)} \right) \tag{3-4}$$

where $v_a \in R^n$. The zero and unit forms will be denoted by

$$VF^m(0) = (0 \quad \ldots \quad 0) \tag{3-5}$$

and

$$VF^m(\delta_i) = (\delta_i \quad \ldots \quad 0) \tag{3-6}$$

where δ_i is a column of zeros except in row i, which contains 1. We denote the scalar form of the i^{th} component of a vector form $VF^m(v_a)$ with a subscript as follows:

$$SF^m(v_{ai}) = VF^m(v_a)_i = \left(v_{ai}, \dot{v}_{ai}, \ldots, v_{ai}^{(m)} \right) \tag{3-7}$$

Next consider matrices. A matrix form of order m is defined as a matrix and m of its time derivatives,

$$MF^m(A) = \left(A, \dot{A}, \ldots, A^{(m)} \right) \tag{3-8}$$

with $A \in R^n \times R^n$. Zero and identity are given by

$$MF^m(0) = (0 \quad 0 \quad \ldots \quad 0) \tag{3-9}$$

$$MF^m(I) = (I \quad 0 \quad \ldots \quad 0) \tag{3-10}$$

and the transpose

$$[MF^m(Z)]^T = MF^m(Z^T) \tag{3-11}$$

The scalar form corresponding to the z_{ij} element of Z will be denoted as

$$SF^m(z_{ij}) = MF^m(Z)_{ij} \tag{3-12}$$

Algebra

Several useful functions of vector forms and matrix forms follow easily either from already-defined functions of scalar forms or from the Leibnitz rule.

If $z_a = ax_a + by_a$ for either scalars or matrices a, b constant in time, then, of course,

$$VF^m(z_a) = aVF^m(x_a) + bVF^m(y_a) = VF^m(ax_a) + VF^m(by_a) \tag{3-13}$$

If $z_a = ax_a$ and a is a scalar function of time, then the derivatives of z_a are given by

$$z_a^{(k)} = \sum_{i=0}^{k} C_i^k a^{(k-i)} x_a^{(i)} \tag{3-14}$$

We denote the resulting algorithm as

$$VF^m(z_a) = SF^m(a) \star VF^m(x_a) \tag{3-15}$$

If $z = \vec{x} \cdot \vec{y}$, then

$$z^{(k)} = \sum_{i=0}^{k} C_i^k (x_a^{(k-i)})^T y_a^{(i)} \tag{3-16}$$

which will be denoted as

$$SF^m(z) = VF^m(x_a) \cdot VF^m(y_a) \tag{3-17}$$

If $\vec{z} = \vec{x} \times \vec{y}$, then

$$z_a^{(k)} = -\sum_{i=0}^{k} C_i^k S(x_a^{(k-i)}) y_a^{(i)} \tag{3-18}$$

which will be denoted as

$$VF^m(z_a) = VF^m(x_a) \times VF^m(y_a) \tag{3-19}$$

If $Z = aX$ for time varying scalar a and matrix X, then by the Leibnitz rule

$$Z^{(k)} = \sum_{i=0}^{k} C_i^k a^{(k-i)} X^{(i)} \tag{3-20}$$

So we have an algorithm for multiplication of a matrix form by a scalar form,

$$MF^m(Z) = SF^m(a) \star MF^m(X) \tag{3-21}$$

If $z_a = Xy_a$ with matrix X and vector y_a, both time-varying, then

$$z_a^{(k)} = \sum_{i=0}^{k} C_i^k X^{(k-i)} y_a^{(i)} \tag{3-22}$$

which is an algorithm for time-varying transformations to be denoted as

$$VF^m(z_a) = MF^m(X) \star VF^m(y_a) \tag{3-23}$$

If $Z = XY$ with time-dependent matrices X and Y, then, again by the Leibnitz rule,

$$Z^{(k)} = \sum_{i=0}^{k} C_i^k X^{(k-i)} Y^{(i)} \tag{3-24}$$

This algorithm will be denoted as

$$MF^m(Z) = MF^m(X) \star MF^m(Y) \tag{3-25}$$

If X is invertible, then in direct analogy to scalar forms the derivatives of X^{-1} are given by $Z^{(0)} = X^{-1}$, and for $1 \leq k \leq m$

$$Z^{(k)} = -\left(\sum_{i=1}^{k} C_i^k Z^{(k-i)} X^{(i)} \right) Z \tag{3-26}$$

So we give meaning to

$$MF^m(Z) = [MF^m(X)]^{-1} \tag{3-27}$$

It is now possible to construct higher level functions of dynamic forms. For example, if $x = |v_a| = [v_a^T v_a]^{1/2}$, then the evolution of the magnitude of v_a is given by the algorithm

$$SF^m(x) = |VF^m(v_a)| = [VF^m(v_a) \cdot VF^m(v_a)]^{1/2} \tag{3-28}$$

where the operations on the right side are all defined.

The unit vector form corresponding to $\vec{u}$ parallel to $\vec{v}$ in $\overleftrightarrow{a}$ is given by

$$VF^m(u_a) = VF^m(v_a)/|VF^m(v_a)| \tag{3-29}$$

Derivatives

Suppose that $x, z \in R^n$ and $z = f(x)$ and that we have raised the order of the vector function f from zero to m,

$$VF^m(z) = f[VF^m(x)] \tag{3-30}$$

Consider constant vectors x and y such that $y^T y = 1$. Let

$$z = f(x + ty)$$

Then the derivative of f at x in the direction of y at $t = 0$ is

$$\dot{z} = f_x(x)y$$

So by choosing

$$VF^m(x) = (x, y, 0, \ldots, 0) = (x, y)$$

we obtain the Taylor series expansion of f at x in direction y:

$$f(x + sy) = \sum_{k=0}^{m} z^{(k)} y s^k / k!$$

The n columns of the Jacobian matrix of f, $\partial f / \partial x$, are given by

$$\frac{\partial f}{\partial x} = (\{f[(x, \delta_1)]\}_1, \ldots, \{f[(x, \delta_n)]\}_1) \tag{3-31}$$

where we are again using the shorthand, for any vectors x, z

$$(x, z) = (x, z, 0, \ldots, 0)$$

and δ_i is a unit column along coordinate x_i. Thus n first-order calls of the vector-function algorithm produces its Jacobian matrix.

Examples

Scalar- and vector-form functions are useful as building blocks of higher level algorithms.

Example 3.1. Spherical and Cylindrical Coordinates. Consider the back and forth transformation between Cartesian coordinates and cylindrical coordinates and their m time derivatives. From the standard relations among coordinates,

$$x_r = \begin{pmatrix} r\cos\psi \\ r\sin\psi \\ z \end{pmatrix} \tag{3-32}$$

and conversely,

$$\begin{pmatrix} r \\ \psi \\ z \end{pmatrix} = \begin{pmatrix} \sqrt{x_{r1}^2 + x_{r2}^2} \\ \arctan(x_{r2}, x_{r1}) \\ x_{r3} \end{pmatrix} \tag{3-33}$$

It follows that for cylindrical coordinates the dynamic forms transform according to the algorithm $XY \prec CY$:

$$VF^m(x_r) = \begin{pmatrix} SF^m(r) \star \cos SF^m(\psi) \\ SF^m(r) \star \sin SF^m(\psi) \\ SF^m(z) \end{pmatrix} \tag{3-34}$$

and conversely $XY \succ CY$:

$$\begin{pmatrix} SF^m(r) \\ SF^m(\psi) \\ SF^m(z) \end{pmatrix} = \begin{pmatrix} [VF^m(x_r)_1^2 + VF^m(x_r)_2^2]^{1/2} \\ \arctan[VF^m(x_r)_2, VF^m(x_r)_1] \\ VF^m(x_r)_3 \end{pmatrix} \tag{3-35}$$

The conversion to spherical coordinates is similarly established. From the standard relations between coordinates, where γ, ψ are the latitude and longitude angles, respectively,

$$x_r = \begin{pmatrix} \rho \cos\gamma \cos\psi \\ \rho \cos\gamma \sin\psi \\ -\rho \sin\gamma \end{pmatrix} \tag{3-36}$$

and, conversely,

$$\begin{pmatrix} \rho \\ \psi \\ \gamma \end{pmatrix} = \begin{pmatrix} |x_r| \\ \arctan(x_{r2}, x_{r1}) \\ \arctan(-x_{r3}, \cos\psi x_{r1} + \sin\psi x_{r2}) \end{pmatrix} \tag{3-37}$$

It follows that for spherical coordinates the dynamic forms transform according to the algorithm $XP \prec SP$:

$$VF^m(x_r) = \begin{pmatrix} SF^m(\rho) \star \cos SF^m(\gamma) \star \cos SF^m(\psi) \\ SF^m(\rho) \star \cos SF^m(\gamma) \star \sin SF^m(\psi) \\ -SF^m(\rho) \star \sin SF^m(\gamma) \end{pmatrix} \tag{3-38}$$

and, conversely, $XP \succ SP$:

$$\begin{pmatrix} SF^m(\rho) \\ SF^m(\psi) \\ SF^m(\gamma) \end{pmatrix} = \begin{pmatrix} |SF^m(x_r)| \\ \arctan[VF^m(x_r)_2, VF^m(x_r)_1] \\ \arctan[-VF^m(x_r)_3, \cos SF^m(\psi) \star VF^m(x_r)_1 + \sin SF^m(\psi) \star VF^m(x_r)_2] \end{pmatrix} \tag{3-39}$$

Thus we have the following bidirectional link between the Cartesian (XY), spherical (SP) and cylindrical (CY) coordinates.

$$SP \iff XY \iff CY$$

Of course, now it is possible to concatenate: the algorithm

$$CY \succ SP = XY \succ SP(XY \prec CY) \tag{3-40}$$

transforms cylindrical to spherical coordinates and their m derivatives by calling first the algorithm $XY \prec CY$ and then algorithm $XY \succ SP$.

Example 3.2. Air Velocity. Suppose that the cylindrical coordinates of the wind field depend only on altitude, that is, the xy-magnitude, the xy-direction, and the vertical components are given by

$$\begin{pmatrix} v^w \\ \psi^w \\ w^w \end{pmatrix} = w(h)$$

Then the Cartesian coordinates of the wind evolve according to

$$VF^m(w_r) = XY \prec CY\{w[SF^m(h)]\}$$

If an aircraft flies along a trajectory $VF^{m+1}(x_r)$, then its altitude evolves according to (for aircraft, by convention, the z-axis points down)

$$SF^m(h) = -VF^m(x_r)_3$$

and its air velocity

$$VF^m(v_r^a) = VF^m(\dot{x}_r) - VF^m(w_r)$$

Its airspeed v^a, glidepath angle γ^a, and heading angle ψ^a, as well as their time derivatives up to order m, are given by

$$\begin{pmatrix} SF^m(v^a) \\ SF^m(\psi^a) \\ SF^m(\gamma^a) \end{pmatrix} = XY \succ SP[VF^m(v_r^a)]$$

Next, we consider rotations.

4 ROTATIONAL FORMS

Rotational forms are useful for describing the attitude of a rigid body, its angular velocity, and its derivatives.

Let $\overleftrightarrow{a}$ and $\overleftrightarrow{b}$ be two arbitrary (right-handed orthonormal) axis systems. The $\overleftrightarrow{a}$ coordinates of $\overleftrightarrow{b}$ will be denoted by the matrix C_{ba}. Thus, for $i = 1, 2, 3$,

$$\vec{b}_i = \sum_{j=1}^{3} C_{ba}(i,j)\vec{a}_j \tag{4-1}$$

Consequently, the coordinates of a vector $\vec{x}$ transform as follows:

$$x_b = C_{ba}x_a \tag{4-2}$$

Since $C_{ba}(i,j) = \vec{b}_i \cdot \vec{a}_j$, C_{ba} is a direction cosine matrix of $\overleftrightarrow{b}$ relative to $\overleftrightarrow{a}$. The j^{th} column of C_{ba} gives the b-coordinates of $\vec{a}_j$; the i^{th} row, the a-coordinates of $\vec{b}_i$. A direction cosine matrix is orthogonal so that its inverse is given by its transpose

$$C_{ba}^{-1} = C_{ba}^T = C_{ab} \tag{4-3}$$

Since $\overleftrightarrow{a}$ and $\overleftrightarrow{b}$ are both right handed, the determinant

$$\det C_{ba} = +1 \tag{4-4}$$

So, C_{ba} always has an eigenvalue of $+1$. The corresponding eigenvector is given by $axis(C_{ba})$ (provided $axis(C_{ba}) \neq 0$) where, for any 3×3 matrix C, the axis function

$$axis(C) = \frac{1}{2}\begin{pmatrix} c_{23} - c_{32} \\ c_{31} - c_{13} \\ c_{12} - c_{21} \end{pmatrix} \tag{4-5}$$

It may be noted that for any three-dimensional x,

$$axis[S(x)] = x \tag{4-6}$$

and for any 3×3 matrix C,

$$S[axis(C)] = 1/2(C - C^T) \tag{4-7}$$

We shall denote the set of all rotations as well as all 3×3 direction cosine matrices by $SO(3)$.

Suppose that C_{ba} is a function of time. Then, since $C_{ba}C_{ba}^T = I$, it follows that $\dot{C}_{ba}C_{ba}^T + C_{ba}\dot{C}_{ba}^T = 0$. Therefore, $\dot{C}_{ba}C_{ba}^T$ is skew symmetric, so let

$$\dot{C}_{ba}C_{ba}^T = S(\omega_{bab}) \tag{4-8}$$

Consequently,

$$\omega_{bab} = axis(\dot{C}_{ba}C_{ba}^T) \tag{4-9}$$

and

$$\dot{C}_{ba} = S(\omega_{bab})C_{ba} \tag{4-10}$$

The 3×1 column matrix ω_{bab} gives the $\overleftrightarrow{b}$ coordinates of the angular velocity of $\overleftrightarrow{b}$ relative to $\overleftrightarrow{a}$. The last subscript in ω_{bab} denotes the coordinate system. Thus, $\omega_{baa} = C_{ab}\omega_{bab}$ gives the $\overleftrightarrow{a}$ coordinates of the same velocity. On the other hand, $\omega_{aba} = -\omega_{baa}$ are the $\overleftrightarrow{a}$ coordinates of the angular velocity of $\overleftrightarrow{a}$ relative to $\overleftrightarrow{b}$. Also from equation (4-2)

$$\dot{x}_b = \dot{C}_{ba}x_a + C_{ba}\dot{x}_a = S(\omega_{bab})C_{ba}x_a + C_{ba}\dot{x}_a = S(\omega_{bab})x_b + C_{ba}\dot{x}_a$$

Thus, we obtain the Coriolis derivative, which computes the derivatives of the $\overleftrightarrow{b}$-coordinates of $\vec{x}$ from the derivatives of the $\overleftrightarrow{a}$-coordinates of $\vec{x}$ and the angular velocity of $\overleftrightarrow{b}$ relative to $\overleftrightarrow{a}$:

$$\dot{x}_b = S(\omega_{bab})x_b + C_{ba}\dot{x}_a \tag{4-11}$$

The matrix form corresponding to the direction cosine matrix C_{ba} locating $\overleftrightarrow{b}$ relative to $\overleftrightarrow{a}$ will be defined by MF_{ba}^m, that is,

$$MF_{ba}^m = \left(C_{ba}, \dot{C}_{ba}, \ldots, C_{ba}^{(m)}\right) \tag{4-12}$$

Often angular velocity and its time derivatives are of interest. Hence, we introduce another type of dynamic forms, namely, rotational forms. The rotational form of order m for attitude C_{ba} will be defined as the direction cosine matrix and angular velocity together with its time derivatives:

$$RF_{ba}^m = \begin{cases} C_{ba} & m = 0 \\ \left(C_{ba}, VF^{(m-1)}(\omega_{bab})\right) & m > 0 \end{cases} \tag{4-13}$$

where attitude $C_{ba} \in SO(3)$ and $\omega_{bab} \in R^3$ gives the $\overleftrightarrow{b}$-coordinates of the angular velocity of $\overleftrightarrow{b}$ relative to $\overleftrightarrow{a}$.

Algebra

The identity is given by

$$RF_{aa}^m = (I, 0) \tag{4-14}$$

Consider the transformation $MF \prec RF$ constructing the matrix form that corresponds to a given rotational form. Since $\dot{C}_{ba} = S(\omega_{bab})C_{ba}$, $MF^m(C_{ba})$ is given iteratively by

$$C_{ba}^{(k+1)} = \sum_{i=0}^{k} C_i^k S(\omega_{bab}^{(k-i)})C_{ba}^{(i)} \tag{4-15}$$

Conversely, it follows from

$$\omega_{bab} = axis(\dot{C}_{ba}C_{ba}^T)$$

that the function $MF \succ RF$ making the rotational form corresponding to a matrix form is given iteratively by

$$\omega_{bab}^{(k+1)} = \sum_{i=0}^{k} C_i^k axis(C_{ba}^{(k-i+1)} C_{ab}^{(i)}) \tag{4-16}$$

Note that C_{ab} is the transpose of C_{ba}. Thus, we have the pair,

$$\begin{cases} RF_{ba}^m = MF \succ RF(MF_{ba}^m) \\ MF_{ba}^m = MF \prec RF(RF_{ba}^m) \end{cases} \tag{4-17}$$

Multiplication of rotational forms may be defined by

$$RF_{cb}^m \star RF_{ba}^m = MF \succ RF[MF \prec RF(RF_{cb}^m) \star MF \prec RF(RF_{ba}^m)] \tag{4-18}$$

and inversion by

$$(RF_{ba}^m)^{-1} = RF_{ab}^m = MF \succ RF\{[MF \prec RF(RF_{ba}^m)]^T\} \tag{4-19}$$

where the transpose acts on a matrix form and so it is already defined. Consider a sequence of rotations. Suppose that there are three coordinate systems, $\overleftrightarrow{a}$, $\overleftrightarrow{b}$, and $\overleftrightarrow{c}$. The rotation of $\overleftrightarrow{c}$ relative to $\overleftrightarrow{a}$ is given in terms of rotations $\overleftrightarrow{c}$ relative to $\overleftrightarrow{b}$ and $\overleftrightarrow{b}$ relative to $\overleftrightarrow{a}$ by the product of forms,

$$RF_{ca}^m = RF_{cb}^m \star RF_{ba}^m \tag{4-20}$$

It may be noted that the rotation and matrix forms behave algebraically just as their generating direction cosine matrix.

