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

October 2001
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.
# 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

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARSYAS</td>
<td>Marshall System for Aerospace Simulation</td>
</tr>
<tr>
<td>MSFC</td>
<td>Marshall Space Flight Center</td>
</tr>
<tr>
<td>ODE</td>
<td>ordinary differential equation</td>
</tr>
<tr>
<td>RF–C</td>
<td>centrally coupled recursive feedback</td>
</tr>
<tr>
<td>RF–D</td>
<td>distributively coupled recursive feedback</td>
</tr>
<tr>
<td>RK–U</td>
<td>unified Runge-Kutta</td>
</tr>
</tbody>
</table>
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
Cd  aerodynamic drag coefficient
cm  center of mass
cpi  center of pressure for the ith facet
d  drag
e  base of natural logarithm
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
n  unit vector normal to surface
R  orbit radius
R  orbit radius vector
rcm  center of mass location vector
rcpi  center of pressure location vector for the ith facet
r  unit vector in direction of orbit radius
satellite

\( \bar{T} \) torque

\( t \) time

\( U \) column matrix collection of all subsystem input vectors

\( u \) subsystem input vector

\( V \) orbit velocity

\( \dot{V} \) orbit velocity vector

\( \hat{y} \) 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

\( \rho \) atmospheric density

\( \phi \) attitude angle

\( \dot{\phi} \) attitude rate

\( \omega \) body rate

\( \bar{\omega} \) angular velocity vector

\( \dot{\omega} \) angular velocity vector
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

\[ \frac{dy(t)}{dt} = ky(t), \]

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

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

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

\[ y^{(0)}(t) = y(0), \]

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

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

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

\[ y^{(2)}(t) = y(0)[1 + kt + \frac{(kt)^2}{2!}]. \]
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 $y(t) = y(0) \left( e^{kt} \right)$. 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 $\omega$ 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 $y(t) = y(0) \left( \cos \omega t \right)$. 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

\[ \dot{Y}(t) = KY(t) \]  

(7)

the coupling matrix is given by

\[ K = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix} \]  

(8)

and the output function is given by

\[ Y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} \]  

(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

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

(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.

Dimensions:
Length (L): 15 m
Width (W): 4 m
Height (H): 3 m

Center of Mass Offset:
X = 0.1 m
Y = 0.1 m
Z = 0.1 m

Notes:
1. _ depicts center of mass location.
2. Roll, pitch, and yaw are Euler angles about the X, Y, and Z axes, respectively. The sequence is yaw-pitch-roll.
3. All axes penetrate facets at their geometric centers.

Aerodynamic Data:
Drag Coefficient: 2.25
Density: $1.95 \times 10^{-11}$ kg/m$^2$

Inertial Pointing Control Gains:
Position: 1,000.0 N-m/rad, All Axes
Rate: 100.0 N-m-sec/rad, All Axes

Mass Properties:
Mass: 10,000 kg
Moments of Inertia:
\[
\begin{bmatrix}
11,250.0 & 0 & 0 \\
0 & 193,125.0 & 0 \\
0 & 0 & 173,812.5
\end{bmatrix}
\] kg$\cdot$m$^2$

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

4.1 Orbit Dynamics

The orbit dynamics is modeled by

\[ m_{\text{sat}} \frac{d^2 \vec{R}}{dt^2} = \vec{F}_{\text{grav}} + \vec{F}_{\text{aero}}, \tag{11} \]

where

\[ \vec{F}_{\text{grav}} = -\frac{G m_{\text{Earth}} m_{\text{sat}}}{R^3} \vec{R}. \tag{12} \]

\( G \) is the universal gravitational constant, \( m_{\text{Earth}} \) and \( m_{\text{sat}} \) 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. \( \vec{F}_{\text{aero}} \), 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

\[ I \dot{\vec{\omega}} + \vec{\omega} \times I \vec{\omega} = \vec{T}_{\text{control}} + \vec{T}_{\text{gg}} + \vec{T}_{\text{aero}}, \tag{13} \]

where \( I \) is the moment of inertia matrix, \( \vec{\omega} \) is the angular velocity vector, \( \vec{T}_{\text{gg}} \) is gravity gradient torque, \( \vec{T}_{\text{aero}} \) 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

\[
\begin{bmatrix}
\dot{\phi}_x \\
\dot{\phi}_y \\
\dot{\phi}_z \\
\end{bmatrix} = \begin{bmatrix}
1 & \sin \phi_x \tan \phi_y & \cos \phi_x \tan \phi_y \\
0 & \cos \phi_x & -\sin \phi_x \\
0 & \sin \phi_x \sec \phi_y & \cos \phi_x \sec \phi_y
\end{bmatrix}
\begin{bmatrix}
\omega_x \\
\omega_y \\
\omega_z \\
\end{bmatrix}, \tag{14}
\]
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

$$
A = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos\phi_x & \sin\phi_x \\
0 & -\sin\phi_x & \cos\phi_x
\end{bmatrix}
\begin{bmatrix}
\cos\phi_y & 0 & -\sin\phi_y \\
0 & 1 & 0 \\
\sin\phi_y & 0 & \cos\phi_y
\end{bmatrix}
\begin{bmatrix}
\cos\phi_z & \sin\phi_z & 0 \\
-\sin\phi_z & \cos\phi_z & 0 \\
0 & 0 & 1
\end{bmatrix},
$$

The attitude control system is modeled by

$$
-\mathbf{T}_{\text{control}} = 
\begin{bmatrix}
da_{0,x}\phi_x + a_{1,x}\omega_x \\
da_{0,y}\phi_y + a_{1,y}\omega_y \\
da_{0,z}\phi_z + a_{1,z}\omega_z
\end{bmatrix},
$$

where $\phi$, $\omega$, 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 by

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

where $\hat{r}$ is a unit vector corresponding to $\mathbf{R}$ 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 formula

$$
\overline{F}_i = \begin{cases} 
-\frac{1}{2} \rho V^2 C_d A_i (\hat{n}_i \cdot \hat{v}) \hat{v} & \text{if } \hat{n} \cdot \hat{v} \geq 0 \\
0 & \text{if } \hat{n} \cdot \hat{v} \leq 0
\end{cases},
$$
where $\rho$ 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 $\hat{V}$. If rotation of the Earth is neglected, no distinction between relative and absolute velocity need be made. The total aerodynamic force is given by

$$F = \sum_{i=1}^{6} \hat{F}_i.$$  \hspace{1cm} (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

$$\bar{T} = \sum_{i=1}^{6} \bar{r}_{cpi} \times \vec{F}_i ,$$  \hspace{1cm} (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

$$\bar{T}_{cm} = \bar{T} - (\bar{r}_{cm} \times \vec{F}),$$  \hspace{1cm} (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

\[ X = [R \mid V \mid \phi \mid \omega]^T, \]  

where R, V, \phi, and \omega 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

\[ \dot{X} = f(X, t), \]  

where

\[ X = [R \mid V \mid \phi \mid \omega]^T, \]  

\[ f(X, t) \]  

are the system vector and the system function, respectively.
Figure 4. Centrally coupled simulation diagram.

Figure 5. Function of system-level coupler.
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)

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).
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, Velocity, x-Component (meters/sec)

- Q: Full-control, standard method
- X: Full-control, measure method

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

- Q: Full-control, standard method
- X: Full-control, measure method
Case 1. Aerodynamic Torque, --Component (newton-meters)

- Full control, standard method
- Full control, reverse method

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

- Full control, standard method
- Full control, reverse method

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

- Full control, standard method
- Full control, reverse method

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

- Full control, standard method
- Full control, reverse method
A.2 Response Plots for Case 2 (Two-Axis Attitude Control)
Case 2, Body Rate, y-15s (meters/sec)

Case 2, Body Rate, z-15s (meters/sec)
Case 2, Aerodynamic Force, x-Component (newtons)

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

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

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

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

Case 2, Aerodynamic Force, z-Component (newtons)
Case 2, Mechanical Torque, y-Component (newton meters)
A.3 Response Plots for Case 3 (No Attitude Control)
Case 3, Euler angle, $\alpha$ is (degree)

- No control, standard method
- No control, recursive method

Case 3, Euler angle, $\beta$ is (degree)

- No control, standard method
- No control, recursive method

Case 3, Euler angle, $\gamma$ is (degree)

- No control, standard method
- No control, recursive method
A.4 Response Plots for Cases 1–3 With Central and Distributed Coupling
Case 2, Euler Angle, $z=15$ (degrees)

- Serial subsystems with distributed coupling
- Parallel subsystems with centralized coupling

Case 2, Euler Angle, $y=15$ (degrees)

- Serial subsystems with distributed coupling
- Parallel subsystems with centralized coupling

Case 2, Body Rate, $-x=15$ (deg/sec)

- Serial subsystems with distributed coupling
- Parallel subsystems with centralized coupling

Case 2, Body Rate, $-y=15$ (deg/sec)

- Serial subsystems with distributed coupling
- Parallel subsystems with centralized coupling
Case 3, Body Rate, $-\omega_M$ (deg/sec)

- Serial subsystems with distributed coupling
- Parallel subsystems with centralized coupling

Case 3, Body Rate, $\omega_M$ (deg/sec)

- Serial subsystems with distributed coupling
- Parallel subsystems with centralized coupling

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

- Serial subsystems with distributed coupling
- Parallel subsystems with centralized coupling

88
APPENDIX B—FORTRAN LISTINGS

B.1 FORTRAN Listings, Including Main Program and All Subroutines, Recursive Feedback With Centralized Coupling (RFC)
This is FORTRAN main program "rkc.f". Development began 6/12/2000 by Doug Nixon, MSFC/TD55, in support of AEI project, and the listing is contained in "aei/rkc"

The purpose is to generate the dynamic solution of three coupled subsystems (orbit dynamics, attitude dynamics, and aerodynamics) by recursive feedback, which involves distributed and possibly multiple methods of numerical integration. The intent is to compare results generated by a standard approach using a centralized single numerical integration method. The standard approach has been implemented as FORTRAN program "rku.f", and the main program is contained in "aei/rku"

*** Declarations

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

dimension uu(18,100),yy(18,100),yp(18,100) !18 input and 18 output signals, 100 time points

dimension uu(6,100),yy(6,100)
dimension uu(6,100),yy(6,100)
dimension uu(6,100),yy(6,100)

!input and output signals for three subsystems,
!six signals per subsystem, 100 time points

dimension yyl(6),yy(6),yy(6)
dimension yyl(6),yy(6),yy(6)

!initial conditions, 3 subsystems, 6 outputs
!final conditions, 3 subsystems, 6 outputs

common/system/uu,yy,yp !system input/output/"previous" output, all vs. time

common/subsys1/uu,yy,yp !orbit dynamics, i/o vs. time
common/endcond1/yy,yp !initial and final output conditions

common/subsys2/uu,yy,yp !attitude dynamics, i/o vs. time
common/endcond2/yy,yp !initial and final output conditions

common/subsys3/uu,yy,yp !aerodynamics, i/o vs. time
common/endcond3/yy,yp !initial and final output conditions

common/a%%%%%%%%%Simulation/ndp,delt, toi,res,t_start,
lt_report, t_print, t_stop, nmt, nmtr, nit, !all except some single-time-point utility routines
2t_initial, t_final
common/simulation2/del_report, del_print, ptol  ! percent done report interval, print interval
common/constants/rearth,gme, pi, cvdr         ! recursive.f, orbit.f, subl, attitude.f, sub2.f

*** Definition of the fundamental variables **********************************************

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

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

******************************************************************************************

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

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

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

******************************************************************************************

yy1(i,j)----- First subsystem output signal. Represents the ith component at the jth time point for the vector output signal to the first subsystem, and is also the first partition of yy(i,j).
point for the vector output signal from the first (orbit dynamics) subsystem, and is also the first (top) partition of yy(i,j).

yy2(i,j)---- Second subsystem output signal. Represents the ith component at the jth time point for the vector output signal from the second (attitude dynamics) subsystem, and is also the second (middle) partition of yy(i,j).

yy3(i,j)---- Third subsystem output signal. Represents the ith component at the jth time point for the vector output signal to the third (aerodynamics) subsystem, and is also the third (bottom) partition of yy(i,j).

