CR-184189

Mechanism Test Bed Flexible Body Model Report

Prepared for:

National Aeronautics and Space Administration
George C. Marshall Space Flight Center
Marshall Space Flight Center, AL 35812

Under Contract No.
NAS8-38771

by

Jimmy Compton

July 1991

LOGICON

Control Dynamics Company

600 BOULEVARD SOUTH SUITE 304 ■ HUNTSVILLE, AL 35802 ■ TELEPHONE: 205 882-2650

Mechanism Test Bed
Flexible Body Model
Report

Prepared for:

National Aeronautics and Space Administration
George C. Marshall Space Flight Center
Marshall Space Flight Center, AL 35812

Under Contract No.
NAS8-38771

by

Jimmy Compton

July 1991

LOGICON

Control Dynamics Company

600 BOULEVARD SOUTH SUITE 304 ■ HUNTSVILLE, AL 35802 ■ TELEPHONE: 205 882-2650

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ACRONYMS AND SYMBOLS

ASCII	-	American standard code for information interchange
B1	-	Body number 1 (target) center of mass/undeformed coordinate frame
B2	-	Body number 2 (chase) center of mass/undeformed coordinate frame
CDy	-	Control Dynamics Company
CM	-	Center of mass
DOF	-	Degree(s) of freedom
D1	-	Body number 1 (target) docking node/deformed coordinate frame
D2	-	Body number 2 (chase) docking node/deformed coordinate frame
EOM	-	Equations of motion
EHOMIT	-	Eliminate higher order modal integral terms
ERNLT	-	Eliminate remaining right hand side nonlinear terms
FEM	-	Finite element model
F/M	-	Force and moment
HWIL	-	Hardware in the loop
Hz	-	Hertz
K	-	Kilo
kg	-	Kilograms
LU	-	Lower-upper; refers to matrices in lower-upper triangular form
m	-	Meters
MTB	-	Mechanism Test Bed
NB	-	Body number

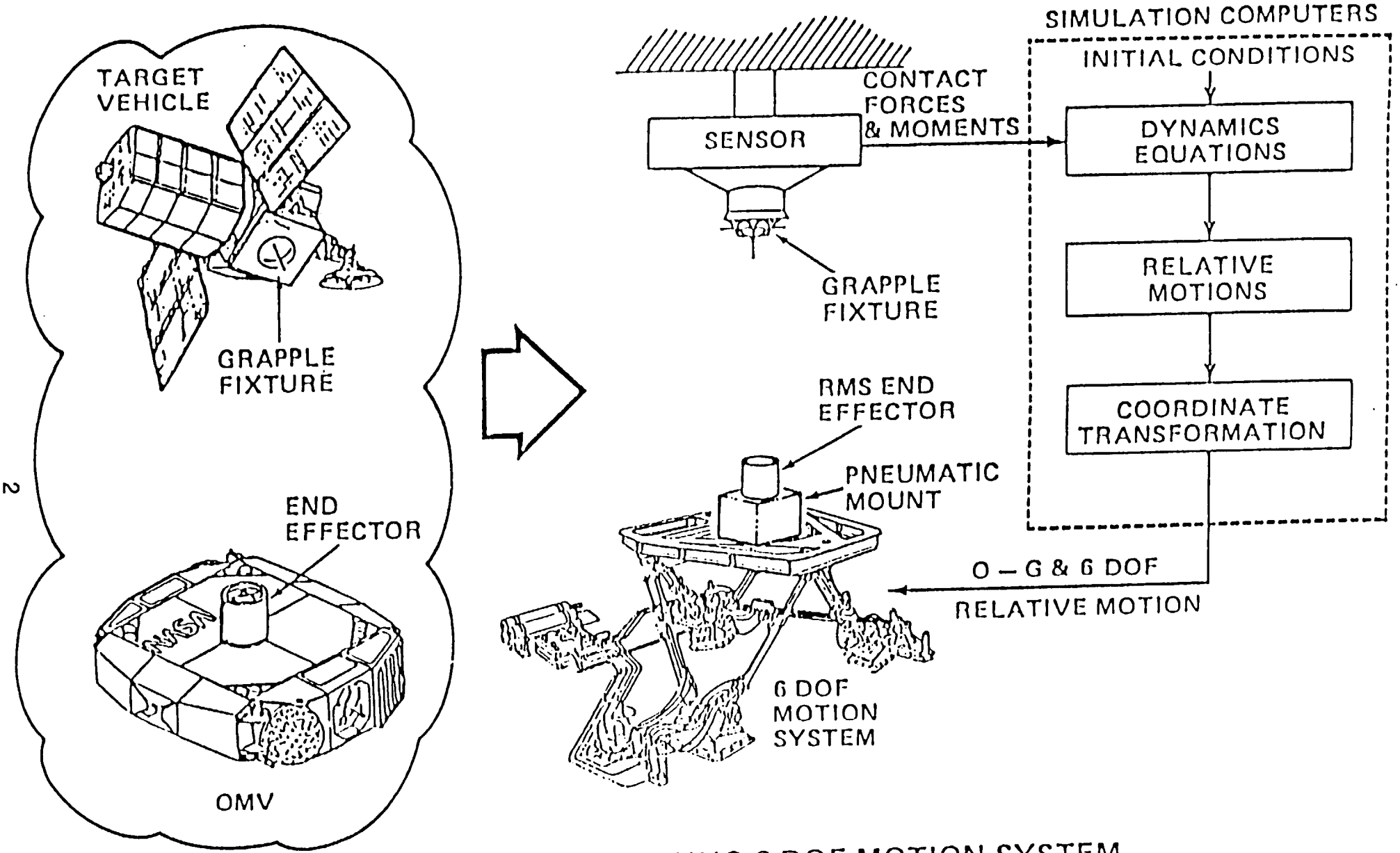
Nt	-	Newtons
PC-386	-	386 class personal computer
SMIS	-	Structures and matrix interpretive system
S1	-	Force and moment sensor location/coordinate frame on body number 1 (target)
wrt	-	With respect to