A generalization of the Coriolis derivative (4-11) may be obtained as follows. Let $VF^m(x_r)$ be the vector form describing the motion of vector $\vec{x}$ to order m with respect to $\overleftrightarrow{r}$. It and its time derivatives up to order m with respect to another set of axes, $\overleftrightarrow{b}$, which is rotating according to RF_{br}^m, are given by

$$VF^m(x_b) = RF_{br}^m \star VF^m(x_r) = MF \prec RF(RF_{br}^m) \star VF^m(x_r) \tag{4-21}$$

Example 4.1. Angular Acceleration of Aircraft Stability Axes. As an application of the algorithms just developed, consider the following problem, which may occur in the synthesis of reference trajectories for an aircraft. Suppose that the velocity of the aircraft relative to the runway $\overleftrightarrow{r}$ (assumed to be inertially fixed) is given as a function of time, $v_r(t)$. Consider an axis system $\overleftrightarrow{v}$ aligned so that $\vec{v}_1$ is parallel to the velocity $\vec{v}$ and the total nongravitational force is in the $\vec{v}_1$–$\vec{v}_3$ plane with negative projection on $\vec{v}_3$. Find the direction cosine matrix C_{vr}, locating $\overleftrightarrow{v}$ relative to $\overleftrightarrow{r}$ as well as the angular velocity ω_{vrv}, and its two time derivatives and the force f_v (in g's) and its derivatives. We translate the problem in the usual way as follows (g is the acceleration of gravity).

$$\vec{v}_1 = \vec{v}/|\vec{v}|$$

$$\vec{f} = g^{-1}\dot{\vec{v}} - \vec{r}_3$$

$$\vec{v}_2 = \vec{v}_1 \times \vec{f}/|\vec{v}_1 \times \vec{f}|$$

$$\vec{v}_3 = \vec{v}_1 \times \vec{v}_2$$

$$C_{vr} = \begin{pmatrix} v_{1r}^T \\ v_{2r}^T \\ v_{3r}^T \end{pmatrix}$$

Now, in order to get $(\omega_{vrv}, \dot{\omega}_{vrv}, \ddot{\omega}_{vrv})$, three time derivatives of these expressions must be taken. Without dynamic forms this task is difficult, but with dynamic forms it is easy. We simply replace any implicit or explicit variable of time in the above set of equations with the corresponding forms in $\overleftrightarrow{r}$:

$$VF^3(v_{1r}) = VF^3(v_r)/|VF^3(v_r)|$$

$$VF^3(f_r) = g^{-1}VF^3(\dot{v}_r) - VF^3(\delta_3)$$

$$VF^3(v_{2r}) = VF^3(v_{1r}) \times VF^3(f_r)/|VF^3(v_{1r}) \times VF^3(f_r)|$$

$$VF^3(v_{3r}) = VF^3(v_{1r}) \times VF^3(v_{2r})$$

$$MF^3_{vr} = \begin{pmatrix} VF^3(v_{1r})^T \\ VF^3(v_{2r})^T \\ VF^3(v_{3r})^T \end{pmatrix}$$

$$RF^3_{vr} = RF \prec MF(MF^3_{vr})$$

The coding is done! The direction cosine matrix, the angular velocity and its two derivatives can now be read off RF^3_{vr} (see appendix C).

The example shows how dynamic forms may be used easily to obtain time derivatives to arbitrary order from complicated expressions involving functions of scalars, vectors, and matrices. The dynamic forms keep track of many variables and many detailed computations. In the next two sections, we consider Euler angle forms and Euler parameter forms and derive the appropriate transformations. Euler angles and parameters are discussed in reference 1.

Euler Angle Form

In this section we develop the direct and inverse functions relating Euler angles and m of their time derivatives to the corresponding (direction cosine) matrix form. Suppose that C_{ba} is a function of time and that $\omega_{bab} = u$, a constant unit vector. Then equation (4-10) becomes

$$\dot{C}_{ba} = S(u)C_{ba} \tag{4-22}$$

Consider this to be a differential equation with $C_{ba}(0) = I$. It is linear with constant coefficients, so the solution at $t = \phi$ is

$$C_{ba}(\phi) = e^{S(u)\phi} \tag{4-23}$$

But

$$e^{S(u)\phi} = I + S(u)\phi + S^2(u)\phi^2/2 + \ldots \tag{4-24}$$

which becomes, on repeated application of the identity $S^3(u) = -S(u)$,

$$e^{S(u)\phi} = I + \sin\phi S(u) + (1 - \cos\phi)S^2(u) = \cos\phi I + \sin\phi S(u) + (1 - \cos\phi)uu^T \tag{4-25}$$

Note that, since $S(u)u = 0$,

$$e^{S(u)\phi}u = u \tag{4-26}$$

that u is the Euler axis of C_{ba}, that

$$axis(e^{S(u)\phi}) = \sin\phi u \tag{4-27}$$

and that the trace

$$tr(e^{S(u)\phi}) = 1 + 2\cos\phi \tag{4-28}$$

According to the Euler theorem on rotations, any attitude C_{br} may be achieved (parameterized) by a single rotation from I about an axis u through an angle ϕ.

An elementary Euler rotation about axis δ_q for $q = 1, 2, 3$ through angle α is given by

$$E_q(\alpha) = e^{S(\delta_q)\alpha} = \cos\alpha I + \sin\alpha S(\delta_q) + (1 - \cos\alpha)\delta_q\delta_q^T \tag{4-29}$$

or explicitly:

$$E_1(\alpha) = e^{S(\delta_1)\alpha} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix} \tag{4-30}$$

$$E_2(\alpha) = e^{S(\delta_2)\alpha} = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix} \tag{4-31}$$

$$E_3(\alpha) = e^{S(\delta_3)\alpha} = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4-32}$$

The nine possible columns are given by

$$E_q(\alpha)\delta_j = \begin{cases} \delta_j & \text{if } j = q \\ \cos\alpha\delta_j + \sin\alpha S(\delta_q)\delta_j & \text{else} \end{cases} \tag{4-33}$$

The following index functions for the index set $\{1, 2, 3\}$ will be used to simplify notation.

For $i \neq j$

$$\nu(i, j) = 6 - i - j \tag{4-34}$$

$$\mu(i, j) = i - (j \bmod 3) \tag{4-35}$$

and

$$h(i, j) = \begin{cases} \mu(i, j) & \text{if } \mu(i, j) \neq 2 \\ -1 & \text{if } \mu(i, j) = 2 \end{cases} \tag{4-36}$$

Then, for example,

$$S(\delta_i)\delta_j = h(i, j)\delta_{\nu(i,j)} \tag{4-37}$$

The function $\nu(i, j)$ returns the integer that is different from either i or j. The function $h(i, j)$ returns the sign of the projection of $S(\delta_i)\delta_j$ on $\delta_{\nu(i,j)}$. It may be noted that

$$h(j, i) = -h(i, j) \tag{4-38}$$

With the aid of these index functions equation (4-33) may be changed to the following more convenient form

$$E_q(\alpha)\delta_j = \begin{cases} \delta_j & \text{if } j = q \\ \cos\alpha\delta_j + h(q, j)\sin\alpha\delta_{\nu(q,j)} & \text{else} \end{cases} \tag{4-39}$$

Therefore, if $C = E_q(\alpha)$, the elements of C are given by

$$C_{ij} = \begin{cases} \delta_{ij} & \text{if } j = q \\ \cos\alpha\delta_{ij} + h(q, j)\sin\alpha\delta_{i\nu(q,j)} & \text{else} \end{cases} \tag{4-40}$$

where axes subscripts on the direction cosine matrix C have been dropped temporarily to simplify notation. It is easy to raise this algorithm to order m:

ALGORITHM: $MF^m(C) = E_q[SF^m(\boldsymbol{\alpha})]$

do $i = 1,3$

do $j = 1,3$

$$MF^m(C)_{ij} = \begin{cases} SF^m(\delta_{ij}) & \texttt{if } j = q \\ \cos SF^m(\boldsymbol{\alpha})\delta_{ij} + h(q,j)\sin SF(\boldsymbol{\alpha})\delta_{i\nu(q,j)} & \texttt{else} \end{cases}$$

end do

end do

Next consider sequences of elementary rotations. Let C be a composite of three elementary rotations, with $q_1 \neq q_2 \neq q_3$, that is, an Euler sequence

$$C = E_{q_1}(\alpha_1)E_{q_2}(\alpha_2)E_{q_3}(\alpha_3) \tag{4-41}$$

Note the usual reverse order. The first rotation in the sequence is $E_{q_3}(\alpha_3)$: it is about axis q_3 through angle α_3. If $q_1 = q_3$, we will refer to the Euler sequence as repeating; if $q_1 \neq q_3$, nonrepeating. The elements of C may be computed by the indicated matrix product in equation (4-41). See page 20 of reference 1 for an expanded view of equation (4-41) for the twelve possible sequences.

Example 4.2. Normally in flight control the attitude of the aircraft body axes $\overleftrightarrow{b}$ relative to the runway axes $\overleftrightarrow{r}$ may be represented by the $(1,2,3)$ Euler sequence:

$$C_{br} = E_1(\phi)E_2(\theta)E_3(\psi) \tag{4-42}$$

Alternatively, aircraft attitude may be represented by a redundant sequence:

$$C_{br} = E_2(\alpha)E_3^T(\beta)E_1(\sigma)E_1(\phi_v)E_2(\gamma_v)E_3(\psi_v) \tag{4-43}$$

The first two rotations (γ_v, ψ_v) define the wind axes $\overleftrightarrow{w}$, in which $\vec{w}_1$ is along the relative-velocity vector $\vec{v}_a = \vec{v} - \vec{w}$, where $\vec{v}$ is the aircraft velocity, $\vec{w}$ is the wind velocity, and $\vec{w}_2$ is horizontal, that is, $\vec{w}_2 \cdot \vec{r}_3 = 0$. The roll through angle ϕ_v defines the aircraft stability axes $\overleftrightarrow{v}$. The body axes are reached by an additional corrective roll σ, sideslip β, and angle of attack α. The aerodynamic forces and moments are typically given as functions of airspeed $|v_a|$, α, β, and other variables.

Consider again equation (4-41). Since the algorithms for $E_q[SF^m(\alpha)]$ and $\star$ have already been constructed, the zero-order algorithm (4-41) may be raised to order m:

$$MF^m(C) = E_{q_1}[SF^m(\alpha_1)] \star E_{q_2}[SF^m(\alpha_2)] \star E_{q_3}[SF^m(\alpha_3)] \tag{4-44}$$

This algorithm will be denoted as

$$MF^m(C) = MF \prec AF[AF_q^m(\alpha)] \tag{4-45}$$

where the Euler angle form

$$AF_q^m(\alpha) = \begin{pmatrix} VF^m(\alpha) \\ \\ q \end{pmatrix} \tag{4-46}$$

where the three angles are represented by a column,

$$\alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} \tag{4-47}$$

and where the rotation sequence is represented by a row,

$$q = (q_1, q_2, q_3) \tag{4-48}$$

Conversely, consider the inverse problem of computing the Euler angles from the direction cosine matrix. Consider first the middle angle α_2. From the defining equation (4-41) it follows that

$$E_{q_2}(\alpha_2) = E_{q_1}(-\alpha_1) C E_{q_3}(-\alpha_3) \tag{4-49}$$

and, since δ_{q1} and δ_{q_3} are eigenvectors of E_{q_1} and E_{q_3}, respectively, that

$$\delta_{q_1}^T E_{q_2}(\alpha_2)\delta_{q_3} = \delta_{q_1}^T [\cos\alpha_2 \delta_{q_3} + h(q_2, q_3) \sin\alpha_2 \delta_{\nu(q_2,q_3)}] = \delta_{q_1}^T C \delta_{q_3} \tag{4-50}$$

If $q_1 = q_3$, then, since $\delta_{q_1}^T \delta_{q_3} = 1$ and $\delta_{q_1}^T \delta_{\nu(q_2,q_3)} = 0$, let

$$\begin{cases} x_2 = C_{q_1 q_1} \\ y_2 = \sqrt{1 - x_2^2} \end{cases} \tag{4-51}$$

If $q_1 \neq q_3$, then, since $\delta_{q_1}^T \delta_{q_3} = 0$ and $\delta_{q_1}^T \delta_{\nu(q_2,q_3)} = 1$, let

$$\begin{cases} y_2 = h(q_2, q_3) C_{q_1 q_3} \\ x_2 = \sqrt{1 - y_2^2} \end{cases} \tag{4-52}$$

Then, in either case,

$$\alpha_2 = \arctan(y_2, x_2) \tag{4-53}$$

The range of α_2 is given by

$$R_2 = \begin{cases} \{0 < \alpha_2 < \pi\}, & \text{if } q_1 = q_3 \\ \{-\pi/2 < \alpha_2 < \pi/2\}, & \text{if } q_1 \neq q_3 \end{cases} \tag{4-54}$$

The solution corresponding to the negative square root is

$$\alpha_2^* = \begin{cases} -\alpha_2 & \text{if } q_1 = q_3 \\ -\alpha_2 + \pi & \text{if } q_1 \neq q_3 \end{cases} \tag{4-55}$$

The range α_2^* is given by

$$R_2^* = \begin{cases} \{-\pi < \alpha_2^* < 0\}, & \text{if } q_1 = q_3 \\ \{\pi/2 < \alpha_2^* \leq \pi\} \cup \{-\pi < \alpha_2^* < -\pi/2\}, & \text{if } q_1 \neq q_3 \end{cases} \tag{4-56}$$

Conditions on the boundary separating regions R_2 and R_2^* are known in practice as gimbal lock.

Next consider angle α_1. From the defining equation (4-41) it follows that

$$E_{q_2}(\alpha_2)E_{q_3}(\alpha_3) = E_{q_1}^T(\alpha_1)C \tag{4-57}$$

so that

$$\delta_{q_2}^T E_{q_2}(\alpha_2)E_{q_3}(\alpha_3)\delta_{q_3} = 0 = \delta_{q_2}^T E_{q_1}^T(\alpha_1)C\delta_{q_3} \tag{4-58}$$

Equation (4-39) may be used to simplify the premultiplier of the last term:

$$0 = \delta_{q_2}^T E_{q_1}^T(\alpha_1)C\delta_{q_3} = \cos\alpha_1 y_1 - \sin\alpha_1 x_1 \tag{4-59}$$

where

$$\begin{cases} x_1 = -\sigma(q)h(q_1, q_2)C_{\nu(q_1,q_2)q_3} \\ y_1 = \sigma(q)C_{q_2q_3} \end{cases} \tag{4-60}$$

and

$$\sigma(q) = \begin{cases} 1 & \text{if } q_1 = q_3 \\ -\delta_{q_1}^T S(\delta_{q_2})\delta_{q_3} & \text{if } q_1 \neq q_3 \end{cases} \tag{4-61}$$

It may be noted that, for $q_1 \neq q_3$,

$$\sigma(q) = h(q_1, q_3) = -h(q_2, q_3) = -h(q_1, q_2) \tag{4-62}$$

If $|x_1| + |y_1| = 0$, then we have a singular case (gimbal lock) where the middle rotation makes the outer rotations equivalent: only the sum $\alpha_1 + \alpha_3$ can be computed from C. As noted previously, gimbal lock occurs at $\alpha_2 = \pm\pi/2$ for a nonrepeating sequence, and at $\alpha_2 = 0, \pi$ for a repeating sequence. Away from gimbal lock, there are two solutions depending on the choice of sign for components in equation (4-59). One solution is given by

$$\alpha_1 = \arctan(y_1, x_1) \tag{4-63}$$

and the other solution is given by

$$\alpha_1^* = \arctan(-y_1, -x_1) = \alpha_1 + \pi \tag{4-64}$$

The first choice corresponds to α_2, whose range is R_2. Thus, if $q_1 \neq q_3$ and $\alpha = 0$ so that $C = I$, then

$$\begin{cases} x_1 = h^2(q_1, q_2)C_{q_3q_3} = 1 \\ y_1 = \sigma(q)C_{q_2q_3} = 0 \end{cases}$$

so $\alpha_1 = 0$ and not π. On the other hand, if $q_1 = q_3$ and $\alpha = (0, \pi/2, 0)$ so that $C = E_{q_2}(\pi/2)$, then (using eq. (4-62))

$$\begin{cases} x_1 = -h(q_1, q_2)\delta_{\nu(q_1,q_2)} E_{q_2}(\pi/2)\delta_{q_3} = h(q_1, q_2)^2 = 1 \\ y_1 = C_{q_2,q_3} = 0 \end{cases}$$

Therefore, in either case we get back α_1 and not α_1^*, which corresponds to α_2^*. Similarly, for α_3:

$$\delta_{q_1}^T C E_{q_3}(-\alpha_3)\delta_{q_2} = 0 = y_3 \cos\alpha_3 - x_3 \sin\alpha_3 \tag{4-65}$$

Let

$$\begin{cases} x_3 = -\sigma(q)h(q_2, q_3)C_{q_1 q_{\nu(q_2,q_3)}} \\ y_3 = \sigma(q)C_{q_1 q_2} \end{cases} \tag{4-66}$$

Then

$$\alpha_3 = \arctan(y_3, x_3) \tag{4-67}$$

The other choice of sign leads to

$$\alpha_3^* = \arctan(-y_3, -x_3) = \alpha_3 + \pi \tag{4-68}$$

The companion to α_2 is α_3, and the companion to α_2^* is α_3^*. Thus we have the following algorithm (ref. 27) for extracting Euler angles from direction cosine matrices:

ALGORITHM: $(C, q) \succ (\boldsymbol{\alpha})$
if $q_1 = q_3$, then
  $x_2 = C_{q_1 q_1}$
  $y_2 = \sqrt{1 - x_2^2}$
else
  $y_2 = h(q_2, q_3)C_{q_1 q_3}$
  $x_2 = \sqrt{1 - y_2^2}$
end if
$x_1 = -\boldsymbol{\sigma}(q)h(q_1, q_2)C_{\boldsymbol{\nu}(q_1,q_2)q_3}$
$y_1 = \boldsymbol{\sigma}(q)C_{q_2 q_3}$
$x_3 = -\boldsymbol{\sigma}(q)h(q_2, q_3)C_{q_1 q_{\boldsymbol{\nu}(q_2,q_3)}}$
$y_3 = \boldsymbol{\sigma}(q)C_{q_1 q_2}$
$\boldsymbol{\alpha}_1 = \arctan(y_1, x_1)$
$\boldsymbol{\alpha}_2 = \arctan(y_2, x_2)$
$\boldsymbol{\alpha}_3 = \arctan(y_3, x_3)$

Example 4.3. Let $q = (1, 2, 3)$. (All twelve cases are given in table 2.1 in ref. 1.) Then

$$C = E_1(\alpha_1)E_2(\alpha_2)E_3(\alpha_3) = \begin{pmatrix} c_2 c_3 & c_2 s_3 & -s_2 \\ -c_1 s_3 + s_1 s_2 c_3 & c_1 c_3 + s_1 s_2 s_3 & s_1 c_2 \\ s_1 s_3 + c_1 s_2 c_3 & -s_1 c_3 + c_1 s_2 s_3 & c_1 c_2 \end{pmatrix}$$

where we abbreviate $\cos\alpha_1 = c_1$, etc. Following the algorithm, $h(2,3) = -1$; so $y_2 = -C_{13} = s_2$ and $x_2 = \sqrt{1 - s_2^2}$.

Next, $\sigma(q) = +1$, $h(1,2) = -1$, and $\nu(1,2) = 3$; so $x_1 = C_{33} = c_1c_2$, and $y_1 = C_{23} = s_1c_2$.

Finally, $\nu(2,3) = 1$; so $x_3 = -C_{11} = c_2c_3$ and $y_3 = C_{12} = c_2s_3$.

Thus

$$\begin{cases} \alpha_1 = \arctan(s_1c_2, c_1c_2) \\ \alpha_2 = \arctan(s_2, |c_2|) \\ \alpha_3 = \arctan(c_2s_3, c_2c_3) \end{cases}$$

If one begins with β so that

$$\begin{cases} -\pi < \beta_1 \leq \pi \\ -\pi/2 < \beta_2 < \pi/2 \\ -\pi < \beta_3 \leq \pi \end{cases}$$

then the combined computation

$$\beta \succ C \succ \alpha$$

will produce an $\alpha = \beta$. For β_2 in the other sector R_2^*, the combined computation will produce an $\alpha = \beta^*$, so that the alternate solution $\alpha^* = \beta$.