*****************************************************************************

yy1l(i) ---- Value of the output signal at the beginning of the current convergence interval for the FIRST (orbit dynamics) subsystem. Represents the ith component of the signal at the first time point, and is equivalent to yy1(i,1)

yy2l(i) ---- Value of the output signal at the beginning of the current convergence interval for the SECOND (attitude dynamics) subsystem. Represents the ith component of the signal at the first time point, and is equivalent to yy2(i,1)

yy3l(i) ---- Value of the output signal at the beginning of the current convergence interval for the THIRD (aerodynamics) subsystem. Represents the ith component of the signal at the first time point, and is equivalent to yy3(i,1)

*****************************************************************************

yy1f(i) ---- Value of the output signal at the end of the current convergence interval for the FIRST (orbit dynamics) subsystem. Represents the ith component of the signal at the last time point, and is equivalent to yy1(i,ndp)

yy2f(i) ---- Value of the output signal at the end of the current convergence interval for the SECOND (attitude dynamics) subsystem. Represents the ith component of the signal at the last time point, and is equivalent to yy2(i,ndp)

yy3f(i) ---- Value of the output signal at the end of the current convergence interval for the THIRD (aerodynamics) subsystem. Represents the ith component of the vector signal yy3(i,j) at the last time point, and is equivalent to yy3(i,ndp)
The above nomenclature has been designed to allow implementation of recursive feedback via
FORTRAN in such a way that independence of the subsystem simulations can be supported. The
partitions and contractions facilitate autonomy of the subsystem simulations while providing
appropriate structure for implementation of the system-level algorithm. The fact that subsystem
data has been placed in COMMON with the MAIN (system level) program does not necessarily mean
that all of that data is required or desired at the system level -- it may mean only that the
SUBROUTINE containing the subsystem simulation needs to have access to data that it generated
previously, and placing that data in COMMON is a way to provide that in FORTRAN.

While the subsystem simulation SUBROUTINES will not literally be executed in parallel, the
information flows in parallel paths, and the order of execution is not important. Therefore,
parallel execution is conceptually possible without impact to the structure of the process.

This program, which involves three subsystems, can serve as a template for implementation with any
number of subsystems. Use of higher order arrays to implement the general case for "n" subsystems
appears to serve no useful purpose in this setting at this time.

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

*** Establishing the output file ******************************************************

open(unit=1,file='rfc.out1',status='unknown')

*** Defining system-level simulation parameters ****************************************

print*, 'This is main program rfc.f. Solution'
print*, 'method is recursive feedback with'
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'

*** Populating the subsystem databases

 call orbit(1) !defines parameter values (data) for subsystem 1 (orbit dynamics, mode 1)

 call attitude(1) !defines parameter values (data) for subsystem 2 (orbit dynamics, mode 1)

 call aerodyn(1) !defines parameter values (data) for subsystem 3 (orbit dynamics, mode 1)

 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

 nmtr=int(t_stop/dtmaj)+1 !number of major time (convergence) intervals required for completion of
 !the simulation. (Time may exceed t_stop by as much as one major interval)
```
c
tol=(1.0D0)*1.0e-12 !relative
tol=(1.0D0)*1.0e-12 !absolute
itmax=100 !maximum allowable number of iterations
ptol=1.0e-3 !tolerance for print time
print*,' Run in progress.'
print*
call clock(n1) !machine start time

*** Initializing time and convergence interval count: -----------------------------------------------------

time=0.0 !time is measured from start of the simulation
ntm=0 !number of major time intervals (convergence intervals) simulated
t_print=0.0 !initializes print time
t_report=0.0 !initializes percent done report time

*** Setting up initial conditions for each sub-simulation at system simulation outset

call orbit(2) !mode 2, original initial conditions setup, orbit dynamics
call attitude(2) !mode 2, original initial conditions setup, attitude dynamics
c
!no initial conditions setup for aerodynamics (algebraic system)

100 continue !******************************COMPUTING INITIAL RECURSION STAGE DATA******************************

nit=0 !number of iterations initialized to zero for first (new) convergence interval

*** Initializing the system-level feedback signal to zero: -----------------------------------------------

do 1 i=1,18 !eighteen scaler feedback signals
do 1 j=1,ndp !ndp number of data (time) points per scaler signal
uu(i,j)=0.0D0 !algorithm ignores feedback at recursion stage number (0)
1 continue

*** Decomposing the system-level feedback vector into individual subsystem input vectors
call decomp(6,ndp,uu,uu1,uu2,uu3) !six scaler input signals per subsystem

call orbit(3)  ! mode 3, run orbit dynamics simulation over convergence interval

call attitude(3)  ! mode 3, run attitude dynamics simulation over convergence interval

call aerodyn(3)  ! mode 3, run aerodynamics (algebraic) simulation over convergence interval

*** Assemble subsystem outputs into single system-level output ******************************************
call assemble(6,ndp,yy1,yy2,yy3,yy)

call coupler(yy,uu) !computes system-level feedback signal (subsystem coupling)

    do 2 i=1,18       ! sets initial recursion stage data to current values
        do 2 j=1,ndp
            yy(i,j)=yy(i,j)
        2 continue

200 continue !************************************************************************RECURSION LOOP ENTRY POINT************************************************************************

*** Decomposing the system-level feedback vector into individual subsystem input vectors

call decomp(6,ndp,uu,uu1,uu2,uu3) !six scaler input signals per subsystem

*** Running individual subsystem simulations over the convergence interval

call orbit(3)  ! mode 3, run orbit dynamics simulation over convergence interval

call attitude(3)  ! mode 3, run attitude dynamics simulation over convergence interval

call aerodyn(3)  ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
*** Assemble subsystem outputs into single system-level output

```
call assemble(6,ndp,yy1,yy2,yy3,yy)
```

*** Check for convergence

```
call converge(18,yy,yyp,iconflag)  ! system-level convergence check
     ! iconflag=0 if converged, 1 if
     ! not converged
idirect=iconflag+1
500 format(//,2i5,/
     go to (6,5) idirect  ! 6 for converged, 5 for not converged
```

5 continue !******************** Not CONVERGED, ADVANCE RECURSION STAGE

```
call coupler(yy,uu)  !computes system-level feedback signal (subsystem coupling)
```

```
do 7 i=1,18  ! number of scaler signals in the system
do 7 j=1,ndp  ! number of time points over each convergence interval,
     ! includes both end points
     yyn(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)
```

```
if(nit.ge.itmax) go to 300  ! protection against persistent non convergence
```

```
go to 200  ! returns to "top" of the recursion loop to run subsystem simulations again
```

6 continue ! ** CONVERGED, ADVANCE TO NEXT CONVERGENCE INTERVAL (or stop if simulation is complete)**

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

*** Proceed ("hand-off") to next major time (convergence) interval

In a more general setting, this would be done at the subsystem level (mode 4) so that the system level is not required to have knowledge of all subsystem states.

do 8 i=1,6
yy1f(i)=yy(i,ndp)
yy2f(i)=yy(i+6,ndp)
yy3f(i)=yy(i+12,ndp)
8 continue

!extracts subsystem final conditions for the current convergence interval
!from the final time point of the system response associated with the current
!convergence interval. SEE PARAGRAPH NUMBER 4 of "General Comments" ABOVE.

do 9 i=1,6
yy1i(i)=yy1f(i)
yy2i(i)=yy2f(i)
yy3i(i)=yy3f(i)
9 continue

!sets initial conditions (for all states) for the next convergence interval
!equal to conditions reached at the end (final conditions) of current
!convergence interval.

go to 100

!run subsystem simulations for the first time across the new (next)
!convergence interval.

300 continue

if(nit.ge.itmax) print*, 'Max. allowable number of'
if(nit.ge.itmax) print*, 'iterations reached.'

call clock(n2) 'machine stop time
irun-n2-n1 'run-time

if(nmt.eq.nmtr) print*, ' Normal exit, run-time = ',irun,' sec.'

print

close(unit=1,status='keep')

stop
subroutine aerocompr(vel_b, force, torque)

Developed by Doug Nixon, TD55, MSFC starting 5/26/2000 to compute
aerodynamic forces and torques for the "truth model" for recursive
feedback simulation method comparison for AEI study.

Modified by Doug Nixon, TD55, MSFC from standard/aerocomp.f starting 6/21/2000 for operation
in the recursive feedback setting. The attitude DCM was removed from the argument because the
inertial to body transformation is now performen by the system-level coupler.

rho = Atmospheric density, kg/m**3
vel_b(i) = Velocity vector in body coordinates, meters/sec
force(i) = Aerodynamic force, body coordinates, newtons
torque(i) = Aerodynamic torque, body coordinates, newton-meters

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

dimension vel_b(3), force(3), torque(3)
dimension vnrm_b(3)
dimension cpfront(3), cpright(3), cpbottom(3)
dimension cprear(3), cpleft(3), cptop(3)
dimension afront(3), airight(3), abottom(3)
dimension arear(3), aleft(3), atop(3)
dimension aproj(6), drag(6), facefor(6, 3), facetor(6, 3)
dimension tforce(3), ttorq(3)
c
common/aerobox/cpfront, cpright, cpbottom,      !aerody.n.f, aerocompr.f
lprear, cpleft, cptop, afront, aright, abottom,
2area, aleft, atop

c
common/aerodat/rho, cd                        !aerody.n.f, aerocompr.f

c
******************************************************************************************

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

c *** computing magnitude drag for each of six facets

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

do 20 i=1, 6
   if(aproj(i) .le. 0.000) aproj(i) = 0.000  ! negative projection implies ith facet
     ! not exposed to velocity vector
   continue

20    

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

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
     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
     continue
23 continue

*** Aerodynamic Torques  ***********************************************

     do 30 i=1,3
        tforce(i)=facefor(1,i)  ! front facet contribution
     continue
     call vcross(cpfront,tforce,ttorq)
     do 31 i=1,3
        facetor(1,i)=ttorq(i)
        torque(i)=torque(i)+ttorq(i)
     continue
31 continue

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

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

     do 36 i=1,3
        tforce(i)=facefor(4,i)  ! adding rear facet contribution
     continue
36 continue
```
37 \text{ continue}

38 \text{ continue}

39 \text{ continue}

40 \text{ continue}

41 \text{ continue}

42 \text{ continue}

\text{ return}

\text{ end}
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

*** This is subprogram "aerodyn.f". Construction began 6/21/2000 by Doug Nixon, TD55, MSFC.
Its purpose is to generate aerodynamic forces and torques as a function of time when
given an orbit state vector and vehicle attitude definition as a function of time. It is
an algebraic system; no numerical integration is involved.

