



System Simulation by Recursive Feedback: Coupling a Set of Stand-Alone Subsystem Simulations

D.D. Nixon

Marshall Space Flight Center, Marshall Space Flight Center, Alabama

The NASA STI Program Office...in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

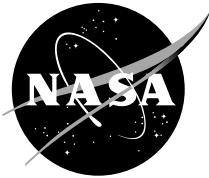
- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.

- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.
- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and mission, often concerned with subjects having substantial public interest.
- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results...even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at <http://www.sti.nasa.gov>
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA Access Help Desk at (301) 621-0134
- Telephone the NASA Access Help Desk at (301) 621-0390
- Write to:
NASA Access Help Desk
NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320
(301)621-0390



System Simulation by Recursive Feedback: Coupling a Set of Stand-Alone Subsystem Simulations

D.D. Nixon

Marshall Space Flight Center, Marshall Space Flight Center, Alabama

National Aeronautics and
Space Administration

Marshall Space Flight Center • MSFC, Alabama 35812

TRADEMARKS

Trade names are used in this report for identification purposes only. This usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
(703) 487-4650

TABLE OF CONTENTS

1. INTRODUCTION	1
2. RECURSIVE FEEDBACK ALGORITHM	2
2.1 Background	2
2.2 Concept	3
2.3 Insights for Specific Design	4
3. EXAMPLE SYSTEM	8
4. MATHEMATICAL MODELS	9
4.1 Orbit Dynamics	9
4.2 Attitude Dynamics	9
4.3 Aerodynamics	10
5. SUBSYSTEM SIMULATIONS	13
6. SYSTEM SIMULATIONS	14
7. NUMERICAL RESULTS	17
7.1 Run Case Definitions	17
7.2 Response Comparisons	18
8. PERFORMANCE	26
9. IMMEDIATE FUTURE DEVELOPMENT	28
10. CONCLUSIONS AND RECOMMENDATIONS	29
APPENDIX A—RESPONSE PLOTS	31
A.1 Response Plots for Case 1 (Full Three-Axis Inertial Attitude Control)	31
A.2 Response Plots for Case 2 (Two-Axis Attitude Control)	51
A.3 Response Plots for Case 3 (No Attitude Control)	69
A.4 Response Plots for Cases 1–3 With Central and Distributed Coupling	81

TABLE OF CONTENTS (Continued)

APPENDIX B—FORTRAN LISTINGS	89
B.1 FORTRAN Listings, Including Main Program and All Subroutines, Recursive Feedback With Centralized Coupling (RF-C)	89
B.2 FORTRAN Listings, Including Main Program and All Subroutines Not Previously Listed in A.1, Recursive Feedback With Distributed Coupling (RF-D)	147
B.3 FORTRAN Listings, Including Main Program and All Subroutines Not Previously Listed in A.1, Unified Approach, Runge-Kutta Method (RK-U)	167
REFERENCES	187

LIST OF FIGURES

1.	Schematic of recursive feedback process	3
2.	Spacecraft properties and geometric body-fixed coordinate system	8
3.	Relationship of body frame to geometric frame	11
4.	Centrally coupled simulation diagram.....	15
5.	Function of system-level coupler.....	15
6.	Distributively coupled simulation diagram.....	16
7.	Inertial coordinate system/initial orbit plane geometry	17
8.	Case 3, Euler angle, X axis (deg)	19
9.	Case 3, Euler angle, Y axis (deg)	19
10.	Case 3, Euler angle, Z axis (deg)	20
11.	Case 3, body rate, X axis (deg)	20
12.	Case 3, body rate, Y axis (deg)	21
13.	Case 3, body rate, Z axis (deg)	21
14.	Case 3, altitude, first time segment (km)	22
15.	Case 3, altitude, second time segment (km)	22
16.	Case 3, altitude, third time segment (km)	23
17.	Altitude, case-method array, full run duration (km)	24
18.	Altitude, case-method array, first time segment (km)	24
19.	Altitude, case-method array, second time segment (km).....	25

LIST OF ACRONYMS

MARSYAS	Marshall System for Aerospace Simulation
MSFC	Marshall Space Flight Center
ODE	ordinary differential equation
RF-C	centrally coupled recursive feedback
RF-D	distributively coupled recursive feedback
RK-U	unified Runge-Kutta

NOMENCLATURE AND SYMBOLS

A	state gain matrix or attitude direction cosine matrix or panel area, depends on context
a	attitude control gain
aero	aerodynamic
B	input gain matrix
C_d	aerodynamic drag coefficient
cm	center of mass
cpi	center of pressure for the i th facet
d	drag
e	base of natural logarithm
\vec{F}	force
$f(X, t)$	mathematical function of state vector and time
G	gravitational constant
gg	gravity gradient
grav	gravity
H	height of spacecraft
I	moment of inertia matrix
i	integer variable
K	feedback gain matrix
k	loop gain
L	length of spacecraft
m	mass
n	integer variable
\hat{n}	unit vector normal to surface
R	orbit radius
\vec{R}	orbit radius vector
\vec{r}_{cm}	center of mass location vector
\vec{r}_{cpi}	center of pressure location vector for the i th facet
\hat{r}	unit vector in direction of orbit radius

NOMENCLATURE AND SYMBOLS (Continued)

sat	satellite
\vec{T}	torque
t	time
U	column matrix collection of all subsystem input vectors
u	subsystem input vector
V	orbit velocity
\vec{V}	orbit velocity vector
\hat{v}	unit vector in direction of orbit velocity
W	width of spacecraft
X	coordinate system axis or system-level state vector, depends on context
\dot{X}	time derivative of system-level state vector
x	coordinate system axis or subsystem state vector, depends on context
\dot{x}	time derivative of subsystem state vector
Y	coordinate system axis or system-level output vector, depends on context
y	coordinate system axis or subsystem output vector, depends on context
\dot{Y}	time derivative of system-level output vector
\dot{y}	time derivative of subsystem output vector
\ddot{y}	second time derivative of subsystem output vector
$y(0)$	initial value of $y(t)$
Z or z	axis of coordinate system
ρ	atmospheric density
ϕ	attitude angle
$\dot{\phi}$	attitude rate
ω	body rate
$\bar{\omega}$	angular velocity vector
$\dot{\bar{\omega}}$	angular velocity vector

TECHNICAL PUBLICATION

SYSTEM SIMULATION BY RECURSIVE FEEDBACK: COUPLING A SET OF STAND-ALONE SUBSYSTEM SIMULATIONS

1. INTRODUCTION

In design and development of space systems, digital dynamic simulations have potential to reduce physical testing requirements, increase confidence in systems that cannot be tested, and ensure that subsystem requirements are not more stringent than necessary. However, simulations that address interactions of multiple disciplines and subsystems can be difficult to build. Knowledge and expertise required at the physics level, especially for high-fidelity developments, do not normally reside within one person or organization for more than one discipline or subsystem. Simulations involving individual disciplines or subsystems usually exist or can be readily constructed, and system engineers often understand the qualitative nature of interactions among them; but a means of obtaining quantitative-coupled system-level results is often unavailable. Therefore, specific practical approaches to construction of high-fidelity system-level simulations are being developed from numerous perspectives across the engineering community.

Digital dynamic simulations are often built by deriving or constructing differential-algebraic equations that define the system model and implementing algorithms to solve those equations numerically, as a single coupled set, using one of several standard or modified numerical integration methods. In some cases, software is written in a language such as FORTRAN or C to define the equations, and an existing or slightly modified ordinary differential equation (ODE) solver, perhaps a Runge-Kutta implementation, is employed to perform the integration. In other cases, an in-house development such as Marshall System for Aerospace Simulation (MARSYAS) or a commercial off-the-shelf product such as MATLAB[®]/SIMULINK[®] is used to act as a higher level facilitator of what is ultimately a mathematically similar approach.

System-level digital solution of coupled “stand-alone” dynamical simulations is a fundamentally different approach to system simulation, and fundamentally different numerical procedures are required. Recursive feedback is a conceptual method from which a family of appropriate algorithms, processes, and specific numerical procedures can be derived.

Coupled-subsystem-simulation architecture provides near complete independence of stand-alone subsimulations and naturally facilitates high fidelity and broad scope through collaboration across interfaces that can be implemented in the same physical and engineering terms that define them in the real system. Such simplicity promotes clarity of communication and ease of understanding, both of which have many positive benefits. Individual subsimulations can potentially be implemented on separate, remote, and dissimilar computational platforms, and this portends numerous advantages and possibilities as computer network capabilities improve.

2. RECURSIVE FEEDBACK ALGORITHM

2.1 Background

Brief discussions of high-fidelity, multidiscipline simulation are set forth in references 1 and 2, which resulted from association with the Structures and Dynamics Laboratory High-Fidelity Multidiscipline Simulation Development team. Reference 1 attempts to illustrate the long-term goal of multidiscipline simulation by example, while reference 2 presents the goal in a more generalized way. Reference 3 documents development of an earlier FORTRAN general purpose dynamical simulation tool (ANASIM) that uses recursive feedback as the system solution algorithm. This tool allows one to construct a simulation by choosing system elements from a menu and specifying connections (signal flow) among them. The elements (integrators, summers, limiters, etc.) are, in effect, small stand-alone subsimulations. A definition and explanation of the method is presented, and an analytical demonstration of response computation for a first-order, single-state, constant-coefficient system with a step input is included. The demonstration, or mathematical “show that,” is accomplished through symbolic algebraic application of the algorithm followed by direct comparison of results with the exact closed-form solution.

In reference 4, the method is analytically demonstrated for some simple examples within a variety of system classes. The first is the n -dimensional linear constant-coefficient case defined by the matrix vector equation $\dot{x}(t) = Ax(t) + Bu(t)$, which is a standard state-variable form. The remaining four are scalar (single state) cases that include two nonlinear, continuous cases; a linear time-varying coefficient case; and a numerically demonstrated case involving a state-dependent discontinuity and an algebraic loop. The analyses of reference 4 provide additional mathematical insights into the process, particularly with respect to convergence.

Analysis of the method is extended to coupling of stand-alone subsystems in references 5 and 6. Reference 5 mathematically defines and analytically demonstrates recursive feedback for generation of responses from a generic linear-dynamic system composed of internally inaccessible subsystems, and reference 6 continues beyond to include subsystems that have potential to produce algebraic loops at the system level because they contain direct feed-through of input.

References 5 and 6 specifically address process convergence and essentially guarantee that recursive feedback can be used as a system-level solution algorithm for simulation of linear systems composed of coupled stand-alone subsystems for a broad range of ordinary circumstances. Combined with the analyses of reference 4 and experience associated with ANASIM (ref. 3), it seems reasonable to expect that the method can be dependably applied to nonlinear system simulation. Only experience, and perhaps more analysis, will determine the effectiveness of extrapolation to these supposedly more stringent conditions.

2.2 Concept

Referring to figure 1, the subsystems are represented by independent, self-contained, time-domain, dynamical simulations that can be commanded to run from a given set of initial conditions over a predetermined segment of time when provided with input signals as functions of time over that interval. In general, the time segment, also referred to as a convergence interval, is short compared with the total desired system simulation duration, but may be significantly larger than the integration step size associated with a typical subsimulation. For each new stage of recursion, subsystem input signals are computed using original system-level input signals summed with system-level feedback (coupling) signals generated at the current stage. New (next recursion stage) subsystem responses are computed using new subsystem input signals in conjunction with the same (original) initial conditions. The process continues recursively until convergence is achieved. After convergence, initial conditions are replaced with final conditions and the process is restarted for the next segment of time. System response over the total desired duration of the simulation is the concatenated set of combined subsystem responses over many time segments.

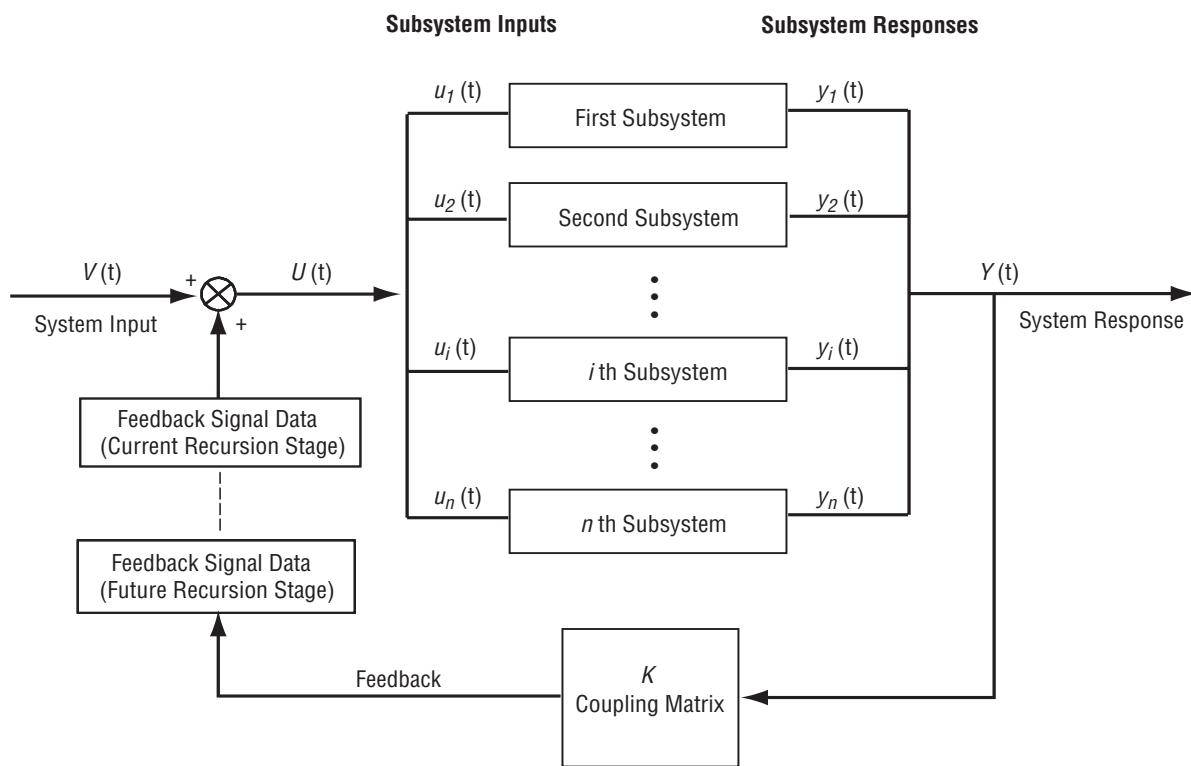


Figure 1. Schematic of recursive feedback process.

2.3 Insights for Specific Design

Because figure 1 describes a system of considerable generality whose complexity for a specific case can range from simple to great, further clarification of the process seems appropriate. The simplest “system” could arguably be a single integrator in the forward path, a constant multiplier (gain) in the backward (feedback) path, no (zero) system-level input, and a nonzero initial condition on the output. In that case, there is one linear subsystem, and that subsystem is a single integrator. The output function is the integral of the subsystem-level input function with respect to time, and can be generated independently and directly by simply computing the area under the input curve over the time interval of interest (convergence interval) and adding the initial condition. The coupling matrix becomes a scalar (value of the loop gain), and actually represents direct feedback rather than coupling. The process, however, does not explicitly make such a distinction, nor does it need to. The summing junction becomes moot since there is no system-level input. The recursive transfer point remains as depicted in figure 1, but does not exist in the analog system to be digitally simulated. The scalar differential equation that models the analog system is $\dot{y}(t) = ky(t)$, where t is time and k is the loop gain, and the exact closed-form solution for the output response is given by the exponential function

$$y(t) = y(0) [e^{kt}] , \quad (1)$$

where $y(0)$ represents the initial value of the output $y(t)$.

Applying recursive feedback to this system, the output is first computed as a function of time over the convergence interval as if there were no (zero) feedback. This function is referred to as the stage zero output response. Because the system input function is also zero in this case, the response is a constant function of time symbolically represented by

$$y^{(0)}(t) = y(0) , \quad (2)$$

where the parenthesized superscript is the stage index. Next, the stage zero feedback function is computed by multiplying the stage zero output response function by the loop gain k . After “moving” the feedback function data across the recursive transfer point and summing it with the system-level input function (zero), it becomes the stage one subsystem input function. Stage one response can now be computed by numerical integration (area under the curve) of the new (stage one) input function while applying the original initial condition $y(0)$. A symbolic representation of the result is

$$y^{(1)}(t) = y(0)[1 + kt] . \quad (3)$$

Repeating the process, symbolic representation of the stage two result is

$$y^{(2)}(t) = y(0)[1 + kt + \frac{(kt)^2}{2!}] . \quad (4)$$

By induction, the n th stage response is given by

$$y^{(n)}(t) = y(0) \left[1 + kt + \frac{(kt)^2}{2!} + \cdots + \frac{(kt)^n}{n!} \right]. \quad (5)$$

Clearly, if sufficiently continued, the recursive process yields a solution that approaches the exact solution $y(t) = y(0) [e^{kt}]$. In theory, the time interval (convergence interval) for this case can be as long as desired; in practice, it must be restricted because of numerical integration and roundoff errors. As mentioned earlier, system response over the problem duration is represented by a concatenation of responses over as many convergence intervals as needed, and a single convergence interval is measured in multiple integration steps. It is noted that each recursion stage improves accuracy of the approximate system response by adding the next higher order term in a series expansion of the exact system response. This type of behavior is seen in all of the linear system analyses of references 3–6. For the nonlinear systems of reference 4, each recursion stage adds higher order terms; but for a given number of correct terms in the series expansion, a greater number of recursion stages is required. For these systems, the convergence interval cannot be arbitrarily long, even without integration and roundoff errors, and the degree of restriction required depends on the initial condition.

Considering the system in figure 1 in its more general context, the off-diagonal elements of the coupling matrix represent the means by which output from one subsystem is processed and channeled to construct input for another. Because direct dependence of the input to any subsystem on the output of that same subsystem is likely to have been handled internally, the elements on the diagonal are likely to be zero; although, as demonstrated, the method does not require this to be the case. In development of a system simulation by coupling of subsystem simulations, the nature of the coupling matrix is, to a large degree, a design choice.

Consider, for example, the “system” to be an undamped oscillator governed by the scalar differential equation $\ddot{y}(t) + \omega^2 y(t) = 0$, where ω is the frequency of oscillation. A simulation of this system by recursive feedback can be set up in at least two very different ways. One design possibility is to define a single subsystem, best described as a double integrator, or two single integrators in series. The output function of the subsystem is determined by integrating its input function twice. The coupling matrix once again degenerates to a scalar constant, $-\omega^2$, and the system input function is zero. Assuming a nonzero initial condition $y(0)$ on the output (displacement), application of recursive feedback to this system results in a truncated series definition of $y(t)$, and at the n th recursion stage, a symbolic representation is

$$y^{(n)}(t) = y(0) \left[1 - \omega^2 \frac{t^2}{2!} + \omega^4 \frac{t^4}{4!} + \cdots + (-1)^n \omega^{2n} \frac{t^{2n}}{(2n)!} \right]. \quad (6)$$

Then for a sufficiently large number of recursion stages, the approximate solution approaches the exact solution $y(t) = y(0) [\cos \omega t]$. Once again the number of terms in the series corresponds directly to the number of recursion stages, but the order of the series with respect to time is twice the number of recursion stages.

In contrast, a second design possibility is to define two subsystems that are both single integrators. The governing differential equation is now given by the two-dimensional matrix-vector equation

$$\dot{Y}(t) = KY(t) , \quad (7)$$

the coupling matrix is given by

$$K = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix}, \quad (8)$$

and the output function is given by

$$Y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}, \quad (9)$$

where $y_1(t)$ represents displacement and $y_2(t)$ represents velocity. Application of recursive feedback to this system results in a truncated series definition of $Y(t)$. At the n th recursion stage, the numerically computed approximate system response function is symbolically represented by

$$Y^{(n)}(t) = \left[I + Kt + K^2 \frac{t^2}{2!} + \cdots + K^n \frac{t^n}{n!} \right] Y(0) . \quad (10)$$

Again, each numerical recursion stage, in effect, adds one term to the truncated series approximation of the exact solution.

For the two-subsystem, twin-integrator approach, it takes two recursion stages to reach the same order of approximation in time that was reached in one stage with the single-subsystem, double-integrator approach. However, the twin integrators represent independent operations that can be performed in parallel, while the double integrator represents two operations that must be performed in series. In the double integrator case, velocity, $\dot{y}(t)$, does not appear at the system level, only one function crosses the recursive transfer point, and only one function is checked for convergence. In the twin integrator case, velocity, $y_2(t)$, appears at the system level, both $y_1(t)$ and $y_2(t)$ cross the recursive transfer point, and both functions can be checked for convergence at the system level. Clearly, in a more complex situation, the relative merits of these approaches become simulation design and performance issues.

For the broader range of systems contained within the framework of figure 1, the nature of the coupling matrix is a function of many possible simulation design choices. In its most basic form, a coupler would simply route signals from one subsystem to another without modifying them. In its most complex form, it could become a multidimensioned nonlinear operator as well. Typical coupler functions are likely to be coordinate transformations, gain multiplications, interpolations, and other data modification or routing tasks that must be performed to make proper interface connections among a set of predefined or existing subsimulations.

As implied by the double integrator example (when viewed as two single integrators in series), it is also possible to distribute the coupling functions throughout the system. In a particular situation, one might choose to view several subsimulations as a combined set if the nature of the coupling among them is such that an output response can be determined from an input function without necessity for recursion (there is no feedback loop internal to the set). In that case, processing information as it is transferred from one set member to another represents a coupling function that does not reside as an element of a centralized coupling matrix. Utilizing the powers of block diagram algebra or signal flow analysis, many arrangements of subsystems and couplers may be possible for representation of a given system. Such design freedom can have a significant impact on the configuration, performance, and efficiency of the simulation.

To be of practical value in dynamical systems simulation, the recursive feedback concept must be applicable to nonlinear models. The “guarantees” and insights that come with analyses like that of reference 6 are weakened at best and totally lost at worst without the linear model assumption. However, one does have a guarantee of solution existence and uniqueness for a broad range of sufficiently well-behaved nonlinear systems (ref. 7). A converged recursive feedback process is indicative of a solution with exception of questions concerning discretization errors, roundoff errors, inappropriate tolerances, etc. The potential for convergence can normally be improved by reducing the length of the convergence interval, not unlike reducing integration stepsize in a conventional situation when satisfactory results are not achieved.

3. EXAMPLE SYSTEM

The example system is designed to be simple but to reflect real possibilities. It consists of a satellite that has an attitude control system, is in low-Earth orbit, and is experiencing aerodynamic effects in addition to gravitational effects. The spacecraft is idealized as a rigid body of rectangular “box” shape, illustrated in figure 2. The center of mass is offset from the geometric center by a small amount in all three axes, and moments of inertia are such that there are two large but somewhat unequal principal values, while the third principal value is much smaller. The principal axes are normal to the box surfaces so that all cross products of inertia are zero. The models are based on constant density atmosphere, spherical Earth (inverse square) gravity, and linear attitude control for small motion about an inertial reference. Attitude control can be partially or totally removed so that the body is allowed to tumble under the influence of gravity and aerodynamics. Gravitational, aerodynamic, and control torques affect the rotational dynamics of the rigid body, while gravitational and aerodynamic forces affect the translational dynamics. Both translational and rotational dynamics affect aerodynamic forces and torques, so a feedback loop is apparent. This example falls in the general category of nonlinear, continuous systems.

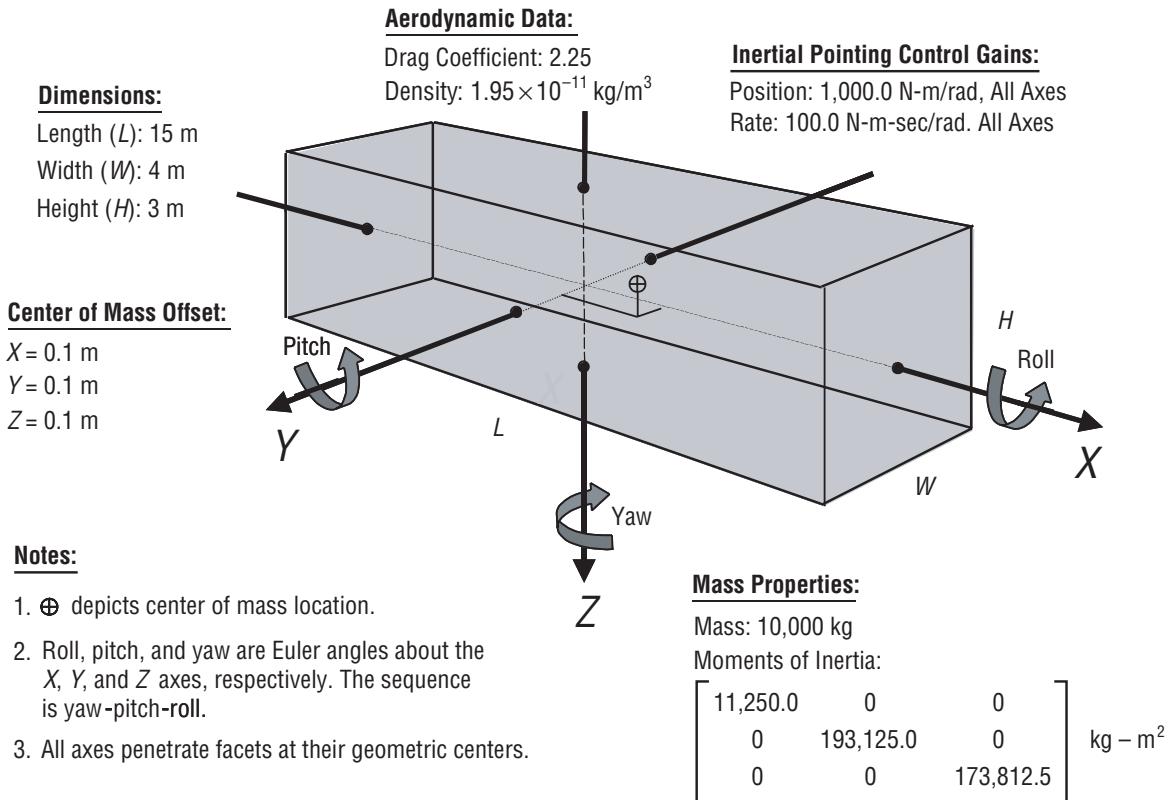


Figure 2. Spacecraft properties and geometric body-fixed coordinate system.

4. MATHEMATICAL MODELS

4.1 Orbit Dynamics

The orbit dynamics is modeled by

$$m_{\text{sat}} \frac{d^2 \vec{R}}{dt^2} = \vec{F}_{\text{grav}} + \vec{F}_{\text{aero}} , \quad (11)$$

where

$$\vec{F}_{\text{grav}} = -\frac{G m_{\text{Earth}} m_{\text{sat}}}{R^3} \vec{R} . \quad (12)$$

G is the universal gravitational constant, m_{Earth} and m_{sat} are masses of the Earth and the satellite. \vec{R} , the orbit radius vector, extends from the center of the Earth to the center of mass of the satellite and is expressed in an Earth-centered reference frame that is defined to be nonrotating and assumed to be nonaccelerating. \vec{F}_{aero} , the aerodynamic force, is derived from an aerodynamics model to be discussed later. This development, as well as the attitude dynamics development, is standard and of the general type found in many dynamics texts such as reference 8.

4.2 Attitude Dynamics

The rotational dynamics is modeled as a rigid body by

$$I \dot{\vec{\omega}} + \vec{\omega} \times I \vec{\omega} = \vec{T}_{\text{control}} + \vec{T}_{\text{gg}} + \vec{T}_{\text{aero}} , \quad (13)$$

where I is the moment of inertia matrix, $\vec{\omega}$ is the angular velocity vector, \vec{T}_{gg} , is gravity gradient torque, \vec{T}_{aero} is aerodynamic torque, and all quantities are expressed in a body-fixed coordinate system whose origin is at the center of mass.

The attitude is defined by a set of Euler angles $[\phi_x, \phi_y, \phi_z]$ that indicate angles traversed by a hypothetical reference frame as it is rotated from coincidence with an inertial (nonrotating) attitude reference frame to coincidence with the body frame using three single-axis rotations in a $z-y-x$ sequence. The instantaneous Euler angle rates are then related to body rates by

$$\begin{bmatrix} \dot{\phi}_x \\ \dot{\phi}_y \\ \dot{\phi}_z \end{bmatrix} = \begin{bmatrix} 1 & \sin \phi_x \tan \phi_y & \cos \phi_x \tan \phi_y \\ 0 & \cos \phi_x & -\sin \phi_x \\ 0 & \sin \phi_x \sec \phi_y & \cos \phi_x \sec \phi_y \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} . \quad (14)$$

Although this particular formulation has a singularity at $\phi_y = \pm\pi/2$, it is valid except at that point, and can serve as the basis for a model if an adequate neighborhood of that point is avoided.

A vector expressed in inertial coordinates is transformed to body coordinates through multiplication of the corresponding column of inertial components by the direction cosine matrix

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_x & \sin\phi_x \\ 0 & -\sin\phi_x & \cos\phi_x \end{bmatrix} \begin{bmatrix} \cos\phi_y & 0 & -\sin\phi_y \\ 0 & 1 & 0 \\ \sin\phi_y & 0 & \cos\phi_y \end{bmatrix} \begin{bmatrix} \cos\phi_z & \sin\phi_z & 0 \\ -\sin\phi_z & \cos\phi_z & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (15)$$

The attitude control system is modeled by

$$-\bar{\mathbf{T}}_{\text{control}} \Leftarrow \begin{bmatrix} a_{0x}\phi_x + a_{1x}\omega_x \\ a_{0y}\phi_y + a_{1y}\omega_y \\ a_{0z}\phi_z + a_{1z}\omega_z \end{bmatrix}, \quad (16)$$

where ϕ , ω , and a are associated with attitude angles, body rates, and control gains, respectively, and the subscripts are self-explanatory.

The gravity gradient torque is modeled by⁹

$$\vec{\mathbf{T}}_{\text{gg}} = \frac{Gm_{\text{Earth}}}{R^3} \hat{\mathbf{r}} \times \mathbf{I} \cdot \hat{\mathbf{r}}, \quad (17)$$

where $\hat{\mathbf{r}}$ is a unit vector corresponding to $\vec{\mathbf{R}}$ expressed in body coordinates.

4.3 Aerodynamics

A box with six rectangular side panels (facets) defines the satellite shape (fig. 2). Aerodynamic drag forces associated with the i th panel are approximated empirically by the formula⁹

$$\bar{\mathbf{F}}_i = \begin{cases} -\frac{1}{2} \rho V^2 C_d A_i (\hat{n}_i \cdot \hat{v}) \hat{v} & \text{if } \hat{n} \cdot \hat{v} \geq 0 \\ 0 & \text{if } \hat{n} \cdot \hat{v} \leq 0 \end{cases}, \quad (18)$$

where ρ is atmospheric density, V is velocity magnitude relative to the atmosphere, C_d is a drag coefficient, A is panel area, \hat{n} is a unit vector normal to the panel and directed outward from the box, and \hat{v} is a unit vector corresponding to \bar{V} . If rotation of the Earth is neglected, no distinction between relative and absolute velocity need be made. The total aerodynamic force is given by

$$\bar{F} = \sum_{i=1}^6 \bar{F}_i . \quad (19)$$

While the vector relationships of equations (18) and (19) are independent of any particular coordinate system, the most convenient choice is a body-fixed “geometric” coordinate system (fig. 2) whose origin and axes are defined from geometry (shape) considerations alone. The aerodynamic force as used in equation (11) is most appropriately expressed in inertial coordinates. Appropriate coordinate transformation depends on the matrix A , defined by equation (15), and on the orientation of the body frame with respect to the geometric frame. As illustrated in figure 3, the axes of the body frame are not rotated with respect to the geometric frame, so the relevant transformation is identity.

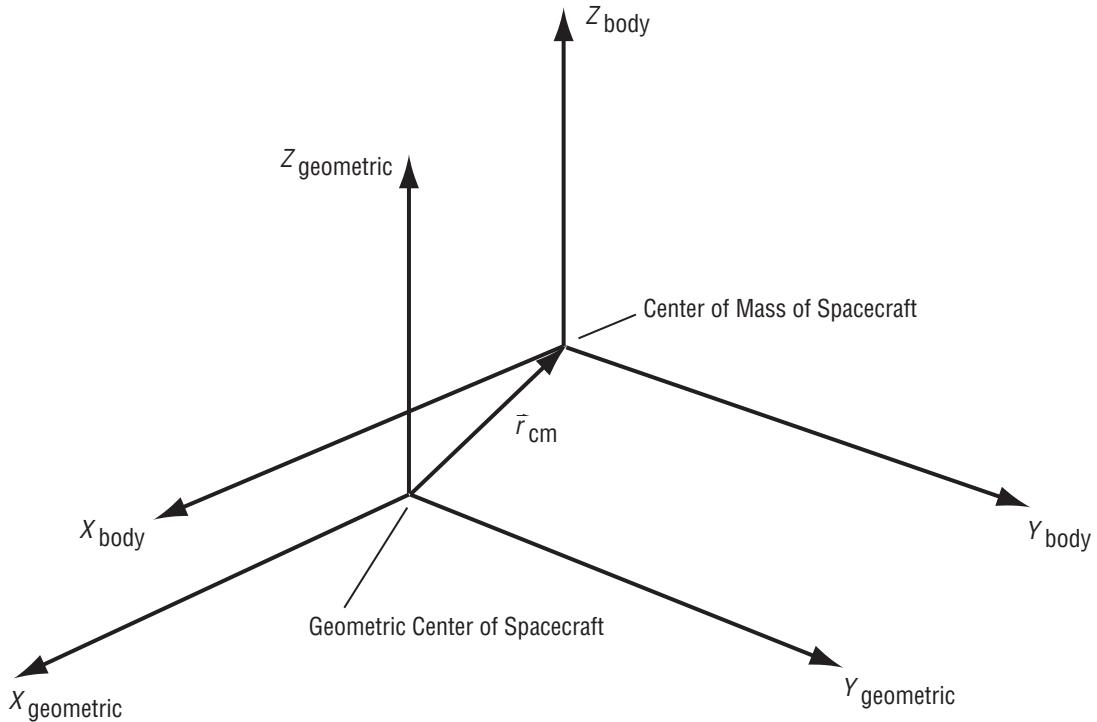


Figure 3. Relationship of body frame to geometric frame.

The total aerodynamic torque about the origin of the geometric frame is calculated from six contributing components and is given by

$$\bar{T} = \sum_{i=1}^6 \bar{r}_{\text{cpi}} \times \bar{F}_i , \quad (20)$$

where \bar{r}_{cpi} is a vector from the origin of the geometric coordinate system to the center of pressure for the i th facet. The total aerodynamic torque about the center of mass of the spacecraft, as used in equation (13), is given by

$$\bar{T}_{\text{cm}} = \bar{T} - (\bar{r}_{\text{cm}} \times \bar{F}) . \quad (21)$$

where \bar{r}_{cm} is a vector from the origin of the geometric coordinate system to the center of mass (fig. 3).

5. SUBSYSTEM SIMULATIONS

Simulations for each of the three subsystems (orbit dynamics, attitude dynamics, and aerodynamics) have been constructed as functionally separate entities, although they are implemented as parts of one single FORTRAN computer program. The attitude and orbit dynamics simulations both have 3 degrees of freedom, and each has its own separate implementation of a Runge-Kutta numerical integration process. In both cases, the three second-order differential equations have been rearranged to six first-order equations for interface with the integration algorithm. The aerodynamics simulation has no integration process because the model is purely logical-algebraic. All three simulations are “stand-alone” because each, in principle, is capable of producing output signals (functions of time) from input signals and initial conditions. Initial conditions, of course, do not apply to the logical-algebraic aerodynamic simulation. Input and output signals are represented by a sequence of linearly connected points equally spaced in time. Additional points are obtained through interpolation by each simulation if needed.

The orbit dynamics simulation input signal is aerodynamic force, and output signals are orbit radius and velocity vectors. The model, defined by equations (11) and (12), considers a point mass, inverse square gravity, and an external force (aerodynamics) that is an arbitrary but known function of time. All orbit dynamics simulation variables are referenced to an Earth-centered inertial coordinate system.

The attitude dynamics model input signals are the orbit radius vector expressed in inertial coordinates, and aerodynamic torque expressed in body coordinates. Output signals are body rates and attitude angles. The model, defined by equations (13)–(17), considers rotational motion of a rigid body, gravity gradient torque, attitude control system torque, and an external torque (aerodynamics) that is an arbitrary but known function of time.

The aerodynamics simulation input signals are orbit radius vector in inertial coordinates and orbit velocity vector in body coordinates. Output signals are aerodynamic force and torque expressed in geometric coordinates. The model is logical-algebraic and is defined by equations (18)–(21).

6. SYSTEM SIMULATIONS

Three system simulations have been constructed. The first simulation uses a centralized coupling implementation of recursive feedback, while the second simulation uses a distributed coupling implementation. The third simulation uses a conventional first-order, single-equation set approach and is the standard against which the recursive feedback approaches are compared.

By design, the function of a coupler is to perform coordinate transformations or other types of data conversions needed to make consistent, correct connections among the subsystem simulations. A single coupler can perform all coupling functions from one central location, or a set of couplers that perform less extensive functions can be distributed throughout the system. Coupling signals, like input and output signals, are represented by a sequence of points equally spaced in time.

Figures 4 and 5 illustrate the centrally coupled system simulation. The subsimulations represent independent processes that can, in principle, be executed in parallel over the timespan of each convergence interval. Because the primary purpose is to investigate functionality rather than efficiency, no attempt has been made to use more than one processor in the single-computer FORTRAN implementation. Therefore, the parallel computation advantage is lost, and quantitative comparison of efficiency with the other simulations is not possible.

Figure 6 illustrates the distributively coupled system simulation. Before applying the idea of recursive feedback, the subsystems have been arranged so that, to the extent possible, feedback is minimized. Because there is an information loop that ultimately must be closed, it is not possible to completely eliminate feedback. The subsimulations are identical to those of the centrally coupled system, but the system-level numerical process is substantially affected because, among other things, the number of scalar signal paths that cross the recursive transfer point has changed. This simulation was developed in part because the centrally coupled simulation, developed first, is noticeably slower than the standard method benchmark. More importantly, it serves to confirm functionality of the different coupling architectures.

The conventional simulation is implemented from the combined set of equations that define the subsystem models. They have been collected and rearranged, by hand, to a single set of 12 first-order differential equations, an arrangement sometimes called a “state variables” approach to system formulation. In that context, the state vector is defined as

$$X = [R \mid V \mid \phi \mid \omega]^T , \quad (22)$$

where R , V , ϕ , and ω are three-element row vectors representing orbit radius, velocity, Euler angles, and body rates. The system differential equations can now be represented as a first-order set by

$$\dot{X} = f(X, t) , \quad (23)$$

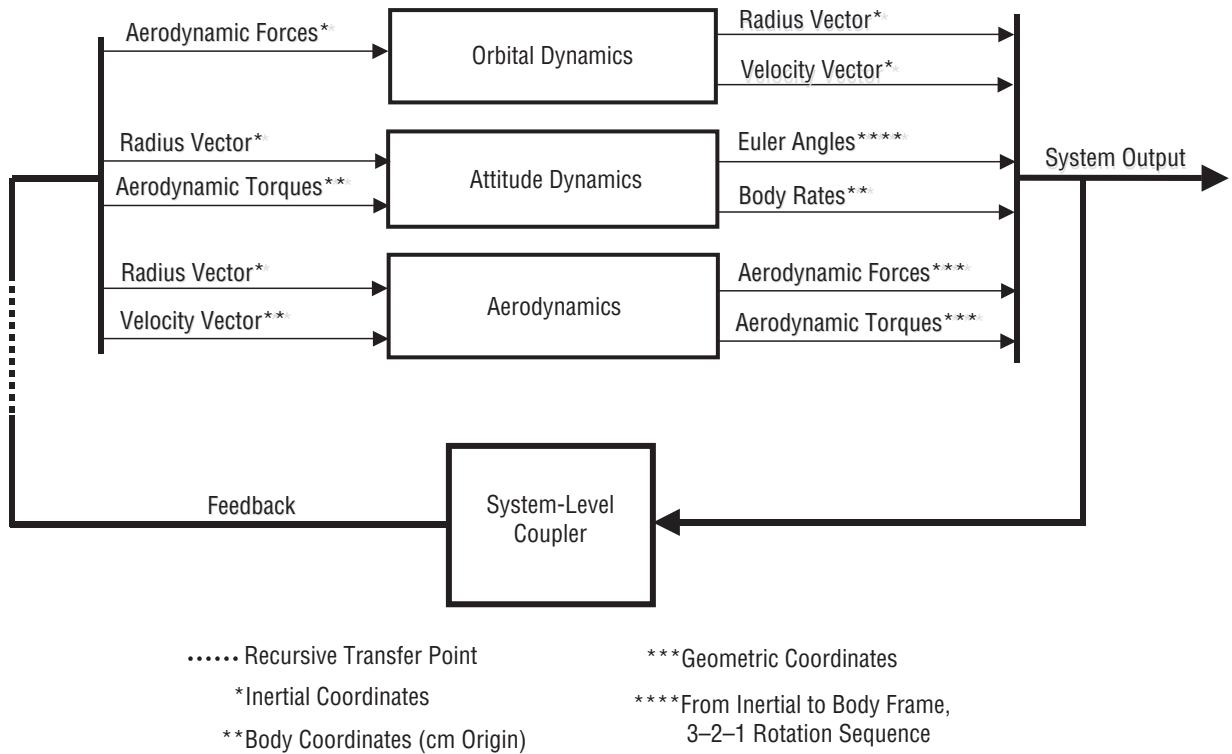


Figure 4. Centrally coupled simulation diagram.

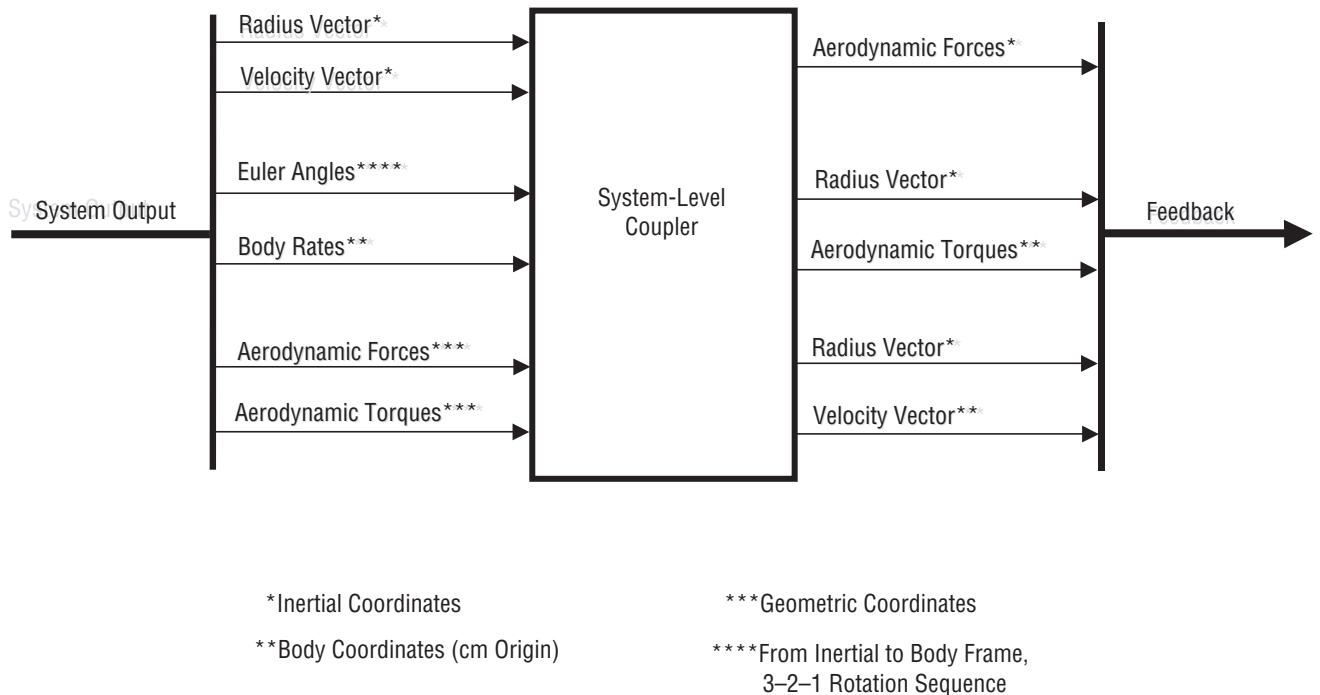


Figure 5. Function of system-level coupler.

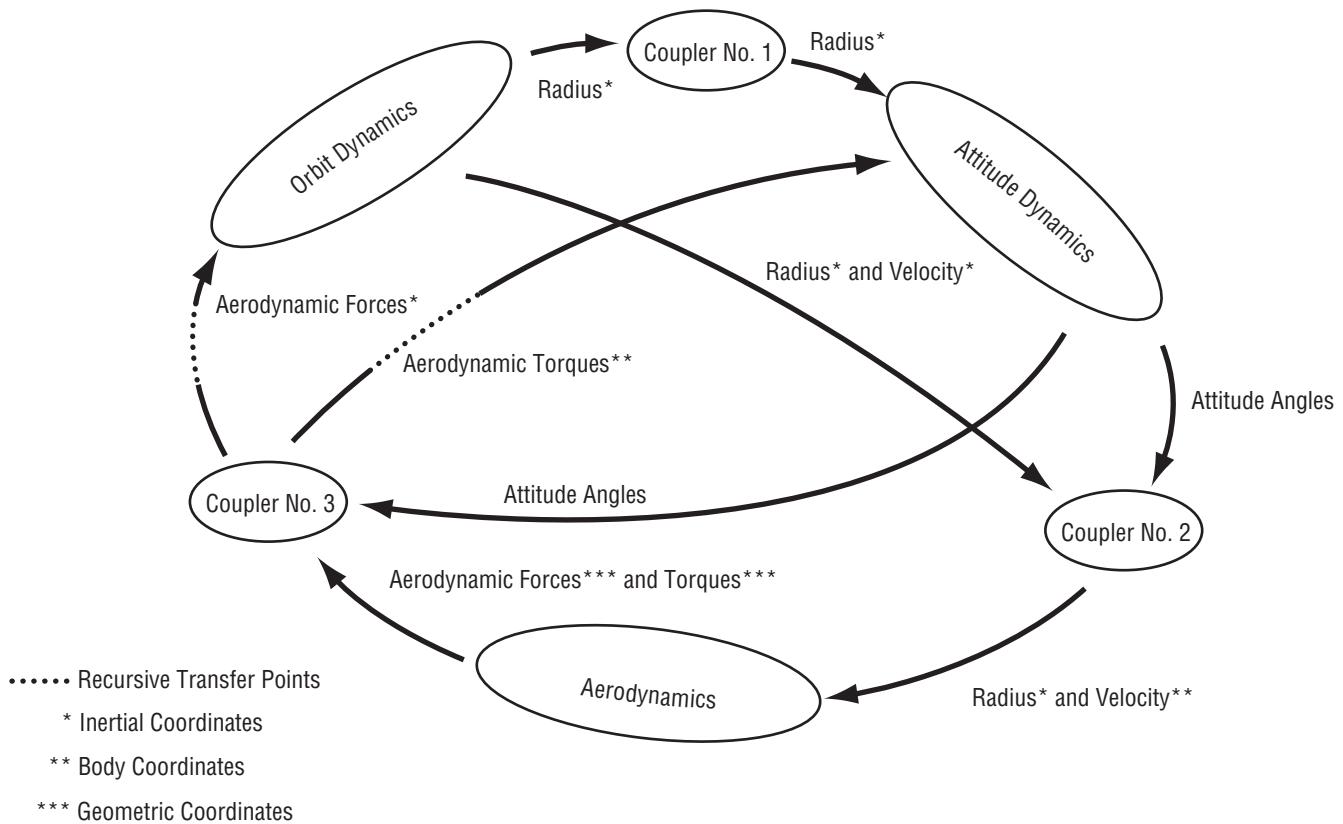


Figure 6. Distributively coupled simulation diagram.

where time has been explicitly included only for completeness of the form. In the current simulation, time is only implicitly involved, because there are no system forces or torques that are explicit functions of time. A fourth-order Runge-Kutta numerical integration algorithm specifically designed to solve systems in the form of equation (13) is employed to propagate the system responses.

All system and subsystem simulations were specially developed for this project. They are written in FORTRAN, all data were generated on the TD55/eds2 UNIX platform at Marshall Space Flight Center (MSFC). Plots were originally produced with an in-house program (developed by Warren Adams, TD55, MSFC) on the same machine. Plot images were migrated to a Windows desktop PC for publication. A full complement of plots is included in appendix A, and listings of the simulation programs are included in appendix B.

7. NUMERICAL RESULTS

7.1 Run Case Definitions

All data cases simulate motion of the same spacecraft starting from the same orbit and attitude related initial conditions. The orbit is initialized so that in the absence of aerodynamic drag it would be circular at an altitude of 300 km and an inclination of 30 deg. The initialization point is in the X-Z plane of the coordinate system shown in figure 7. Spacecraft mass properties, dimensions, and control gains are given in figure 2. Initial attitude angles and body rates are zero. Other data constants and physical constants are identified in the computer listings. Case variations are developed from three attitude control system configurations and three simulation methods resulting in a nine-element run matrix. The attitude control configurations are listed as follows:

- (1) Full inertial control
- (2) Partial inertial (two-axis) control
- (3) No control (tumble).

Case numbers correspond directly to the control system configuration list entry numbers, and method variations are associated with each case. The simulation methods are:

- (1) Centrally coupled recursive feedback (RF-C)
- (2) Distributively coupled recursive feedback (RF-D)
- (3) Conventional (unified) Runge-Kutta (RK-U).

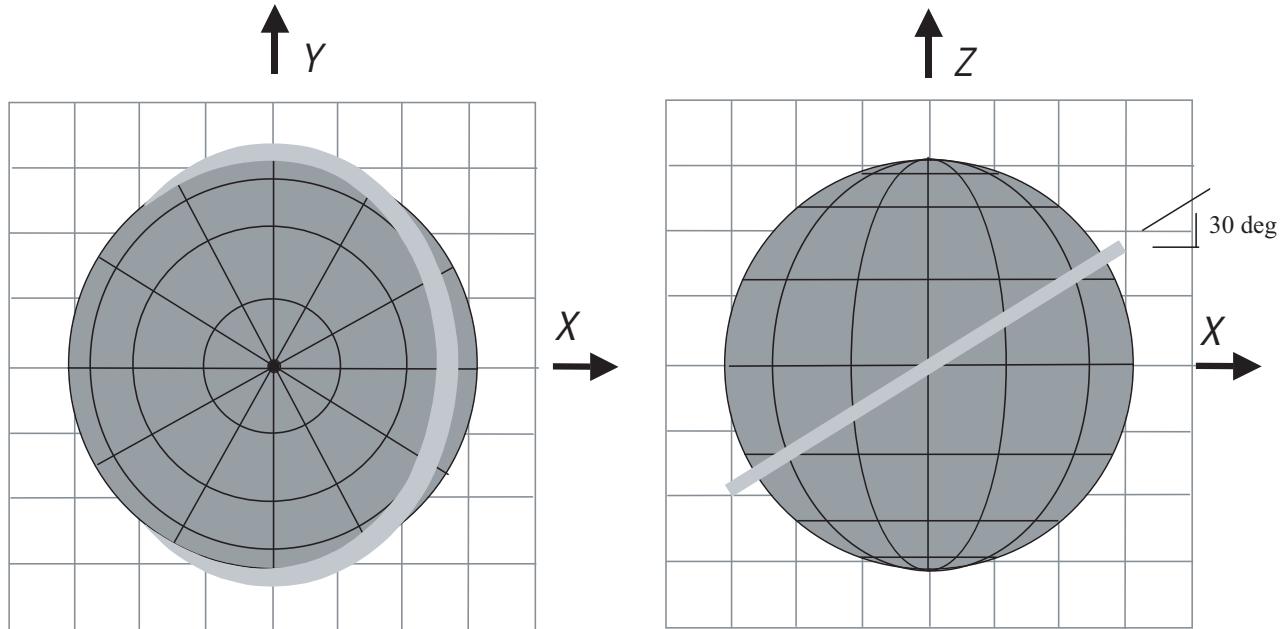


Figure 7. Inertial coordinate system/initial orbit plane geometry.

Numerical algorithm parameter settings were the same for all cases. Both recursive methods used five sample points per convergence interval (including both end points), and the interval length was 0.4 sec. Relative (normalized) and absolute convergence tolerances were both 10^{-12} for the RF-C and 10^{-14} for the RF-D. The orbit and attitude subsimulations both used a fourth-order Runge-Kutta algorithm with a step size of 0.1 sec, while the aerodynamics subsimulation computed aerodynamic effects at a 0.1-sec sample interval. The conventional simulation used the same Runge-Kutta algorithm with a step size of 0.1 sec.

Results for all possible (nine) combinations of control system configuration and solution methods were generated. For case 1, in the absence of any well-defined requirement, a set of control system gains was chosen that produce reasonable (small angle) attitude dynamics responses. For case 2, the Z-axis control gains were set to zero; and for case 3, all control gains were set to zero. Simulation responses have been generated for 60,000 sec of problem time using all solution approaches for all cases.

7.2 Response Comparisons

The responses for case 1 (full control) are identical over the full 60,000-sec duration of the simulation runs for all solution approaches (RF-C, RF-D, and RK-U). Appendix A.1 contains 108 plots that compare RF-D results to conventional RK-U results for case 1. Radius, velocity, attitude angles, body rates, aerodynamic forces, aerodynamic torques, and altitude are shown as functions of time. The general format of the appendix presentation is to show each response over time intervals of 0–20,000, 20,000–40,000, and 40,000–60,000 sec, and additionally, to show some selected plots that cover smaller time intervals of particular interest.

The responses for case 2 (two-axis control) are, like case 1, identical over the full 60,000-sec duration of the simulation runs for all solution approaches. Appendix A.2 contains 92 plots that compare RF-D results to conventional RK-U results for case 2. As in the appendix presentation for case 1, radius, velocity, attitude angles, body rates, aerodynamic forces, aerodynamic torques, and altitude are shown as functions of time, and the presentation format is also similar to that used for case 1.

For case 3 (tumble), the RF-C, RF-D, and RK-U simulation responses all agree for approximately 45,000 sec of the duration; after that, the RF-C and RF-D responses begin to diverge from the RK-U while they continue to agree with each other for the full 60,000 sec. Figures 8–16 present plots selected specifically to illustrate the early agreement and subsequent deviation of responses generated recursively from those generated conventionally. Attitude angles are shown in figures 8–10, body rates are shown in figures 11–13, and altitude is shown in figures 14–16. Only the 40,000- to 60,000-sec time period is covered because no earlier deviation is apparent. Appendix A.3 contains 60 plots that compare results of the RF-D approach to the conventional RK-U approach for this case.

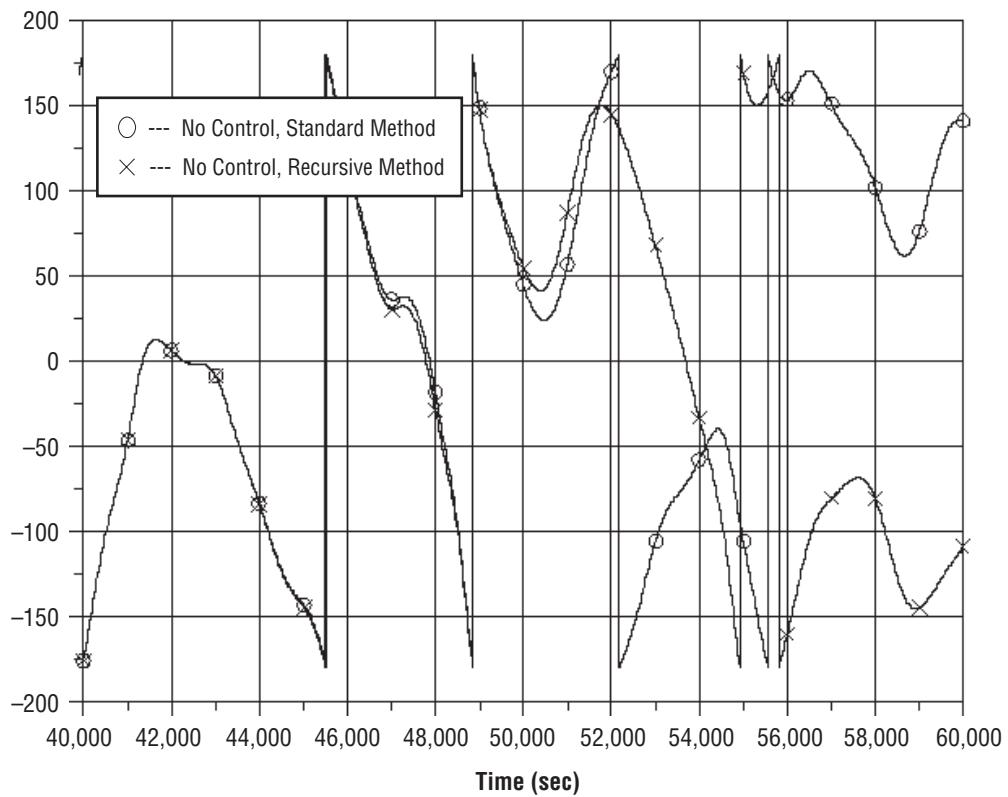


Figure 8. Case 3, Euler angle, X axis (deg).

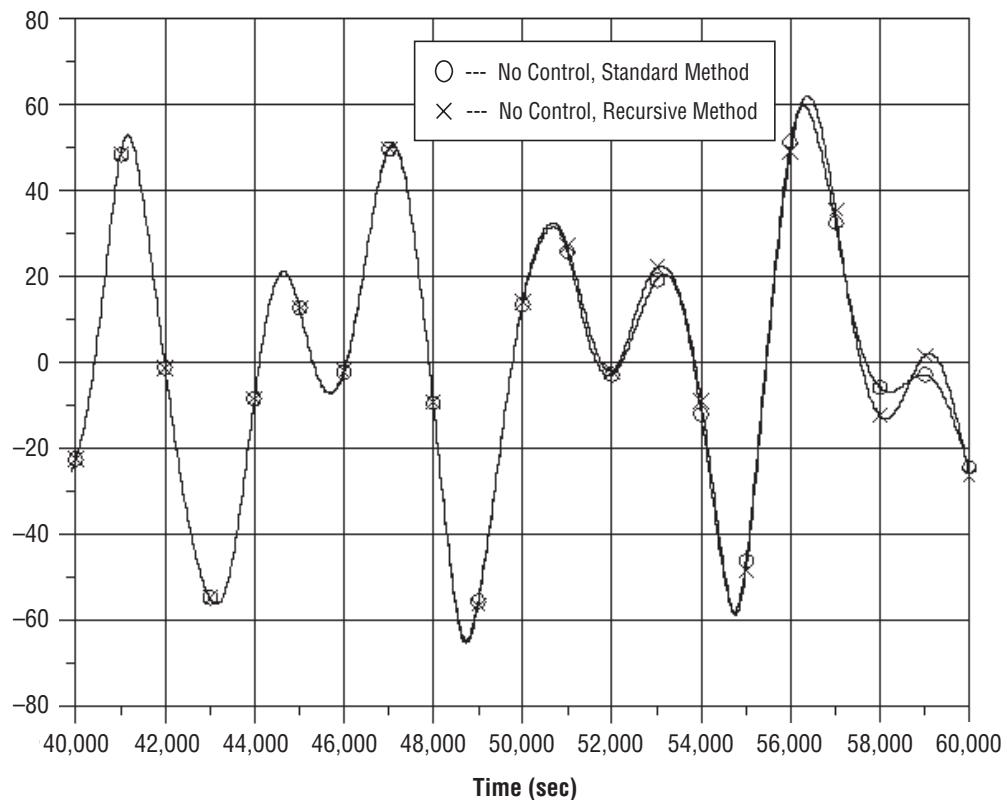


Figure 9. Case 3, Euler angle, Y axis (deg).

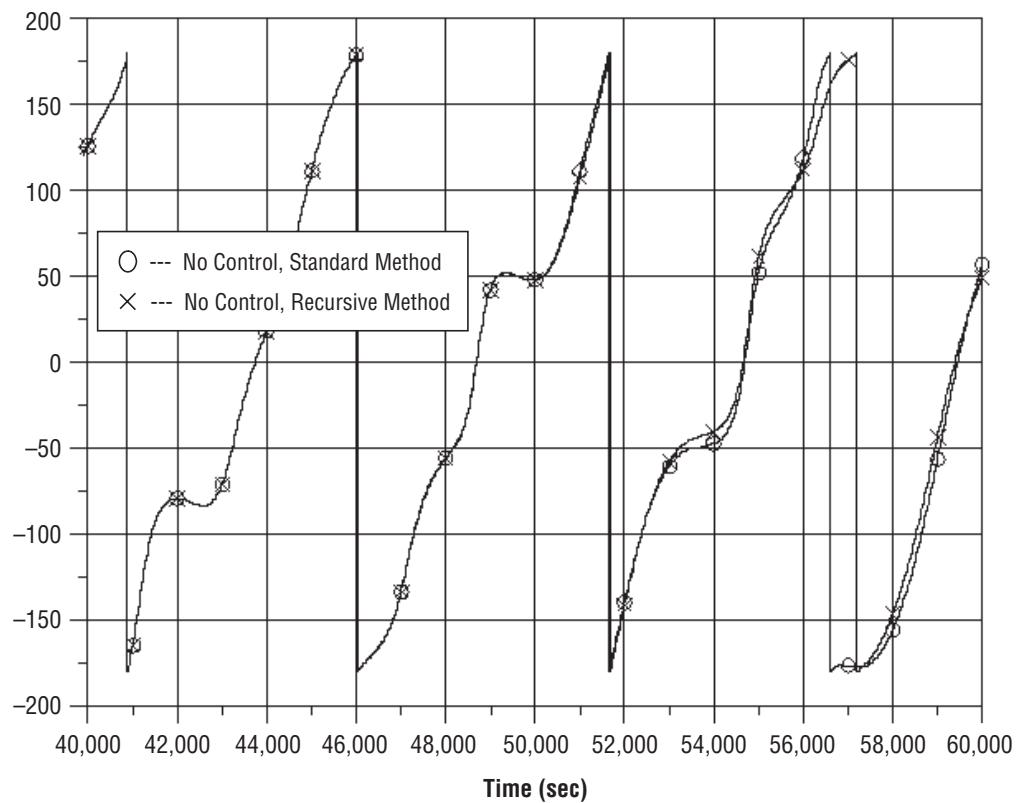


Figure 10. Case 3, Euler angle, Z axis (deg).

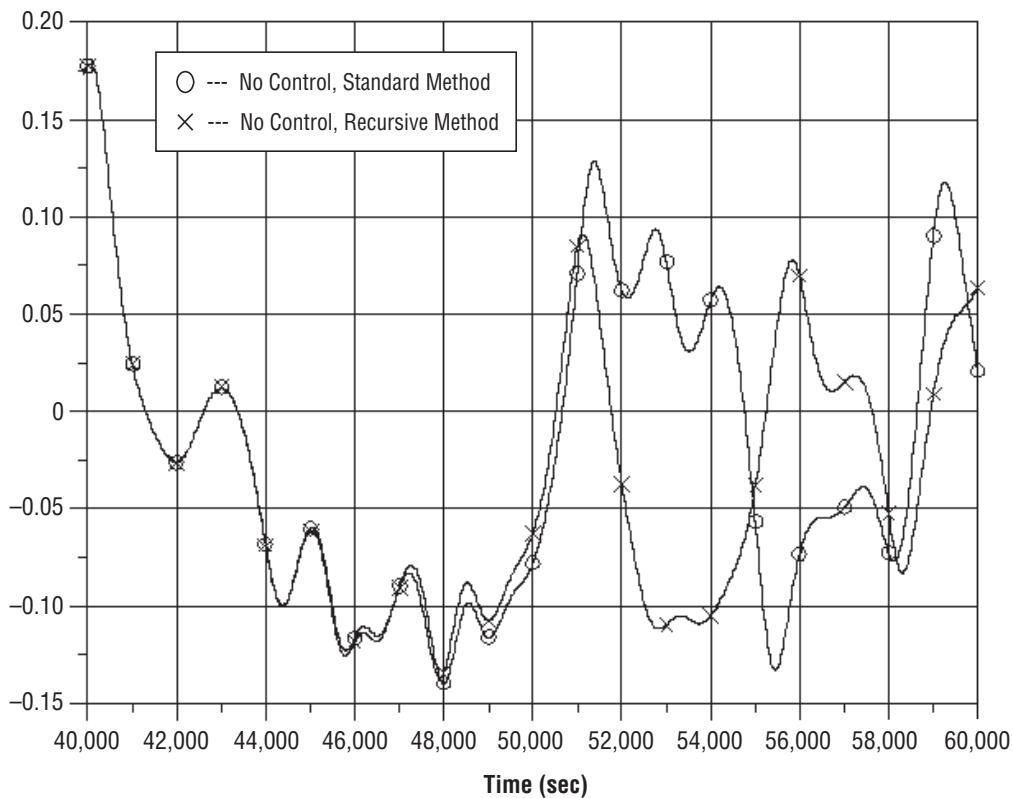


Figure 11. Case 3, body rate, X axis (deg).

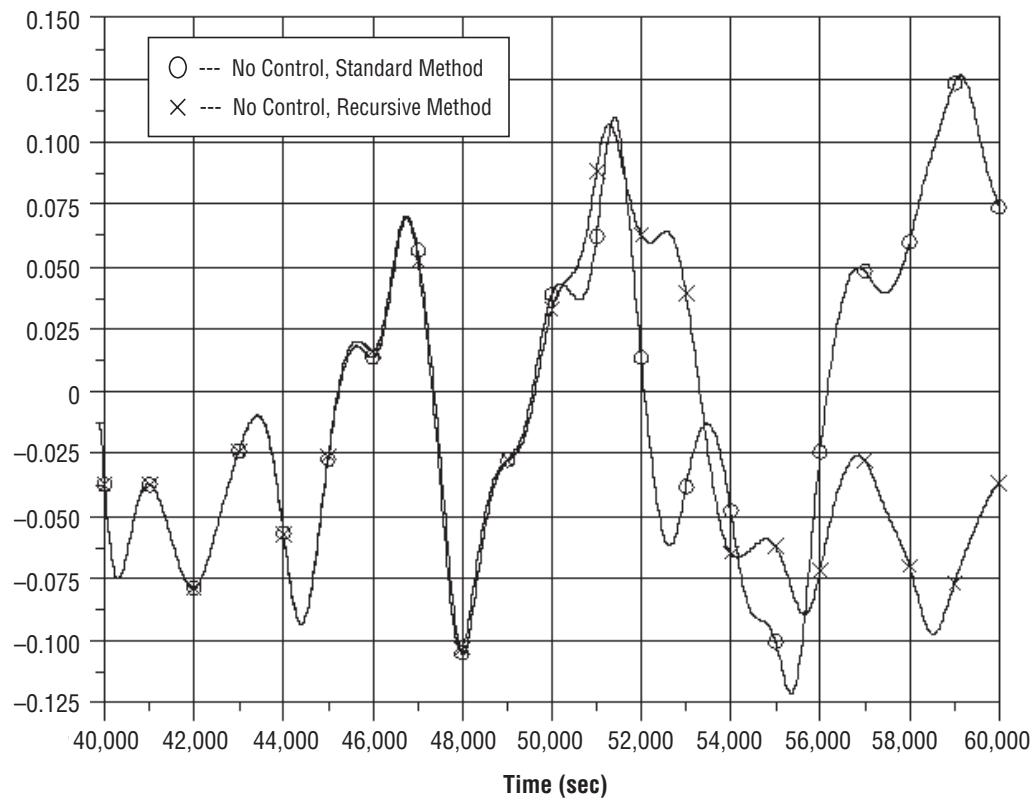


Figure 12. Case 3, body rate, Y axis (deg).