The general algorithm $C \succ \alpha$ may be raised to order m as follows:

ALGORITHM: $AF_q^m(\boldsymbol{\alpha}) = MF \succ AF_q[MF^m(C)]$
if $q_1 = q_3$, then
$SF^m(x_2) = MF^m(C)_{q_1q_1}$
$SF^m(y_2) = \sqrt{(1) - [SF^m(x_2)]^2}$
else
$SF^m(y_2) = h(q_2, q_3)MF^m(C)_{q_1q_3}$
$SF^m(x_2) = \sqrt{(1) - [SF^m(y_2)]^2}$
end if
$SF^m(x_1) = -\boldsymbol{\sigma}(q)h(q_1, q_2)MF^m(C)_{\boldsymbol{\nu}(q_1,q_2)q_3}$
$SF^m(y_1) = \boldsymbol{\sigma}(q)MF^m(C)_{q_2q_3}$
$SF^m(x_3) = -\boldsymbol{\sigma}(q)h(q_2, q_3)MF^m(C)_{q_1q_{\boldsymbol{\nu}(q_2,q_3)}}$
$SF^m(y_3) = \boldsymbol{\sigma}(q)MF^m(C)_{q_1q_2}$
$SF^m(\boldsymbol{\alpha}_1) = \arctan[SF^m(y_1), SF^m(x_1)]$
$SF^m(\boldsymbol{\alpha}_2) = \arctan[SF^m(y_2), SF^m(x_2)]$
$SF^m(\boldsymbol{\alpha}_3) = \arctan[SF^m(y_3), SF^m(x_3)]$

We have just developed an algorithm for extracting Euler angles and their m time derivatives from the direction cosine matrix and its m time derivatives for any Euler sequence. In fact we have also

linked the Euler angle forms (AF) to the rotational forms (RF):

$$AF_q \quad \Leftrightarrow \quad MF \quad \Leftrightarrow \quad RF$$

So, for example, the algorithm $RF \succ AF_q$ extracting the Euler angle form from a rotation form may be executed by first transforming the rotational form RF_{br}^m into the matrix form $MF^m(C_{br})$; then the matrix form is transformed into the Euler angle form $AF_q^m(\alpha_{br})$, namely,

$$AF_q^m(\alpha_{br}) = MF \succ AF_q[MF \prec RF(RF_{br}^m)] \tag{4-69}$$

and, conversely, the algorithm for computing the rotational form from the Euler angle form is

$$RF_{br}^m = MF \succ RF\{MF \prec AF_q[AF_q^m(\alpha_{br})]\} \tag{4-70}$$

As a reminder, we note that angular velocity, angular acceleration, and higher time derivatives, which are of particular interest in practice, are contained in the rotational form

$$RF_{br}^m = (\, C_{br}, \quad VF^{m-1}(\omega_{brb})\,) \tag{4-71}$$

so that, if we have computed RF_{br}^m, then we have also computed ω_{brb} and $m-1$ of its time derivatives.

Example 4.4. Consider again example 4.1, where the aircraft trajectory represented by the vector form $VF^4(v_r)$ was transformed into the rotational form RF_{vr}^3, which describes the rotation of the stability axes $\overleftrightarrow{v}$ relative to the runway axes $\overleftrightarrow{r}$. Suppose that, following convention, we represent the attitude of the stability axes relative to the runway as

$$C_{vr} = E_1(\phi_v)E_2(\gamma_v)E_3(\psi_v)$$

Then the roll, flight path, heading angles

$$\alpha_{vr} = \begin{pmatrix} \phi_v \\ \gamma_v \\ \psi_v \end{pmatrix}$$

and their time derivatives up to order 3 are given by

$$AF_{(1,2,3)}^3(\alpha_{vr}) = MF \succ AF_{(1,2,3)}[MF \prec RF(RF_{vr}^3)]$$

Furthermore, if the body attitude $C_{br} = C_{bv}C_{vr}$ and, following convention,

$$C_{bv} = E_2(\alpha)E_3(-\beta)E_1(\rho)$$

then, for the already computed body rotation RF_{br}^3, the evolution of the angle of attack, the (negative) sideslip angle, and the relative roll angle ρ

$$\alpha_{bv} = \begin{pmatrix} \alpha \\ -\beta \\ \rho \end{pmatrix}$$

and their time derivatives are given simply by

$$AF^3_{(2,3,1)}(\alpha_{bv}) = MF \succ AF_{(2,3,1)}[MF \prec RF(RF^3_{br} \star RF^3_{rv})]$$

where all operations have been already defined. A numerical example is given in appendix C.

Example 4.5. Suppose that a sequence q and Euler angles α_{br} are given and that the Jacobian $J_q(\alpha_{br})$ relating angular velocity ω_{brb} to the Euler angle rates $\dot{\alpha}_{br}$ is desired so that

$$\omega_{brb} = J_q(\alpha_{br})\dot{\alpha}_{brb} \tag{4-72}$$

Then the three columns of J_q are given by three calls to $RF \prec AF_q$:

$$J_q(\alpha_{br}) = (\,\{RF \prec AF_q[(\alpha_{br}, \delta_1)]\}_4, \quad \{RF \prec AF_q[(\alpha_{br}, \delta_2)]\}_4, \quad \{RF \prec AF_q[(\alpha_{br}, \delta_3)]\}_4\,) \tag{4-73}$$

where, as before, $(x, y) = (x, y, 0, \ldots, 0)$.

Example 4.6. Suppose a satellite is required to move so that the Euler angles for a sequence q are a given function of time, $\alpha_{br} = f(t)$. In order to check whether the maneuver is executable, we wish to check the required angular acceleration and acceleration rate. Let

$$VF^3(\alpha_{br}) = [\,f(t), \dot{f}(t), \ddot{f}(t), f^{(3)}(t)\,]$$

The required angular acceleration and acceleration rate are given by the third and fourth items in

$$RF^3_{br} = (\,C_{br}, \omega_{brb}, \dot{\omega}_{brb}, \ddot{\omega}_{brb}, \ldots\,) = MF \succ RF\{MF \prec AF_q[AF^3_q(\alpha_{br})]\}$$

Note that the Jacobian is not needed for this computation.

Example 4.7. Still another example is provided by patching two Euler angle parameterizations. Let one set of Euler angles α_{br} be given in the nonrepeating sequence $q = (1, 2, 3)$ and another set β_{br} be given in the repeating sequence $p = (3, 2, 3)$. Both represent the same motion of the body RF^m_{br}. The two coordinate systems (angles and m time derivatives) are related by the back-and-forth maps,

$$AF^m_p(\beta_{br}) = MF \succ AF_p\{MF \prec AF_q[AF^m_q(\alpha_{br})]\}$$

and, conversely,

$$AF^m_q(\alpha_{br}) = MF \succ AF_q\{MF \prec AF_p[AF^m_p(\beta_{br})]\}$$

Such changes in coordinates are useful when the trajectory passes near gimbal lock in one set of coordinates.

Euler Parameter Form

Often in attitude control it is desirable to express attitude error in terms of Euler parameters,

$$\begin{cases} \epsilon = \sin(\phi/2)u \\ \\ \eta = \cos(\phi/2) \end{cases} \tag{4-74}$$

where the unit column u is the Euler axis of rotation and ϕ is the angle. The direction cosine matrix is given in terms of Euler parameters by

$$C = I + 2\eta S(\epsilon) + 2S^2(\epsilon) \tag{4-75}$$

and, conversely,

$$\begin{cases} \eta = \frac{1}{2}[tr(C) + 1]^{1/2} \\ \epsilon = \frac{1}{2} axis(C)/\eta \end{cases} \tag{4-76}$$

The constraint is $\epsilon^2 + \eta^2 = 1$. A singularity exists at $\phi = \pi$ since there are two solutions $\pm u$. The Euler parameter form is defined by the Euler parameters and their m time derivatives,

$$PF^m = \begin{pmatrix} VF^m(\epsilon) \\ \\ SF^m(\eta) \end{pmatrix} \tag{4-77}$$

The relation between MF^m_{br} and PF^m_{br} is obtained simply by rewriting the defining equations in terms of dynamic forms. That is, the algorithm for

$$MF^m_{br} = MF \prec PF(PF^m_{br}) \tag{4-78}$$

is obtained simply by converting equation (4-75) to dynamic forms:

$$MF^m(C) = MF^m(I) + 2SF^m(\eta) \star S[VF^m(\epsilon)] + 2S[VF^m(\epsilon)] \star S[VF^m(\epsilon)] \tag{4-79}$$

and, conversely, the inverse algorithm

$$PF^m_{br} = MF \succ PF(MF^m_{br}) \tag{4-80}$$

is the translation of equation (4-76):

$$\begin{cases} SF^m(\eta) = \frac{1}{2}\{tr[MF^m(C)] + SF^m(1)\}^{1/2} \\ VF^m(\epsilon) = \frac{1}{2} axis[MF^m(C)]/SF^m(\eta) \end{cases} \tag{4-81}$$

The angle forms (AF_q), rotational forms (RF), and parameter forms (PF) are now linked to the (direction cosine) matrix forms (MF) as follows:

$$\begin{array}{ccccc} AF_q & \Leftrightarrow & MF & \Leftrightarrow & RF \\ & & \Updownarrow & & \\ & & PF & & \end{array}$$

In other words, the algorithm for converting Euler angle form to Euler parameters form is given by

$$PF^m_{br} = MF \succ PF\{MF \prec AF_q[AF^m_q(\alpha_{br})]\} \tag{4-82}$$

and similarly for any two representations of rotational motion shown in the diagram.

Attitude Servo

Consider a simplified attitude control problem. Suppose that the object to be controlled (the plant) is spherically symmetric and that it is described by the following set of equations:

$$d/dt \begin{pmatrix} C_{br} \\ \omega_{brb} \\ z \end{pmatrix} = \begin{pmatrix} S(\omega_{brb})C_{br} \\ z + \nu \\ u \end{pmatrix} \tag{4-83}$$

where C_{br} and ω_{brb} are, respectively, attitude and angular velocity of the body $\overleftrightarrow{b}$ relative to a reference $\overleftrightarrow{r}$. There is an intervening integral between the angular acceleration $\dot{\omega}_{brb}$ and the control u. In addition, there is a disturbance ν. A block diagram of this plant is shown in figure 4.1. Note that the plant is of the pure feedback type. The objective is to devise a control scheme for tracking time-varying servo input $C_{cr}(t)$. The servo design shown in figure 2.8 may be specialized to attitude control as follows:

Step 1. Command. First consider the command. Construct the corresponding third-order matrix form MF^3_{cr} from the known function of time for the servo input $C_{cr}(t)$ for $t \geq 0$. For example, suppose that the command generator produces Euler angles and derivatives up to order three, $AF^3_q(\alpha_{cr})$. Then

$$MF^3_{cr} = MF \prec AF_q[AF^3_q(\alpha_{cr})]$$

Alternatively, MF^3_{cr} may have been computed from a translational trajectory as in example 1 where the dynamic form of the stability axes was computed.

Step 2. Output. Next, from the plant state estimate (assumed to be available at all times $t \geq 0$),

$$\hat{x} = \begin{pmatrix} \hat{C}_{br} \\ \hat{\omega}_{brb} \\ \hat{z} \end{pmatrix} \tag{4-84}$$

and an estimate of the disturbance and its rate $VF^1(\hat{\nu})$ with $\hat{\dot{\omega}}_{brb} = \hat{z} + \hat{\nu}$, construct the plant output as follows: First compute an estimate of the plant rotation form,

$$\widehat{RF}^2_{br} = (\hat{C}_{br}, \quad \hat{\omega}_{brb}, \quad \hat{\dot{\omega}}_{brb}) \tag{4-85}$$

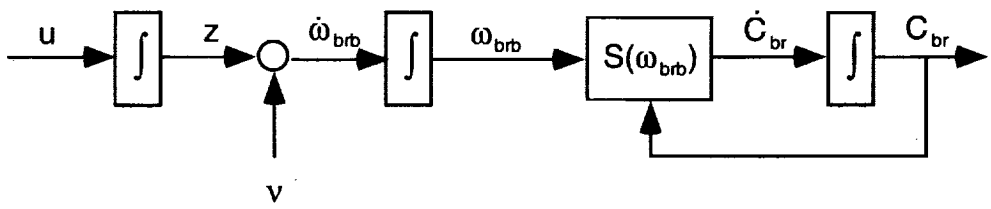

Figure 4.1. Plant block diagram.

Then transform it into matrix form,

$$\widehat{MF}_{br}^{2} = \left(\hat{C}_{br}, \quad \dot{\hat{C}}_{br}, \quad \ddot{\hat{C}}_{br} \right) = MF \prec RF(\widehat{RF}_{br}^{2}) \tag{4-86}$$

Step 3. Error. Now define the tracking error by the relative transformation, as in reference 28,

$$MF_{bc}^{2} = \widehat{MF}_{br}^{2} \star MF_{rc}^{2} \tag{4-87}$$

The goal is to drive the error close to $(I, 0, 0)$ and keep it there by manipulating the last element of $RF_{br}^{(3)}$, namely $\ddot{\omega}_{bcb}$, which is directly accessible for control. The control law for the error may be based on various parameterizations, such as Euler angles and Euler parameters. Following reference 29, we choose the Euler parameters (see eq. (4-81)),

$$PF_{bc}^{2} = \begin{pmatrix} \epsilon^{(0)}, & \epsilon^{(1)}, & \epsilon^{(2)} \\ \eta^{(0)}, & \eta^{(1)}, & \eta^{(2)} \end{pmatrix} = MF \succ PF(MF_{bc}^{2}) \tag{4-88}$$

and view the vector part $VF^{2}(\epsilon)$ as the error state equation, where u_ϵ is written in place of $\epsilon^{(3)}$ to emphasize that $\epsilon^{(3)}$ is a control variable

$$d/dt \begin{pmatrix} \epsilon^{(0)} \\ \epsilon^{(1)} \\ \epsilon^{(2)} \end{pmatrix} = \begin{pmatrix} \epsilon^{(1)} \\ \epsilon^{(2)} \\ u_\epsilon \end{pmatrix} \tag{4-89}$$

The equation is a set of three strings each three integrators long (i.e., a Brunowsky form with Kronecker indices $(3, 3, 3)$), as shown in figure 4.2, where each $\epsilon^{(i)}$ is three dimensional. The control u_ϵ is as yet undefined. We design an asymptotically stable regulator law g in the usual way (additional dynamics such as integral feedback could be added to the control law if desired),

$$u_\epsilon = g(\epsilon^{(0)}, \epsilon^{(1)}, \epsilon^{(2)}) \tag{4-90}$$

Step 4. Plant Control. In order to implement this control law, u_ϵ must be transformed back into the plant input u. Thanks to the regulator law, at this point we have raised the order m of $VF^{m}(\epsilon)$ from two to three:

$$VF^{3}(\epsilon) = [VF^{2}(\epsilon), u_\epsilon] \tag{4-91}$$

and we impose the Euler parameter constraint,

$$SF^{3}(\eta) = [SF^{3}(1) - VF^{3}(\epsilon) \cdot VF^{3}(\epsilon)]^{1/2} \tag{4-92}$$

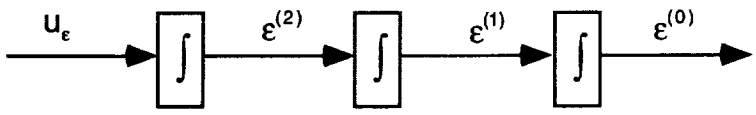

Figure 4.2. Brunowsky form for error dynamics.

thereby raising the order of PF_{bc}^{m} to three. Then we compute the new

$$MF_{bc}^{3} = MF \prec PF(PF_{bc}^{3}) \tag{4-93}$$

and change coordinates to $\widehat{MF}_{br}^{m}$:

$$\widehat{MF}_{br}^{3} = MF_{bc}^{3} \star MF_{cr}^{3} \tag{4-94}$$

Finally, we compute the expanded rotation form,

$$\widehat{RF}_{br}^{3} = RF \prec MF(\widehat{MF}_{br}^{3}) \tag{4-95}$$

The last column of $\widehat{RF}_{br}^{3}$ is $\hat{\bar{\omega}}_{brb}$. Hence, the new value of the plant input is given by

$$u = \hat{\bar{\omega}}_{brb} - \hat{\nu} \tag{4-96}$$

Thus, we have designed a control law for the attitude servo, which will track the variable input attitude $C_{bc}(t)$ provided that the total error angle $\phi < \pi$ (eq. (4-74)) and that the control constraints are not violated.

The block diagram of the resulting servo is shown in figure 4.3.

The input may be given in terms of Euler angles with a fixed sequence, and the sequence may also be switched. In addition, Euler parameters and direction cosines may be used in any combination during operation. In all cases the input is transformed into the matrix form MF_{cr}^{3}.

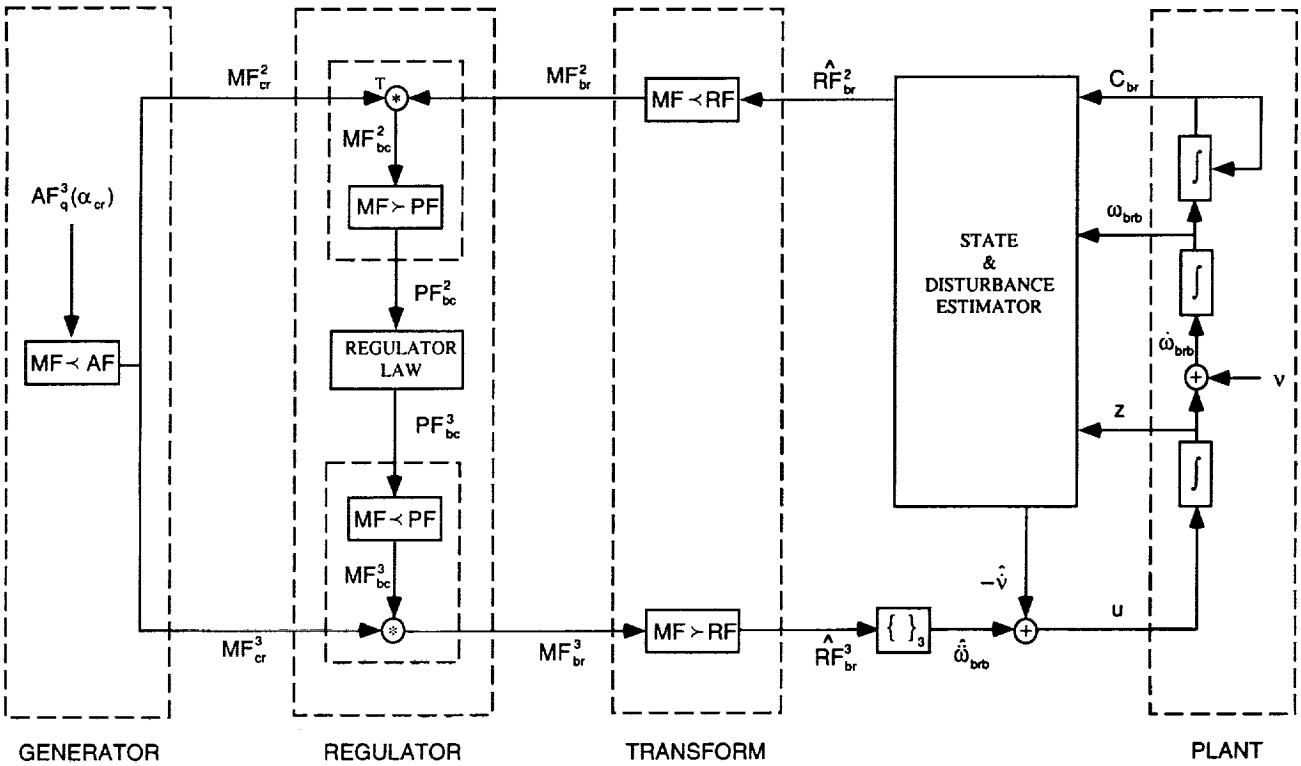

Figure 4.3. Attitude servo block diagram.