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

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

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

dimension cpfront(3), cpright(3), cbottom(3)
dimension cprear(3), cpleft(3), ctop(3)
dimension afront(3), aright(3), abottom(3)
dimension arear(3), aleft(3), atop(3)

common/subsys3/uu3, yy3

common/endcond3/yy3i, yy3f

common/simulation/ndp, delt, tol, res, t_start,
lt_report, t_print, t_stop, nmt, nmtr, nit,
2t_initial, t_final

common/aerobox/cpfront, cpright, cbottom,
lcprear, cpleft, ctop, afront, aright, abottom,
2arear, aleft, atop

common/aerodat/rho, cd

if(mode.eq.2 .or. mode.eq.3) go to 200
100 continue ! (mode=1, define constants, system parameters)

*** Setting Values of System Parameters (mode 1 function): ****************************

*** Density and drag coefficient: ***

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

cd=2.25 ! drag coefficient, same for all facets

*** "Panel" aerodynamics setup (six-panel box): ***

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")

Work with six facets ----

do 10 i=1,3 ! setting all vector components to 0.0D0

afront(i)=0.0D0 ! area vectors, normal to facet surfaces, through arear(i)=0.0D0 ! centers, positive (point) outward from center of aheight(i)=0.0D0 ! box, back sides not considered (no contribution)
aleft(i)=0.0D0
atop(i)=0.0D0
abottom(i)=0.0D0

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

afront(1)=ax ! setting non-0.000 components of facet area vectors
arear(1)=ax
aright(2)=ay
aleft(2)=-ay
abottom(3)=az
atop(3)=-az

cpfront(1)=x1/2.000 ! setting non-zero components of facet
cprear(1)=-x1/2.000 ! centers-of-pressure location vectors
cpright(2)=w/2.000
cpleft(2)=-w/2.000
cptop(3)=-h/2.000
cpbottom(3)=h/2.000

return

********************************************************************
********************************************************************
********************************************************************
********************************************************************
********************************************************************

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

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

return

********************************************************************
********************************************************************
********************************************************************
********************************************************************
********************************************************************
300 continue ! (mode=3, run simulation over the convergence interval)

*** Propagate Output Signals (mode 3 function): ***

*** begin propagation loop

    do 20 ip=1,ndp    ! propagates forces and torques across convergence interval,
        do 6 j=1,3
        rad_b(j)=uu3(j,ip)    ! extracts radius and velocity vectors at the ith time point from the
        vel_b(j)=uu3(j+3,ip)  ! subsystem 3 input signal array. Radius is expressed in inertial coordinates,
        6 continue
        ! velocity in body coordinates. Radius is not used now because density is
        ! defined to be constant --- will be used in later version.
        call aero compr(vel_b, force, torque) ! computes force and torque at a single time point
        do 7 j=1,3    ! enters output (force and torque vectors) into block common /susbsyl/.
            yy3(j,ip)=force(j)
            yy3(j+3,ip)=torque(j)
        7 continue
    20 continue

*** end propagation loop

return

*** End of mode 3

end
subroutine assemble(nsubdim, ndp, xx1, xx2, xx3, xx)

*** Assembles subsystem input or output signals into system-level input or output signals

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

dimension xx1(6,100), xx2(6,100), xx3(6,100), xx(18,100)

Assumes nsubdim number of scalar signals per subsystem

do 10 i=1, nsubdim
   do 10 j=1, ndp
      xx(i,j)=xx1(i,j)
      xx(i+6,j)=xx2(i,j)
      xx(i+12,j)=xx3(i,j)
   10 continue

return
end

subroutine attitude(mode) ! mode 1 - set physical constants, system parameters, etc.
   ! mode 2 - set initial conditions
   ! mode 3 - run simulation over the convergence interval

*** This is subprogram "attitude.f". Construction began 6/28/2000 by Doug Nixon, TD55, MSFC.
"attitude.f" is the main driver for the six-state dynamic system which is integrated
by "rksys2.f", a fourth-order Runge-Kutta subroutine. "sub2.f" defines the
dynamics in first-order state-variable form.

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


c  dimension uu2(6,100),yy2(6,100)
c  dimension yy2i(6),yy2f(6)
c  dimension x(6),xdot(6)
c  dimension phi(3),omega(3)
c  dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)
c
c  common/subsys2/uu2,yy2
  common/endcond2/yy2i,yy2f

  common/simulation/ndp,delt,tol,res,t_start,
  lt_report,t_print,t_stop,nmt,nmtr,nit,
  2t_initial,t_final

  common/constants/rearth,gme,pi,cvdr
  common/mass/xmass,xmoi,xmoi_inv,rcm
  common/gains/a0,a1,b0,b1,c0,c1

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

100 continue ! (mode=1, define constants, system parameters)

**-----------------------------------------------------------------------------------**
**-----------------------------------------------------------------------------------**
**-----------------------------------------------------------------------------------**
**-----------------------------------------------------------------------------------**

Gravity constant (gme) is defined by orbit.f, and is obtained through block
common /constants/; pi (3.14xx...) and cvdr (deg. to rad. mult. factor) are
are defined and obtained similarly.
c
************
Setting Values of System Parameters (mode 1 function): ************
c
Satellite mass (xmass) is defined by orbit.f; obtained here through common/mass/.

*** Moments of inertia and center of mass:

xmoi(1,1)=11250.0D0  ! spacecraft 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) ! inverse matrix

rcm(1)=1.0D0  ! center of mass location in geometric body
rcm(2)=0.1D0  ! coordinates, meters
rcm(3)=0.1D0

*** Control system gains:

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

print*,', Input stepsize reduction factor'
print*,', for ATTITUDE DYNAMICS simulation'
print*,', (integer):'
read*,idiv
return

********************
End of mode 1, beginning of mode 2

200 if(mode.eq.3) go to 300
    continue ! (mode=2, set initial conditions)

Setting Initial Conditions (mode 2 function):

phi(1)=0.0D0
phi(2)=0.0D0
phi(3)=0.0D0

omega(1)=0.0D0
omega(2)=0.0D0
omega(3)=0.0D0

do 5 i=1,3
    yy21(i)=phi(i)
    yy21(i+3)=omega(i)
5 continue

return

End of mode 2, beginning of mode 3

300 continue ! (mode=3, run simulation over the convergence interval)

Setting up the initial state vector (6 states, mode 2 function):

do 6 i=1,3
    !retrieving initial conditions from block common /endcond2/ (radians)
phi(i)=yy2(i)
omega(i)=yy2(i+3)

yy2(i,1)=phi(i)  !entering initial output vector (time point number 1) into common/subsys2/
yy2(i+3,1)=omega(i)

6 continue

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

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

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

continue !****************** begin integration loop *********************************************
do 20 i=2, ndp       ! generates points 2 through ndp

dt=delt/float(idiv)
do 30 ii=1, idiv
     call rksys2(tau, x, ndim, dt) ! propagates attitude states across convergence interval
30 continue

do 7 j=1, 6          ! enters output (Euler angles and body rates) into common/subsys2/
yy2(j,i)=x(j)
7 continue
     20 continue
     c
     c *********************************************************** end integration loop ***********************************************************
     c
     return
     c
     c *********************************************************** End of mode 3 ***********************************************************
     c
     c ***********************************************************
     c
     end

subroutine clock(n)
   integer 4 n, time
   n=time()
   return
end

subroutine converge(nsysdim,yy,yyp,iconflag)  !system-level convergence check
implicit double precision(a-h,o-z)

dimension yy(18,100), yyp(18,100)
dimension ynew(100), yold(100)

common/simulation/ndp,delt,tol,res,t_start,t_report,t_print,
lt_stop,nmt,nmtr,nit,t_initial,t_final

******************************************************************************
*** begin check loop *********************************************************
******************************************************************************

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).
       ynew(j)=yy(i,j)
       yold(j)=yyp(i,j)
   19 continue

19 continue

call convrg(ynew,yold,ndp,tol,res,iflag) ! iflag=1 for non-convergence.

iconflag=iconflag+iflag ! computes status of system-level convergence

20 continue

20 continue

if(iconflag.gt.0) iconflag=1 ! routine returns 0 or 1 for iconflag

******************************************************************************
*** end check loop ***********************************************************
******************************************************************************

return

end
SUBROUTINE CONVRC(YNEW, YOLD, NDP, TOL, RES, IFLAG)
C
C *** THIS ROUTINE COMPARES TWO CURVES FOR EQUALITY.
C
C YNEW(I) DEFINES THE CURRENT CURVE. YOLD(I) DEFINES THE PREVIOUS CURVE. NDP IS THE NUMBER OF DATA POINTS
C USED TO SPECIFY EACH CURVE. TOL IS THE TOLERANCE WITH WHICH THE RMS VALUE OF THE DIFFERENCE CURVE NORMALIZED WITH RESPECT TO
C RMS VALUE OF THE SUM CURVE IS COMPARED TO DETERMINE WHEN CONVERGENCE HAS BEEN ACHIEVED. IFLAG=0 INDICATES CONVERGENCE,
C IFLAG=1 INDICATES LACK OF CONVERGENCE.
C
C implicit double precision(a-h,o-z) !6/22/00
C
C DIMENSION YNEW(100), YOLD(100)
C
C IFLAG=1
C CHECK1=0.0
C
C DO 10 I=1, NDP
C 10 CHECK1=CHECK1+(YNEW(I)-YOLD(I))**2
C
C FNPD=DFLOAT(NDP) !6/22/00
C CHECK1=DSQRT(CHECK1/FNPD) !6/22/00
C
C YMAX=YNEW(1)
C YMIN=YNEW(1)
C
C DO 11 I=2, NDP
C IF(YNEW(I).GT.YMAX) YMAX=YNEW(I)
C
IF(YNEW(I).LT.YMIN) YMIN=YNEW(I)
C
11 CONTINUE
C
SPAN=dABS(YMAX-YMIN)
CHECK2=TOL*SPAN
C
IF(CHECK1.LE.CHECK2.OR.CHECK1.LE.RES) IFLAG=0
C
RETURN
END

subroutine coupler(yy,uu)
C
implicit double precision(a-h,o-z)
C
dimension uu(18,100),yy(18,100) !system-level I/O, 18 input and 18 output signals, 100 time points
C
dimension f_aero(3)
dimension f_aero_b(3), t_aero_b(3)
dimension f_aero_q(3), t_aero_q(3)
dimension rxf(3)
dimension rad(3),vel(3)
dimension rad_b(3),vel_b(3)
dimension phi(3)
dimension amat(3,3)
dimension amat_t(3,3)
dimension xmoi(3,3),xmoi_inv(3,3)
dimension rcm(3)

!aerodynamic force (drag), inertial coordinates
!aerodynamic force and torque, body frame
!aerodynamic force and torque, geometric frame
!aerodynamic force transfer couple (from geometric origin to c.m.)
!orbit radius and velocity vectors, inertial coordinates
!orbit radius and velocity vectors, body coordinates
!euler angles, inertial frame to body frame, 3-2-1 sequence
!direction cosine matrix (d.c.m.), inertial to body
!transpose of amat(i,j)
!inertia matrix, inverse inertia matrix
!center of mass location, geometric coordinates
common/simulation/ndp,delt,tol,rel, Nit_start,                  !all except some single-time-point utility routines
lt_report,t_print,t_stop,nmt,nmtr,nit,
2t_initial,t_final
common/mass/xmass,xmoi,xmoi_inv,rcm         !orb.f, subl.f, attitude.f, sub2.f, coupler.f

*****************************************************************************
Coupling Aerodynamics/Orbit Dynamics
*****************************************************************************

do 10 j=1,ndp

*** from aerodynamics to orbit dynamics:

f_aero_b(1)=yy(13,j)                  !extracting aerodynamic (drag) force from system output signal
f_aero_b(2)=yy(14,j)                  !computed to act at origin of geometric body frame,
f_aero_b(3)=yy(15,j)                  !but can be assumed to act at c.m. for this purpose

phi(1)=yy(7,j)                        !extracting attitude (Euler) angles from the system output signal
phi(2)=yy(8,j)
phi(3)=yy(9,j)

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

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

call matvec(amat_t,f_aero_b,f_aero)   !transforms force vector from body to inertial coordinates

uu(1,j)=f_aero(1)                     !feedback signal is aerodynamic force (drag) in inertial coordinates,
uu(2,j)=f_aero(2)                     !and is assumed to act at the satellite center of mass (c.m.)
uu(3,j)=f_aero(3)
uu(4,j)=0.000                      !slot holders, nothing yet identified
uu(5,j)=0.000
uu(6,j)=0.000

*** from orbit dynamics to attitude dynamics:
rad(1)=yy(1,j)  !Extracting orbit radius from the system output signal  
rad(2)=yy(2,j)  !variable rad(i) is used for listing readability only  
rad(3)=yy(3,j)  

uu(7,j)=rad(1)  !feedback signal is orbit radius vector in inertial coordinates  
uu(8,j)=rad(2)  
uu(9,j)=rad(3)  

*** from aerodynamics to attitude dynamics:  
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  

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)  

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

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

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

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

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)  

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

117
vel(2)=yy(5,j)
vel(3)=yy(6,j)

call matvec amat vel vel_b  !transforms orbit velocity vector to body coordinates
uu(16,j)=vel_b(1)
uu(17,j)=vel_b(2)
uu(18,j)=vel_b(3)

10 continue

c return
end

subroutine dcm2ang321 amat phi

*** computes equivalent 3-2-1 sequence Euler angles for d.c.m amat(i,j)

implicit double precision (a-h,o-z)
dimension amat(3,3), phi(3)

phi(1)=datan2 amat(2,3), amat(3,3)  ! (+/- pi  rad.)
phi(2)=-dasin amat(1,3)               ! (+/- pi/2  rad.)
phi(3)=datan2 amat(1,2), amat(1,1)   ! (+/- pi  rad.)

c return
end
subroutine dcm321(phi, amat)

*** Created 5/12/2000 to support "standard.f/sub.f". constructs
a DCM (direction cosine matrix) from the Euler angles phi(i).
The rotation sequence is 3-2-1, and the resulting DCM (amat)
transforms vector components FROM the original coordinate frame
(inertial) TO the final (body) coordinate frame.