1.0 INTRODUCTION

The Space Station Mechanism Test Bed is a six degree of freedom (DOF) motion simulation facility used to evaluate docking and berthing hardware mechanisms. The major components of the Mechanism Test Bed (MTB) are shown in Figure 1.0-1. The chase vehicle docking mechanism is mounted on the hydraulically driven, computer controlled, six DOF motion system. The target vehicle docking mechanism is mounted in conjunction with a force and moment sensor to the facility ceiling. Mechanism contact forces and moments are measured and supplied to the host computer (i.e., currently, an Alliant computer) for use in the dynamics model.

Under contract NAS8-36570, Control Dynamics (CDy) developed a generalized rigid body math model to replace the "old" model which was based on several restrictive assumptions (e.g., one body was assumed to have much greater mass that the second and therefore was unaffected by contact forces and moments). The "new" model allowed the computation of vehicle relative motion in six DOF due to forces and moments from mechanism contact, attitude control systems, and gravity. No vehicle size limitations were imposed in the model. The equations of motions were based on Hill's equations for translational motion with respect to a nominal circular earth orbit and Newton-Euler equations for rotational motion. Over the past several years, CDy has worked with NASA in refining this rigid body model and the supporting software.

This report documents the development of a generalized flexible body math model to further enhance the MTB simulation capabilities. Although the original contract plan was to modify the current rigid body model, early investigations showed that a "fresh start" approach to the flex body model would yield a more efficient simulation. The development and major components of the flex body math model parallel those of the rigid body model.



CONTACT DYNAMICS USING 6 DOF MOTION SYSTEM

Figure 1.0-1

Section 2.0 of this report summarizes the rather mathematically intense derivation of the equations of motion for a single generic flexible body. The derivation is based on Lagrange's quasi-coordinate equations. Section 3.0 describes the method used to transform contact forces and moments from the sensor location to each body docking port. Section 4.0 discusses the computation of the relative body motion data: (1) relative orientation, (2) relative position, (3) relative translational velocity, and (4) relative angular velocity. This data is required in the interface between the dynamic math model and the main simulation. Section 5.0 describes the major components of the math model FORTRAN simulation including the required input data and a pre-processing algorithm for flexible body data. The model was coded with user selectable options regarding the method of integration and the complexity of the equations of motion. These options are discussed in Section 5.0 along with the governing input data. Section 6.0 discusses the many tests used to verify the flex body math model in progressive steps. Time domain comparisons to the multi-flex body code called TREETOPS are also presented. Section 7.0 contains a brief summary with concluding remarks. Appendices A and B contain a listing of the flexible body math model code and definitions for the math model global simulation variables, respectively. Finally, Appendix C contains a listing of the flexible body data pre-processing algorithm.

2.0 FLEXIBLE BODY MODEL

Table 2.0-1 lists the most popular methods, along with advantages and disadvantages, for deriving equations of motion (EOM) of dynamical systems. The D'Alembert / Newton-Euler equations are best suited for rigid body applications; for complex systems, the form of the equations depend to a large extent on the insight of the analyst. The other methods (i.e., Lagrange's Eqns. and Kane's Eqns.) offer a more systematic approach.