If the moment-of-inertia matrix J_b of the plant is not spherically symmetric, the gyroscopic terms may be viewed as a disturbance,

$$\nu = J_b^{-1} S(\omega_{brb}) J_b \omega_{brb}$$

and, as before,

$$\hat{\omega}_{brb}^{(k+1)} = \hat{z}^{(k)} + \hat{\nu}^{(k)}$$

so $VF^k(\hat{\nu})$ can be easily computed by the techniques already described.

The servo control algorithm is summarized in table 4-1, where n_z is the number of integrators between u and $\dot{\omega}_{brb}$, which is in this case one. The algorithm is easily generalized to cases with $n_z > 1$ by making the order a variable, that is, by taking the highest order to be not 3 but $2 + n_z$. A numerical example of a scanning, spinning satellite is given in appendix D.

Table 4-1. Servo control algorithm $n_z = 1$

Purpose	Algorithm
Given input at $t = t_n$	$AF_q^3(\alpha_{cr})$
Change input to matrix form	$MF_{cr}^3 = MF \prec AF_q[AF_q^3(\alpha_{cr})]$
Estimated plant state at $t = t_n$	$(\hat{C}_{br},\ \hat{\omega}_{brb})$
Estimated disturbance at $t = t_n$	$VF^1(\hat{\nu}) = (\hat{\nu}, \dot{\hat{\nu}})$
Estimated plant rotation form	$\widehat{RF}_{br}^2 = (\hat{C}_{br},\ \hat{\omega}_{brb},\ \hat{z} + \hat{\nu})$
Change output to matrix form	$\widehat{MF}_{br}^2 = MF \prec RF(\widehat{RF}_{br}^2)$
Change coordinates	$MF_{bc}^2 = \widehat{MF}_{br}^2 \star MF_{rc}^2$
Change to Euler parameters	$PF_{bc}^2 = MF \succ PF(MF_{bc}^2)$
Error state	$(\epsilon,\ \dot{\epsilon},\ \ddot{\epsilon}) = VF^2(\epsilon)$
Raise order with regulator law	$\epsilon^{(3)} = g(\epsilon,\ \dot{\epsilon},\ \ddot{\epsilon})$
Error state and control	$VF^3(\epsilon) = \left(\epsilon,\ \dot{\epsilon},\ \ddot{\epsilon},\ \epsilon^{(3)}\right)$
Enforce constraint	$SF^3(\eta) = [SF^3(1) - VF^3(\epsilon) \cdot VF^3(\epsilon)]^{1/2}$
Change error to matrix form	$MF_{bc}^3 = MF \prec PF(PF_{bc}^3)$
Change coordinates	$\widehat{MF}_{br}^3 = MF_{bc}^3 \star MF_{cr}^3$
Change to rotation form	$\widehat{RF}_{br}^3 = MF \succ RF(\widehat{MF}_{br}^3)$
Control at $t = t_{n+1}$	$u = (\widehat{RF}_{br}^3)_3 - \dot{\hat{\nu}}$

5 CONCLUSION

The formalism of dynamic forms has been presented. The formalism may be used to translate effectively and systematically zero-order algorithms into m-order algorithms. In this way, for example, a large subroutine that computes the total force and moment acting on an aircraft for a given input, such as aircraft state, controls, and wind, may be routinely converted into the corresponding subroutine that computes the forces, moments, and time derivatives up to any order for given input and time derivatives to that order. Similarly, derivatives can easily be passed through elaborate coordinate changes. It was shown how this capability may be used to organize and simplify the design of control systems. Whereas many examples were provided to illustrate the application to automatic control, the emphasis was on the formalism of dynamic forms. Current research is concerned with specific aircraft applications and with the interpretation of control problems, in general, in terms of dynamic forms.

APPENDIX A

NOE NUMERICAL EXAMPLE

A specific numerical case of the nap-of-the-Earth (NOE) maneuver discussed in example 2.2 is given in this appendix. Let the height h and width σ in feet of the Gaussian cap

$$h = h_{max} e^{-(x/\sigma_x)^2 - (y/\sigma_y)^2}$$

above a flat terrain be

$$\begin{pmatrix} h_{max} \\ \sigma_x \\ \sigma_y \end{pmatrix} = \begin{pmatrix} 50 \\ 50 \\ 100 \end{pmatrix}$$

and let the horizontal path be as shown in figure A-1. The peak is at 50 ft; the level curves are 10 ft apart. The helicopter slides along the cap from the third quadrant, scoots around the peak, and leaves

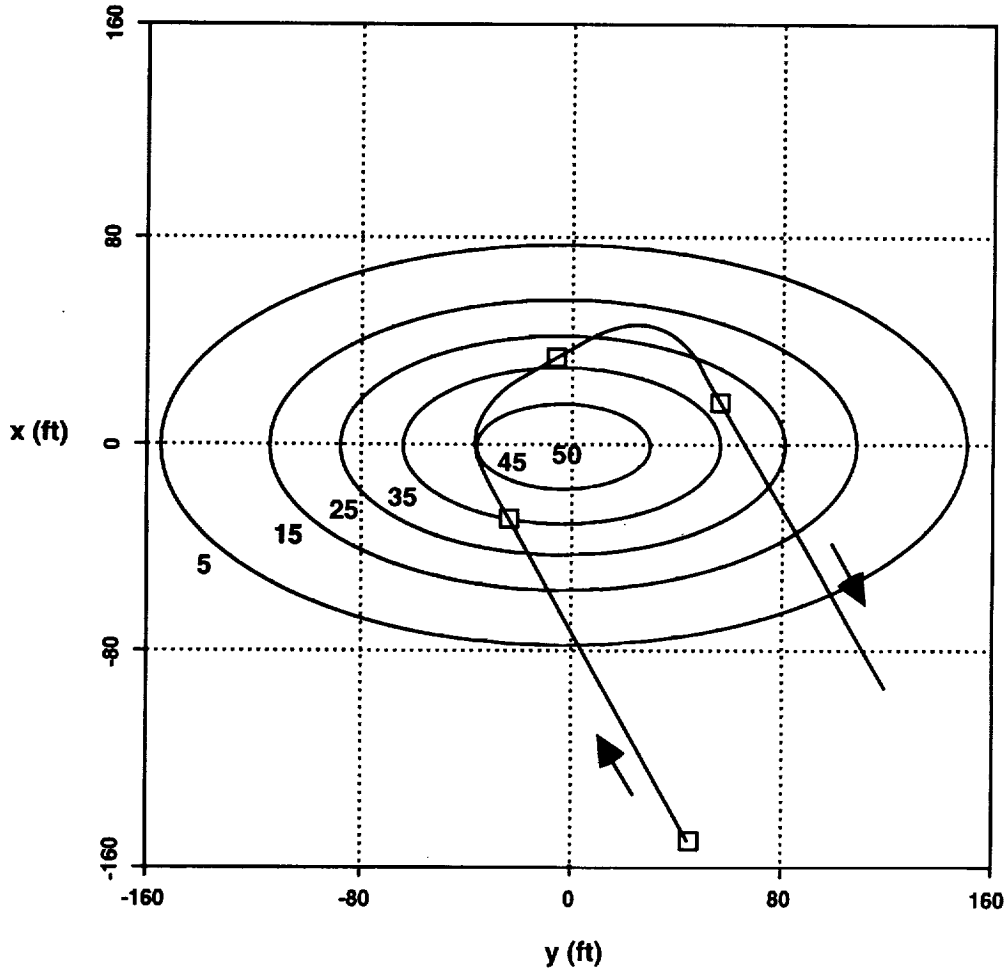

Figure A-1. Plan view of the flightpath and the Gaussian cap.

the area. The maneuver consists of four control points linked by polynomial segments each lasting 4 sec.

The scalar form of arc length $SF^5(s)$ is evolving according to figure A-2. The arc length s shown in the first panel grows from 0 to about 400 ft. The speed $\dot{s}$ shown as the solid line in the second panel is initially 50 ft/sec. During the first 4 sec it is reduced to 20 ft/sec, where it is held during the turn. Finally the helicopter accelerates at a constant 10 ft/sec^2. The acceleration $\ddot{s}$ is shown dotted in the second panel. The first $s^{(3)}$ and second $s^{(4)}$ derivatives of acceleration are shown in the third panel, and the fifth time derivative of the arc length $s^{(5)}$ is shown in the bottom panel of the figure. The higher derivatives may be of interest for the verification that neither the pitch control nor the pitch-control rate exceeds its limit. Thus, if the approximation is made that longitudinal acceleration is controlled by the pitch angle, say θ, namely $\ddot{s} = -g\theta$, then $\ddot{\theta} = -s^{(4)}/g$ determines the pitch-control requirement, whereas $\theta^{(3)} = -s^{(5)}/g$ determines the pitch-control rate.

The scalar form of the heading angle $SF^3(\psi)$ is shown in figure A-3. Initially the heading ψ is –30 deg, and it is held constant for the first 4 sec. Then follow two successive 90-deg turns to the right. Finally, the helicopter exits at 150 deg. The angular rate is bounded by 1 rad/sec, and angular acceleration, by 1 rad/sec^2. The time rate of angular acceleration $\psi^{(3)}$ is shown in the last panel. This derivative is useful for the verification that the yaw-control rate is not excessive during the maneuver.

The shape of the Gaussian cap and the evolution of the arc length and the heading angle are considered in this example to be independent inputs. The helicopter is constrained to stay on the Gaussian cap while executing the maneuver. Consequently the altitude becomes a dependent variable, whose evolution $SF^5(h)$ is shown in figure A-4. The vertical acceleration, which is bounded by 16 ft/sec^2, has a major influence on the required thrust. The higher derivatives indicate the dynamic requirement from the thrust generator.

Figure A-5 shows the (scaled) energy flow $SF^2(e)$. The total kinetic and potential energy e, the power $\dot{e}$, and the time rate of power $\ddot{e}$ are all shown.

This example illustrates the ease with which a variety of time derivatives of transformed functions of time can be computed by means of dynamic forms. It should be noted that, since we are discussing command-trajectory generation, the differentiation of noisy data is not an issue.

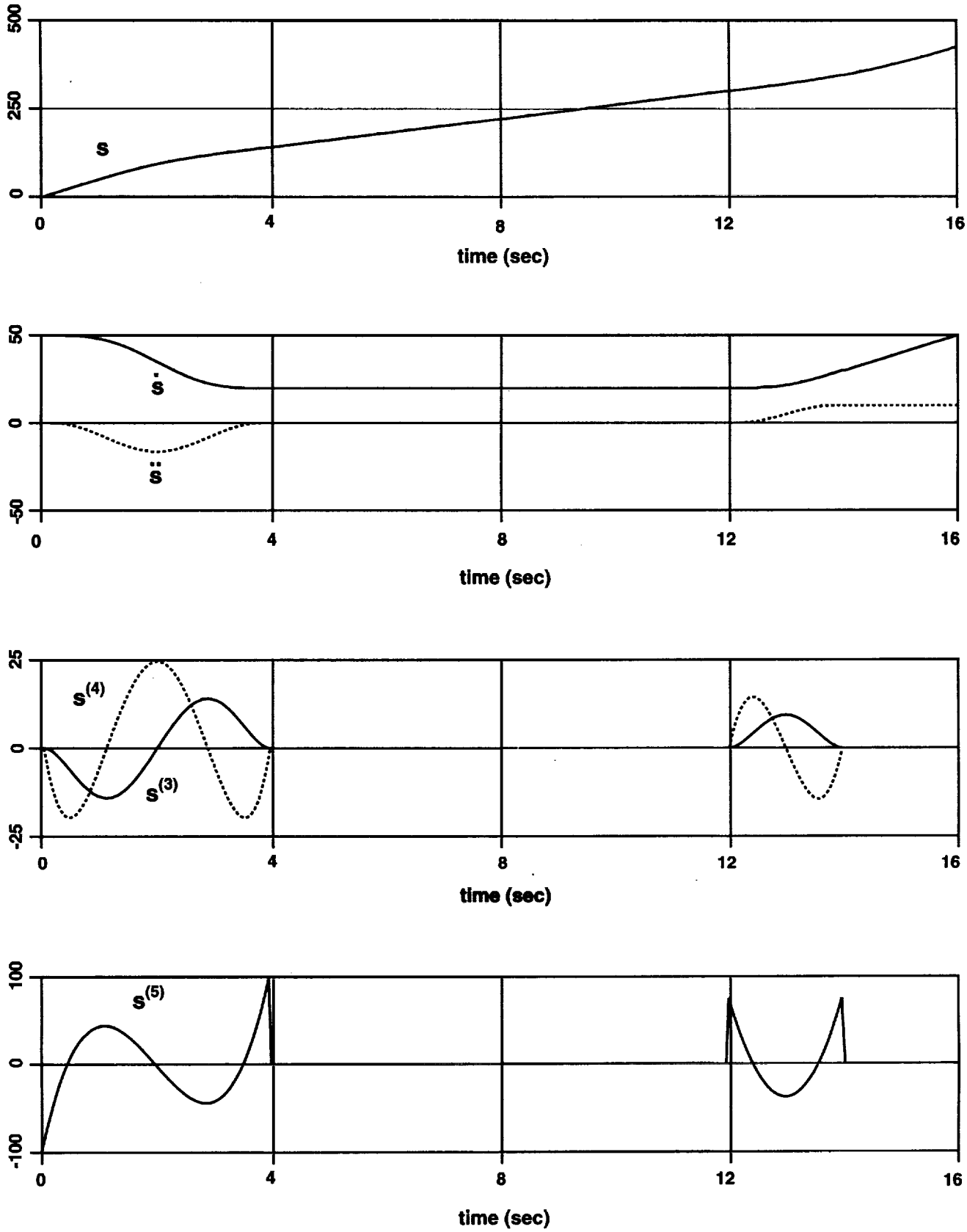

Figure A-2. Evolution of the arc-length scalar form $SF^5(s)$.

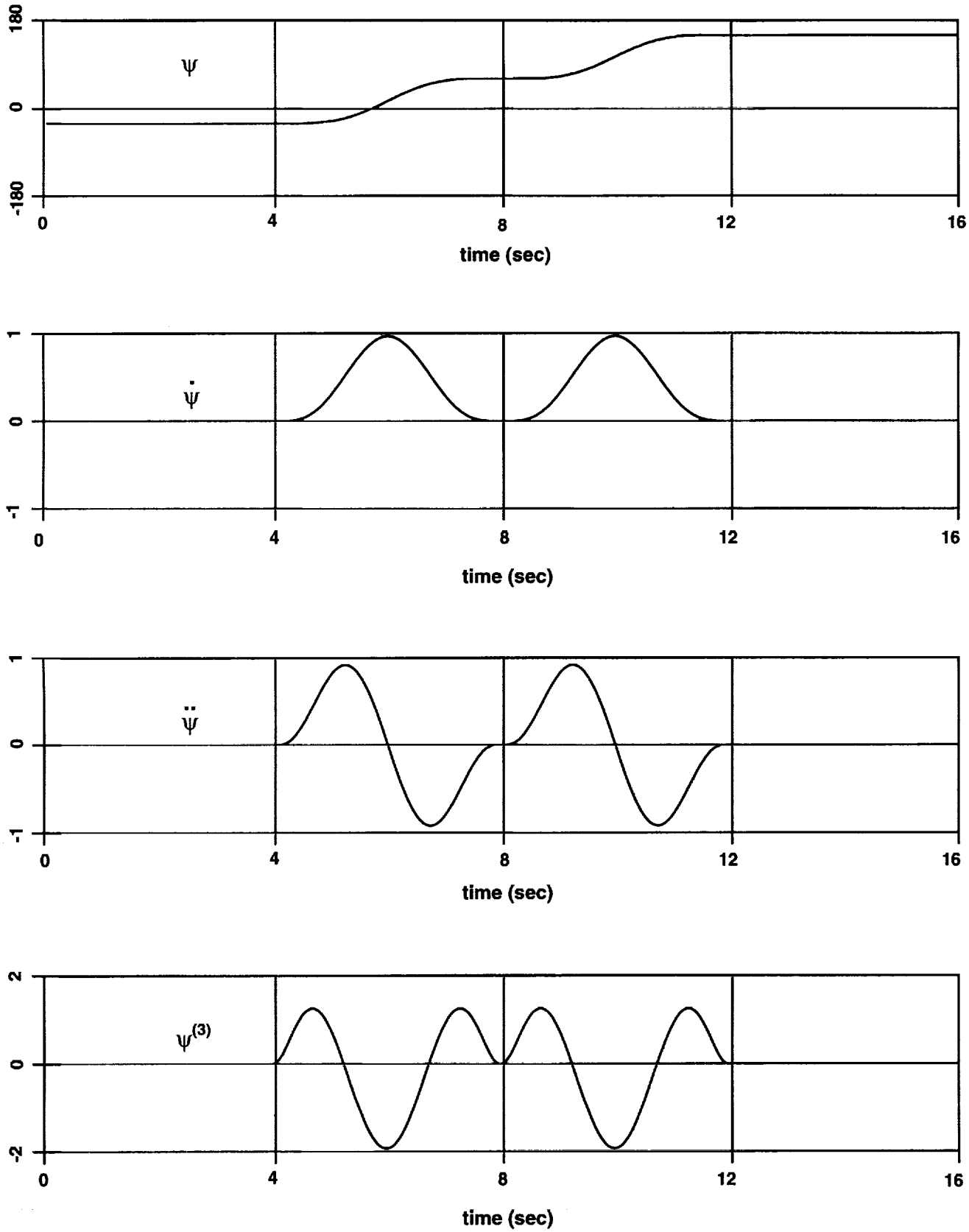

Figure A-3. Evolution of the heading-angle scalar form $SF^5(\psi)$.