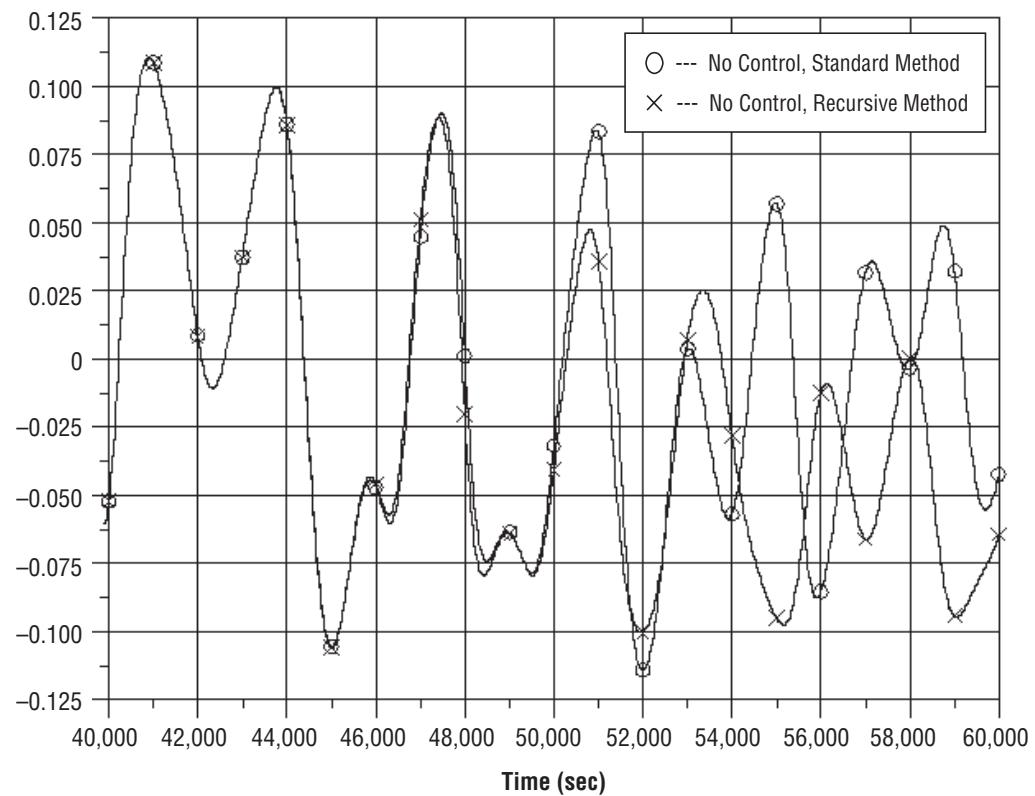


Figure 13. Case 3, body rate, Z axis (deg).

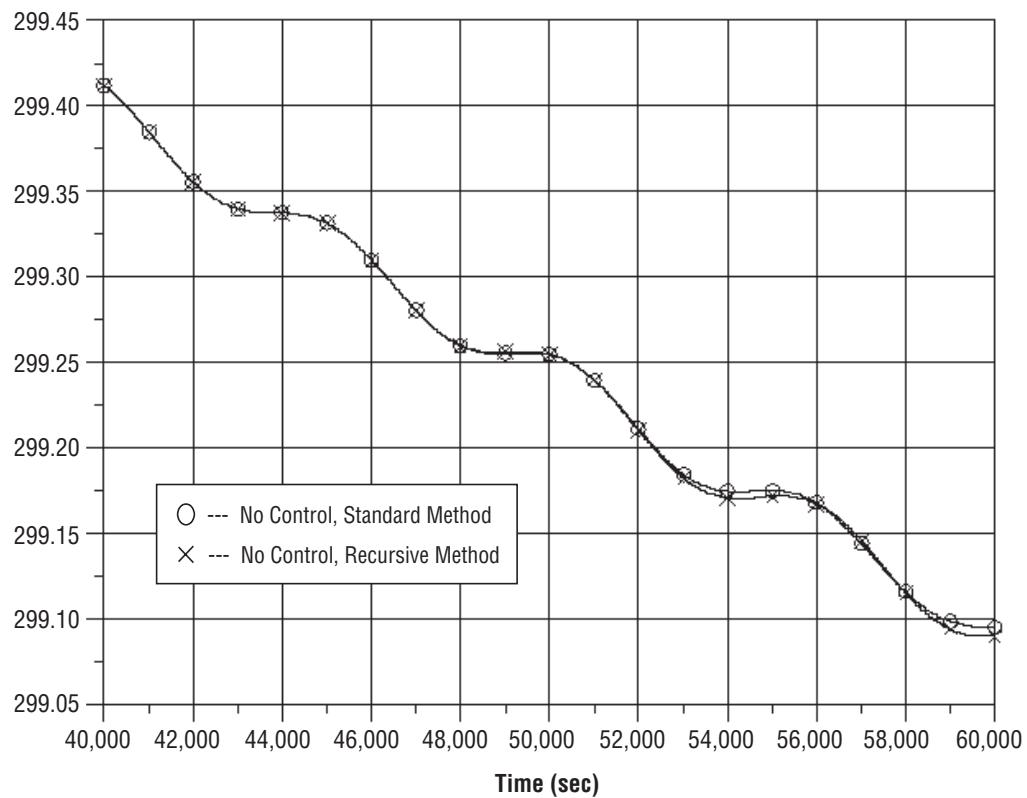


Figure 14. Case 3, altitude, first time segment (km).

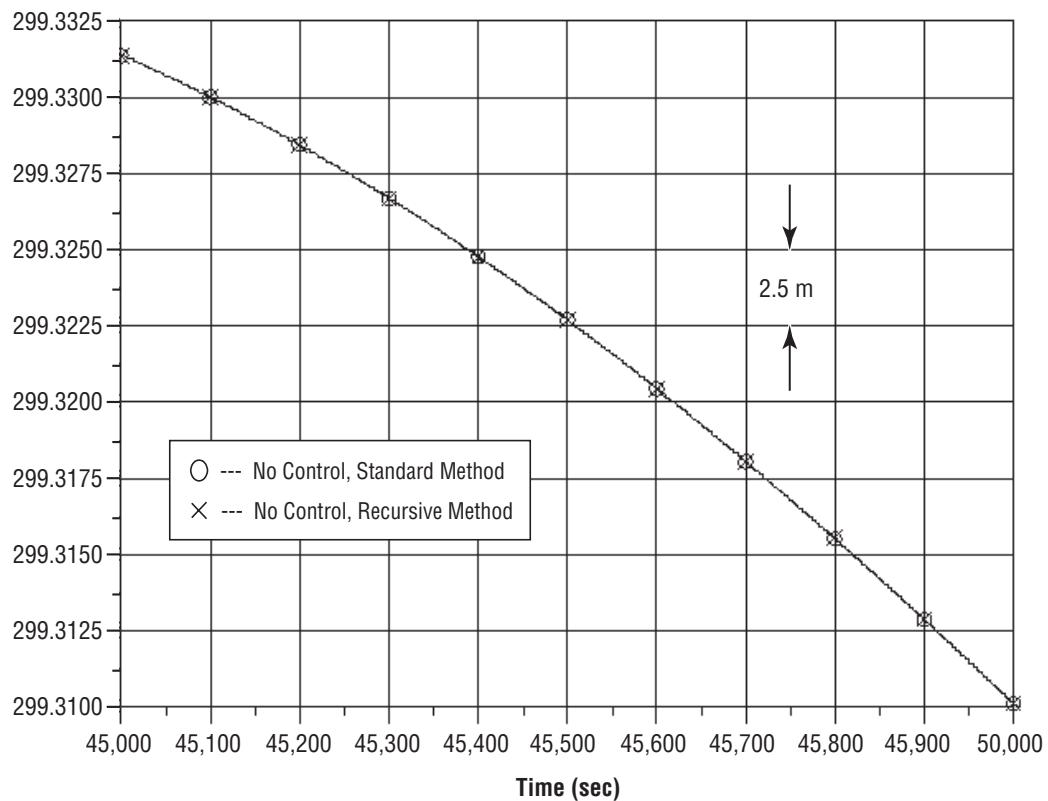


Figure 15. Case 3, altitude, second time segment (km).

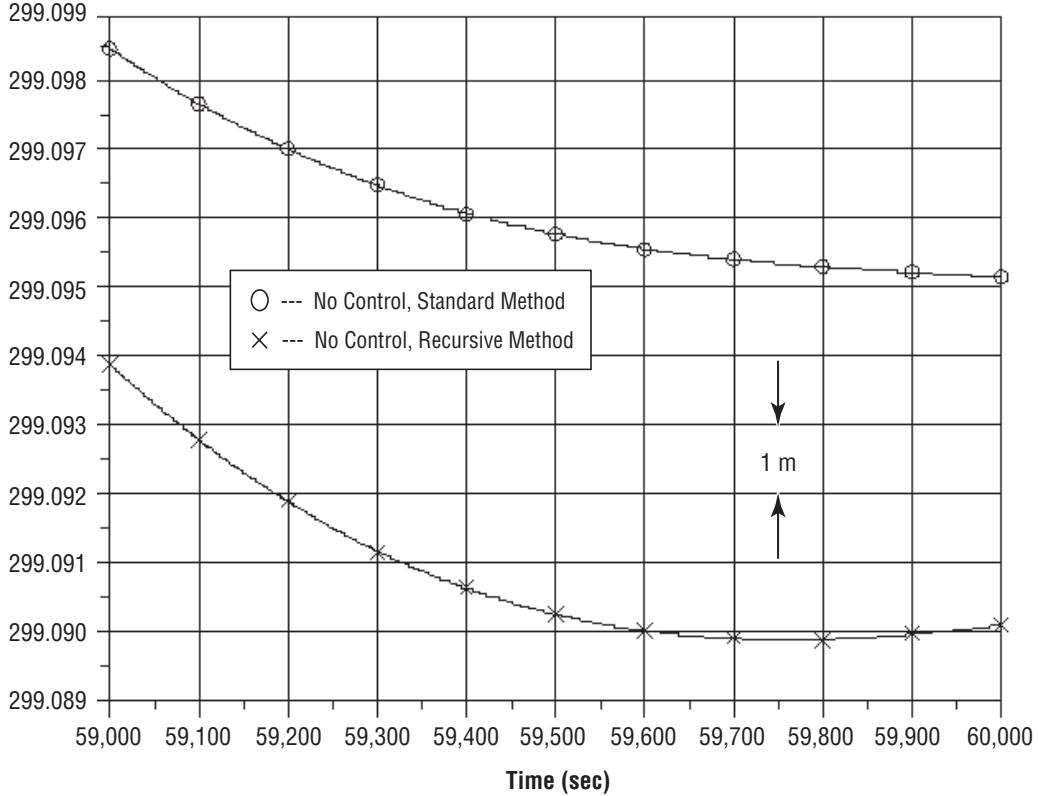


Figure 16. Case 3, altitude, third time segment (km).

Altitude results for all control system configurations and two solution methods (RF-D and RK-U) are superposed in figures 17–19. The main point of illustration is the variation in orbit decay rate as a function of attitude control. Figure 17 shows the full 60,000-sec run duration, while figures 18 and 19 show shorter time segments taken near the end of the run. Figures 17–19 illustrate again the agreement of recursive and conventional solution methods for cases 1 and 2, while showing the method-induced deviation apparent in case 3, the no control (tumble) case.

Appendix A.4 contains 42 plots that compare results from the two recursive approaches (RF-C and RF-D) to each other for all three cases. These plots are presented primarily for completeness since, for all practical purposes, the responses are identical for both methods over the full 60,000-sec duration of the simulation runs.

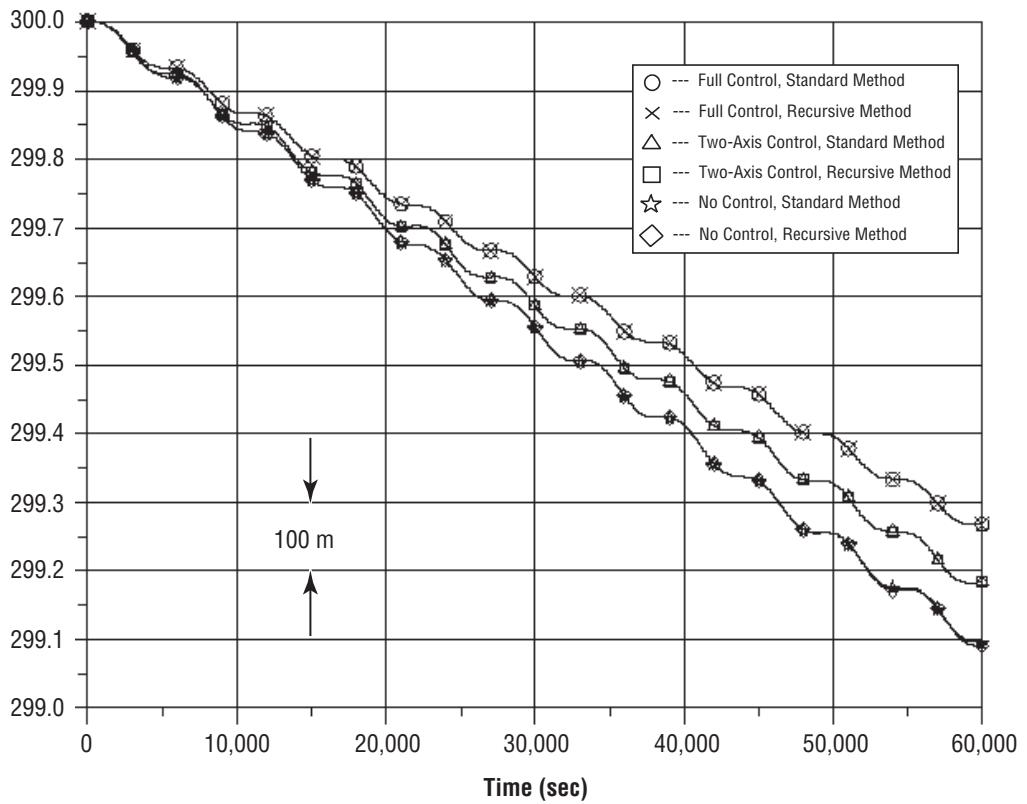


Figure 17. Altitude, case-method array, full run duration (km).

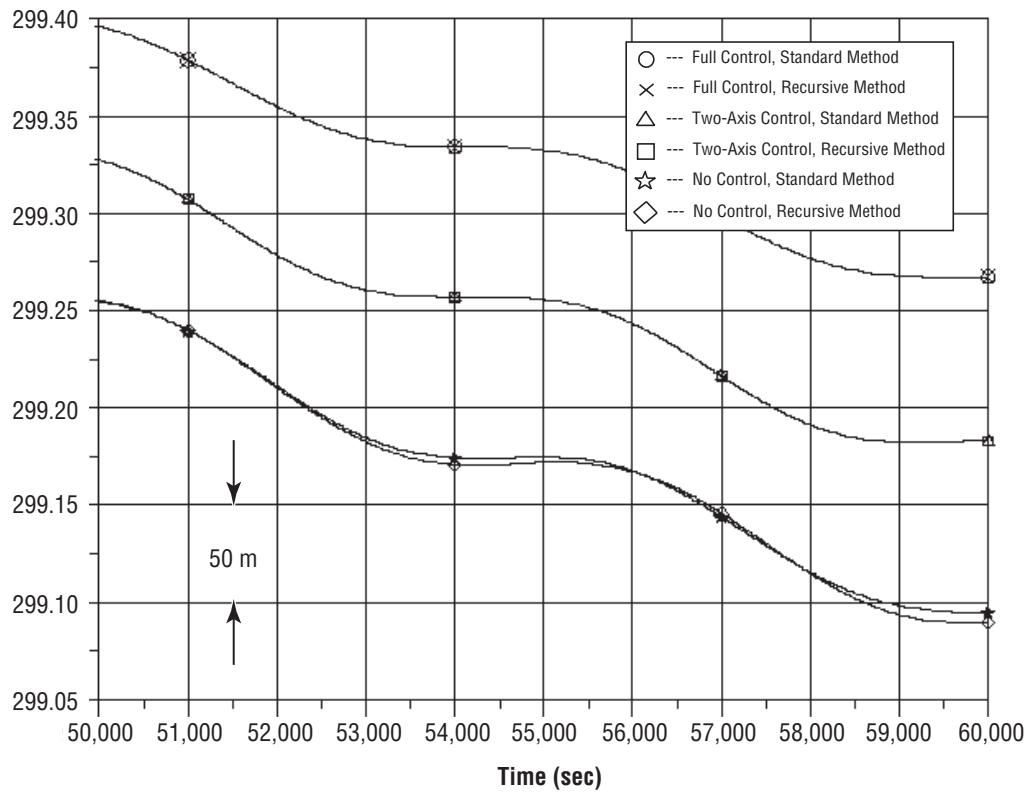


Figure 18. Altitude, case-method array, first time segment (km).

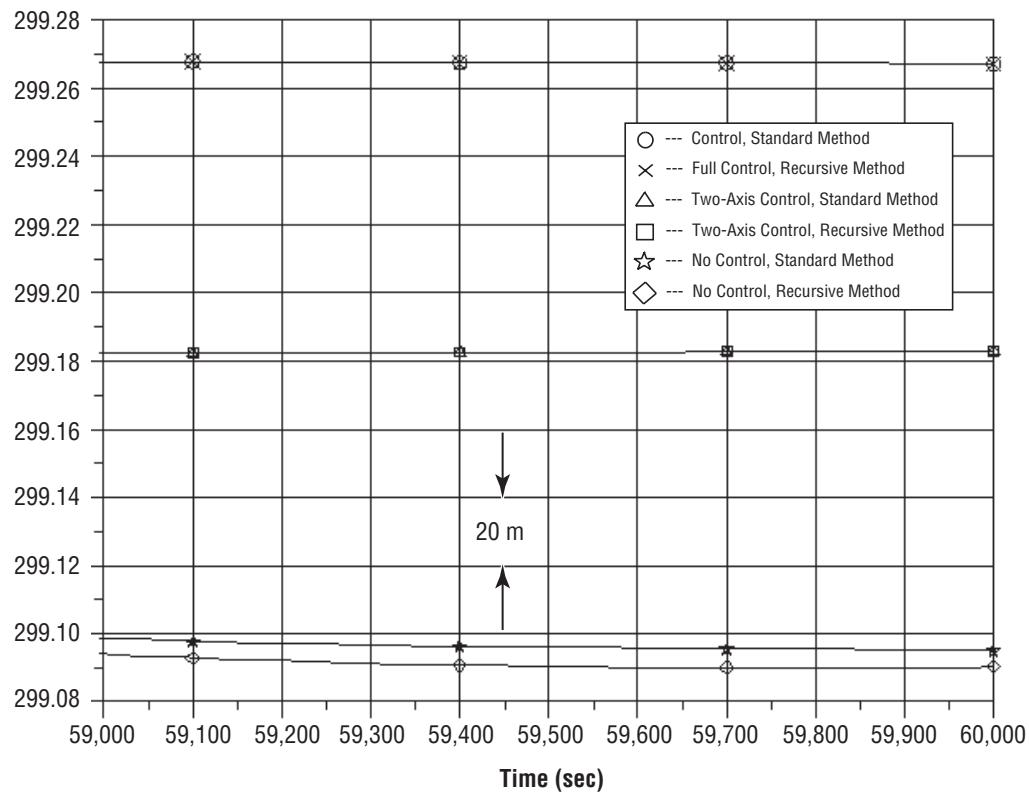


Figure 19. Altitude, case-method array, second time segment (km).

8. PERFORMANCE

The numerical results presented in figures 8–19 and in appendix A very clearly confirm that coupling of discipline-oriented subsimulations by recursive feedback can be used successfully as an architecture and solution approach to simulation of the example system. In cases 1 and 2, 60,000 sec of run time is achieved with no significant deviation of results. In case 3, deviation begins at approximately 45,000 sec. The earlier deviation of case 3 is attributed primarily to linear interpolation of aerodynamic forces and torques over the subsimulation integration step size in the recursive (RF–C and RF–D) implementations, while aerodynamic computations are made at points internal to the integration interval in the unified (RK–U) implementation. Improvement could likely be found through a more sophisticated interpolation method, a shorter convergence interval, or an increased number of points per convergence interval for signal definition. It must be remembered, however, that eventual disagreement of dynamic simulations driven by different numerical processes is always expected.

The results also confirm that the location and distribution of coupling functions, while partially defined by the fundamental nature of the subsystem interactions, can be part of the simulation design and can have a significant affect on performance. For the current example, the convergence interval length (0.4 sec) and the number of points per convergence interval (five) are the same for both recursive approaches, and the accompanying subsimulations as well as the conventional simulation all use the same integration step size (0.1 sec). Convergence error tolerances were in the range of 10^{-12} to 10^{-14} for the recursive methods. Typical reported processor utilization times for the RF–C, and RF–D simulations are approximately 1,200 and 800 sec, respectively, while the RK–U time is approximately 200 sec.

As mentioned earlier, the potential parallel computation advantage of the RF–C simulation is lost because of the particular single-computer FORTRAN implementation. Given that fact, the RF–D (fig. 6) simulation is understandably faster than the RF–C (fig. 4) because the amount of data being processed and checked to confirm or deny convergence at the recursive transfer point is significantly less. The speed of the RF–C simulation would be increased if the subsimulations were actually run in parallel. The possibility for running subsimulations in parallel does not exist for the RF–D simulation because the process design requires them to be run sequentially. In either recursive method, output from the last stage of the process is used to compare against the previous stage for confirmation of convergence. In the conventional method, no active control of error is currently present, and inclusion of any such mechanism would require additional computation; conversely, predetermination of the number of recursion stages and elimination of the convergence check could significantly reduce the computation load for either recursive method. For the current example, the maximum number of recursion stages is typically three, one of which could arguably be said to serve no purpose other than to confirm satisfaction of error tolerances. As the number of recursion stages increases, the extra time required for the confirmation stage becomes less significant in a relative sense, but elimination of the checking process entirely would save time at every stage.

None of the current simulations were constructed with emphasis on efficiency. After correctness, programmer convenience and listing readability were the next higher priority considerations. It is believed that many strengths of recursively coupled subsimulations architecture lie in areas other than computational efficiency and those strengths may sometimes be well afforded at considerable expense.

9. IMMEDIATE FUTURE DEVELOPMENT

More analytical investigation and development are appropriate. Analyses that investigate error sources, error propagation, computational efficiency, and stability are needed. Analyses that address applicability to stochastic systems and discrete data systems are also needed. Further reduction of conservatism of the criterion for convergence derived for systems with algebraic loops (ref. 6) should be explored. The basic concept should be extended to include automatic control of convergence interval duration and other process parameters to improve performance. Implications of central versus distributed or mixed coupling architectures should be considered from viewpoints of algorithm performance and computational resource utilization.

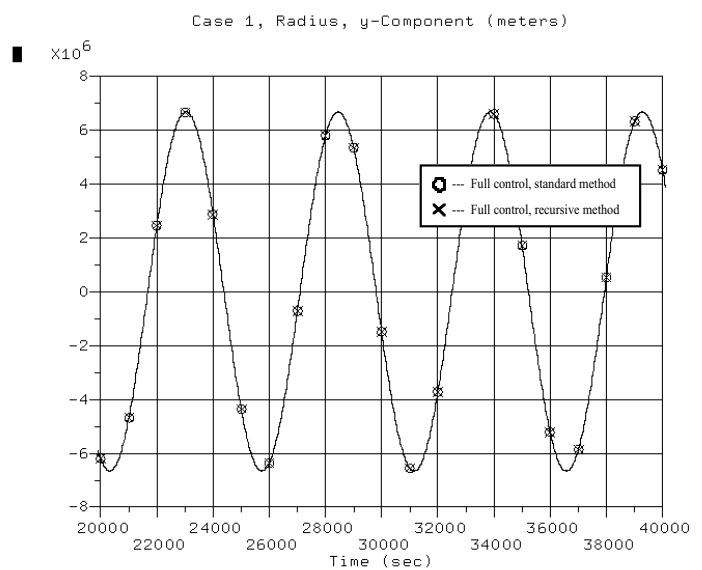
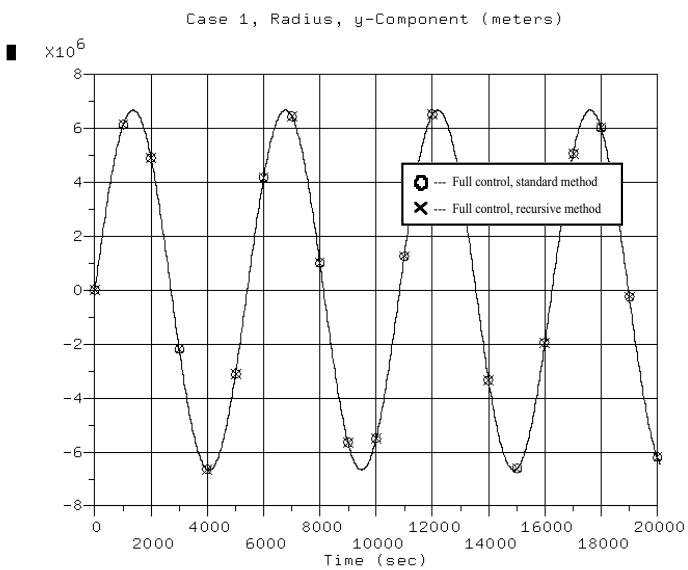
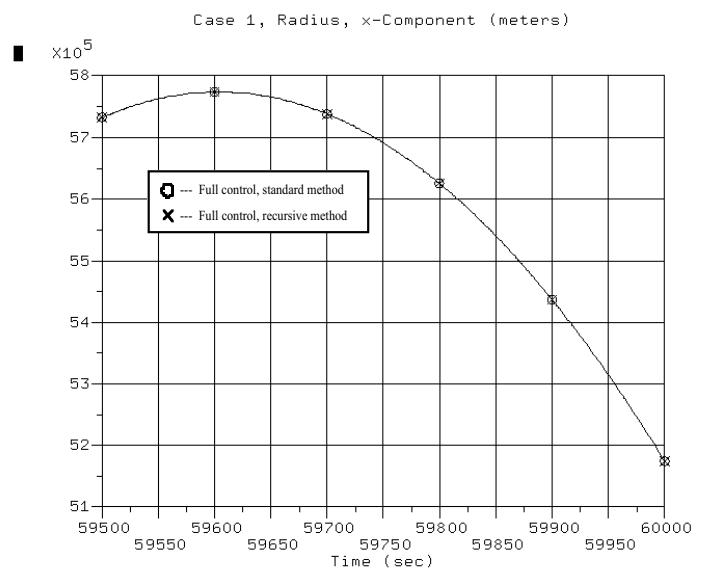
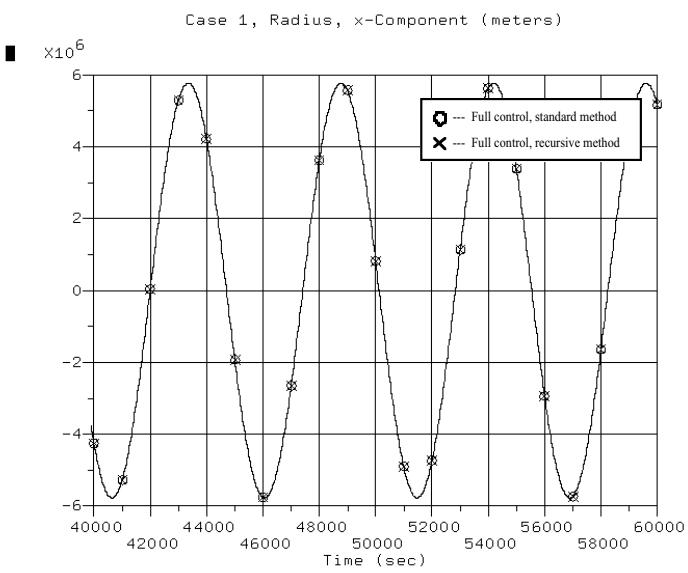
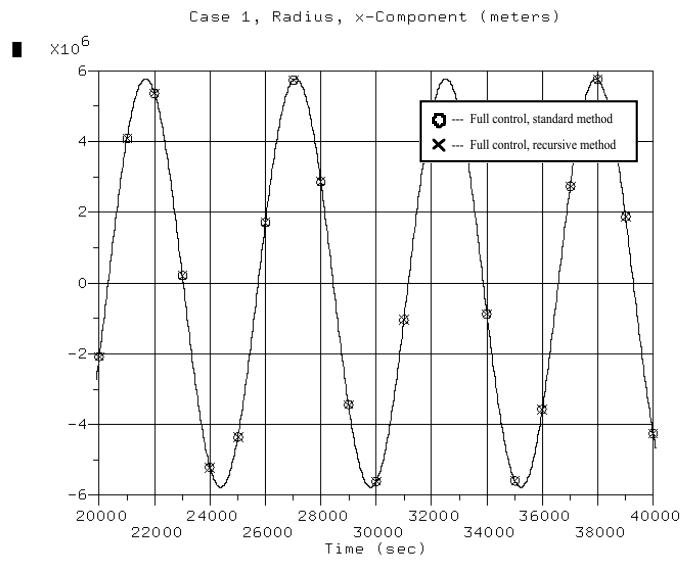
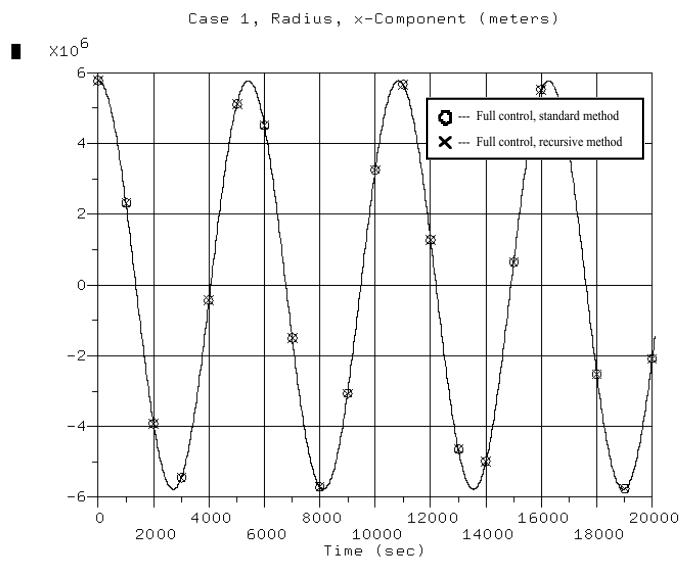
To facilitate a multicomputer implementation, a high-level standard format for digital definition of a vector analog signal segment (function of time) needs to be chosen or created so that signal segments can be transmitted and received throughout the system. In addition, system-level command and control functions must be defined so that standard subsimulation and coupler interfaces can be developed. A single-computer multiprocessor development that takes advantage of parallel computation could advance development of requirements for a multicomputer implementation as well as facilitate a better assessment of performance. Construction of an example involving more than one computer will serve to identify a detailed set of platform- and network-specific requirements in data systems and software terms; it will also serve to explicitly demonstrate technical integrity for a multiplatform application.

10. CONCLUSIONS AND RECOMMENDATIONS

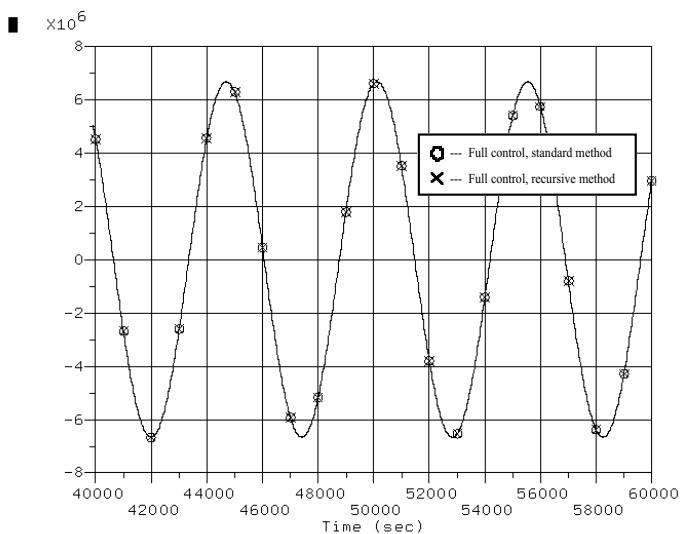
An example nonlinear, continuous simulation of satellite motion has been successfully constructed using two variations of recursive feedback to couple three separate subsimulations of orbit dynamics, attitude dynamics and control, and aerodynamics. Results have been verified by direct comparison with a conventionally constructed simulation of the same system. Each subsimulation deals with one engineering discipline, and appropriate interactions are implemented recursively at the system level. The subsimulations can, in principle, be run separately on remote and dissimilar platforms while coupling is implemented by way of network. Depending on the system-level design configuration, it is possible to run the subsimulations in parallel. The coupling signals represent physically identifiable variables associated with physical interfaces of the subsystems. The example employs three subsimulations; the method framework accommodates n . Clearly, further investigation of recursively coupled subsimulations as a systems simulation architecture is warranted. It is recommended that the present effort be expanded to include all of the future development areas mentioned above, and that longer range plans should include a pilot multisubsystem, multidiscipline, multiplatform development of a complexity level that demonstrates effectiveness and efficiency through collaboration.

APPENDIX A—RESPONSE PLOTS

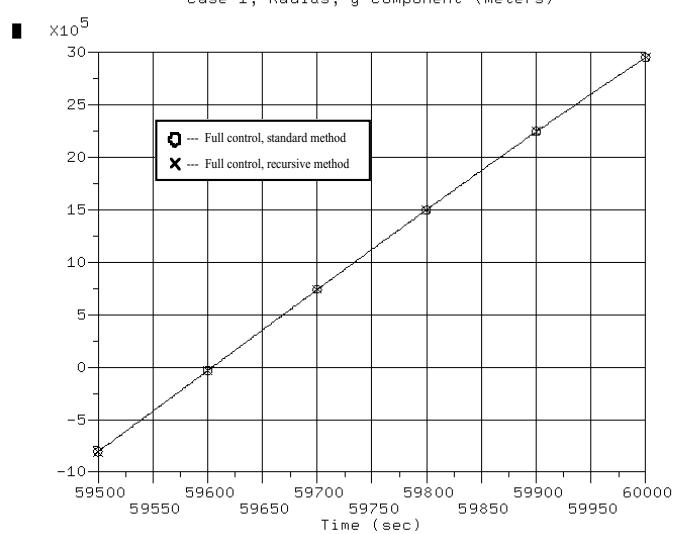
A.1 Response Plots for Case 1 (Full Three-Axis Inertial Attitude Control)



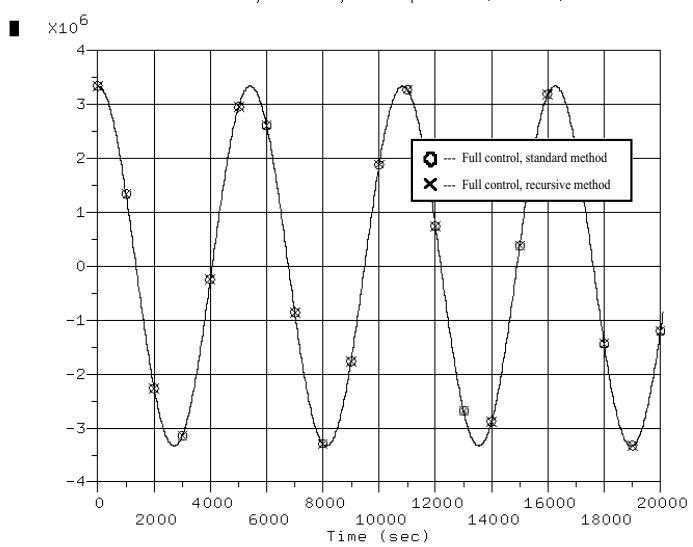
Case 1, Radius, y-Component (meters)



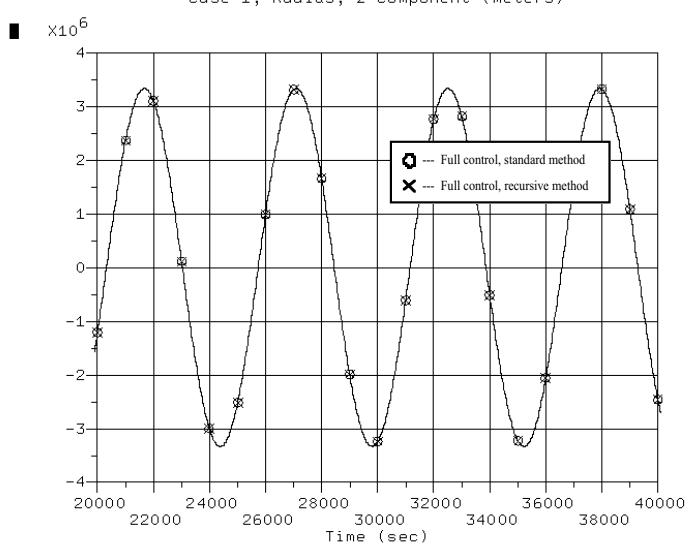
Case 1, Radius, y-Component (meters)



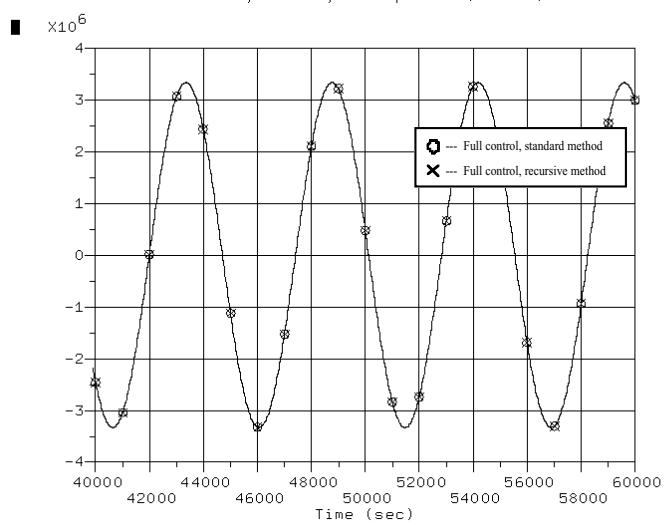
Case 1, Radius, z-Component (meters)



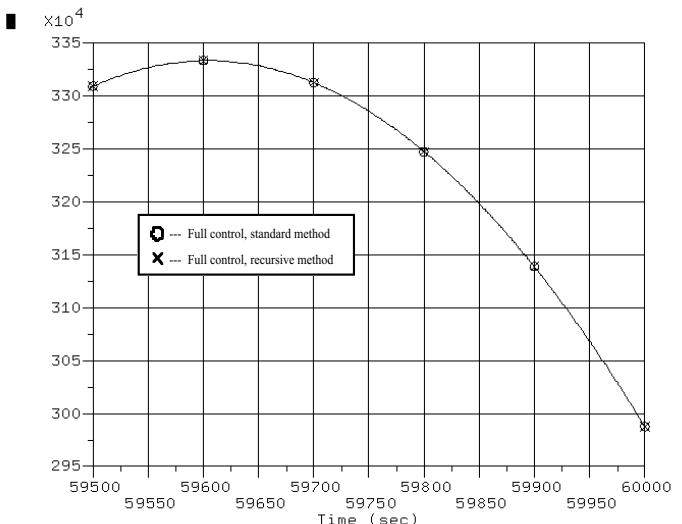
Case 1, Radius, z-Component (meters)



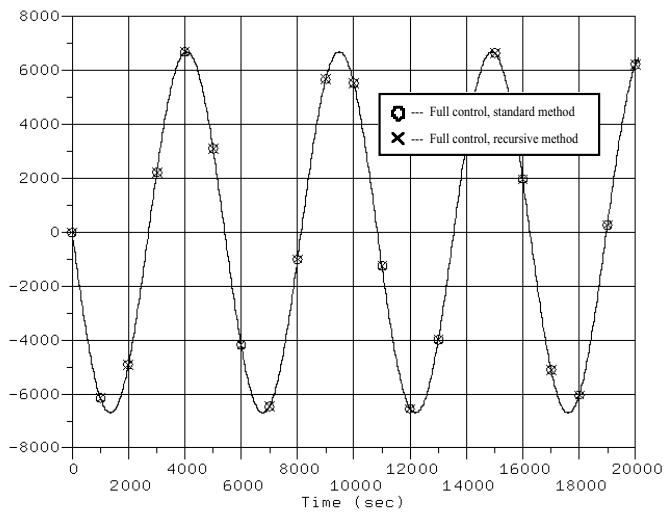
Case 1, Radius, z-Component (meters)



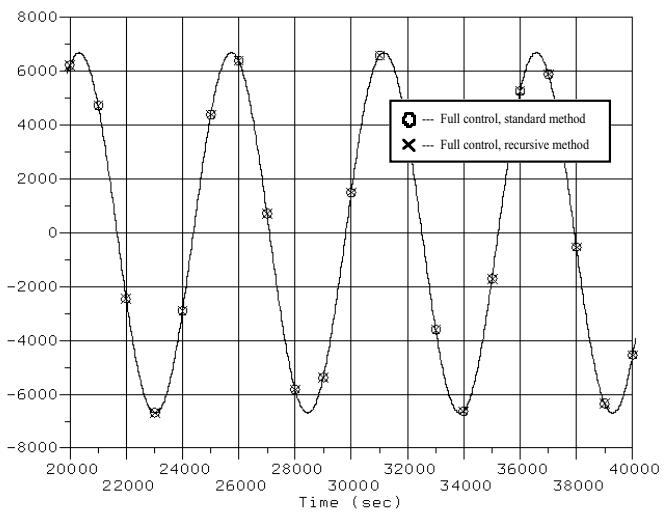
Case 1, Radius, z-Component (meters)



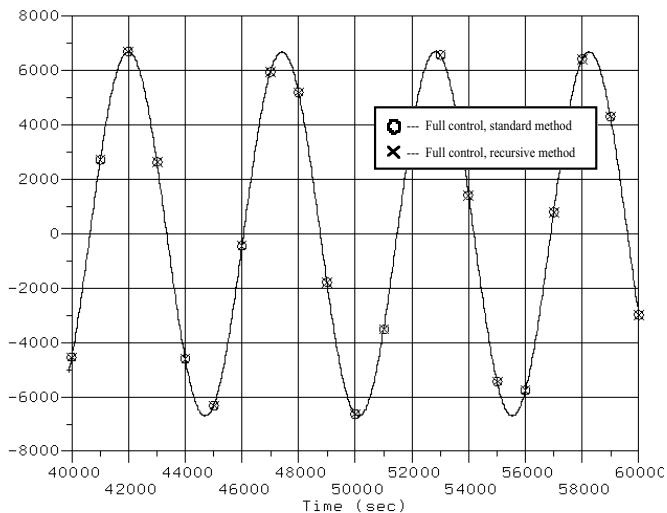
Case 1, Velocity, x-Component (meters/sec)



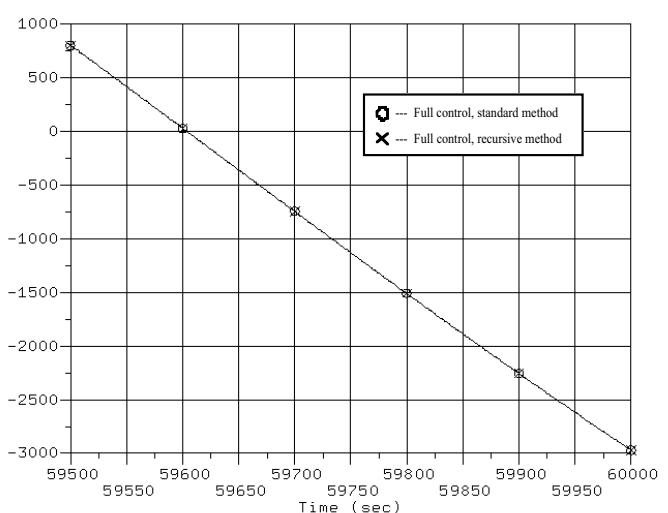
Case 1, Velocity, x-Component (meters/sec)



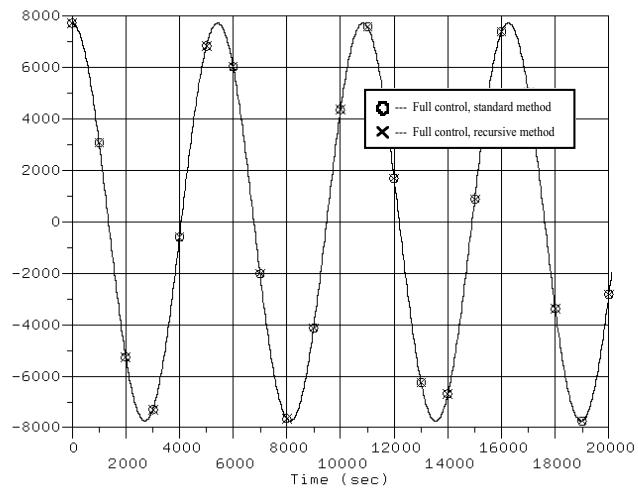
Case 1, Velocity, x-Component (meters/sec)



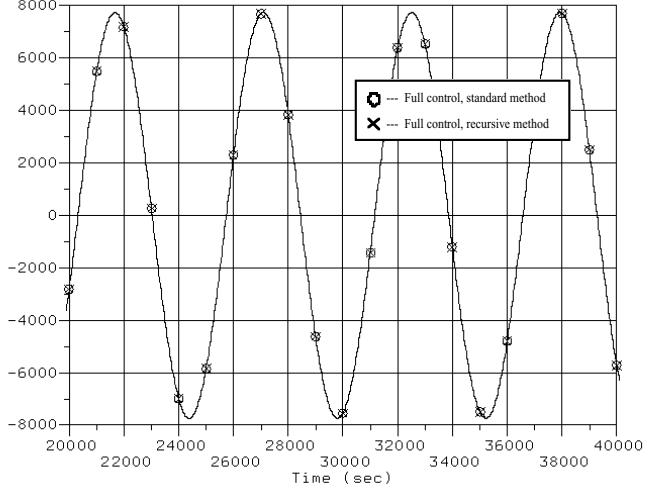
Case 1, Velocity, x-Component (meters/sec)



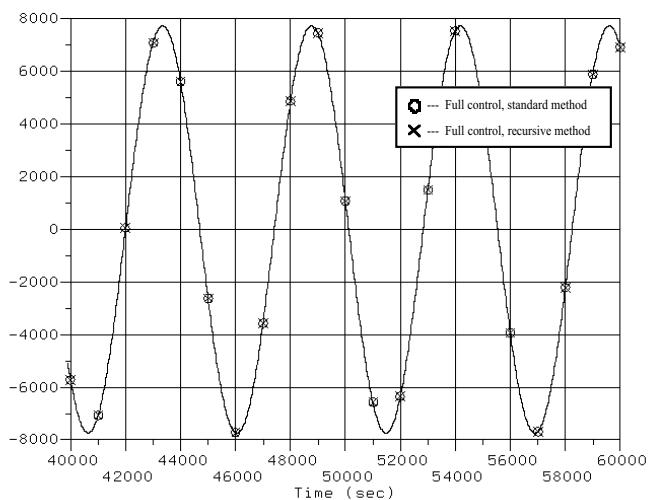
Case 1, Velocity, y-Component (meters/sec)



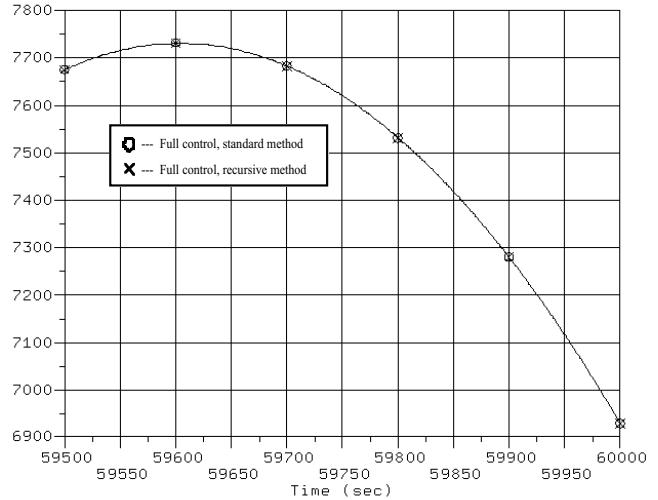
Case 1, Velocity, y-Component (meters/sec)



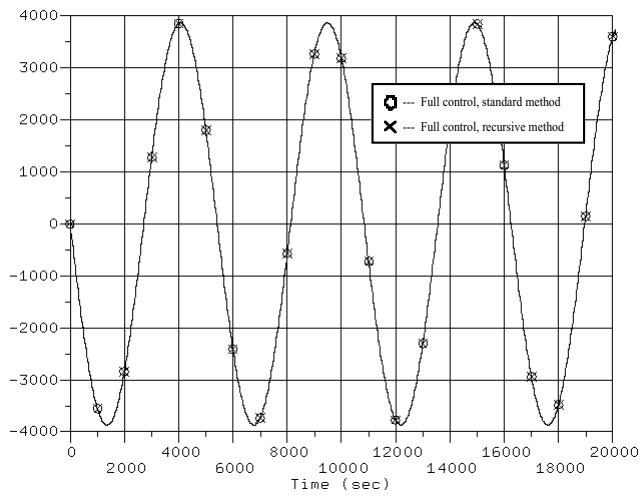
Case 1, Velocity, y-Component (meters/sec)



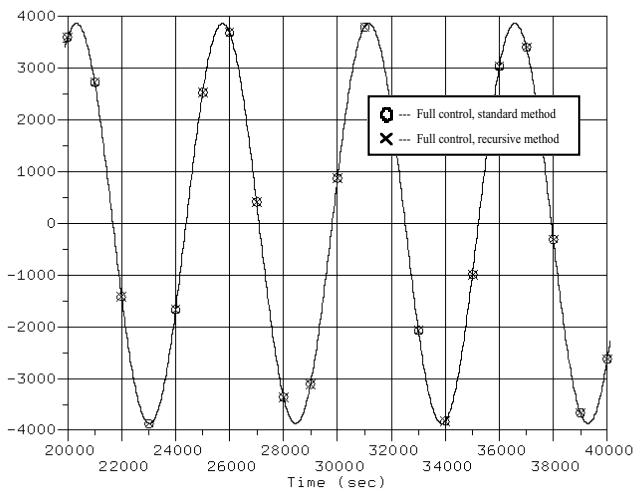
Case 1, Velocity, y-Component (meters/sec)



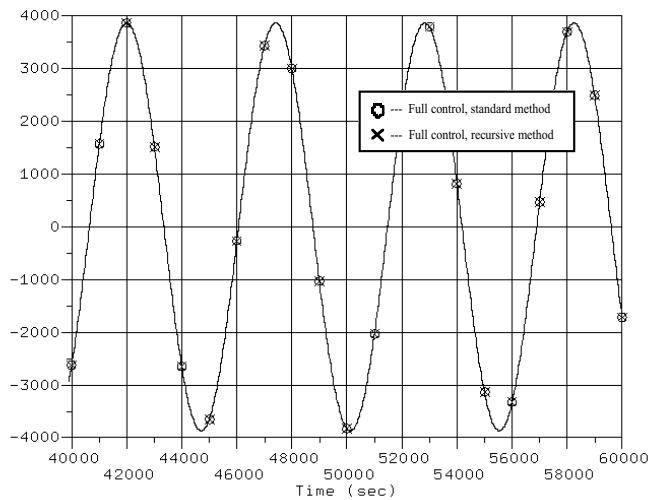
Case 1, Velocity, z-Component (meters/sec)



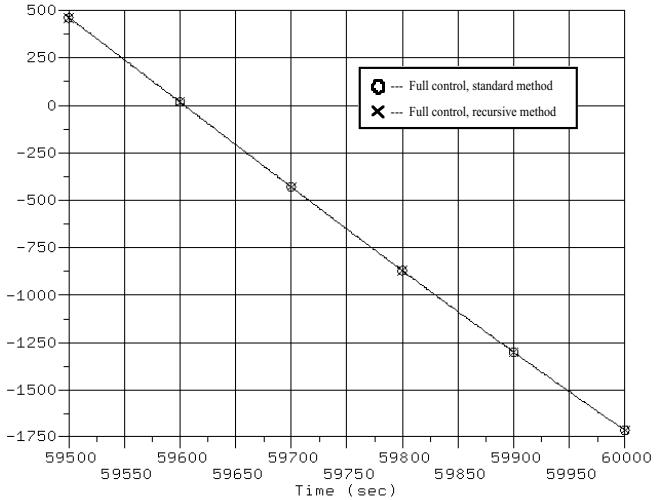
Case 1, Velocity, z-Component (meters/sec)



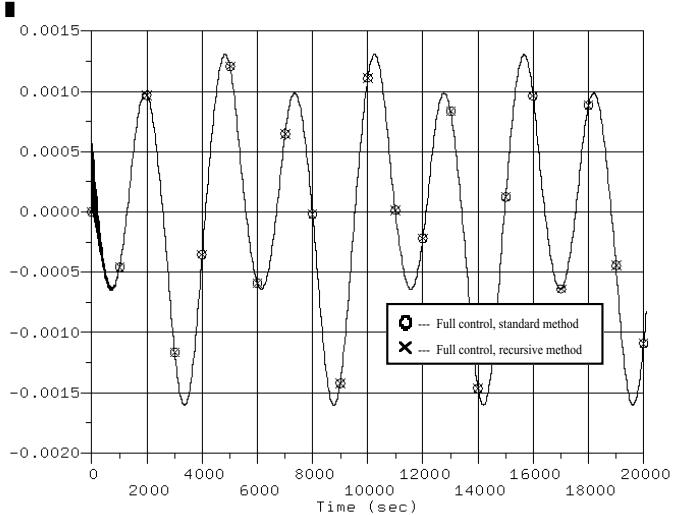
Case 1, Velocity, z-Component (meters/sec)



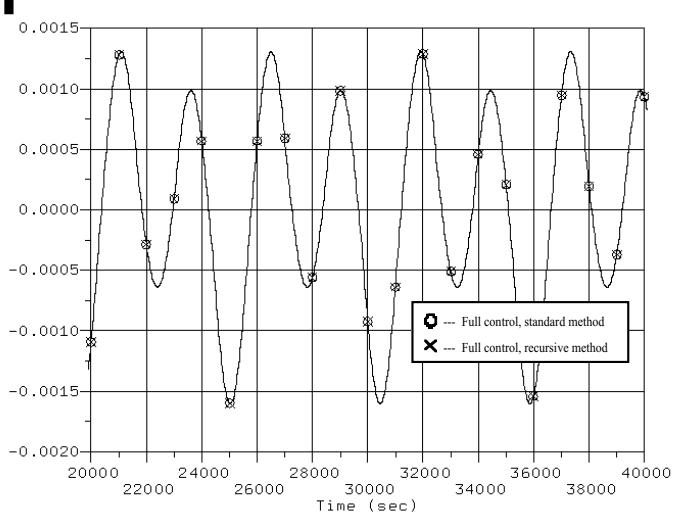
Case 1, Velocity, z-Component (meters/sec)



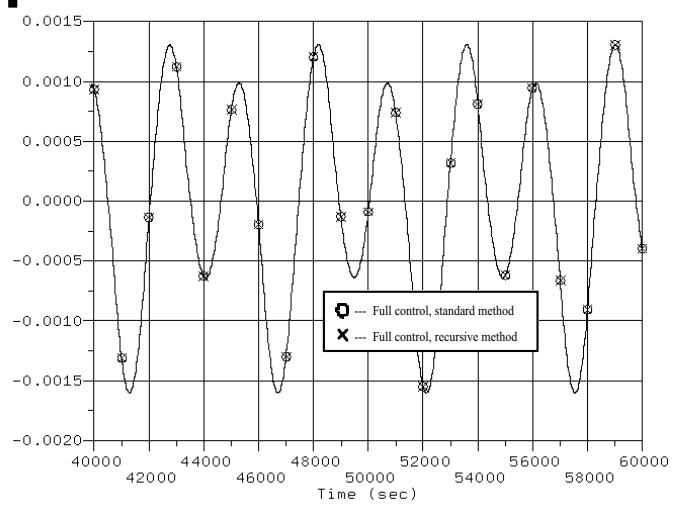
Case 1, Euler Angle, x-Axis (degrees)



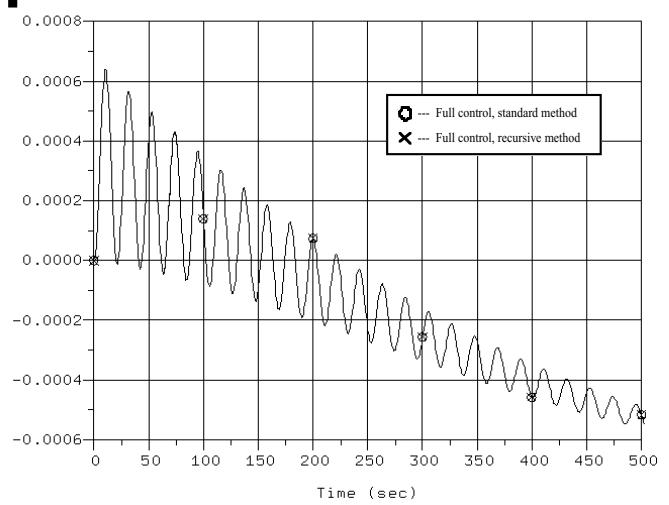
Case 1, Euler Angle, x-Axis (degrees)



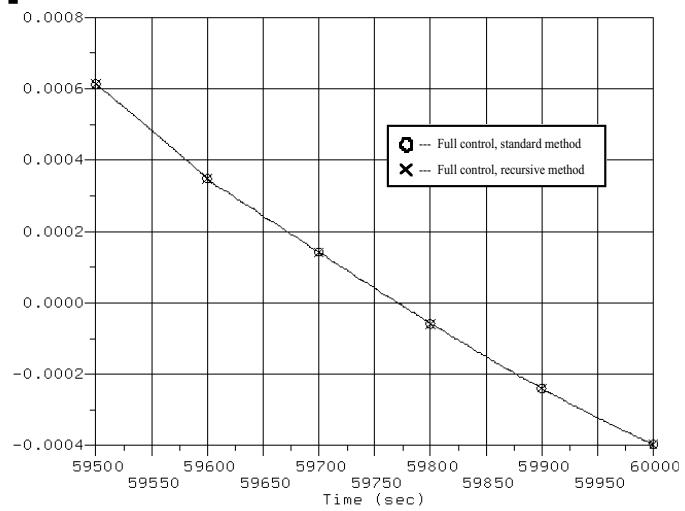
Case 1, Euler Angle, x-Axis (degrees)



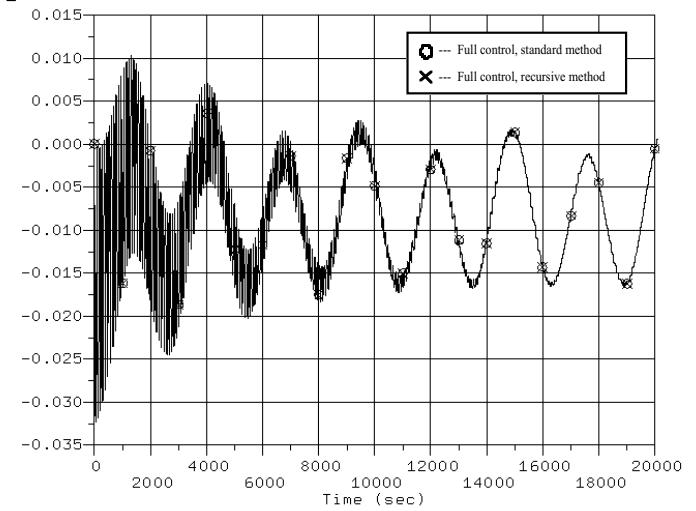
Case 1, Euler Angle, x-Axis (degrees)



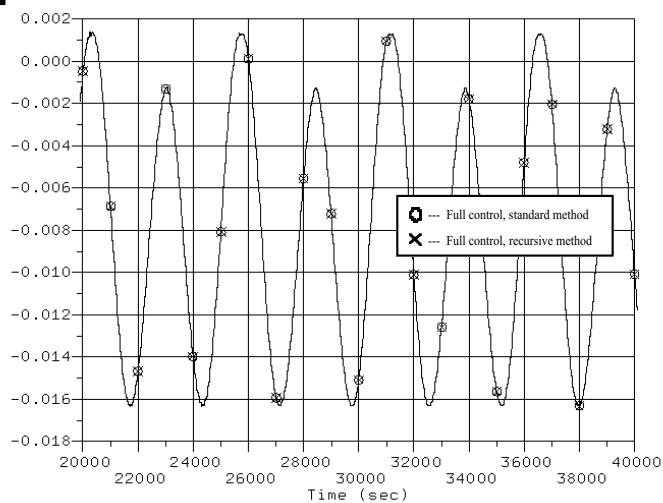
Case 1, Euler Angle, x-Axis (degrees)



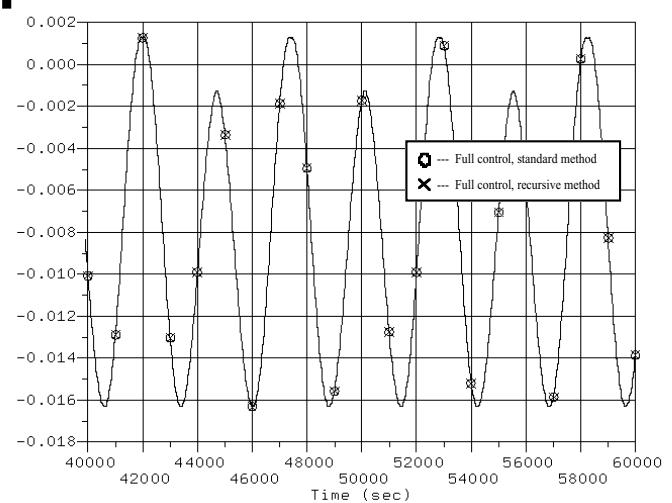
Case 1, Euler Angle, y-Axis (degrees)



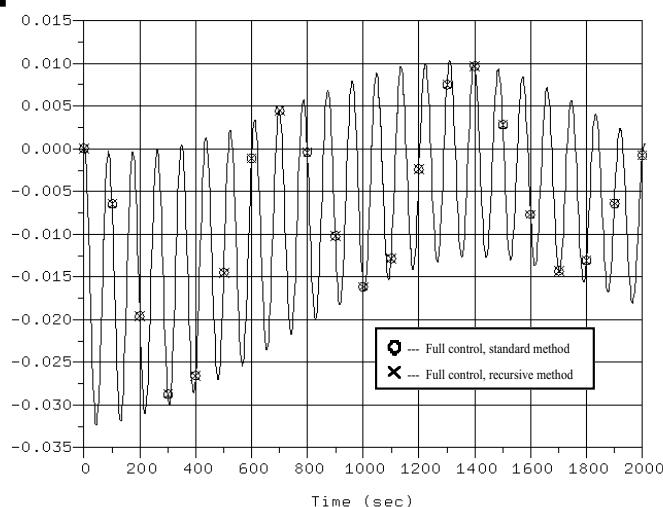
Case 1, Euler Angle, y-Axis (degrees)



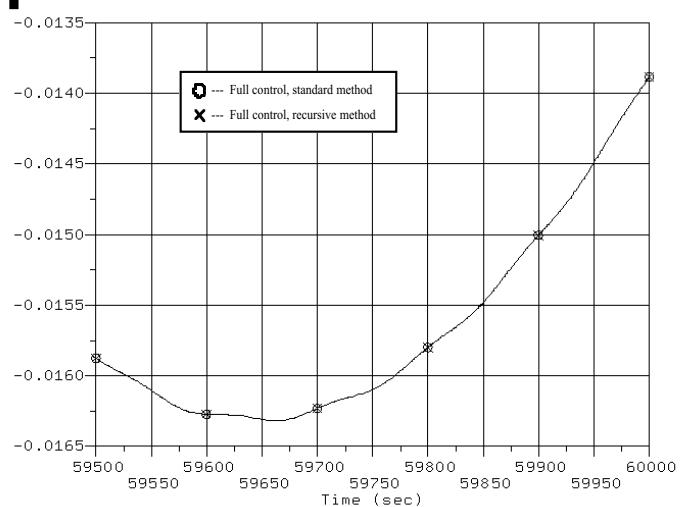
Case 1, Euler Angle, y-Axis (degrees)



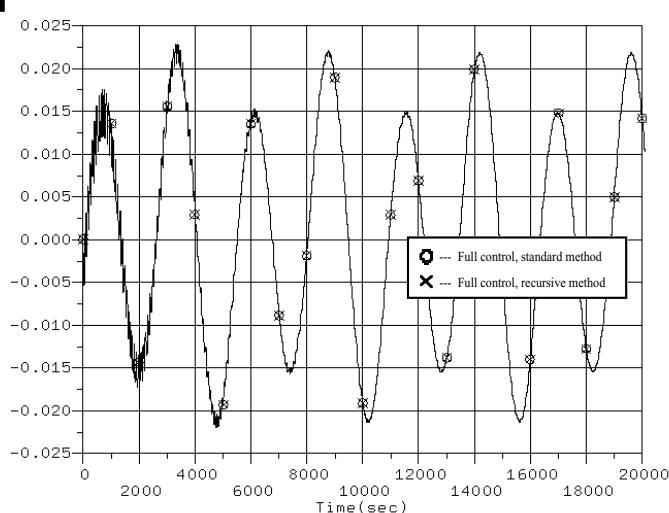
Case 1, Euler Angle, y-Axis (degrees)



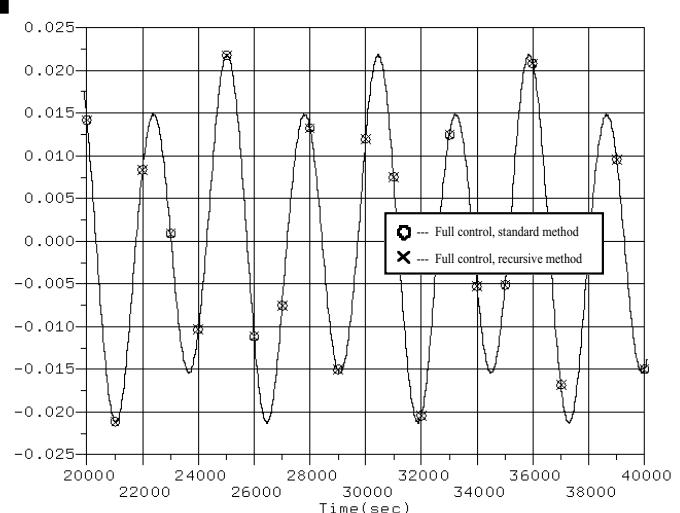
Case 1, Euler Angle, y-Axis (degrees)



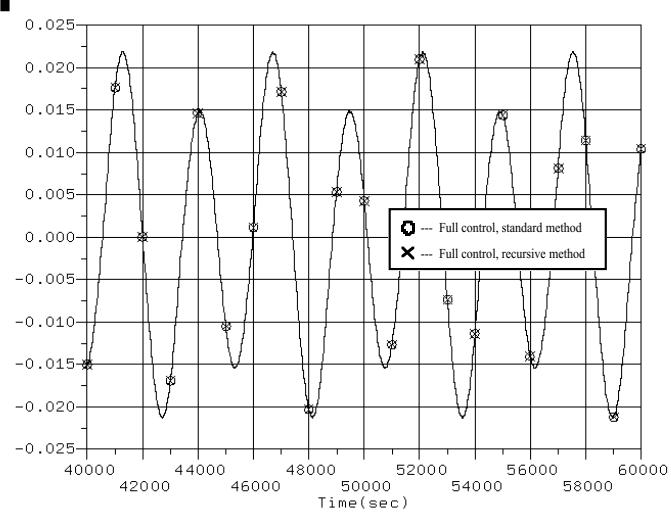
Case 1, Euler Angle, z-Axis (deg)



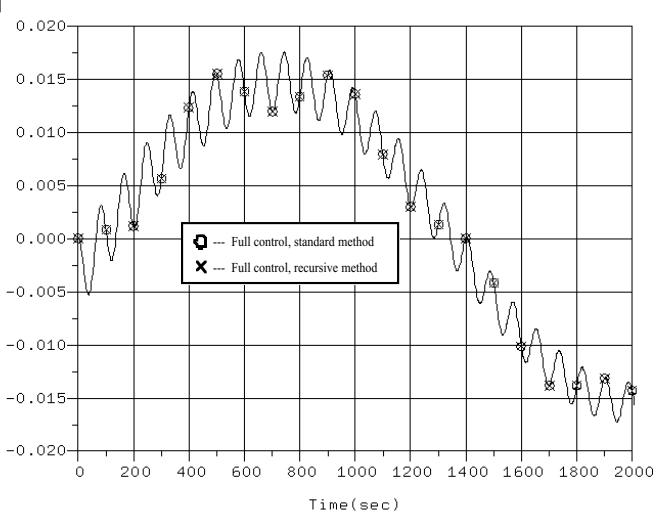
Case 1, Euler Angle, z-Axis (deg)