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

dimension phi(3), amat(3,3), ax(3,3), ay(3,3), az(3,3)

cx=dcos(phi(1))
sx=dsin(phi(1))
cy=dcos(phi(2))
sy=dsin(phi(2))
cz=dcos(phi(3))
sz=dsin(phi(3))

do 1 i=1,3
do 1 j=1,3
ax(i,j)=0.0D0
ay(i,j)=0.0D0
az(i,j)=0.0D0
1 continue

ax(1,1)=1.0D0
ax(2,2)=cx
ax(2,3)=sx
ax(3,2)=-sx
ax(3,3)=cx

ay(2,2)=1.0D0
ay(1,1)=cy
ay(1,3)=-sy
ay(3,1)=sy
ay(3,3)=cy

az(3,3)=1.0D0
az(1,1)=cz
az(1,2)=sz
az(2,1)=-sz
az(2,2)=cz

d0 2 i=1,3
d0 2 j=1,3
amai(i,1)=0.0D0
d0 2 k=1,3
d0 2 j=1,3
amai(i,1)=amai(i,1)+ax(i,j)*ay(j,k)*az(k,l)
2 continue

creturn
cend

subroutine decomp(nsubdim,ndp,xx,xx1,xx2,xx3)
c*** Decomposes system-level input or output signals into subsystem input or output signals
cimplicit double precision (a-h,o-z)
c
dimension xx(18,100),xx1(6,100),xx2(6,100),xx3(6,100)
c
Assumes nsubdim number of scalar signals per subsystem
c
do 10 i=1,nsubdim
   !eighteen scaler signals at the system level,
do 10 j=1,ndp
   !ndp time points each
   xx1(i,j)=xx(i,j)
   xx2(i,j)=xx(i+6,j)
   xx3(i,j)=xx(I+12,j)
10 continue
c
return
c
end

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
   go to 1
3 continue
go to 1

3 continue    ! performing the interpolation

i=interval
fac=(time-t1)/delt
val=func(i)*fac*(func(i+1)-func(i))

return
end

SUBROUTINE INV3(A, AI)
C
C *** THIS ROUTINE TAKES THE INVERSE OF A 3X3 MATRIX.
C
C IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
DIMENSION A(3,3),AI(3,3),TEMP(3,3)
C
DET=A(1,1)* (A(2,2)*A(3,3)-A(3,2)*A(2,3))
C-A(2,1)* (A(1,2)*A(3,3)-A(3,2)*A(1,3))
C+A(3,1)* (A(1,2)*A(2,3)-A(2,2)*A(1,3))
TEMP(1,1)=A(2,2)*A(3,3)-A(3,2)*A(2,3)
TEMP(1,2)=A(3,1)*A(2,3)-A(2,1)*A(3,3)
TEMP(1,3)=A(2,1)*A(3,2)-A(3,1)*A(2,2)
TEMP(2,1)=A(3,2)*A(1,3)-A(1,2)*A(3,3)
TEMP(2,2)=A(1,1)*A(3,3)-A(3,1)*A(1,3)
TEMP(2,3)=A(3,1)*A(1,2)-A(1,1)*A(3,2)
TEMP(3,1)=A(1,2)*A(2,3)-A(2,2)*A(1,3)
TEMP(3,2)=A(2,1)*A(1,3)-A(1,1)*A(2,3)
\begin{verbatim}
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

SUBROUTINE MATVEC(A,V,AV)
C
C *** THIS ROUTINE MULTIPLIES A VECTOR BY A MATRIX.
C (3X3)*(3X1)=(3X1)
C
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
DIMENSION A(3,3),V(3),AV(3)
C
DO 10 I=1,3
AV(I)=0.0
DO 10 J=1,3
10 AV(I)=AV(I)+A(I,J)*V(J)
RETURN
END
\end{verbatim}
gravconst=(6.673D0)*(1.0e-11) ! Constant of Gravitation,  
! m**3/kg-sec**2, CRC, page 6

rearth= 6367.65D0 ! Earth's radius, (polar+equatorial)/2.0,  
! km, CRC, page 6

dearth=5522.0D0 ! Average density of the earth,  
! kg/m**3, CRC, page 6

pi=datan2(0.0D0,-1.0D0) ! Computes pi to machine precision

cvdr=pi/180.0D0 ! Converts degrees to radians

volume=(4.0D0/3.0D0)*pi*(rearth*1000.0D0)**3 ! Volume of earth, m**3

xmearth=dearth*volume ! Mass of the earth, kg

gme=gravconst*xmearth ! Gravity Constant times mass of the earth.

***********************************************************************
* Setting Values of System Parameters (mode 1 function): *******************
***********************************************************************

xmass=10000.0D0 ! spacecraft mass, kilograms

print*,! Input stepsize reduction factor'
print*,! for ORBIT DYNAMICS simulation'
print*,! (integer):'
read*,idiv return

***********************************************************************
* End of mode 1, beginning of mode 2 *************************************
***********************************************************************

200 if(mode.eq.3) go to 300
continue ! (mode=2, set initial conditions)
c

******************************************************************************
Setting Initial Conditions (mode 2 function):
******************************************************************************

alt=300.0D0                          ! altitude, kilometers

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

vmag=dsqrt(gme/rmag)                  ! Circular orbit, meters/sec
rcube=rmag**3
wsq=gme/rcube
ctperiod=2.0D0*pi/dsqrt(wsq)       ! Period for circular orbit
v(1)=0.0
v(2)=vmag
v(3)=0.0

do 5 i=1,3
   yyli(i)=r(i)
   yyli(i+3)=v(i)
5 continue

return

******************************************************************************
End of mode 2, beginning of mode 3
******************************************************************************

300 continue ! (mode=3, run simulation over the convergence interval)

******************************************************************************
Setting up the initial state vector (6 states, mode 2 function):
******************************************************************************
do 6 i=1,3
   r(i)=yy1(i)
v(i)=yy1(i+3)
   yy1(i,1)=r(i)
   yy1(i+3,1)=v(i)
6 continue

!entering initial output vector (time point number 1) into block !common /subsys1/

6 continue

do 10 i=1,3
   x(i)=r(i)
x(i+3)=v(i)
10 continue

! changing from kinematic to state variable notation
! states 1-3, orbit radius, meters
! states 4-6, orbit velocity, meters/sec

tau=0.0D0 !initializing time, measured from the start of the convergence interval
ndim=6 !defining dimension of the state vector
continue !********************************** begin integration loop *********************************************
do 20 i=2,ndp
dt=delt/float(idiv) ! smaller stepsize than delt
do 30 ii=1,idiv
   call rksys1(tau,x,ndim,dt) ! propagates orbit states across convergence interval
30 continue

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

! enters output (orbit state vector) into block common /susbsys1/.
! data (time) points 2 through ndp are dealt with here

continue

*************************************** end integration loop *********************************************
**Remark:**

The contents of the image suggest that it is a page from a technical document, possibly a computer program or a scientific text, in the form of a listing of a computer program. The text appears to be written in Fortran, a programming language, and includes keywords and structures typical of Fortran programming. The visible parts of the text indicate a subroutine named `rfc_out` and various declarations for variables and arrays. The text includes comments and directives that are common in Fortran, such as `implicit double precision` and `common`. The overall context suggests that this is a section of a larger program, possibly related to simulation or scientific computing, given the references to `simulation` and `system-level output`. The specific content and purpose of the program are not entirely clear from the visible text alone, but it involves numerical computations and data processing, as indicated by the declarations for arrays and variables.
common/constants/rearth,gme,pi,cvdr
common/mass/xmass,xmoi,xmoi_inv,rcm

!recursive.f, orbit.f, sub1, attitude.f, sub2.f
!orbit.f, sub1.f, attitude.f, sub2.f, coupler.f
!added 9/28/2000, for aero torque output

do 3 i=1,18
    do 3 j=1,ndp
        zz(i,j)=yy(i,j)
    continue

3 continue

do 4 i=1,3
    do 4 j=1,ndp
        phi(i)=zz(i+6,j)
        call dcm321(phi,amat)
        call dcm2ang321(amat,phi)
        zz(i+6,j)=phi(i)/cvdr
        zz(i+9,j)=yy(i+9,j)/cvdr

4 continue

!added 9/28/2000 for aero. torque computation for output

do 5 j=1,ndp
    f_aero(1)=zz(13,j)
    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

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

! begin first write loop

!post processing orbit data for altitude
rad(k)=yy(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
c 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 = ',
1 ipd
if(time.ge.t_report) t_report=t_report+del_report
100 format(20E20.10,i5)
time=time+delt
1000 continue
c c time=timel !re-initializing time to first point of convergence interval
c c do 2000 i=1,ndp-1 ! begin second write loop
c c write(2,100) time,(uu(j,i),j=1,18) !system-level feedback signal
c 200 format(19E20.10)
time=time+delt
2000 continue
c return
end

SUBROUTINE RKSYS1(X,Y,N,H) ! "1" added to identify with subsystem #1
! Subsystem #1 is orbit dynamics
C IMPlicit DOUBLE PRECISION(A-H,O-Z)
THIS SUBROUTINE PERFORMS ONE STEP IN THE RUNGE-KUTTA INTEGRATION OF A SYSTEM OF FIRST ORDER DIFFERENTIAL EQUATIONS OF THE FORM Y'(X)=F(X,Y), WHERE Y AND F ARE VECTOR FUNCTIONS, AND X IS THE INDEPENDENT VARIABLE. THE ALGORITHM IS DESCRIBED BY THE FOLLOWING VECTOR RELATIONS:

\[ \begin{align*}
F1 &= F(X,Y), \\
F2 &= F(X+H/2,Y+F1*H/2), \\
F3 &= F(X+H/2,Y+F2*H/2), \\
F4 &= F(X+H,Y+F3*H), \\
X &= X+H,
\end{align*} \]

\[ \begin{align*}
Y &= Y+Y* \frac{H}{6} \\
Y &= Y+Y* \frac{H}{3} \\
Y &= Y+Y* \frac{H}{3} \\
Y &= Y+Y* \frac{H}{6} \\
\text{NEXT STEP}
\end{align*} \]

DIMENSION Y(6), YSTART(6), YTEMP(6), FDERIV(6) ! Set for point mass orbit dynamics

H2=H/2.0
H3=H/3.0
H6=H/6.0
X0=X
DO 200 I=1,N
200 YSTART(I)=Y(I)
   CALL SUB1(FDERIV,Y,X) ! "1" added to identify with subsystem #1
   DO 300 I=1,N
   YTEMP(I)=YSTART(I)+FDERIV(I)*H6
   300 Y(I)=YSTART(I)+FDERIV(I)*H2

*** END OF FIRST PASS ***

X=X0+H2
   CALL SUB1(FDERIV,Y,X) ! "1" added to identify with subsystem #1
   DO 400 I=1,N
   YTEMP(I)=YTEMP(I)+FDERIV(I)*H3
   400 Y(I)=YSTART(I)+FDERIV(I)*H2

*** END OF SECOND PASS ***

CALL SUB1(FDERIV,Y,X) ! "1" added to identify with subsystem #1
   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 SUB1(FDERIV,Y,X) ! "1" added to identify with subsystem #1
DO 600 I = 1, N
600 Y(I) = YTEMP(I) + FDERIV(I) * H6

C
C *** END OF FOURTH PASS ***
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
RETURN
END

SUBROUTINE RKSYS2(X,Y,N,H) ! "2" added to identify with subsystem #2
IMPLICIT DOUBLE PRECISION(A-H,O-Z)