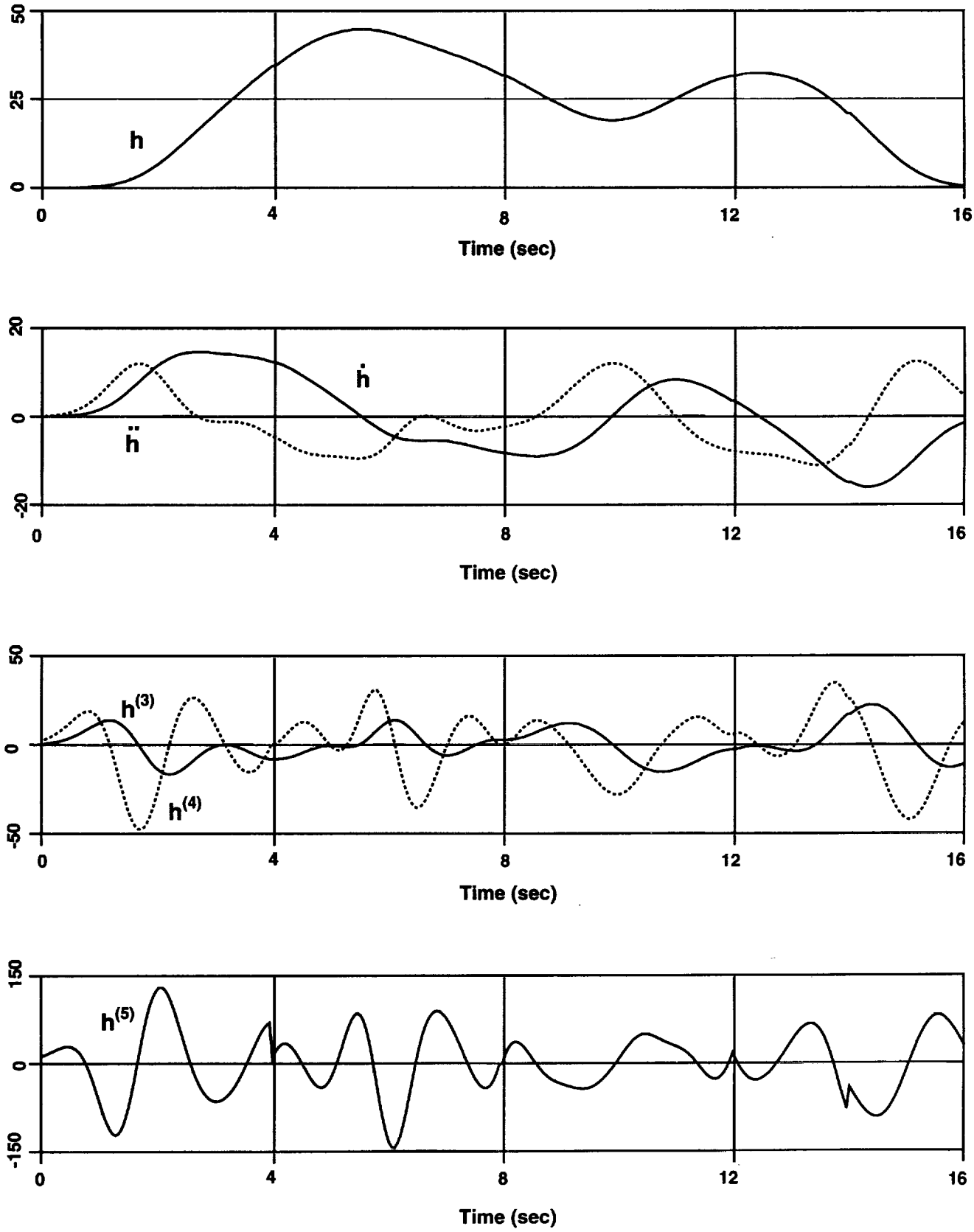

Figure A-4. Evolution of the resulting altitude scalar form $SF^5(h)$.

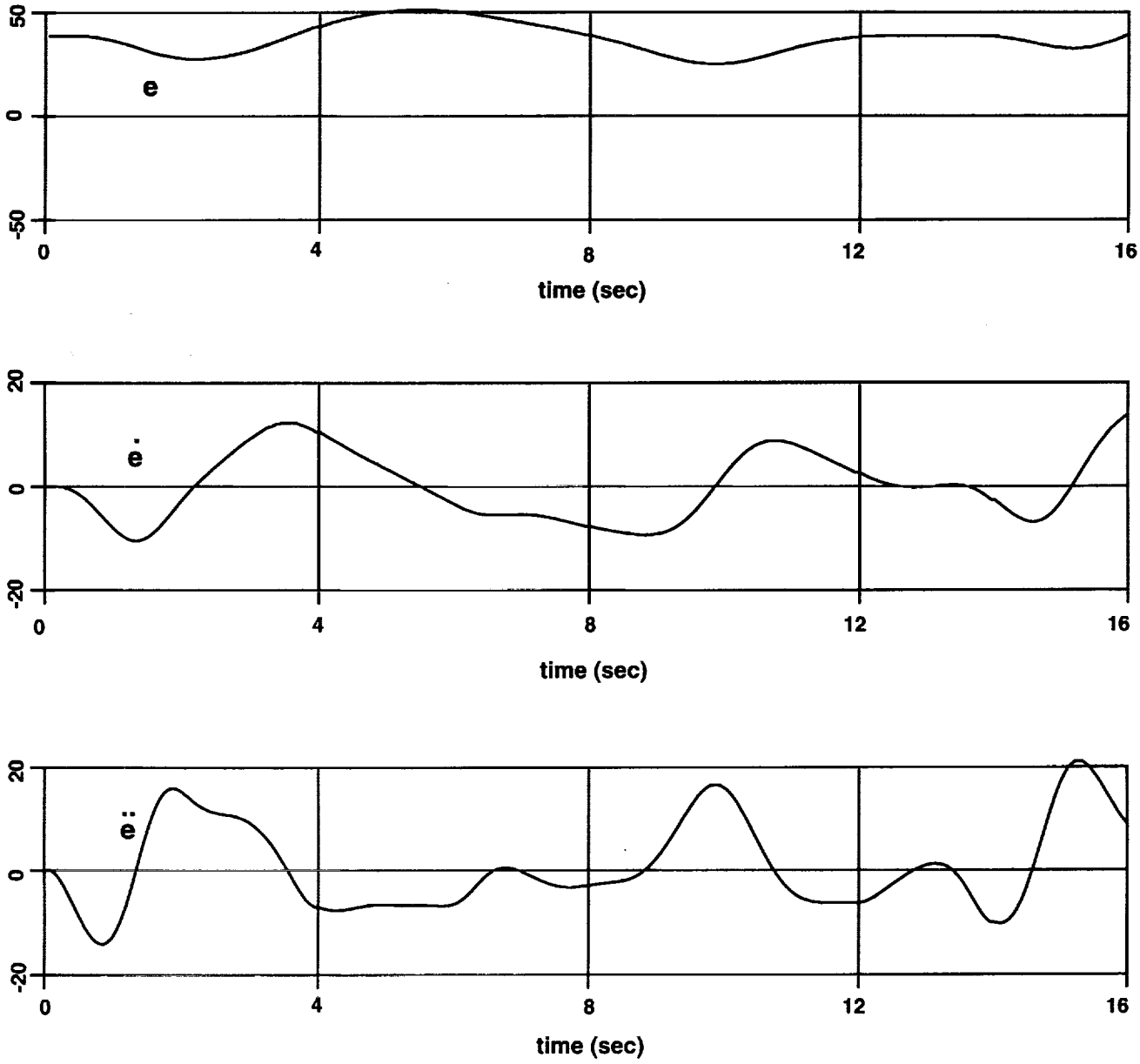

Figure A-5. Evolution of the resulting energy scalar form $SF^2(e)$.

APPENDIX B

AUTOMATIC CONTROL EXAMPLE

This appendix contains an application of the control structure shown in figure 2.8 and the corresponding control algorithm C_f, which is repeated below for convenience.

ALGORITHM: $u_p = C_f[x_p, SF^n(x_{1c})]$
$SF^{n-1}(x_{1p}) = T_{1f}(x_p, \emptyset)$
$SF^{n-1}(e) = SF^{n-1}(x_{1p}) - SF^{n-1}(x_{1c})$
$e^{(n)} = k[SF^{n-1}(e)]$
$SF^n(x_{1p}) = SF^n(x_{1c}) + SF^n(e)$
$[SF^0(u_p), x_p] = T_{1f}^{-1}[SF^n(x_{1p})]$
$u_p = [SF^0(u_p)]_0$

Suppose that the plant is described by a four-dimensional state and that the state equation is as follows:

$$\dot{x}_p = \begin{pmatrix} x_{2p} \\ \sin(0.2x_{1p})x_{2p} + (2 + \cos t)x_{3p} \\ x_{4p} \\ u_p \end{pmatrix}$$

This system is the same one used in example 2.3, where the algorithm T_{1f} was constructed, and in example 2.4, where the algorithm T_{1f}^{-1} was constructed. The only item remaining to be specified is the regulator. We choose a simple linear, constant-gain regulator:

$$e^{(4)} = -k_1 e - k_2 e^{(1)} - k_3 e^{(2)} - k_4 e^{(3)}$$

and place one critically damped pole pair at 0.5 rad/sec and the other at 1 rad/sec. Of course limiters and additional dynamics such as integrators could have been used. Now C_f is completely specified.

Next consider the inputs. Suppose that the input to be tracked consists of four segments, each lasting 4 sec and taking the system from $x_{1c} = 0$ at the beginning of the maneuver to

$$x_{1c}(16) = \begin{pmatrix} 64 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

at the end. The time history of the input $SF^4(x_{1c})$ is shown as solid lines in figure B-1. The system is

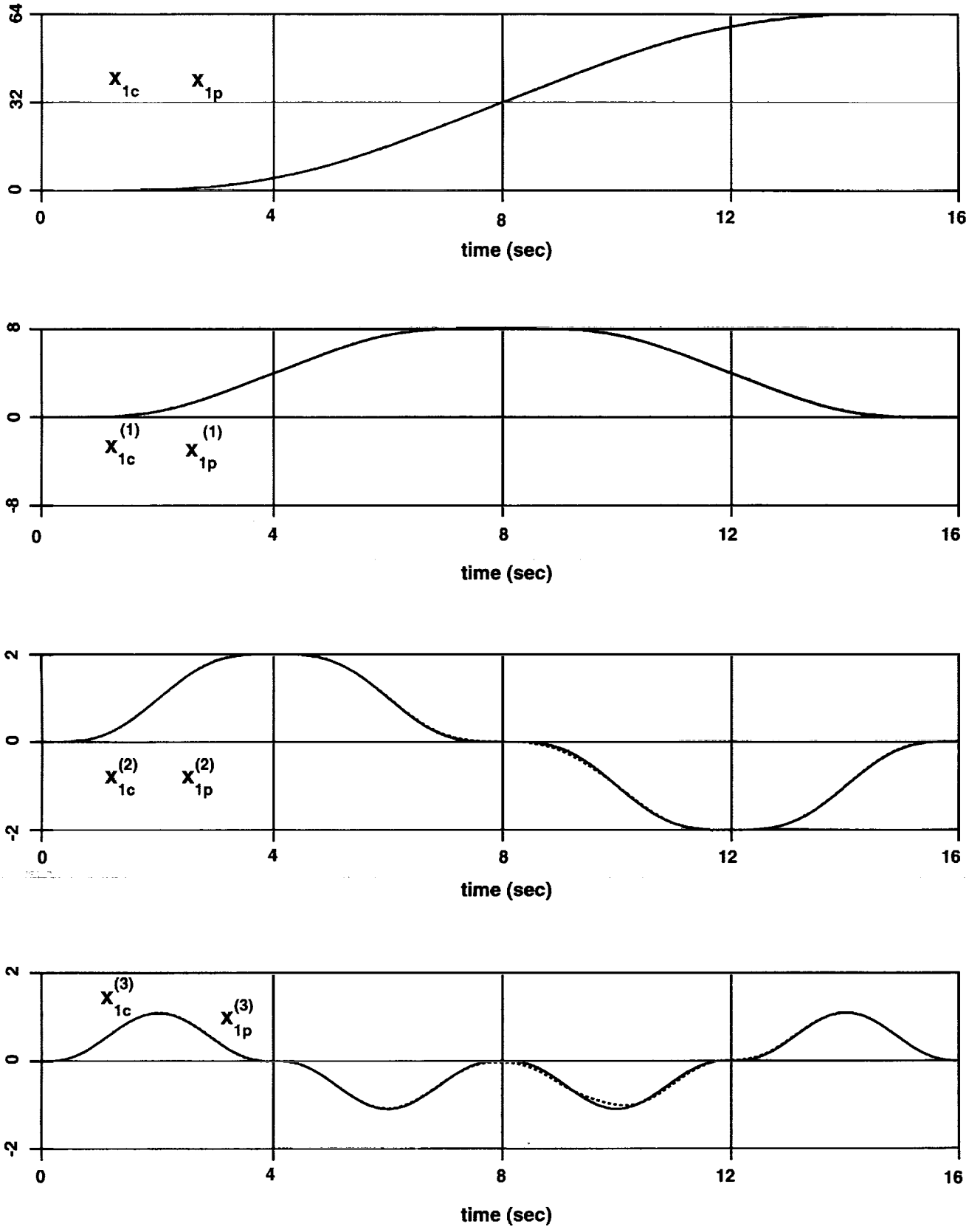

Figure B-1. Evolution of input $SF^4(x_{1c})$ and response $SF^4(x_{1p})$.

commanded to reach steady acceleration $x_{1c}^{(2)} = 2$ at the first waypoint at $t = 4$, then to reach $x_{1c}^{(2)} = 0$ at the second waypoint at $t = 8$, where the speed $x_{1c}^{(1)} = 8$; the boundary condition at the third waypoint is a steady $x_{1c}^{(2)} = -2$; finally, the boundary condition at the fourth waypoint at $t = 16$ is the desired steady position $x_{1c} = 64$. The response of the plant $SF^4(x_{1p})$ with initial condition $x_p = 0$ is shown in the figure by dotted lines. The higher plant state coordinates (x_{3p}, x_{4p}) and the control u_p are shown in figure B-2. Also shown in the figure are the "damping" term $q_1 = a\sin(bx_{1p})x_{2p}$ as a solid line and the "effectiveness" of x_{3p}, namely $q_2 = [2 + \cos(ct)]$, as a dotted line. The regulator error $SF^4(e)$ is shown in figure B-3. It may be noted that tracking is good despite considerable activity in q_1 and q_2. The region near $t = 9$ is especially disruptive because q_1 is maximum while the effectiveness of x_{3p}, namely q_2, is near minimum.

The response to initial offset,

$$x_p(0) = \begin{pmatrix} 10 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

is shown next for the same input. Figure B-4, as figure B-1, shows both the reference and the response. The plant tracks the input asymptotically. In addition, as shown in figure B-5, the error behaves as an autonomous response of constant linear dynamics with the assigned poles. The sampling rate was 100, and the integration step size, 0.01.

This simple example illustrates the application of dynamic forms to control. It should be clear that the same control algorithm C_f can be easily extended to practical cases in which, as noted previously, the implementation of the zero-order function f requires more than 4,000 lines of FORTRAN. An application of C_f to attitude control is given in appendix D.

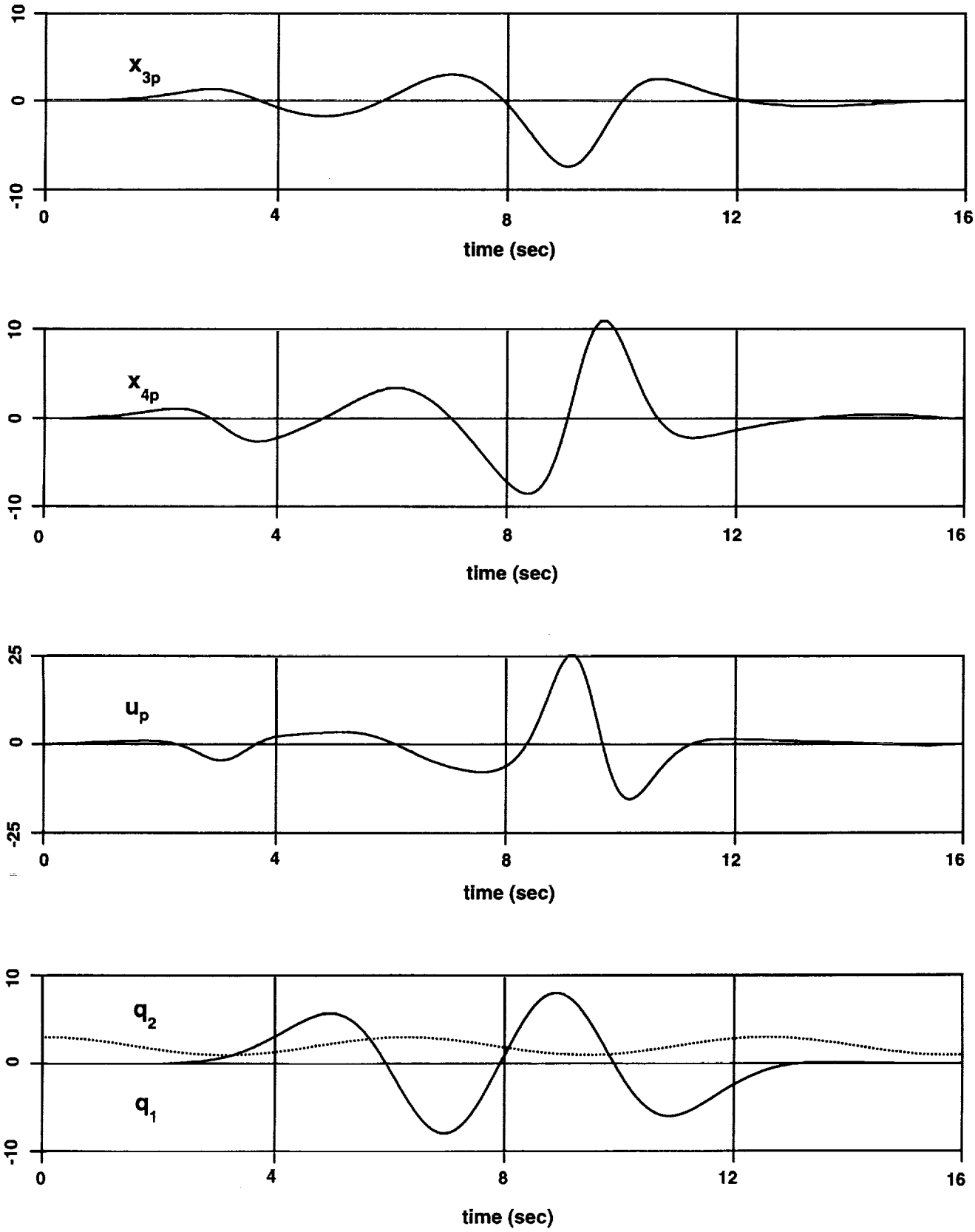

Figure B-2. System response in terms of natural coordinates.