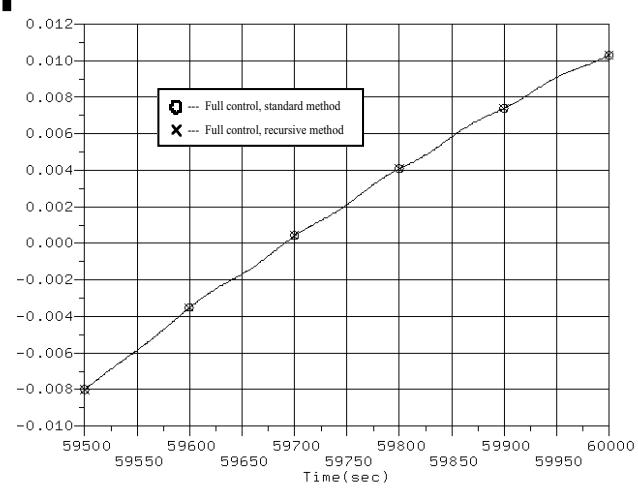
Case 1, Euler Angle, z-Axis (deg)



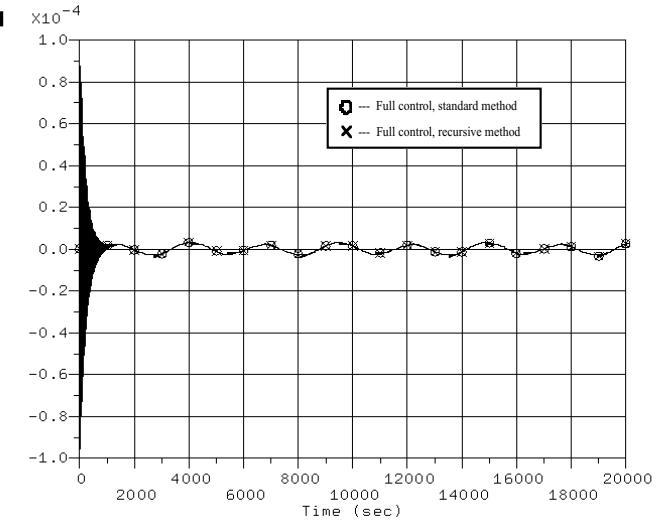
Case 1, Euler Angle, z-Axis (deg)



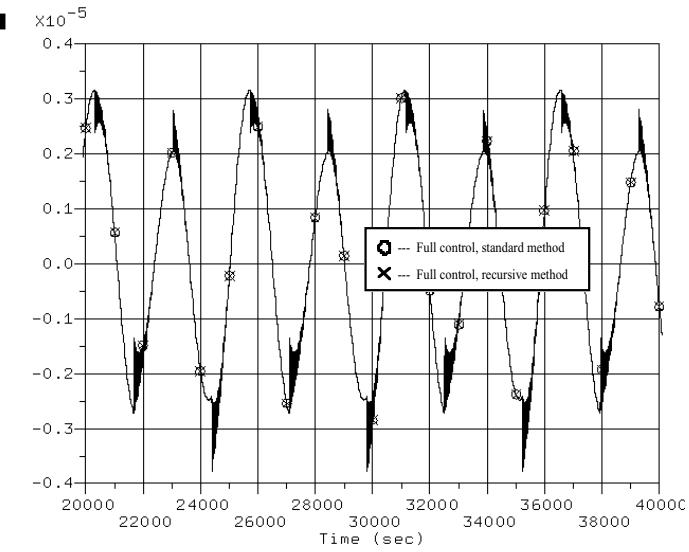
Case 1, Euler Angle, z-Axis (deg)



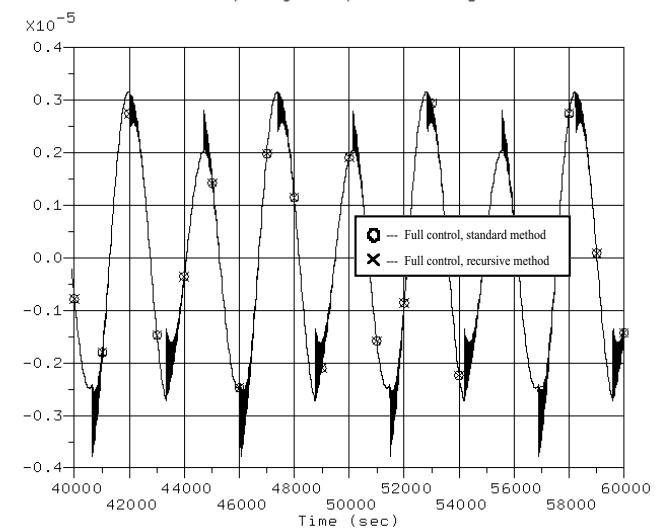
Case 1, Body Rate, x-Axis (deg/sec)

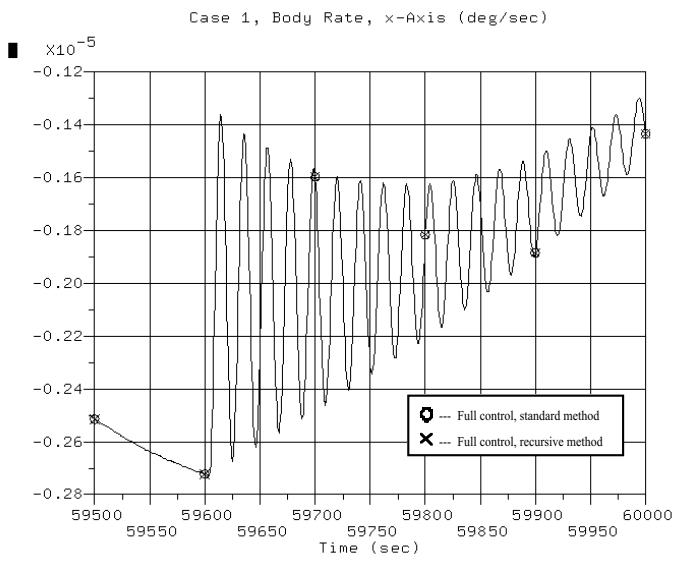
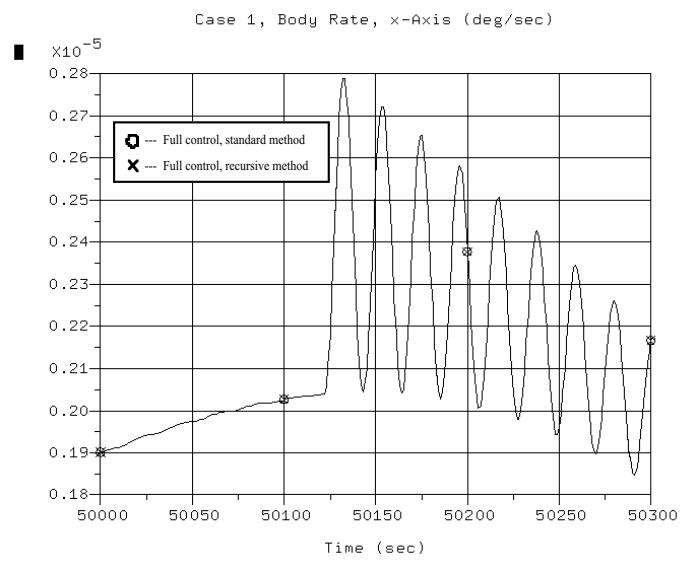
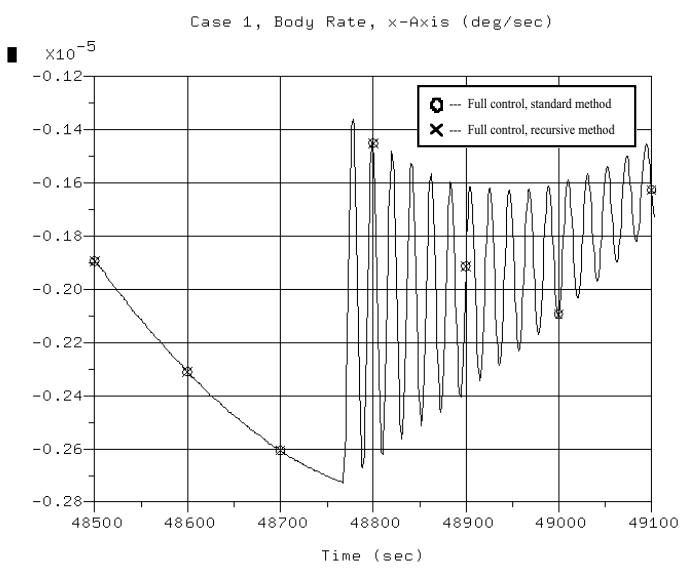
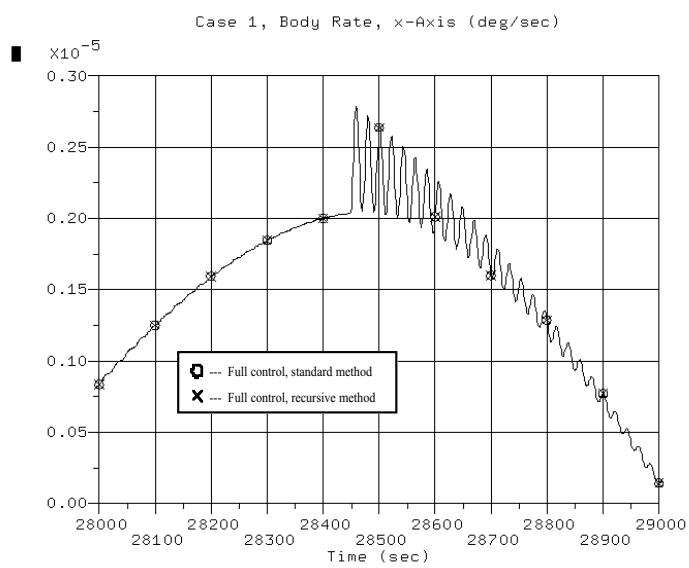
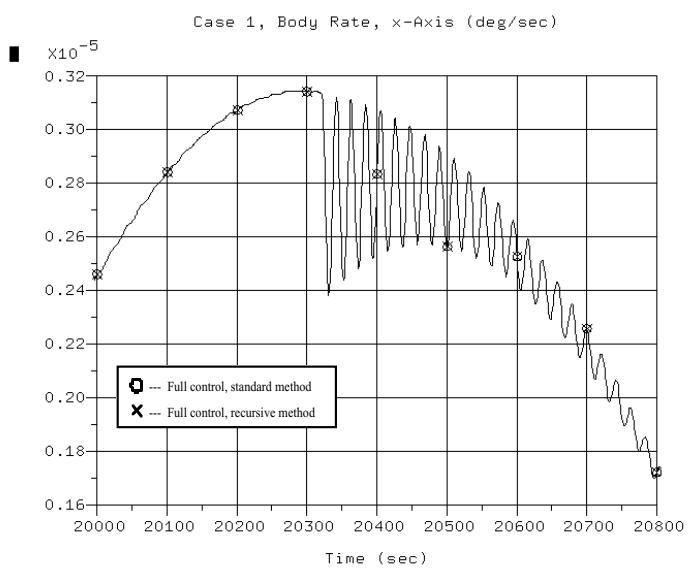
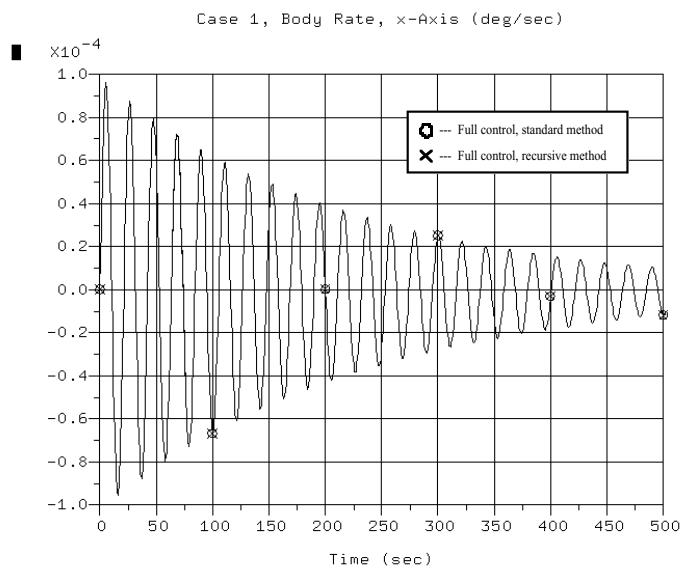


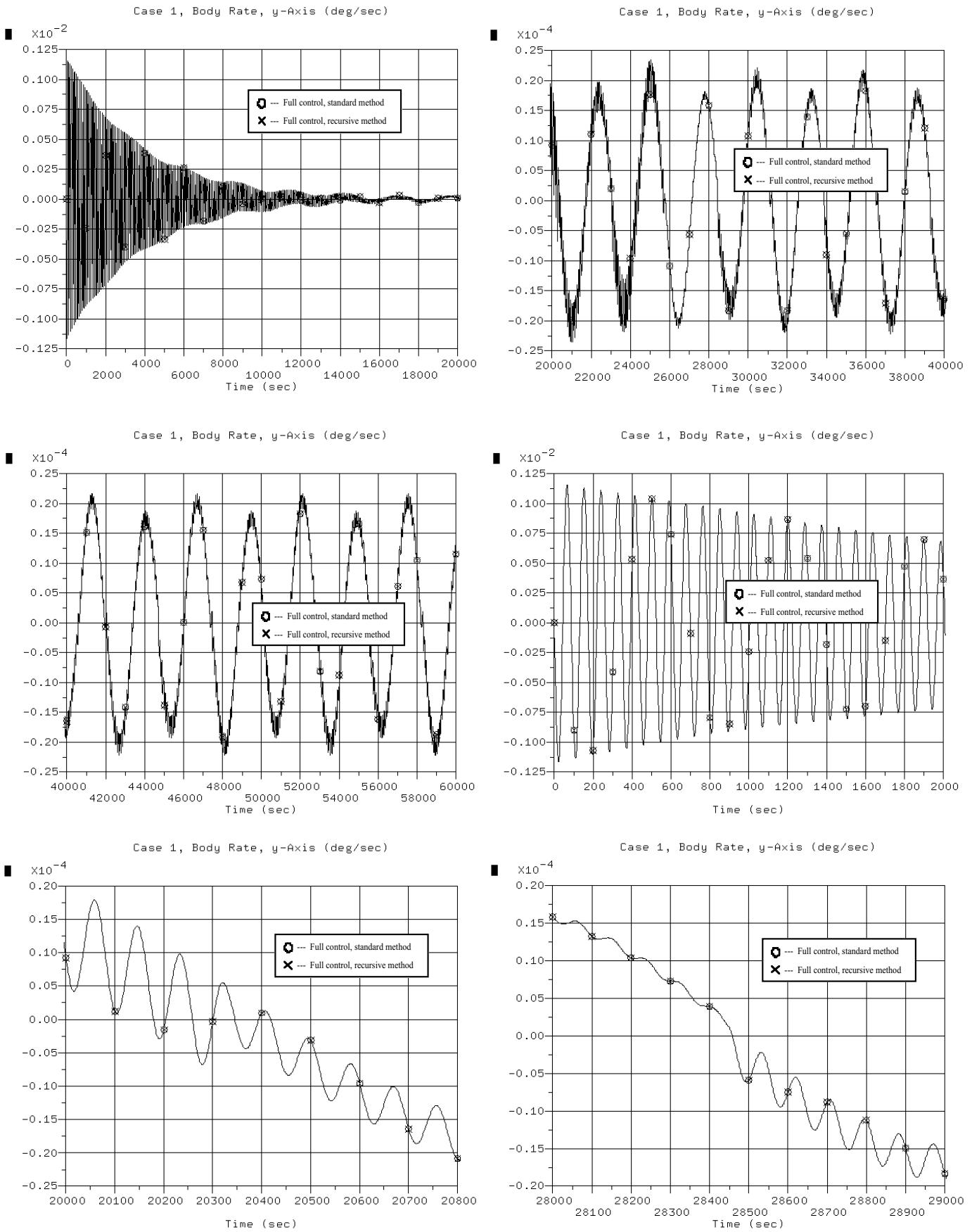
Case 1, Body Rate, x-Axis (deg/sec)

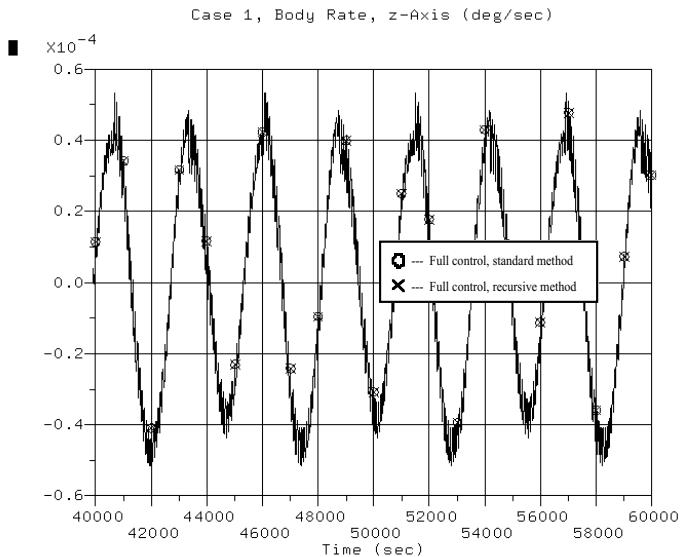
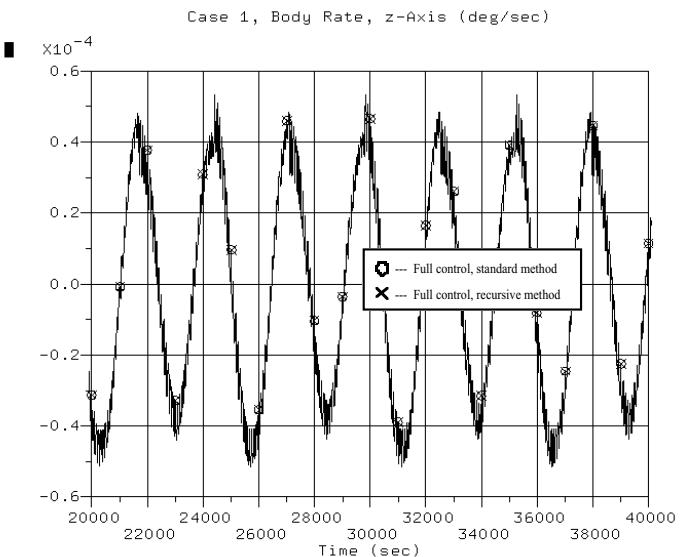
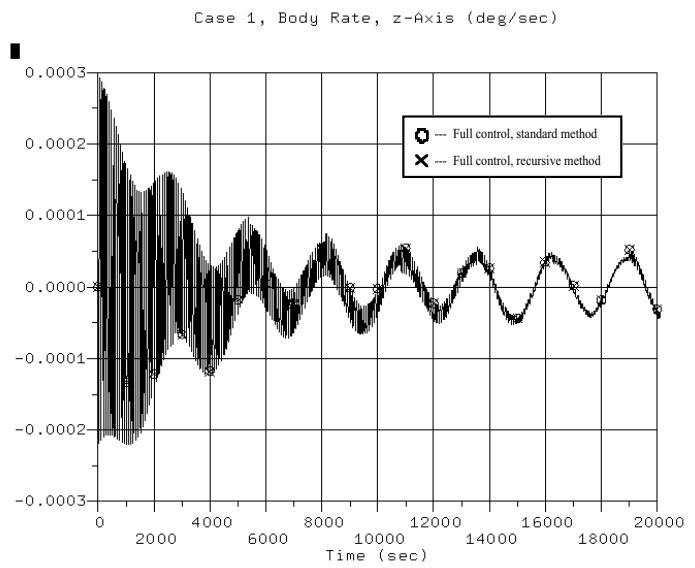
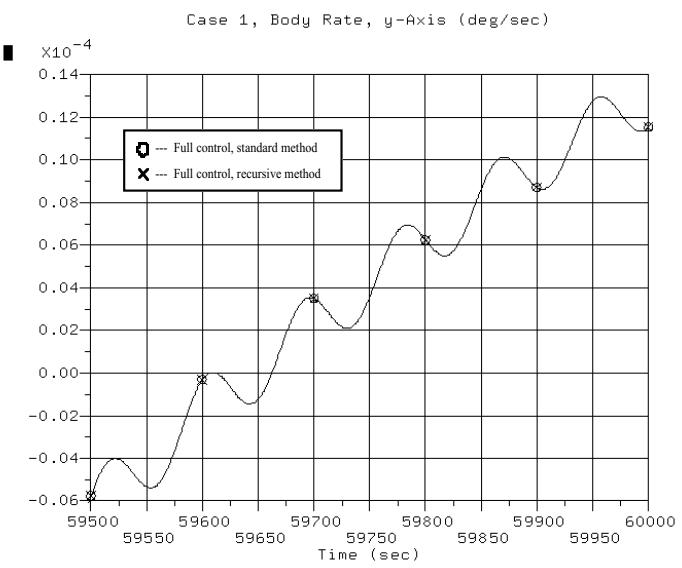
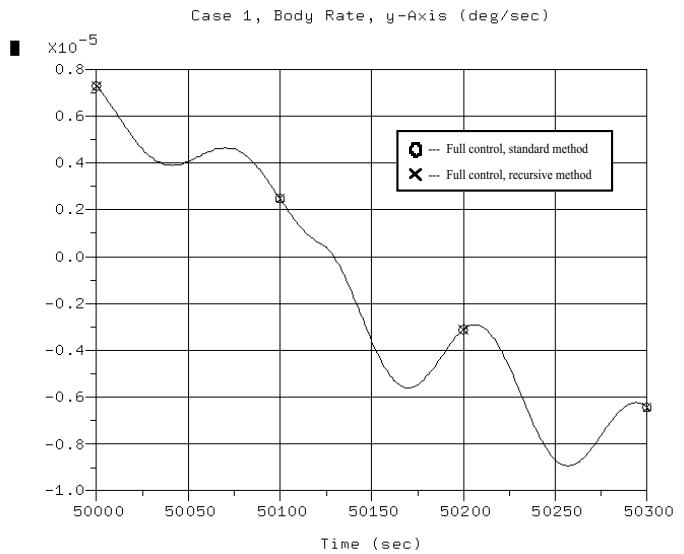
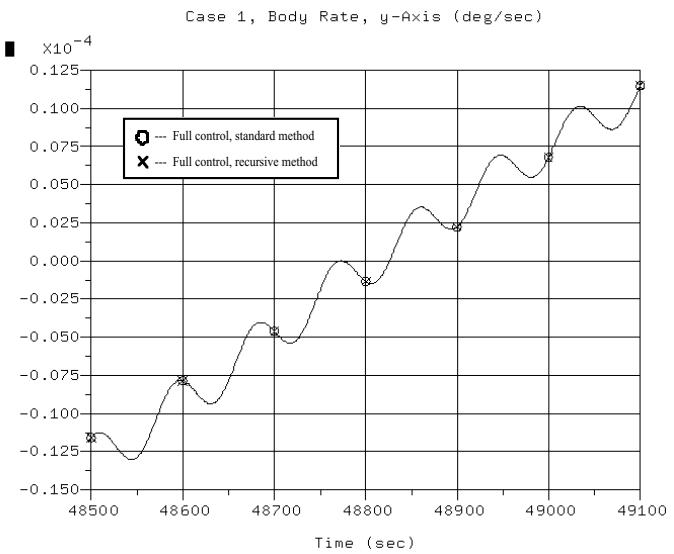


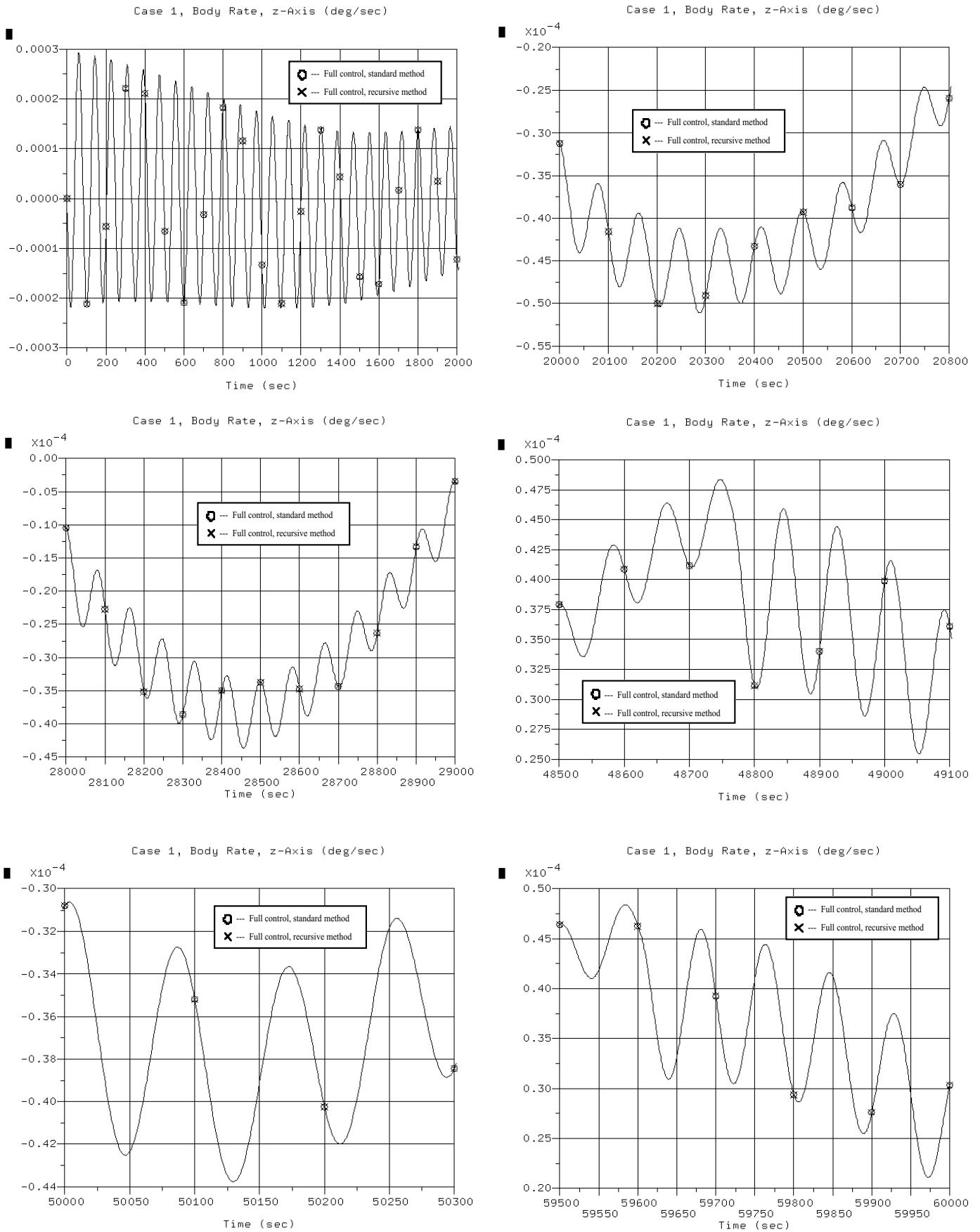
Case 1, Body Rate, x-Axis (deg/sec)



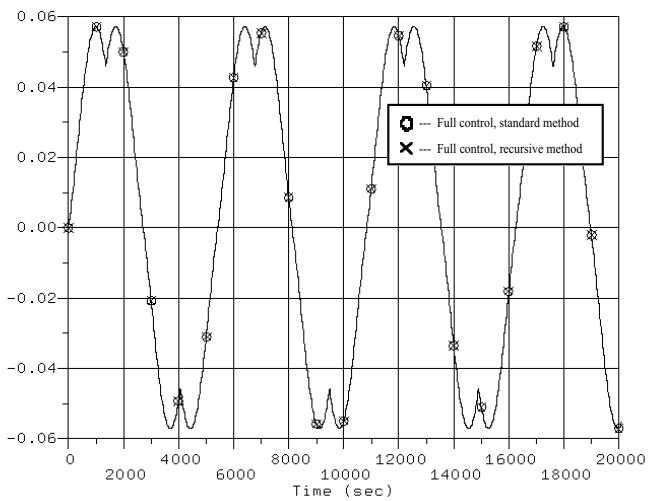




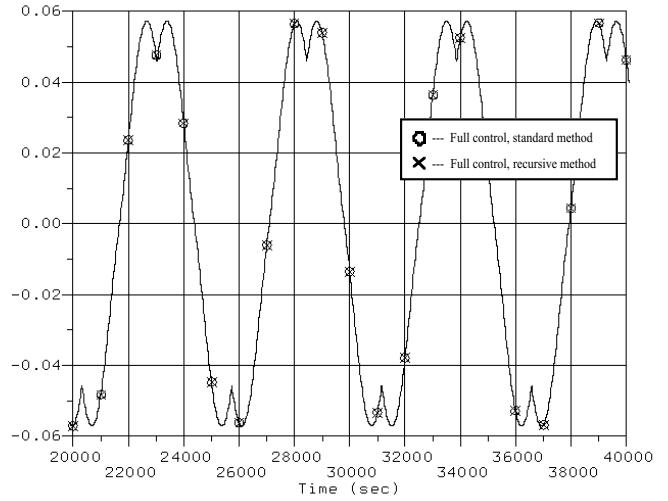




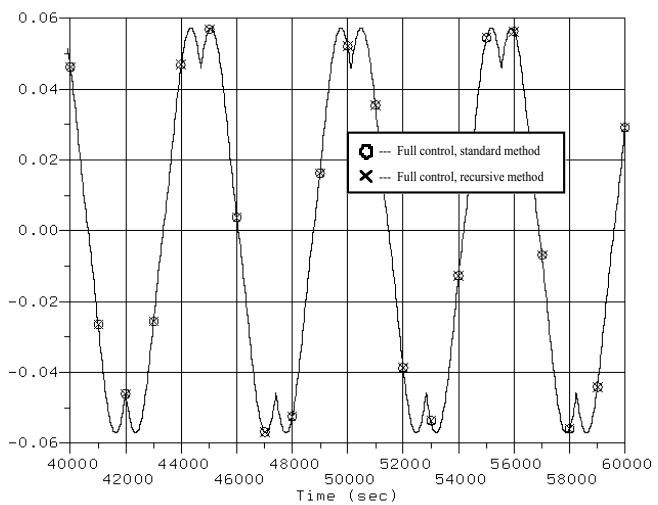
Case 1, Aerodynamic Force, x-Component (newtons)



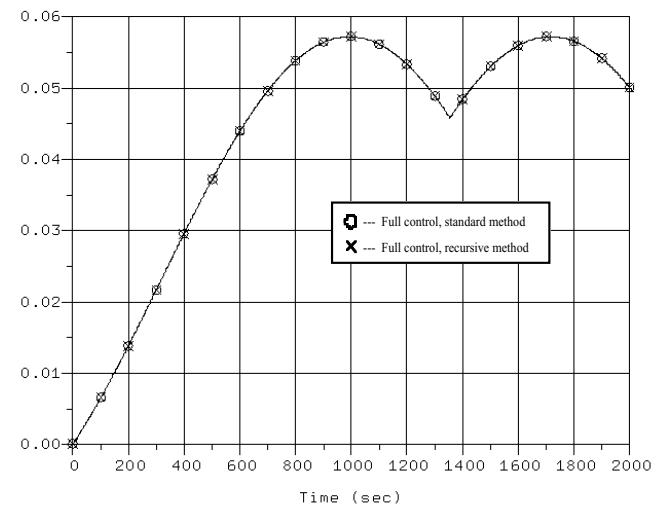
Case 1, Aerodynamic Force, x-Component (newtons)



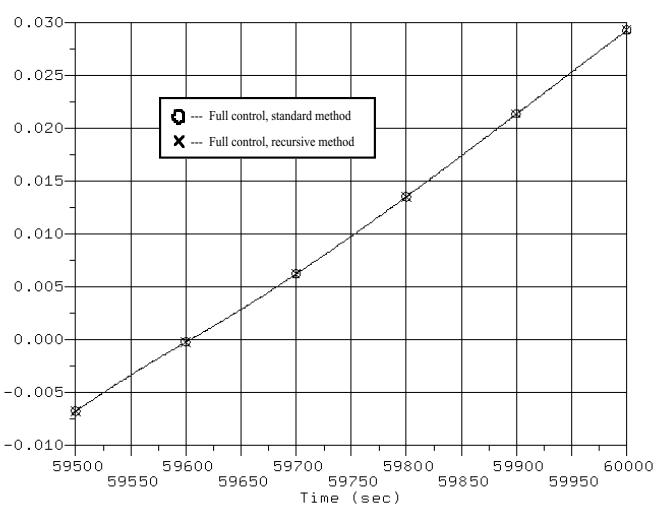
Case 1, Aerodynamic Force, x-Component (newtons)



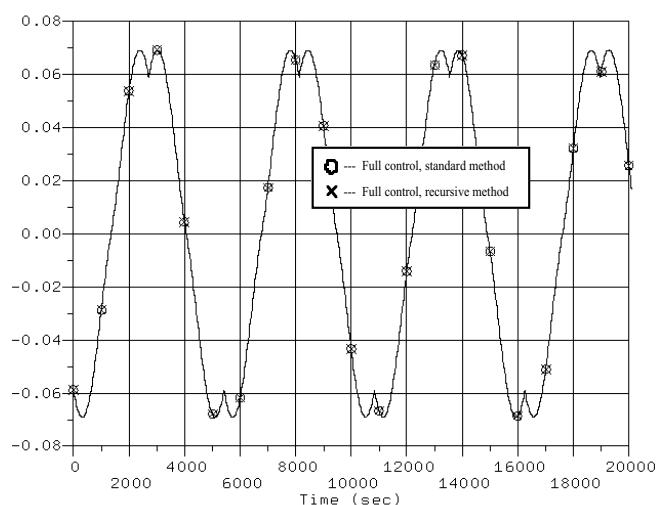
Case 1, Aerodynamic Force, x-Component (newtons)



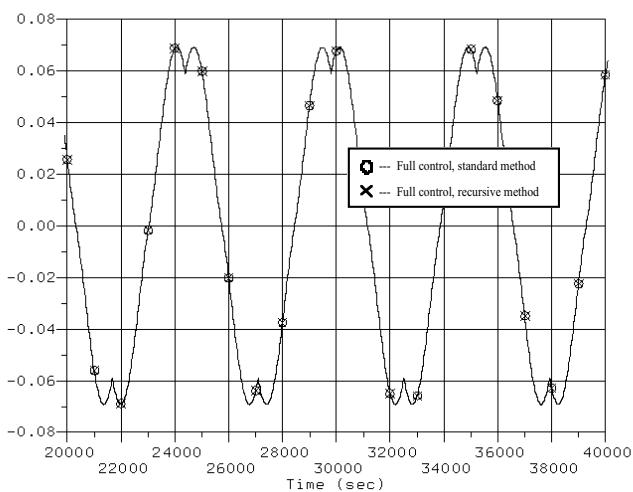
Case 1, Aerodynamic Force, x-Component (newtons)



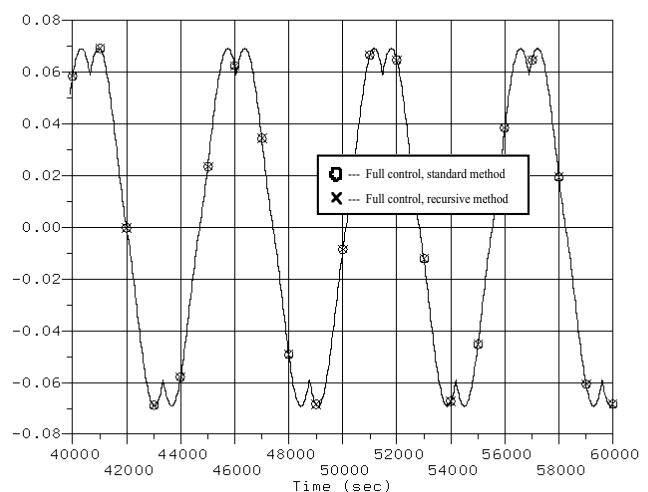
Case 1, Aerodynamic Force, y-Component (newtons)



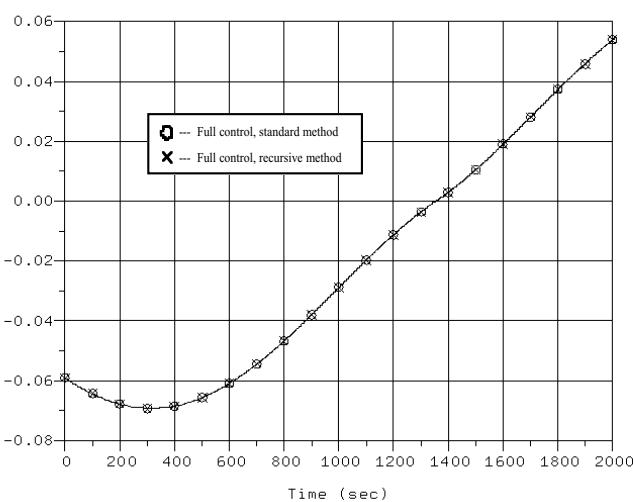
Case 1, Aerodynamic Force, y-Component (newtons)



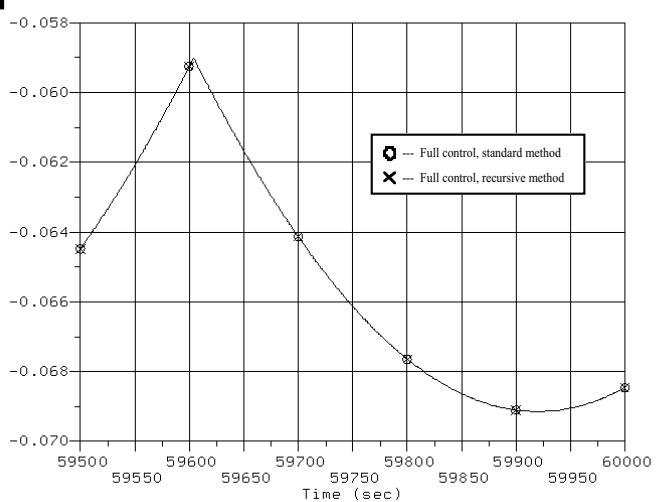
Case 1, Aerodynamic Force, y-Component (newtons)



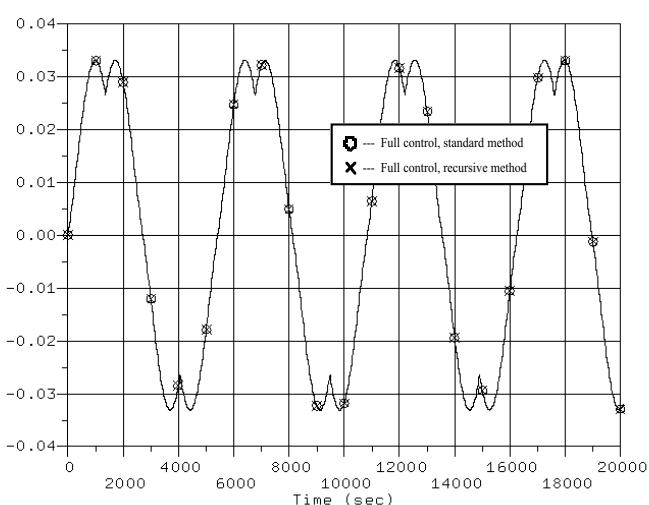
Case 1, Aerodynamic Force, y-Component (newtons)



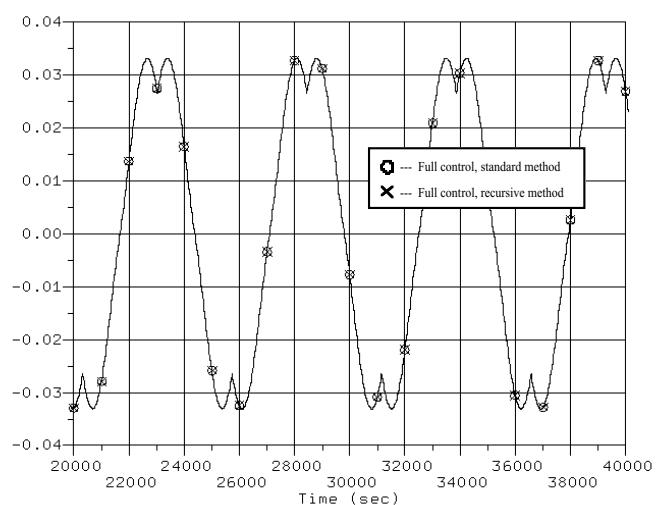
Case 1, Aerodynamic Force, y-Component (newtons)



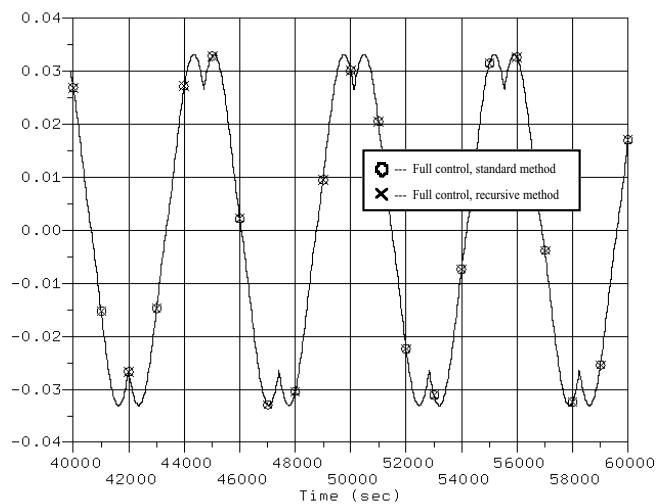
Case 1, Aerodynamic Force, z-Component (newtons)



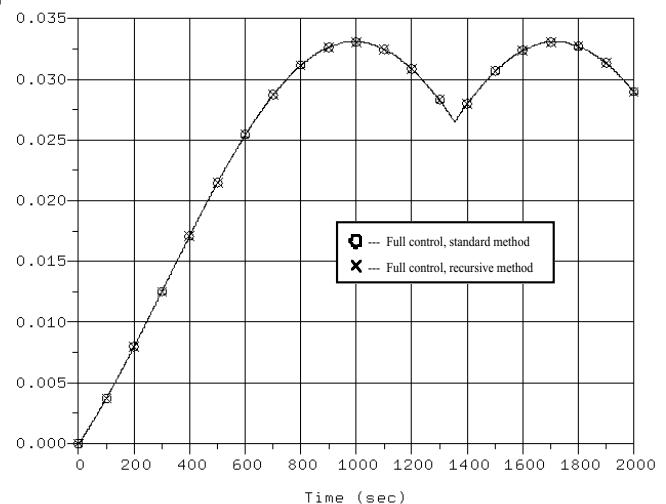
Case 1, Aerodynamic Force, z-Component (newtons)



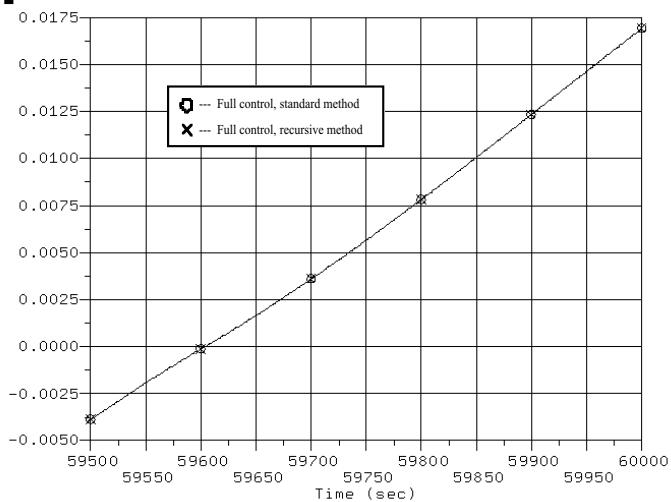
Case 1, Aerodynamic Force, z-Component (newtons)



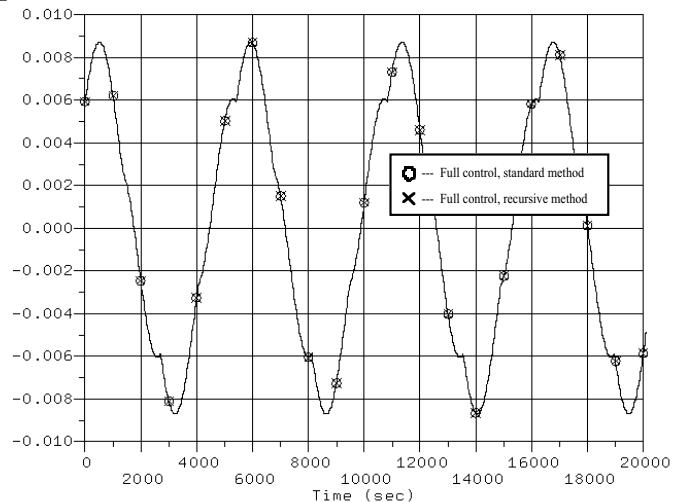
Case 1, Aerodynamic Force, z-Component (newtons)



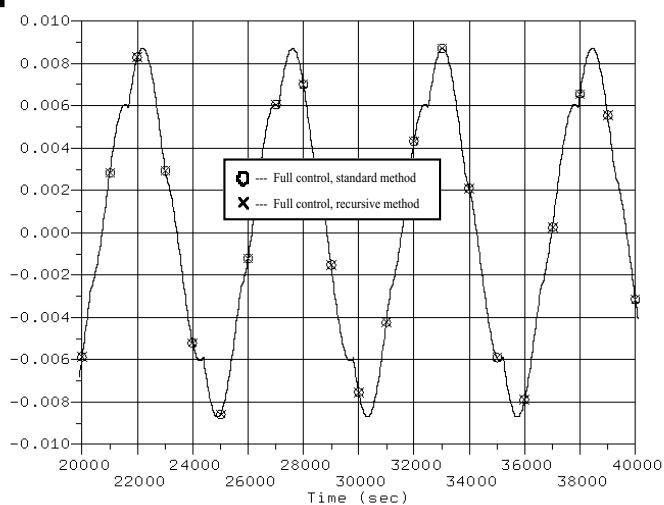
Case 1, Aerodynamic Force, z-Component (newtons)



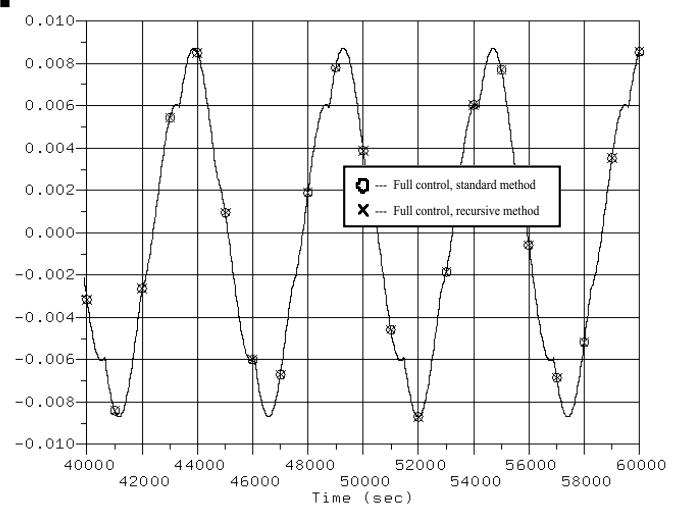
Case 1, Aerodynamic Torque, x-Component (newton-meters)



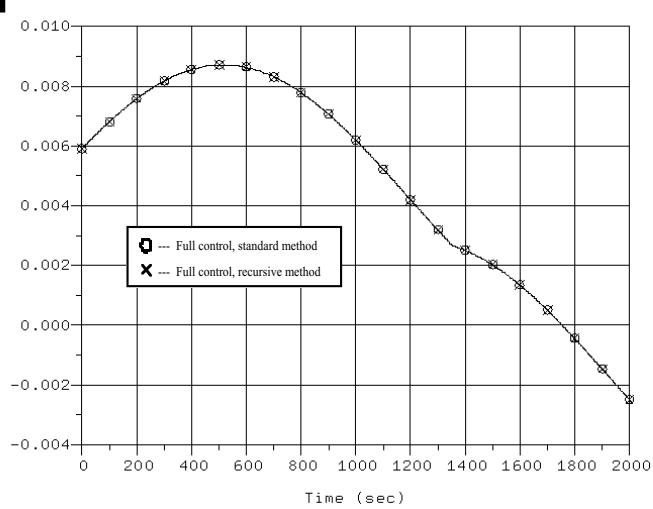
Case 1, Aerodynamic Torque, x-Component (newton-meters)



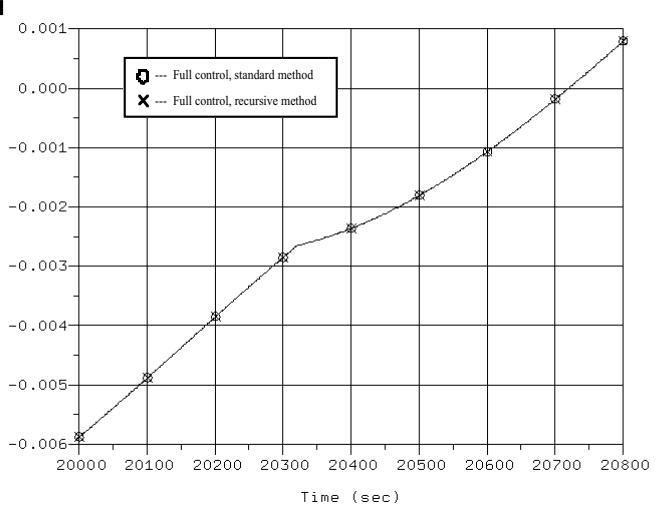
Case 1, Aerodynamic Torque, x-Component (newton-meters)



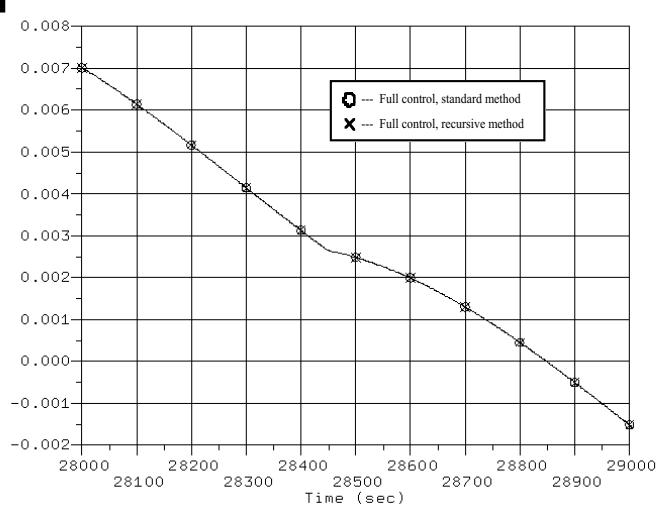
Case 1, Aerodynamic Torque, x-Component (newton-meters)



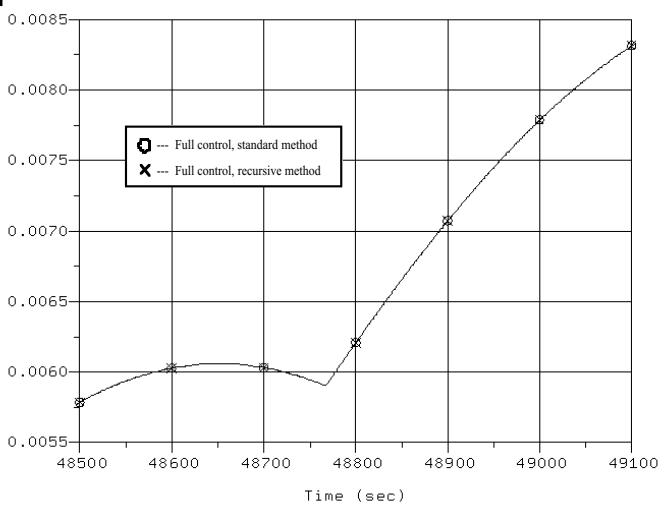
Case 1, Aerodynamic Torque, x-Component (newton-meters)



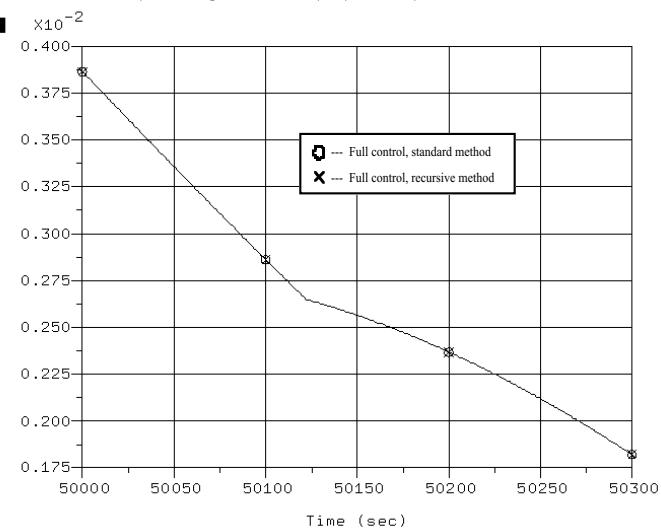
Case 1, Aerodynamic Torque, x-Component (newton-meters)



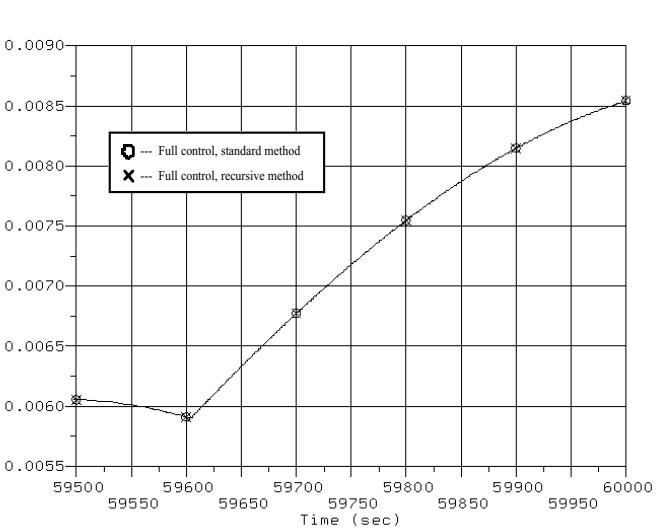
Case 1, Aerodynamic Torque, x-Component (newton-meters)



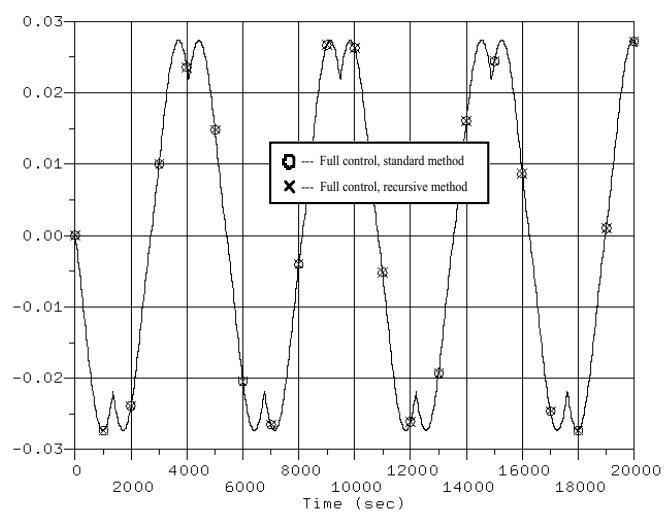
Case 1, Aerodynamic Torque, x-Component (newton-meters)



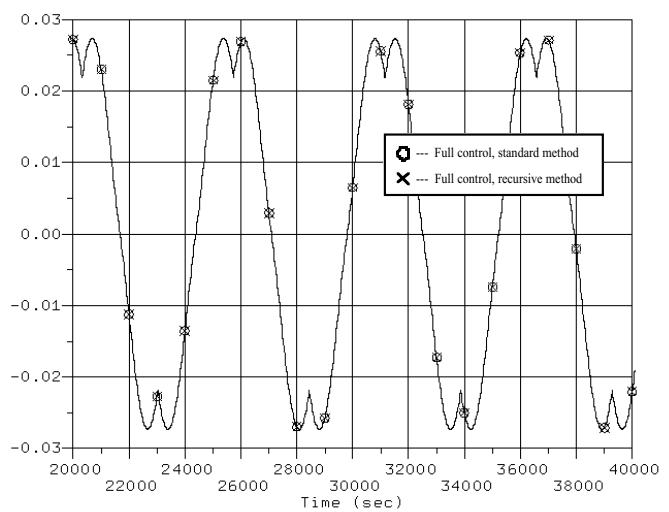
Case 1, Aerodynamic Torque, x-Component (newton-meters)



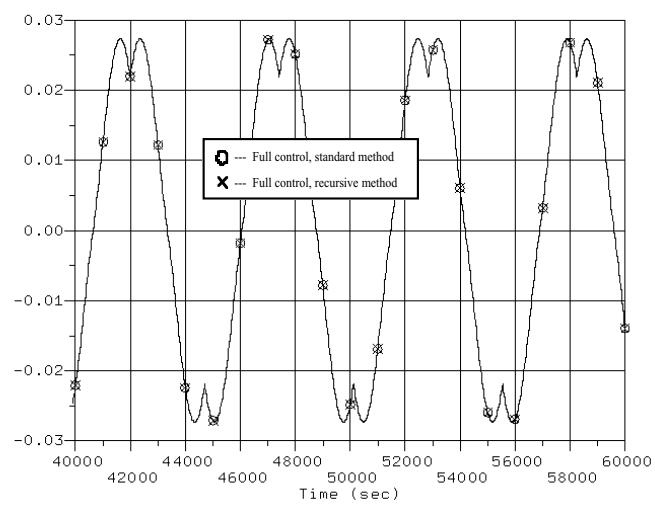
Case 1, Aerodynamic Torque, y-Component (newton-meters)



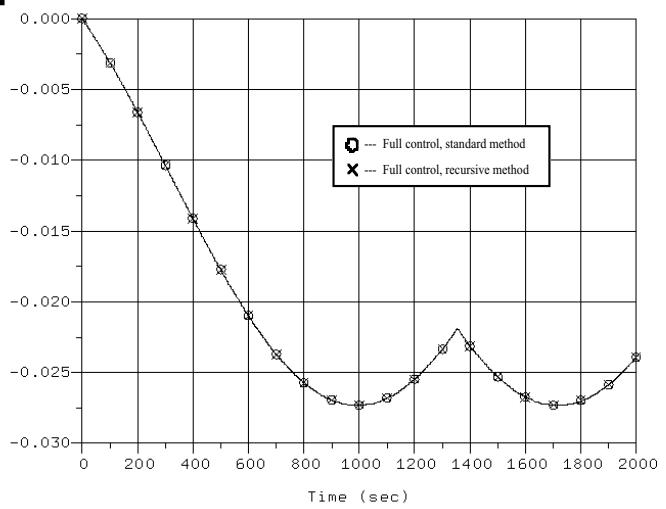
Case 1, Aerodynamic Torque, y-Component (newton-meters)



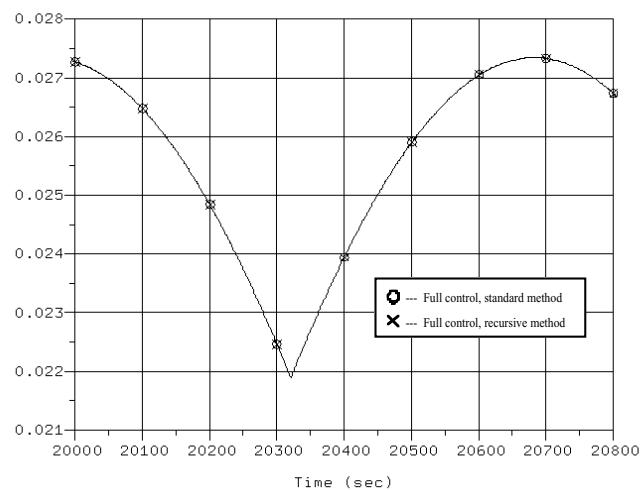
Case 1, Aerodynamic Torque, y-Component (newton-meters)



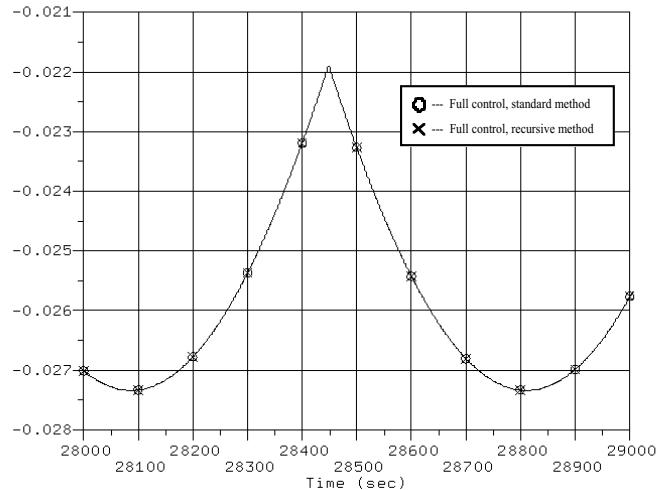
Case 1, Aerodynamic Torque, y-Component (newton-meters)



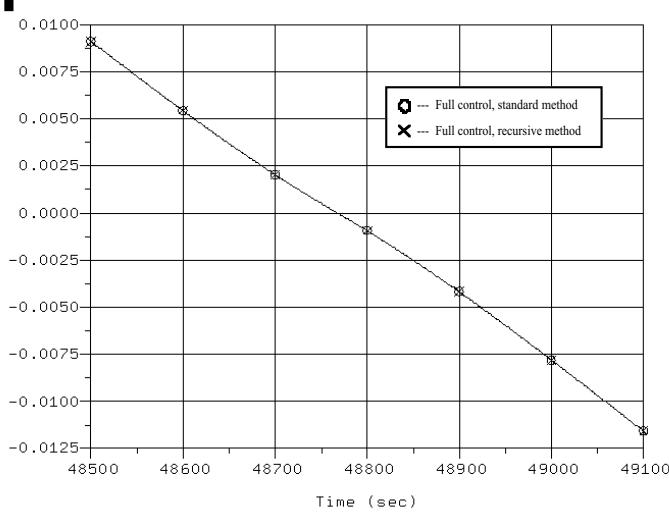
Case 1, Aerodynamic Torque, y-Component (newton-meters)



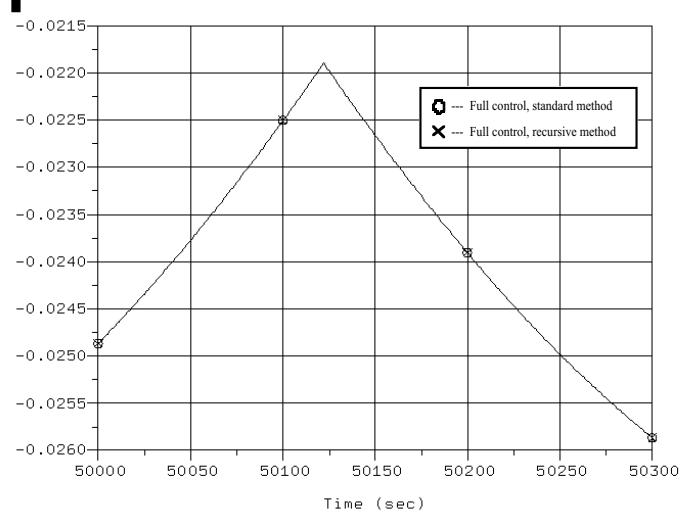
Case 1, Aerodynamic Torque, y-Component (newton-meters)



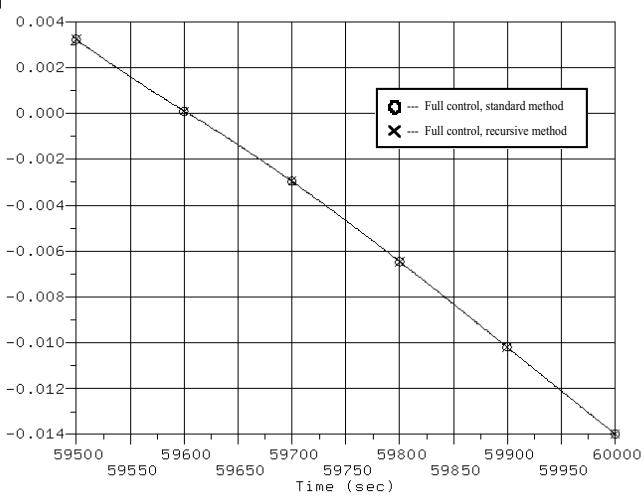
Case 1, Aerodynamic Torque, y-Component (newton-meters)



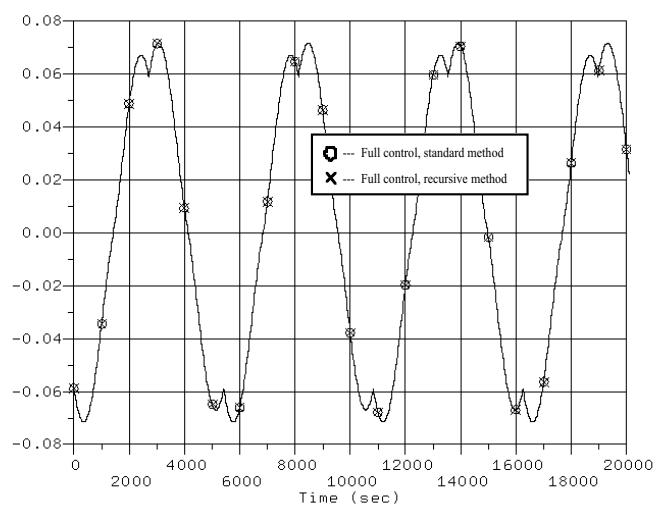
Case 1, Aerodynamic Torque, y-Component (newton-meters)



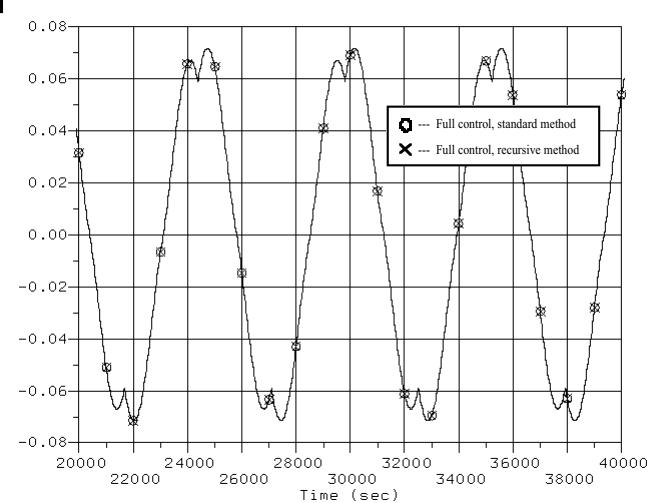
Case 1, Aerodynamic Torque, y-Component (newton-meters)



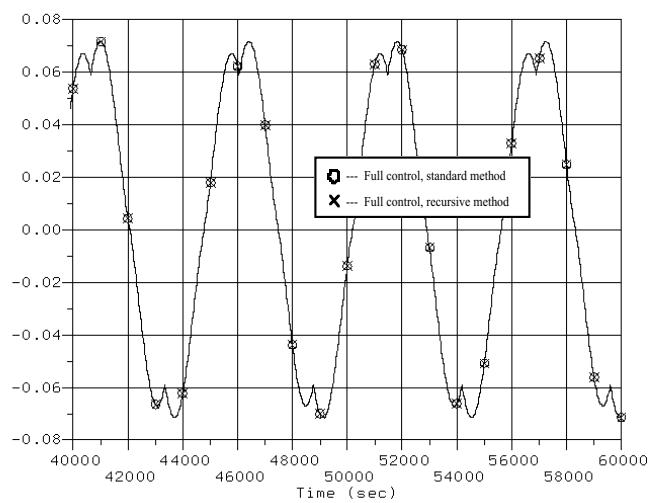
Case 1, Aerodynamic Torque, z-Component (newton-meters)



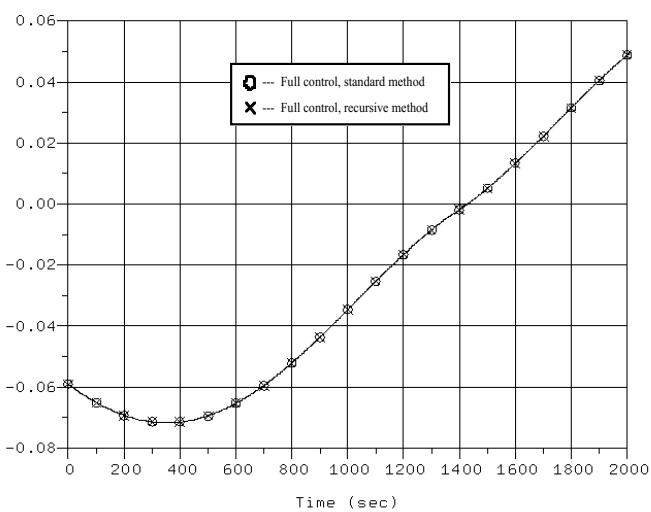
Case 1, Aerodynamic Torque, z-Component (newton-meters)