Lagrange's generalized coordinate equations are represented by the vector equation

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{Q} \qquad (2.0\text{-}1)$$

where L = kinetic energy - potential energy (Lagrangian)

$\mathbf{q}$ = vector of generalized coordinates

$\mathbf{Q}$ = vector of generalized forces.

This "standard" form of Lagrange's equations is well known and produces EOM with the same basic structure for all applications. A limitation of Lagrange's "standard" equations is that the generalized coordinate vector derivative ($\dot{\mathbf{q}}$) must be integrable to a set of coordinates ($\mathbf{q}$) which completely describe the system configuration. Specifically, this precludes the use of body angular rates (ω) in the $\dot{\mathbf{q}}$ vector (i.e., the integral of $\{\omega\, dt\}$ does not adequately describe body attitude). This limitation can be overcome by employing a modified form of Lagrange's equations known as Lagrange's quasi-coordinate equations or the Boltzmann-Hamel equations. Quasi-coordinates ($\mathbf{w}$) are quantities whose differentials may be written as linear combinations of generalized coordinate derivatives, e.g.,

Table 2.0-1 Candidate Methods for Equations of Motion Derivation.

METHOD	ADVANTAGES	DISADVANTAGES
1. D'Alembert / Newton - Euler Eqns.	Physically visualizable quantities Vector / dyadic representation	Max number of DOFs No standard equation form
2. Lagrange's Generalized Coordinate Eqns.	Systematic approach Invariant D.E. structure Equations facilitate integration	Differentiation of "compex" scalar energy functions. Must use "integrable" DOFs
3. Lagrange's Quasi-Coordinate Eqns. (Boltzmann-Hamel)	Subsumes method (2) Relieves constraint of using generalized coordinate derivatives: $T(\mathbf{q},\dot{\mathbf{q}},t) \rightarrow \overline{T}(\mathbf{w},\dot{\mathbf{q}},t)$	Differentiation of energy functions
4. Kane's Eqns. (TREETOPS)	Systematic approach Reduces system to minimal representation	Lack of physical insight Not as well known

$$\frac{d}{dt}(w_1) = a_1 \frac{d}{dt}(q_1) + a_2 \frac{d}{dt}(q_2) + \ldots \qquad (2.0\text{-}2)$$

Thus, the quasi-coordinate form of Lagrange's equations allows the direct use of body angular rates in the EOM coordinate vector which replaces the $\dot{q}$ coordinate vector (i.e., $(\dot{q}, q, t) \rightarrow (w, q, t)$). Lagrange's quasi-coordinate method was used to derive the EOM for the MTB generic flexible body model.

Kane's equations combines certain advantages from D'Alembert's principle and Lagrange's method. The multi-flex body code TREETOPS is based on Kane's equations. However, Kane's method is not as well known as the Lagrangian formulation.

2.1 Lagrange's Quasi-Coordinate Equations

The quasi-coordinate vector definition used in the flex body model is

$$\mathbf{w} = \begin{Bmatrix} \dot{\mathbf{P}}^B \\ \boldsymbol{\omega}^B \\ \dot{\eta} \end{Bmatrix} \qquad (2.1\text{-}1)$$

where $\dot{\mathbf{P}}^B$ = translational velocity of undeformed body CM with respect to (wrt) inertial space expressed in the body frame

$\boldsymbol{\omega}^B$ = angular velocity of undeformed body frame wrt inertial space expressed in the body frame

$\dot{\eta}$ = time derivative of the generalized modal coordinate vector.

The associated generalized coordinate vector was defined using the four Euler parameters (i.e., a quaternion) for attitude

$$q = \begin{Bmatrix} P^I \\ \varepsilon_{BI} \\ \eta \end{Bmatrix} \tag{2.1-2}$$

where P^I = position of undeformed body CM wrt inertial space expressed in the inertial frame

ε_{BI} = inertial to body quaternion

η = generalized modal coordinate vector.