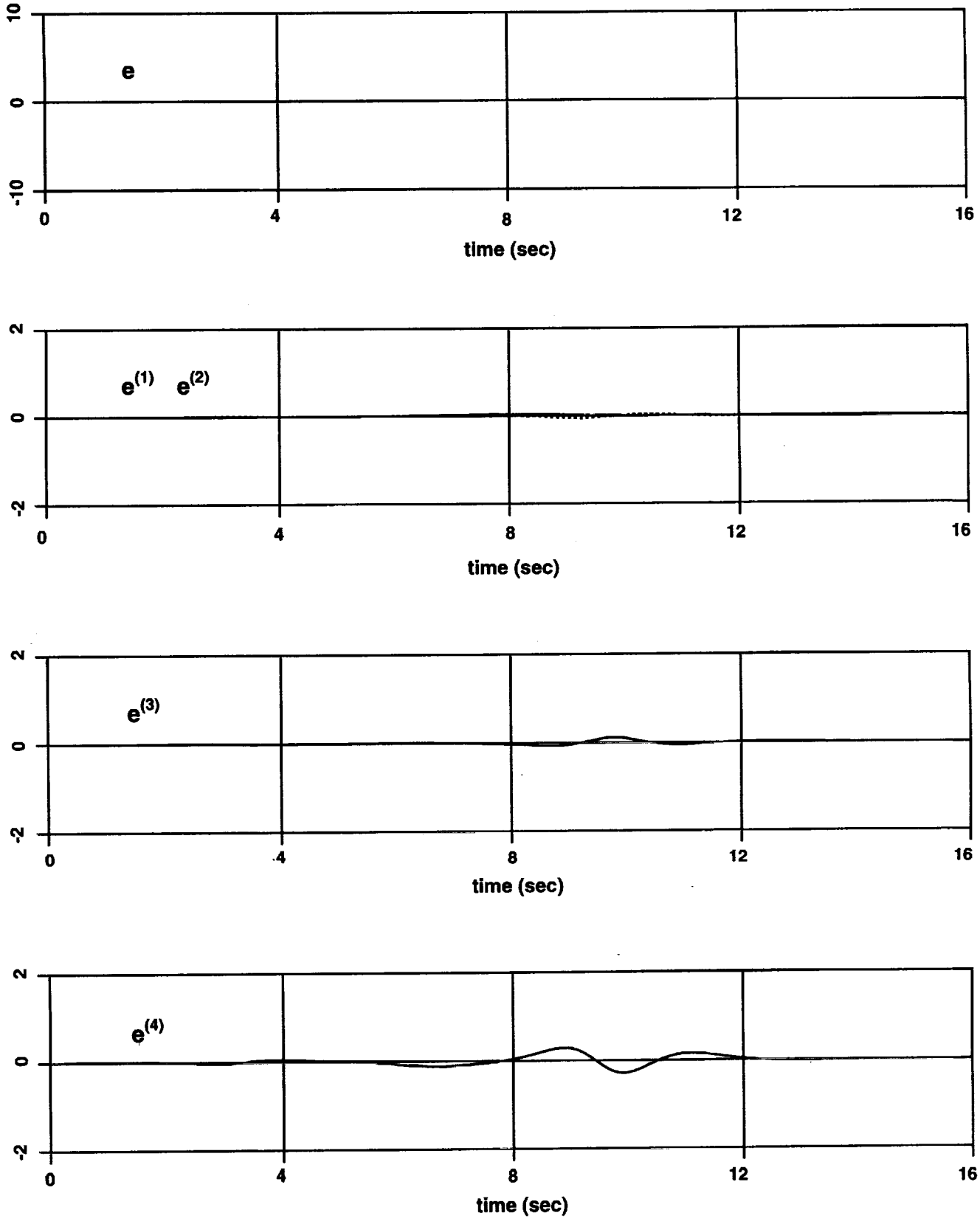

Figure B-3. Error response $SF^4(e)$ for $x_p(0) = 0$.

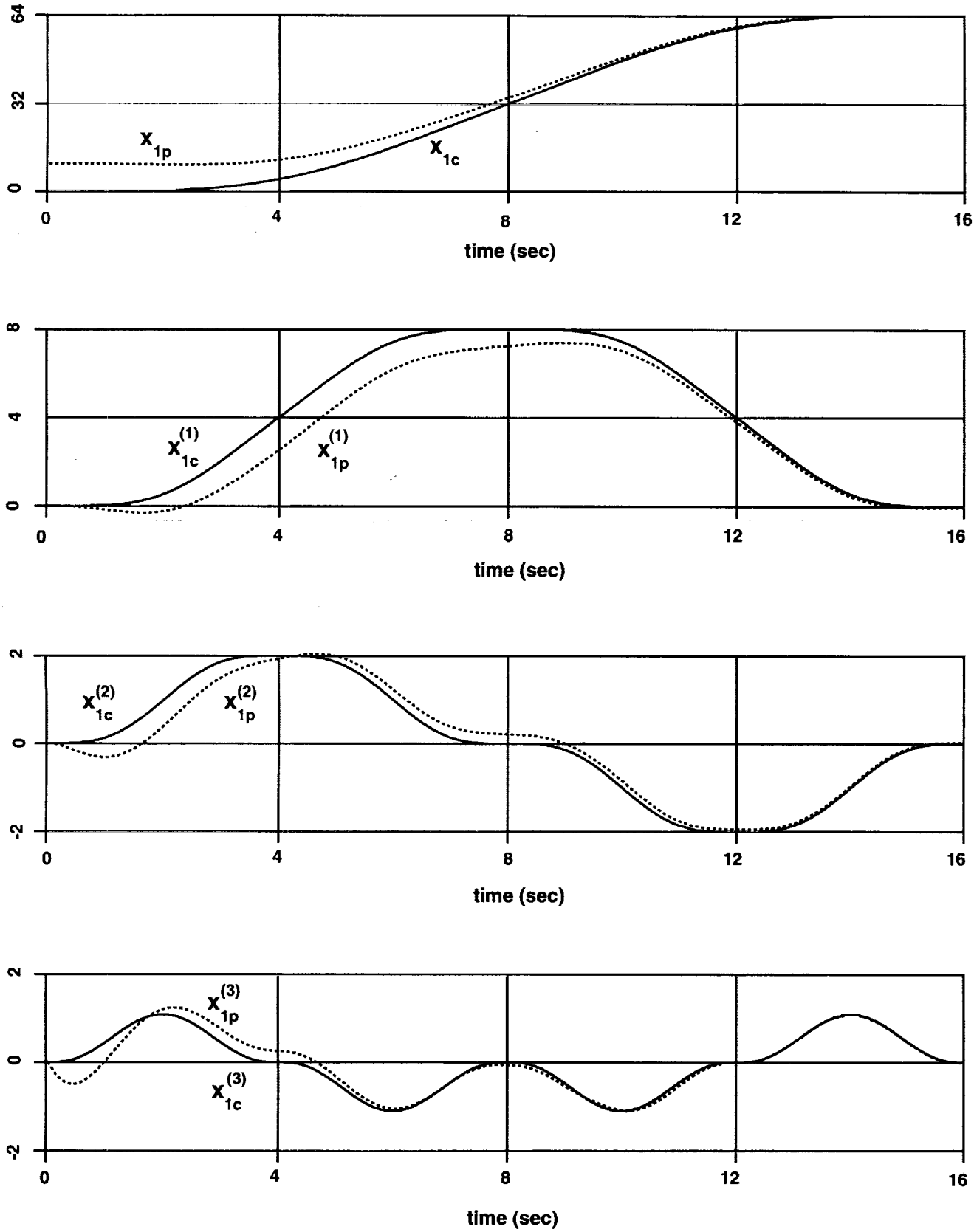

Figure B-4. Evolution of input $SF^4(x_{1c})$ and response $SF^4(x_{1p})$ for $x_{1p}(0) = 10$.

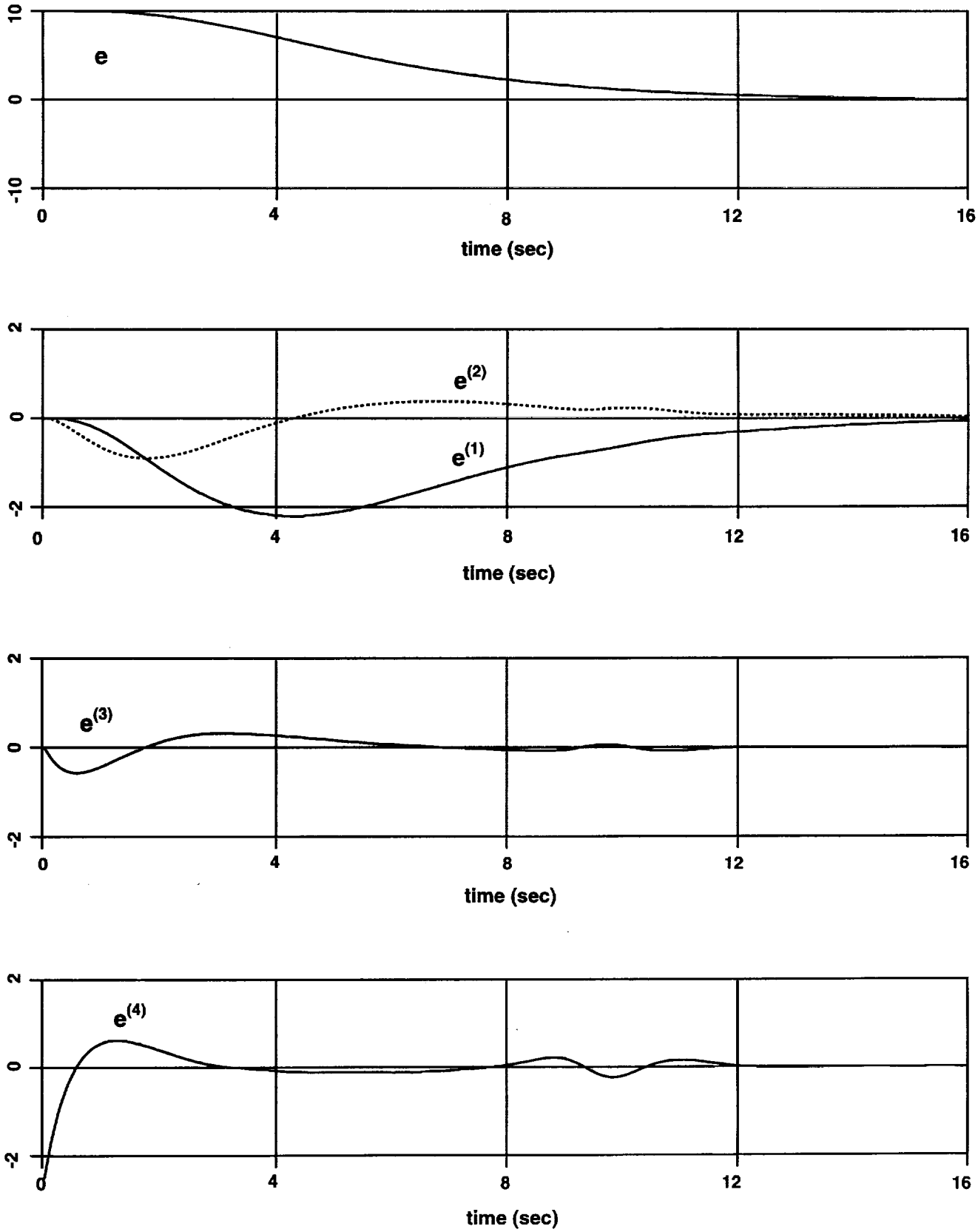

Figure B-5. Error response $SF^4(e)$ for $x_{1p}(0) = 10$.

APPENDIX C

STABILITY AXES NUMERICAL EXAMPLE

For a specific numerical case of example 4.1, we consider the Euler angles of the stability-axes system. Let

$$v_r = vu_r = v \begin{pmatrix} \cos\gamma\cos\psi \\ \cos\gamma\sin\psi \\ -\sin\gamma \end{pmatrix}$$

and let the speed v, the heading angle ψ, and the glidepath angle γ evolve according to

$$\begin{pmatrix} SF^4(v) \\ SF^4(\psi) \\ SF^4(\gamma) \end{pmatrix} = \begin{pmatrix} v, \dot{v}, \ldots, v^{(4)} \\ \psi, \dot{\psi}, \ldots, \psi^{(4)} \\ \gamma, \dot{\gamma}, \ldots, \gamma^{(4)} \end{pmatrix} = \begin{pmatrix} 100 & 10 & -15 & 20 & 5 \\ 1.0 & 0.3 & 0.4 & -0.5 & -0.6 \\ 0.3 & 0.1 & -0.2 & -0.1 & 0 \end{pmatrix}$$

These conditions may be at time t along the reference trajectory, which meets some given boundary conditions at the next waypoint. Then the evolution of the corresponding sines and cosines is given by

$$\begin{pmatrix} \cos SF^4(\psi) \\ \sin SF^4(\psi) \\ \cos SF^4(\gamma) \\ \sin SF^4(\gamma) \end{pmatrix} = \begin{pmatrix} \cos\psi, (\cos\psi)^{(1)}, \ldots, (\cos\psi)^{(4)} \\ \sin\psi, (\sin\psi)^{(1)}, \ldots, (\sin\psi)^{(4)} \\ \cos\gamma, (\cos\gamma)^{(1)}, \ldots, (\cos\gamma)^{(4)} \\ \sin\gamma, (\sin\gamma)^{(1)}, \ldots, (\sin\gamma)^{(4)} \end{pmatrix} = \begin{pmatrix} 0.54 & -0.25 & -0.9 & 0.25 & 0.76 \\ 0.84 & 0.16 & 0.14 & -0.59 & -0.33 \\ 0.96 & -0.03 & 0.05 & 0.09 & -0.08 \\ 0.30 & 0.10 & -0.19 & -0.08 & -0.01 \end{pmatrix}$$

Consequently, the path tangent vector u_r evolves according to

$$VF^4(u_r) = \begin{pmatrix} \cos SF^4(\gamma) \star \cos SF^4(\psi) \\ \cos SF^4(\gamma) \star \sin SF^4(\psi) \\ -\sin SF^4(\gamma) \end{pmatrix} = \begin{pmatrix} 0.52 & -0.26 & -0.33 & 0.28 & 0.45 \\ 0.80 & 0.13 & 0.17 & -0.45 & -0.22 \\ -0.30 & -0.10 & 0.19 & 0.08 & 0.01 \end{pmatrix}$$

the runway coordinates of the velocity and their four derivatives are

$$VF^4(v_r) = SF^4(v) \star VF^4(u_r) = \begin{pmatrix} 51 & -20 & -45 & 40 & 67 \\ 80 & 21 & 7 & -32 & -41 \\ -30 & -12 & 22 & 12 & -22 \end{pmatrix}$$

and the runway coordinates of the total generated force and its three derivatives are (in g's)

$$VF^3(f_r) = g^{-1}VF^3(\dot{v}_r) - VF^3(\delta_3) = \begin{pmatrix} -0.64 & -1.41 & 1.25 & 2.09 \\ 0.65 & 0.22 & -1.01 & -1.28 \\ -1.39 & 0.68 & 0.38 & -0.69 \end{pmatrix}$$

For the unit vector along v_r

$$VF^3(v_{1r}) = VF^3(v_r)/|VF^3(v_r)| = \begin{pmatrix} 0.52 & -0.26 & -0.33 & 0.28 \\ 0.80 & 0.13 & 0.17 & -0.48 \\ -0.30 & -0.10 & 0.19 & 0.08 \end{pmatrix}$$

It may be noted that (happily), $VF^3(v_{1r}) = VF^3(u_r)$, as expected. The normalized cross product,

$$VF^3(v_{2r}) = VF^3(v_{1r}) \times VF^3(f_r)/|VF^3(v_{1r}) \times VF^3(f_r)| = \begin{pmatrix} -0.60 & 0.40 & -0.17 & -1.65 \\ 0.59 & -0.23 & -0.30 & 1.53 \\ 0.55 & 0.68 & -1.10 & 0.66 \end{pmatrix}$$

and, for third-axis vector,

$$VF^3(v_{3r}) = VF^3(v_{1r}) \times VF^3(v_{2r}) = \begin{pmatrix} 0.61 & 0.61 & -0.86 & 0.64 \\ -0.11 & -0.27 & 0.95 & 0.04 \\ 0.78 & -0.51 & -0.09 & 2.31 \end{pmatrix}$$

The matrix form, $MF^3_{vr} = \left(C_{vr}, \dot{C}_{vr}, \ddot{C}_{vr}, C^{(3)}_{vr} \right)$, is just a rearrangement of the vector form $VF^3(v_{ir})$:

$$MF^3_{vr} = \left[\begin{pmatrix} 0.52 & 0.80 & -0.30 \\ -0.60 & 0.59 & 0.55 \\ 0.61 & -0.11 & 0.78 \end{pmatrix}, \begin{pmatrix} -0.26 & 0.13 & -0.10 \\ 0.40 & -0.23 & 0.68 \\ 0.61 & -0.27 & -0.51 \end{pmatrix}, \ldots \right]$$

The v-coordinates of velocity $\vec{v}$ and its derivatives are given by the Coriolis product,

$$VF^3(v_v) = MF^3_{vr} \star VF^3(v_r) = \begin{pmatrix} 100 & 10 & -15 & 20 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Thus, as expected, the activity is only in the top row, which is equal to $SF^3(v)$. The v-coordinates of the force $\vec{f}$ and its derivatives are given by

$$VF^3(f_v) = MF^3_{vr} \star VF^3(f_r) = \begin{pmatrix} 0.61 & -0.37 & 0.42 & 0.08 \\ 0 & 0 & 0 & 0 \\ -1.55 & -0.21 & -0.06 & 3.90 \end{pmatrix}$$

It may be noted that there is no lateral component: f_v moves only in the $\vec{v}_1$ - $\vec{v}_3$ plane, as required.

Next, consider the angular velocity of $\overleftrightarrow{v}$ relative to $\overleftrightarrow{r}$ and its derivatives. We transform the matrix form MF^3_{vr} into the rotational form RF^3_{vr} by means of the function $MF \succ RF$ to obtain:

$$RF^3_{vr} = [C_{vr}, VF^2(\omega_{vrv})] = \left[C_{vr}, \begin{pmatrix} 0.80 & -0.98 & -0.20 \\ 0.25 & 0.21 & 0.04 \\ 0.18 & 0.20 & -0.37 \end{pmatrix} \right]$$

The direction cosine matrix C_{vr} is the first matrix in MF^3_{vr}, above, and

$$VF^2(\omega_{vrv}) = (\omega_{vrv}, \dot{\omega}_{vrv}, \ddot{\omega}_{vrv})$$

contains the angular velocity and its two time derivatives. The two derivatives of angular velocity are used to determine whether moment controls and control rates are within their limits.

Finally, we obtain the Euler angles and their time derivatives for the sequence $q = (1, 2, 3)$.

$$AF^3_q = RF \succ AF_q(RF^3_{vr}) = \begin{pmatrix} 0.61 & 0.89 & -0.83 & -0.33 \\ 0.30 & 0.10 & -0.20 & -0.10 \\ 1.00 & 0.30 & 0.40 & -0.50 \\ 0 & 1 & 2 & 3 \end{pmatrix}$$

The first row corresponds to the evolution of the roll angle $SF^3(\phi)$. The next two rows give $SF^3(\gamma)$ and $SF^3(\psi)$, respectively, which are the same (as of course they should be) as the spherical coordinates given at the beginning of this numerical example.

APPENDIX D

ATTITUDE SERVO NUMERICAL EXAMPLE

A specific numerical case of the attitude servo system developed in section 4 is presented in this appendix. The control algorithm for one intervening integrator is summarized in table 4-1. The regulator-control law chosen for this example is a simple, spherically symmetric (scalar gains) linear law with limited position feedback, namely

$$u_\epsilon = -sat(20.0\epsilon^{(0)}, 1) - 18.14\epsilon^{(1)} - 7.83\epsilon^{(2)} \tag{D-1}$$

The feedback gains place the three poles at $(s = -5,\ \zeta = 0.707, \omega_n = 2)$. The position feedback saturates at 1/sec^3 for each axis. Thus, saturation occurs for $|\epsilon_i| = \sin(\phi/2)|u_i| = 0.05$, which, for $u_i = 1$, is approximately 6 deg. The overall block diagram of the servo system is shown in figure 4.3.