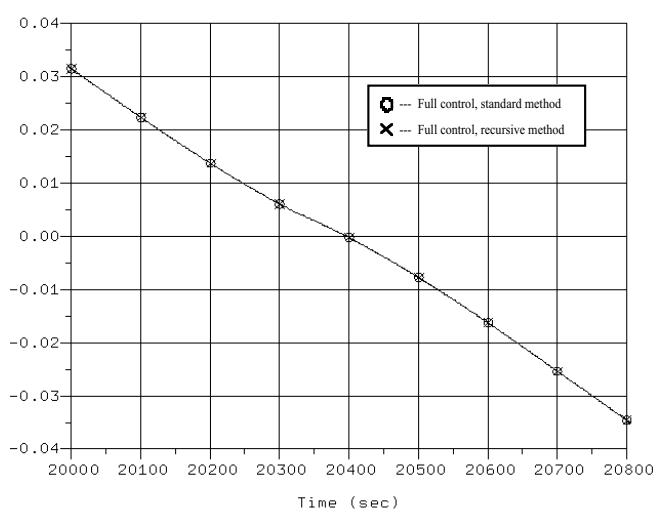
Case 1, Aerodynamic Torque, z-Component (newton-meters)



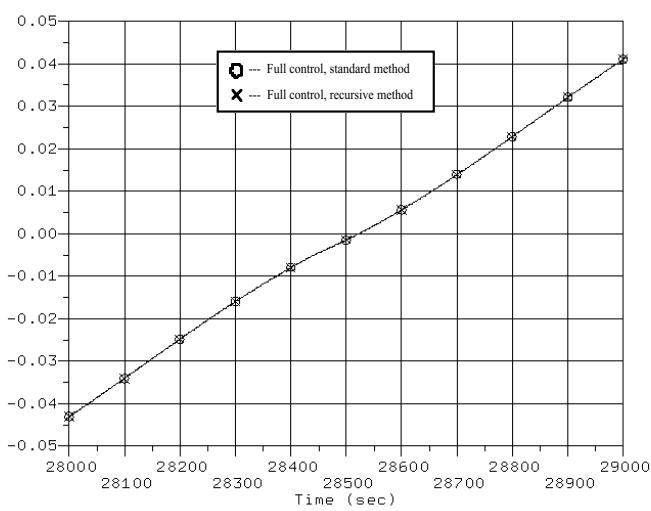
Case 1, Aerodynamic Torque, z-Component (newton-meters)



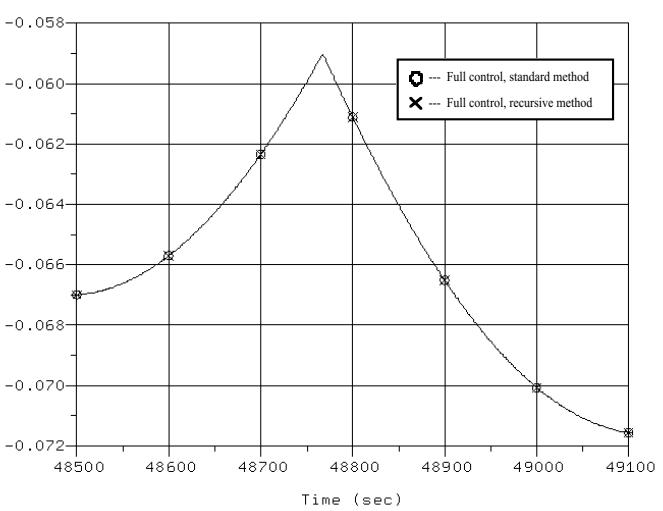
Case 1, Aerodynamic Torque, z-Component (newton-meters)



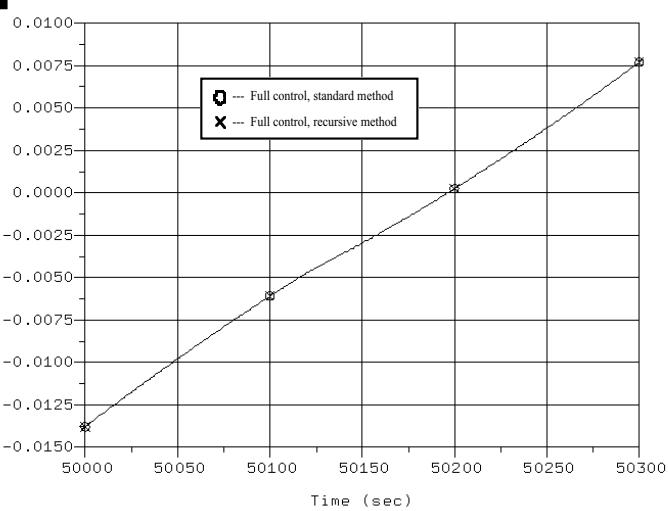
Case 1, Aerodynamic Torque, z-Component (newton-meters)



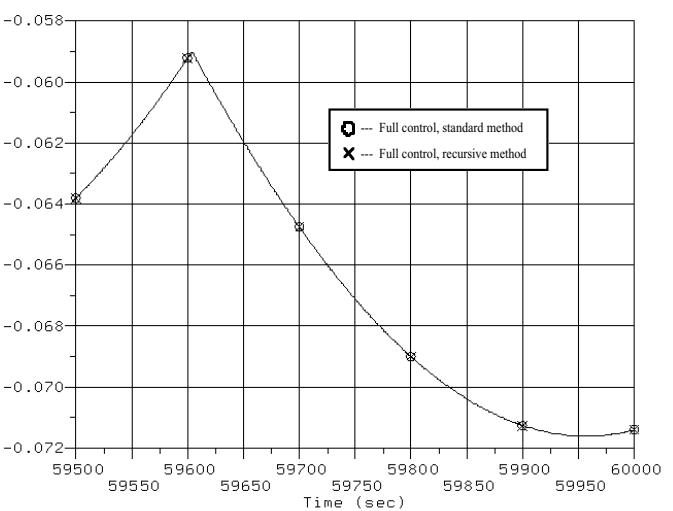
Case 1, Aerodynamic Torque, z-Component (newton-meters)



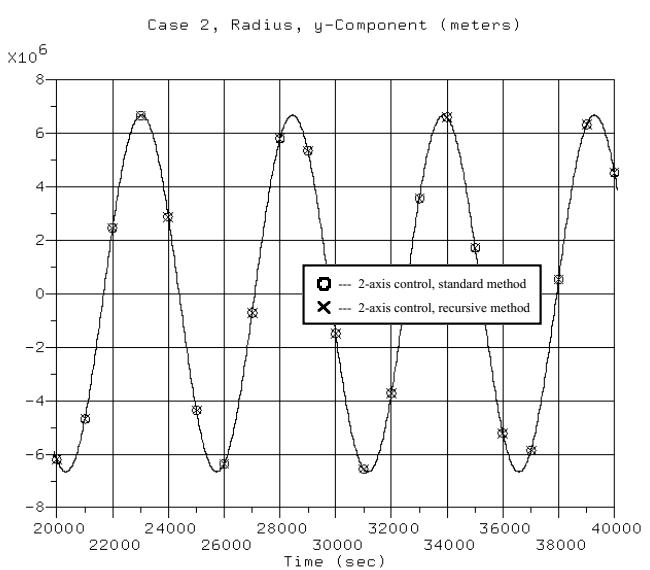
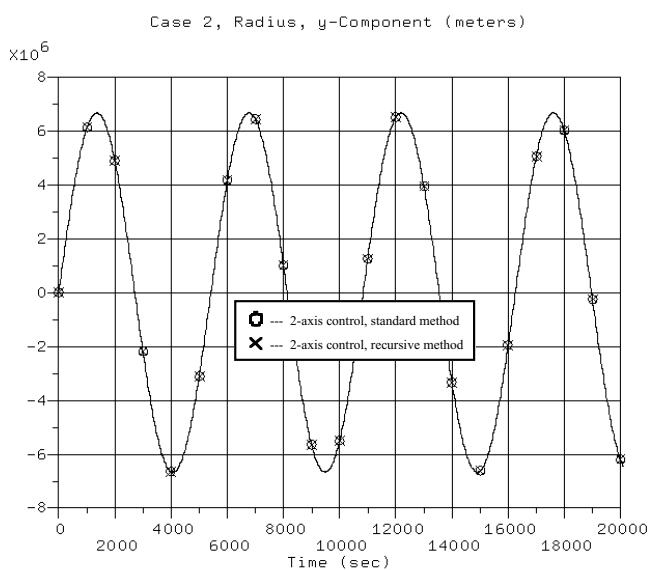
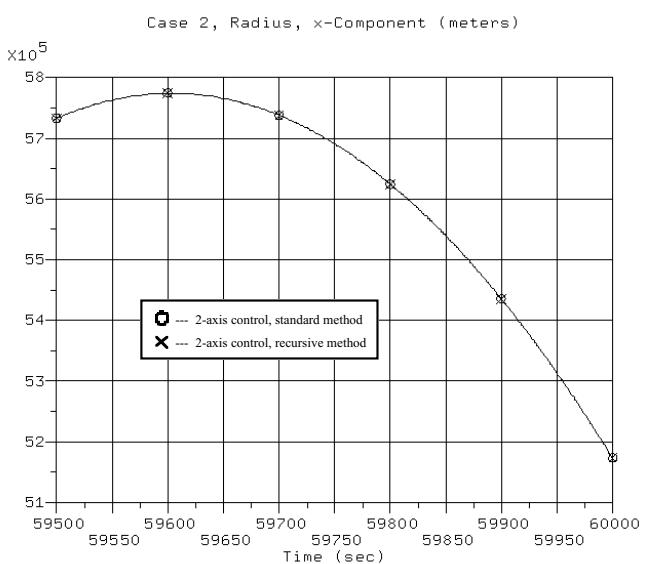
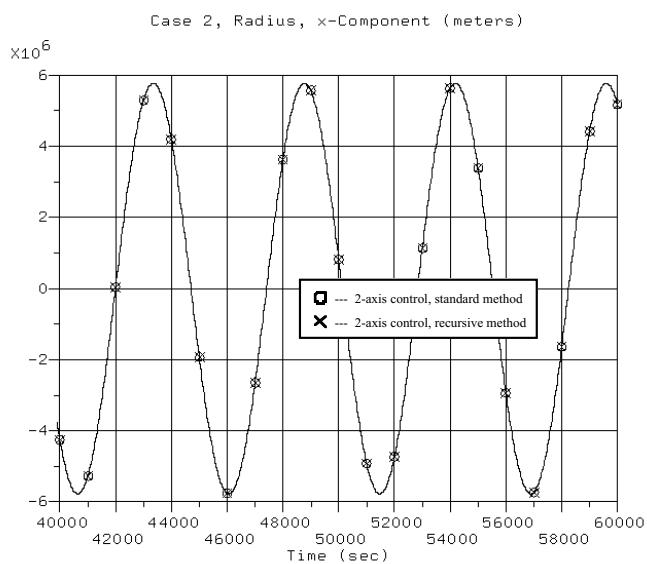
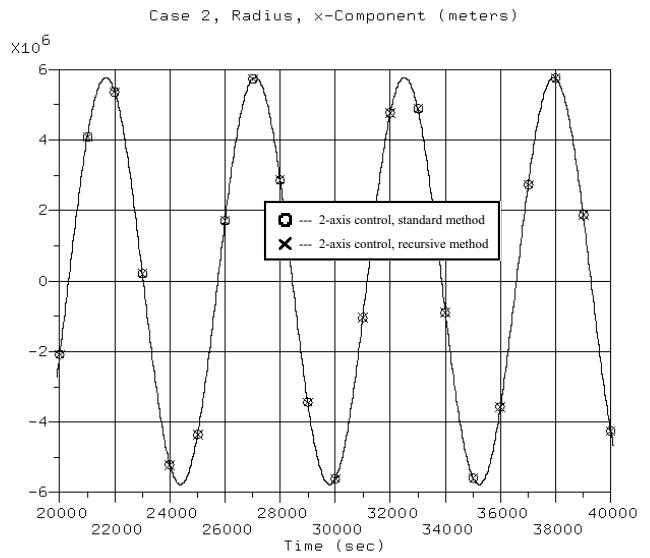
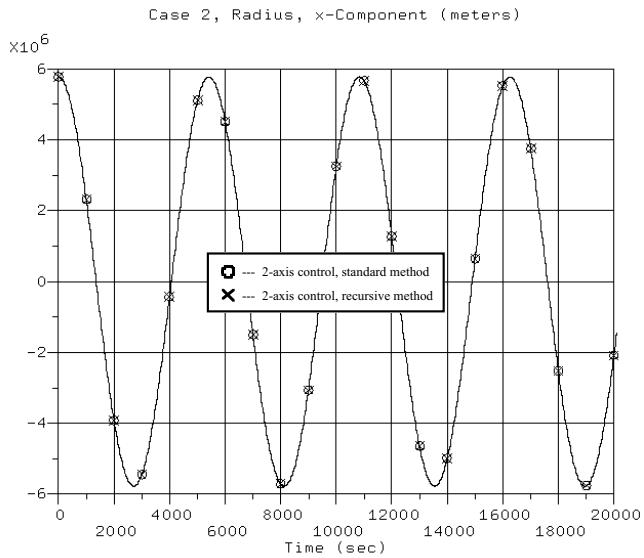
Case 1, Aerodynamic Torque, z-Component (newton-meters)

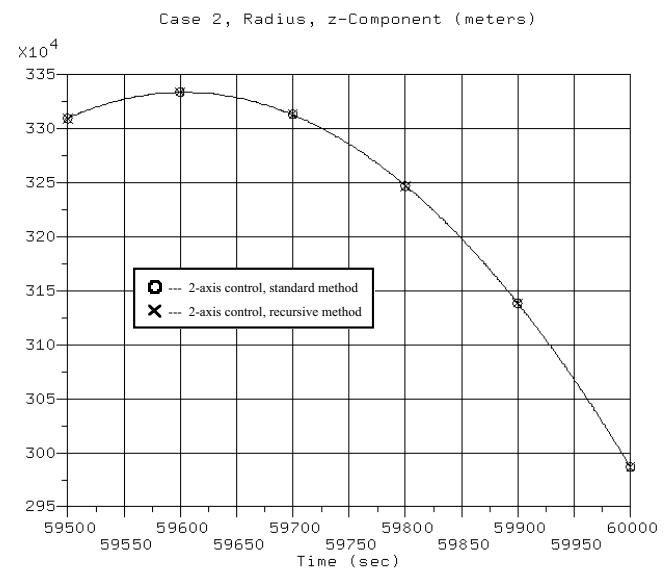
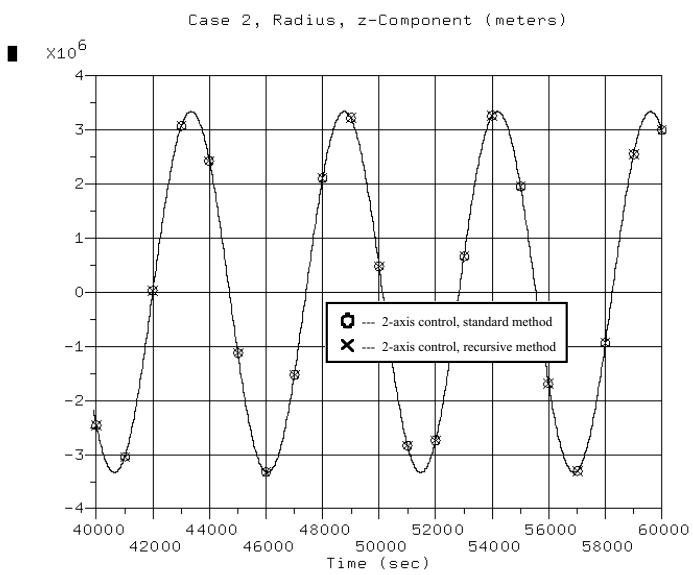
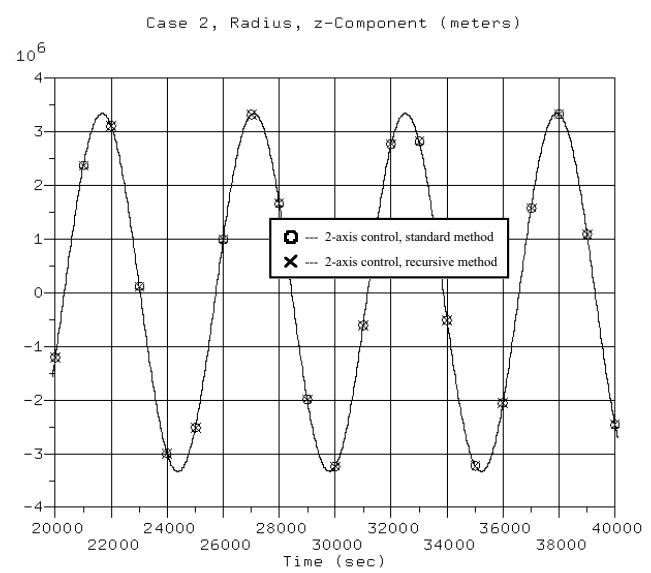
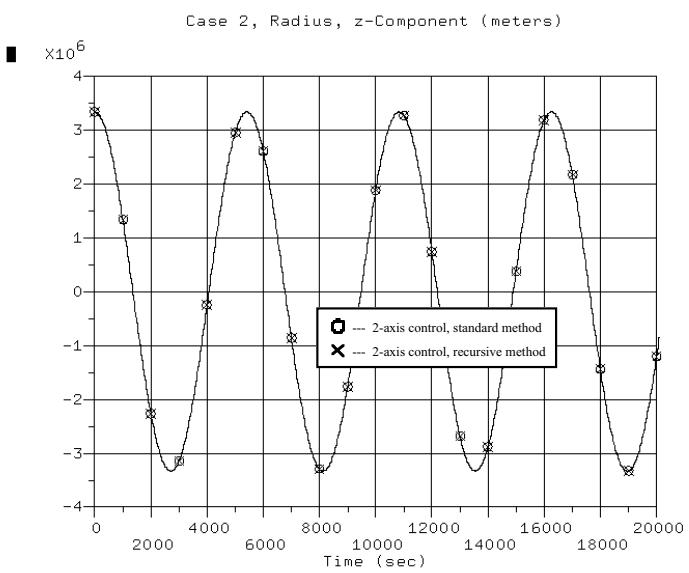
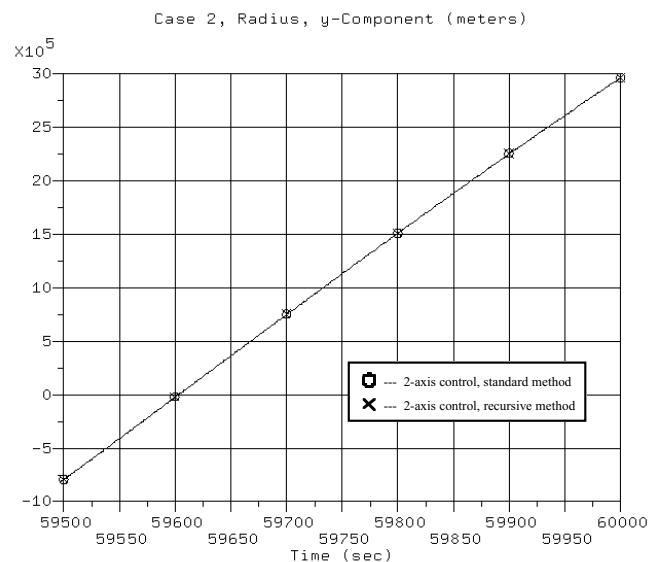
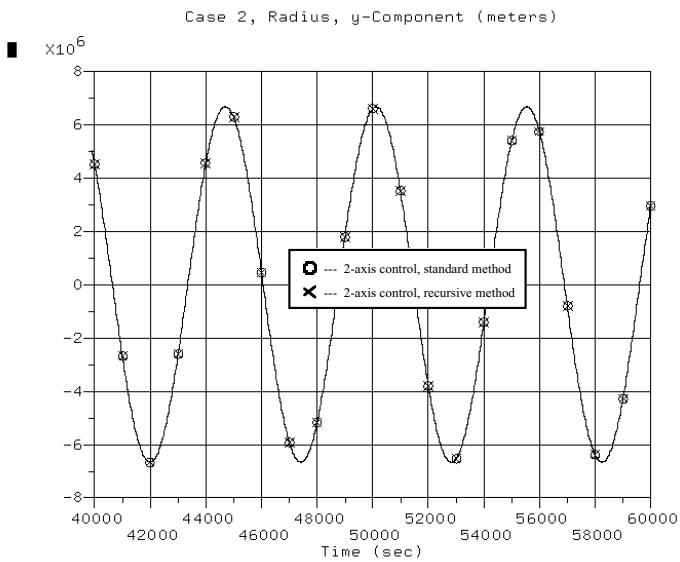


Case 1, Aerodynamic Torque, z-Component (newton-meters)

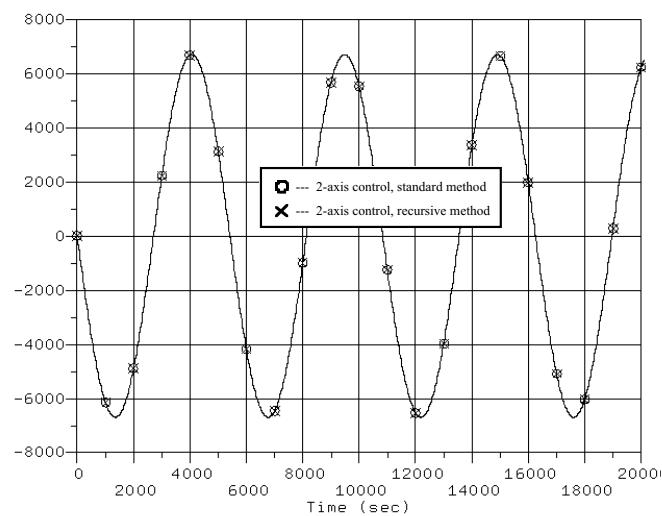


A.2 Response Plots for Case 2 (Two-Axis Attitude Control)

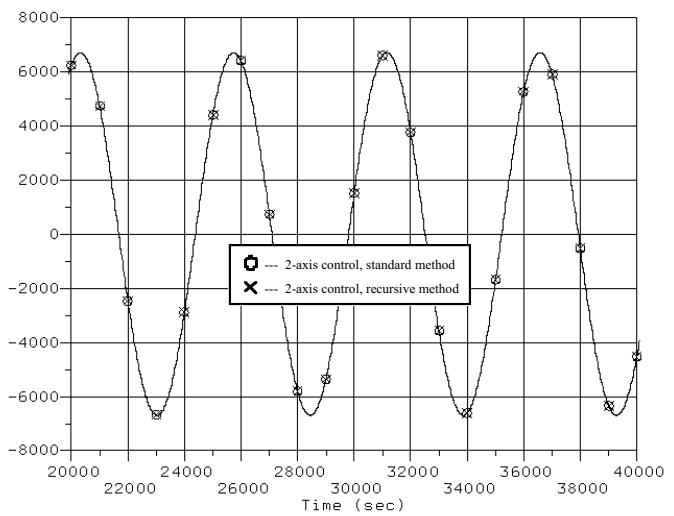




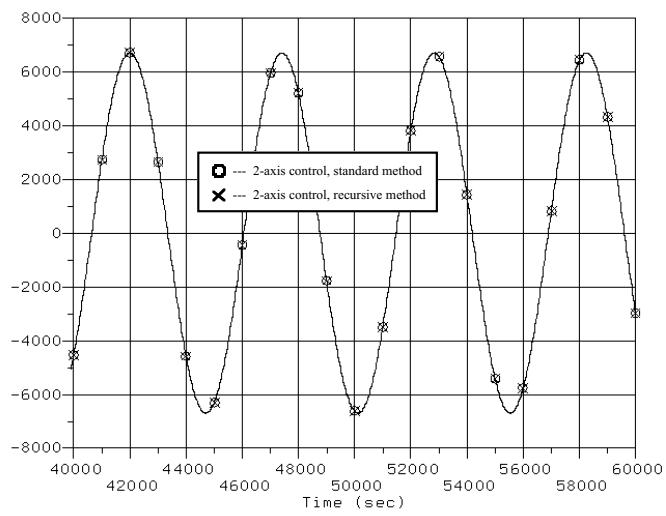
Case 2, Velocity, x-Component (meters/sec)



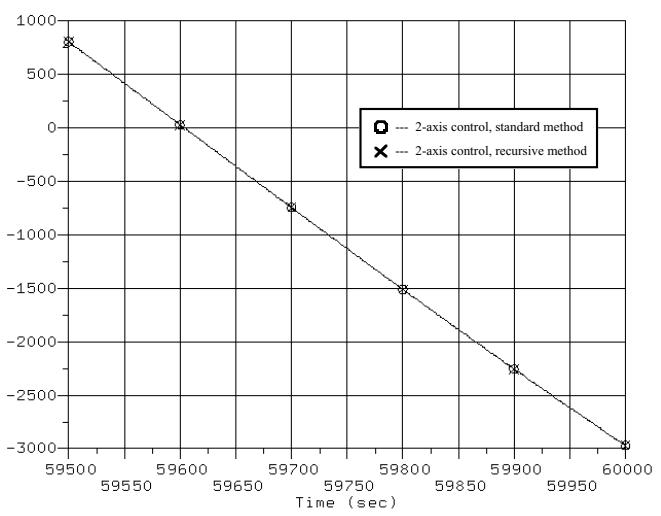
Case 2, Velocity, x-Component (meters/sec)



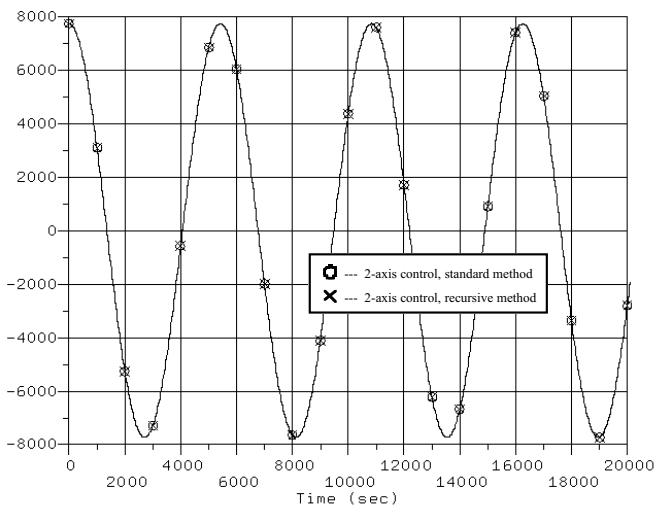
Case 2, Velocity, x-Component (meters/sec)



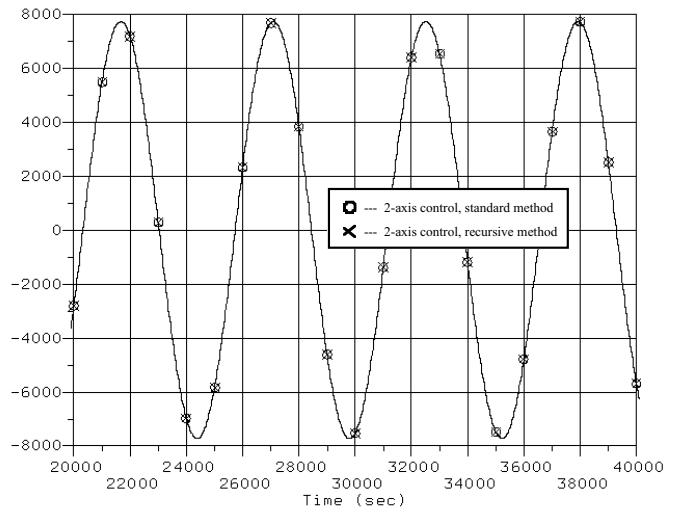
Case 2, Velocity, x-Component (meters/sec)



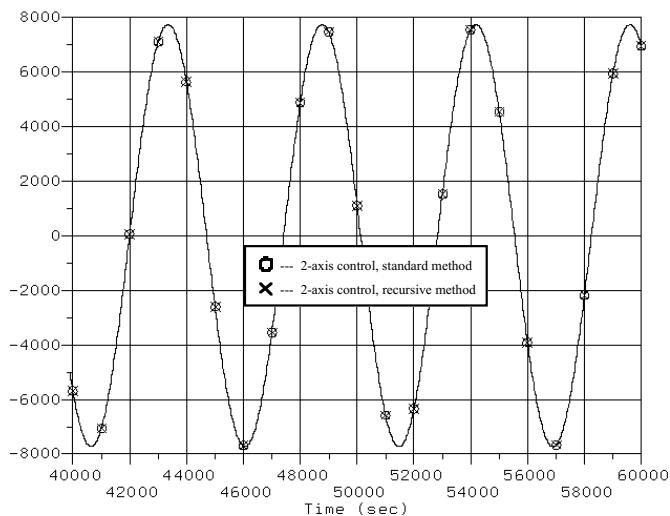
Case 2, Velocity, y-Component (meters/sec)



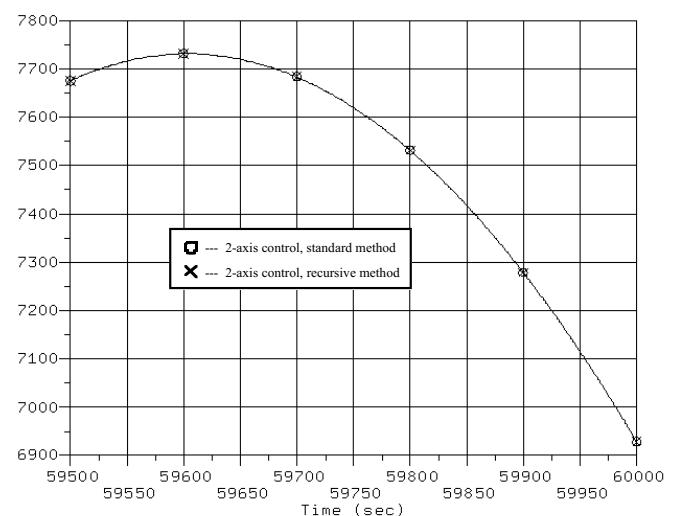
Case 2, Velocity, y-Component (meters/sec)



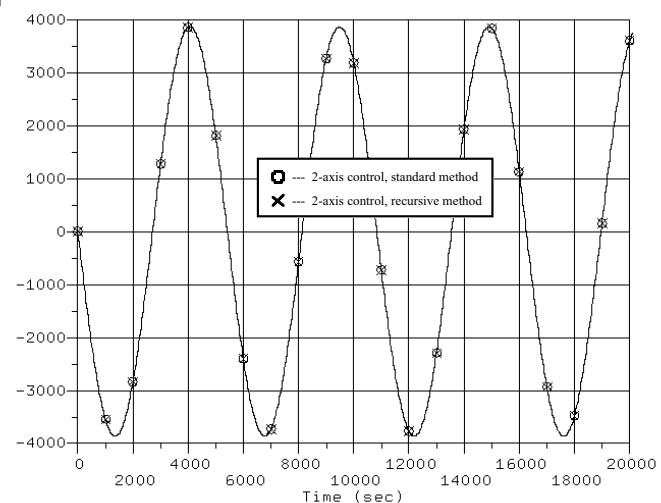
Case 2, Velocity, y-Component (meters/sec)



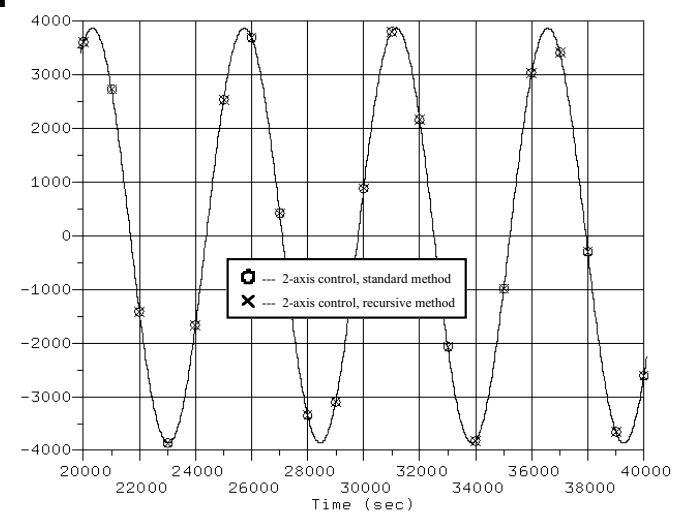
Case 2, Velocity, y-Component (meters/sec)



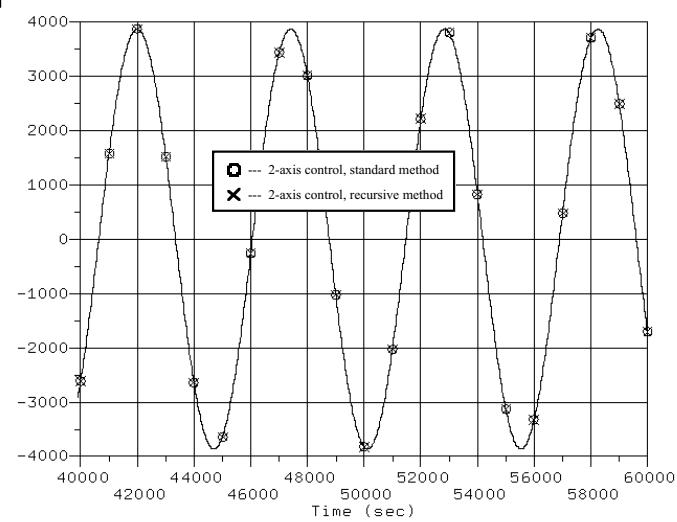
Case 2, Velocity, z-Component (meters/sec)



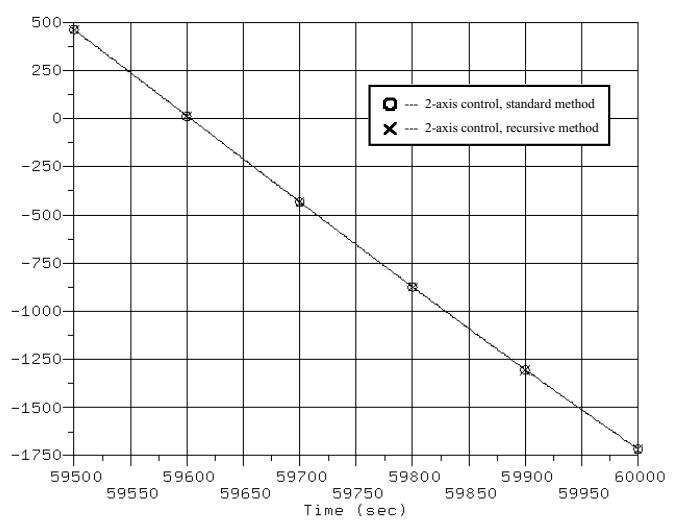
Case 2, Velocity, z-Component (meters/sec)



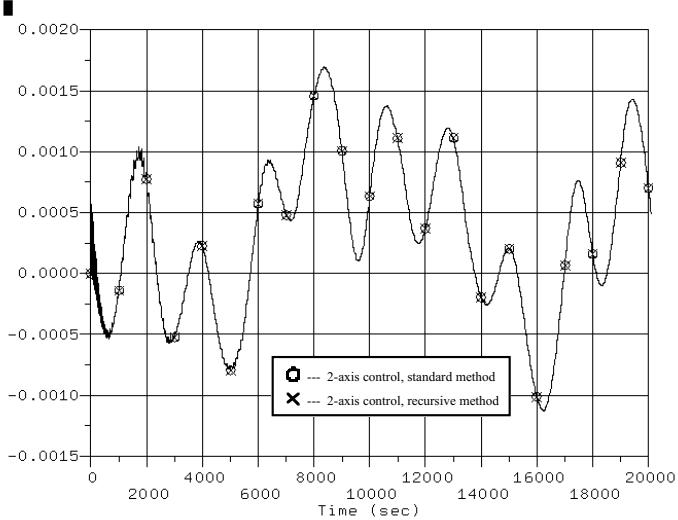
Case 2, Velocity, z-Component (meters/sec)



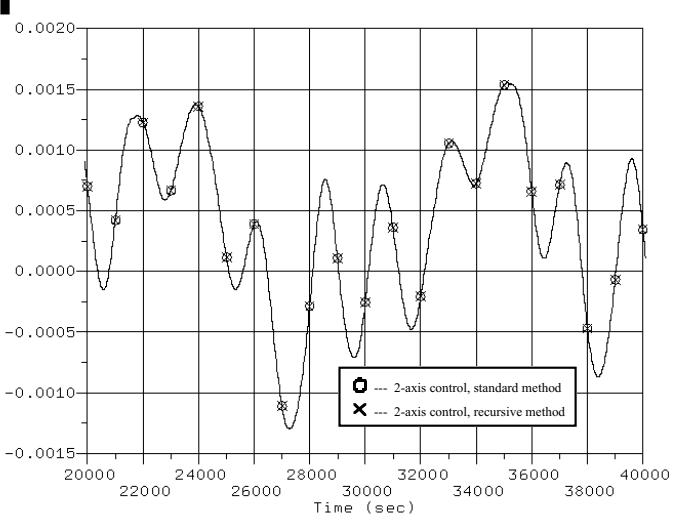
Case 2, Velocity, z-Component (meters/sec)



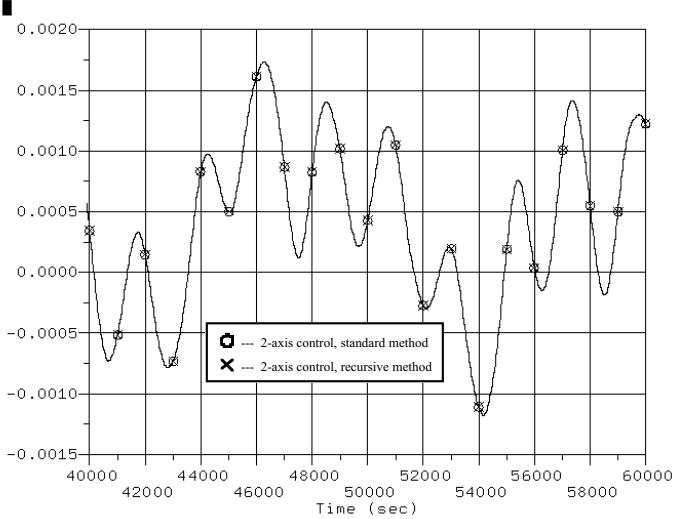
Case 2, Euler Angle, x-Axis (degrees)



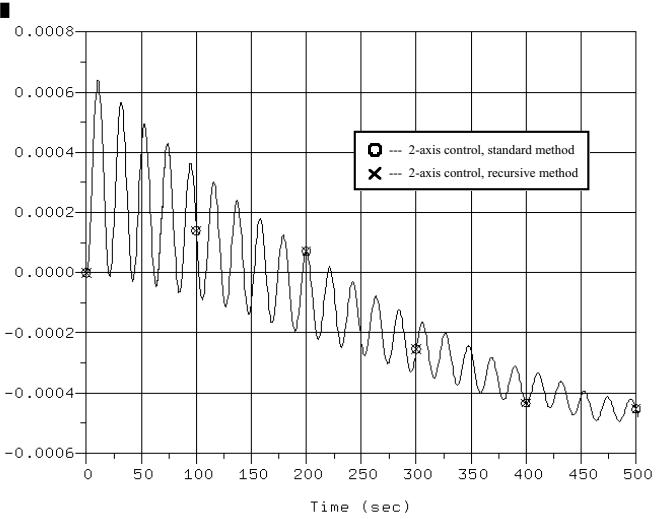
Case 2, Euler Angle, x-Axis (degrees)



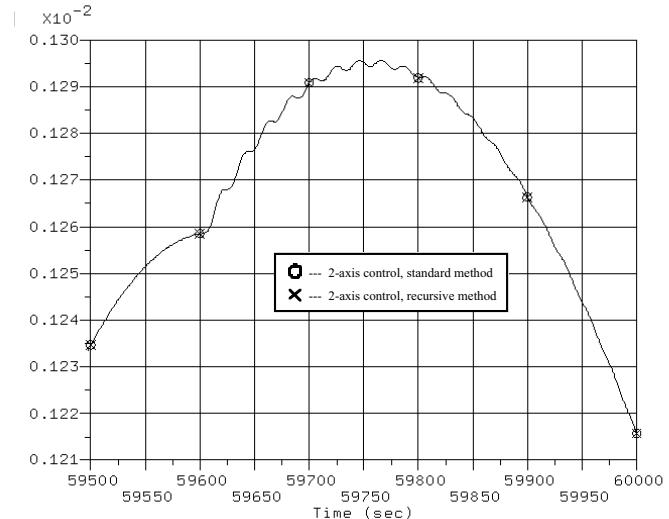
Case 2, Euler Angle, x-Axis (degrees)



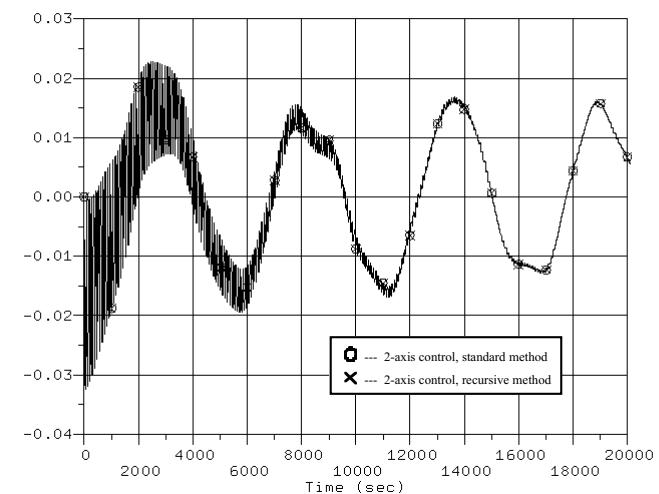
Case 2, Euler Angle, x-Axis (degrees)



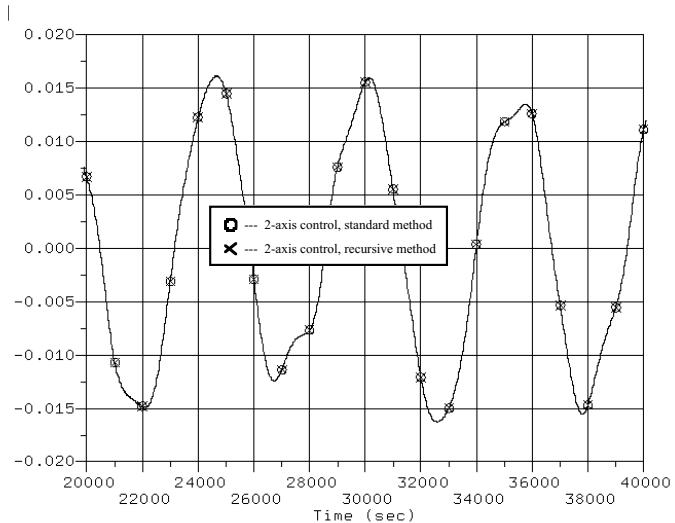
Case 2, Euler Angle, x-Axis (degrees)



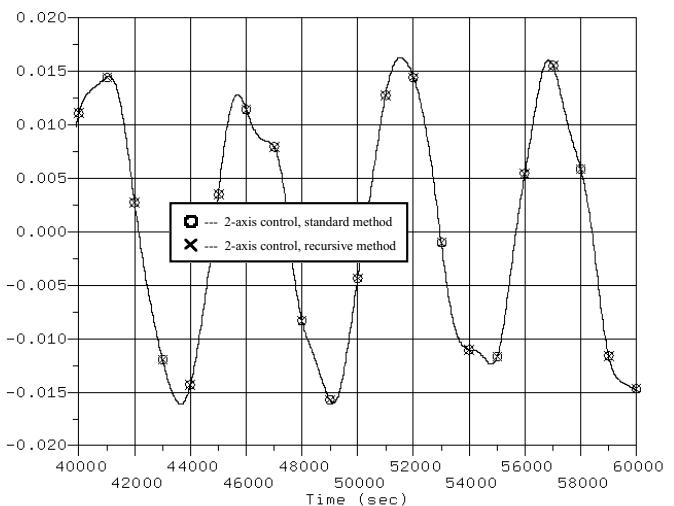
Case 2, Euler Angle, y-Axis (degrees)



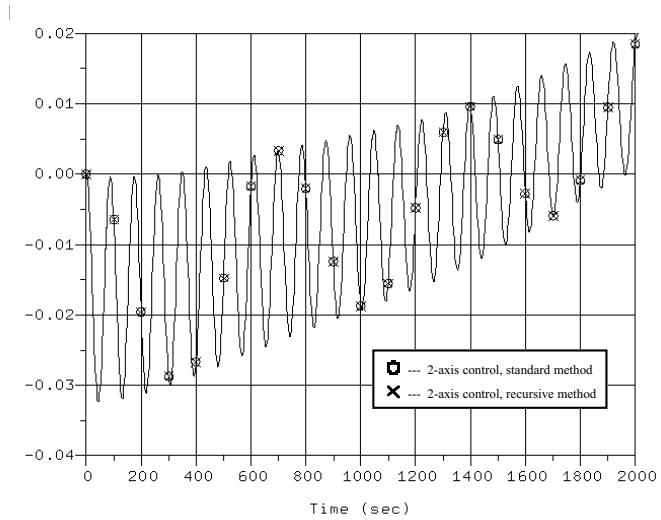
Case 2, Euler Angle, y-Axis (degrees)



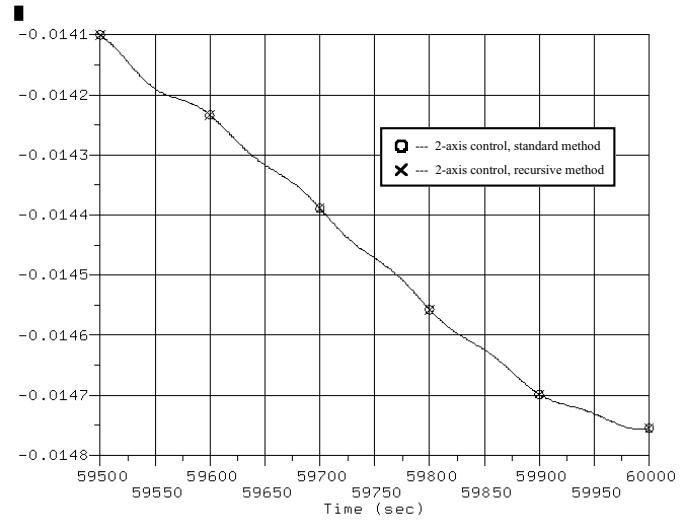
Case 2, Euler Angle, y-Axis (degrees)



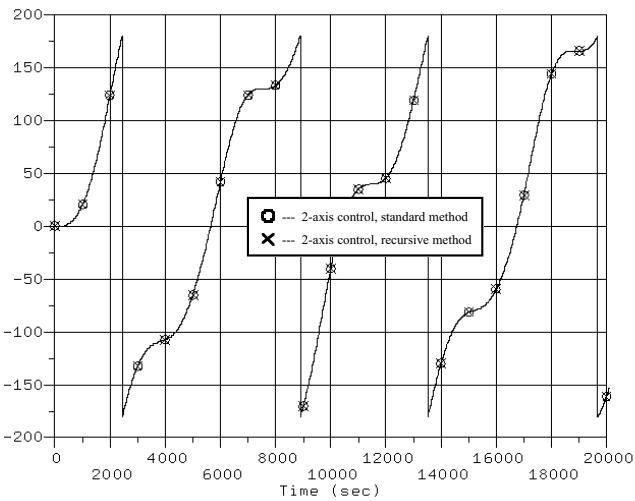
Case 2, Euler Angle, y-Axis (degrees)



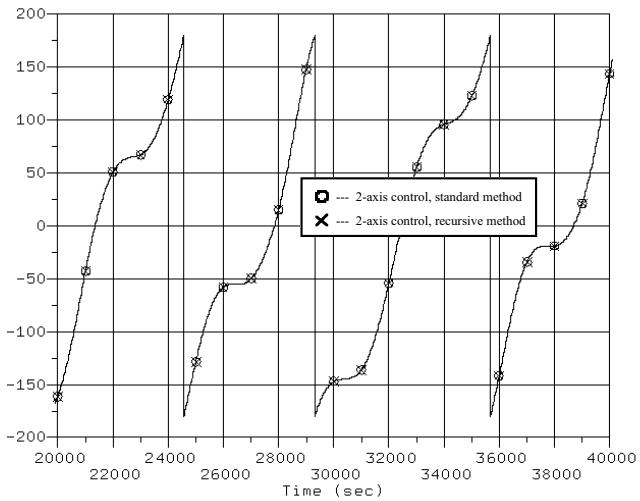
Case 2, Euler Angle, y-Axis (degrees)



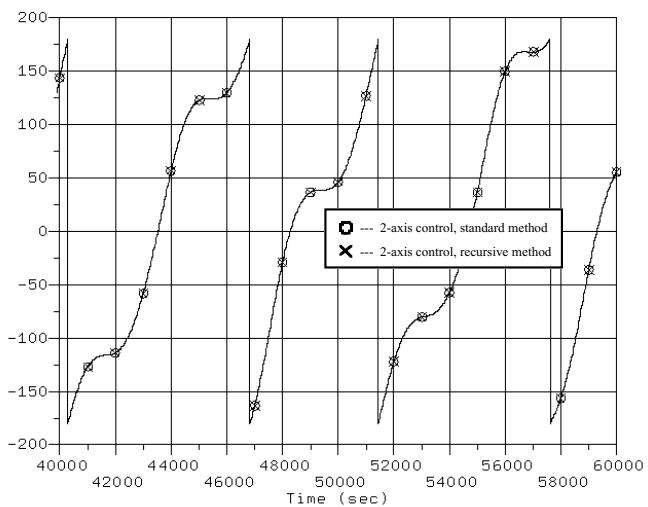
Case 2, Euler Angle, z-Axis (degrees)



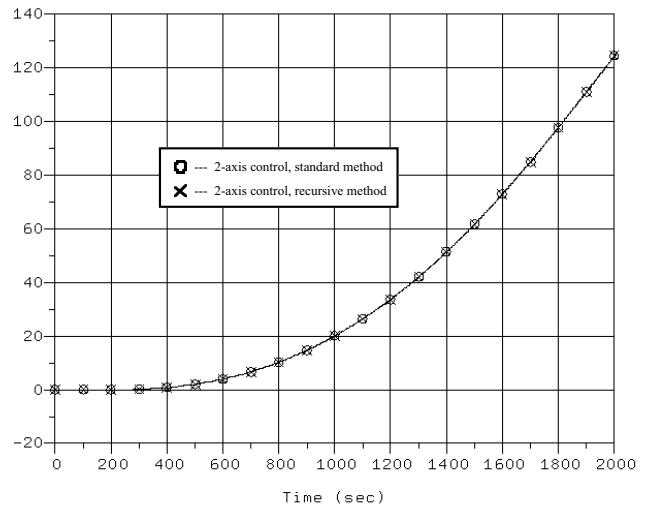
Case 2, Euler Angle, z-Axis (degrees)



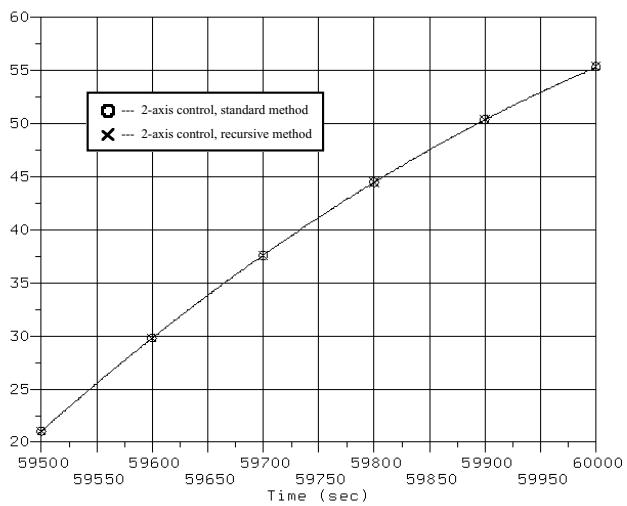
Case 2, Euler Angle, z-Axis (degrees)



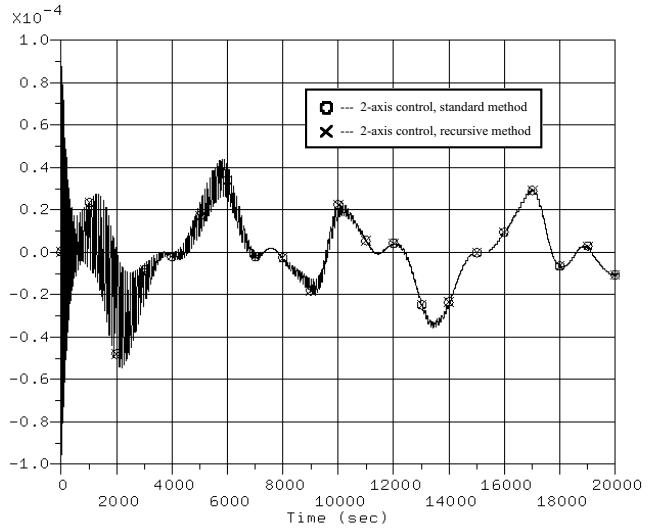
Case 2, Euler Angle, z-Axis (degrees)



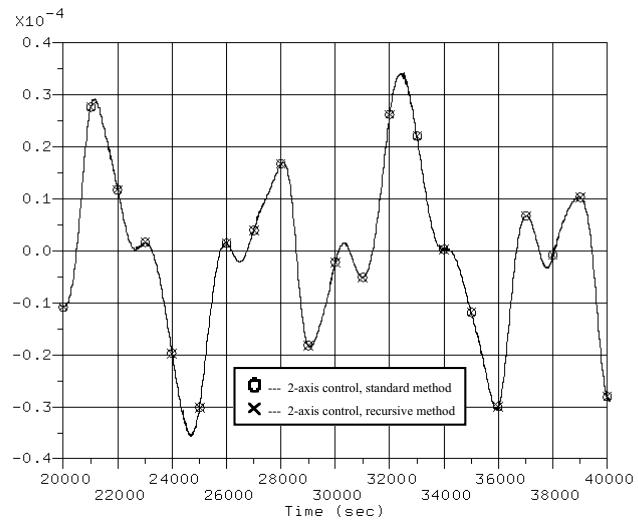
Case 2, Euler Angle, z-Axis (degrees)



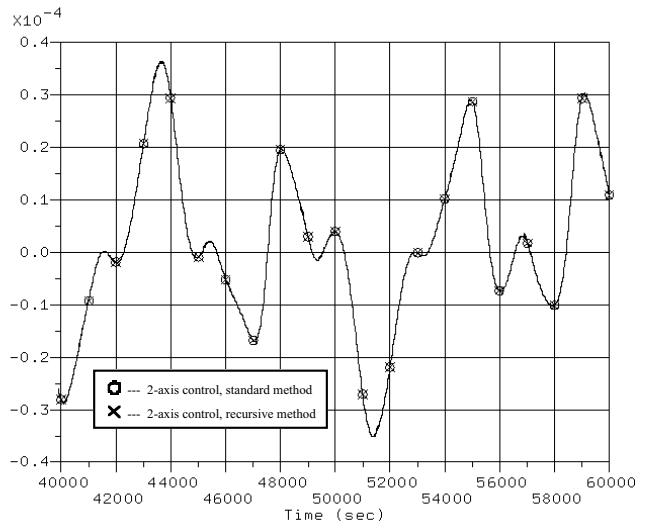
Case 2, Body Rate, x-Axis (deg/sec)

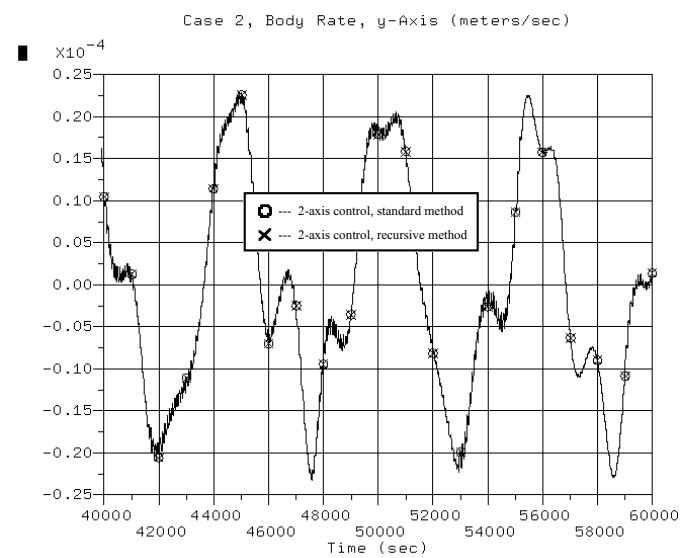
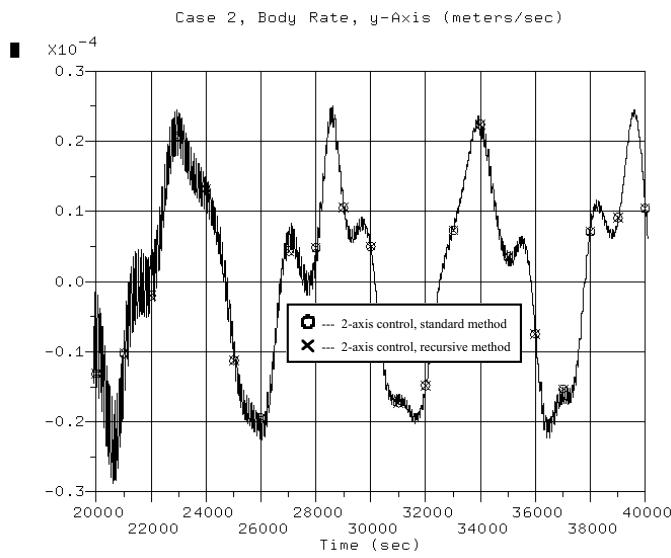
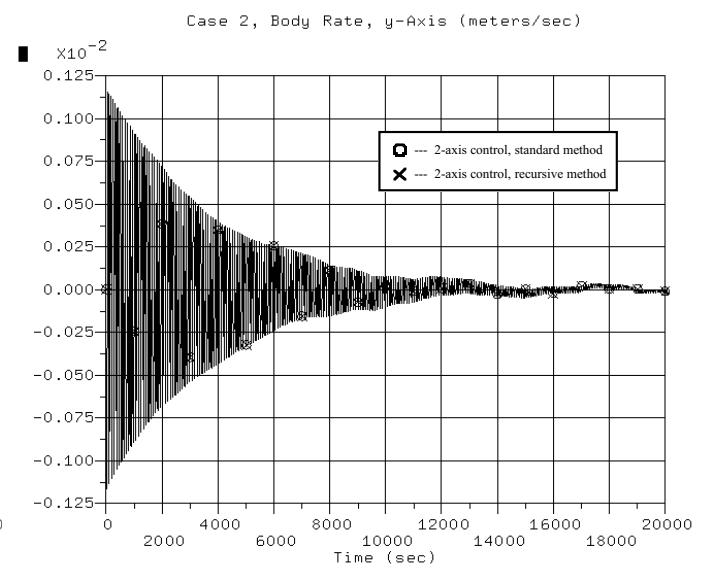
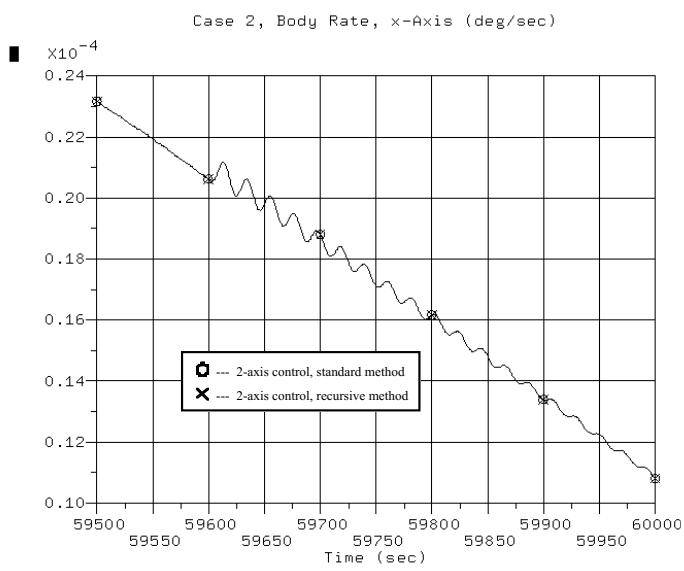
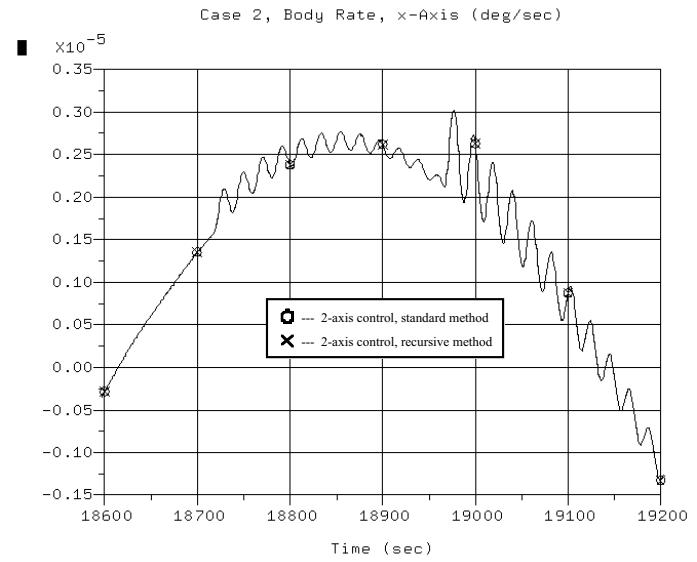
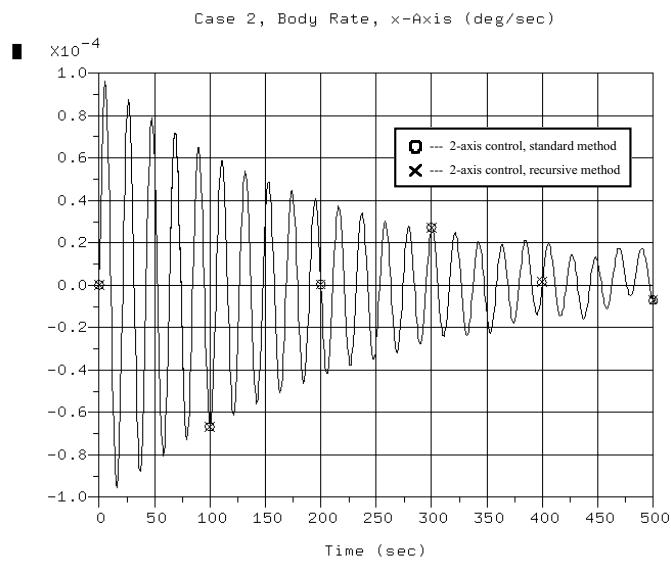


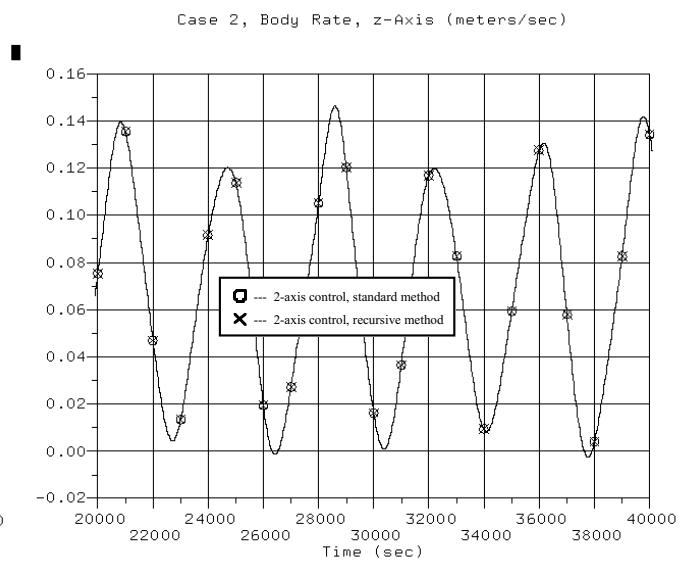
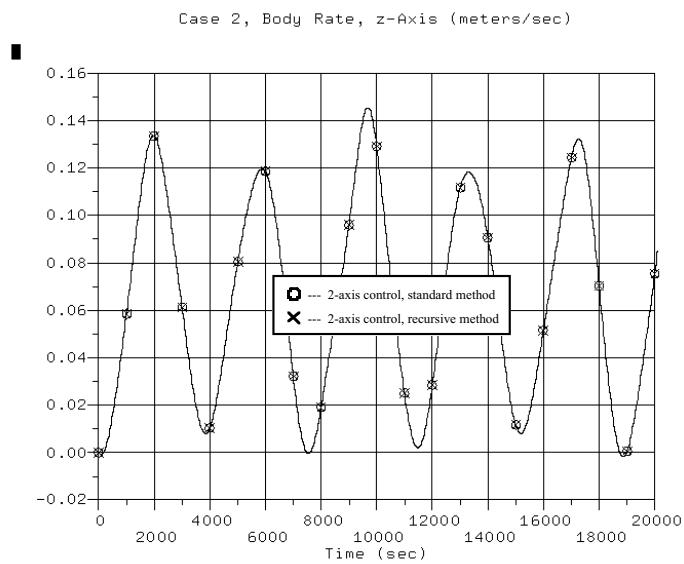
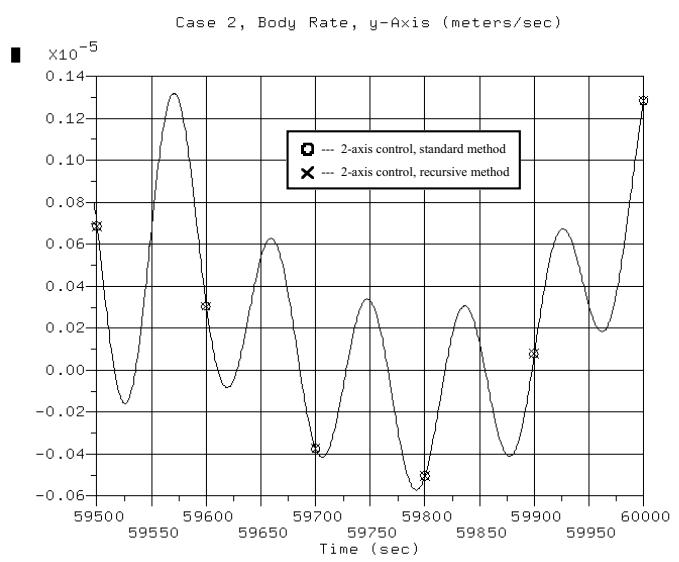
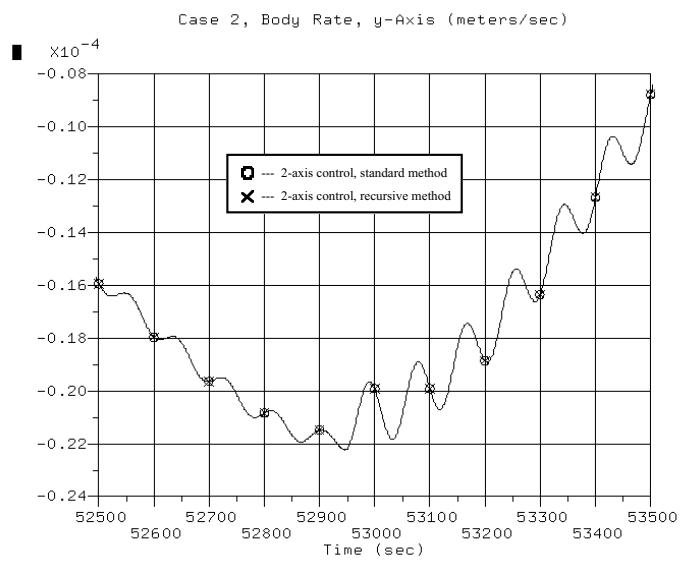
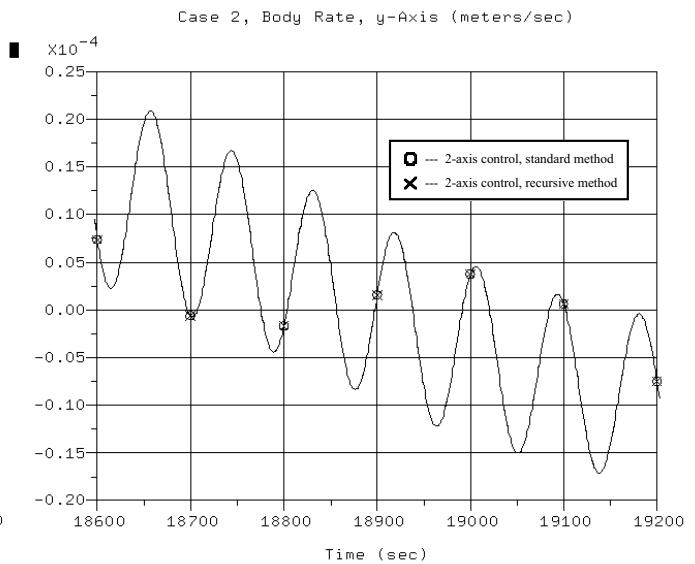
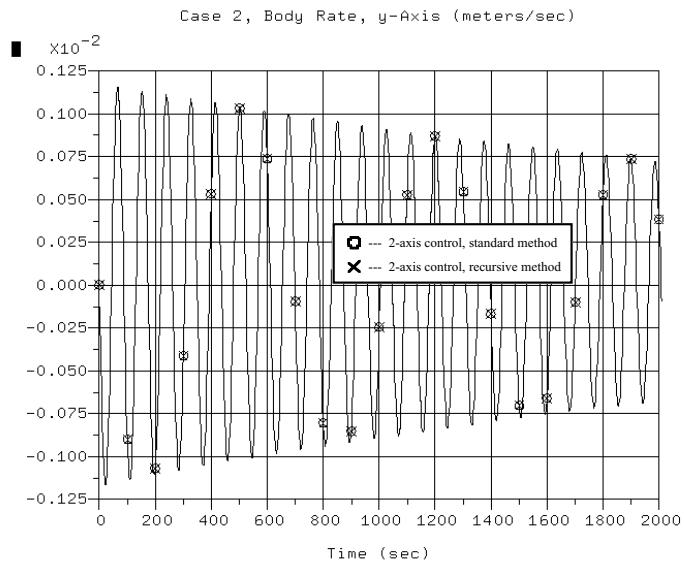
Case 2, Body Rate, x-Axis (deg/sec)