Notice that the generalized coordinates are not independent due to the unity magnitude quaternion constraint

$$\left[\sum_{i=1}^{4} (\varepsilon_{BI})_i^2\right]^{1/2} = 1 \tag{2.1-3}$$

The transformation from the quasi-coordinate vector to the derivative of the generalized coordinate vector can be expressed as

$$\dot{q} = \begin{bmatrix} [B\ I]^T & 0 & 0 \\ 0 & \frac{1}{2}E^T & 0 \\ 0 & 0 & I \end{bmatrix} w \tag{2.1-4}$$

where $[B\ I]^T$ = 3 x 3 body to inertial transformation matrix

$(1/2)\ E^T$ = 4 x 3 transformation from the body angular velocity vector to the quaternion time derivative

$$E^T = \begin{bmatrix} \varepsilon_4 & -\varepsilon_3 & \varepsilon_2 \\ \varepsilon_3 & \varepsilon_4 & -\varepsilon_1 \\ -\varepsilon_2 & \varepsilon_1 & \varepsilon_4 \\ -\varepsilon_1 & -\varepsilon_2 & -\varepsilon_3 \end{bmatrix}$$

I = 3 x 3 identity matrix.

Starting with Lagrange's "standard" equations (2.0-1), Lagrange's quasi-coordinate equations can be derived for the coordinate vector definitions given in eqns. (2.1-1) through (2.1-4). The EOM separate nicely into vector equations for translation, rotation, and flexibility. The resulting equations can be expressed in general as

Translational EOM:

$$\frac{d}{dt}\left(\frac{\partial \bar{T}}{\partial \dot{\mathbf{P}}^B}\right) + \boldsymbol{\omega}^B \times \frac{\partial \bar{T}}{\partial \dot{\mathbf{P}}^B} - [BI]\frac{\partial \bar{T}}{\partial \mathbf{P}^I} + [BI]\frac{\partial V^*}{\partial \mathbf{P}^I} = [BI]\,\mathbf{Q}_{P^I} \qquad (2.1\text{-}5)$$

Rotational EOM:

$$\frac{d}{dt}\left(\frac{\partial \bar{T}}{\partial \boldsymbol{\omega}^B}\right) + \boldsymbol{\omega}^B \times \frac{\partial \bar{T}}{\partial \boldsymbol{\omega}^B} + \dot{\mathbf{P}}^B \times \frac{\partial \bar{T}}{\partial \dot{\mathbf{P}}^B} - \tfrac{1}{2}\mathbf{E}\frac{\partial \bar{T}}{\partial \boldsymbol{\varepsilon}} + \tfrac{1}{2}\mathbf{E}\frac{\partial V^*}{\partial \boldsymbol{\varepsilon}} = \tfrac{1}{2}\mathbf{E}\,\mathbf{Q}_\varepsilon \qquad (2.1\text{-}6)$$

Flexibility EOM:

$$\frac{d}{dt}\left(\frac{\partial \bar{T}}{\partial \dot{\eta}}\right) - \frac{\partial \bar{T}}{\partial \eta} + \frac{\partial V^*}{\partial \eta} = \mathbf{Q}_\eta \qquad (2.1\text{-}7)$$

where $\bar{T}$ = kinetic energy function in terms of quasi-coordinates

V^* = potential energy function augmented with constraint equations.

Notice that application of Lagrange's equations requires a derivation of both kinetic and potential energy functions (scalars) for the system under consideration.

Constraints are included with the potential energy function using the Lagrange multiplier technique to form an augmented potential energy function

$$V^* = V(q) - \lambda^T \phi(q) \qquad (2.1\text{-}8)$$

where λ is a vector of Lagrange multipliers and the constraints are of the form

$$\phi(q) = 0. \qquad (2.1\text{-}9)$$

For an unconstrained flexible body (as considered here), the only constraint equations will be those resulting from the use of dependent coordinates (e.g., quaternion for attitude).

2.2 Flexible Body Kinetic Energy Function

The kinetic energy function for a flexible body can be derived using a mass particle approach. Using the configuration definitions given in Fig. 2.2-1, the inertial position of the i^{th} mass particle of a generic flexible body may be written as

$$r_i = R_o + \rho_i + \delta_i. \qquad (2.2\text{-}1)$$

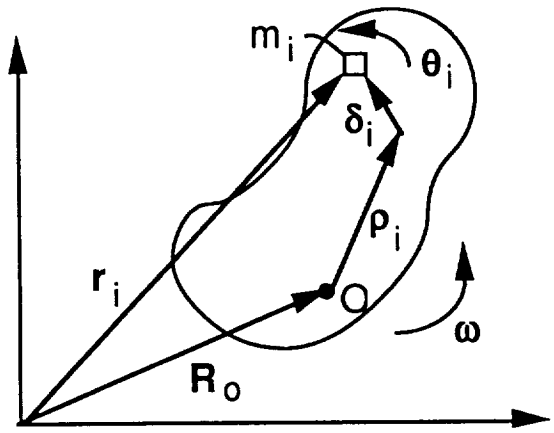
Taking the time derivative of the mass particle position vector with respect to the inertial frame leads to

$$\dot{r}_i = \dot{R}_o + \omega \times (\rho_i + \delta_i) + (\dot{\delta}_i)_B \qquad (2.2\text{-}2)$$

where $(\dot{\delta}_i)_B$ is a derivative wrt the body fixed frame.