Now consider the input, given by the command generator as an Euler angle form $AF_q^3(\alpha_{cr})$. Let the Euler sequence $q = (1, 2, 3)$, so that

$$C_{cr} = E_1(\alpha_{cr1})E_2(\alpha_{cr2})E_3(\alpha_{cr3})$$

The maneuver consists of four segments, each lasting 4 sec, as shown in figures D-1 and D-2. Figure D-1 shows the commanded motion in the yaw-pitch plane. Initially all three commanded angles are zero. The initial segment takes pitch α_{cr2} and yaw α_{cr3} to 50 and –90 deg, respectively, and, as shown in figure D-2, brings the roll rate $\dot{\alpha}_{cr1}$ to 1 rad/sec. In the second segment the pitch and roll rates are held constant while the yaw scans from –90 to 90 deg. During segments 3 and 4 the command remains at $\dot{\alpha}_{cr1} = 1$ rad/sec, $\alpha_{cr2} = 50$ deg, and $\alpha_{cr3} = 90$ deg, while the second component of the disturbance ν is raised from 0 to 0.1 rad/sec^2, as shown in the bottom panel in figure D-3. The other components $\nu_1 = \nu_3 = 0$. This figure also shows the command in terms of the angular velocity and two of its derivatives.

The state estimate is assumed to be exact; the disturbance is estimated erroneously by $\hat{\nu} \equiv 0$. The plant, which is initially offset from the origin by $\alpha_{br} = (15,\ \ 15,\ \ -15)$ deg, responds as shown in figure D-4 in terms of Euler angles and in figure D-5 in terms of the angular velocity and its derivatives.

The regulator error and control $SF^3(\epsilon)$ are shown in figure D-6. It may be noted that the position feedback saturates, thereby reducing the error at constant rate, as shown in the second panel in the figure. The initial-condition transient is essentially over during the first segment. The final steady offset in ϵ_2 counteracts the disturbance in the usual way. Figure D-7 shows the error in a phase plane, where constant-speed sections are clearly visible. Note also that the changes in the input $AF_q^3(\alpha_{cr})$ are practically decoupled from the error. Finally, figure D-8 shows the tracking in the yaw-pitch plane with initial offset and disturbance. The hangoff in terms of Euler angle error is not steady (see fig. D-4).

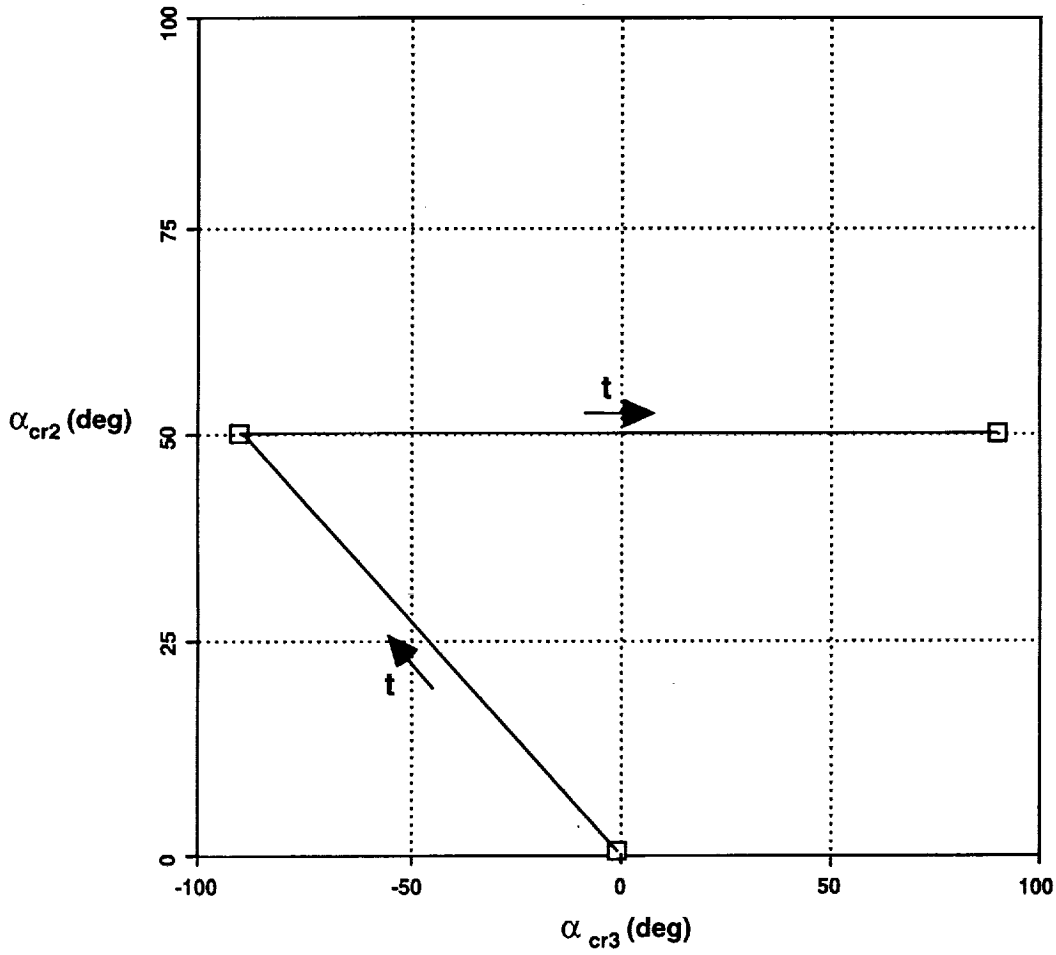

Figure D-1. Evolution of commanded Euler angle form $AF^3_{123}(\alpha_{cr})$ in yaw-pitch plane.

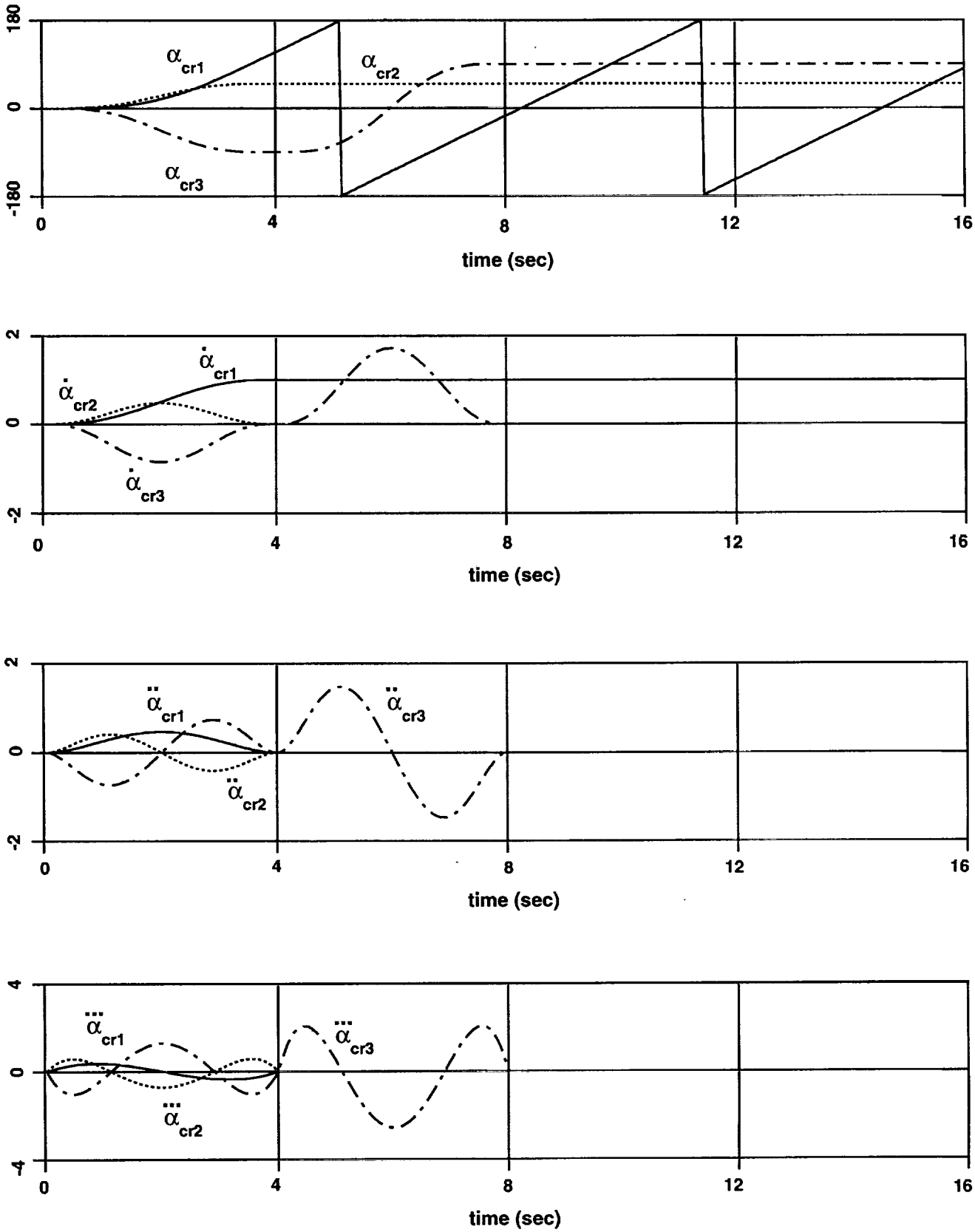

Figure D-2. Evolution of commanded Euler angle form $AF^3_{123}(\alpha_{cr})$.

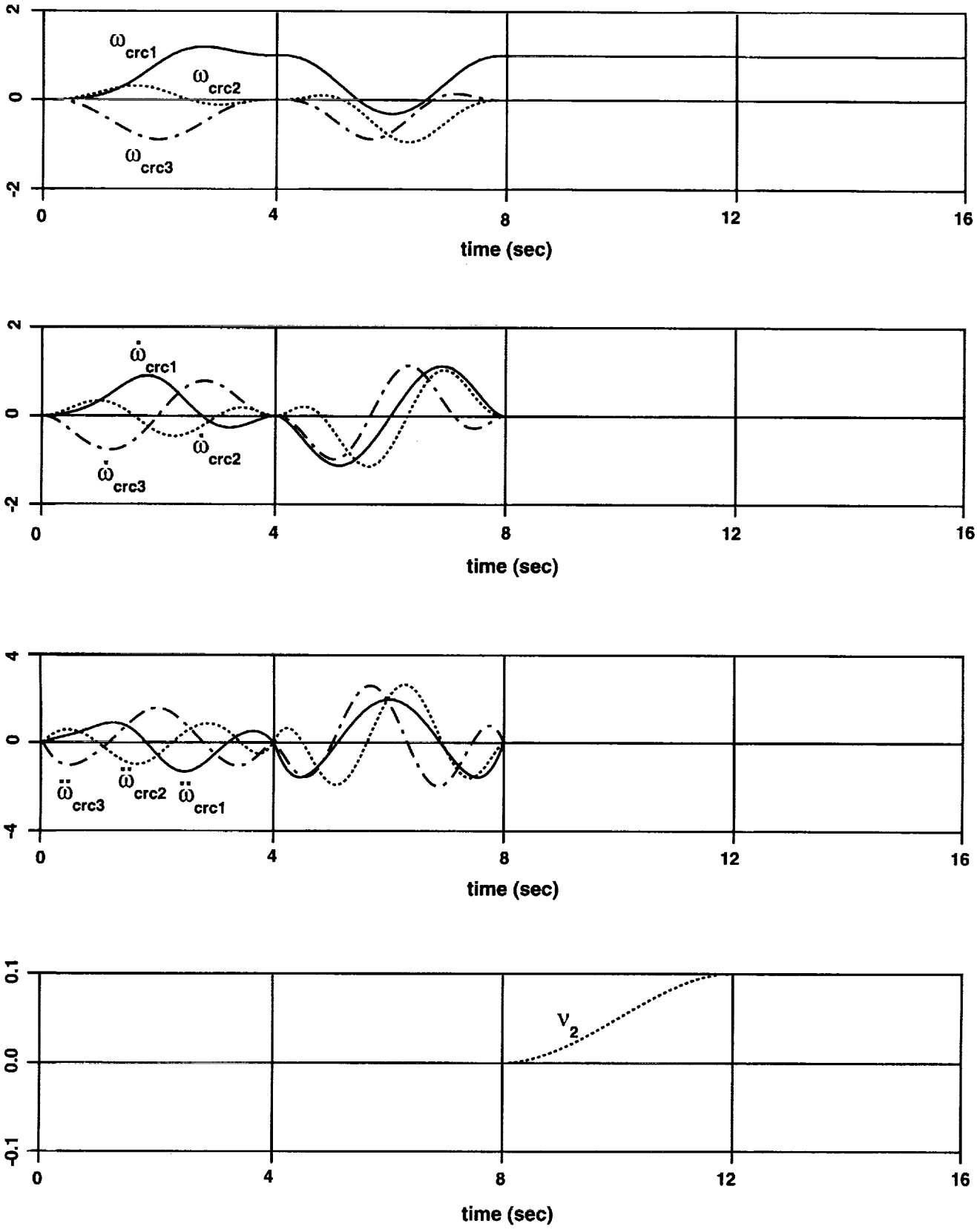

Figure D-3. Evolution of commanded rotational form RF_{cr}^{3} and disturbance ν.

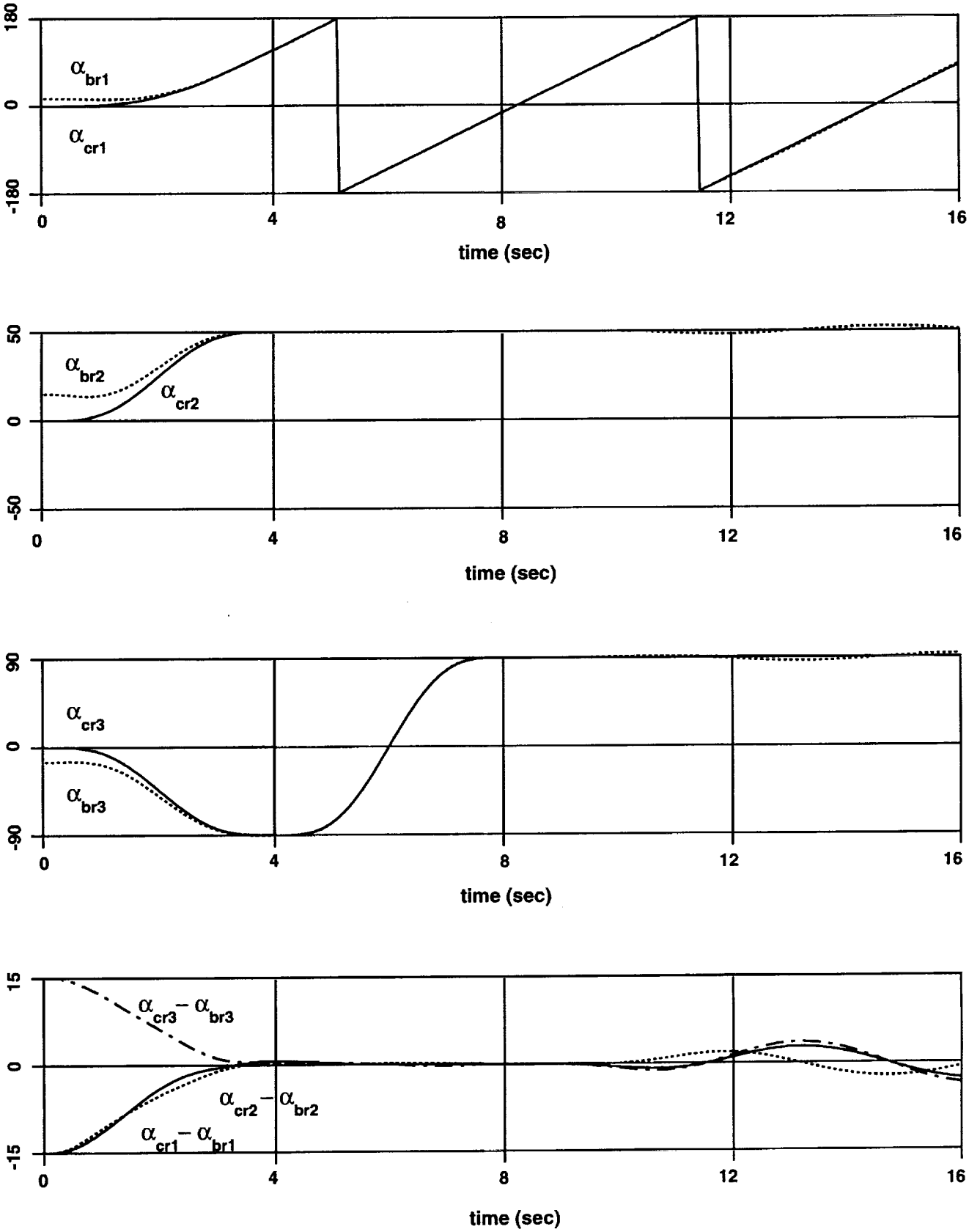

Figure D-4. Input $AF_{123}^{3}(\alpha_{cr})$–output $AF_{123}^{3}(\alpha_{br})$ tracking.