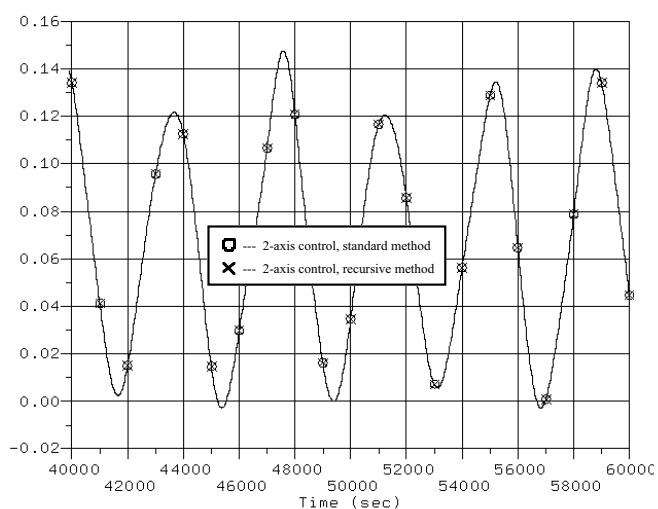
Case 2, Body Rate, x-Axis (deg/sec)



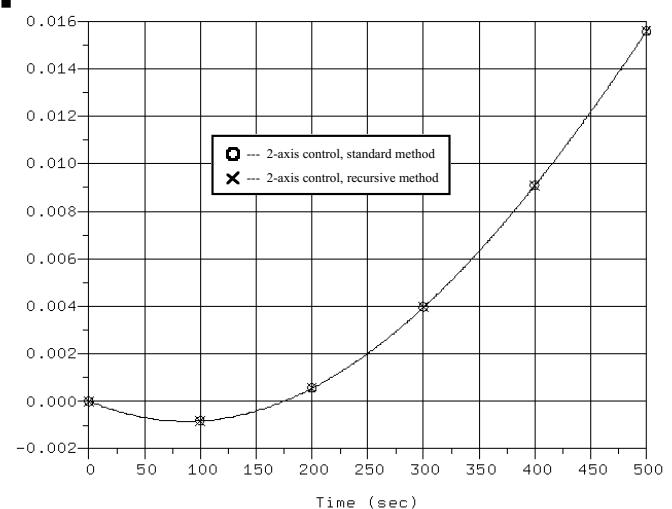




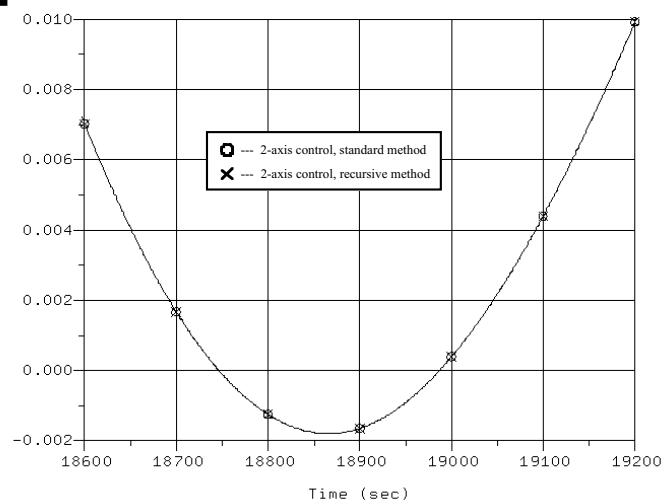
Case 2, Body Rate, z-Axis (meters/sec)



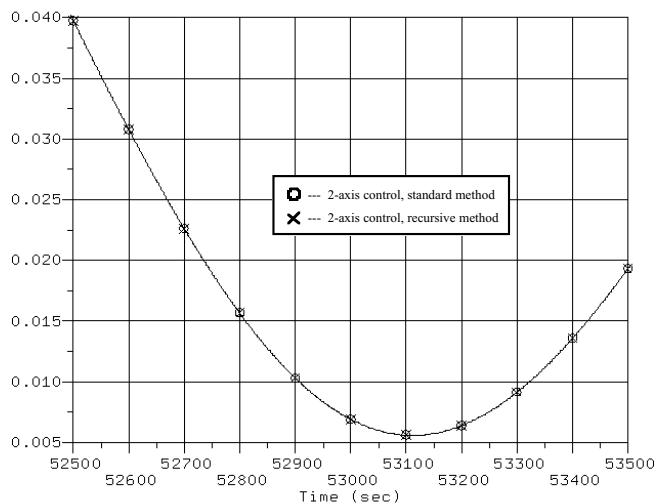
Case 2, Body Rate, z-Axis (meters/sec)



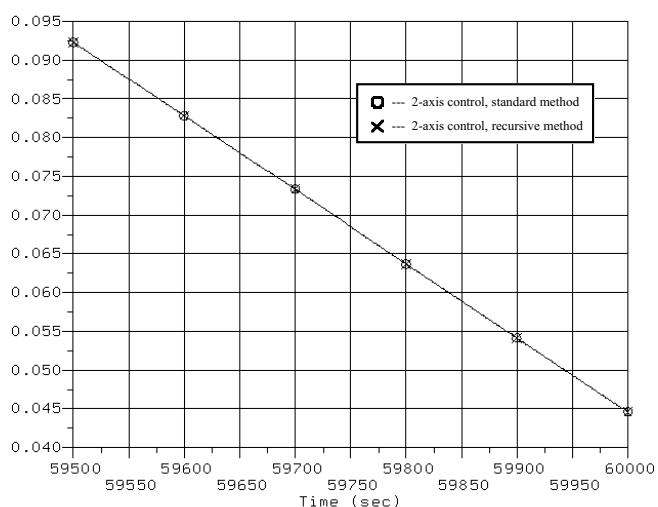
Case 2, Body Rate, x-Axis (meters/sec)



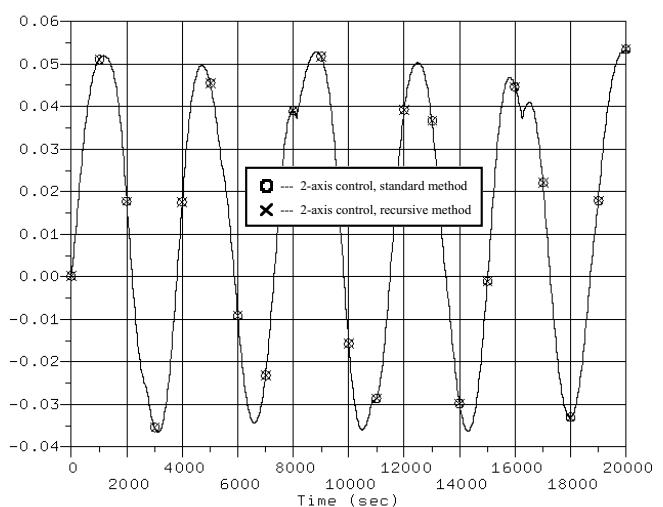
Case 2, Body Rate, z-Axis (meters/sec)



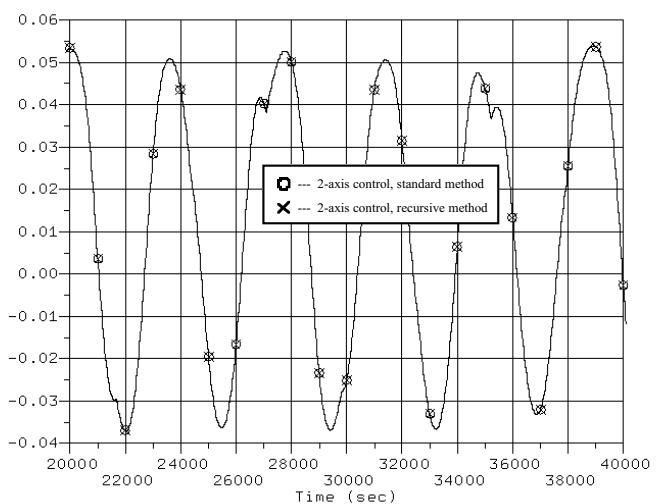
Case 2, Body Rate, z-Axis (meters/sec)



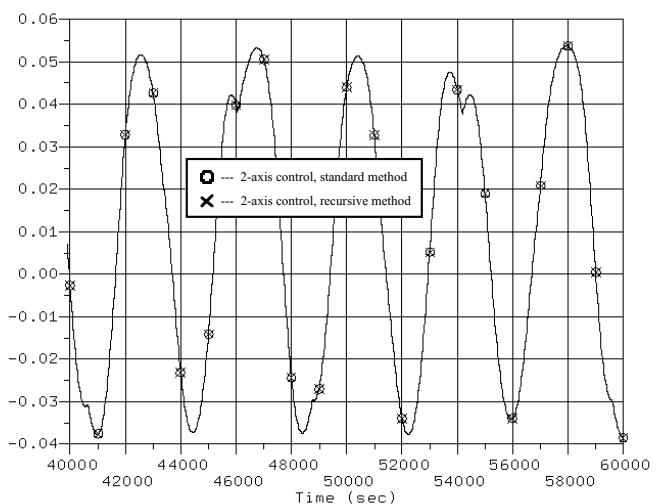
Case 2, Aerodynamic Force, x-Component (newtons)



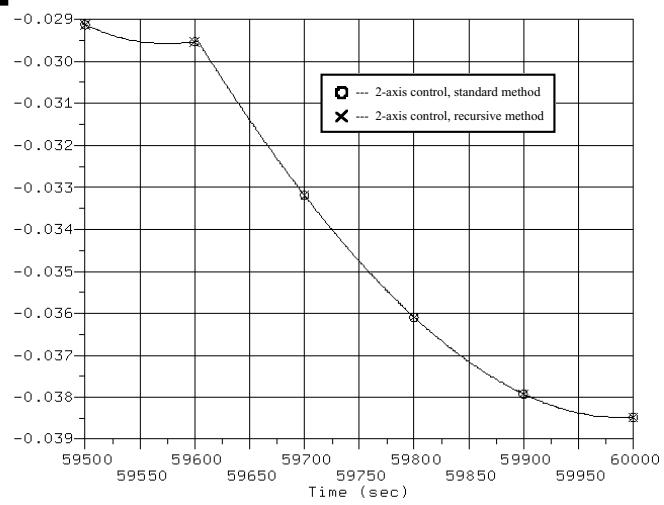
Case 2, Aerodynamic Force, x-Component (newtons)



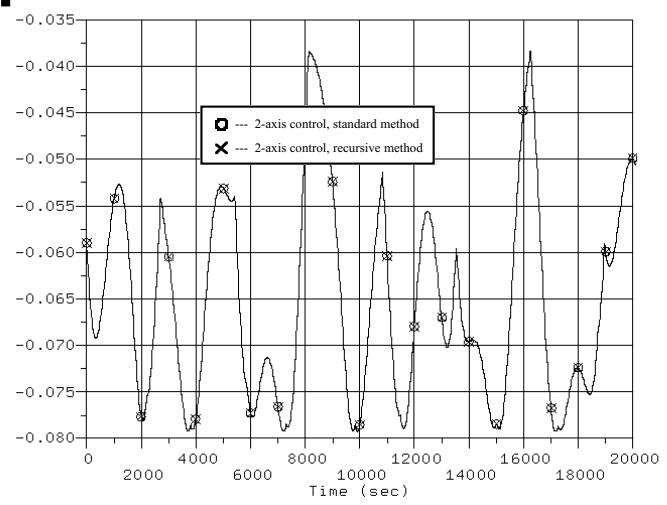
Case 2, Aerodynamic Force, x-Component (newtons)



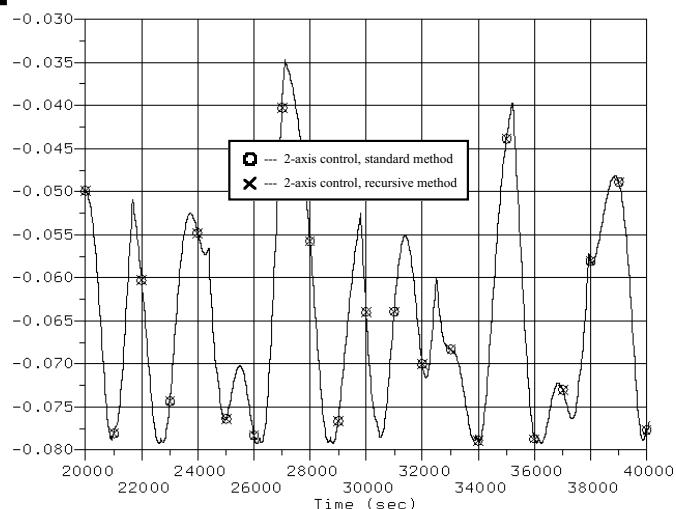
Case 2, Aerodynamic Force, x-Component (newtons)



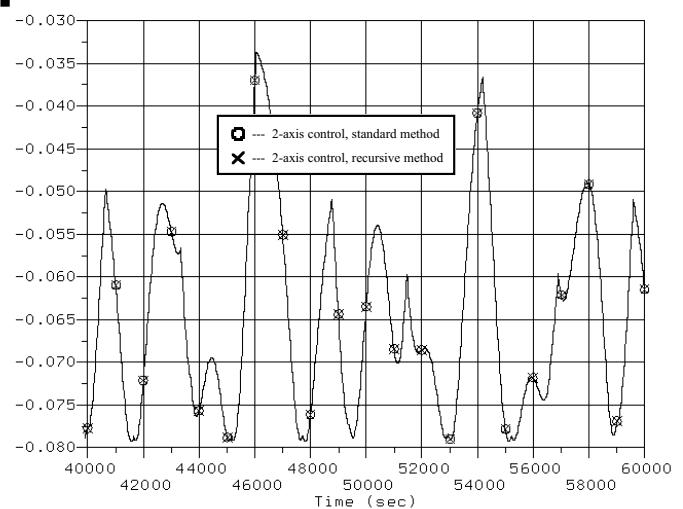
Case 2, Aerodynamic Force, y-Component (newtons)



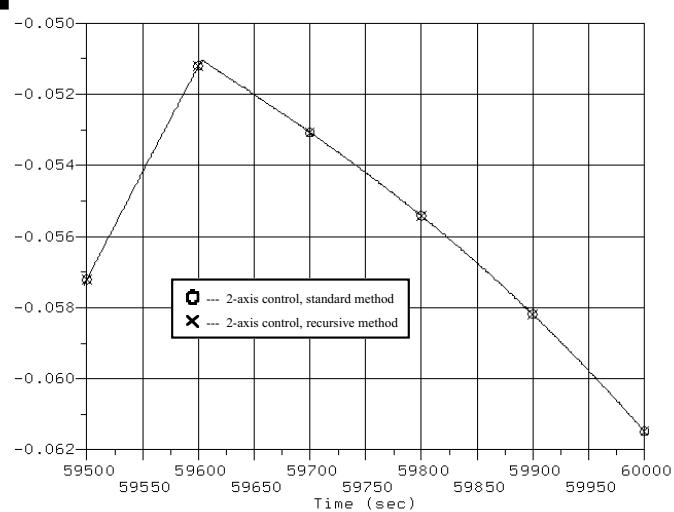
Case 2, Aerodynamic Force, y-Component (newtons)



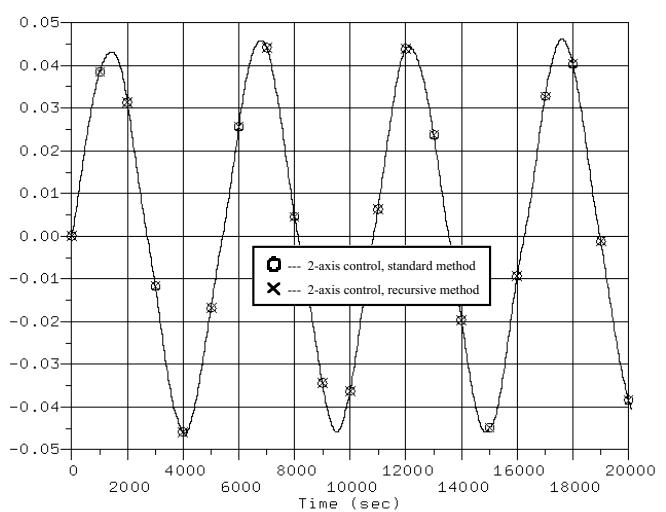
Case 2, Aerodynamic Force, y-Component (newtons)



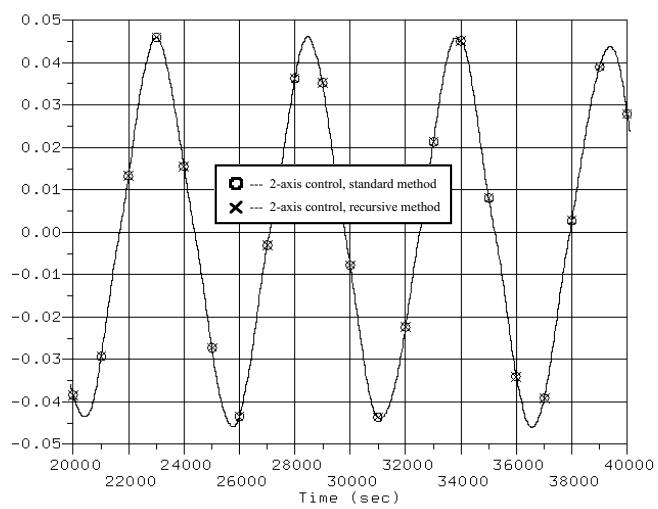
Case 2, Aerodynamic Force, y-Component (newtons)



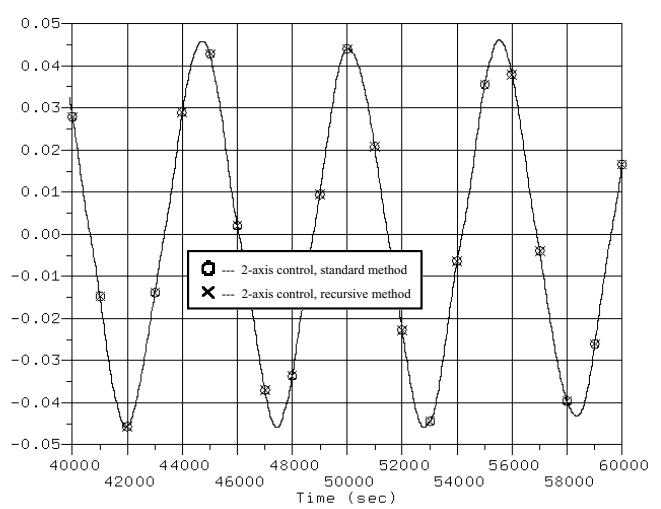
Case 2, Aerodynamic Force, z-Component (newtons)



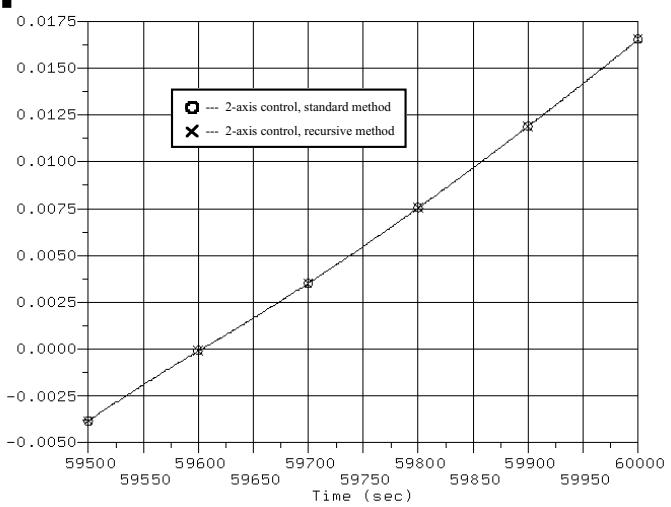
Case 2, Aerodynamic Force, z-Component (newtons)



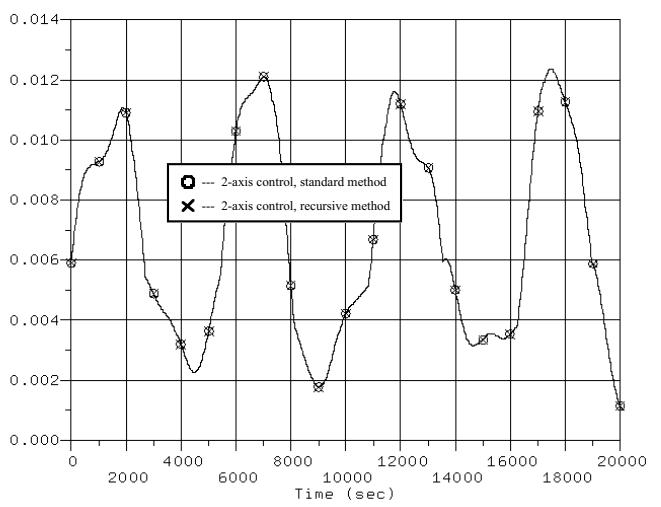
Case 2, Aerodynamic Force, z-Component (newtons)



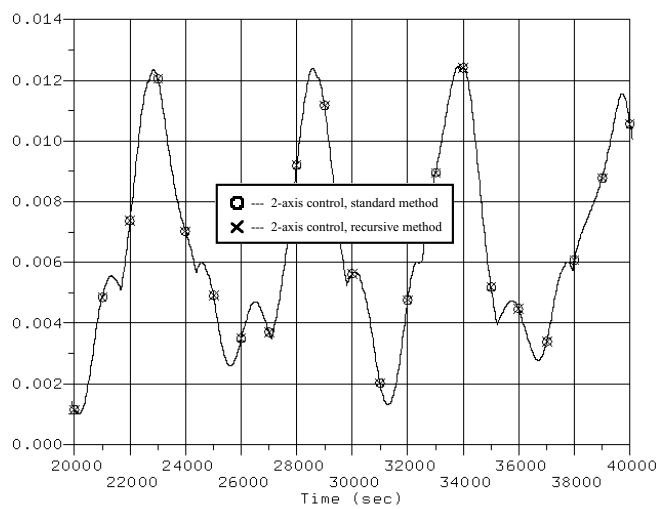
Case 2, Aerodynamic Force, z-Component (newtons)



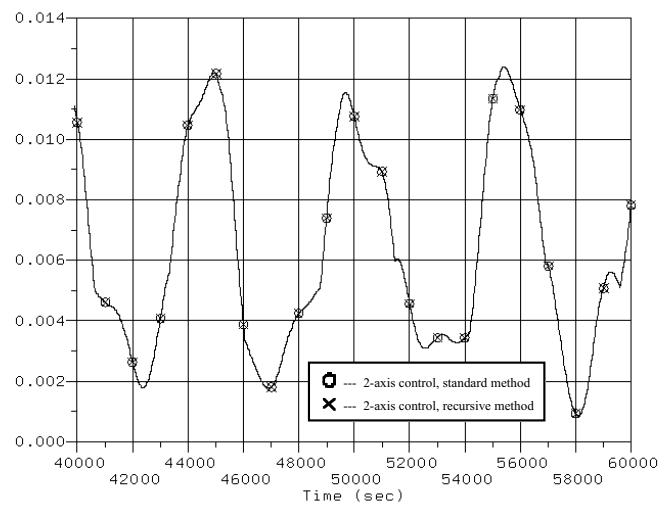
Case 2, Aerodynamic Torque, x-Component (newton-meters)



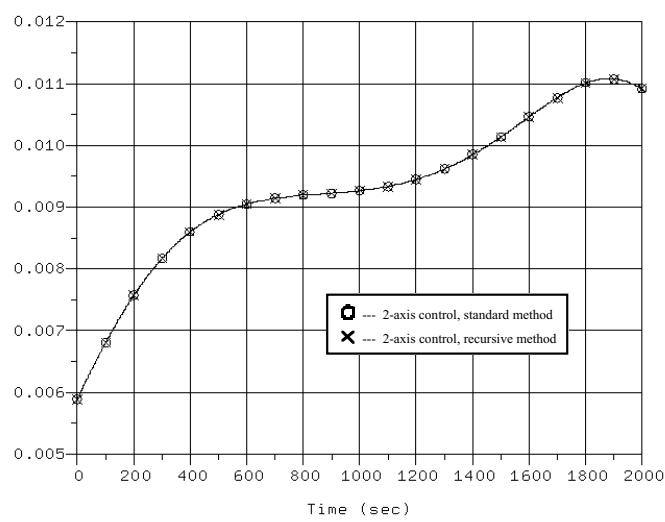
Case 2, Aerodynamic Torque, x-Component (newton-meters)



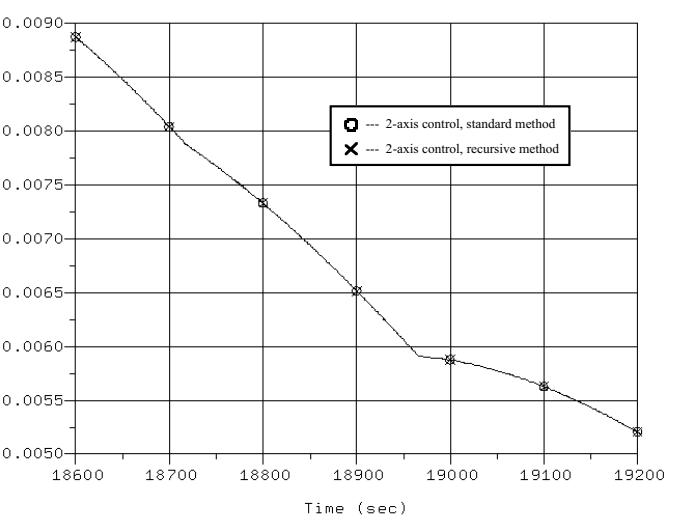
Case 2, Aerodynamic Torque, x-Component (newton-meters)



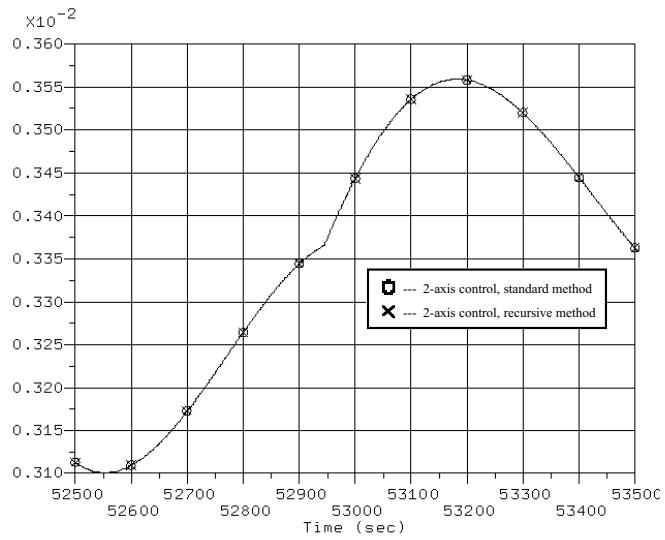
Case 2, Aerodynamic Torque, x-Component (newton-meters)



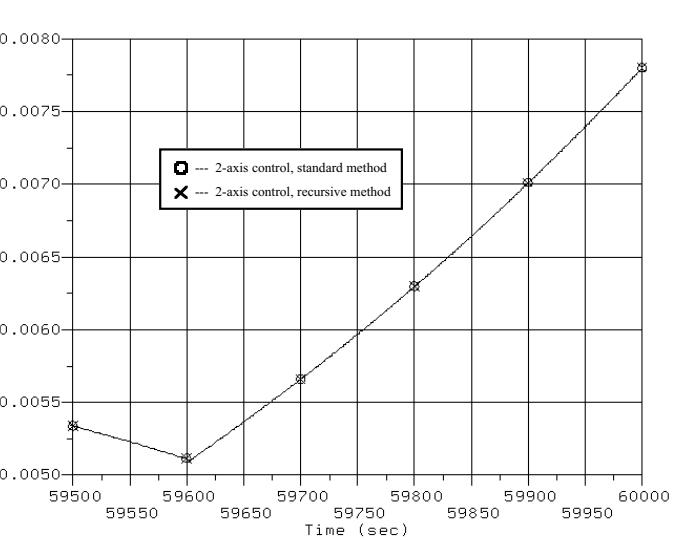
Case 2, Aerodynamic Torque, x-Component (newton-meters)



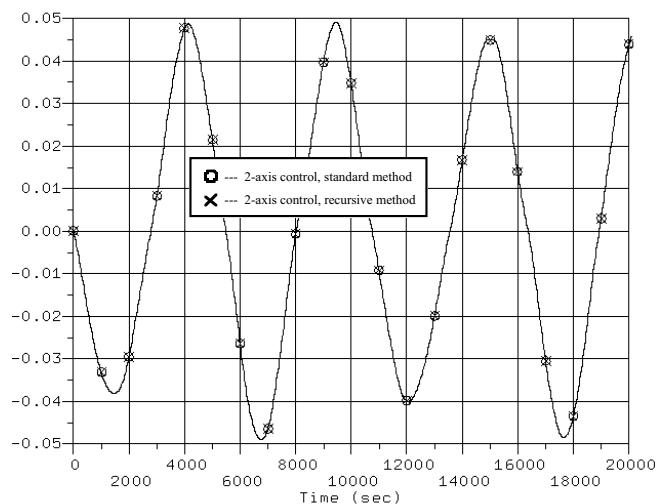
Case 2, Aerodynamic Torque, x-Component (newton-meters)



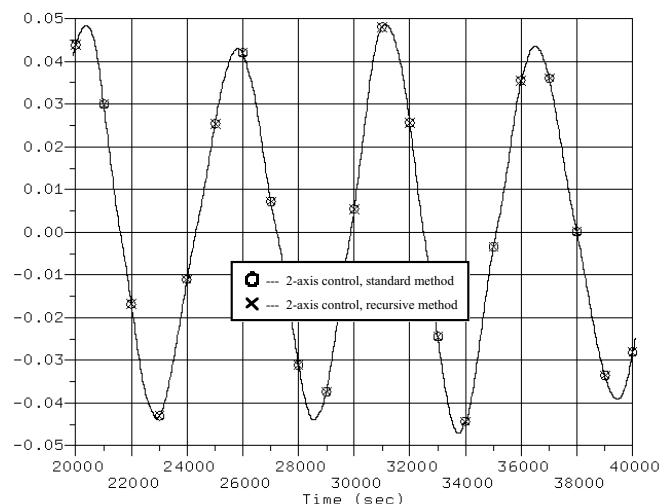
Case 2, Aerodynamic Torque, x-Component (newton-meters)



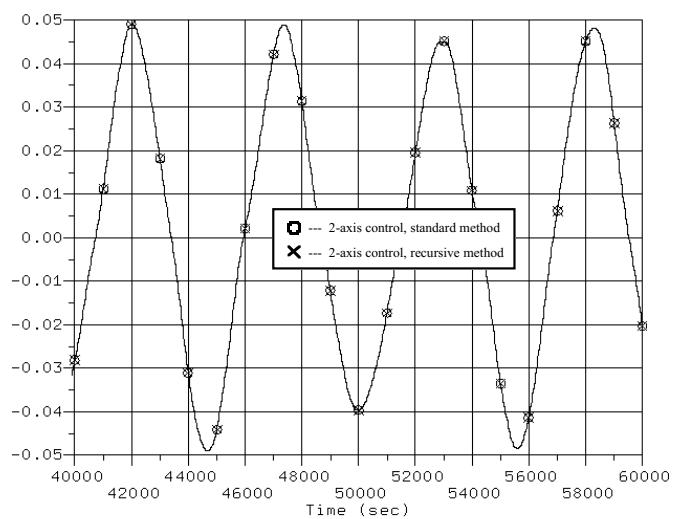
Case 2, Aerodynamic Torque, y-Component (newton meters)



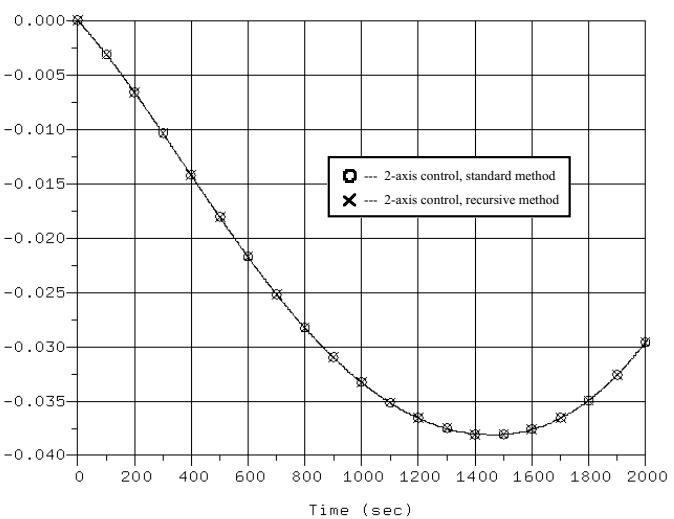
Case 2, Aerodynamic Torque, y-Component (newton meters)



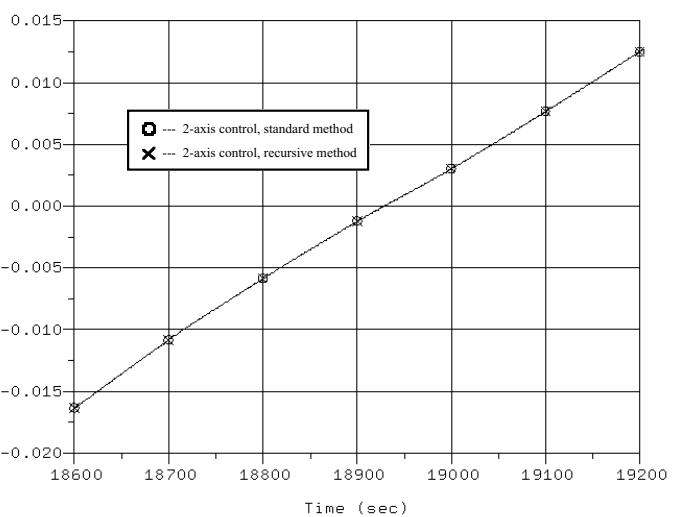
Case 2, Aerodynamic Torque, y-Component (newton meters)



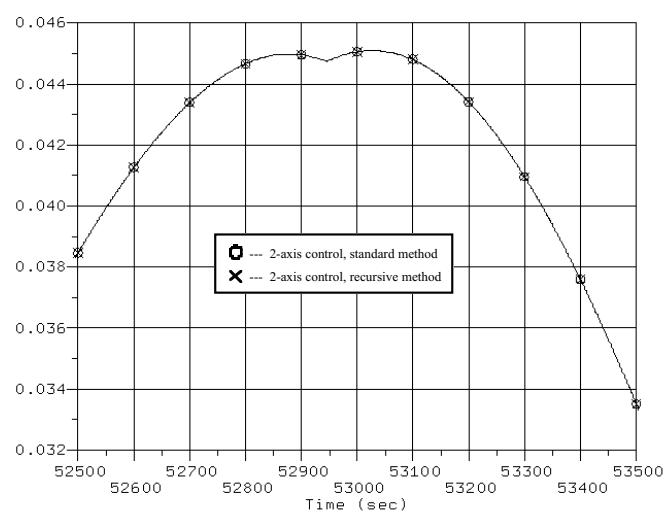
Case 2, Aerodynamic Torque, y-Component (newton meters)



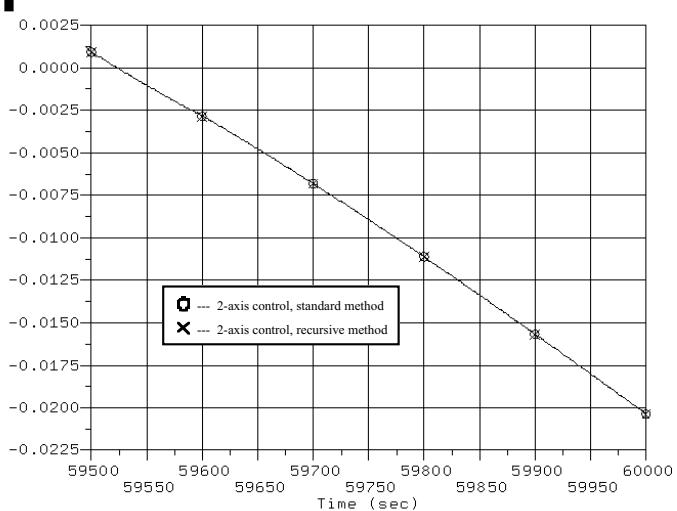
Case 2, Aerodynamic Torque, y-Component (newton meters)



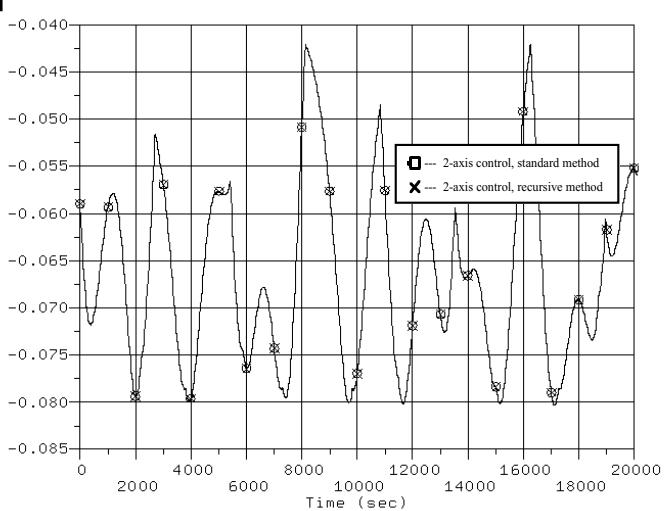
Case 2, Aerodynamic Torque, y-Component (newton meters)



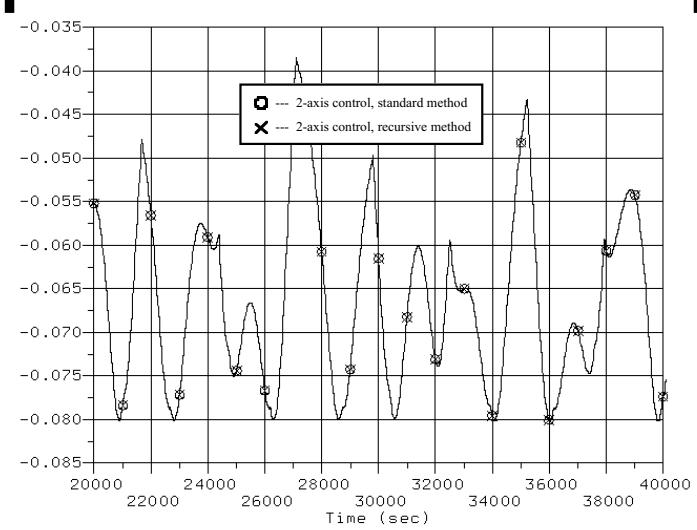
Case 2, Aerodynamic Torque, y-Component (newton meters)



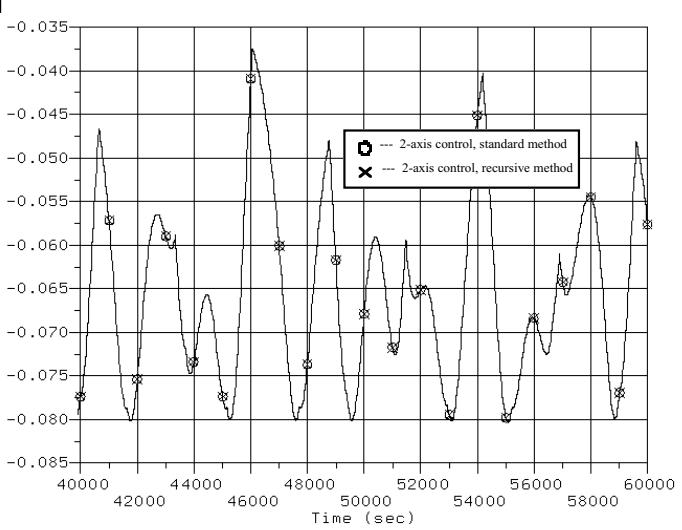
Case 2, Aerodynamic Torque, z-Component (newton-meters)



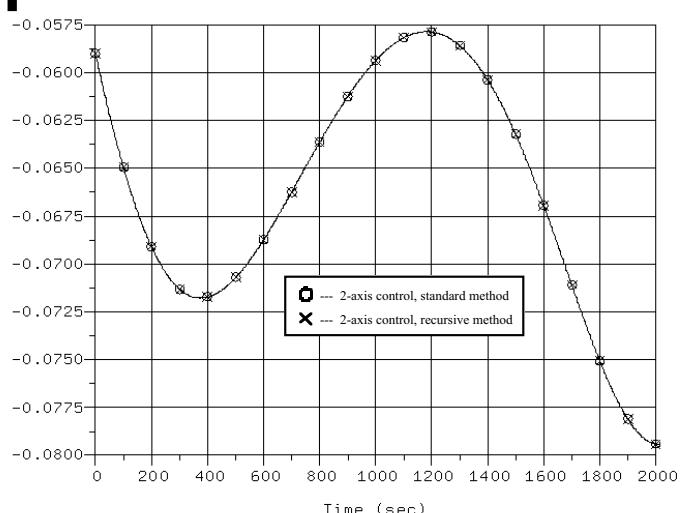
Case 2, Aerodynamic Torque, z-Component (newton-meters)



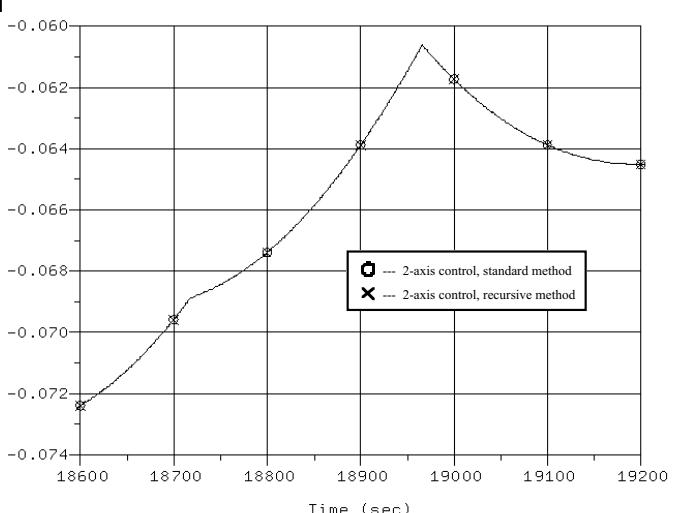
Case 2, Aerodynamic Torque, z-Component (newton-meters)



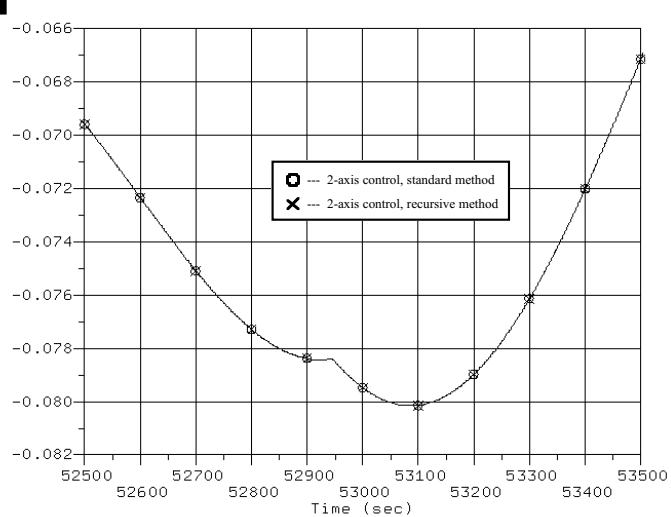
Case 2, Aerodynamic Torque, z-Component (newton-meters)



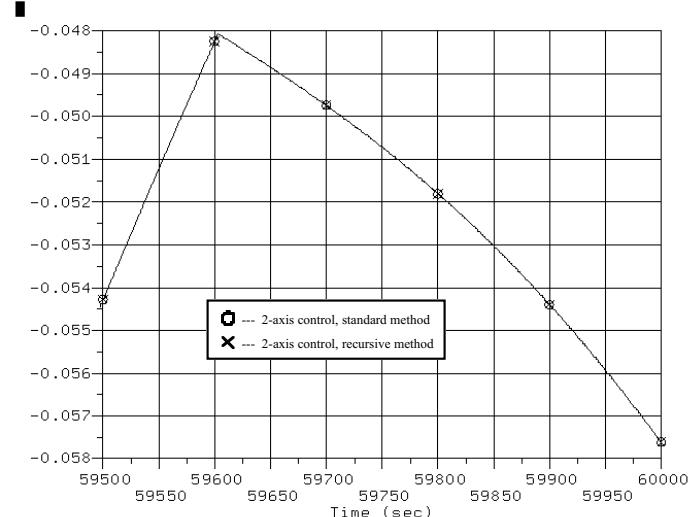
Case 2, Aerodynamic Torque, z-Component (newton-meters)



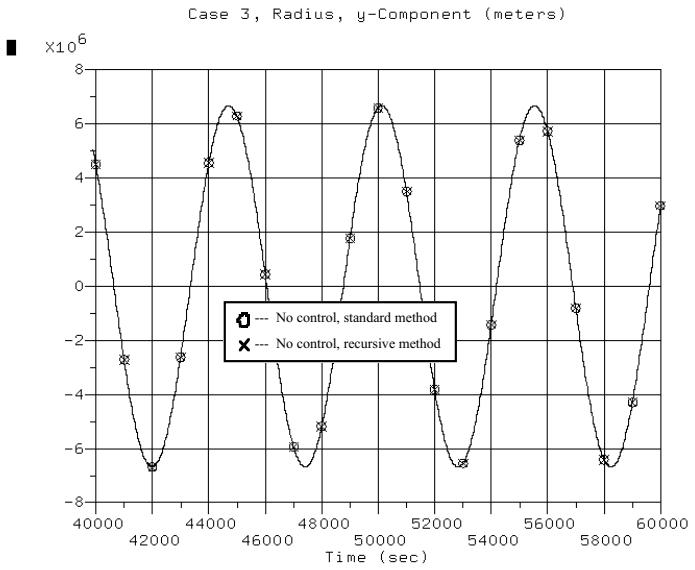
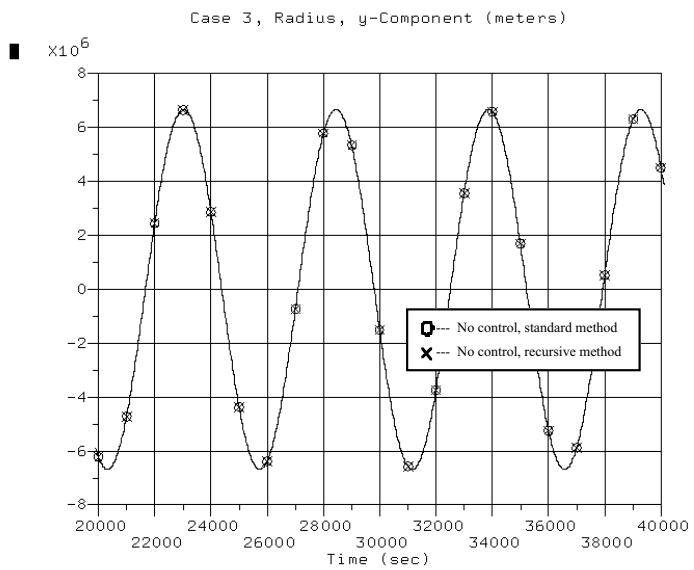
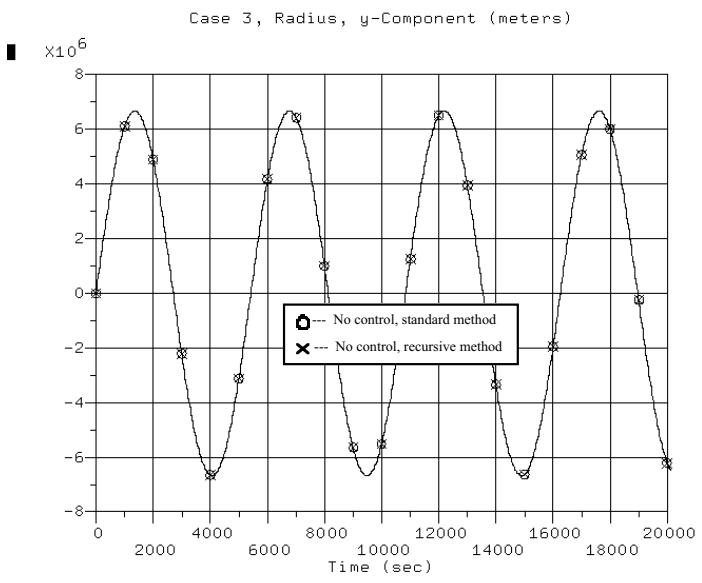
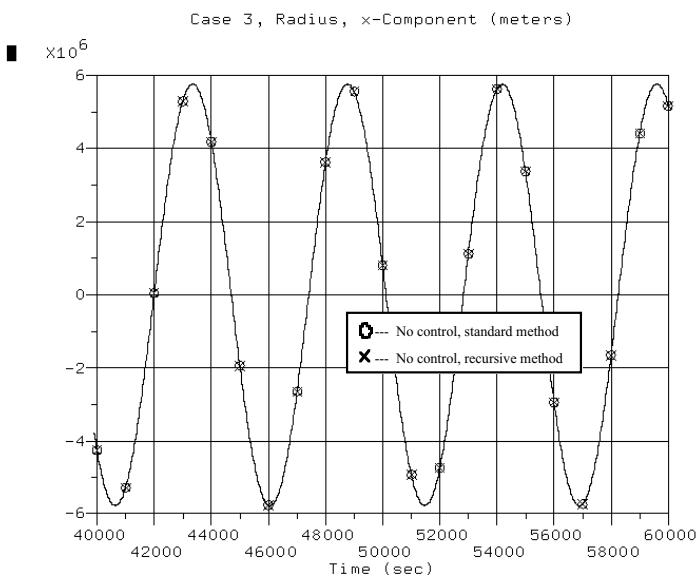
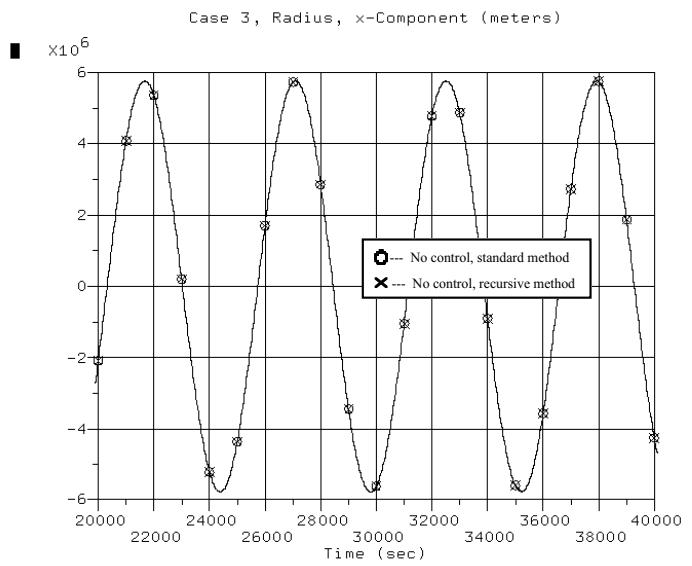
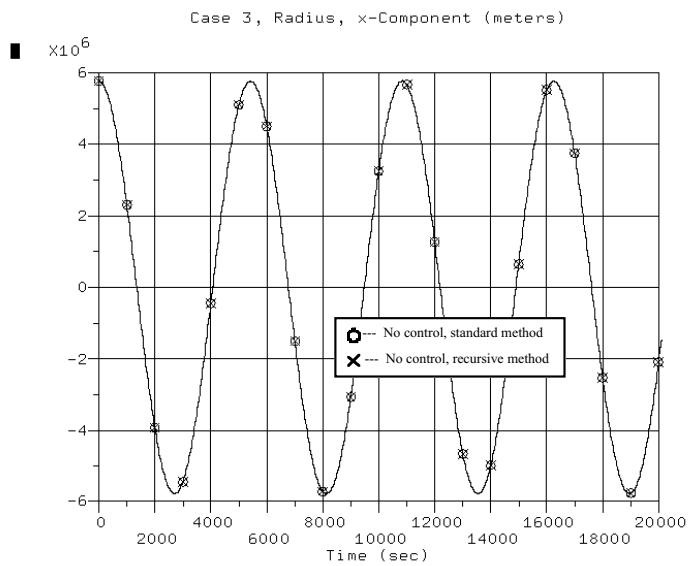
Case 2, Aerodynamic Torque, z-Component (newton-meters)

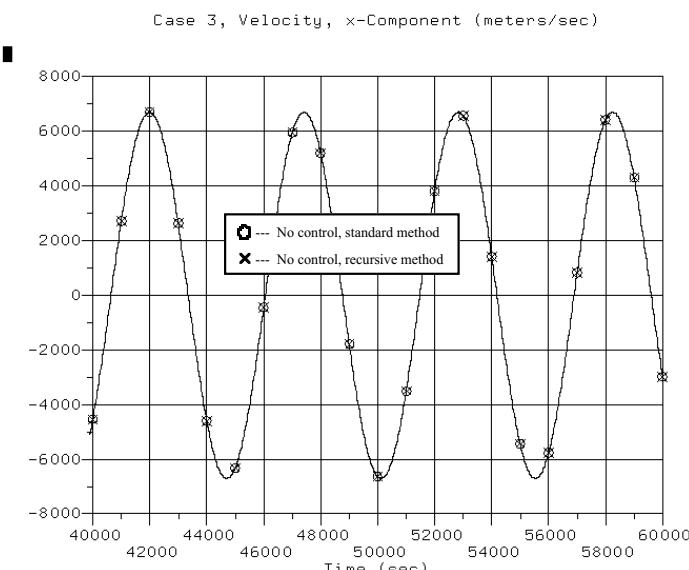
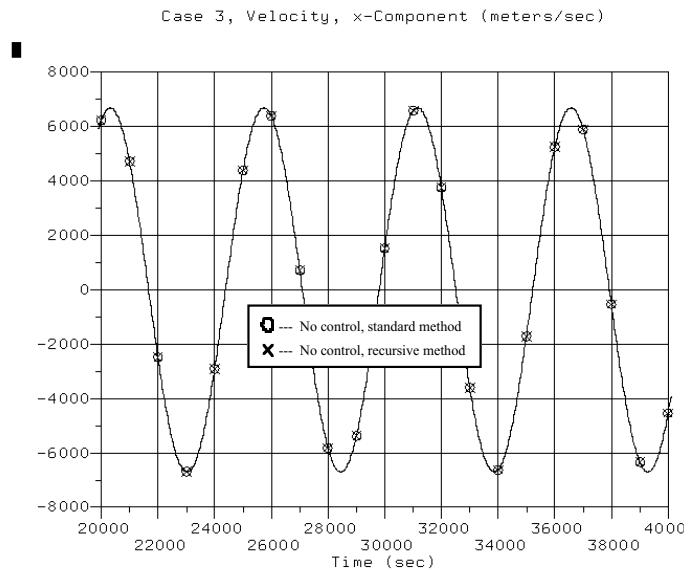
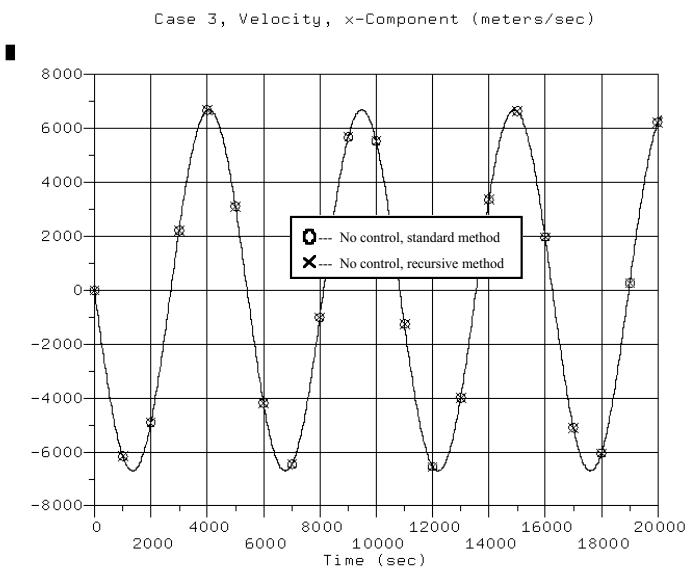
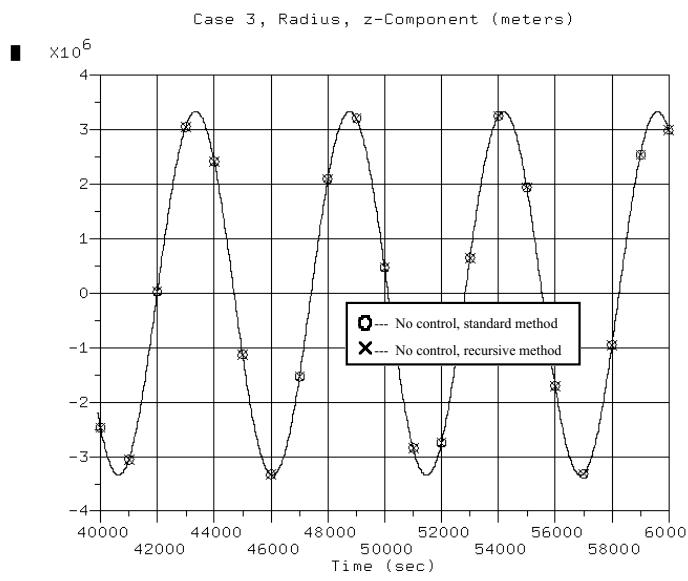
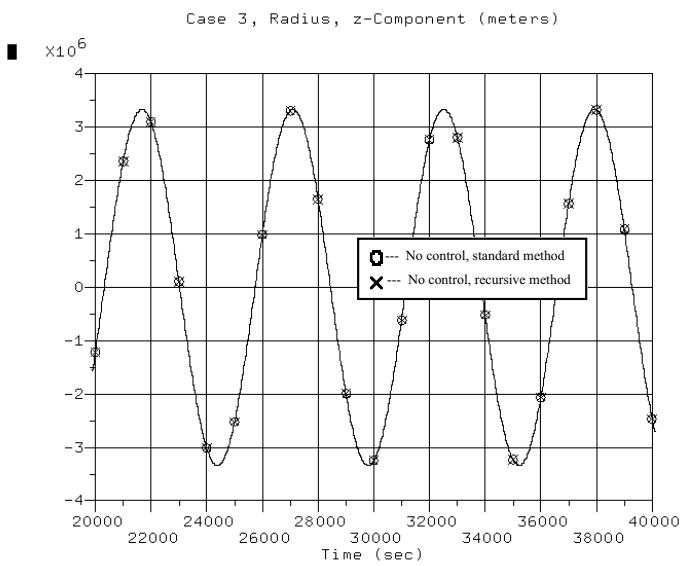
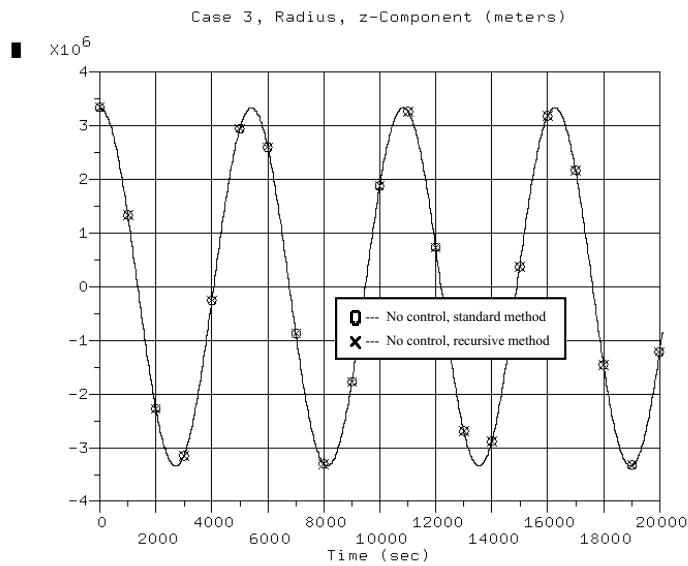


Case 2, Aerodynamic Torque, z-Component (newton-meters)

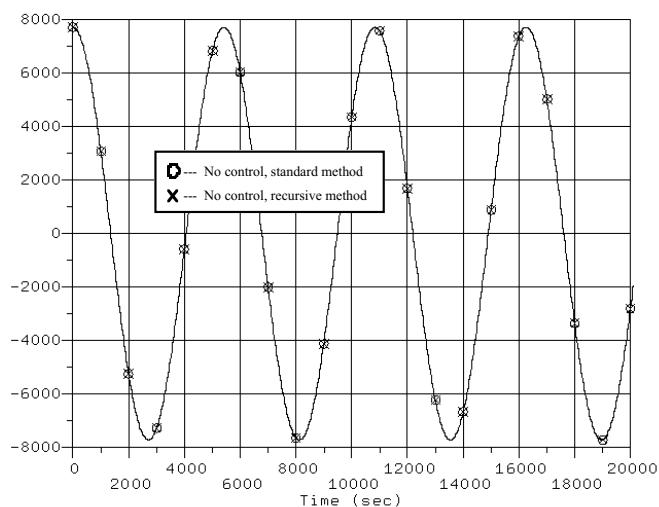


A.3 Response Plots for Case 3 (No Attitude Control)

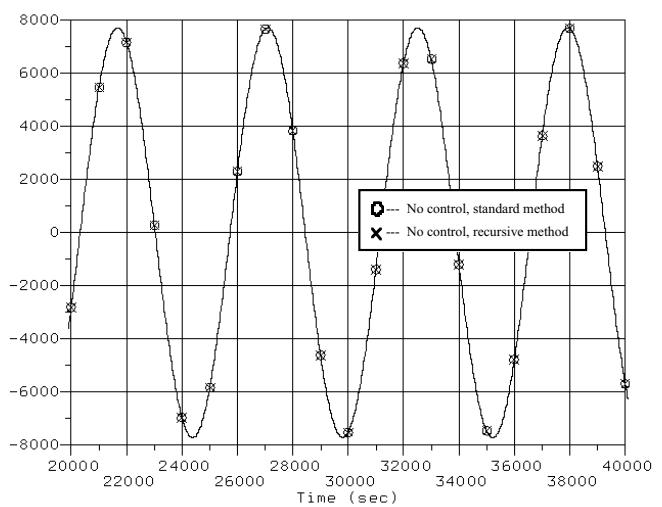




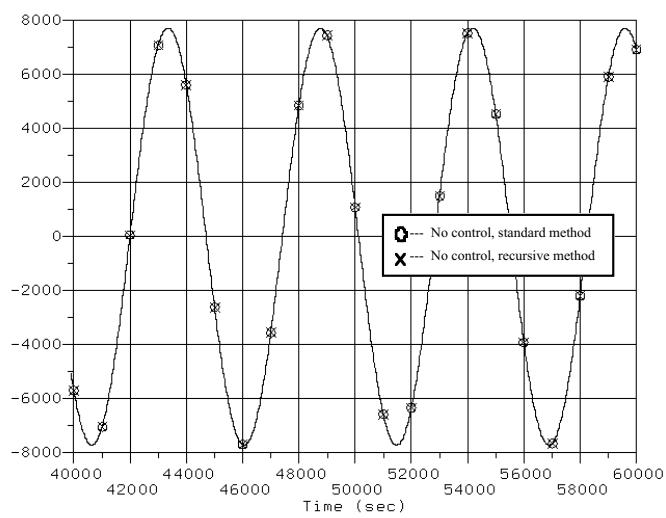
Case 3, Velocity, y-Component (meters/sec)



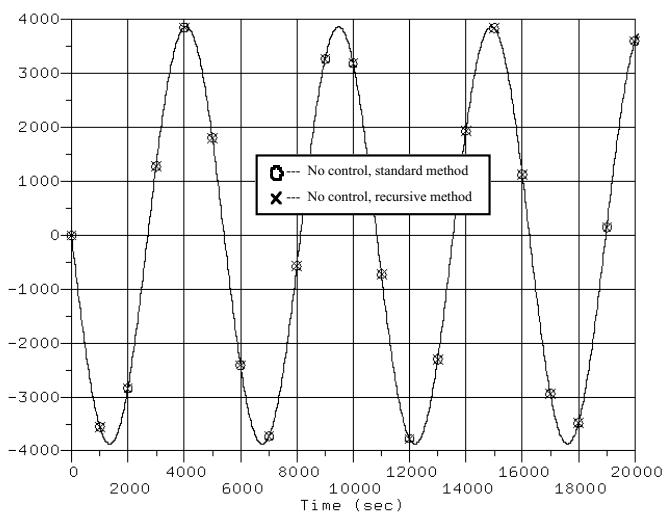
Case 3, Velocity, y-Component (meters/sec)



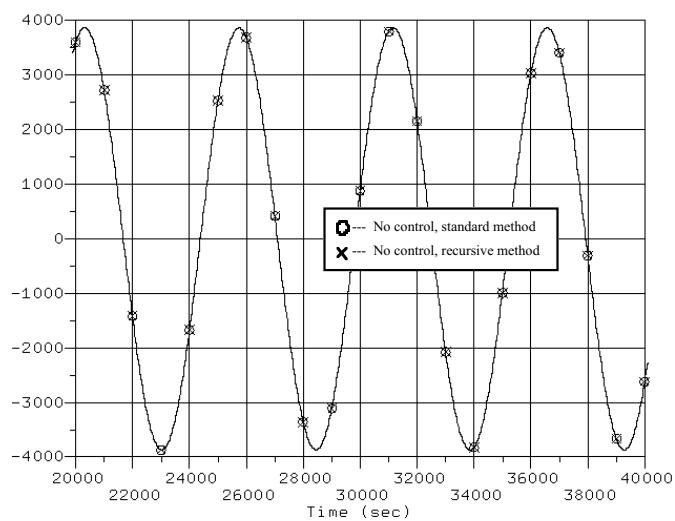
Case 3, Velocity, y-Component (meters/sec)