THIS SUBROUTINE PERFORMS ONE STEP IN THE RUNGE-KUTTA
INTEGRATION OF A SYSTEM OF FIRST ORDER DIFFERENTIAL
EQUATIONS OF THE FORM Y'(X) = FD(X,Y), WHERE Y AND FD
ARE VECTOR FUNCTIONS, AND X IS THE INDEPENDENT
VECTOR. THE ALGORITHM IS DESCRIBED BY THE FOLLOWING
VECTOR RELATIONS:
C
F1=F0(X,Y),

C
F2=F0(X+H/2,Y+H/2),

C
F3=F0(X+H/2,Y+H/2),

C
F4=F0(X+H, Y+F3*H),

C
X=X+H,

C

DIMENSION Y(6),YSTART(6),YTEMP(6),FDERIV(6) ! Set for rigid body rotational dynamics
H2=H/2.0
H3=H/3.0
H6=H/6.0
X0=X
DO 200 I=1,N
200 YSTART(I)=Y(I)
   CALL SUB2(FDERIV,Y,X) ! "Z" added to identify with subsystem #2
   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 SUB2(FDERIV,Y,X) ! "Z" added to identify with subsystem #2
   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 SUB2(FDERIV,Y,X) ! "Z" added to identify with subsystem #2
   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 SUB2(FDERIV,Y,X) ! "Z" added to identify with subsystem #2
   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
RETURN
END

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 "subsysl" variable uul(i,j), where the following
C definitions apply:
C
x(i) = value of the ith component of the orbit state vector. This is an
INPUT VARIABLE, and is not changed by this routine.
C
xdot(i) = time derivative of the ith component of the orbit state vector.
This is an OUTPUT VARIABLE, and is created by this routine.
C
tau = time elapsed from the beginning of the convergence interval. This
is an INPUT VARIABLE, and is not changed.
C
uul(i,j) = value at the jth time point of the x-component of body force f(t)
expressed in inertial coordinates. Units are newtons. This is an
INPUT VARIABLE, and is not changed.

\[ u\text{u}1(2,j) = \text{value at the } j\text{th time point of the } y\text{-component of body force } f(t) \]
expressed in inertial coordinates. Units are newtons. This is an INPUT VARIABLE, and is not changed.

\[ u\text{u}1(3,j) = \text{value at the } j\text{th time point of the } z\text{-component of body force } f(t) \]
expressed in inertial coordinates. Units are newtons. This is an INPUT VARIABLE and is not changed.

\[ y\text{y}1(i,j) = \text{value at the } j\text{th time point of the } i\text{th component of the orbit state vector. THIS VARIABLE IS NOT USED BY THIS ROUTINE.} \]

\[ \text{ndp} = \text{number of signal sample points per convergence interval, including both end points} \]

\[ \text{delt} = \text{time between signal sample points, same as integration stepsize.} \]

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

dimension u\text{u}1(6,100), y\text{y}1(6,100)

dimension f\_aero(3)

! aerodynamic force, single time point, inertial coordinates

dimension f\_grav(3)

! gravitational force, single time point, inertial coordinates

dimension r\text{n}orm(3)

! normalized radius vector, inertial coordinates

dimension func\_x(100), func\_y(100), func\_z(100) ! vector input signal, 100 time points

! (aerodynamic force)

dimension x(6), xdot(6), r(3), r\text{dot}(3), v(3), v\text{dot}(3)

dimension xmoi(3,3), xmoi\_inv(3,3) ! inertia matrix, inverse inertia matrix

dimension r\text{cm}(3)

! center of mass location, geometric coordinates

c\text{ommon}/subsys1/u\text{u}1,y\text{y}1

c\text{ommon}/endcond1/y\text{y}l1,y\text{y}lf

! orbit dynamics, input/output vs. time

! initial and final output conditions
common/simulation/ndp,delt,tol,res,t_start,
lt_report,t_print,t_stop,nmt,nmtr,nit,
2t_initial,t_final

common/constants/rearth,gme,pi,cvdr
! recursive.f, orbit.f, sub1, attitude.f, sub2.f
common/mass/xmass,xmoi,xmoi_inv,rcm
! orbit.f, sub1.f, attitude.f, sub2.f, coupler.f

*******************************************************************************

Orbit States (Vehicle position and velocity, inertial coordinates):

x(1)=r(1): x-component of orbit radius vector (meters)
x(2)=r(2): y-component
x(3)=r(3): z-component

x(4)=v(1): x-component of spacecraft velocity vector, (meters/second)
x(5)=v(2): y-component
x(6)=v(3): z-component

*******************************************************************************

r(1)=x(1) ! Changing from mathematical (state variable) to kinematic notation.
r(2)=x(2)
r(3)=x(3)
v(1)=x(4)
v(2)=x(5)
v(3)=x(6)

*** Gravity forces ***

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

f_grav(1)=-gmag*rnorm(1)
f_grav(2)=-gmag*rnorm(2) ! result in inertial coordinates
c
return
end

subroutine sub2(xdot,x,tau)
c*** Construction began 6/28/2000, by Doug Nixon, TD55. This is the attitude dynamics (subsystem 2)
routine driven by rksys2.f which supports attitude.f. Attitude dynamics is is the 2nd subsystem
of three (orbit dynamics, attitude dynamics, and aerodynamics) that form a system simulation
implemented by recursive feedback.

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

dimension uu2(6,100),yy2(6,100)
dimension x(6),xdot(6)
dimension phi(3),phidot(3),omega(3),omegadot(3)
dimension rad(3),rad_norm(3),rad_norm_b(3)
dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)
dimension t_gg(3)
dimension t_aero(3)
dimension t_control(3)
dimension t_total(3)
dimension cmat(3,3)
dimension amat(3,3),amat_t(3,3)
dimension h(3), wxh(3)
dimension xir(3), rxxir(3)
dimension func_x(100), func_y(100), func_z(100)
common/subsys2/uu2,yy2
common/simulation/ndp,delt,tol,req,esst,rs_report,rs_print,rs_stop,nmtn,nmtr,ni
2t_initial,t_final
common/consts/rear,ge,pi,cvdr
common/mass/xmass,xmoi,xmoi_inv,rcm
common/gains/a0,a1,b0,b1,c0,c1

*** Some variable definitions:

gme : universal gravitational constant times mass of the earth (lb-ft*2/slug)
xmass: spacecraft mass (slug)
xmoi : spacecraft moment of inertia tensor (3x3 matrix, slug-ft**2)

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

Vehicle Rotational States:

x(l)=phi(l): Euler angle rotation from inertial to body system, about x-axis,
3rd in sequence. (radians)

\[ x(2) = \phi(2) \]: About y-axis, 2nd in sequence.
\[ x(3) = \phi(3) \]: About z-axis, 1st in sequence.

\[ x(4) = \omega(1) \]: Body rate, x-axis (radians/second)
\[ x(5) = \omega(2) \]: Body rate, x-axis
\[ x(6) = \omega(3) \]: Body rate, x-axis

***************************************************************************************

\[ \phi(1) = x(1) \]
\[ \phi(2) = x(2) \]
\[ \phi(3) = x(3) \]

\[ \omega(1) = x(4) \]
\[ \omega(2) = x(5) \]
\[ \omega(3) = x(6) \]

call dcm321(\phi,amat) ! computes transformation (DCM) from inertial ! to body coordinates
call tran3(amat,amat_t) ! transformation from body to inertial coordinates
do 1 i=1,ndp ! extracts scalar components of the input vector signal from the
  func_x(i) = uu2(1,i) ! common block "subsys2". The variable uu2(1-3,j) contains the orbit
  func_y(i) = uu2(2,i) ! radius vector inertial coordinates.
  func_z(i) = uu2(3,i)
1 continue
tl=0.0D0 ! time (tau) is measured from the left end point of the convergence interval
call interp(ndp,tl,delt,tau,func_x,val) ! interpolates x-component of the radius vector
  rad(1)=val
call interp(ndp,tl,delt,tau,func_y,val) !... y-component ...
  rad(2)=val
call interp(ndp,tl,delt,tau,func_z,val) !... z-component ...
rad(3)=val

call vecnorm(rad,rad_norm,rmag)  ! normalized radius vector in inertial coordinates

c  *** Gravity Gradient Torques ***
c
call matvec amat,rad_norm,rad_norm_b)  ! normalized radius vector in body coordinates
call matvec xmol,rad_norm_b,xir)  ! [I](r), unit vector, body coordinates
call vcross(rad_norm_b,xir,rxir)  ! (r)X[I](r)

c  if(rmag.lt.1.0e-6) ggmul=0.0
if(rmag.ge.1.0e-6) ggmul=3.0DO0*gm/rmag**3

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

c  *** Aerodynamic torque calculation at time=t1+tau

do 2 i=1,ndp  ! extracts scalar components of the input vector signal from the
  func_x(i)=uu2(4,i)  ! common block "subsyst2". The variable uu2(4-6,j) contains the aerodynamic
  func_y(i)=uu2(5,i)  ! torque in body (c.m. origin) coordinates.
  func_z(i)=uu2(6,i)
  2 continue

ct1=0.0D0  ! time (tau) is measured from the left end point of the convergence interval

call interp(ndp,t1,delt,tau,func_x,val)  ! interpolates x-component of aerodynamic torque
t_aero(1)=val
call interp(ndp,t1,delt,tau,func_y,val)  ! ... y-component...
t_aero(2)=val
call interp(ndp,t1,delt,tau,func_z,val)  ! ... z-component...
t_aero(3)=val

c  *** Control System Torques ***
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))  

*** Attitude dynamics  

   call matvec(xmoi,omega,h)  
   call vcross(omega,h,wxh)  
   t_total(1) = t_GG(1) + t_aero(1) + t_control(1) - wxh(1)  
   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)  
   call matvec(xmoi_inv,t_total,omegadot)  

   ! angular acceleration of body frame in body  
   ! coordinates  

Converting body rates to Euler angle derivatives  

   ! (Rotation sequence precludes 90deg angle about y-axis)  

   sin_1 = d*sin(phi(1))  
   cos_1 = d*cos(phi(1))  
   sin_2 = d*sin(phi(2))  
   cos_2 = d*cos(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  

*** Changing back to mathematical notation from physical notation.  

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

C
return
end

SUBROUTINE TRAN3(A,AT)
C
*** THIS ROUTINE TAKES THE TRANSPOSE OF A 3X3 MATRIX.
C
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
DIMENSION A(3,3),AT(3,3)
C
DO 10 I=1,3
DO 10 J=1,3
10 AT(I,J)=A(J,I)
RETURN
END
SUBROUTINE VCROSS(A,B,C)
C
*** THIS ROUTINE COMPUTES THE CROSS PRODUCT OF TWO VECTORS
A AND B AND STORES THE RESULT AS C. (C=AXB)
C
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
DIMENSION A(3),B(3),C(3)
C
C(1)=A(2)*B(3)-B(2)*A(3)
C(2)=B(1)*A(3)-A(1)*B(3)
C(3)=A(1)*B(2)-B(1)*A(2)
RETURN
END

SUBROUTINE VDOTPRO(A,B,ADOTB)
C
*** computes dot product of two vectors
C
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
DIMENSION A(3),B(3)
C
ADOTB=A(1)*B(1)+A(2)*B(2)+A(3)*B(3)
C
RETURN
SUBROUTINE VECNORM(X,XNORM,XMAG)
C
C *** THIS ROUTINE NORMALIZES A VECTOR (3X1).
C
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
DIMENSION X(3),XNORM(3)
C
XMAG=DSQRT(X(1)**2+X(2)**2+X(3)**2)
IF(XMAG.LE.0.0D0)GO TO 11
DO 10 I=1,3
XNORM(I)=X(I)/XMAG
10 CONTINUE
C
RETURN
C
11 XMAG=0.0D0
C
DO 12 I=1,3
XNORM(I)=0.0D0
12 CONTINUE
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)
This is FORTRAN MAIN program "rfd.f", and is a variation of FORTRAN main program "rfc.f".
Development began 6/12/2000 by Doug Nixon, MSFC/TD55, in support of AEI. The main and subprogram
listings are contained in "aei/rfd"

This example of simulation by recursive feedback uses distributed coupling, and convergence is
checked on the minimum number of signals (aerodynamic forces and torques).