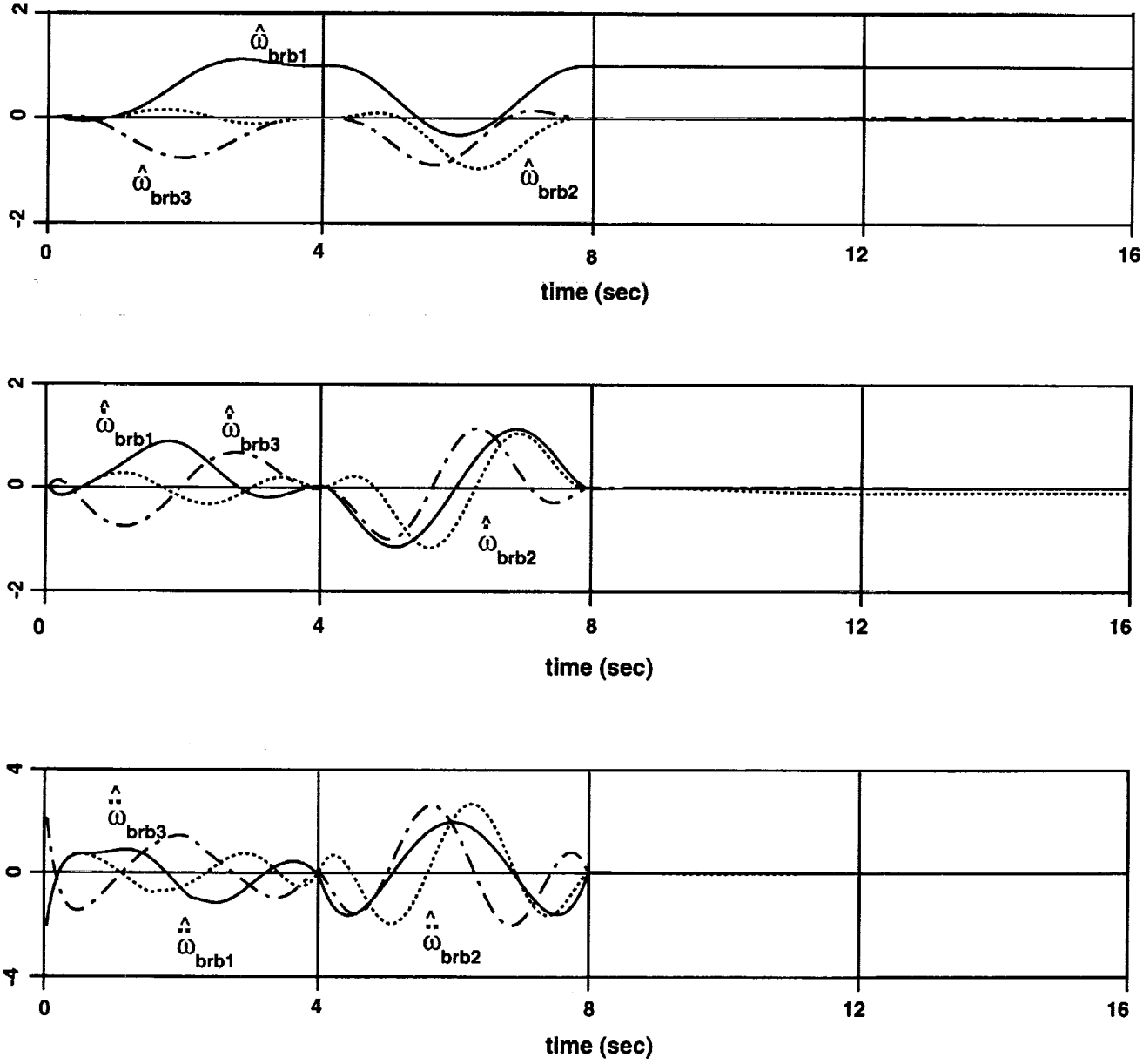

Figure D-5. Evolution of estimated body rotation form RF_{br}^{3}.

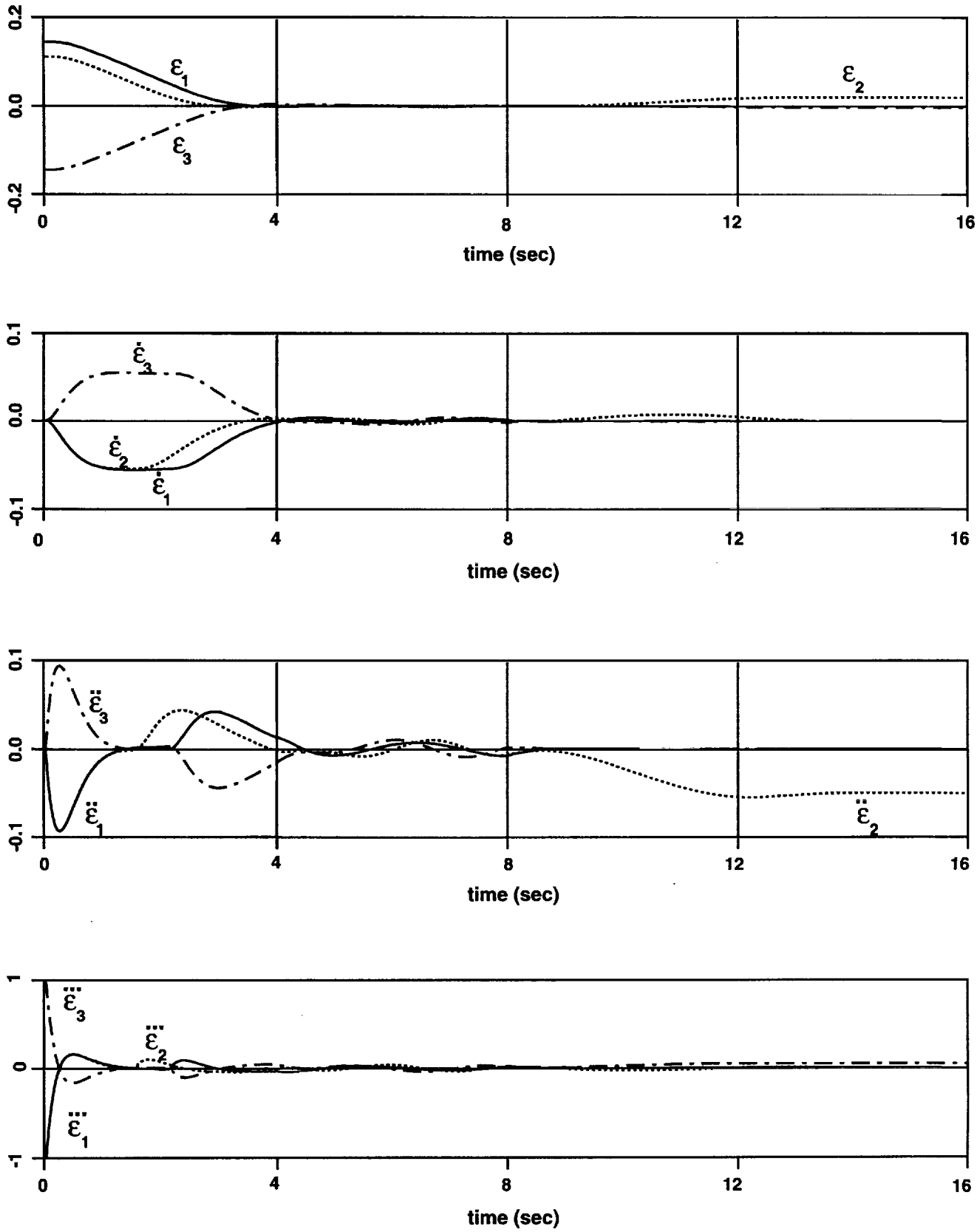

Figure D-6. Evolution of error Euler parameter form PF_{bc}^{3}.

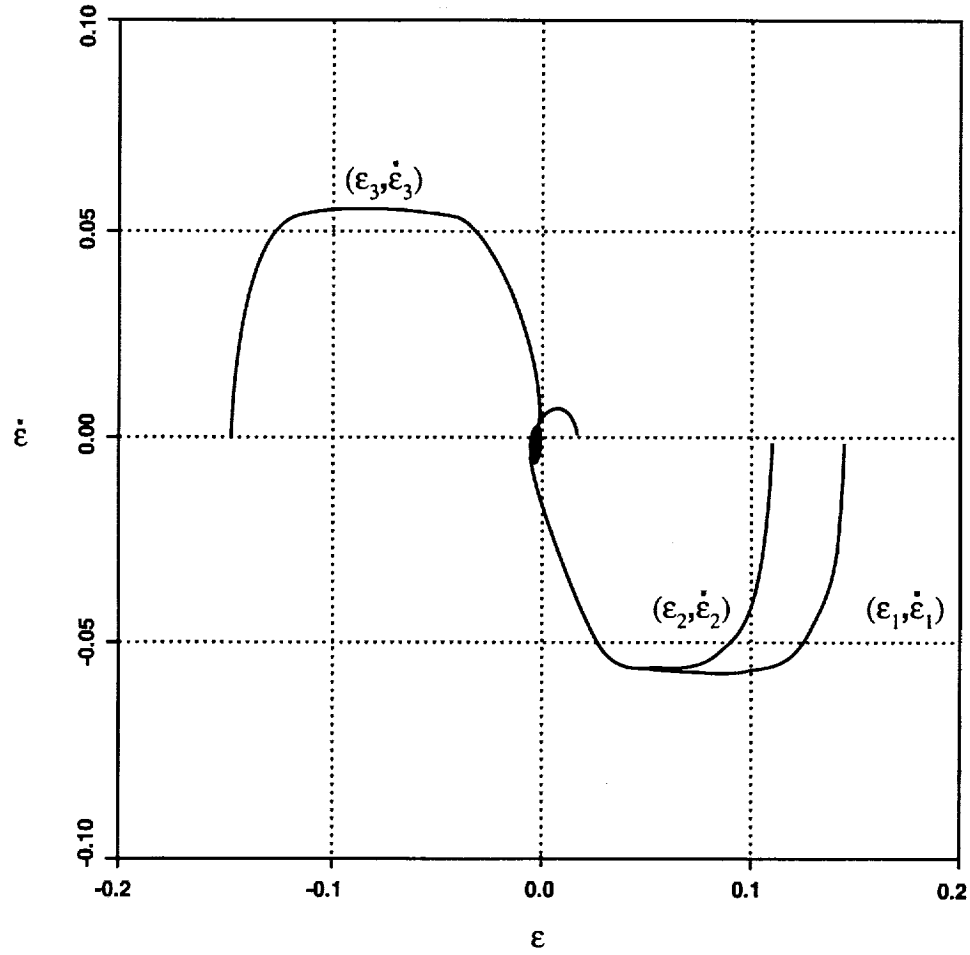

Figure D-7. Phase plane of error Euler parameter form PF_{bc}^{3}.

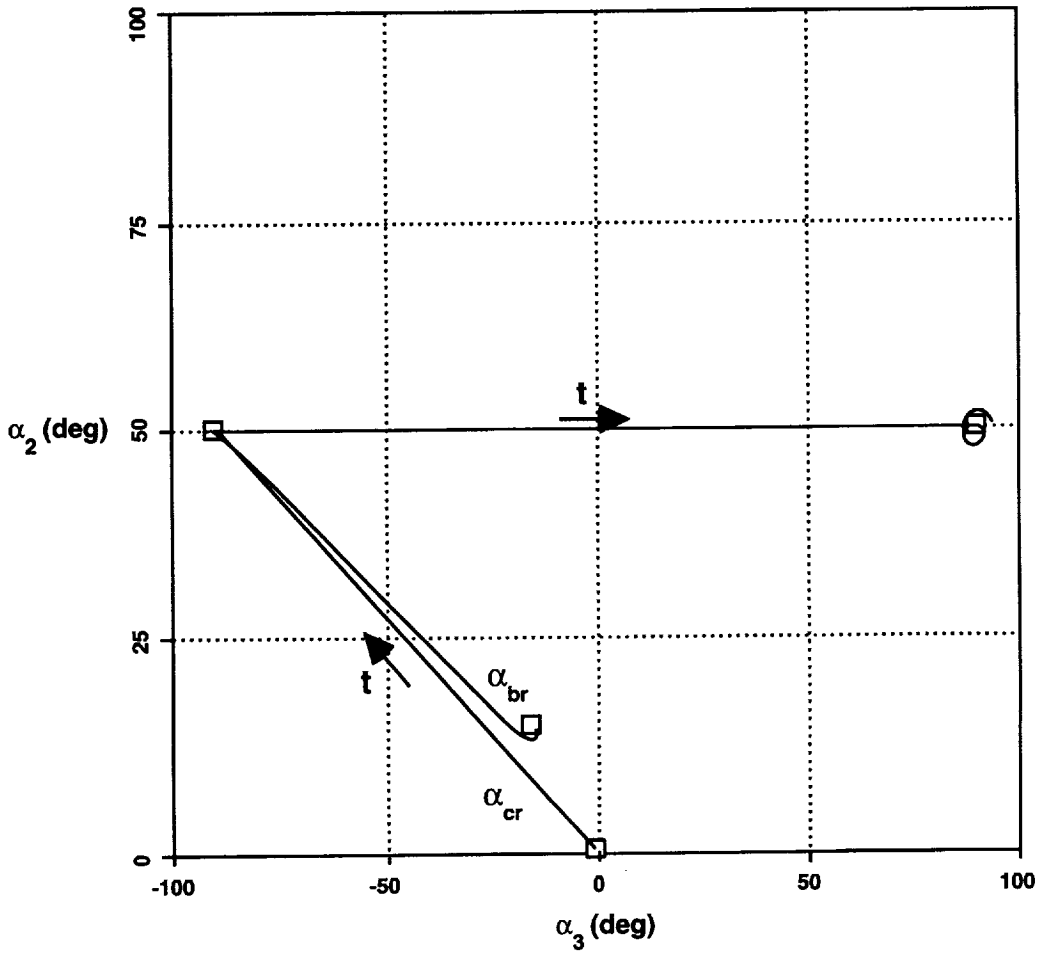

Figure D-8. Tracking in yaw-pitch plane with initial offset and disturbance.

REFERENCES

1. Hughes, P. C.: Spacecraft Attitude Dynamics. John Wiley and Sons, 1986.

2. YAV-8B Simulation and Modeling, Report MDC A7910, vol. 1, McDonnell Douglas Corporation, St. Louis, Missouri, 1982.

3. Meyer, G.; and Cicolani, L.: Application of Nonlinear System Inverses to Automatic Flight Control. Theory and Applications of Optimal Control in Aerospace Systems, P. Kant, ed., AGARDograph No. 251, 1981.

4. Smith, G. A.; and Meyer, G.: Applications of the Concept of Dynamic Trim Control to Automatic Landing of Carrier Aircraft. NASA TP-1512, 1980.

5. Doyle, J.; Glover, K.; Khargonekar, P.; and Francis, B.: State-Space Solutions to Standard h_2 and H_∞ Control Problems. IEEE Transactions on Automatic Control, AC-34(8), 1989, pp. 831–847.

6. Francis, B. A.: The Linear Multivariable Regulator Problem. SIAM Journal on Control and Optimization, vol. 15, 1977, pp. 486–505.

7. Meyer, G.; and Cicolani, L.: A Formal Structure for Advanced Flight Control Systems. NASA TN D-7940, 1975.

8. Franklin, J. A.; Hynes, C. S.; Hardy, G. H.; Martin, J. L.; and Innis, R. C.: Flight Evaluation of Augmented Controls for Approach and Landing of Powered-Lift Aircraft. AIAA Journal of Guidance, Control, and Dynamics, vol. 9, 1986, pp. 555–565.

9. Franklin, J. A.; and Engelland, S. A.: Design and Piloted Simulation Evaluation of Integrated Flight/Propulsion Controls for STOVL Aircraft. AIAA Conference Paper 91-3108, Proceedings of the Aircraft Design Systems and Operations Meeting, Baltimore, Maryland, 1991.

10. Smith, G. A.; and Meyer, G.: Aircraft Automatic Flight Control System with Model Inversion. C. T. Leondes, ed., Control and Dynamic Systems, vol. 38, 1990.

11. Menon, P. K.; Badgett, M. E.; Walker, R. A.; and Duke, E. L.: Nonlinear Flight Test Trajectory Controllers for Aircraft. Journal of Guidance, Control, and Dynamics, vol. 10, no. 1, 1987, pp. 67–72.

12. Romano, J. J.; and Singh, S. N.: I-O Map Inversion, Zero Dynamics and Flight Control. IEEE Transactions on Aerospace and Electronic Systems, vol. 26, no. 6, Nov. 1990.

13. Lane, S.; and Stengel, R. F.: Flight Control Design Using Nonlinear Inverse Dynamics. Automatica, vol. 24, 1988, pp. 471–484.

14. Isidori, A.: Nonlinear Control Systems. Second ed., Springer–Verlag, 1989.

15. Slotine, J.-J. E.; and Li, W.: Applied Nonlinear Control. Prentice Hall, 1991.

16. Meyer, G.; Su, R.; and Hunt, R. L.: Application of Nonlinear Transformations to Automatic Flight Control. Automatica, vol. 20, no. 1, 1984, pp. 103–107.

17. Isidori, A.; and Byrnes, C.: Output Regulation of Nonlinear Systems. IEEE Transactions on Automatic Control, vol. 35, no. 2, 1990, pp. 131–140.

18. Huang, J.; and Rugh, J.: Stabilization on Zero-Error Manifolds and the Nonlinear Servomechanism Problem. IEEE Transactions on Automatic Control, vol. 37, no. 7, July 1992.

19. Akhrif, O.; and Blankenship, G. L.: Symbolic Computation in Differential Geometry Applied to Nonlinear Control Systems. Symbolic Computation: Applications to Scientific Computing, Robert Grossman, ed., SIAM, 1989, pp. 53–75.

20. Campbell, S. L.; Moore E.; and Zhong, Y.: Utilization of Automatic Differentiation in Control Algorithms. Proceedings of the 31st IEEE Conference on Decision and Control, Dec. 1992.

21. Griewank, A.; and Corliss, G. F.: Automatic Differentiation of Algorithms: Theory, Implementation, and Application. Proceedings of the SIAM Workshop on Automatic Differentiation of Algorithms, SIAM, 1991.

22. Meyer, G.: Dynamic Forms and Their Application to Control. NASA TM-103936, April 1992.

23. Courant, R.; and McShane, E. J.: Differential and Integral Calculus. Vol. 1. Interscience, New York, 1955.

24. Sifer, J. F.; and Meyer, G.: Modeling and Model Inversion Issues in a Nonlinear Automatic Flight Control System for the YAV-8B Harrier V/STOL Aircraft. Memo UCB/ERL M89/4, University of California, Berkeley, 1989.

25. Cheng, V. H. I.; and Sridhar, B.: Considerations for Automated Nap-Of-The-Earth Rotorcraft Flight. AHS Journal, vol. 36, no. 2, April 1991, pp. 61–69.

26. Meyer, G.: The Design of Nonlinear Exact Model Followers. Proceedings of Joint Automatic Control Conf., 1981, FA3A.

27. Meyer, G.; Wehrend, W.; and Lee, H. Q.: A Method for Expanding a Direction Cosine Matrix into an Euler Sequence of Rotations. NASA TM X-1384, 1967.

28. Meyer, G.: On the Use of Euler's Theorem on Rotations for the Synthesis of Attitude Control Systems. NASA TN D-3643, 1966.

29. Paielli, R. A.; and Bach, R. E.: Attitude Control with Realization of Linear Error Dynamics. Journal of Guidance, Control, and Dynamics, vol. 16, no. 1, 1993.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1993	Technical Paper

4. TITLE AND SUBTITLE
Dynamic Forms Part I: Functions

5. FUNDING NUMBERS
505-64-52

6. AUTHOR(S)
George Meyer and G. Allan Smith

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ames Research Center
Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER
A-93078

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TP-3397

11. SUPPLEMENTARY NOTES
Point of Contact: George Meyer, Ames Research Center, MS 210-3, Moffett Field, CA 94035-1000
(415) 604-5750

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified — Unlimited
Subject Category 31

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

The formalism of dynamic forms is developed as a means for organizing and systematizing the design of control systems. The formalism allows the designer to easily compute derivatives to various orders of large composite functions that occur in flight-control design. Such functions involve many function-of-a-function calls that may be nested to many levels. The component functions may be multiaxis, nonlinear, and they may include rotation transformations.

A dynamic form is defined as a variable together with its time derivatives up to some fixed but arbitrary order. The variable may be a scalar, a vector, a matrix, a direction cosine matrix, Euler angles, or Euler parameters. Algorithms for standard elementary functions and operations of scalar dynamic forms are developed first. Then vector and matrix operations and transformations between parameterization of rotations are developed in the next level in the hierarchy. Commonly occurring algorithms in control-system design, including inversion of pure feedback systems, are developed in the third level.

A large-angle, three-axis attitude servo and other examples are included to illustrate the effectiveness of the developed formalism. All algorithms have been implemented in FORTRAN code. Practical experience shows that the proposed formalism may significantly improve the productivity of the design and coding process.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Automatic control, Algorithms, Differentiation	84
	16. PRICE CODE A05

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		