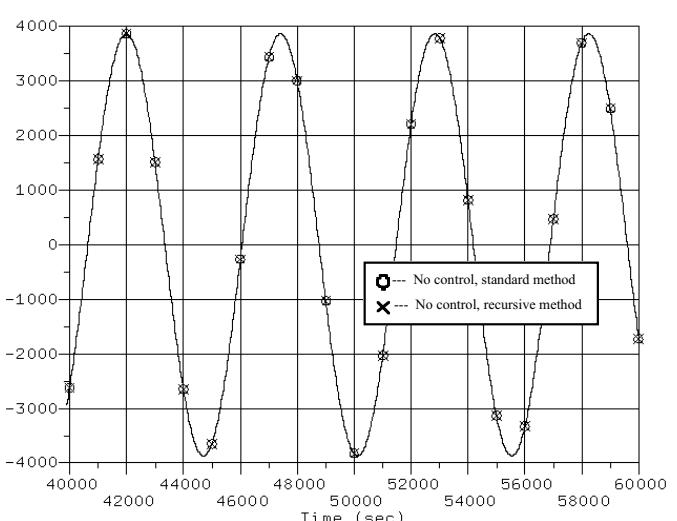
Case 3, Velocity, z-Component (meters/sec)



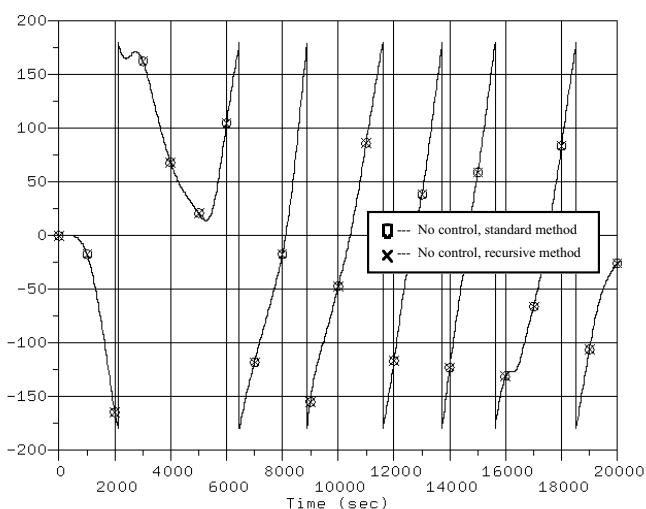
Case 3, Velocity, z-Component (meters/sec)



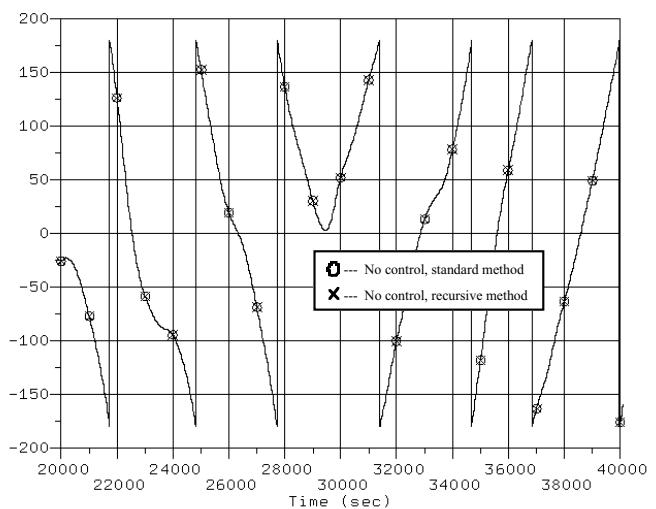
Case 3, Velocity, z-Component (meters/sec)



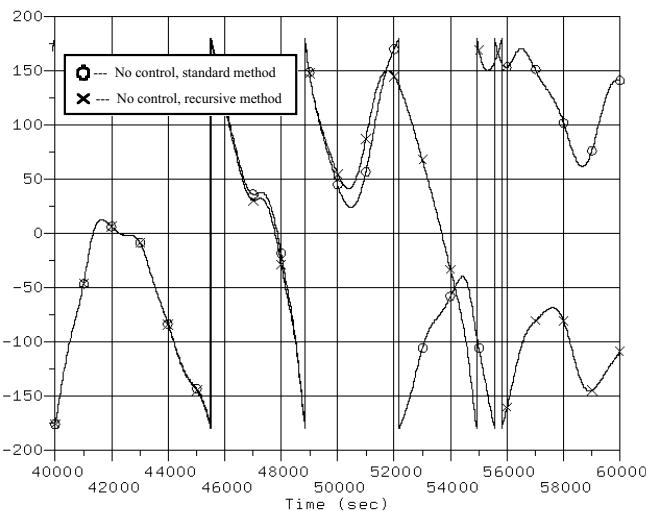
Case 3, Euler Angle, x-Axis (degrees)



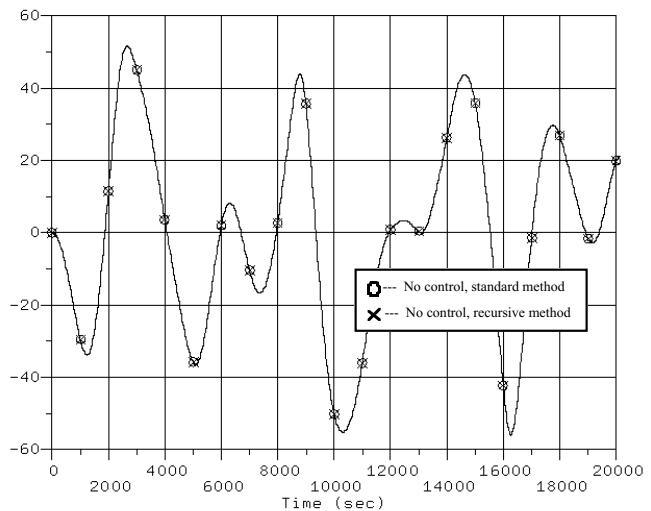
Case 3, Euler Angle, x-Axis (degrees)



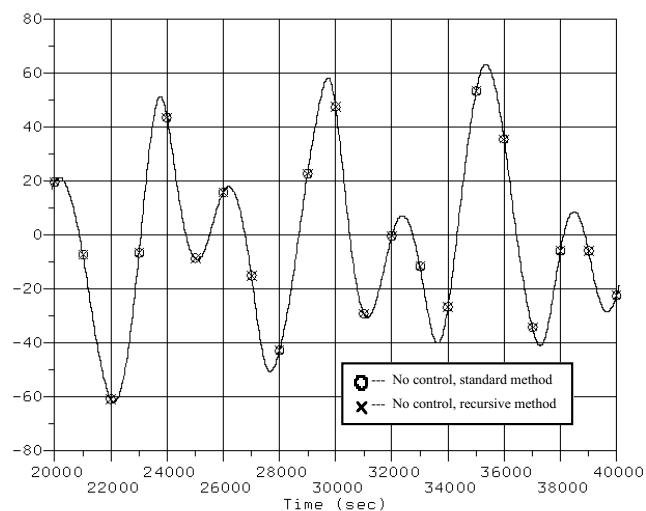
Case 3, Euler Angle, x-Axis (degrees)



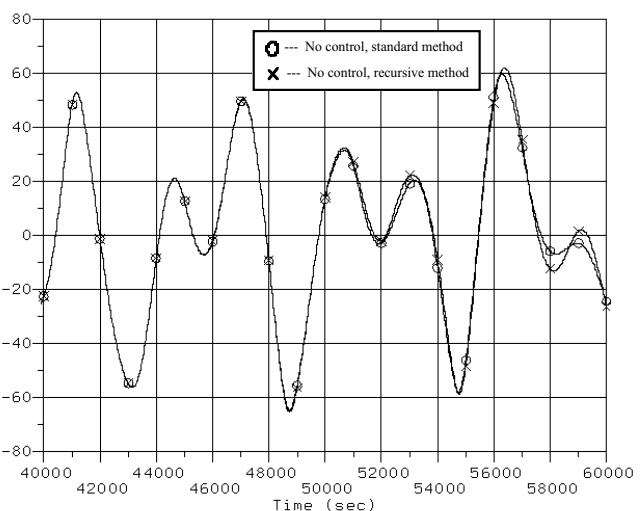
Case 3, Euler Angle, y-Axis (degrees)



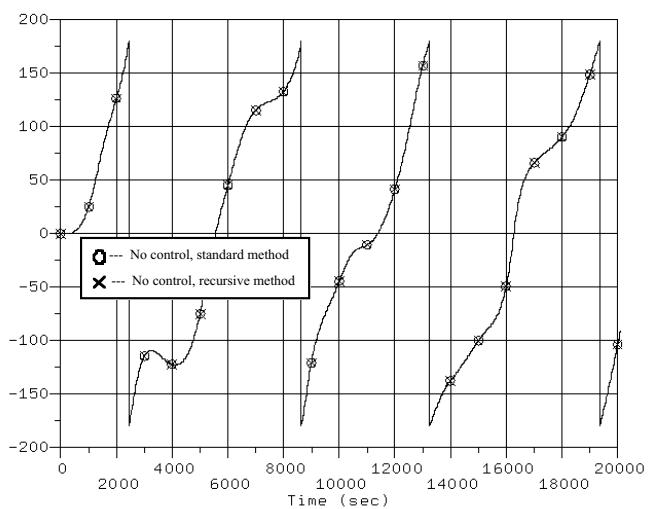
Case 3, Euler Angle, y-Axis (degrees)



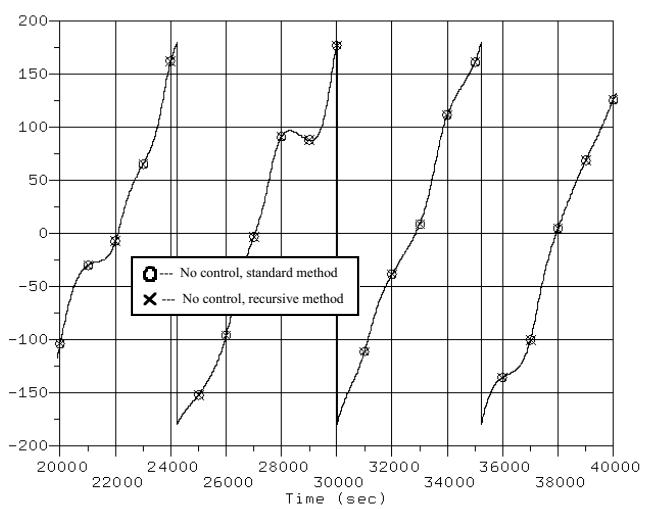
Case 3, Euler Angle, y-Axis (degrees)



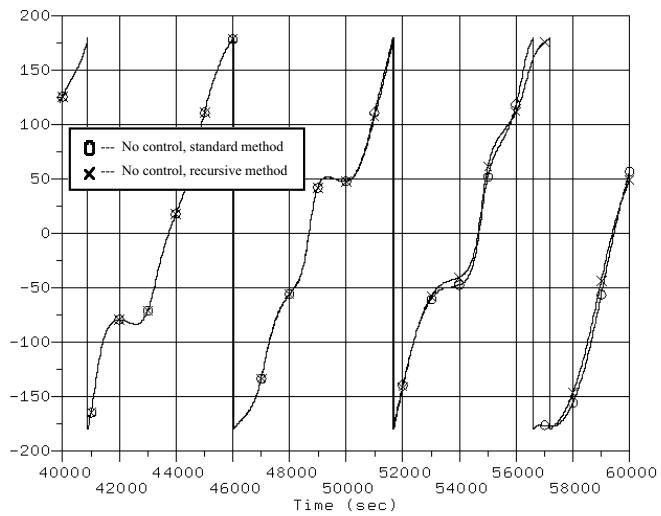
Case 3, Euler Angle, z-Axis (degrees)



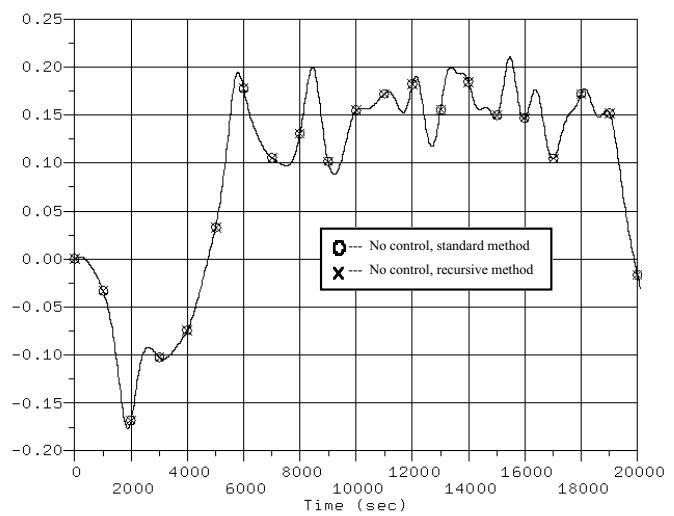
Case 3, Euler Angle, z-Axis (degrees)



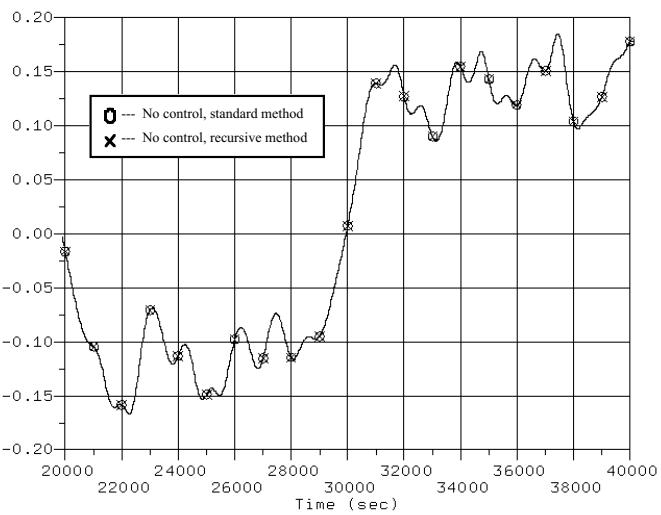
Case 3, Euler Angle, z-Axis (degrees)



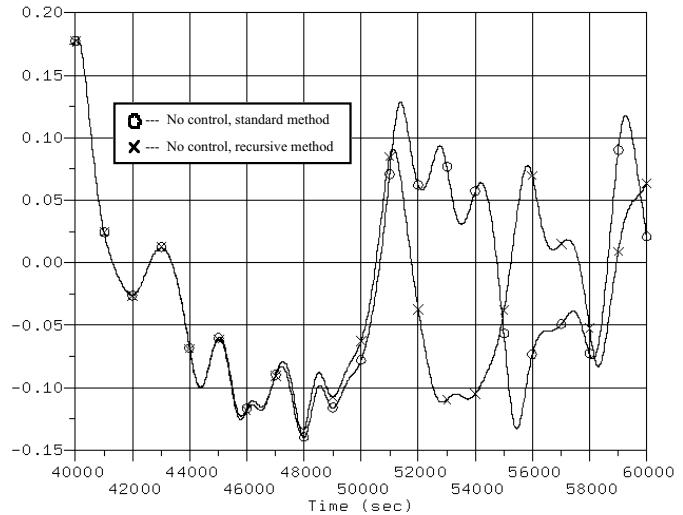
Case 3, Body Rate, x-Axis (deg/sec)



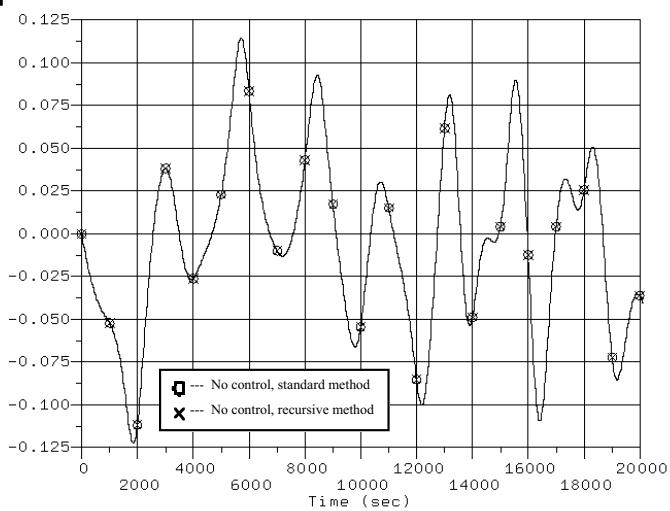
Case 3, Body Rate, x-Axis (deg/sec)



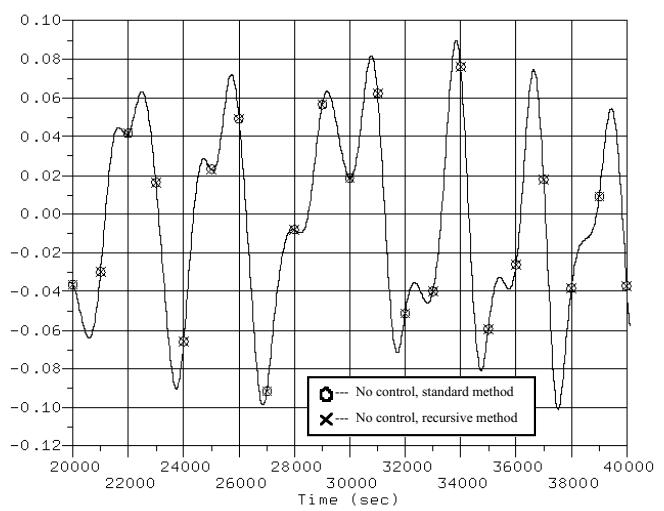
Case 3, Body Rate, x-Axis (deg/sec)



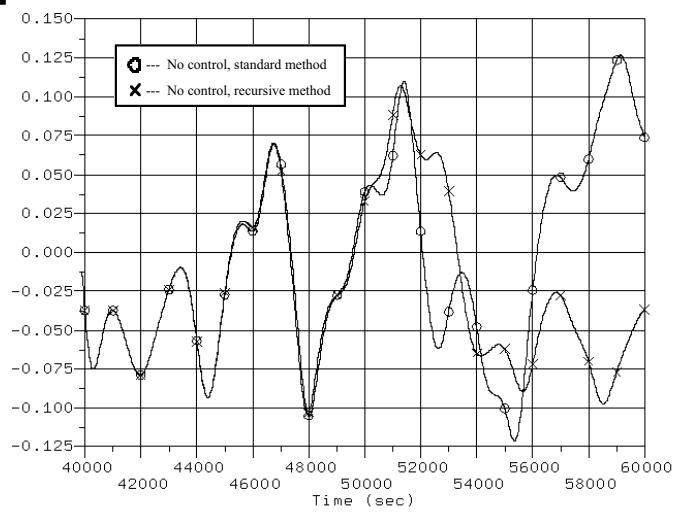
Case 3, Body Rate, y-Axis (deg/sec)



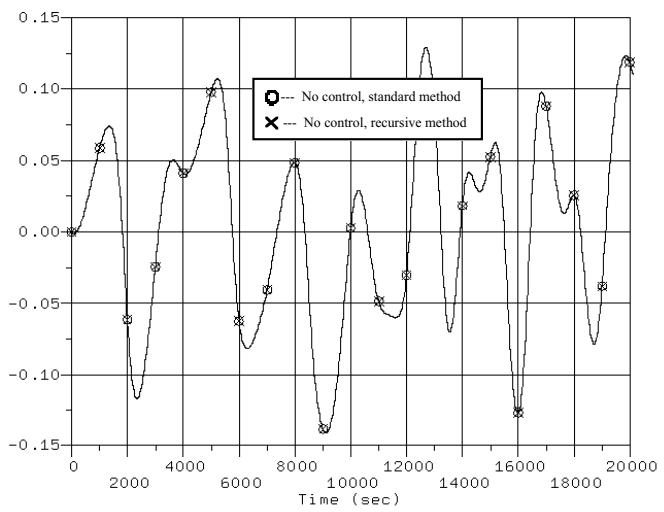
Case 3, Body Rate, y-Axis (deg/sec)



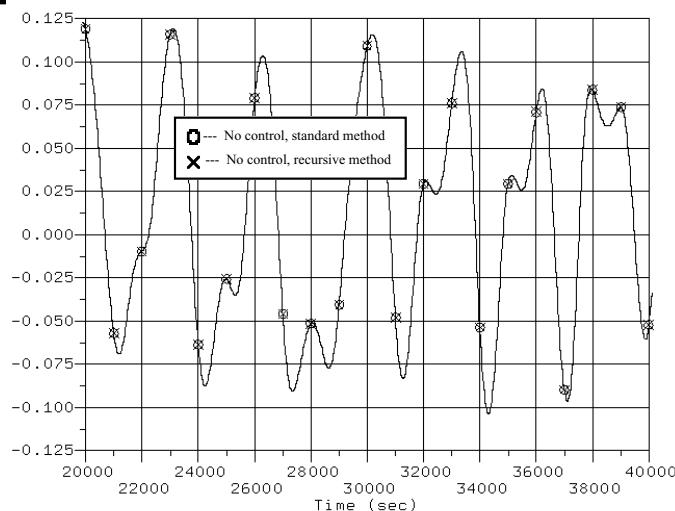
Case 3, Body Rate, y-Axis (deg/sec)



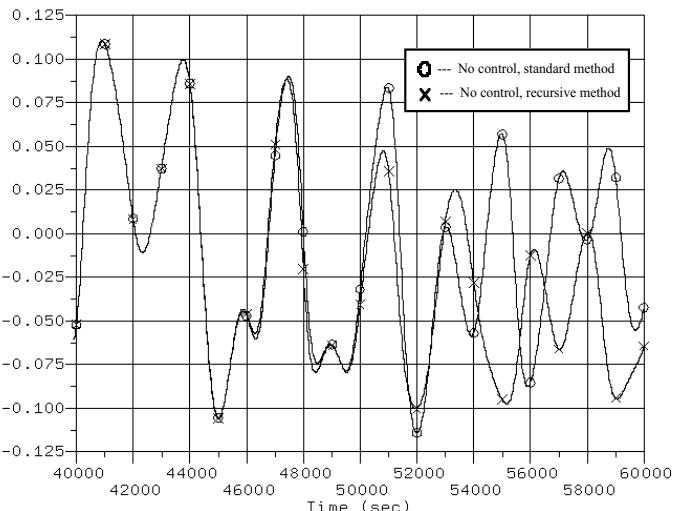
Case 3, Body Rate, z-Axis (deg/sec)



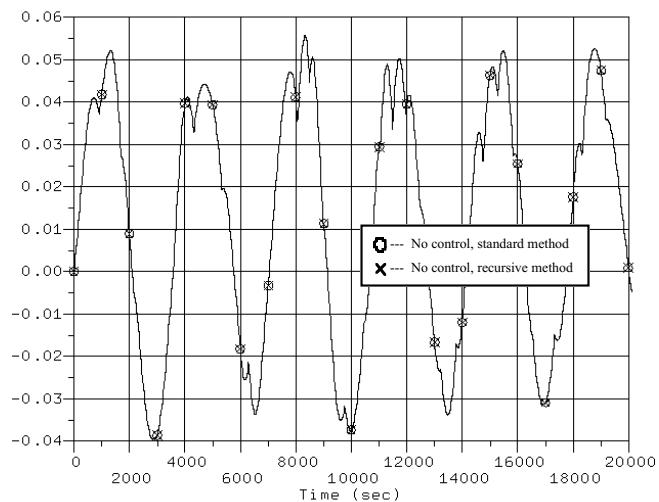
Case 3, Body Rate, z-Axis (deg/sec)



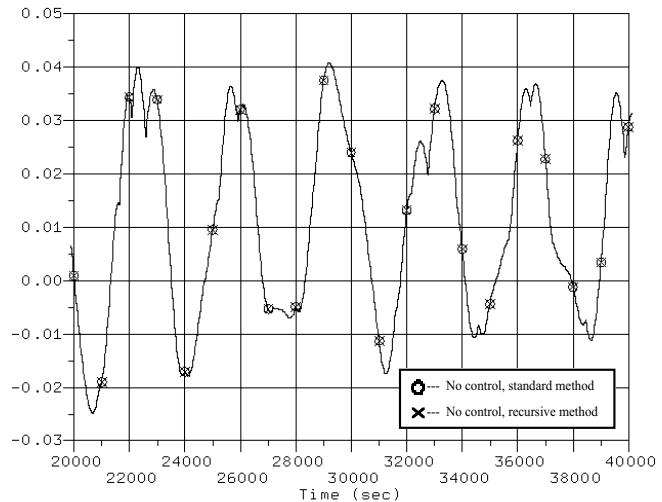
Case 3, Body Rate, z-Axis (deg/sec)



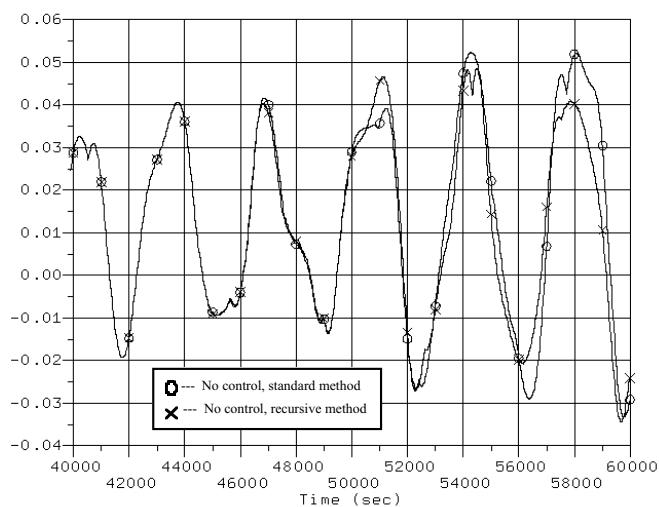
Case 3, Aerodynamic Force, x-Component (newtons)



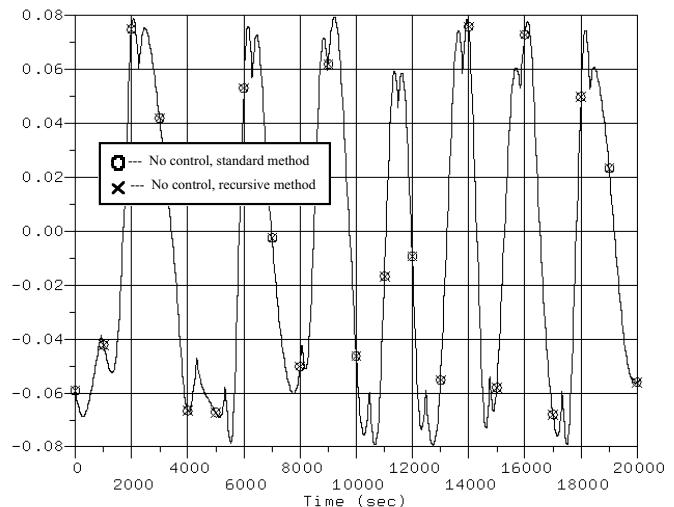
Case 3, Aerodynamic Force, x-Component (newtons)



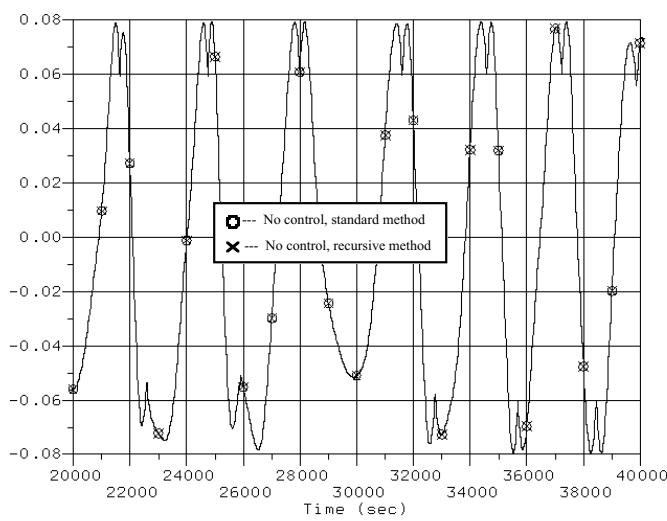
Case 3, Aerodynamic Force, x-Component (newtons)



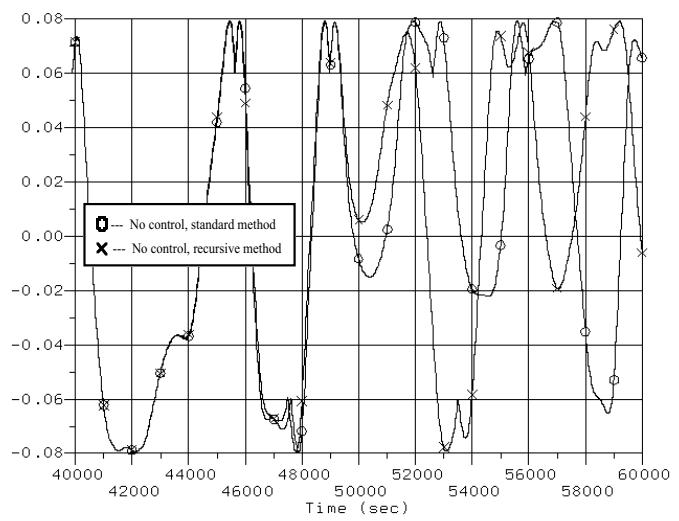
Case 3, Aerodynamic Force, y-Component (newtons)



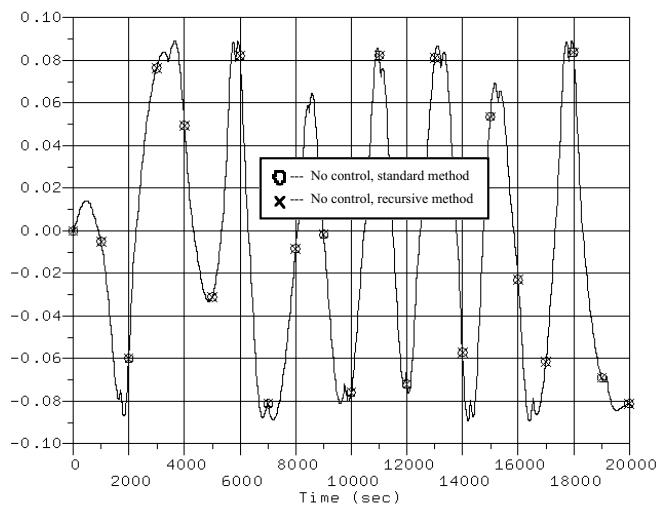
Case 3, Aerodynamic Force, y-Component (newtons)



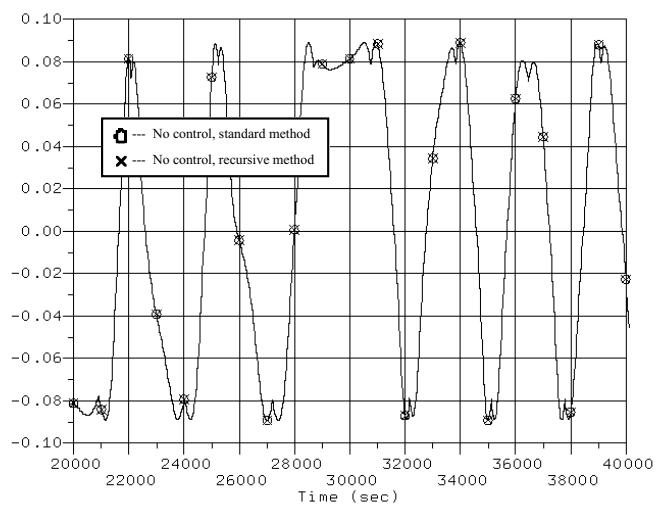
Case 3, Aerodynamic Force, y-Component (newtons)



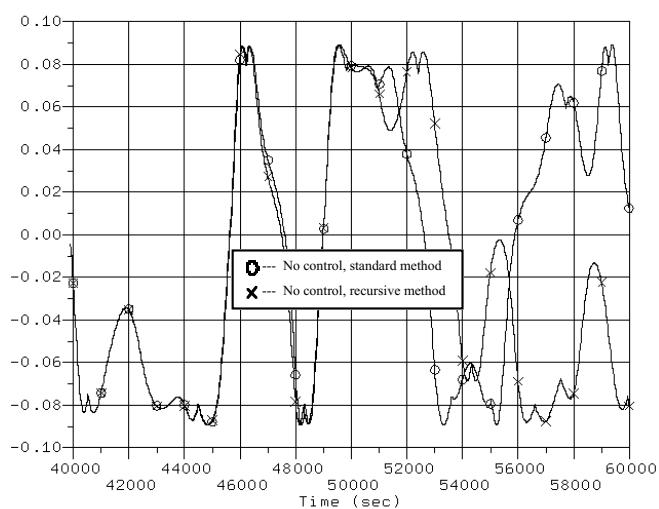
Case 3, Aerodynamic Force, z-Component (newtons)



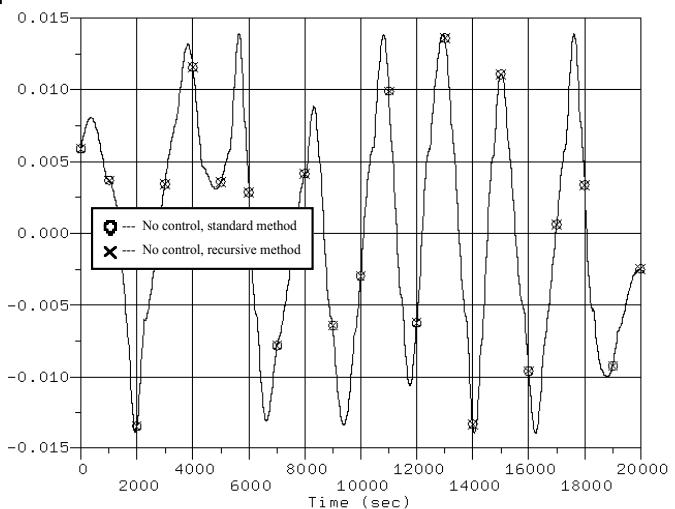
Case 3, Aerodynamic Force, z-Component (newtons)



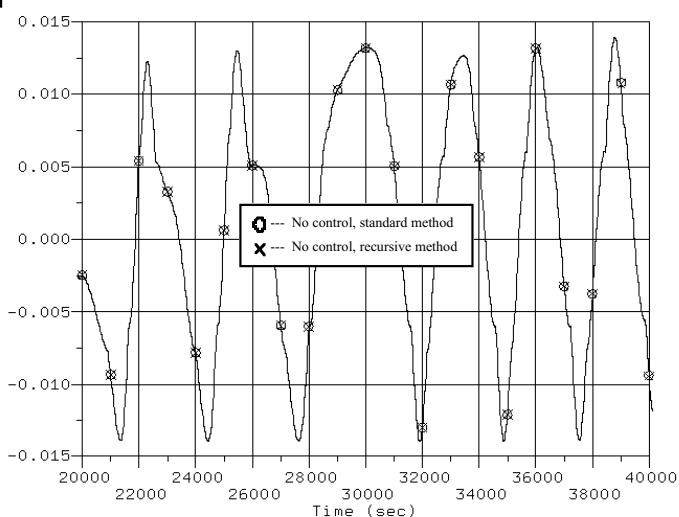
Case 3, Aerodynamic Force, z-Component (newtons)



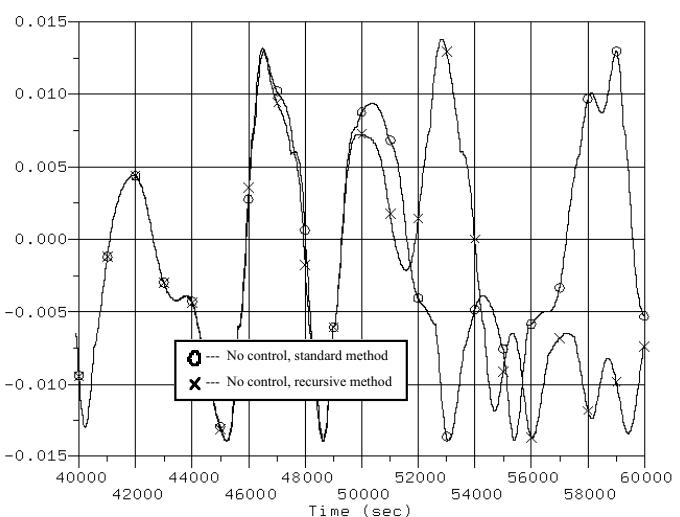
Case 3, Aerodynamic Torque, x-Component (newton-meters)



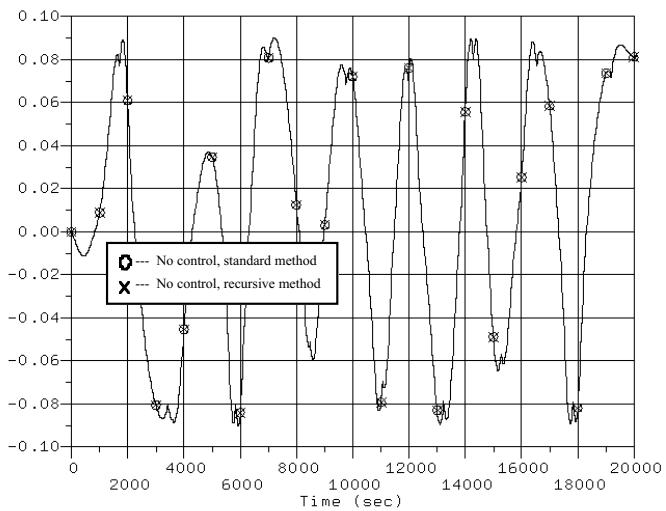
Case 3, Aerodynamic Torque, x-Component (newton-meters)



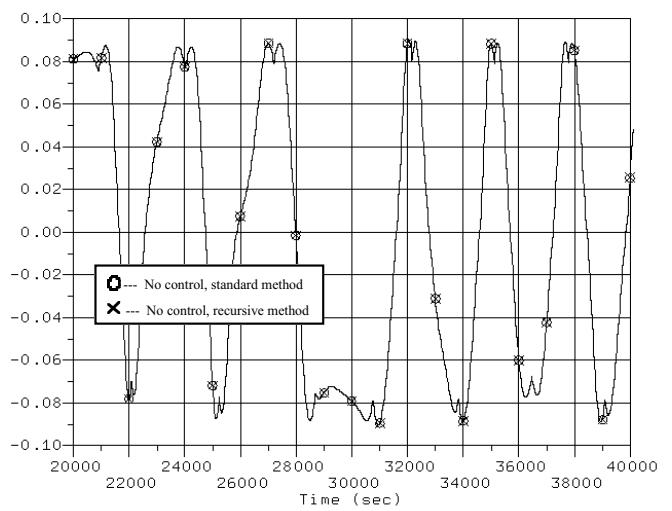
Case 3, Aerodynamic Torque, x-Component (newton-meters)



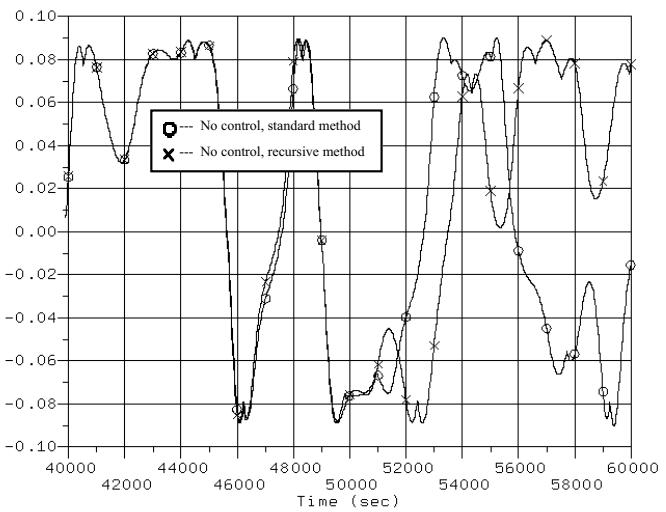
Case 3, Aerodynamic Torque, y-Component (newton-meters)



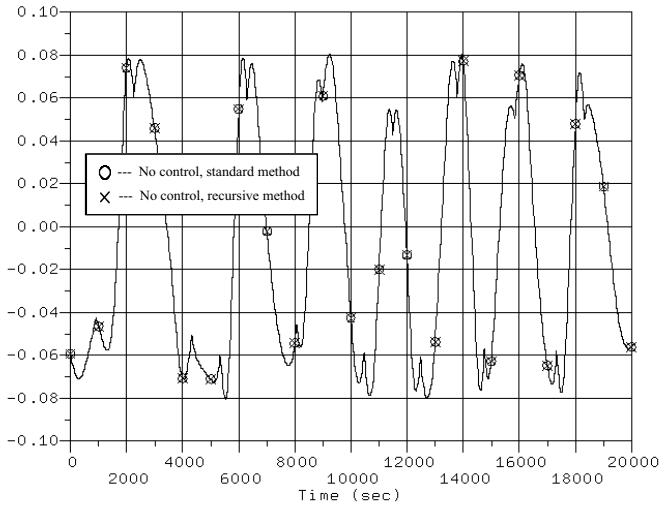
Case 3, Aerodynamic Torque, y-Component (newton-meters)



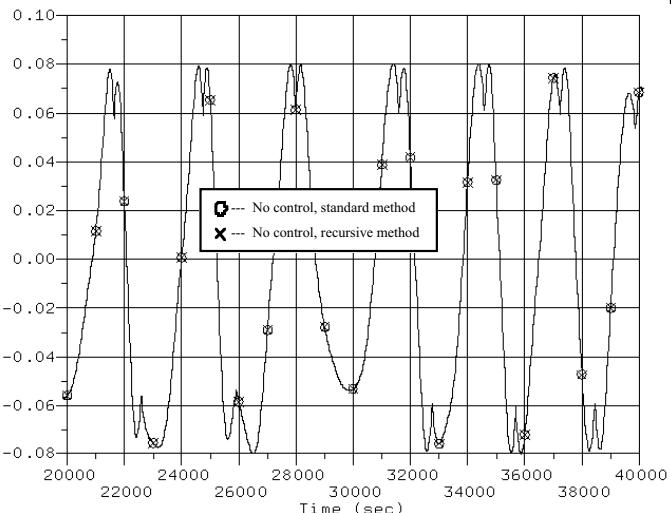
Case 3, Aerodynamic Torque, y-Component (newton-meters)



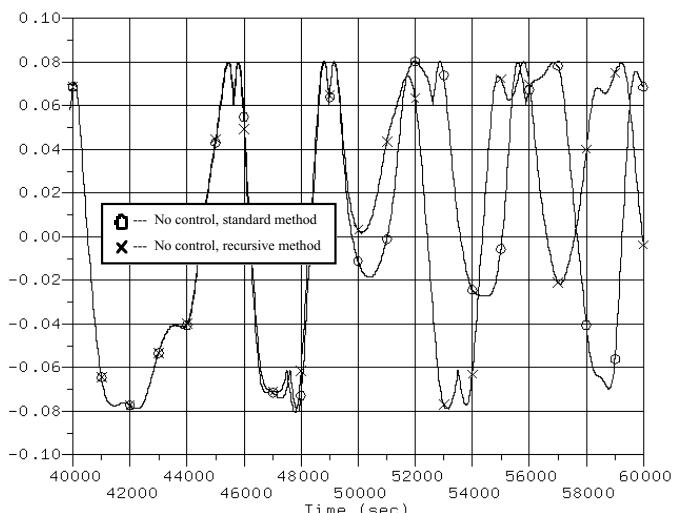
Case 3, Aerodynamic Torque, z-Component (newton-meters)



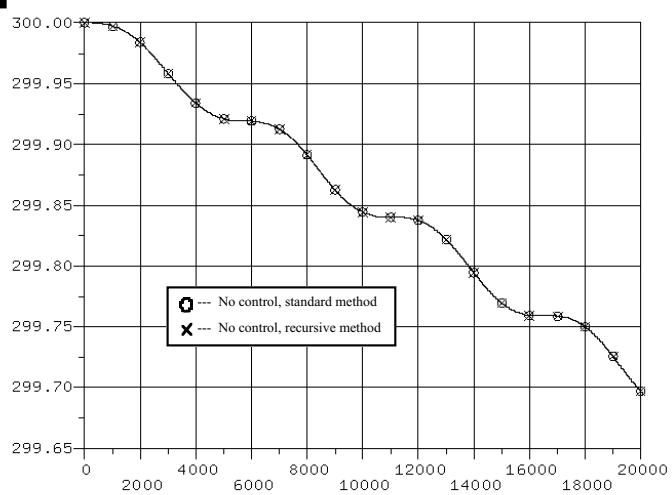
Case 3, Aerodynamic Torque, z-Component (newton-meters)



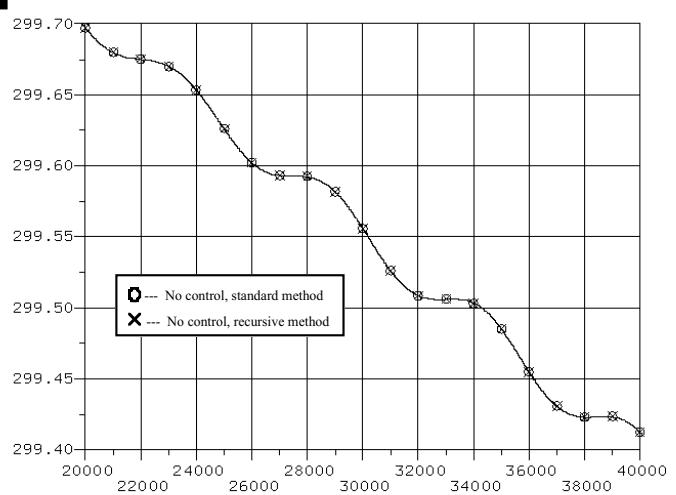
Case 3, Aerodynamic Torque, z-Component (newton-meters)



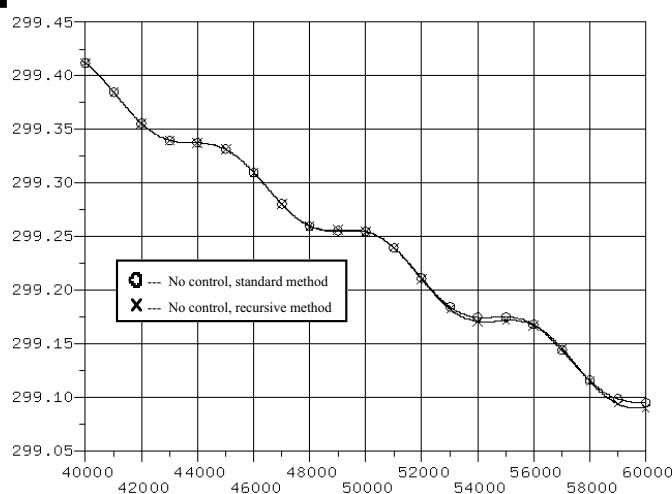
Case 3, Altitude (kilometers)



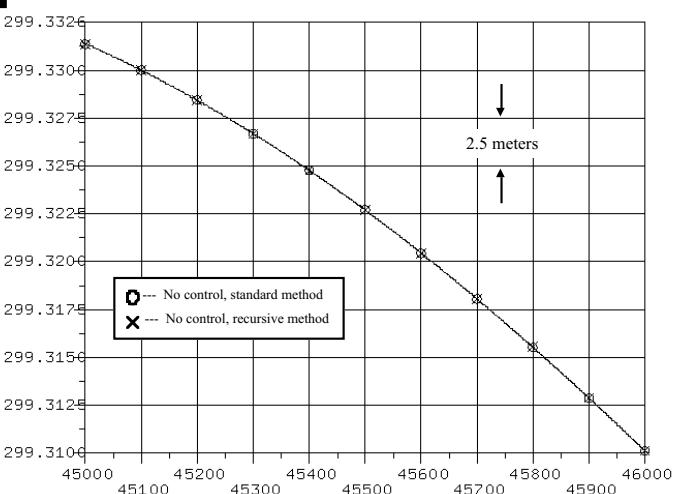
Case 3, Altitude (kilometers)



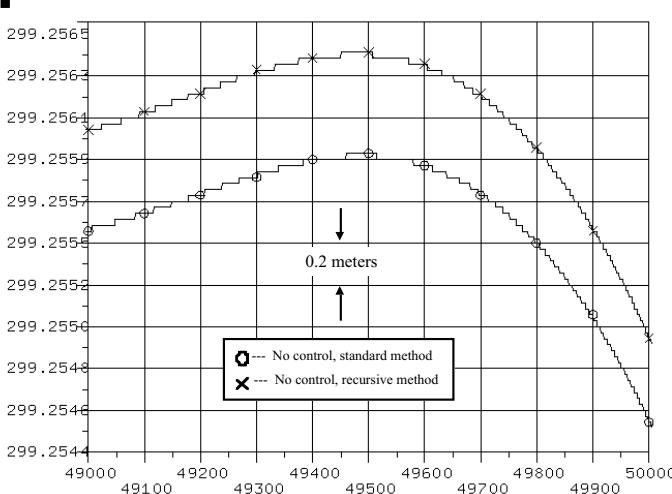
Case 3, Altitude (kilometers)



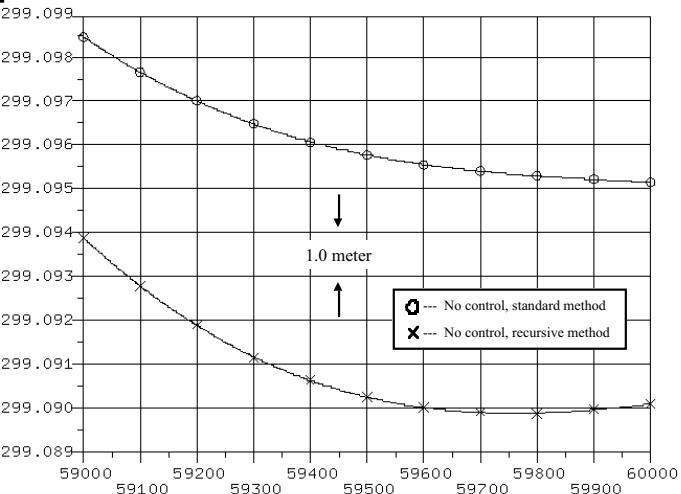
Case 3, Altitude (kilometers)



Case 3, Altitude (kilometers)

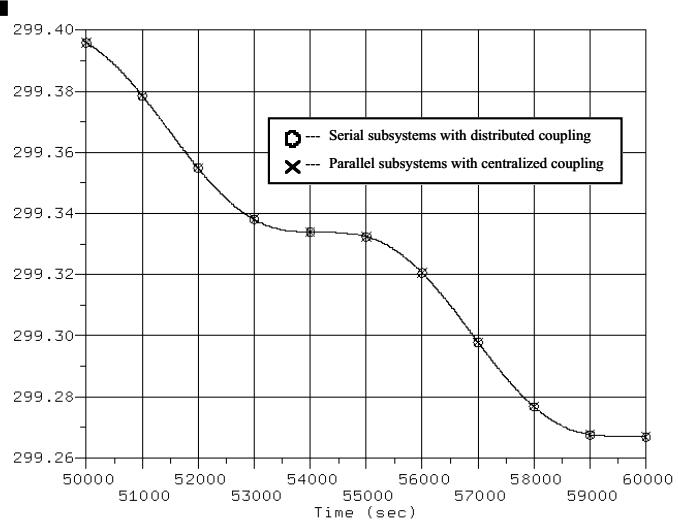


Case 3, Altitude (kilometers)

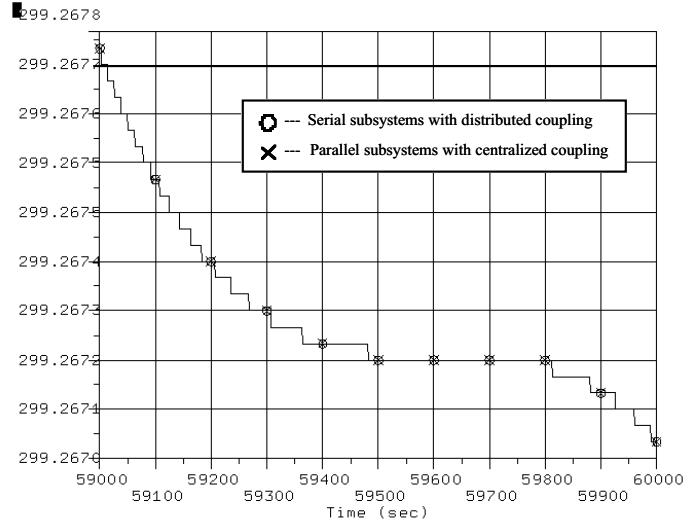


A.4 Response Plots for Cases 1–3 With Central and Distributed Coupling

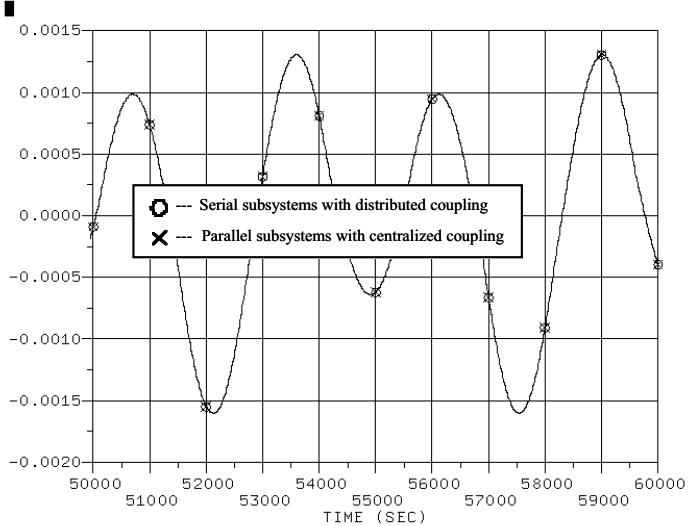
Case 1, Altitude, (meters)



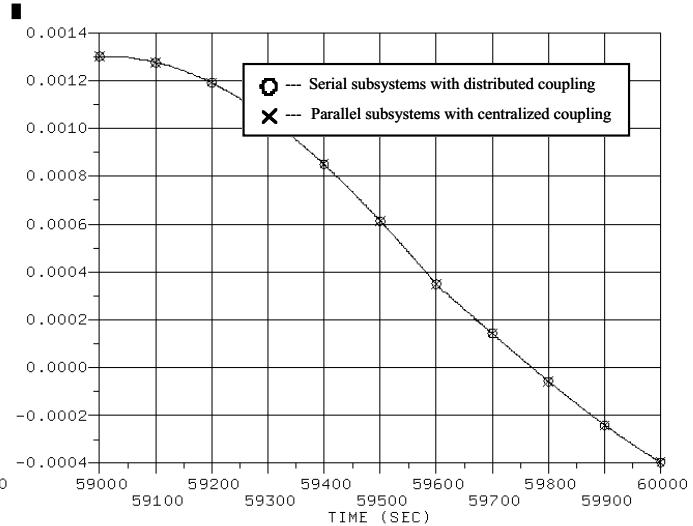
Case 1, Altitude (meters)



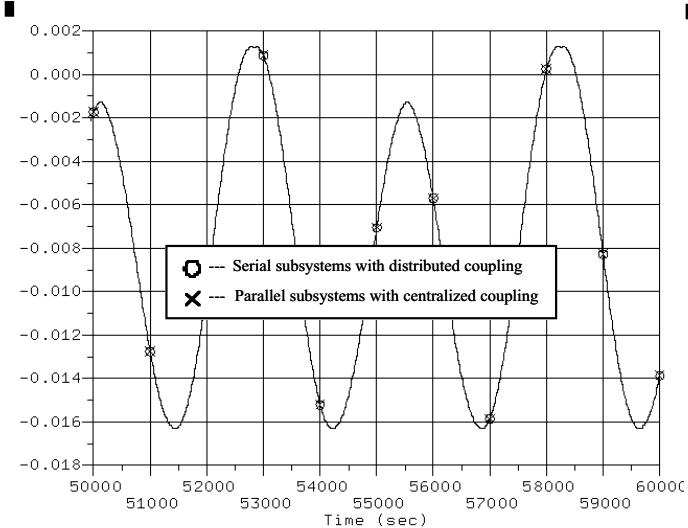
Case 1, Euler Angle, x-Axis (degrees)



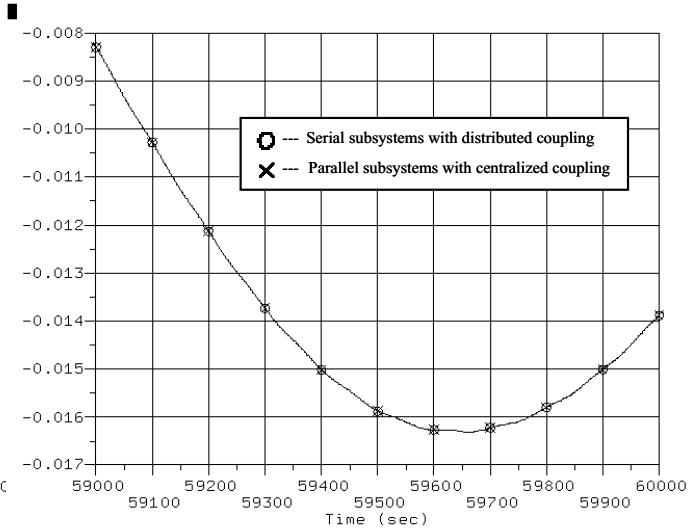
Case 1, Euler Angle, x-Axis (degrees)



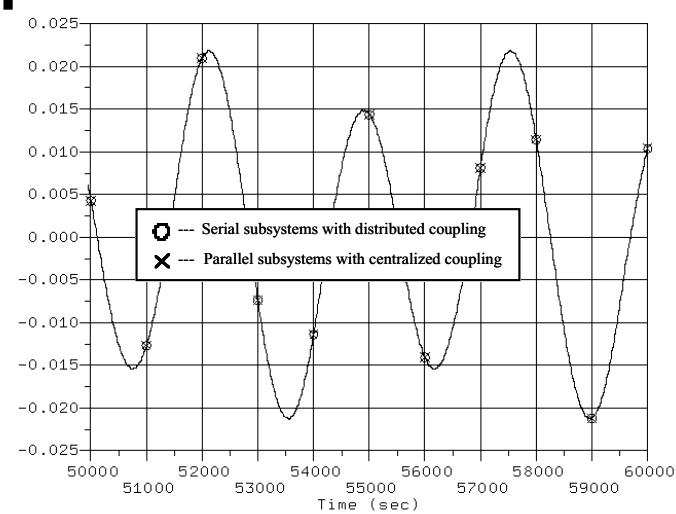
Case 1, Euler Angle, y-Axis (degrees)



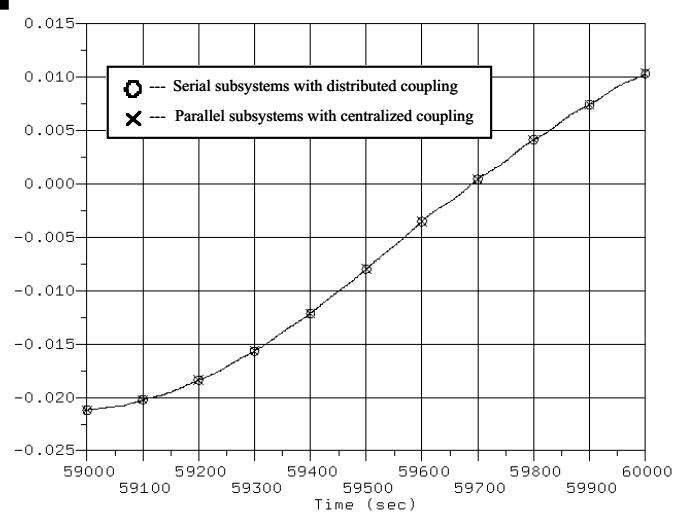
Case 1, Euler Angle, y-Axis (degrees)



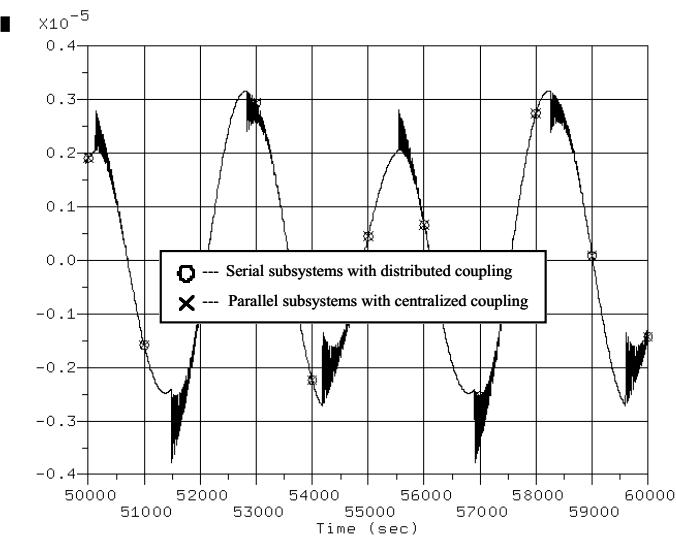
Case 1, Euler Angle, z-Axis (degrees)



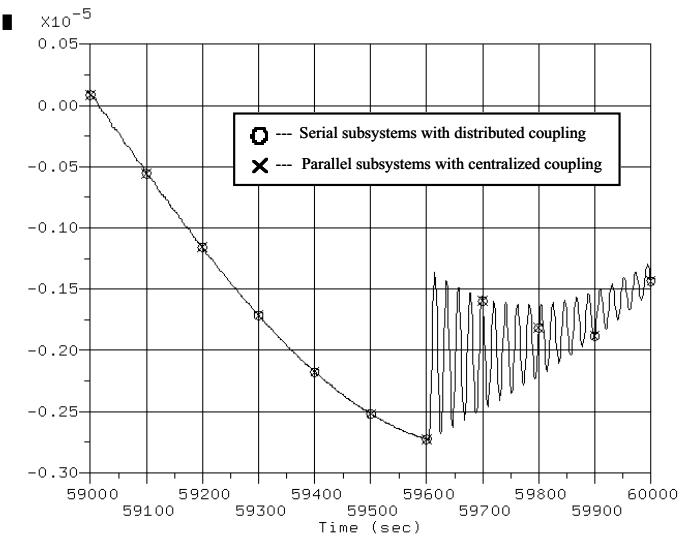
Case 1, Euler Angle, z-Axis (degrees)



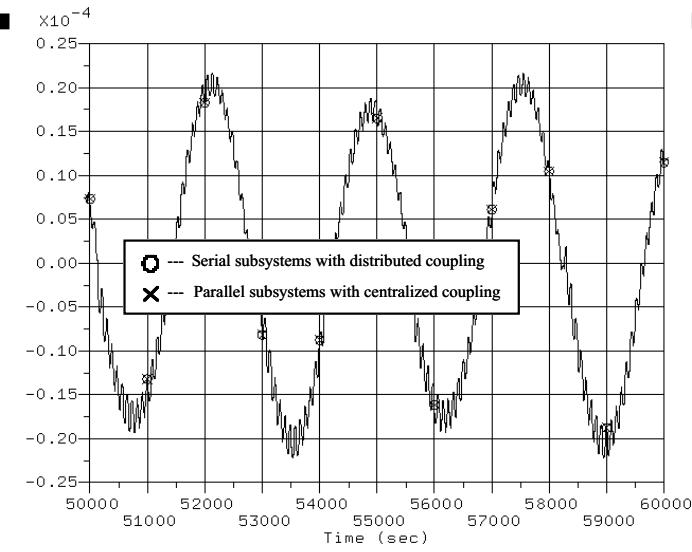
Case 1 Body Rate, x-Axis (deg/sec)



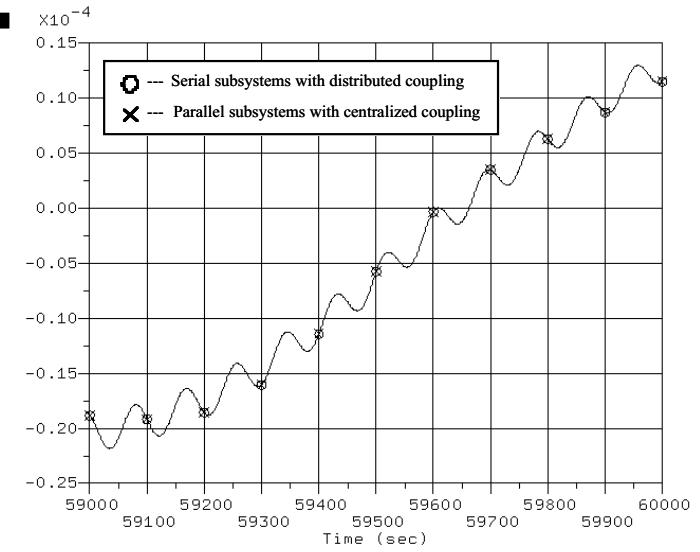
Case 1 Body Rate, x-Axis (deg/sec)

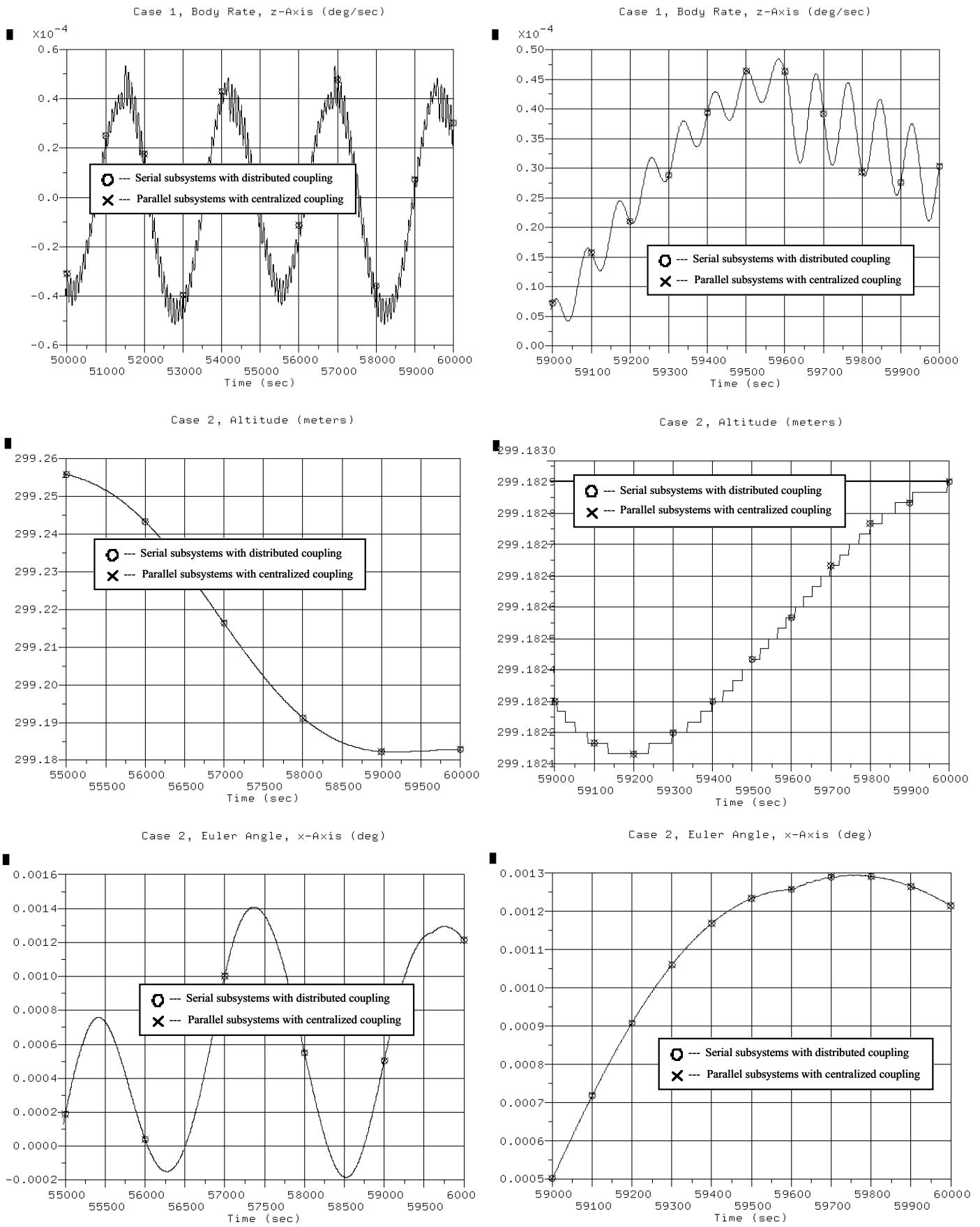


Case 1, Body Rate, y-Axis (deg/sec)