The purpose is to generate the dynamic solution of three coupled subsystems (orbit dynamics, attitude dynamics, and aerodynamics) by recursive feedback, which involves distributed and possibly multiple methods of numerical integration. The intent is to compare results generated by a standard approach using a centralized single numerical integration method. The standard approach has been implemented as FORTRAN MAIN program "rku.f", and the main and subprogram listings are contained in "aei/rku".

*** Declarations

```fortran
implicit double precision(a-h,o-z)
dimension uu1(6,100),yy1(6,100)
dimension uu2(6,100),yy2(6,100)
dimension uu3(6,100),yy3(6,100)
dimension fb(6,100),fbp(6,100)
dimension yy1i(6),yy2i(6),yy3i(6)
dimension yy1f(6),yy2f(6),yy3f(6)
common/system/uu,yy,yy
common/subsys1/uu1,yy1
common/endcond1/yy1i,yy1f
common/subsys2/uu2,yy2
common/endcond2/yy2i,yy2f
common/subsys3/uu3,yy3
common/endcond3/yy3i,yy3f
```

!input and output signals for three subsystems,
six signals per subsystem, 100 time points
!for convergence check
!initial conditions, 3 subsystems, 6 outputs
!final conditions, 3 subsystems, 6 outputs
!system input/output/"previous" output, all vs. time
!orbit dynamics, i/o vs. time
!initial and final output conditions
!attitude dynamics, i/o vs. time
!initial and final output conditions
!aerodynamics, i/o vs. time
!initial and final output conditions
**Definition of the fundamental variables**

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

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

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

**yy1(i,j)**——First subsystem output signal. Represents the ith component at the jth time point for the vector output signal from the first (orbit dynamics) subsystem, and is also the first (top) partition of yy(i,j).

**yy2(i,j)**——Second subsystem output signal. Represents the ith component at the jth time point for the vector output signal from the second (attitude dynamics) subsystem, and is also the second (middle) partition of yy(i,j).

**yy3(i,j)**——Third subsystem output signal. Represents the ith component at the jth time point for the vector output signal to the third (aerodynamics) subsystem, and is also the third (bottom) partition of yy(i,j).
yy1(i) ------ Value of the output signal at the beginning of the current convergence interval for the FIRST (orbit dynamics) subsystem. Represents the ith component of the signal at the first time point, and is equivalent to yy1(i,1)

yy2(i) ------ Value of the output signal at the beginning of the current convergence interval for the SECOND (attitude dynamics) subsystem. Represents the ith component of the signal at the first time point, and is equivalent to yy2(i,1)

yy3(i) ------ Value of the output signal at the beginning of the current convergence interval for the THIRD (aerodynamics) subsystem. Represents the ith component of the signal at the first time point, and is equivalent to yy3(i,1)

yy1f(i) ------ Value of the output signal at the end of the current convergence interval for the FIRST (orbit dynamics) subsystem. Represents the ith component of the signal at the last time point, and is equivalent to yy1(i,ndp)

yy2f(i) ------ Value of the output signal at the end of the current convergence interval for the SECOND (attitude dynamics) subsystem. Represents the ith component of the signal at the last time point, and is equivalent to yy2(i,ndp)

yy3f(i) ------ Value of the output signal at the end of the current convergence interval for the THIRD (aerodynamics) subsystem. Represents the ith component of the vector signal yy3(i,j) at the last time point, and is equivalent to yy3(i,ndp)

*****General Comments ****************************

The above nomenclature has been designed to allow implementation of recursive feedback via FORTRAN in such a way that independence of the subsystem simulations can be supported. Definition of submatrices (partitioning) facilitates independence of the subsystem simulations while providing appropriate structure for implementation of the system-level algorithm. The fact that subsystem data has been placed in COMMON with the MAIN (system-level) program does not necessarily mean that all of that data is required or desired at the system level -- it may mean only that the SUBROUTINE containing the subsystem simulation needs to have access to data that it generated previously, and placing that data in COMMON is a way to provide that in FORTRAN.
Because of the distributed coupling configuration, information flows from one sub-simulation to the next in a sequential manner. The order of execution is important, and parallel execution is not possible.

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

*** Establishing the output file

```
open(unit=1,file='rfd.out1',status='unknown')
```

*** Defining system-level simulation parameters

```
fac=1.0/float(ndp-1)  !ratio of integration time step to convergence interval duration
```