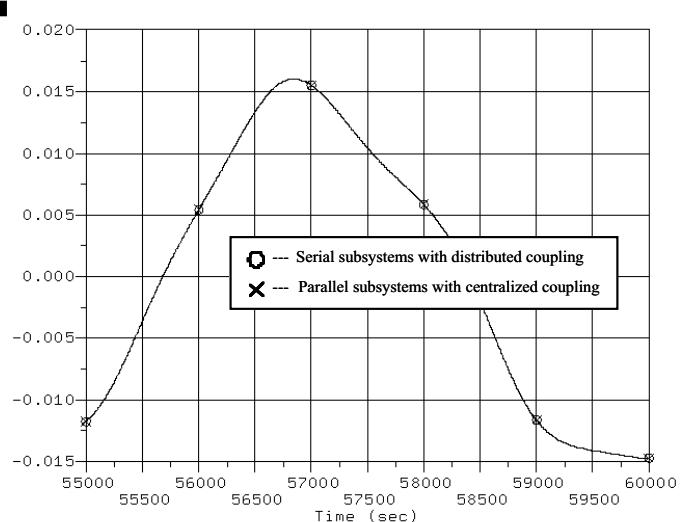


Case 1, Body Rate, y-Axis (deg/sec)

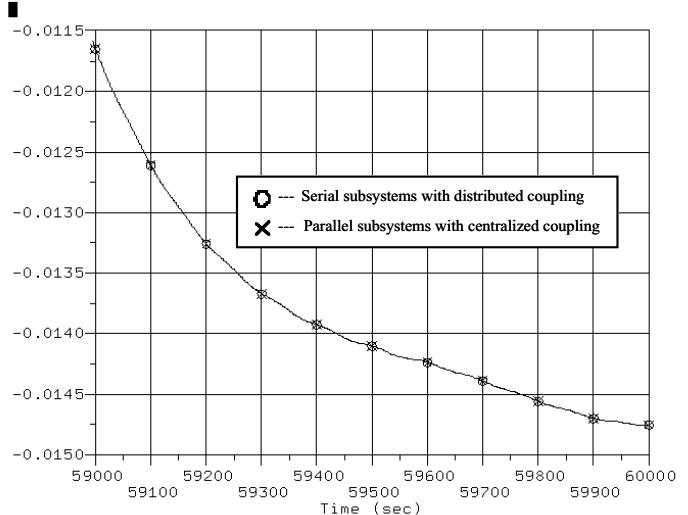




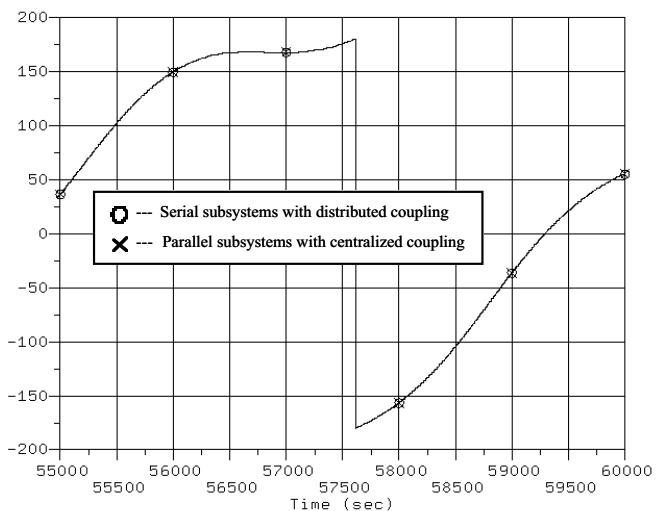
Case 2, Euler Angle, y-Axis (degrees)



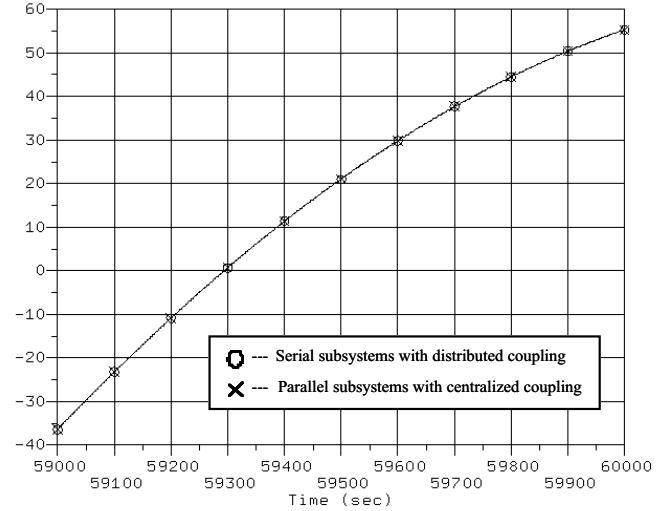
Case 2, Euler Angle, y-Axis (degrees)



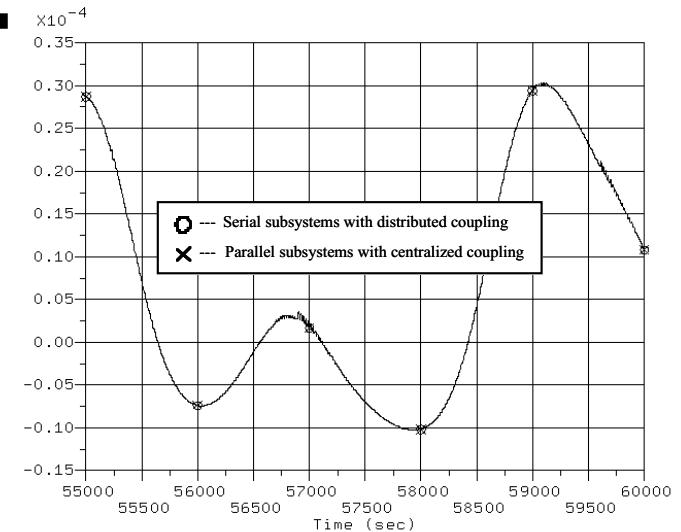
Case 2, Euler Angle, z-Axis (deg)



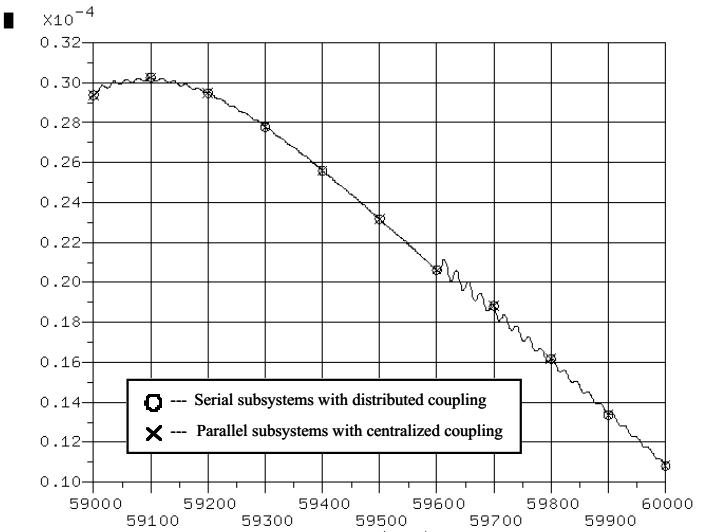
Case 2, Euler Angle, z-Axis (deg)

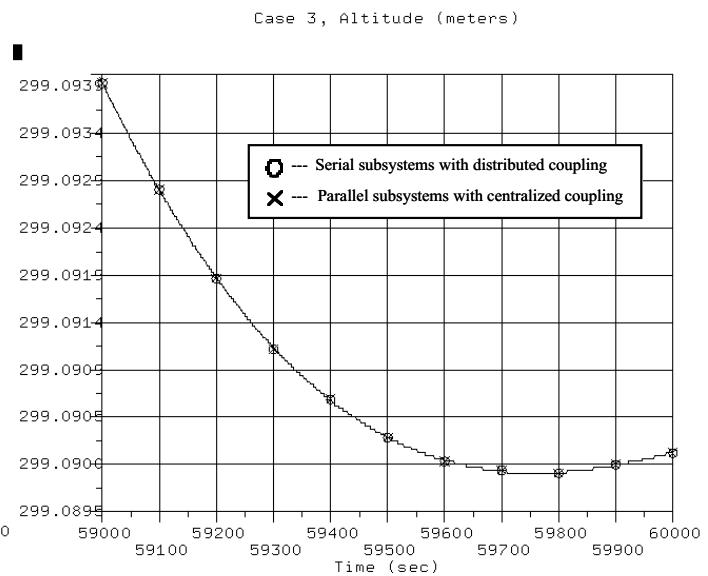
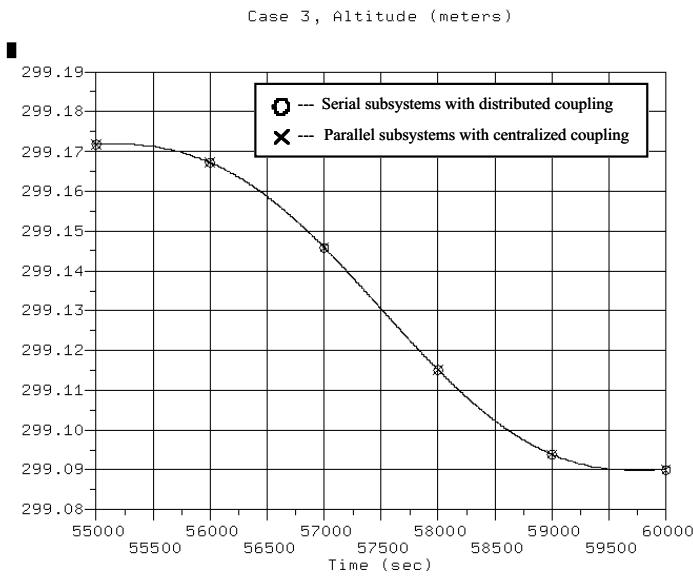
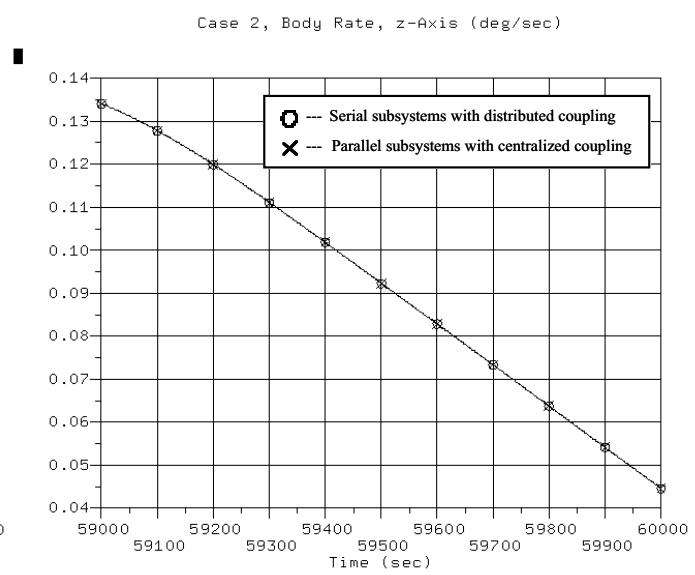
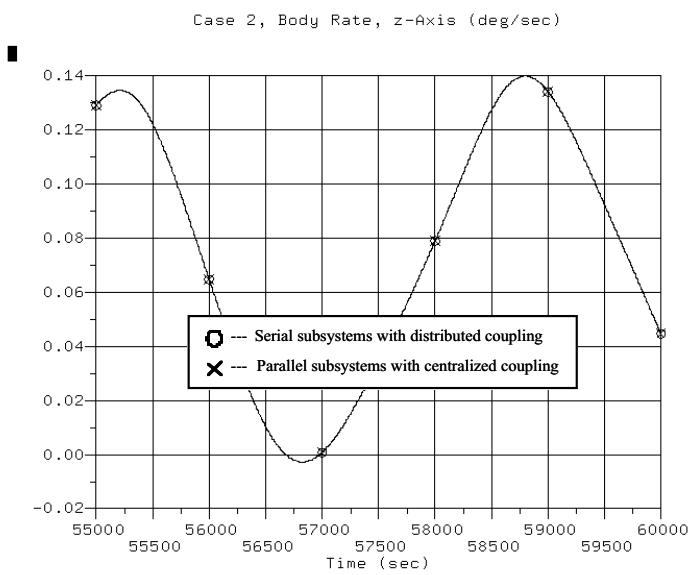
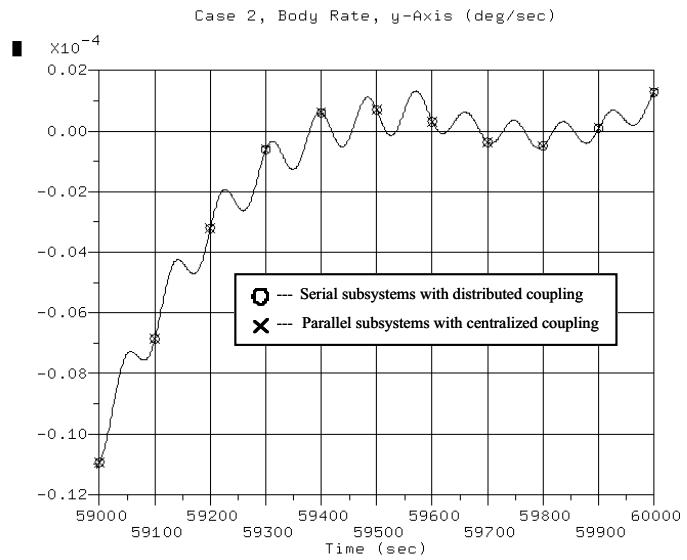
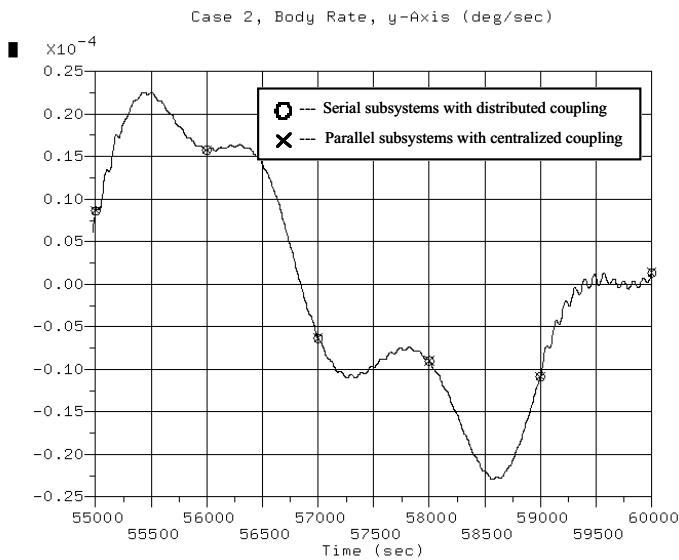


Case 2, Body Rate, x-Axis (deg/sec)

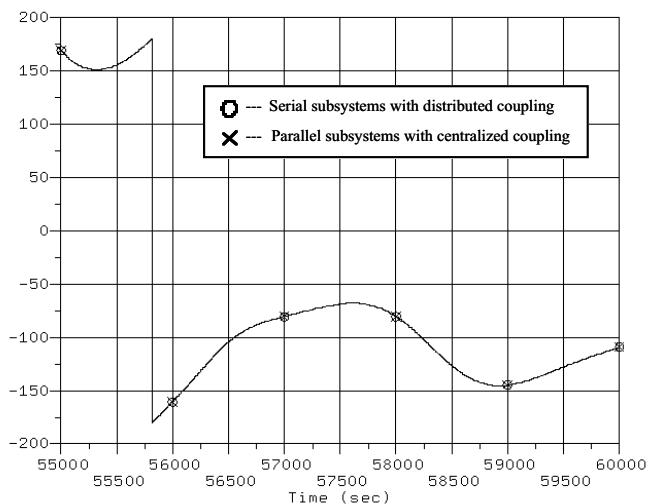


Case 2, Body Rate, x-Axis (deg/sec)

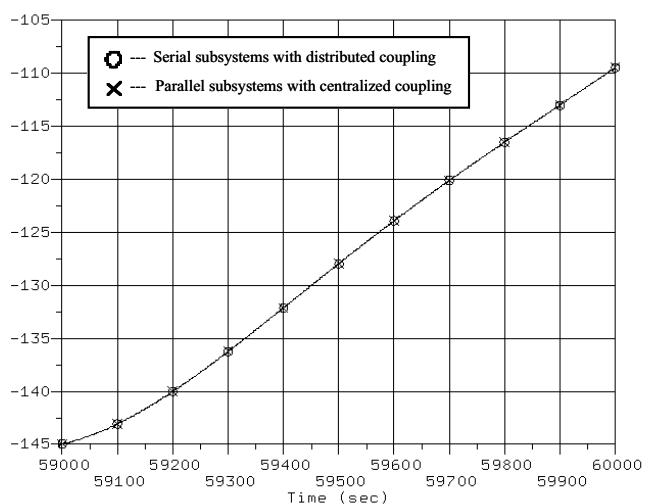




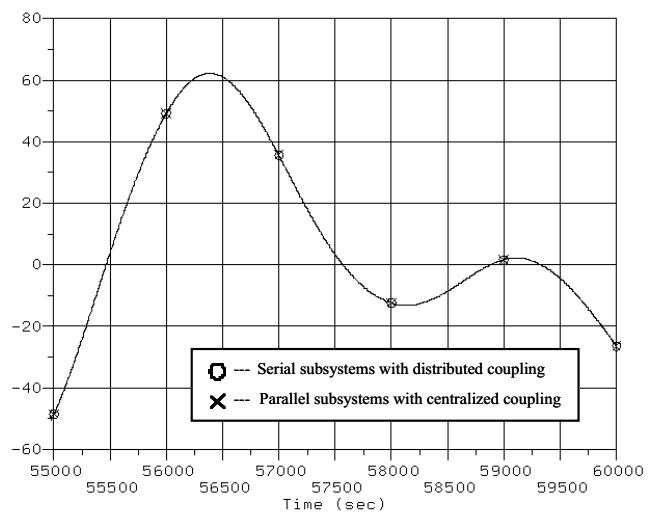
Case 3, Euler Angle, x-Axis (degrees)



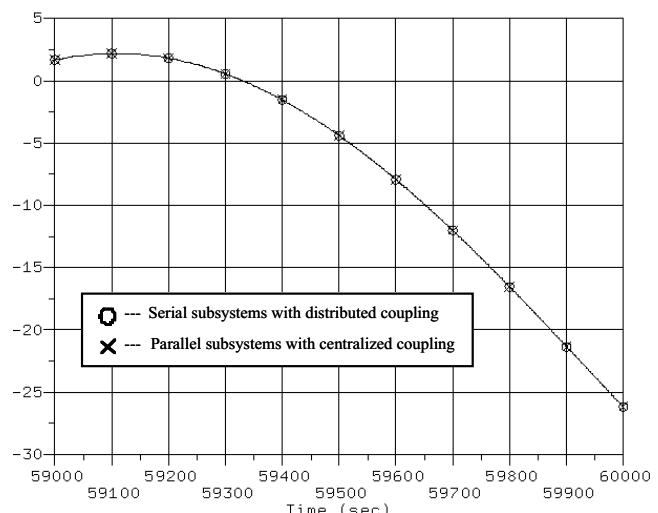
Case 3, Euler Angle, x-Axis (degrees)



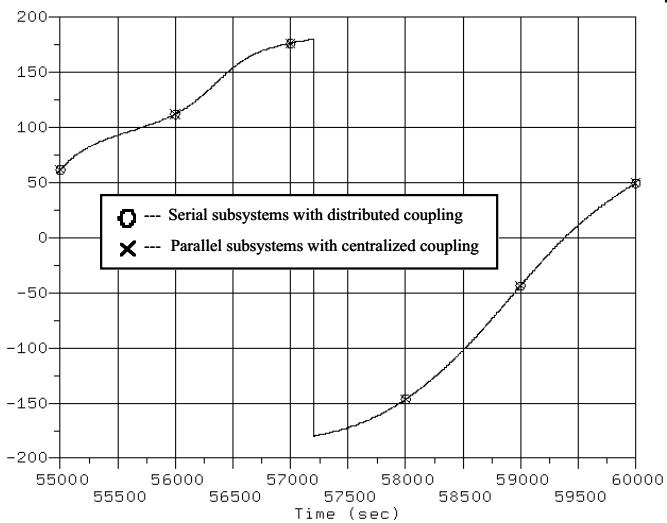
Case 3, Euler Angle, y-Axis (degrees)



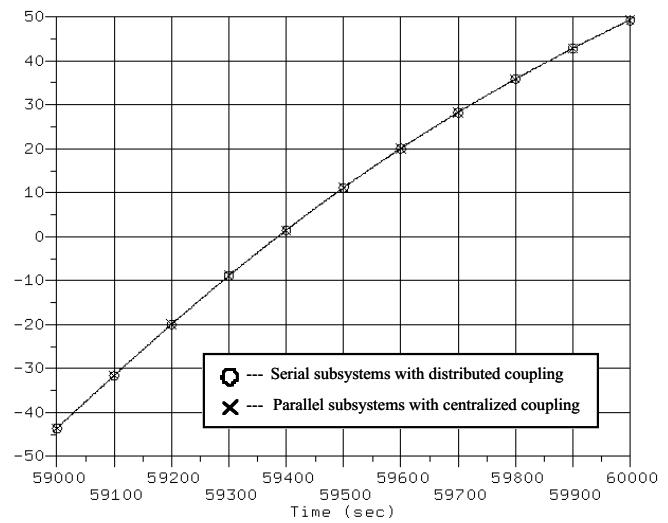
Case 3, Euler Angle, y-Axis (degrees)



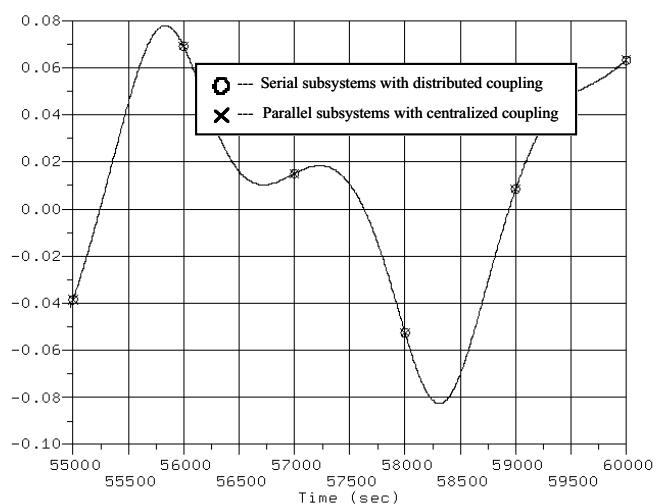
Case 3, Euler Angle, z-Axis (degrees)



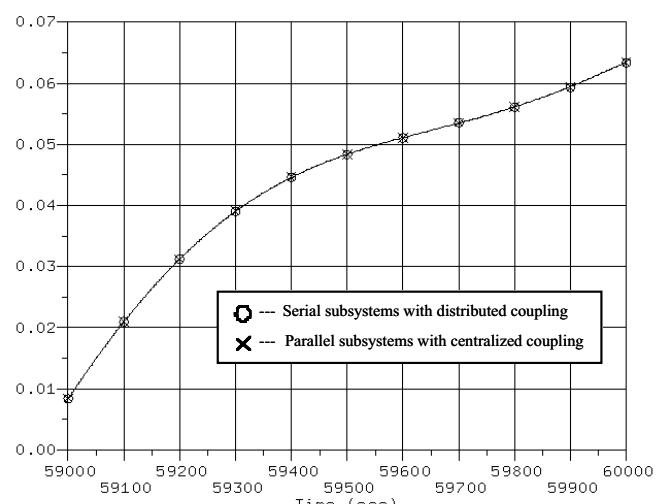
Case 3, Euler Angle, z-Axis (degrees)



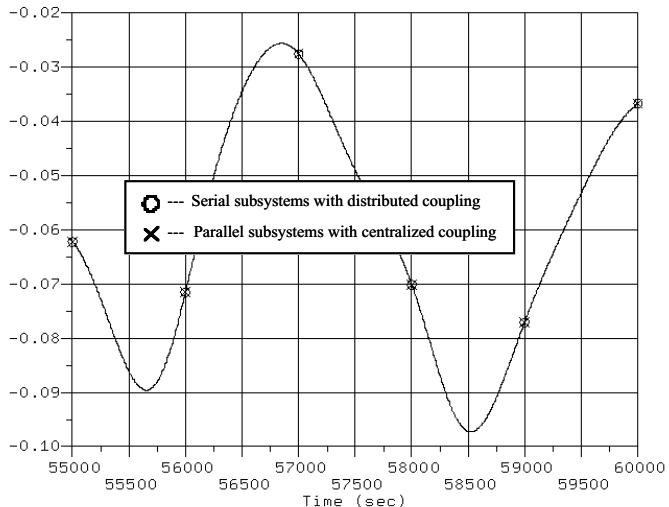
Case 3, Body Rate, x-Axis (deg/sec)



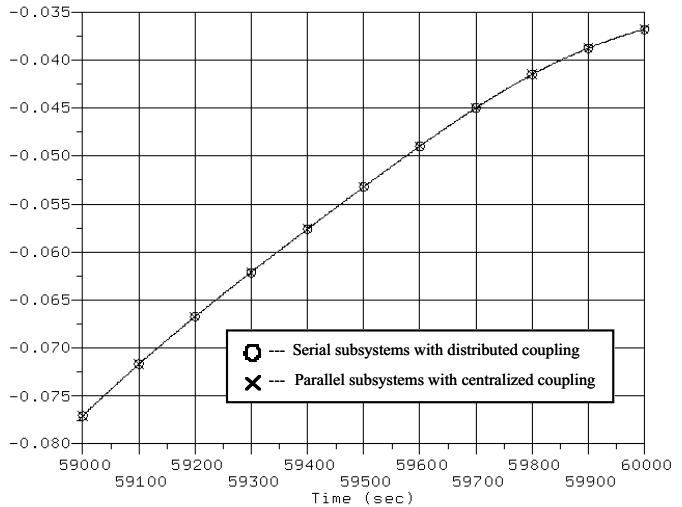
Case 3, Body Rate, x-Axis (deg/sec)



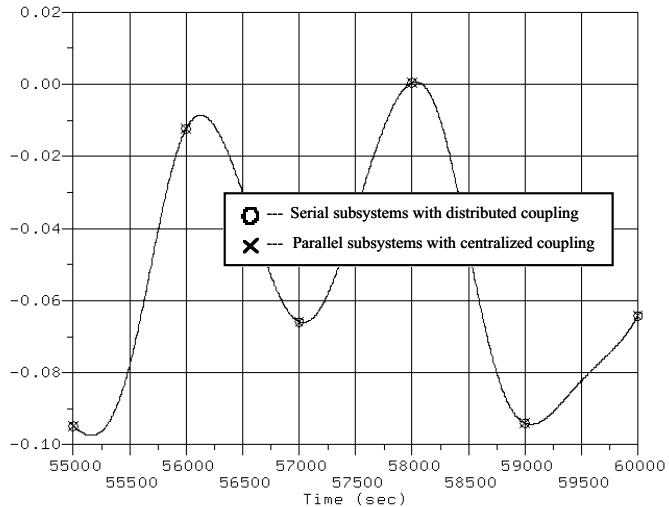
Case 3, Body Rate, y-Axis (deg/sec)



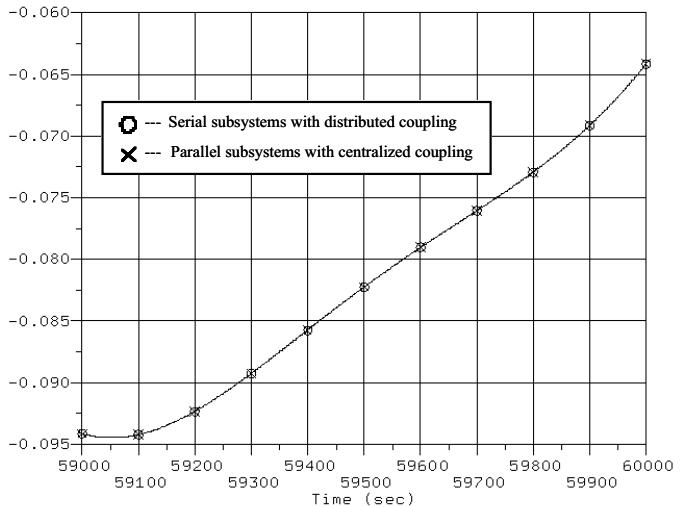
Case 3, Body Rate, y-Axis (deg/sec)



Case 3, Body Rate, z-Axis (deg/sec)



Case 3, Body Rate, z-Axis (deg/sec)



APPENDIX B—FORTRAN LISTINGS

B.1 FORTRAN Listings, Including Main Program and All Subroutines, Recursive Feedback With Centralized Coupling (RFC)

```

C   This is FORTRAN main program "rfc.f". Development began 6/12/2000 by Doug Nixon, MSFC/TD55,
C   in support of AEI project, and the listing is contained in "aei/rfc"
C
C   The purpose is to generate the dynamic solution of three coupled subsystems (orbit dynamics,
C   attitude dynamics, and aerodynamics) by recursive feedback, which involves distributed and
C   possibly multiple methods of numerical integration. The intent is to compare results generated
C   by a standard approach using a centralized single numerical integration method. The standard
C   approach has been implemented as FORTRAN program "rku.f", and the main program is contained
C   in "aei/rku"
C
C   Declarations *****
C
C   implicit double precision(a-h,o-z)
C
C   dimension uu(18,100),yy(18,100),yyp(18,100)      !18 input and 18 output signals, 100 time points
C
C   dimension uu1(6,100),yy1(6,100)                  !input and output signals for three subsystems,
C   dimension uu2(6,100),yy2(6,100)                  !six signals per subsystem, 100 time points
C   dimension uu3(6,100),yy3(6,100)
C
C   dimension yy1i(6),yy2i(6),yy3i(6)              !initial conditions, 3 subsystems, 6 outputs
C   dimension yy1f(6),yy2f(6),yy3f(6)              !final conditions, 3 subsystems, 6 outputs
C
C   common/system/uu,yy,yyp                         !system input/output/"previous" output, all vs. time
C
C   common/subsys1/uu1,yy1                          !orbit dynamics, i/o vs. time
C   common/endcond1/yy1i,yy1f                      !initial and final output conditions
C
C   common/subsys2/uu2,yy2                          !attitude dynamics, i/o vs. time
C   common/endcond2/yy2i,yy2f                      !initial and final output conditions
C
C   common/subsys3/uu3,yy3                          !aerodynamics, i/o vs. time
C   common/endcond3/yy3i,yy3f                      !initial and final output conditions
C
C   common/simulation/ndp,delt,tole,res,t_start,    !all except some single-time-point utility routines
C   ltr_report,t_print,t_stop,nmt,nmtr,nit,

```

```

C 2t_initial,t_final
C common/simulation2/del_report,del_print,ptol !percent done report interval, print interval
C common/constants/rearth,gme,pi,cvdr !recursive.f, orbit.f,subl, attitude.f, sub2.f

C *** Definition of the fundamental variables ****
C
C uu(i,j) ----- System-level feedback signal. Represents the ith component at the jth time
C point for the total system vector feedback signal. Dimension is that of the
C total system, and all subsystem input vectors are contained within appropriate
C partitions. Data for the first subsystem is contained in the first (top)
C sub-column. In a more general setting, this signal would be summed with a
C system-level input signal vv(i,j) to form the subsystem input signals. vv(i,j)
C is not needed for this implementation, and has not been included.

C yy(i,j) ----- System-level output signal. Represents the ith component at the jth time
C point for the system vector output signal. Dimension is that of the total
C system, and all subsystem output vectors are contained within appropriate
C partitions. Data for the first subsystem is contained in the first (top)
C partition.

C *****

C uu1(i,j) ----- First subsystem input signal. Represents the ith component at the jth time
C point for the vector input signal to the first (orbit dynamics) subsystem,
C and is also the first (top) partition of uu(i,j).

C uu2(i,j) ----- Second subsystem input signal. Represents the ith component at the jth time
C point for the vector input signal to the second (attitude dynamics) subsystem,
C and is also the second (middle) partition of uu(i,j).

C uu3(i,j) ----- Third subsystem input signal. Represents the ith component at the jth time
C point for the vector input signal to the third (aerodynamics) subsystem, and
C is also the third (bottom) partition of uu(i,j).

C *****

C yy1(i,j) ----- First subsystem output signal. Represents the ith component at the jth time

```

```

c point for the vector output signal from the first (orbit dynamics) subsystem,
c and is also the first (top) partition of YY(i,j).
c
c YY2(i,j) ----- Second subsystem output signal. Represents the ith component at the jth time
c point for the vector output signal from the second (altitude dynamics) subsystem,
c and is also the second (middle) partition of YY(i,j).
c
c YY3(i,j) ----- Third subsystem output signal. Represents the ith component at the jth time
c point for the vector output signal to the third (aerodynamics) subsystem, and is
c also the third (bottom) partition of YY(i,j).
c
c *****-----*
c
c YY1i(i) ----- Value of the output signal at the beginning of the current convergence interval
c for the FIRST (orbit dynamics) subsystem. Represents the ith component of the
c signal at the first time point, and is equivalent to YY1(i,1)
c
c YY2i(i) ----- Value of the output signal at the beginning of the current convergence interval
c for the SECOND (altitude dynamics) subsystem. Represents the ith component of the
c signal at the first time point, and is equivalent to YY2(i,1)
c
c YY3i(i) ----- Value of the output signal at the beginning of the current convergence interval
c for the THIRD (aerodynamics) subsystem. Represents the ith component of the signal
c at the first time point, and is equivalent to YY3(i,1)
c
c YY1f(i) ----- Value of the output signal at the end of the current convergence interval for
c the FIRST (orbit dynamics) subsystem. Represents the ith component of the signal
c at the last time point, and is equivalent to YY1(i,npd)
c
c YY2f(i) ----- Value of the output signal at the end of the current convergence interval for
c the SECOND (altitude dynamics) subsystem. Represents the ith component of the
c signal at the last time point, and is equivalent to YY2(i,npd)
c
c YY3f(i) ----- Value of the output signal at the end of the current convergence interval for
c the THIRD (aerodynamics) subsystem. Represents the ith component of the vector
c signal YY3(i,j) at the last time point, and is equivalent to YY3(i,npd)
c
c

```

```

C **** General Comments ****
C
C The above nomenclature has been designed to allow implementation of recursive feedback via
C FORTRAN in such a way that independence of the subsystem simulations can be supported. The
C partitions and contractions facilitate autonomy of the subsystem simulations while providing
C appropriate structure for implementation of the system-level algorithm. The fact that subsystem
C data has been placed in COMMON with the MAIN (system level) program does not necessarily mean
C that all of that data is required or desired at the system level -- it may mean only that the
C SUBROUTINE containing the subsystem simulation needs to have access to data that it generated
C previously, and placing that data in COMMON is a way to provide that in FORTRAN.
C
C While the subsystem simulation SUBROUTINES will not literally be executed in parallel, the
C information flows in parallel paths, and the order of execution is not important. Therefore,
C parallel execution is conceptually possible without impact to the structure of the process.
C
C This program, which involves three subsystems, can serve as a template for implementation with any
C number of subsystems. Use of higher order arrays to implement the general case for "n" subsystems
C appears to serve no useful purpose in this setting at this time.
C
C This implementation assumes that all subsystem states are needed at the system level, and transfer of
C final conditions to initial conditions after convergence is handled at that level. For this specific
C system, the assumption is valid; however, in general, subsystem states can exist and remain at the
C subsystem level without the necessity of knowing them at the system level. Because of this,
C it would be more appropriate to define and "bookkeep" initial conditions at the subsystem level for a
C more general implementation. In the more general implementation, transfer of final conditions from
C one convergence interval to initial conditions of the next would be handled totally at the subsystem
C level upon receipt of "command" from the system level.
C
C ***
C   Establishing the output file ****
C
C   open(unit=1,file='rfc.out1',status='unknown')
C
C ***
C   Defining system-level simulation parameters ****
C
C   print*
C   print*, 'This is main program rfc.f. Solution'
C   print*, 'method is recursive feedback with'
C   print*, 'centralized coupling.'

```

```

print*, ! Input the number of data (time) points'
print*, ! per convergence interval, including,
print*, ! both end (boundary) points:
read*, ndp
print*, ! Input the convergence interval duration'
print*, ! (sec.):
read*, dtmaj

C   fac=1.0/float(ndp-1)           !ratio of integration time step to convergence interval duration
delt=fac*dtmaj                     !integration stepsize

C   print*, ! The above data limits all integration'
print*, ! stepsizes to 'delt,' sec.'
print*, ! or less'
print*
print*

C *** Populating the subsystem databases ****
C   call orbit(1)                  !defines parameter values (data) for subsystem 1 (orbit dynamics, mode 1)
C   call attitude(1)               !defines parameter values (data) for subsystem 2 (orbit dynamics, mode 1)
C   call aerodyn(1)                !defines parameter values (data) for subsystem 3 (orbit dynamics, mode 1)

C   print*, ! Input the simulation stop time (sec.):
read*, t_stop
print*, ! Input the data output(print) interval'
print*, ! (sec.):
read*, del_print
print*, ! Input percent done status interval'
print*, ! (%):
read*, per_report
del_report=(per_report/100.0)*t_stop

C   C   nmtr=int(t_stop/dtmaj)+1    !number of major time (convergence) intervals required for completion of
C   !the simulation. (Time may exceed t_stop by as much as one major interval)

```

```

C          tol=(1.0D0)*1.0e-12      !relative
C          res=(1.0D0)*1.0e-12      !absolute
C          itmax=100                !maximum allowable number of iterations
C          ptol=1.0e-3               !tolerance for print time
C          print*, ' Run in progress.'
C          print*
C          call clock(n1)           !machine start time
C
C          *** Initializing time and convergence interval count: ****
C
C          time=0.0                 !time is measured from start of the simulation
C          nmt=0                    !number of major time intervals (convergence intervals) simulated
C          t_print=0.0               !initializes print time
C          t_report=0.0              !initializes percent done report time
C
C          *** Setting up initial conditions for each sub-simulation at system simulation outset
C
C          call orbit(2)            !mode 2, original initial conditions setup, orbit dynamics
C
C          call attitude(2)          !mode 2, original initial conditions setup, attitude dynamics
C
C          !no initial conditions setup for aerodynamics (algebraic system)
C
C          *** COMPUTING INITIAL RECURSION STAGE DATA ****
C          100 continue !*****COMPUTING INITIAL RECURSION STAGE DATA*****
C
C          nit=0 !number of iterations initialized to zero for first (new) convergence interval
C
C          *** Initializing the system-level feedback signal to zero: ****
C
C          do 1 i=1,18                !eighteen scalar feedback signals
C          do 1 j=1,ndp               !ndp number of data (time) points per scalar signal
C          uu(i,j)=0.0D0             !algorithm ignores feedback at recursion stage number (0)
C          1 continue
C
C          *** Decomposing the system-level feedback vector into individual subsystem input vectors

```

```

C   call decomp(6,ndp,uu,uul,uu2,uu3) !six scalar input signals per subsystem
C
C *** Running individual subsystem simulations over the convergence interval
C
C   call orbit(3)      ! mode 3, run orbit dynamics simulation over convergence interval
C
C   call attitude(3)   ! mode 3, run attitude dynamics simulation over convergence interval
C
C   call aerodyn(3)    ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
C
C *** Assemble subsystem outputs into single system-level output *****
C
C   call assemble(6,ndp,yy1,yy2,yy3,yy)
C
C   call coupler(yy,uu) !computes system-level feedback signal (subsystem coupling)
C
C   do 2 i=1,18        !sets initial recursion stage data to current values
C   do 2 j=1,ndp
C     YYP(i,j)=YY(i,j)
C   2 continue
C
C *** RECURSION LOOP ENTRY POINT *****
C 200 continue !*****
C *** Decomposing the system-level feedback vector into individual subsystem input vectors
C
C   call decomp(6,ndp,uu,uul,uu2,uu3) !six scalar input signals per subsystem
C
C *** Running individual subsystem simulations over the convergence interval
C
C   call orbit(3)      ! mode 3, run orbit dynamics simulation over convergence interval
C
C   call attitude(3)   ! mode 3, run attitude dynamics simulation over convergence interval
C
C   call aerodyn(3)    ! mode 3, run aerodynamics (algebraic) simulation over convergence interval

```

```

C   C *** Assemble subsystem outputs into single system-level output ****
C
C   call assemble(6,ndp,yy1,yy2,yy3,yy)
C
C   *** Check for convergence ****
C
C   call converge(18,yy,yyp,iconflg)          ! system-level convergence check
C                                             ! iconflg=0 if converged, 1 if
C                                             ! notconverged
C
C   indirect=iconflg+1
500  format ('/2i5,/')
go to (6,5) indirect          !6 for converged, 5 for not converged
C
C   5 continue ! **** Not CONVERGED, ADVANCE RECURSION STAGE ****
C
C   call coupler(yy,uu)          ! computes system-level feedback signal (subsystem coupling)
C
do 7 i=1,18          ! number of scalar signals in the system
do 7 j=1,ndp          ! number of time points over each convergence interval,
C                                             ! includes both end points
C
    yyp(i,j)=yy(i,j)          ! sets previous recursion stage data to current values
7 continue
nit=nit+1          !updating the iterations count (number of recursion stages)
C
if(nit.ge.itmax) go to 300          !protection against persistent non convergence
C
go to 200          !returns to "top" of the recursion loop to run subsystem simulations again
C
C   6 continue ! ** CONVERGED, ADVANCE TO NEXT CONVERGENCE INTERVAL (or stop if simulation is complete) ***
C
C   call rfc_out          !appends system-level output signals to output file (recursive.out)
nmt=nmt+1          !increments number of major time (convergence) intervals completed
time=nmt*dtmaj      !simulated problem time , for reference only

```

```

C      if(nmt.eq.nmtr) go to 300 !end of run, time is greater than or equal to the stop time (t_stop)
C ****
C
C      *** Proceed ("hand-off") to next major time (convergence) interval
C
C      In a more general setting, this would be done at the subsystem level (mode 4) so that the system
C      level is not required to have knowledge of all subsystem states.
C
C      do 8 i=1,6
C      !extracts subsystem final conditions for the current convergence interval
C      YY1f(i)=YY(i,npd)
C      !from the final time point of the system response associated with the current
C      YY2f(i)=YY(i+6,npd)
C      !convergence interval. SEE PARAGRAPH NUMBER 4 of "General Comments" ABOVE.
C      YY3f(i)=YY(i+12,npd)
C
C      8 continue
C
C      do 9 i=1,6
C      !sets initial conditions (for all states) for the next convergence interval
C      YY1i(i)=YY1f(i)
C      !equal to conditions reached at the end (final conditions) of current
C      YY2i(i)=YY2f(i)
C      !convergence interval.
C      YY3i(i)=YY3f(i)
C
C      9 continue
C
C      go to 100
C      !run subsystem simulations for the first time across the new (next)
C      !convergence interval.
C
C      300 continue
C
C      if(nit.ge.itmax) print*, 'Max. allowable number of '
C      if(nit.ge.itmax) print*, 'iterations reached.'
C
C      call clock(n2) !machine stop time
C      irun=n2-n1 !run-time
C
C      if(nmt.eq.nmtr) print*, ' Normal exit, run-time = ',irun,' sec.'
C      print*
C
C      close(unit=1,status='keep')
C
C      stop

```



```

dimension aproj(6),drag(6),facefor(6,3),facetor(6,3)
dimension tforce(3),ttorg(3)

c   common/aerobbox/cpfront,cpright,cpbottom,          !aerodyn.f, aerocompr.f
    1cprear,cpleft,cptop,afront,aright,abottom,
    2arear,aleft,atop

c   common/aerodat/rho,cd                                !aerodyn.f, aerocompr.f

c   **** Aerodynamic forces ***
c   call vnorm(vel_b,vnorm_b,vmag)                      ! normalize velocity vector in body
c   ! coordinates

c   *** computing magnitude drag for each of six facets
c
c   call vdotpro(afront,vnorm_b,aproj(1))               ! computing dot product of each area
c   call vdotpro(aright,vnorm_b,aproj(2))                ! vector with normalized velocity
c   call vdotpro(abottom,vnorm_b,aproj(3))              ! (projected areas)
c   call vdotpro(arear,vnorm_b,aproj(4))
c   call vdotpro(aleft,vnorm_b,aproj(5))
c   call vdotpro(atop,vnorm_b,aproj(6))                 ! these statements define the facet
c   ! numbering system

c   do 20 i=1,6                                         ! negative projection implies ith facet
      if(aproj(i).le.0.0D0) aproj(i)=0.0D0             ! not exposed to velocity vector
20  continue

c   fac=(0.5D0)*rho*(vmag**2)*cd                      ! scaling by dynamic pressure times
do 21 i=1,6
  drag(i)=fac*aproj(i)                                ! drag coefficient
21  continue

c   do 22 i=1,6
do 22 j=1,3
  facefor(i,j)=-drag(i)*vnorm_b(j)                  ! this is jth component of force on the ith

```

```

22 continue
      ! facet, expressed in body coordinates

C
do 23 i=1,3
force(i)=0.0D0
do 23 j=1,6
force(i)=force(i)+facefor(j,i) ! total aerodynamic force, body coordinates, newtons
23 continue

C *** Aerodynamic Torques *****
C
do 30 i=1,3
tforce(i)=facefor(1,i) ! front facet contribution
30 continue
call vcross(cpfront,tforce,ttorq)
do 31 i=1,3
facotor(1,i)=ttorq(i)
torque(i)=torque(i)+ttorq(i)
31 continue

C
do 32 i=1,3
tforce(i)=facefor(2,i) ! adding right facet contribution
32 continue
call vcross(cpright,tforce,ttorq)
do 33 i=1,3
facotor(2,i)=ttorq(i)
33 continue

C
do 34 i=1,3
tforce(i)=facefor(3,i) ! adding bottom facet contribution
34 continue
call vcross(cpbottom,tforce,ttorq)
do 35 i=1,3
facotor(3,i)=ttorq(i)
35 continue

C
do 36 i=1,3
tforce(i)=facefor(4,i) ! adding rear facet contribution
36 continue

```

```
call vcross(cprear,tforce,ttorq)
```

```
do 37 i=1,3
```

```
facetor(4,i)=ttorq(i)
```

```
37 continue
```

```
c
```

```
do 38 i=1,3
```

```
tforce(i)=facefor(5,i)
```

```
38 continue
```

```
call vcross(cpleft,tforce,ttorq)
```

```
do 39 i=1,3
```

```
facetor(5,i)=ttorq(i)
```

```
39 continue
```

```
c
```

```
do 40 i=1,3
```

```
tforce(i)=facefor(6,i)
```

```
40 continue
```

```
call vcross(cptop,tforce,ttorq)
```

```
do 41 i=1,3
```

```
facetor(6,i)=ttorq(i)
```

```
41 continue
```

```
c
```

```
do 42 i=1,3
```

```
torque(i)=0.0D0
```

```
do 42 j=1,6
```

```
torque(i)=torque(i)+facetor(j,i)
```

```
42 continue ! total aerodynamic torque, body
```

```
c ! coordinates, newton-meters
```

```
c
```

```
return
```

```
end
```

```
c
```

```
XXXXXXXXXXXXX
```

```

subroutine aerodyn(mode) ! mode 1= set physical constants, system parameters, etc.
! mode 2= blank, no initial conditions for algebraic system
! mode 3= run simulation over the convergence interval
C
C *** This is subprogram "aerodyn.f". Construction began 6/21/2000 by Doug Nixon, TD55, MSFC.
C Its purpose is to generate aerodynamic forces and torques as a function of time when
C given an orbit state vector and vehicle attitude definition as a function of time. It is
C an algebraic system; no numerical integration is involved.

C implicit double precision (a-h,o-z)

C dimension uu3(6,100),yy3(6,100)

C dimension rad_b(3),vel_b(3)
C dimension force(3),torque(3)

C dimension cpfront(3),cpright(3),cpbottom(3) !box panel data
C dimension cprear(3),cpleft(3),cptop(3)
C dimension afront(3),aright(3),abottom(3)
C dimension arear(3),aleft(3),atop(3)

C common/subsys3/uu3,yy3
C common/endcond3/yy3i,yy3f

C common/simulation/ndp,delt,tol,res,t_start,
C 1t_report,t_print,t_stop,nmtr,nit,
C 2t_initial,t_final

C common/aero/box/cprear,cpleft,cptop,abottom,
C 1arear,aleft,atop
C common/aerodat/rho,cd

C if(mode.eq.2.or.mode.eq.3) go to 200

```

```

C 100 continue ! (mode=1, define constants, system parameters)
C **** Setting Values of System Parameters (mode 1 function): ****
C
C *** Density and drag coefficient: ***
C
C rho=1.95E-11 ! kg/m**3, Wertz, inside back cover, 300km altitude , mean value
C
C cd=2.25
C           ! drag coefficient, same for all facets
C
C *** "Panel" aerodynamics setup (six-panel box): ***
C
C x1=15.0D0
C           ! length, meters (box shape, x-axis dimension)
C w=4.0D0
C           ! width, y-axis dimension
C h=3.0D0
C           ! height, z-axis dimension
C ax=w*h
C           ! end area, normal to x-axis (+ "front")
C ay=x1*h
C           ! side area, normal to y-axis (+ "right side")
C az=x1*w
C           ! top or bottom area, normal to (+ z-axis "bottom")
C
C Work with six facets ----
C
C do 10 i=1,3
C           ! setting all vector components to 0.0D0
C
C afront(i)=0.0D0
C           ! area vectors, normal to facet surfaces, through
C arear(i)=0.0D0
C           ! centers, positive (point) outward from center of
C aright(i)=0.0D0
C           ! box, back sides not considered (no contribution)
C aleft(i)=0.0D0
C atop(i)=0.0D0
C abottom(i)=0.0D0
C
C cpfront(i)=0.0D0
C           ! centers of pressure location vectors, from geometric
C cprear(i)=0.0D0
C           ! center of box to center of each facet
C cpright(i)=0.0D0
C cpleft(i)=0.0D0
C cptop(i)=0.0D0
C cpbottom(i)=0.0D0
C

```

```

10 continue
C
      afront(1)=ax      ! setting non-0.0D0 components of facet area vectors
      arear(1)=-ax
      aright(2)=ay
      aleft(2)=-ay
      abottom(3)=az
      atop(3)=-az

C
      cpfront(1)=x1/2.0D0      !setting non-zero components of facet
      cprear(1)=-x1/2.0D0      !centers-of-pressure location vectors
      cpright(2)=w/2.0D0
      cpleft(2)=-w/2.0D0
      cptop(3)=-h/2.0D0
      cpbottom(3)=h/2.0D0

C
      return

C
C ***** End of mode 1, beginning of mode 2 *****
C ***** End of mode 2, beginning of mode 3 *****
C ***** End of mode 3, beginning of mode 4 *****

C
200 if(mode.eq.3) go to 300
      continue ! (mode=2, normally sets initial conditions for dynamic system)

C
      print*, ''
      print*, 'This sub-simulation (aerodynamics) is'
      print*, 'algebraic; therefore initial conditions'
      print*, 'are not applicable.'
      print*, ''
      print*, 'Enter 0 to continue, 1 to stop'
      read*, igo
      if(igo.eq.1) stop

C
      return

C
C ***** End of mode 4, beginning of mode 5 *****
C ***** End of mode 5, beginning of mode 6 *****
C ***** End of mode 6, beginning of mode 7 *****
C ***** End of mode 7, beginning of mode 8 *****
C ***** End of mode 8, beginning of mode 9 *****
C ***** End of mode 9, beginning of mode 10 *****

```

```

C *****
C
C   300 continue ! (mode=3, run simulation over the convergence interval)
C   **** Propagate Output Signals (mode 3 function) : ****
C
C
C   begin propagation loop
C
C   do 20 ip=1,ndp      ! propagates forces and torques across convergence interval,
C
C   do 6 j=1,3
C     rad_b(j)=uu3(j,ip) ! extracts radius and velocity vectors at the i'th time point from the
C     vel_b(j)=uu3(j+3,ip) ! subsystem 3 input signal array. Radius is expressed in inertial coordinates,
C   6 continue             ! velocity in body coordinates. Radius is not used now because density is
C                           ! defined to be constant --- will be used in later version.
C
C   call aerocompr(vel_b,force,torque) ! computes force and torque at a single time point
C
C   do 7 j=1,3           ! enters output (force and torque vectors) into block common /susbsy1/.
C     yy3(j,ip)=force(j)
C     yy3(j+3,ip)=torque(j)
C   7 continue
C
C   20 continue
C
C   end propagation loop
C
C   return
C
C
C   **** End of mode 3 ****
C   ****
C
C

```



```

C dimension uu2(6,100),yy2(6,100)          !I/O for subsystem 2 (altitude dynamics)
C dimension yy2i(6),yy2f(6)                !initial and final output conditions, subsystem 2
C dimension x(6),xdot(6)                  !state variables for altitude dynamics
C dimension phi(3),omega(3)               !altitude angles and body rates (state variables)
C dimension xmoi(3,3),xmoi_inv(3,3),rcm(3) !moments of inertia, about c.m., c.m. location vector
C dimension xmoi(3,3),xmoi_inv(3,3),rcm(3) !in geometric body coordinates
C common/subsys2/uu2,yy2                  !recursive.f, attitude.f, sub2.f
C common/endcond2/yy2i,yy2f              !recursive.f, attitude.f
C common/simulation/ndp,delt,tol,res,t_start, !all except some single-time-point utility routines
C                                         !recursive.f, orbit.f, sub1, attitude.f, sub2.f
C                                         !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
C                                         !attitude.f, sub2.f
C                                         !recursive.f, orbit.f, sub1, attitude.f, sub2.f
C                                         !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
C common/gains/a0,a1,b0,b1,c0,c1
C if(mode.eq.2.or.mode.eq.3) go to 200
C 100 continue ! (mode=1, define constants, system parameters)
C ***** Numerically Defining Constants (mode 1 function) : *****
C ***** Gravity constant (gme) is defined by orbit.f, and is obtained through block
C common /constants/; pi (3.14xx...) and cvdr (deg. to rad. mult. factor) are
C are defined and obtained similarly.
C

```

```

C **** Setting Values of System Parameters (mode 1 function) : ****
C
C Satellite mass (xmass) is defined by orbit.f; obtained here through common/mass/.
C
C *** Moments of inertia and center of mass:
C
C
C xmoi(1,1)=11250.0D0          ! spacecraft moments of inertia,
C xmoi(1,2)=0.0D0                ! kilogram-meters**2
C xmoi(1,3)=0.0D0
C xmoi(2,1)=0.0D0
C xmoi(2,2)=193125.0D0          ! 3m diameter x 15m length uniform-
C xmoi(2,3)=0.0D0                ! density right-circular cylinder
C xmoi(3,1)=0.0D0
C xmoi(3,2)=0.0D0
C xmoi(3,3)=193125.0D0* .9      ! IZZ=.9*IYY
C call inv3(xmoi,xmoi_inv)       ! inverse matrix
C
C rcm(1)=1.0D0                  ! center of mass location in geometric body
C rcm(2)=0.1D0                  ! coordinates, meters
C rcm(3)=0.1D0
C
C *** Control system gains:
C
C a0=1000.0D0                  ! control gain, x-axis position, n-m/rad
C a1=100.0D0                     ! control gain, x-axis rate, n-m-sec/rad
C b0=1000.0D0                     ! y-axis
C b1=100.0D0
C c0=1000.0D0
C c1=100.0D0
C
C print*, ' Input stepsize reduction factor'
C print*, ' for ATTITUDE DYNAMICS simulation'
C print*, ' (integer): '
C read*, idiv
C return
C
C ****

```

```

C **** End of mode 1, beginning of mode 2 ****
C **** Setting Initial Conditions (mode 2 function) : ****
C ****
C
C 200 if(mode.eq.3) go to 300
  continue ! (mode=2, set initial conditions)
C
C **** Setting Initial Conditions (mode 2 function) : ****
C ****
C
C phi(1)=0.0D0          ! initial values of the Euler angles (radians)
C phi(2)=0.0D0
C phi(3)=0.0D0

C
C omega(1)=0.0D0        ! initial values of the body rates (radians)
C omega(2)=0.0D0
C omega(3)=0.0D0

C
C do 5 i=1,3           !entering initial conditions into block common /endcond2/ (radians)
C   YY2i(i)=phi(i)
C   YY2i(i+3)=omega(i)
  5 continue

C
C   return
C
C **** Beginning of mode 3 ****
C ****
C
C 300 continue ! (mode=3, run simulation over the convergence interval)
C
C **** Setting up the initial state vector (6 states, mode 2 function) : ****
C ****
C
C do 6 i=1,3           !retrieving initial conditions from block common /endcond2/ (radians)

```

```

phi(i)=yy2i(i)
omega(i)=yy2i(i+3)

C   YY2(i,1)=phi(i)           !entering initial output vector (time point number 1) into common/subsys2/
C   YY2(i+3,1)=omega(i)

C   6 continue

C
C   do 10 i=1,3               ! changing from kinematic to state variable notation
C   x(i)=phi(i)                ! states 1-3, Euler angles, radians
C   x(i+3)=omega(i)            ! states 4-6, body rates, rad/sec
10 continue

C   print*, 'from att.f*****'
C   print*, 'uu2', (uu2(1,j),j=1,ndp)
C   print*
C   print*, ' ', (uu2(2,j),j=1,ndp)
C   print*
C   print*, ' ', (uu2(3,j),j=1,ndp)
C   print*

C   tau=0.0D0 !initializing time, measured from the start of the convergence interval
C   !                                         !
C   ndim=6 ! defining dimension of the state vector

C   continue !***** begin integration loop *****
C   do 20 i=2,ndp               ! generates points 2 through ndp
C
C   dt=delt/float(idiv)
C   do 30 ii=1,idiv
C   call rksys2(tau,x,ndim,dt) ! propagates attitude states across convergence interval
30 continue

C   do 7 j=1,6                  ! enters output (Euler angles and body rates) into common/susbsy2/
C   YY2(j,i)=x(j)
7 continue

C

```

112

```

implicit double precision(a-h,o-z)
c
dimension yy(18,100),yyp(18,100)
dimension ynew(100),yold(100)
c
common/simulation/ndp,delt,tol,res,t_start,t_report,t_print,
1t_stop,nmt,nmtr,nit,t_initial,t_final
c
c **** begin check loop ****
c **** comparison (convergence check) ****
c
iconflag=0           ! initializes system-level flag to "converged"
do 20 i=1,nsysdim  !(nsysdim=18). Loads current and previous scalar signals for
do 19 j=1,ndp       !comparison (convergence check).
c
ynew(j)=yy(i,j)
yold(j)=yyp(i,j)
c
19 continue
c
call convrg(ynew,yold,ndp,tol,res,iflag) ! iflag=1 for non-convergence.
c
iconflag=iconflag+iflag !computes status of system-level convergence
c
20 continue
c
if(iconflag.gt.0) iconflag=1 !routine returns 0 or 1 for iconflag
c
c **** end check loop ****
c **** comparison (convergence check) ****
c
c
return
c
end

```



```

      IF (YNEW ( I ) .LT. YMIN)  YMIN=YNEW ( I )
11  CONTINUE

C           SPAN=dABS (YMAX-YMIN)
C           CHECK2=TOL*SPAN
C
C           IF (CHECK1 .LE .CHECK2.OR.CHECK1 .LE .RES)  IFLAG=0
C
C           RETURN
C
C           END
C
C
C           subroutine coupler(yy,uu)
C
C           implicit double precision(a-h,o-z)
C
C           dimension uu(18,100),yy(18,100) !system-level I/O, 18 input and 18 output signals, 100 time points
C
C           dimension f_aero(3)           !aerodynamic force (drag), inertial coordinates
C           dimension f_aero_b(3), t_aero_b(3)   !aerodynamic force and torque, body frame
C           dimension f_aero_g(3), t_aero_g(3)   !aerodynamic force and torque, geometric frame
C           dimension rxf(3)               !aerodynamic force transfer couple (from geometric origin to c.m.)
C           dimension rad(3),vel(3)         !orbit radius and velocity vectors, inertial coordinates
C           dimension rad_b(3),vel_b(3)       !orbit radius and velocity vectors, body coordinates
C           dimension phi(3)              !Euler angles, inertial frame to body frame, 3-2-1 sequence
C           dimension amat(3,3)            !direction cosine matrix (d.c.m.), inertial to body
C           dimension amat_t(3,3)          !transpose of amat(i,j)
C           dimension xmoi(3,3),xmoi_inv(3,3) !inertia matrix, inverse inertia matrix
C           dimension rcm(3)              !center of mass location, geometric coordinates

```

```

C   common/simulation/ndp,delt,tol,res,t_start, !all except some single-time-point utility routines
C   lt_report,t_print,t_stop,nmt,nmtr,nit,
C   2t_initial,t_final
C   common/mass/xmass,xmoi,xmoi_inv,rcm          !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
C
C   **** Coupling Aerodynamics/Orbit Dynamics ****
C   ****
C   ****
C
C   do 10 j=1,ndp
C
C   *** from aerodynamics to orbit dynamics:
C
C   f_aero_b(1)=yy(13,j)      !extracting aerodynamic (drag) force from system output signal
C   f_aero_b(2)=yy(14,j)      !computed to act at origin of geometric body frame,
C   f_aero_b(3)=yy(15,j)      !but can be assumed to act at c.m. for this purpose
C
C   phi(1)=yy(7,j)            !extracting attitude (Euler) angles from the system output signal
C   phi(2)=yy(8,j)
C   phi(3)=yy(9,j)
C
C   call dcm321(phi,amat)     !computes d.c.m that transforms vectors from inertial to body coordinates
C   call trans3(amat,amat_t)   !computes d.c.m that transforms vectors from body to inertial coordinates
C
C   call matvec(amat_t,f_aero_b,f_aero) !transforms force vector from body to inertial coordinates
C
C   uu(1,j)=f_aero(1)          !feedback signal is aerodynamic force (drag) in inertial coordinates,
C   uu(2,j)=f_aero(2)          !and is assumed to act at the satellite center of mass (c.m.)
C   uu(3,j)=f_aero(3)
C
C   uu(4,j)=0.0D0              !slot holders, nothing yet identified
C   uu(5,j)=0.0D0
C   uu(6,j)=0.0D0
C
C   *** from orbit dynamics to attitude dynamics:
C

```

```

rad(1)=yy(1,j)
rad(2)=yy(2,j)
rad(3)=yy(3,j)

C
uu(7,j)=rad(1)
uu(8,j)=rad(2)
uu(9,j)=rad(3)

C *** from aerodynamics to attitude dynamics:
C
f_aero_g(1)=yy(13,j) !aerodynamic (drag) force; acts at origin of geometric body frame.
f_aero_g(2)=yy(14,j) !f_aero_g(i) and f_aero_b(i) are numerically identical, but their
f_aero_g(3)=yy(15,j) !physical interpretations (points of action) may be different

C
t_aero_g(1)=yy(16,j) !aerodynamic torque with force at origin of geometric body frame
t_aero_g(2)=yy(17,j)
t_aero_g(3)=yy(18,j)

C
call vcross(rcm,f_aero_g,rxf) ! torque of drag force about c.m.

C
t_aero_b(1)=t_aero_g(1)+rxf(1) ! transferring drag force to c.m. and adding an
t_aero_b(2)=t_aero_g(2)+rxf(2) ! equivalent couple.
t_aero_b(3)=t_aero_g(3)+rxf(3)

C
uu(10,j)=t_aero_b(1)
uu(11,j)=t_aero_b(2)
uu(12,j)=t_aero_b(3)

C *** from orbit dynamics to aerodynamics
C
call matvec(amat,rad,rad_b) !transforms orbit radius vector to body coordinates

C
uu(13,j)=rad_b(1) !feedback signal is orbit radius vector in body coordinates
uu(14,j)=rad_b(2)
uu(15,j)=rad_b(3)

C
vel(1)=yy(4,j) !variable vel(i) is used for listing readability only

```



```
C dimension xx(18,100),xx1(6,100),xx2(6,100),xx3(6,100)
C Assumes nsubdim number of scalar signals per subsystem
C
C do 10 i=1,nsubdim               !eighteen scalar signals at the system level,
C do 10 j=1,ndp                     !ndp time points each
C   xx1(i,j)=xx(i,j)
C   xx2(i,j)=xx(i+6,j)
C   xx3(i,j)=xx(i+12,j)
C 10 continue
C
C   return
C end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

xxx

```
subroutine interp(ndp,tstart,delt,time,func,val)

C *** Linearly interpolates func(t) data for a particular value of time.
C implicit double precision(a-h,o-z)
C dimension func(100) !single scalar signal, 100 time points
interval=1
t1=tstart

C 1 if(time.lt.t1+delt) go to 3
C 2 continue                           ! continuing search for the interval
                                           interval=interval +1
                                           t1=t1+delt
```



```
TEMP(3,3)=A(1,1)*A(2,2)-A(2,1)*A(1,2)
DO 10 I=1,3
DO 10 J=1,3
 10 AI(I,J)=TEMP(J,I)/DET
RETURN
END
```

XX
XX

SUBROUTINE MATVEC (A,V,AV)

```
C
C *** THIS ROUTINE MULTIPLIES A VECTOR BY A MATRIX.
C   (3X3) * (3X1) = (3X1)
C
C IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
C DIMENSION A(3,3),V(3),AV(3)
C
C DO 10 I=1,3
C     AV(I)=0.0
C     DO 10 J=1,3
C       10 AV(I)=AV(I)+A(I,J)*V(J)
C
C RETURN
C END
```

XX
XX

```

subroutine orbit(mode) ! mode 1= set physical constants, system parameters, etc.
! mode 2= set initial conditions
! mode 3= run simulation over the convergence interval
C
C *** This is subprogram "orbit.f". Construction began 6/19/2000 by Doug Nixon, TD55, MSFC.
C "orbit.f" is the main driver for the six-state dynamic system which is integrated
C by "rkssys.f", a fourth-order Runge-Kutta subroutine. "sub1.f" defines the
C dynamics in first-order state-variable form.
C
C implicit double precision (a-h,o-z)
C
C dimension uul(6,100),yy1(6,100)
C dimension yy1i(6),yy1f(6)
C
C dimension r(3),v(3)
C dimension x(12),xdot(12)
C dimension xmoi(3,3),xmoi_inv(3,3) !inertia matrix, inverse inertia matrix
C dimension rcm(3) !center of mass location, geometric coordinates
C
C common/subsys1/uul,yy1 !orbit dynamics, input/output vs. time
C common/endcond1/yy1i,yy1f !initial and final output conditions
C
C common/simulation/ndp,delt,tol,res,t_start, !all except some single-time-point utility routines
C 1t_report,t_print,t_stop,nmt,nmtr,nit,
C 2t_initial,t_final !recursive.f, orbit.f, sub1, attitude.f, sub2.f
C
C common/constants/rearth,gme,pi,cvdr
C common/mass/xmass,xmoi,xmoi_inv,rcm !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
C
C if(mode.eq.2.or.mode.eq.3) go to 200
C
C 100 continue ! (mode=1, define constants, system parameters)
C
C **** Numerically Defining Constants (mode 1 function) : ****
C ****
C ****

```

```

C     gravconst=(6.673D0)*(1.0e-11)      ! Constant of Gravitation,
C                                         ! m**3/kg-sec**2, CRC, page 6
C
C     rearth= 6367.65D0                  ! Earth's radius, (polar+equatorial)/2.0,
C                                         ! km, CRC, page 6
C
C     dearth=5522.0D0                   ! Average density of the earth,
C                                         ! kg/m**3, CRC, page 6
C     pi=datan2(0.D0,-1.D0)           ! Computes pi to machine precision
C     cvdr=pi/180.0D0                  ! Converts degrees to radians
C
C     volume=(4.0D0/3.0D0)*pi*(rearth**1000.0D0)**3 ! Volume of earth, m**3
C
C     xmearth=dearth*volume            ! Mass of the earth, kg
C
C     gme=gravconst*xmearth          ! Gravity Constant times mass of the earth.
C
C     *****
C     **** Setting Values of System Parameters (mode 1 function): ****
C     ****
C
C     xmass=10000.0D0                 ! spacecraft mass, kilograms
C
C     print*, ' Input stepsize reduction factor'
C     print*, ' for ORBIT DYNAMICS simulation'
C     print*, ' (integer): '
C     read*, idiv
C     return
C
C     ****
C     **** End of mode 1, beginning of mode 2 ****
C     ****
C
C     200 if(mode.eq.3) go to 300
C     continue ! (mode=2, set initial conditions)

```

```

C   **** Setting Initial Conditions (mode 2 function) : ****
C   ****
C
C      alt=300.0D0
C
C      ! altitude, kilometers
C
C      rmag= (rearth+alt)*1000.0D0
C      ! meters
C      r(1)=rmag*dcos (30.0d0*cvdr)
C      r(2)=0.0d0
C      r(3)=rmag*dsin (30.0d0*cvdr)
C
C      vmag=dsqrt (gme/rmag)
C      ! Circular orbit, meters/sec
C      rcube=rmag**3
C      wsq=gme/rcube
C      cperiod=2.0d0*pi/dsqrt (wsq)
C      ! Period for circular orbit
C
C      v(1)=0.0
C      v(2)=vmag
C      v(3)=0.0
C
C      do 5 i=1,3
C      yy1i(i)=r(i)
C      yy1i(i+3)=v(i)
C      5 continue
C
C      return
C
C      **** End of mode 2, beginning of mode 3 ****
C
C      300 continue ! (mode=3, run simulation over the convergence interval)
C
C   **** Setting up the initial state vector (6 states, mode 2 function) : ****
C   ****

```

```

C      do 6 i=1,3          ! retrieving initial conditions from block common /endcond1/
C      r(i)=yy1(i)
C      v(i)=yy1(i+3)

C      yy1(i,1)=r(i)        ! entering initial output vector (time point number 1) into block
C      yy1(i+3,1)=v(i)      ! common /subsys1/

C      6 continue

C      do 10 i=1,3          ! changing from kinematic to state variable notation
C      x(i)=r(i)            ! states 1-3, orbit radius, meters
C      x(i+3)=v(i)          ! states 4-6, orbit velocity, meters/sec
C      10 continue

C      tau=0.0D0 ! initializing time, measured from the start of the convergence interval
C      ndim=6 ! defining dimension of the state vector

C      continue !***** begin integration loop *****
C      do 20 i=2,npd
C      dt=delt/float(idiv)   ! smaller stepsize than delt
C      do 30 ii=1,idiv
C      call rksys1(tau,x,ndim,dt) ! propagates orbit states across convergence interval
C      30 continue

C      do 7 j=1,6
C      yy1(j,i)=x(j)
C      7 continue

C      20 continue

C      ! enters output (orbit state vector) into block common /susbsy1/.
C      ! data (time) points 2 through npd are dealt with here
C      7 continue

C      20 continue

C      !***** end integration loop *****
C      return

```

```

C   *
C   * **** End of mode 3 ****
C   *
C   * **** End of report interval ****
C   *
C   end

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx................................................................

```

Subroutine rfc_out

```

C   implicit double precision(a-h,o-z)
C
C   dimension uu(18,100),yy(18,100),yyp(18,100)
C   dimension zz(18,100)
C   dimension phi(3),amat(3,3) ! (10/16/00)
C
C   dimension rad(3),rad_n(3) !for processing orbit data to obtain altitude
C
C   dimension f_aero(3), t_aero(3) ! added 9/28/2000, for aero torque output
C   dimension xmoi(3,3),xmoi_inv(3,3),rcm(3) ! moments of inertia, about c.m., c.m. location vector
C
C   common/system/uu,yy,yyp !system-level input/output/"previous" output
C
C   common/simulation/ndp,delt,tol,res,t_start, !all except some single-time-point utility routines
C   1t_report,t_print,t_stop,nmt,nmtr,nit,
C   2t_initial,t_final
C
C   common/simulation2/del_report,del_print,ptol !percent done report interval, print interval
C

```

```

common/constants/rearth,gme,pi,cvdr          !recursive.f, orbit.f, sub1, attitude.f, sub2.f
c
common/mass/xmass,xmoi,xmoi_inv,rcm          !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
c                                               !added 9/28/2000, for aero torque output
c
do 3 i=1,18
do 3 j=1,ndp
zz(i,j)=yy(i,j)
3 continue
c
do 4 i=1,3
do 4 j=1,ndp
phi(i)=zz(i+6,j)                           ! unlimited angle (10/16/00)
call dcm321(phi,amat)
call dcm2ang321(amat,phi)
zz(i+6,j)=phi(i)/cvdr                      ! limited angle (10/16/00)
zz(i+9,j)=yy(i+9,j)/cvdr
c
4 continue
c
do 5 j=1,ndp
f_aero(1)=zz(13,j)                         ! added 9/28/2000 for aero. torque computation for output
f_aero(2)=zz(14,j)
f_aero(3)=zz(15,j)
call vcross(rcm,f_aero,t_aero)
zz(16,j)=t_aero(1)
zz(17,j)=t_aero(2)
zz(18,j)=t_aero(3)
5 continue
c
time=nmt*(ndp-1)*delt                      !nmt=number of major time (convergence) intervals complete
c                                               !ndp=number of data points per major interval, including both end points
c                                               !delt=time between data points (also integration stepsize)
time=time1                                     !initializing time to first point of convergence interval
c
do 1000 i=1,ndp-1
c
do 1 k=1,3

```



```

C THIS SUBROUTINE PERFORMS ONE STEP IN THE RUNGE-KUTTA
C INTEGRATION OF A SYSTEM OF FIRST ORDER DIFFERENTIAL
C EQUATIONS OF THE FORM Y' (X)=FD(X,Y) , WHERE Y AND FD
C ARE VECTOR FUNCTIONS, AND X IS THE INDEPENDENT
C VARIABLE . THE ALGORITHM IS DESCRIBED BY THE FOLLOWING
C VECTOR RELATIONS:
C
C F1=FD (X,Y) ,
C F2=FD (X+H/2 ,Y+F1* H/2) ,
C F3=FD (X+H/2 ,Y+F2* H/2) ,
C F4=FD (X+H ,Y+F3* H) ,
C
C Y=Y+H ,
C X=X+H ,
C
C DIMENSION Y(6) ,YSTART(6) ,YTEMP(6) ,FDERIV(6) ! Set for point mass orbit dynamics
C
C H2=H/2.0
C H3=H/3.0
C H6=H/6.0
C
C X0=X
C DO 200 I=1,N
C 200 YSTART(I)=Y(I)
C CALL SUB1(FDERIV,Y,X) ! "1" added to identify with subsystem #1
C DO 300 I=1,N
C YTEMP(I)=YSTART(I)+FDERIV(I)*H6
C 300 Y(I)=YSTART(I)+FDERIV(I)*H2
C
C *** END OF FIRST PASS ***
C
C X=X0+H2
C CALL SUB1(FDERIV,Y,X) ! "1" added to identify with subsystem #1
C DO 400 I=1,N
C YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
C 400 Y(I)=YSTART(I)+FDERIV(I)*H2
C
C *** END OF SECOND PASS ***
C
C CALL SUB1(FDERIV,Y,X) ! "1" added to identify with subsystem #1
C DO 500 I=1,N

```

```

      YTEMP ( I )=YTEMP ( I )+FDERRIV ( I )*H3
  500  Y ( I )=YSTART ( I )+FDERRIV ( I )*H
C
C *** END OF THIRD PASS ***
C
C           X=X0+H
C           CALL SUB1 ( FDERRIV, Y, X)
C           ! "1" added to identify with subsystem #1
C
C           DO 600 I=1,N
C           600 Y ( I )=YTEMP ( I )+FDERRIV ( I )*H6
C
C *** END OF FOURTH PASS ***
C
C           AT THIS POINT ONE INTEGRATION STEP HAS BEEN COMPLETED.
C           Y ( X ) HAS BEEN REPLACED BY Y ( X+H ), AND
C           X HAS BEEN REPLACED BY X+H.
C
C           RETURN
C           END
C
C
C
C
C
C
C           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
C           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
C
C
C           SUBROUTINE RKSYS2 ( X, Y, N, H)               ! "2" added to identify with subsystem #2
C
C           IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
C           THIS SUBROUTINE PERFORMS ONE STEP IN THE RUNGE-KUTTA
C           INTEGRATION OF A SYSTEM OF FIRST ORDER DIFFERENTIAL
C           EQUATIONS OF THE FORM Y' ( X )=FD ( X, Y ), WHERE Y AND FD
C           ARE VECTOR FUNCTIONS, AND X IS THE INDEPENDENT
C           VARIABLE. THE ALGORITHM IS DESCRIBED BY THE FOLLOWING
C           VECTOR RELATIONS:
C

```

```

C          F1=FD (X,Y) ,          YT=Y+F1* H/6
C          F2=FD (X+H/2 , Y+F1* H/2) ,   YT=YT+F2* H/3
C          F3=FD (X+H/2 , Y+F2* H/2) ,   YT=YT+F3* H/3
C          F4=FD (X+H , Y+F3* H) ,      YT=YT+F4* H/6
C          X=X+H,          NEXT STEP
C
C          DIMENSION Y (6) ,YSTART (6) ,YTEMP (6) ,FDERIV (6) ! Set for rigid body rotational dynamics
C          H2=H/2.0
C          H3=H/3.0
C          H6=H/6.0
C          X0=X
C          DO 200 I=1,N
C          YSTART (I)=Y (I)
C          CALL SUB2 (FDERIV ,Y,X) ! "2" added to identify with subsystem #2
C          DO 300 I=1,N
C          YTEMP (I)=YSTART (I)+FDERIV (I)* H6
C          300 Y (I)=YSTART (I)+FDERIV (I)* H2
C
C          C *** END OF FIRST PASS ***
C
C          X=X0+H2
C          CALL SUB2 (FDERIV ,Y,X) ! "2" added to identify with subsystem #2
C          DO 400 I=1,N
C          YTEMP (I)=YTEMP (I)+FDERIV (I)* H3
C          400 Y (I)=YSTART (I)+FDERIV (I)* H2
C
C          C *** END OF SECOND PASS ***
C
C          CALL SUB2 (FDERIV ,Y,X) ! "2" added to identify with subsystem #2
C          DO 500 I=1,N
C          YTEMP (I)=YTEMP (I)+FDERIV (I)* H3
C          500 Y (I)=YSTART (I)+FDERIV (I)* H
C
C          C *** END OF THIRD PASS ***
C
C          X=X0+H
C          CALL SUB2 (FDERIV ,Y,X) ! "2" added to identify with subsystem #2
C          DO 600 I=1,N

```

```

600 Y(I)=YTEMP(I)+FDERIV(I)*H6
C
C *** END OF FOURTH PASS ***
C
C AT THIS POINT ONE INTEGRATION STEP HAS BEEN COMPLETED.
C Y(X) HAS BEEN REPLACED BY Y(X+H), AND
C X HAS BEEN REPLACED BY X+H.
C
C RETURN
C
C
C subroutine subl(xdot,x,tau)
C
C *** This subroutine simulates three-degree-of-freedom (six-state) orbital dynamics with inverse
C square gravity and point mass assumptions. An aerodynamic drag force f_aero(i) can be applied
C to the satellite through common block "subsys1" variable uul(i,j), where the following
C definitions apply:
C
C x(i) = value of the ith component of the orbit state vector. This is an
C INPUT VARIABLE, and is not changed by this routine.
C
C xdot(i) = time derivative of the ith component of the orbit state vector.
C This is an OUTPUT VARIABLE, and is created by this routine.
C
C tau = time elapsed from the beginning of the convergence interval. This
C is an INPUT VARIABLE, and is not changed.
C
C uul(1,j) = value at the jth time point of the x-component of body force f(t)
C expressed in inertial coordinates. Units are newtons. This is an

```

```

c INPUT VARIABLE, and is not changed.
c
c uu1(2,j) = value at the jth time point of the y-component of body force f(t)
c expressed in inertial coordinates. Units are newtons. This is an
c INPUT VARIABLE, and is not changed.
c
c uu1(3,j) = value at the jth time point of the z-component of body force f(t)
c expressed in inertial coordinates. Units are newtons. This is an
c INPUT VARIABLE and is not changed.
c
c yy1(i,j) = value at the jth time point of the ith component of the orbit state
c vector. THIS VARIABLE IS NOT USED BY THIS ROUTINE.
c
c ndp = number of signal sample points per convergence interval, including
c both end points
c
c delt = time between signal sample points, same as integration stepsize.
c
c implicit double precision(a-h,o-z)
c
c dimension uu1(6,100),yy1(6,100)
c
c dimension f_aero(3) !aerodynamic force, single time point, inertial coordinates
c dimension f_grav(3) !gravitational force, single time point, inertial coordinates
c dimension rnorm(3) !normalized radius vector, inertial coordinates
c
c dimension func_x(100),func_y(100),func_z(100) ! vector input signal, 100 time points
c ! (aerodynamic force)
c
c dimension x(6),xdot(6),r(3),rdot(3),v(3),vdot(3)
c
c dimension xmoi(3,3),xmoi_inv(3,3) !inertia matrix, inverse inertia matrix
c
c dimension rcm(3) !center of mass location, geometric coordinates
c
c common/subsys1/uu1,yy1 !orbit dynamics, input/output vs. time
c common/endcond1/yy1i,yy1f !initial and final output conditions

```

```

C common/simulation/ndp,delt,tol,res,t_start, !all except some single-time-point utility routines
C lt_report,t_print,t_stop,nmt,nmtr,nit,
C 2t_initial,t_final
C
C common/constants/rearth,gme,pi,cvdr          !recursive.f, orbit.f, sub1, attitude.f, sub2.f
C
C common/mass/xmass,xmoi,xmoi_inv,rcm        !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
C
C *****
C
C Orbit States (Vehicle position and velocity, inertial coordinates):
C
C
C x(1)=r(1) : x-component of orbit radius vector (meters)
C x(2)=r(2) : y-component
C x(3)=r(3) : z-component
C
C x(4)=v(1) : x-component of spacecraft velocity vector, (meters/second)
C x(5)=v(2) : y-component
C x(6)=v(3) : z-component
C
C
C *****
C
C r(1)=x(1) ! Changing from mathematical (state variable) to kinematic notation.
C r(2)=x(2)
C r(3)=x(3)
C v(1)=x(4)
C v(2)=x(5)
C v(3)=x(6)
C
C *** Gravity forces ***
C
C call vecnorm(r,rnorm,rmag) ! normalized radius vector (inertial coordinates)
C gmag=gme*xmass/rmag*2
C
C f_grav(1)=-gmag*xnorm(1) ! result in inertial coordinates
C f_grav(2)=-gmag*xnorm(2)

```

```

f_grav(3)=-gmag*rnorm(3)

C
C *** Aerodynamic forces ***
C
t1=0.0D0 ! time (tau) is measured from the left end point of the convergence interval
C
do 1 i=1,ndp !extracts scalar components of the input vector signal form the
  func_x(i)=uu1(1,i) !system-level common block "subsys1". The variable uu1(i,j) contains
  func_y(i)=uu1(2,i) !aerodynamic forces as described above.
  func_z(i)=uu1(3,i)
1 continue

C
call interp(ndp,t1,delt,tau,func_x,val) !interpolates x-component of aerodynamic force
f_aero(1)=val
call interp(ndp,t1,delt,tau,func_y,val) !interpolates y-component of aerodynamic force
f_aero(2)=val
call interp(ndp,t1,delt,tau,func_z,val) !interpolates z-component of aerodynamic force
f_aero(3)=val
cccc
print*, 'sub1, after interp', (f_aero(i),i=1,3)

C
C *** orbital dynamics (Orbit State Derivatives) ***
C
vdot(1)=(1.0D0/xmass)*(f_grav(1)+f_aero(1)) !acceleration, inertial coordinates
vdot(2)=(1.0D0/xmass)*(f_grav(2)+f_aero(2))
vdot(3)=(1.0D0/xmass)*(f_grav(3)+f_aero(3))

C
rdot(1)=v(1)
rdot(2)=v(2)
rdot(3)=v(3)

C
C ****
C
xdot(1)=rdot(1) !changing back to mathematical notation from physical notation
xdot(2)=rdot(2)
xdot(3)=rdot(3)
xdot(4)=vdot(1)
xdot(5)=vdot(2)
xdot(6)=vdot(3)

```

```

c
c      return
c      end
c
c      **** Construction began 6/28/2000, by Doug Nixon, TD55. This is the attitude dynamics (subsystem 2)
c      routine driven by rksys2.f which supports attitude.f. Attitude dynamics is the 2nd subsystem
c      of three (orbit dynamics, attitude dynamics, and aerodynamics) that form a system simulation
c      implemented by recursive feedback.
c
c      implicit double precision (a-h,o-z)
c
c      dimension uu2(6,100),yy2(6,100)          ! I/O for subsystem 2 (attitude dynamics)
c
c      dimension x(6),xdot(6)                  ! state variables for attitude dynamics
c
c      dimension phi(3),phidot(3),omega(3),omegadot(3) ! attitude angles and body rates (state variables)
c
c      dimension rad(3),rad_norm(3),rad_norm_b(3)   ! orbit radius vector (meters)
c
c      dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)   ! moments of inertia, about c.m., c.m. location vector
c
c      dimension t_gg(3)                         ! in geometric body coordinates
c
c      dimension t_aero(3)                      ! gravity gradient torque (body coordinates)
c
c      dimension t_aero(3)                      ! aerodynamic torque (body coordinates)
c
c      dimension t_control(3)                   ! control torque (body coordinates)

```

```

c dimension t_total(3)                                !total external torque minus gyroscopic torque
c dimension cmat(3,3)                                !transforms body rates to Euler angle rates
c dimension amat(3,3),amat_t(3,3)                      !transforms vectors from inertial to body coordinates
c dimension h(3), wkh(3)                             !angular momentum about c.m., gyroscopic torque
c dimension xir(3), rxxir(3)                         ! [ I ]{ r }, { r } X[ I ]{ r }, I=xmoi(i,j), r=rad_norm_b(i)
c dimension func_x(100), func_y(100), func_z(100)    !scalar version of aerodynamic torque input signal
c common/subsys2/uu2,yy2                             !recursive.f, attitude.f, sub2.f
c common/simulation/ndp,delt,tol,res,t_start,       !all except some single-time-point utility routines
1t_report,t_print,t_stop,nmt,nmtr,nit,
2t_initial,t_final                                  !recursive.f, orbit.f, sub1, attitude.f, sub2.f
c common/constants/rearth,gme,pi,cvdr               !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
c common/mass/xmass,xmoi,xmoi_inv,rcm              !attitude.f, sub2.f
c common/gains/a0,a1,b0,b1,c0,c1
c *** Some variable in definitions:
c gme : universal gravitational constant times mass of the earth (lb-ft**2 slug)
c xmass: spacecraft mass (slugs)
c xmoi : spacecraft moment of inertia tensor (3x3 matrix, slug-ft**2)
c
c The system has three degrees of freedom, or six states. The System states are
c defined as follows:
c
c Vehicle Rotational States:
c
c x(1)=phi(1) : Euler angle rotation from inertial to body system, about x-axis,
```

```

C      3rd in sequence. (radians)
C      x(2)=phi(2) : About y-axis, 2nd in sequence.
C      x(3)=phi(3) : About z-axis, 1st in sequence.

C      x(4)=omega(1) : Body rate, x-axis (radians/second)
C      x(5)=omega(2) : Body rate, x-axis
C      x(6)=omega(3) : Body rate, x-axis
C
C      *****
C
C      phi(1)=x(1)          ! Changing from mathematical to physical notation.
C      phi(2)=x(2)
C      phi(3)=x(3)

C      omega(1)=x(4)
C      omega(2)=x(5)
C      omega(3)=x(6)

C      call dcm321(phi,amat) !computes transformation (DCM) from inertial
C                           !to body coordinates

C      call trans3(amat,amat_t) !transformation from body to inertial coordinates

C      do 1 i=1,ndp           !extracts scalar components of the input vector signal from the
C      func_x(i)=uu2(1,i)     !common block "subsys2". The variable uu2(1-3,j) contains the orbit
C      func_y(i)=uu2(2,i)     !radius vector inertial coordinates.
C      func_z(i)=uu2(3,i)
C      1 continue

C      t1=0.0D0 ! time (tau) is measured from the left end point of the convergence interval

C      call interp(ndp,t1,delt,tau,func_x,val) !interpolates x-component of the radius vector
C      rad(1)=val
C      call interp(ndp,t1,delt,tau,func_y,val) !... y-component ...
C      rad(2)=val
C      call interp(ndp,t1,delt,tau,func_z,val) !... z-component ...

```

```

rad(3)=val
C   call vecnorm(rad,rad_norm,rmag)      ! normalized radius vector in inertial coordinates
C   C *** Gravity Gradient Torques ***
C   call matvec(amat,rad_norm,rad_norm_b)    ! normalized radius vector in body coordinates
C   call matvec(xmoi,rad_norm_b,xir)        ! [ I ] { r }, unit vector, body coordinates
C   call vcross(rad_norm_b,xir,rxxir)       ! { r } X[ I ] { r }
C
C   if(rmag.lt.1.0e-6) ggmul=0.0
C   if(rmag.ge.1.0e-6) ggmul=3.0D0*gme/rmag**3

C   t_gg(1)=ggmul*rxxir(1)                  ! gravity gradient torque in body coordinates
C   t_gg(2)=ggmul*rxxir(2)
C   t_gg(3)=ggmul*rxxir(3)

C   C *** Aerodynamic torque calculation at time=t1+tau
C
C   do 2 i=1,ndp                          !extracts scalar components of the input vector signal from the
C   func_x(i)=uu2(4,i)                    !common block "subsys2". The variable uu2(4-6,j) contains the aerodynamic
C   func_y(i)=uu2(5,i)                    !torque in body (c.m. origin) coordinates.
C   func_z(i)=uu2(6,i)
C   2 continue

C   t1=0.0D0 ! time (tau) is measured from the left end point of the convergence interval
C
C   call interp(ndp,t1,delt,tau,func_x,val) !interpolates x-component of aerodynamic torque
t_aero(1)=val
C   call interp(ndp,t1,delt,tau,func_y,val) !... y-component...
t_aero(2)=val
C   call interp(ndp,t1,delt,tau,func_z,val) !... z-component...
t_aero(3)=val

C   C *** Control System Torques ***
C
C   t_control(1)=- (a0*phi(1)+a1*omega(1)) ! control torque in body coordinates (c.m. origin)

```

```

t_control(2)=- (b0*phi(2)+b1*omega(2))
t_control(3)=- (c0*phi(3)+c1*omega(3))

C *** Attitude dynamics
C
C   call matvec(xmoi,omega,h)
C   call vcross(omega,h,wxh)
t_total(1)=t_GG(1)+t_aero(1)+t_control(1)-wxh(1) !disturbance plus gyroscopic torque in body frame
t_total(2)=t_GG(2)+t_aero(2)+t_control(2)-wxh(2)
t_total(3)=t_GG(3)+t_aero(3)+t_control(3)-wxh(3) !angular acceleration of body frame in body
C
C
C   call matvec(xmoi_inv,t_total,omegadot) !coordinates

C
C   Converting body rates to Euler angle derivatives
C
C   (Rotation sequence precludes 90deg angle about y-axis)

sin_1=dsin(phi(1))
cos_1=dcos(phi(1))
sin_2=dsin(phi(2))
cos_2=dcos(phi(2))
tan_2=sin_2/cos_2
cmat(1,1)=1.0D0
cmat(1,2)=sin_1*tan_2
cmat(1,3)=cos_1*tan_2
cmat(2,1)=0.0D0
cmat(2,2)=cos_1
cmat(2,3)=-sin_1
cmat(3,1)=0.0
cmat(3,2)=sin_1/cos_2
cmat(3,3)=cos_1/cos_2
call matvec(cmat,omega,phidot) ! phidot in non-orthogonal Euler-frame coordinates

C *** Changing back to mathematical notation from physical notation.
C
C   xdot(1)=phidot(1)

```

```

xdot (2)=phidot (2)
xdot (3)=phidot (3)
xdot (4)=omegadot (1)
xdot (5)=omegadot (2)
xdot (6)=omegadot (3)

return
end

```

```

C      SUBROUTINE TRAN3 (A,AT)
C      **** THIS ROUTINE TAKES THE TRANSPOSE OF A 3X3 MATRIX.
C
C      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
C      DIMENSION A (3, 3) , AT (3, 3)
C
C      DO 10 I=1, 3
C      DO 10 J=1, 3
C      10 AT (I, J) =A (J, I)
C
C      RETURN
C      END

```



```
end
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
C  
C      THIS ROUTINE NORMALIZES A VECTOR (3X1).  
C  
C      IMPLICIT DOUBLE PRECISION (A-H,O-Z)  
C  
C      DIMENSION X (3) ,XNORM (3)  
C  
C      XMAG=DSQRT(X(1)**2+X(2)**2+X(3)**2)  
C      if (xmag .le. 0.0D0) go to 11  
C      DO 10 I=1,3  
C      XNORM (I)=X (I) /XMAG  
C      10 CONTINUE  
C  
C      RETURN  
C  
C      11 xmag=0.0D0  
C  
C      do 12 i=1,3  
C            xnorm(i)=0.0D0  
C      12 continue  
C  
C      return  
END
```

B.2 FORTRAN Listings, Including Main Program and All Subroutines Not Previously Listed in A.1, Recursive Feedback With Distributed Coupling (RF–D)

```

C   This is FORTRAN MAIN program "rfd.f", and is a variation of FORTRAN main program "rfc.f".
C   Development began 6/12/2000 by Doug Nixon, MSFC/TD55, in support of AEI. The main and subprogram
C   listings are contained in "aei/rfd"
C
C   This example of simulation by recursive feedback uses distributed coupling, and convergence is
C   checked on the minimum number of signals (aerodynamic forces and torques).
C
C   The purpose is to generate the dynamic solution of three coupled subsystems (orbit dynamics,
C   attitude dynamics, and aerodynamics) by recursive feedback, which involves distributed and
C   possibly multiple methods of numerical integration. The intent is to compare results generated
C   by a standard approach using a centralized single numerical integration method. The standard
C   approach has been implemented as FORTRAN MAIN program "rku.f", and the main and subprogram listings
C   are contained in "aei/rku".
C
C   **** Declarations ****
C
C   implicit double precision(a-h,o-z)
C
C   dimension uu1(6,100),yy1(6,100)          !input and output signals for three subsystems,
C   dimension uu2(6,100),yy2(6,100)          !six signals per subsystem, 100 time points
C   dimension uu3(6,100),yy3(6,100)
C
C   dimension fb(6,100),fbp(6,100)           !for convergence check
C
C   dimension yy1i(6),yy2i(6),yy3i(6)        !initial conditions, 3 subsystems, 6 outputs
C   dimension yy1f(6),yy2f(6),yy3f(6)        !final conditions, 3 subsystems, 6 outputs
C
C   common/system/uu,yy,yyyp                 !system input/output/"previous" output, all vs. time
C
C   common/subsys1/uu1,yy1                  !orbit dynamics, i/o vs. time
C   common/endcond1/yy1i,yy1f               !initial and final output conditions
C
C   common/subsys2/uu2,yy2                  !attitude dynamics, i/o vs. time
C   common/endcond2/yy2i,yy2f               !initial and final output conditions
C
C   common/subsys3/uu3,yy3                  !aerodynamics, i/o vs. time
C   common/endcond3/yy3i,yy3f               !initial and final output conditions

```



```

C *****
C
C   yy1(i) ----- Value of the output signal at the beginning of the current convergence interval
C   for the FIRST (orbit dynamics) subsystem. Represents the ith component of the
C   signal at the first time point, and is equivalent to yy1(i,1)
C
C   yy2(i) ----- Value of the output signal at the beginning of the current convergence interval
C   for the SECOND (attitude dynamics) subsystem. Represents the ith component of the
C   signal at the first time point, and is equivalent to yy2(i,1)
C
C   yy3(i) ----- Value of the output signal at the beginning of the current convergence interval
C   for the THIRD (aerodynamics) subsystem. Represents the ith component of the signal
C   at the first time point, and is equivalent to yy3(i,1)
C *****
C
C   yy1f(i) ----- Value of the output signal at the end of the current convergence interval for
C   the FIRST (orbit dynamics) subsystem. Represents the ith component of the signal
C   at the last time point, and is equivalent to yy1(i,np)
C
C   yy2f(i) ----- Value of the output signal at the end of the current convergence interval for
C   the SECOND (attitude dynamics) subsystem. Represents the ith component of the
C   signal at the last time point, and is equivalent to yy2(i,np)
C
C   yy3f(i) ----- Value of the output signal at the end of the current convergence interval for
C   the THIRD (aerodynamics) subsystem. Represents the ith component of the vector
C   signal yy3(i,j) at the last time point, and is equivalent to yy3(i,np)
C
C **** General Comments ****
C
C The above nomenclature has been designed to allow implementation of recursive feedback via
C FORTRAN in such a way that independence of the subsystem simulations can be supported. Definition
C of submatrices (partitioning) facilitates independence of the subsystem simulations while providing
C appropriate structure for implementation of the system-level algorithm. The fact that subsystem
C data has been placed in COMMON with the MAIN (system-level) program does not necessarily mean
C that all of that data is required or desired at the system level -- it may mean only that the
C SUBROUTINE containing the subsystem simulation needs to have access to data that it generated
C previously, and placing that data in COMMON is a way to provide that in FORTRAN.

```

```

C Because of the distributed coupling configuration, information flows from one sub-simulation to the
C next in a sequential manner. The order of execution is important, and parallel execution is not
C possible.

C This implementation assumes that all subsystem states are needed at the system level, and transfer of
C final conditions to initial conditions after convergence is handled at that level. For this specific
C system, the assumption is valid; however, in general, subsystem states can exist and remain at the
C subsystem level without the necessity of knowing them at the system level. Because of this,
C it would be more appropriate to define and "bookkeep" initial conditions at the subsystem level for a
C more general implementation. In the more general implementation, transfer of final conditions from
C one convergence interval to initial conditions of the next would be handled totally at the subsystem
C level upon receipt of "command" from the system level.

C *** Establishing the output file *****
C
C open(unit=1,file='rfd.outl',status='unknown')

C *** Defining system-level simulation parameters *****
C
C
C print*
print*, 'This is main program rfd.f. Solution'
print*, 'method is recursive feedback with'
print*, 'distributed coupling.'
print*
print*, ' Input the number of data (time) points'
print*, ' per convergence interval, including'
print*, ' both end (boundary) points:'
print*
read*, ndp
print*
print*, ' Input the convergence interval duration'
print*, '(sec.) :'
print*
read*, dtmaj
print*
C
fac=1.0/float(ndp-1) !ratio of integration time step to convergence interval duration

```

```

delt=fac*dtmaj          !integration stepsize
c
c      print*, '   The above data limits all integration'
c      print*, '   stepsizes to ',delt,' sec.'
c      print*, '   or less'
c      print*
c
c *** Populating the subsystem databases ****
c
c      call orbit(1)      !defines parameter values (data) for subsystem 1 (orbit dynamics, mode 1)
c
c      call attitude(1)   !defines parameter values (data) for subsystem 2 (orbit dynamics, mode 1)
c
c      call aerodyn(1)    !defines parameter values (data) for subsystem 3 (orbit dynamics, mode 1)
c
c      print*, '   Input the simulation stop time (sec.):'
c      print*
c      read*, t_stop
c      print*
c      print*, '   Input the data output(print) interval'
c      print*, '(sec.):'
c      print*
c      read*, del_print
c      print*, '   Input percent done status interval'
c      print*, '(%):'
c      print*
c      read*, per_report
c      print*
c      del_report=(per_report/100.0)*t_stop
c
c      nmtr=int(t_stop/dtmaj)+1      !number of major time (convergence) intervals required for completion of
c                                     !the simulation. (Time may exceed t_stop by as much as one major interval)
c
c      tol=(1.0D0)*1.0e-14          !relative
c      res=(1.0D0)*1.0e-14          !absolute
c      itmax=100                     !maximum allowable number of iterations
c      ptol=1.0e-3                   !tolerance for print time

```

```

print*, ' Run in progress.'
print*
call clock(n1)          !machine start time

C *** Initializing time and convergence interval count: ****
C
C   time=0.0           !time is measured from start of the simulation
C   nmt=0             !number of major time intervals (convergence intervals) simulated
C   t_print=0.0         !initializes print time
C   t_report=0.0        !initializes percent done report time

C *** Setting up initial conditions for each sub-simulation at system simulation outset
C
C   call orbit(2)      !mode 2, original initial conditions setup, orbit dynamics
C
C   call attitude(2)   !mode 2, original initial conditions setup, attitude dynamics
C
C   !no initial conditions setup for aerodynamics (algebraic system)
C
C *** Continue !***** COMPUTING INITIAL RECURSION STAGE DATA***** ****
C   100 continue
C *** Initialize subsystem signals to zero (0)
C
C   nit=0 !number of iterations initialized to zero for first (new) convergence interval
C
C *** Initializing the system-level feedback signal to zero: ****
C
C   do 1 i=1,6           !six scalar input signals per subsystem
C   do 1 j=1,npd          !npd number of data (time) points per scalar signal
C   uu1(i,j)=0.0D0        !initializing all subsystem input signals to zero (0)
C   uu2(i,j)=0.0D0
C   uu3(i,j)=0.0D0
C   1 continue
C
C *** Running individual subsystem simulations over the convergence interval
C
C   call orbit(3)        ! mode 3, run orbit dynamics simulation over convergence interval
C

```

```

call coupler1
C   call attitude(3) ! mode 3, run attitude dynamics simulation over convergence interval
C   call coupler2
C   call aerodyn(3) ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
C   call coupler3 !computes feedback signal from aerodynamics to orbit and attitude dynamics
C
do 2 i=1,3 !sets initial recursion stage data to current values (aerodynamics, feedback)
do 2 j=1,ndp
fbp(i,j)=uul(i,j) !aerodynamic force, inertial coordinates
fbp(i+3,j)=uu2(i+3,j) !aerodynamic torque, body coordinates
2 continue
C *****
C 200 continue !***** RECURSION LOOP ENTRY POINT***** RECURSION LOOP ENTRY POINT*****
C *****
nit=nit+1 !updating the iterations count (number of recursion stages)

C *** Running individual subsystem simulations over the convergence interval
C   call orbit(3) ! mode 3, run orbit dynamics simulation over convergence interval
C   call coupler1
C   call attitude(3) ! mode 3, run attitude dynamics simulation over convergence interval
C   call coupler2
C   call aerodyn(3) ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
C   call coupler3
C   C*** Check for convergence *****
C   do 3 i=1,3

```

```

do 3 j=1,3
  fb(i,j)=uu1(i,j)
  fb(i+3,j)=uu2(i+3,j)
3 continue

c   call fastverge(6,fb,fbp,iconflag)           ! aerodynamic force, inertial coordinates
c                                         ! iconflag=0 if converged, 1 if
c                                         ! notconverged

500  format(/,2i5,/)
  go to (6,5) indirect                      ! 6 for converged, 5 for not converged

c *****
c ***** 5 continue ! ***** Not CONVERGED, ADVANCE RECURSION STAGE *****
c ***** includes both end points
c
c   do 7 i=1,6                                !number of scalar signals at recursive transfer point
c   do 7 j=1,npd                               !number of time points in each convergence interval,
c
c   fbp(i,j)=fb(i,j)                          !sets previous recursion stage data to current values
7 continue

c   if(nit.ge.itmax) go to 300                !protection against persistent non convergence
c
c   go to 200                                 !returns to "top" of the recursion loop to run subsystem simulations again

c *****
c ***** 6 continue ! ** CONVERGED, ADVANCE TO NEXT CONVERGENCE INTERVAL (or stop if simulation is complete) *****
c ***** includes both end points
c
c   call rfd_out                             !appends system-level output signals to output file (rfd.out1)
c   nmt=nmt+1                               !increments number of major time (convergence) intervals completed
c   time=nmt*dtmaj                           !simulated problem time, for reference only

c   if(nmt.eq.nmtr) go to 300               !end of run, time is greater than or equal to the stop time (t_stop)

c *****
c ***** includes both end points
c

```

```

C *** Proceed ("hand-off") to next major time (convergence) interval
C
C In a more general setting, this would be done at the subsystem level (mode 4) so that the system
C level is not required to have knowledge of all subsystem states.
C
C do 8 i=1,6
C           !extracts subsystem final conditions for the current convergence interval
C           !from the final time point of the system response associated with the current
C           !convergence interval. SEE PARAGRAPH NUMBER 3 of "General Comments" ABOVE.
C
C           yy1f(i)=yy1(i,ndp)
C           yy2f(i)=yy2(i,ndp)
C           yy3f(i)=yy3(i,ndp)
C           8 continue
C
C do 9 i=1,6
C           !sets initial conditions (for all states) for the next convergence interval
C           !equal to conditions reached at the end (final conditions) of current
C           !convergence interval.
C
C           yy1i(i)=yy1f(i)
C           yy2i(i)=yy2f(i)
C           yy3i(i)=yy3f(i)
C           9 continue
C
C go to 100
C           !run subsystem simulations for the first time across the new (next)
C           !convergence interval.
C
C 300 continue
C
C print*
C           if(nit.ge.itmax) print*, 'Max. allowable number of '
C           if(nit.ge.itmax) print*, 'iterations reached.'
C
C call clock(n2) !machine stop time
C           irun=n2-n1
C           !run-time
C
C           if(nmt.eq.nmtr) print*, ' Normal exit, run-time = ',irun,' sec.'
C           print*
C
C           close(unit=1,status='keep')
C
C           stop
C
C **** END OF MAIN PROGRAM ****
C
C ****
C           end !*****
C
C ****

```



```

c           !extracting orbit radius from subsystem 1 output
rad(1)=yy1(1,j)
rad(2)=yy1(2,j)
rad(3)=yy1(3,j)

c           !extracting attitude (Euler) angles from subsystem 2 output signal
phi(1)=yy2(1,j)
phi(2)=yy2(2,j)
phi(3)=yy2(3,j)

c           !computes d.c.m that transforms vectors from inertial to body coordinates
call dcm321(phi,amat)           !computes d.c.m that transforms vectors from body to inertial coordinates
call tran3(amat,amat_t)          !computes d.c.m that transforms vectors from body to inertial coordinates

c           *** from orbit dynamics to aerodynamics
c           call matvec(amat,rad,rad_b) !transforms orbit radius vector to body coordinates

c           !feedback signal is orbit radius vector in body coordinates
uu3(1,j)=rad_b(1)
uu3(2,j)=rad_b(2)
uu3(3,j)=rad_b(3)

c           !variable vel(i) is used for listing readability only
vel(1)=yy1(4,j)
vel(2)=yy1(5,j)
vel(3)=yy1(6,j)

c           !transforms orbit velocity vector to body coordinates
call matvec(amat,vel,vel_b)           !feedback signal is orbit velocity vector in body coordinates

uu3(4,j)=vel_b(1)
uu3(5,j)=vel_b(2)
uu3(6,j)=vel_b(3)

c           10 continue
c           return
end

```



```

do 10 j=1,ndp

C   C *** from aerodynamics to orbit dynamics:
C
C     f_aero_b(1)=yy3(1,j)          !extracting aerodynamic (drag) force from system output signal
C     f_aero_b(2)=yy3(2,j)          !computed to act at origin of geometric body frame,
C     f_aero_b(3)=yy3(3,j)          !but can be assumed to act at c.m. for this purpose
C
C     phi(1)=yy2(1,j)              !extracting attitude (Euler) angles from the system output signal
C     phi(2)=yy2(2,j)
C     phi(3)=yy2(3,j)

C     call dcm321(phi,amat)        !computes d.c.m that transforms vectors from inertial to body coordinates
C     call trans(amat,amat_t)       !computes d.c.m that transforms vectors from body to inertial coordinates
C
C     call matvec(amat_t,f_aero_b,f_aero) !transforms force vector from body to inertial coordinates
C
C     uul(1,j)=f_aero(1)          !feedback signal is aerodynamic force (drag) in inertial coordinates,
C     uul(2,j)=f_aero(2)          !and is assumed to act at the satellite center of mass (c.m.)
C     uul(3,j)=f_aero(3)

C     uul(4,j)=0.0D0             !slot holders, nothing yet identified
C     uul(5,j)=0.0D0
C     uul(6,j)=0.0D0

C   C *** from aerodynamics to attitude dynamics:
C
C     f_aero_g(1)=yy3(1,j)          !aerodynamic (drag) force; acts at origin of geometric body frame.
C     f_aero_g(2)=yy3(2,j)          !f_aero_g(i) and f_aero_b(i) are numerically identical, but their
C     f_aero_g(3)=yy3(3,j)          !physical interpretations (points of action) may be different
C
C     t_aero_g(1)=yy3(4,j)          !aerodynamic torque with force at origin of geometric body frame
C     t_aero_g(2)=yy3(5,j)
C     t_aero_g(3)=yy3(6,j)

C     call vcross(rcm,f_aero_g,rxf) ! torque of drag force about c.m.
C
C     t_aero_b(1)=t_aero_g(1)+rxf(1) ! transferring drag force to c.m. and adding an

```



```

dimension uu1 (6,100),yy1 (6,100)
dimension uu2 (6,100),yy2 (6,100)
dimension uu3 (6,100),yy3 (6,100)
c
dimension zz(18,100) !single output array at system level
c
dimension rad(3),rad_n(3) !for processing orbit data to obtain altitude
c
dimension rcm(3),f_aero(3),t_aero(3) !added 9/21/2000, for aero torque output
dimension xmoi(3,3),xmoi_inv(3,3),rcm(3) !moments of inertia, about c.m., c.m. location vector
c
dimension phi(3) !Euler angles, added 10/2/00. needed here to limit values for output
c
common/subsys1/uu1,yy1
common/subsys2/uu2,yy2
common/subsys3/uu3,yy3
c
common/simulation/ndp,delt,tol,res,t_start, !all except some single-time-point utility routines
1t_report,t_stop,nmt,nmtr,nit,
2t_initial,t_final !percent done report interval, print interval
c
common/simulation2/del_report,del_print,ptol !percent done report interval, print interval
c
common/constants/rearth,gme,pi,cvdr !recursive.f, orbit.f, sub1, attitude.f, sub2.f
c
common/mass/xmass,xmoi,xmoi_inv,rcm !orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
!added 9/21/2000, for aero torque output
c
do 3 i=1,6
do 3 j=1,ndp
c
zz(i,j)=yy1(i,j)
zz(i+6,j)=yy2(i,j)
zz(i+12,j)=yy3(i,j)
c
3 continue
c
do 4 i=1,3
do 4 j=1,ndp

```

```

phi(i)=zz(i+6,j)                                ! unlimited angle (10/2/00)
call dcm321(phi,amat)
call dcm2ang321(amat,phi)
zz(i+6,j)=phi(i)/cvdr                          ! limited angle (10/2/00)
zz(i+9,j)=yy2(i+3,j)/cvdr

c      4 continue

c      do 5 j=1,ndp
f_aero(1)=zz(13,j)                            !added 9/21/2000 for aero. torque computaion for output
f_aero(2)=zz(14,j)
f_aero(3)=zz(15,j)
call vcross(rcm,f_aero,t_aero)
zz(16,j)=t_aero(1)
zz(17,j)=t_aero(2)
zz(18,j)=t_aero(3)
5 continue

c      time1=nmt*(ndp-1)*delt                  !nmt=number of major time (convergence) intervals complete
c      !ndp=number of data points per major interval, including both end points
c      !delt=time between data points (also integration stepsize)
c      !initializing time to first point of convergence interval

c      do 1000 i=1,ndp-1                         ! begin first write loop
c      !post processing orbit data for altitude
c      do 1 k=1,3
rad(k)=yy1(k,i)
1     continue
call vecnorm(rad,rad_n,rmag)
alt=rmag/1000.0D0-rearth !altitude, kilometers

c      if(time.ge.t_print) write(1,100) time,(zz(j,i),j=1,18),alt,nit
if(time.ge.t_print) t_print=t_print+del_print

c      if(time.ge.t_report) ipd=int((time/t_stop)*100.0)
if(time.ge.t_report) print*, 'Percent_complete = ', ipd
1     if(time.ge.t_report) t_report=t_report+del_report

```

```
100  format (20E20.10,i5)
      time=time+delt
1000 continue
C
      return
      end
```

B.3 FORTRAN Listings, Including Main Program and All Subroutines Not Previously Listed in A.1, Unified Approach, Runge-Kutta Method (RK-U)


```

print*, ' Input the output (print) interval, '
print*, ' (sec) : '
read*, delprint
print*, ' Input the stop time, (sec) '
read*, tstop
print*, ' Run in progress.'
call clock(n1) ! machine start time

C *** Numerically Defining Constants: ***
C
gravconst=(6.673D0)*(1.0e-11) ! Constant of Gravitation,
                                ! m**3/kg-sec**2, CRC, page 6
C
rearth= 6367.65D0 ! Earth's radius, (polar+equatorial)/2.0,
                     ! km, CRC, page 6
C
dearth=5522.0D0 ! Average density of the earth,
                  ! kg/m**3, CRC, page 6
pi=datan2(0.D0,-1.D0) ! Computes pi to machine precision
cvdr=pi/180.0D0 ! Converts degrees to radians
C
volume=(4.0D0/3.0D0)*pi*(rearth*1000.0D0)**3 ! Volume of earth, m**3
C
xmearth=dearth*volume ! Mass of the earth, kg
C
gme=gravconst*xmearth ! Gravity Constant times mass of the earth.
C
C *** Numerically Defining System Data: ***
C
alt=300.0D0 ! altitude, kilometers
C
xmass=10000.0D0 ! spacecraft mass, kilograms
C
xmoi(1,1)=11250.0D0 ! spacecraf moments of inertia,
xmoi(1,2)=0.0D0 ! kilogram-meters**2
xmoi(1,3)=0.0D0
xmoi(2,1)=0.0D0
xmoi(2,2)=193125.0D0 ! 3m diameter x 15m length uniform-

```

```

xmoi(2,3)=0.0D0          ! density right-circular cylinder
xmoi(3,1)=0.0D0
xmoi(3,2)=0.0D0
xmoi(3,3)=193125.0D0* .9 !IZZ=.9*IYY
call inv3(xmoi,xmoi_inv)

C
rcm(1)=1.0D0             ! center of mass location in geometric body
rcm(2)=0.1                ! coordinates, meters !offset in y and z 7/13/00
rcm(3)=0.1

C
control=1.0D0            !control on
control=0.0D0            !control off
a0=1000.0D0*control       ! control gain, x-axis position, n-m/rad
a1=100.0D0*control        ! control gain, x-axis rate, n-m-sec/rad
b0=1000.0D0*control       ! y-axis
b1=100.0D0*control        ! z-axis
c0=1000.0D0*control
c1=100.0D0*control

C
C *** Setting up fixed aerodynamic data, in common/aeroconst/ with rksys/sub/aerocomp
C
x1=15.0D0                ! length, meters (box shape, x-axis dimension)
w=4.0D0                   ! width, y-axis dimension
h=3.0D0                   ! height, z-axis dimension
ax=w*h                     ! end area, normal to x-axis (+ "front")
ay=x1*h                    ! side area, normal to y-axis (+ "right side")
az=x1*w                   ! top or bottom area, normal to (+ z-axis "bottom")

C
C Work with six facets ----
C
do 9 i=1,3                ! setting all vector components to zero
C
afront(i)=0.0D0            ! area vectors, normal to facet surfaces, through
arear(i)=0.0D0              ! centers, positive (point) outward from center of
aright(i)=0.0D0             ! box, back sides not considered (no contribution)
aleft(i)=0.0D0
atop(i)=0.0D0
abottom(i)=0.0D0

```

```

C cpfront(i)=0.0D0 ! centers of pressure location vectors, from geometric
C cprear(i)=0.0D0 ! center of box to center of each facet
C cpright(i)=0.0D0
C cpleft(i)=0.0D0
C cptop(i)=0.0D0
C cpbottom(i)=0.0D0

C 9 continue

C afront(1)=ax ! setting non-zero components of facet area vectors
C arear(1)=-ax
C aright(2)=ay
C aleft(2)=-ay
C abottom(3)=az
C atop(3)=-az

C cpfront(1)=x1/2.0D0 !setting non-zero components of facet
C cprear(1)=-x1/2.0D0 !centers-of-pressure location vectors
C cpright(2)=w/2.0D0
C cpleft(2)=-w/2.0D0
C cptop(3)=-h/2.0D0
C cpbottom(3)=h/2.0D0

C cd=2.25 ! drag coefficient, same for all facets,
C           ! not a function of direction

C rho=1.95E-11 ! kg/m^*3, Wertz, inside back cover, 300km altitude , mean value

C *** Setting Initial Conditions, assuming a circular orbit at a given altitude: ***
C

C rmag=(rearth+alt)*1000.0D0 ! Radius magnitude, meters
C r(1)=rmag*dcos(30.0d0*cvdr) ! Assumes a 30 deg "beta angle", (between ecliptic
C r(2)=0.0d0 ! and orbit planes)
C r(3)=rmag*dsin(30.0d0*cvdr)

```

```

C      vmag=dsqrt (gme/rmag)          ! Velocity magnitude for a circular orbit, meters/sec
C
C      rcube=rmag**3
C      wsq=gme/rcube
C      cperiod=2.0d0*pi/dsqrt (wsq)    ! Period for circular orbit
C
C      v(1)=0.0                         ! Velocity vector, circular orbit, meters/sec
C      v(2)=vmag
C      v(3)=0.0
C
C      phideg(1)=0.0                   ! Zero initial attitude error, degrees.
C      phideg(2)=0.0
C      phideg(3)=0.0
C
C      omegadeg(1)=0.0                 ! Zero initial body rates, degrees/sec
C      omegadeg(2)=0.0
C      omegadeg(3)=0.0
C
C      *** Changing Angular variables from degrees to radians: ***
C
C      phi(1)=phideg(1)*cvdr          ! deg to rad (attitude error)
C      phi(2)=phideg(2)*cvdr
C      phi(3)=phideg(3)*cvdr
C      omega(1)=omegadeg(1)*cvdr     ! deg/sec to rad/sec (body rates)
C      omega(2)=omegadeg(2)*cvdr
C      omega(3)=omegadeg(3)*cvdr
C
C      *** Setting up the initial state vector (12 states): ***
C
C      do 10 i=1,3
C          x(i)=r(i)                   ! states 1-3,          orbit radius, meters
C          x(i+3)=v(i)                ! states 4-6,          orbit velocity, meters/sec
C          x(i+6)=phi(i)              ! states 7-9,          body angles (attitude errors), radians
C          x(i+9)=omega(i)            ! states 10-12,         body rates, radians/sec
C
C      10 continue
C
C      *** Initializing time, opening data files, etc.

```

```

C          ! number of system states
C          ! initial time (problem start time), sec
C          call sub(xdot,x,time) ! computes dependent variables for initial output
C          tprint=time           ! initializing time to print output data set
C          ptol=.001*delt        ! tolerance on time for print

C          theta1=datan2(r(2),r(1)) ! initial value of true anomaly, radians
C          theta1=theta1/cvdr      ! convert to degrees
C          add=0.0D0               ! initial value for multiple revolution correction

C          if(theta1.lt.0.0D0) theta1=theta1+360.0D0 ! full circle before reset

C          open(unit=1,file='rku.out1',status='unknown') !opens data output file
C          20 continue !***** begin integration loop *****
C          if(abs(time-tprint).lt.ptol) call rku_out(time,x,delprint,tprint) !writes to file, updates tprint
C          call rksys(time,x,num,delt) ! propagates states, increments time
C          if(time.lt.tstop) go to 20 ! return to top of integration loop, or stop
C          continue !***** end integration loop *****
C          call rku_out(time,x,delprint,tprint) !writes final data set to file
C          close(unit=1,status='keep') !closes data output file
C          call clock(n2)           !machine stop time
C          iruntime=n2-n1
C          print*, 'Normal exit, run-time = ',iruntime,' sec.'
C          stop
C          end
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx................................................................

```

```

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx................................................................
C subroutine aerocomp(vel, amat, force, torque)
C
C Developed starting 5/26/2000 by Doug Nixon, TD55, MSFC to compute
C aerodynamic forces and torques for the "truth model" for recursive
C feedback simulation method comparison for AEI study.
C
C rho      = Atmospheric density, kg/m**3
C vel(i)   = Velocity vector in inertial coordinates, meters/sec
C amat(i,j) = Direction cosine matrix, transforms vectors from inertial to
C              body coordinates
C force(i)  = Aerodynamic force, body coordinates, newtons
C torque(i) = Aerodynamic torque, body coordinates, newton-meters
C
C implicit double precision (a-h,o-z)
C
C dimension vel(3),amat(3,3),force(3),torque(3)
C dimension vnorm(3),vnorm_b(3)
C
C dimension cpfront(3),cpright(3),cpbottom(3)          !constant system parameters computed or defined
C dimension cprear(3),cpleft(3),cptop(3)               !in the main program, standard f
C dimension afront(3),aright(3),abottom(3)
C dimension arear(3),aleft(3),atop(3)
C
C common/aeroconst/ cpfront,cpright,cpbottom,cprear,cpleft,    !in common with standard
C lctop,afront,aright,abottom,arear,aleft,atop,cd,rho
C
C dimension aproj(6),drag(6),facefor(6,3),facotor(6,3) !internally computed variables
C dimension tforce(3),ttorq(3)                         !output variables
C
C *** Aerodynamic forces ***
C
C call vecnorm(vel,vnorm,vmag) ! normalize velocity vector in inertial

```

```

c      ! coordinates
c      call matvec(amat,vnorm,vnorm_b) ! normalized velocity vector in body
c      ! coordinates
c
c      *** computing magnitude drag for each of six facets
c
c      call vdotpro(afront,vnorm_b,aproj(1)) ! computing dot product of each area
c      call vdotpro(aright,vnorm_b,aproj(2)) ! vector with normalized velocity
c      call vdotpro(abottom,vnorm_b,aproj(3)) ! (projected areas)
c      call vdotpro(arear,vnorm_b,aproj(4)) ! these statements define the facet
c      call vdotpro(aleft,vnorm_b,aproj(5)) ! numbering system
c      call vdotpro(atop,vnorm_b,aproj(6)) ! numbering system
c
c      do 20 i=1,6
c      if(aproj(i).le.0.0D0) aproj(i)=0.0D0 ! negative projection implies ith facet
c      20 continue ! not exposed to velocity vector
c
c      fac=(0.5D0)*rho*(vmag**2)*cd ! scaling by dynamic pressure times
c      do 21 i=1,6 ! drag coefficient
c      drag(i)=fac*aproj(i)
c      21 continue
c
c      do 22 i=1,6
c      do 22 j=1,3
c      facefor(i,j)=-drag(i)*vnorm_b(j) ! this is jth component of force on the ith
c      22 continue ! facet, expressed in body coordinates
c
c      do 23 i=1,3
c      force(i)=0.0D0
c      do 23 j=1,6
c      force(i)=force(i)+facefor(j,i) ! total aerodynamic force, body coordinates, newtons
c      23 continue
c
c      *** Aerodynamic Torques ****
c
c      do 30 i=1,3
c      tforce(i)=facefor(1,i) ! front facet contribution

```

```

30 continue
call vcross(cppfront,tforce,ttorq)
do 31 i=1,3
facotor(1,i)=ttorq(i)
torque(i)=torque(i)+ttorq(i)
31 continue
c
do 32 i=1,3
tforce(i)=facefor(2,i)
32 continue
call vcross(cpright,tforce,ttorq)
do 33 i=1,3
facotor(2,i)=ttorq(i)
33 continue
c
do 34 i=1,3
tforce(i)=facefor(3,i)
34 continue
call vcross(cpbottom,tforce,ttorq)
do 35 i=1,3
facotor(3,i)=ttorq(i)
35 continue
c
do 36 i=1,3
tforce(i)=facefor(4,i)
36 continue
call vcross(cprear,tforce,ttorq)
do 37 i=1,3
facotor(4,i)=ttorq(i)
37 continue
c
do 38 i=1,3
tforce(i)=facefor(5,i)
38 continue
call vcross(cpleft,tforce,ttorq)
do 39 i=1,3
facotor(5,i)=ttorq(i)
39 continue

```



```

C      F2=FD (X+H/2, Y+F1* H/2),   YT=YT+F2* H/3
C      F3=FD (X+H/2, Y+F2* H/2),   YT=YT+F3* H/3
C      F4=FD (X+H, Y+F3* H) ,      YT=YT+F4* H/6
C      X=X+H,
C      NEXT STEP
C
C      DIMENSION Y(12),YSTART(12),YTEMP(12),FDERIV(12)
H2=H/2.0
H3=H/3.0
H6=H/6.0
X0=X
DO 200 I=1,N
  YSTART(I)=Y(I)
  CALL SUB(FDERIV,Y,X)
DO 300 I=1,N
  YTEMP(I)=YSTART(I)+FDERIV(I)*H6
300  Y(I)=YSTART(I)+FDERIV(I)*H2
C      C *** END OF FIRST PASS ***
C
X=X0+H2
CALL SUB(FDERIV,Y,X)
DO 400 I=1,N
  YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
400  Y(I)=YSTART(I)+FDERIV(I)*H2
C      C *** END OF SECOND PASS ***
C
CALL SUB(FDERIV,Y,X)
DO 500 I=1,N
  YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
500  Y(I)=YSTART(I)+FDERIV(I)*H
C      C *** END OF THIRD PASS ***
C
X=X0+H
CALL SUB(FDERIV,Y,X)
DO 600 I=1,N
  Y(I)=YTEMP(I)+FDERIV(I)*H6
600

```



```

common/aerotorq/t_aero_b           ! so that this data is current at print time. (6/27/00)

C
C
C **** Post processing for a single time point ****
C **** (supports standard centralized integration method) ****
C ****
C
C fgmag=dssqrt(f_grav(1)**2+f_grav(2)**2+f_grav(3)**2) !gravity force magnitude, newtons
C
C famag=dssqrt(f_aero(1)**2+f_aero(2)**2+f_aero(3)**2) !aerodynamic force (drag)
C
C
C alt=(rmag/1000.0D0-rearth)          !altitude, kilometers
C
C theta=datan2(x(2),x(1))            !true anomaly, radians
C theta=theta/cvdr                   !converting to degrees
C if(theta.lt.0.0) theta=theta+360.0   !range=0 to 360 deg.

C
C theta2=theta+add
if(theta2.ge.theta1) go to 1         !updates range correction factor after each revolution
add=add+360.0D0
theta2=theta2+360.0

C
C 1 continue

C
C *** limiting range of Euler angles for output: (added 9/20/2000)
C
C do 3 i=1,3
phi(i)=x(i+6)
3 continue
call dcm321(phi,amat)
call dcmang321(amat,phi)
do 2 i=1,3
      ! renaming the state variables, converting angular states to degree units
C
C xcv(i)=x(i)                         ! radius (meters)
xcv(i+3)=x(i+3)                      ! velocity (meters/sec)

```

```

    subroutine sub(xdot,x,time)
C2345678901234567890123456789012345678901234567890123456789012

```

```

C   ***
C   Construction began 4/28/2000, by Doug Nixon, TD55. The purpose is to
C   supply the dynamics xdot=F(x,time) for integration by rksys, a
C   fourth-order Runge-Kutta routine. The problem is to set up a
C   standard-method simulation of a simple multidisciplined system involving
C   orbital dynamics, vehicle attitude dynamics, and aerodynamics.
C
C   implicit double precision (a-h,o-z)
C
C   common/constants/gme
C   common/mass/xmass,xmoi,xmoi_inv,rcm
C   common/gains/a0,a1,b0,b1,c0,c1
C   common/tgg/t_gg
C   common/tcontrol/t_cont
C   common/gravforce/f_grav
C   common/aeroforce/f_aero,f_aero_b
C   common/aerotorque/t_aero_b
C   common/altitude/rmag,rearth
C
C   ***
C   definition of variables in common:
C
C   gme : universal gravitational constant times mass of the earth (lb-ft**2 slug)
C   xmass: spacecraft mass (slugs)
C   xmoi : spacecraft moment of inertia tensor (3x3 matrix, slug-ft**2)
C
C   dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)
C   dimension x(12),xdot(12)
C   dimension r(3),rnorm(3),rbody(3),rdot(3)
C   dimension v(3),vnorm(3),vbody(3),vdot(3)
C   dimension phi(3),phidot(3),omegadot(3)
C   dimension f_grav(3)
C   dimension t_gg(3)
C   dimension f_aero(3),f_aero_b(3)
C   dimension t_aero_b(3)
C   dimension t_cont(3)
C   dimension t_tot(3)
C   dimension cmat(3,3)
C   dimension amat(3,3),amat_t(3,3)

```

```
dimension temp1(3),temp2(3)
```

```
C C C The system has six degrees of freedom, or twelve states. The System states are  
C defined as follows:
```

```
C Orbit States (Vehicle Position States):
```

```
C C C x(1)=r(1) : x-component of orbit radius vector in inertial coordinates,  
C from origin of inertial system to c.m. of spacecraft (meters)  
C x(2)=r(2) : y-component  
C x(3)=r(3) : z-component  
  
C x(4)=v(1) : x-component of spacecraft velocity vector in inertial coordinates  
C (meters/second)  
C x(5)=v(2) : y-component  
C x(6)=v(3) : z-component
```

```
C Vehicle Rotational States:
```

```
C C C x(7)=phi(1) : Euler angle rotation from inertial to body system, about x-axis,  
C 3rd in sequence. (radians)  
C x(8)=phi(2) : About y-axis, 2nd in sequence.  
C x(9)=phi(3) : About z-axis, 1st in sequence.
```

```
C C C x(10)=omega(1) : Body rate, x-axis (radians/second)  
C x(11)=omega(2) : Body rate, x-axis  
C x(12)=omega(3) : Body rate, x-axis
```

```
C Aerodynamics States:
```

```
C C No states are defined because the aerodynamics model has been reduced to  
an algebraic system.
```

```
C *****
```

```
C r(1)=x(1) ! Changing from mathematical to physical notation.  
C r(2)=x(2)  
C r(3)=x(3)
```

```

v(1)=x(4)
v(2)=x(5)
v(3)=x(6)
phi(1)=x(7)
phi(2)=x(8)
phi(3)=x(9)
omega(1)=x(10)
omega(2)=x(11)
omega(3)=x(12)

C   call dcm321(phi,amat)           !computes transformation (DCM) from inertial
C                                         !to body coordinates

C   call trans(amat,amat_t)          !transformation from body to inertial coordinates

C   C *** Gravity forces ***
C   call vecnorm(r,rnorm,rmag)      ! normalized radius vector in inertial coordinates
C   gmag=gme*xmass/rmag**2

C   f_grav(1)=-gmag*rnorm(1)        ! result in inertial coordinates
C   f_grav(2)=-gmag*rnorm(2)
C   f_grav(3)=-gmag*rnorm(3)

C   C *** Gravity Gradient Torques ***
C
C   ggmul=3.0D0*gme/rmag**3
C   call matvec(amat,rnorm,rbody)    ! normalized radius vector in body coordinates
C   call matvec(xmoi,rbody,temp1)    ! [ I ]{ rbody}
C   call vcross(rbody,temp1,temp2)   ! { rbody} X[ I ]{ rnorm}

C   t_gg(1)=ggmul*temp2(1)          ! result in body coordinates
C   t_gg(2)=ggmul*temp2(2)
C   t_gg(3)=ggmul*temp2(3)

C   C *** Aerodynamic drag/torque calculations
C

```

```

call aerocomp(v,amat,f_aero_b,t_aero_b)          ! Inertial velocity and attitude are inputs
C                                         ! Body forces and torques (geometric origin)
C                                         ! are outputs

C   call vcross(rcm,f_aero_b,temp1)
t_aero_b(1)=t_aero_b(1)+temp1(1)                  ! transferring drag force from geometric
t_aero_b(2)=t_aero_b(2)+temp1(2)                  ! origin to center of mass and adding couple
t_aero_b(3)=t_aero_b(3)+temp1(3)

C   C *** Control System Torques ***
C
C   t_cont(1)=- (a0*phi(1)+a1*omega(1))           ! result in body coordinates
t_cont(2)=- (b0*phi(2)+b1*omega(2))
t_cont(3)=- (c0*phi(3)+c1*omega(3))

C   C *** orbital dynamics (Orbit State Derivatives) ***
C
C   call matvec(amat_t,f_aero_b,f_aero)            ! transform aerodynamic drag vector
C                                         ! from body to inertial coordinates

C   vdot(1)=(1.0D0/xmass)*(f_grav(1)+f_aero(1))  ! result in inertial coordinates
vdot(2)=(1.0D0/xmass)*(f_grav(2)+f_aero(2))
vdot(3)=(1.0D0/xmass)*(f_grav(3)+f_aero(3))

C
rdot(1)=v(1)
rdot(2)=v(2)
rdot(3)=v(3)

C   C *** Attitude dynamics
C
C   call matvec(xmoi,omega,temp1)
call vcross(omega,temp1,temp2)
t_tot(1)=t_GG(1)+t_aero_b(1)+t_cont(1)-temp2(1) ! result in body coordinates
t_tot(2)=t_GG(2)+t_aero_b(2)+t_cont(2)-temp2(2)
t_tot(3)=t_GG(3)+t_aero_b(3)+t_cont(3)-temp2(3)
call matvec(xmoi_inv,t_tot,omegadot)               ! result in body coordinates

C   C Converting body rates to Euler angle derivatives

```

```

C   (Rotation sequence precludes 90deg angle about y-axis)
C
Cmat(1,1)=1.0D0
Cmat(1,2)=dsin(phi(1))*dtan(phi(2))
Cmat(1,3)=dcos(phi(1))*dtan(phi(2))
Cmat(2,1)=0.0D0
Cmat(2,2)=dcos(phi(1))
Cmat(2,3)=-dsin(phi(1))
Cmat(3,1)=0.0
Cmat(3,2)=dsin(phi(1))/dcos(phi(2))
Cmat(3,3)=dcos(phi(1))/dcos(phi(2))
call matvec(cmat, omega, phidot) ! result in non-orthogonal Euler-frame coordinates

C *** Changing back to mathematical notation from physical notation.
C
xdot(1)=rdot(1)
xdot(2)=rdot(2)
xdot(3)=rdot(3)
xdot(4)=vdot(1)
xdot(5)=vdot(2)
xdot(6)=vdot(3)

C
xdot(7)=phidot(1)
xdot(8)=phidot(2)
xdot(9)=phidot(3)
xdot(10)=omegadot(1)
xdot(11)=omegadot(2)
xdot(12)=omegadot(3)

C
return
end

```

REFERENCES

1. ED11 (12-98-02), "ED Lab High-Fidelity Multidiscipline Simulation," January 1998.
2. ED11 (12-98-114), "ED Lab High-Fidelity Multidiscipline Simulation," June 1998.
3. ED11 (12-98-246), "A Recursive Feedback Algorithm for Simulation of Dynamic Systems," December 1998.
4. ED11 (12-98-251), "An Assorted Collection of Analyses that Support the Concept of Recursive Feedback for Simulation of Dynamic Systems," December 1998.
5. ED11 (12-98-255), "Analytical Demonstration of Solution by Recursive Feedback for a Linear System Composed of Internally Inaccessible Subsystems," December 1998.
6. TD55 (00-18), "Analytical Demonstration of Recursive Feedback as a Solution Method for Simulation of Linear Dynamical Systems with Algebraic Loops," August 2000.
7. Ogata, K.: *State Space Analysis of Control Systems*, Prentice Hall, Englewood Cliffs, NJ, pp. 12-15, 1967.
8. Greenwood, D.T.: *Principles of Dynamics*, Prentice Hall, Englewood Cliffs, NJ, 1965.
9. Wertz, J.R.; and Wiley, J.L.: *Spacecraft Mission Analysis and Design*, 3rd ed., Microcosm Press, Torrance, CA, p. 324, 1999.

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operation and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503			
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED	
	October 2001	Technical Publication	
4. TITLE AND SUBTITLE		5. FUNDING NUMBERS	
System Simulation by Recursive Feedback: Coupling a Set of Stand-Alone Subsystem Simulations			
6. AUTHORS			
D.D. Nixon			
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)		8. PERFORMING ORGANIZATION REPORT NUMBER	
George C. Marshall Space Flight Center Marshall Space Flight Center, AL 35812		M-1032	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
National Aeronautics and Space Administration Washington, DC 20546-0001		NASA/TP—2001-211331	
11. SUPPLEMENTARY NOTES			
Prepared for Vehicle and System Development Department, Space Transportation Directorate			
12a. DISTRIBUTION/AVAILABILITY STATEMENT		12b. DISTRIBUTION CODE	
Unclassified-Unlimited Subject Category 59 Standard Distribution			
13. ABSTRACT (Maximum 200 words)			
Conventional construction of digital dynamic system simulations often involves collecting differential equations that model each subsystem, arranging them to a standard form, and obtaining their numerical solution as a single coupled, total-system simultaneous set. Simulation by numerical coupling of independent stand-alone subsimulations is a fundamentally different approach that is attractive because, among other things, the architecture naturally facilitates high fidelity, broad scope, and discipline independence. Recursive feedback is defined and discussed as a candidate approach to multidiscipline dynamic system simulation by numerical coupling of self-contained, single-discipline subsystem simulations. A satellite motion example containing three subsystems (orbit dynamics, attitude dynamics, and aerodynamics) has been defined and constructed using this approach. Conventional solution methods are used in the subsystem simulations. Distributed and centralized implementations of coupling have been considered. Numerical results are evaluated by direct comparison with a standard total-system, simultaneous-solution approach.			
14. SUBJECT TERMS		15. NUMBER OF PAGES	
system simulation, coupled subsystems, multidiscipline, high fidelity, recursive, feedback, dynamic		196	
		16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

National Aeronautics and

Space Administration

AD33

George C. Marshall Space Flight Center

Marshall Space Flight Center, Alabama

35812