print*, 'This is main program rfd.f. Solution'
print*, 'method is recursive feedback with'
print*, 'distributed coupling.'
print*, 'Input the number of data (time) points'
print*, 'per convergence interval, including'
print*, 'both end (boundary) points:'
print*, ndp
print*, 'Input the convergence interval duration'
print*, ' (sec.):'
print*, dtmaj
print*
```
delt=fac*dtmaj                    !integration stepsize

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

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

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

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

    tol=(1.000)*1.0e-14            !relative
    res=(1.000)*1.0e-14            !absolute
    itmax=100                      !maximum allowable number of iterations
    ptol=1.0e-3                    !tolerance for print time
print*, ' Run in progress.'
print *
call clock(n1) !machine start time

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

*** Setting up initial conditions for each sub-simulation at system simulation outset

call orbit(2) !mode 2, original initial conditions setup, orbit dynamics

call attitude(2) !mode 2, original initial conditions setup, attitude dynamics

!no initial conditions setup for aerodynamics (algebraic system)

100 continue !***************COMPUTING INITIAL RECURSION STAGE DATA*******************************

*** Initializing the system-level feedback signal to zero: **************************************

do 1 i=1,6 !six scalar input signals per subsystem
do 1 j=1,ndp !ndp number of data (time) points per scalar signal
uu1(i,j)=0.0D0 !initializing all subsystem input signals to zero (0)
uu2(i,j)=0.0D0
uu3(i,j)=0.0D0
1 continue

*** Running individual subsystem simulations over the convergence interval

call orbit(3) ! mode 3, run orbit dynamics simulation over convergence interval
call coupler1

call attitude(3)  ! mode 3, run attitude dynamics simulation over convergence interval
call coupler2
call aerodyn(3)   ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
call coupler3    ! computes feedback signal from aerodynamics to orbit and attitude dynamics
do 2 i=1,3
   do 2 j=1,ndp
      fbp(i,j)=uu1(i,j)  ! aerodynamic force, inertial coordinates
      fbp(i+3,j)=uu2(i+3,j) ! aerodynamic torque, body coordinates
   2 continue

200 continue ! RECURSION LOOP ENTRY POINT

nit=nit+1    ! updating the iterations count (number of recursion stages)

*** Running individual subsystem simulations over the convergence interval

   call orbit(3) ! mode 3, run orbit dynamics simulation over convergence interval
   call coupler1
   call attitude(3) ! mode 3, run attitude dynamics simulation over convergence interval
call coupler2
   call aerodyn(3) ! mode 3, run aerodynamics (algebraic) simulation over convergence interval
call coupler3

*** Check for convergence

   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 system convergence check (feedback)  
  ! iconflag=0 if converged, 1 if  
  ! not converged  
  c  
  idirect=iconflag+1  
  500  format(/,2i5,/)  
  go to (6,5) idirect  
  ! 6 for converged, 5 for not converged  
  c  
  5  continue  ! ******************* Not CONVERGED, ADVANCE RECURSION STAGE  
  ***********************  
  c  
  do  7  i=1,6  
    do  7  j=1,ndp  
      fbp(i,j)=fb(i,j)  
    7  continue  
    c  
    if(nit.ge.itmax) go to 300  
    ! protection against persistent non convergence  
    go to 200  
    ! returns to "top" of the recursion loop to run subsystem simulations again  
  c  
  6  continue  ! ** CONVERGED, ADVANCE TO NEXT CONVERGENCE INTERVAL (or stop if simulation is complete)  
  **  
  c  
  call  rfd_out  
  ! appends system-level output signals to output file (rfd.out)  
  nmt=nmt+1  
  ! increments number of major time (convergence) intervals completed  
  time=nmt*dtma  
  ! 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

In a more general setting, this would be done at the subsystem level (mode 4) so that the system level is not required to have knowledge of all subsystem states.

do 8 i=1,6
yy1f(i)=yy1(i,ndp)
yy2f(i)=yy2(i,ndp)
yy3f(i)=yy3(i,ndp)
8 continue

!extracts subsystem final conditions for the current convergence interval
!from the final time point of the system response associated with the current
!convergence interval. SEE PARAGRAPH NUMBER 3 of "General Comments" ABOVE.

do 9 i=1,6
yy1i(i)=yy1f(i)
yy2i(i)=yy2f(i)
yy3i(i)=yy3f(i)
9 continue

goto 100

!sets initial conditions (for all states) for the next convergence interval
!equal to conditions reached at the end (final conditions) of current
!convergence interval.

!run subsystem simulations for the first time across the new (next)
!convergence interval.

print*
if(nit.ge.itmax) print*, 'Max. allowable number of '
if(nit.ge.itmax) print*, ' iterations reached.'
call clock(n2) !machine stop time
irt=n2-n1 !run-time

if(nmt.eq.nmtr) print*, ' Normal exit, run-time = ',irun, ' sec.'
print*
close(unit=1,status='keep')
c
stop

*******************************************************************************
end !*******************************************************************************

*******************************************************************************
END OF MAIN PROGRAM **************************************************************************
subroutine coupler1

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

dimension uu1(6,100),yy1(6,100)
dimension uu2(6,100),yy2(6,100)

common/subsys1/uu1,yy1
common/subsys2/uu2,yy2

common/simulation/ndp,delt,tol,res,t_start,
lt_report,t_print,t_stop,nmt,nmtr,nit,
2t_initial,t_final

!all except some single-time-point utility routines

*** from orbit dynamics to attitude dynamics:

uu2(1,j)=yy1(1,j)       !Extracting orbit radius from the orbit dynamics output signal
uu2(2,j)=yy1(2,j)
uu2(3,j)=yy1(3,j)

10 continue
return
deend

subroutine coupler2

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

dimension uu1(6,100),yy1(6,100)
dimension uu2(6,100),yy2(6,100)
dimension uu3(6,100),yy3(6,100)

dimension rad(3),vel(3)
dimension rad_b(3),vel_b(3)
dimension phi(3)
dimension amat(3,3)
dimension amat_t(3,3)

!orbit radius and velocity vectors, inertial coordinates
!orbit radius and velocity vectors, body coordinates
!Euler angles, inertial frame to body frame, 3-2-1 sequence
!direction cosine matrix (d.c.m.), inertial to body
!transpose of amat(i,j)

!all except some single-time-point utility routines

common/simulation/ndp,delt,tol,tes,t_sart,
1t_report,t_print,t_stop,nmt,nmtr,nit,
2t_initial,t_final

common/subsys1/uu1,yy1
common/subsys2/uu2,yy2
common/subsys3/uu3,yy3

do 10 j=1,ndp

****************** Coupling from orbit and attitude dynamics to aerodynamics ******************

****************** Coupling from orbit and attitude dynamics to aerodynamics ******************
c
rad(1)=yy1(1,j) !extracting orbit radius from subsystem 1 output
rad(2)=yy1(2,j)
rad(3)=yy1(3,j)
c
phi(1)=yy2(1,j) !extracting attitude (Euler) angles from subsystem 2 output signal
phi(2)=yy2(2,j)
phi(3)=yy2(3,j)
c
call dcm32l(phi amat) !computes d.c.m that transforms vectors from inertial to body 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
tu3(1,j)=rad_b(1) !feedback signal is orbit radius vector in body coordinates
tu3(2,j)=rad_b(2)
tu3(3,j)=rad_b(3)
c
vel(1)=yy1(4,j) !variable vel(i) is used for listing readability only
vel(2)=yy1(5,j)
vel(3)=yy1(6,j)
c
call matvec(amat vel vel_b) !transforms orbit velocity vector to body coordinates

tu3(4,j)=vel_b(1) !feedback signal is orbit velocity vector in body coordinates
tu3(5,j)=vel_b(2)
tu3(6,j)=vel_b(3)
c
10 continue
c
return
c
end
subroutine coupler3

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

c dimension uu1(6,100), yy1(6,100)
dimension uu2(6,100), yy2(6,100)
dimension uu3(6,100), yy3(6,100)

c dimension f_aero(3) ! aerodynamic force (drag), inertial coordinates
dimension f_aero_b(3), t_aero_b(3) ! aerodynamic force and torque, body frame
dimension f_aero_g(3), t_aero_g(3) ! aerodynamic force and torque, geometric frame
dimension rxf(3) ! aerodynamic force transfer couple (from geometric origin to c.m.)
dimension phi(3) ! Euler angles, inertial frame to body frame, 3-2-1 sequence
dimension amat(3,3) ! direction cosine matrix (d.c.m.), inertial to body
dimension amat_t(3,3) ! transpose of amat(i,j)
dimension xmoi(3,3), xmoi_inv(3,3) ! inertia matrix, inverse inertia matrix
dimension rcm(3) ! center of mass location, geometric coordinates

c common/simulation/ndp, delt, tol, res, t_start, 
 1t_report, t_print, t_stop, nmt, nmtr, nit, 
 2t_initial, t_final " all except some single-time-point utility routines

c common/mass/xmass, xmoi, xmoi_inv, rcm " orbit.f, subl.f, attitude.f, sub2.f, coupler.f

c common/subsys1/uu1, yy1
common/subsys2/uu2, yy2
common/subsys3/uu3, yy3

c************************************************************************************
c************************************************************************************
c coupling from aerodynamics to orbit and attitude dynamics ************************************************************************************
c************************************************************************************
do 10 j=1,ndp

c *** from aerodynamics to orbit dynamics:

c ! extracting aerodynamic (drag) force from system output signal
f_aero_b(1)=yy3(1,j)
f_aero_b(2)=yy3(2,j)
f_aero_b(3)=yy3(3,j)
! computed to act at origin of geometric body frame,
! but can be assumed to act at c.m. for this purpose

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

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

call matvec(amat_t,f_aero_b,f_aero)
! transforms force vector from body to inertial coordinates

uu1(1,j)=f_aero(1)
uu1(2,j)=f_aero(2)
uu1(3,j)=f_aero(3)
! feedback signal is aerodynamic force (drag) in inertial coordinates,
! and is assumed to act at the satellite center of mass (c.m.)

uu1(4,j)=0.0D0
uu1(5,j)=0.0D0
uu1(6,j)=0.0D0
! slot holders, nothing yet identified

c *** from aerodynamics to attitude dynamics:

c ! aerodynamic (drag) force; acts at origin of geometric body frame.
f_aero_g(1)=yy3(1,j)
f_aero_g(2)=yy3(2,j)
f_aero_g(3)=yy3(3,j)
! f_aero_g(i) and f_aero_b(i) are numerically identical, but their
! physical interpretations (points of action) may be different

t_aero_g(1)=yy3(4,j)
t_aero_g(2)=yy3(5,j)
t_aero_g(3)=yy3(6,j)
! aerodynamic torque with force at origin of geometric body frame

call vcross(rcm,f_aero_g,rxf)
! torque of drag force about c.m.
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)+rf(2)  ! equivalent couple.
t_aero_b(3)=t_aero_g(3)+rf(3)

uu2(4,j)=t_aero_b(1)
uu2(5,j)=t_aero_b(2)
uu2(6,j)=t_aero_b(3)

10 continue
return
end

subroutine fastverge(ndim,yy3,yy3p,iconflag)  ! system-level convergence check
implicit double precision(a-h,o-z)
dimension yy3(6,100),yy3p(6,100)
dimension ynew(100),yold(100)
common/simulation/ndp,delt,tol,res,t_start,t_report,t_print,
lt_stop,nmt,nmtr,nit,t_initial,t_final

iconflag=0  ! initializes system-level flag to "converged"
do 20 i=1,ndim  ! loads vector elements for scalar signal convergence check
  do 19 j=1,ndp
    ! code goes here
  end do
20 continue
ynew(j)=yy3(i,j)
yold(j)=yy3p(i,j)
19 continue

call convrg(ynew,yold,ndp,tol,res,iflag) ! iflag=1 for non-convergence.
iconflag=iconflag+iflag ! computes status of system-level convergence
20 continue

if(iconeflag.gt.0)iconeflag=1 ! routine returns 0 or 1 for iconflag

*******************************************************************************
****************************************************************************** end check loop ************
******************************************************************************

return

end

*******************************************************************************
******************************************************************************
******************************************************************************
subroutine rfd_out

implicit double precision(a-h,o-z)
dimension uul(6,100), yyl(6,100)
dimension uu2(6,100), yy2(6,100)
dimension uu3(6,100), yy3(6,100)

dimension zz(18,100) !single output array at system level
dimension rad(3), rad_n(3) !for processing orbit data to obtain altitude
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
dimension phi(3) !Euler angles, added 10/2/00. needed here to limit values for output

common/subsys1/uul, yyl
common/subsys2/uu2, yy2
common/subsys3/uu3, yy3

common/simulation/ndp, delt, tol, res, t_start, !all except some single-time-point utility routines
  t_report, t_print, t_stop, nmt, nmtr, nit,
  2t_initial, t_final

common/simulation2/del_report, del_print, ptol !percent done report interval, print interval

common/constants/rearth, gme, pi, cvdr

common/mass/xmass, xmoi, xmoi_inv, rcm

  do 3 i=1,6
  do 3 j=1,ndp
  zz(i,j)=yyl(i,j)
  zz(i+6,j)=yy2(i,j)
  zz(i+12,j)=yy3(i,j)
  3 continue

  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

4 continue

do 5 j=1,ndp
f_aero(1)=zz(13,j)
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)
z(17,j)=t_aero(2)
z(18,j)=t_aero(3)
5 continue

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

do 1000 i=1,ndp-1
! begin first write loop

do 1 k=1,3
rad(k)=yy1(k,i)
1 continue
call vecnorm(rad,rad_n,rmag)
alt=rmag/1000.000-rearth !altitude, kilometers

if(time.ge.t_print) write(1,100) time,(zz(j,i)),j=1,18),alt,nlt
if(time.ge.t_print) t_print=t_print+del_print
if(time.ge.t_report) ipd=int(((time/t_stop)*100.0)
if(time.ge.t_report) print*,'Percent complete =',
1 ipd
if(time.ge.t_report) t_report=t_report+del_report
100 format (20E20.10, i5)
1000 continue
end
B.3 FORTRAN Listings, Including Main Program and All Subroutines Not Previously Listed in A.1, Unified Approach, Runge-Kutta Method (RK–U)
This is main program "rku.f". Construction began 5/9/2000 by Doug Nixon, TD55, MSFC.

It is designed to produce a standard-method benchmark to support an evaluation of recursive feedback as a multi-discipline time-domain simulation solution method.

The test-case is a system involving orbital mechanics, vehicle rotational dynamics and aerodynamics. "rku.f" is the main driver for the twelve-state dynamic system which is integrated by "rksys.f", a fourth-order Runge-Kutta subroutine. "sub.f" defines the dynamics in first-order state-variable form. Listings are contained in "aei/rku.f".

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

dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)
dimension r(3),v(3),phideg(3),phi(3),omegadeg(3),omega(3)
dimension x(12),xdot(12)

dimension cpfront(3),cpright(3),cpbottom(3)  !constant aerodynamic system parameters
dimension cprear(3),cpleft(3),cptop(3)
dimension afront(3),aright(3),abottom(3)
dimension arear(3),aleft(3),atop(3)

common/constants/qme
common/thetasave/thetal,add
common/convert/cvdr
common/mass/xmass,xmoi,xmoi_inv,rcm
common/gains/a0,a1,b0,b1,c0,c1
common/altitude/rmag,rearth

common/aeroconst/ cpfront,cpright,cpbottom,cprear,cpleft, !in common with aero-comp
cptop,afront,aright,abottom,arear,aleft,atop,cd,rho

print* print*', 'This main program rku.f. It uses a'
print* print*', 'conventional centralized integration'
print* print*', 'method. (Runge-Kutta).'
print* print*', 'Input the integration stepsize,'
print* print* (sec.):'
read* delta
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

*** Numerically Defining Constants: ***

gravconst=(6.673D0)*(1.0e-11) ! Constant of Gravitation,
! m**3/kg-sec**2, CRC, page 6

rearth=6367.65D0 ! Earth's radius, (polar+equatorial)/2.0,
! km, CRC, page 6

dearth=5522.0D0 ! Average density of the earth,
! kg/m**3, CRC, page 6
pi=datan2(0.0D0,-1.0D0) ! Computes pi to machine precision
cvdr=pi/180.0D0 ! Converts degrees to radians

volume=(4.0D0/3.0D0)*pi*(rearth*1000.0D0)**3 ! Volume of earth, m**3

xmearth=dearth*volume ! Mass of the earth, kg

gme=gravconst*xmearth ! Gravity Constant times mass of the earth.

*** Numerically Defining System Data: ***

alt=300.0D0 ! altitude, kilometers

xmass=10000.0D0 ! spacecraft mass, kilograms

xmoi(1,1)=11250.0D0 ! spacecraft 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)

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
  c0=1000.0D0*control  ! z-axis
  c1=100.0D0*control

*** Setting up fixed aerodynamic data, in common/aeroconst/ with rksys/sub/aerocomp

xl=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=xl*h  ! side area, normal to y-axis (+ "right side")
az=xl*w  ! top or bottom area, normal to (+ z-axis "bottom")

Work with six facets ----

do 9 i=1,3  ! setting all vector components to zero
dafront(i)=0.0D0  ! area vectors, normal to facet surfaces, through
darear(i)=0.0D0  ! centers, positive (point) outward from center of
daright(i)=0.0D0  ! box, back sides not considered (no contribution)
daleft(i)=0.0D0
datop(i)=0.0D0
abottom(i)=0.0D0
c cpfront(i)=0.0D0 ! centers of pressure location vectors, from geometric
cprear(i)=0.0D0 ! center of box to center of each facet
cpright(i)=0.0D0
cpleft(i)=0.0D0
cptop(i)=0.0D0
cpbottom(i)=0.0D0
c
9 continue

c afront(1)=ax ! setting non-zero 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
c cd=2.25 ! drag coefficient, same for all facets,
! not a function of direction
c
c rho=1.95E-11 ! kg/m**3, Wertz,inside back cover, 300km altitude , mean value
c
c *** Setting Initial Conditions, assuming a circular orbit at a given altitude: ***
c
c rmag=(rearth+alt)*1000.0D0 ! Radius magnitude, meters
r(1)=rmag*dcos(30.0D0*cvdr)
r(2)=0.0D0 ! Assumes a 30 deg "beta angle", (between ecliptic
r(3)=rmag*dsin(30.0D0*cvdr) ! and orbit planes)
c

v_mag = dsqrt(gme/rmag)
! Velocity magnitude for a circular orbit, meters/sec

cube = rmag**3
w_sq = gme/cube

period = 2.0d0*pi/dsqrt(w_sq)
! Period for circular orbit

c
v(1) = 0.0
v(2) = v_mag
v(3) = 0.0

phideg(1) = 0.0
phideg(2) = 0.0
phideg(3) = 0.0

omegadeg(1) = 0.0
omegadeg(2) = 0.0
omegadeg(3) = 0.0

*** Changing Angular variables from degrees to radians: ***

phi(1) = phideg(1)*cvdr
phi(2) = phideg(2)*cvdr
phi(3) = phideg(3)*cvdr

omega(1) = omegadeg(1)*cvdr
omega(2) = omegadeg(2)*cvdr
omega(3) = omegadeg(3)*cvdr

*** Setting up the initial state vector (12 states): ***

do 10 i = 1, 3
x(i) = r(i)
! states 1-3, orbit radius, meters
x(i+3) = v(i)
! states 4-6, orbit velocity, meters/sec
x(i+6) = phi(i)
! states 7-9, body angles (attitude errors), radians
x(i+9) = omegadeg(i)
! states 10-12, body rates, radians/sec
10 continue

*** Initializing time, opening data files, etc.***
num=12
! number of system states
time=0.0D0
! initial time (problem start time), sec
call sub(xdot,x,time)
! computes dependent variables for initial output
tprint=time
! initializing time to print output data set
ptol=.001*delt
! tolerance on time for print
theta=datan2(r(2),r(1))
! initial value of true anomaly, radians
theta=theta/cvdr
! convert to degrees
add=0.0D0
! initial value for multiple revolution correction
if(theta.lt.0.0D0) theta=theta+360.0D0 ! full circle before reset
open(unit=1, file='rkoutl', status='unknown')
! opens data output file
20 continue
! begin integration loop
**************
if(abs(time-tprint).lt.ptol) call rku_out(time,x,delprint,tprint)
! writes to file, updates tprint
call rksys(time,x,num,delt)
! propagates states, increments time
if(time.lt.tstop) go to 20
! return to top of integration loop, or stop
continue
! end integration loop
**************
call rku_out(time,x,delprint,tprint)
! writes final data set to file
close(unit=1, status='keep')
! closes data output file
call clock(n2)
! machine stop time
i runtime=n2-n1
print*
print*,
' Normal exit, run-time = ',i runtime,' sec.'
stopt
end
subroutine aerocomp(vel, amat, force, torque)

Developed starting 5/26/2000 by Doug Nixon, TD55, MSFC to compute
aerodynamic forces and torques for the "truth model" for recursive
feedback simulation method comparison for AEI study.

rho = Atmospheric density, kg/m**3
vel(i) = Velocity vector in inertial coordinates, meters/sec
amat(i,j) = Direction cosine matrix, transforms vectors from inertial to
body coordinates
force(i) = Aerodynamic force, body coordinates, newtons
force(i) = Aerodynamic torque, body coordinates, newton-meters

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

dimension vel(3), amat(3,3), force(3), torque(3)
dimension vnorm(3), vnorm_b(3)

dimension cpfront(3), cpwright(3), cpbottom(3) !constant system parameters computed or defined
dimension cprear(3), cpleft(3), cptop(3)
dimension afront(3), aright(3), abottom(3)
dimension arear(3), aleft(3), atop(3)

common/aeroconst/ cpfront, cpwright, cpbottom, cprear, cpleft, !in common with standard
lptop, afront, aright, abottom, arear, aleft, atop, cd, rho
dimension aproj(6), drag(6), facefor(6,3), facetor(6,3) !internally computed variables
dimension tforce(3), ttorg(3) !output variables

*** Aerodynamic forces ***
call vecnorm(vel,vnorm,vmag) ! normalize velocity vector in inertial
c

call matvec amat,vnorm,vnorm_b ! normalized velocity vector in body coordinates

*** computing magnitude drag for each of six facets

! computing dot product of each area vector with normalized velocity (projected areas)
! these statements define the facet numbering system

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

c  fac=(0.5D0)*rho*(vmag*2)*cd ! scaling by dynamic pressure times drag coefficient

do 21 i=1,6
drag(i)=fac*proj(i)
21 continue

do 22 i=1,6
do 22 j=1,3
  facfor(i,j)=-drag(i)*vnorm_b(j) ! this is jth component of force on the ith facet, expressed in body coordinates
22 continue

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

*** Aerodynamic Torques

! front facet contribution

do 30 i=1,3
tforce(i)=facefor(1,i)
30 continue
   call vcross(cpfront,tforce,ttorq)
   do 31 i=1,3
      facetor(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
      facetor(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
      facetor(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
      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

! adding right facet contribution

! adding bottom facet contribution

! adding rear facet contribution

! adding left facet contribution
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
   do 42 i=1,3
      torque(i)=0.0D0
   do 42 j=1,6
      torque(i)=torque(i)+facetor(j,i) ! total aerodynamic torque, body
      ! coordinates, newton-meters
42 continue
   return
end

SUBROUTINE RKSYS(X,Y,N,H)
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), Y=YT+F4*H/6
C X=X+H,
C NEXT STEP
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
200 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 *** 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 *** 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 *** END OF THIRD PASS ***
C
X=X0+H
   CALL SUB(FDERIV,Y,X)
   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
RETURN
END

subroutine rku_out(time,x,delprint,tprint)
c
C *** writes state data for main program "rku.f"
c
implicit double precision (a-h,o-z)
c
dimension x(12),xcv(12),rad(3),rad_norm(3),vel(3),vel_norm(3)
dimension f_aero(3),f_aero_b(3),f_aero_n(3)
dimension t_aero_b(3)
dimension f_grav(3)
dimension t_gg(3),t_cont(3)
dimension facefor(6,3),facefor(6,3)
dimension phi(3),amat(3,3)
c
c c c c c c c c c c c c c c c c c c c c c common/constants/gme
c c c c c c c c c c c c c c c c c c common/thetasave/thetal,add
common/convert/cvdr
common/altitude/rmag,realtrth
common/aeroforce/f_aero,f_aero_b  ! Added a fifth (ultimately redundant) call to sub in rksys.f
common/aerotorg/t_aero_b
!
so that this data is current at print time. (6/27/00)

*****************************************************************************
Post processing for a single time point
*****************************************************************************

** (supports standard centralized integration method) ********************************
*****************************************************************************

fgmag=dlsqrt(f_grav(1)**2+f_grav(2)**2+f_grav(3)**2) !gravity force magnitude, newtons

famag=dlsqrt(f_aero(1)**2+f_aero(2)**2+f_aero(3)**2) !aerodynamic force (drag)
!magnitude, newtons

alt=(rmag/1000.0D0-rearth) !altitude, kilometers

theta=datan2(x(2),x(1))
theta=theta/cv4dr
if(theta.lt.0.0) theta=theta+360.0

theta2=theta+add
if(theta2.ge.thetal) go to 1
    add=add+360.0D0
    theta2=theta2+360.0

1 continue

** limiting range of Euler angles for output: (added 9/20/2000)

do 3 i=1,3
  phi(i)=x(i+6)
3 continue

call dcm321(phi,amat)
call dcm2ang321(amat,phi)
do 2 i=1,3
    vcvc(i)=x(i)
    vcvc(i+3)=x(i+3)

! renaming the state variables, converting angular states to degree units

! radius (meters)
! velocity (meters/sec)
xcv(i+6)=phi(i)/cvdr  ! limited Euler angles (deg)
xcv(i+9)=x(i+9)/cvdr  ! body rates (deg/sec)
2 continue

*******************************************************************************
** End post processing**
*******************************************************************************

*******************************************************************************
** Write to file**
*******************************************************************************

**write(1,100) time,(xcv(i),i=1,12),
** 1(f_aero_b(i),i=1,3),(t_aero_b(i),i=1,3),
** 2 alt,theta,thonal

**100 format(22e20.10)
**  ! file name is standard.out1

** tprint=tprint+delprint
**  ! compute time for output of next data set

** thetal=theta2
**  ! update continuous true anomaly for future reference

return
** end

subroutine sub(xdot,x,time)
**
c23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
Construction began 4/28/2000, by Doug Nixon, TD55. The purpose is to supply the dynamics xdot=F(x,time) for integration by rksys, a fourth-order Runge-Kutta routine. The problem is to set up a standard-method simulation of a simple multidisciplined system involving orbital dynamics, vehicle attitude dynamics, and aerodynamics.

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

common/constants/gme
common/mass/xmass,xmoi,xmoi_inv,rcm
common/gains/a0,a1,b0,b1,c0,c1
common/tgg/t_gg
common/tcontrl/t_cont
common/gravforce/f_grav
common/aeroforce/f_aero,f_aero_b
common/aerotorg/t_aero_b
common/altitude/rmag,rearth

*** definition of variables in common:

gme : universal gravitational constant times mass of the earth (lb-ft'^2/slug)
xmass: spacecraft mass (slugs)
xmoi : spacecraft moment of inertia tensor (3x3 matrix, slug-ft'^2)

dimension xmoi(3,3),xmoi_inv(3,3),rcm(3)
dimension x(12),xdot(12)
dimension r(3),rnorm(3),rbody(3),rdot(3)
dimension v(3),vnorm(3),vbody(3),vdot(3)
dimension phi(3),phidot(3),omega(3),omegadot(3)
dimension f_grav(3)
dimension t_gg(3)
dimension f_aero(3),f_aero_b(3)
dimension t_aero_b(3)
dimension t_cont(3)
dimension t_tot(3)
dimension cmat(3,3)
dimension amat(3,3),amat_t(3,3)
dimension temp1(3), temp2(3)

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

** Orbit States (Vehicle Position States): **

\[
\begin{align*}
x(1) &= r(1): \text{x-component of orbit radius vector in inertial coordinates,} \\
& \quad \text{from origin of inertial system to c.m. of spacecraft (meters)} \\
x(2) &= r(2): \text{y-component} \\
x(3) &= r(3): \text{z-component} \\
x(4) &= v(1): \text{x-component of spacecraft velocity vector in inertial coordinates} \\
& \quad \text{(meters/second)} \\
x(5) &= v(2): \text{y-component} \\
x(6) &= v(3): \text{z-component}
\end{align*}
\]

** Vehicle Rotational States:**

\[
\begin{align*}
x(7) &= \phi(1): \text{Euler angle rotation from inertial to body system, about x-axis,} \\
& \quad \text{3rd in sequence. (radians)} \\
x(8) &= \phi(2): \text{About y-axis, 2nd in sequence.} \\
x(9) &= \phi(3): \text{About z-axis, 1st in sequence.} \\
x(10) &= \omega(1): \text{Body rate, x-axis (radians/second)} \\
x(11) &= \omega(2): \text{Body rate, x-axis} \\
x(12) &= \omega(3): \text{Body rate, x-axis}
\end{align*}
\]

** Aerodynamics States:**

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

------------------------------------------

\[
\begin{align*}
r(1) &= x(1): \text{Changing from mathematical to physical notation.} \\
r(2) &= x(2) \\
r(3) &= x(3)
\end{align*}
\]

183
\[ 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) \\
\[ \omega(3) = x(12) \]

\begin{verbatim}
c call dcm321(phi,amat)  ! computes transformation (DCM) from inertial 
                     ! to body coordinates

c call tran3 amat,amat_t  ! transformation from body to inertial coordinates

c *** Gravity Forces ***

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

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

c *** Gravity Gradient Torques ***

ggmul = 3.000 * gme / rmag ** 3

call matvec(amat,rnorm,rbbody)  ! normalized radius vector in body coordinates

call matvec(xmoi,rbbody,temp1)  ![ rbody] ! [ I][rbbody]

call vcross(rbody,temp1,temp2)  ![ rbody] X ![ I]

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

c *** Aerodynamic drag/torque calculations
\end{verbatim}
call aerocomp(v,amat,f_aero_b,t_aero_b) ! Inertial velocity and attitude are inputs
! Body forces and torques (geometric origin) ! are outputs

call vcross(rcm,f_aero_b,templ)
t_aero_b(1)=t_aero_b(1)+templ(1)
t_aero_b(2)=t_aero_b(2)+templ(2)
t_aero_b(3)=t_aero_b(3)+templ(3)

*** Control System Torques ***

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))

*** orbital dynamics (Orbit State Derivatives) ***
call matvec(amat_t,f_aero_b,f_aero) ! transform aerodynamic drag vector
! 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) !result in inertial coordinates
rdot(2)=v(2)
rdot(3)=v(3)

*** Attitude dynamics

call matvec(xmoi,omega,templ)
call vcross(omega,templ,templ2)
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 Converting body rates to Euler angle derivatives
(Rotation sequence precludes 90deg angle about y-axis)

```
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
```

*** Changing back to mathematical notation from physical notation.

```
xdot(1)=rdot(1)
xdot(2)=rdot(2)
xdot(3)=rdot(3)
xdot(4)=vdot(1)
xdot(5)=vdot(2)
xdot(6)=vdot(3)

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

return
end
REFERENCES


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

**Authors**: D.D. Nixon

**Performing Organization**: George C. Marshall Space Flight Center
Marshall Space Flight Center, AL 35812

**Sponsoring/Monitoring Agency**: National Aeronautics and Space Administration
Washington, DC 20546–0001

**Abstract**: 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 sub simulations 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.

**Subject Terms**: system simulation, coupled subsystems, multidiscipline, high fidelity, recursive, feedback, dynamic
National Aeronautics and Space Administration
AD33
George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama
35812