The total kinetic energy (i.e., translational plus rotational contributions) for a body may be written as

$$T = \frac{1}{2} \sum_i \left[m_i (\dot{r}_i \cdot \dot{r}_i) + (\omega + \dot{\theta}_i)^T I_i (\omega + \dot{\theta}_i) \right] \qquad (2.2\text{-}3)$$



m_i = i^{th} mass particle
O = reference point on flexible body
r_i = mass particle inertial position vector
R_O = body reference point inertial position vector
ρ_i = mass particle nominal position vector wrt O
δ_i = translational deformation
θ_i = rotational deformation
ω = "rigid body" angular velocity vector

Figure 2.2-1 Generic Flexible Body Configuration.

The somewhat lengthy derivation proceeds by substituting eqn. (2.2-2) into eqn. (2.2-3) and manipulating the kinetic energy function into a useful form. The translational and rotational elastic deformation vectors for the i^{th} mass particle are computed from modal expansions as given in eqn. (2.2-4).

$$\delta_i = \sum_{j}^{\#modes} \Phi_{ij}\,\eta_j \qquad \theta_i = \sum_{j}^{\#modes} \Psi_{ij}\,\eta_j \tag{2.2-4}$$

The translational and rotational mode shapes (Φ and Ψ, respectively) are commonly obtained from a finite element model (FEM) of the body. The resulting kinetic energy function is written in "shorthand" notation as

$$T = \frac{1}{2} M \dot{R}_o^{B2} + \frac{1}{2} \dot{\eta}^T M \dot{\eta} + \dot{R}_o^B \cdot \left[\omega \times \left(\overline{M\rho} + \sum_k^{nm} \Gamma_{ok} \eta_k \right) \right] + \dot{R}_o^B \cdot \sum_k^{nm} (\Gamma_{ok} \dot{\eta}_k)$$
$$+ \omega \cdot \sum_j^{nm} \sum_k^{nm} (\Gamma_{1k} \dot{\eta}_k + \Gamma_{2jk} \eta_j \dot{\eta}_k) + \frac{1}{2} \omega^T \sum_j^{nm} \sum_k^{nm} (I_{2jk} \eta_j \eta_k + 2 I_{1k} \eta_k + I_o) \omega$$

(2.2-5)

where Γ_{ok} and Γ_{1k} are the "standard" modal integrals and Γ_{2jk}, I_{1k}, and I_{2jk} are called the higher order modal integrals. The modal integral terms are functions of the mode shapes, FEM element masses, and FEM element inertias. The modal integrals are constants and appear in the final form of the equations of motion; they are typically computed from the FEM output data for use as inputs to a flex body dynamic simulation.

2.3 Flexible Body Potential Energy Function

The potential energy function for a flexible body may be assumed to consist only of the strain energy due to linear elastic deformation. In this case, all other conservative forces which are derivable from potential functions (e.g., gravity effects) must be applied as forcing functions (i.e., external or generalized forces) on the right hand side of the EOM.

The strain energy in a linear elastic material may be expressed in the form

$$V = \frac{1}{2} \overline{\delta}^T \overline{K} \overline{\delta} \qquad (2.3\text{-}1)$$

where $\overline{\delta}$ is the deformation due to flexibility and $\overline{K}$ is the FEM stiffness matrix. Here $\overline{\delta}$ represents both the translational and rotational deformation. Replacing $\overline{\delta}$ in eqn. (2.3-1) with a modal expansion yields

$$V = \frac{1}{2} \eta^T \overline{\Phi}^T \overline{K} \overline{\Phi} \eta \qquad (2.3\text{-}2)$$

where the modal matrix $\overline{\Phi}$ includes both the translational and rotational modal gains. If the mode shapes, $\overline{\Phi}$, are the system othonormal eigenvectors, then a new diagonalized stiffness matrix can be defined

$$\mathbf{K} = \overline{\Phi}^T \overline{K} \overline{\Phi} = \begin{bmatrix} \omega_1^2 & 0 & 0 \\ 0 & \omega_2^2 & 0 \\ 0 & 0 & \omega_n^2 \end{bmatrix} \tag{2.3-3}$$

where the diagonal elements are the squared natural frequencies of the system modes. The assumption of othonormal modes follows from NASA's request in the statement of work for the capability to input second order bending modes of the form

$$\ddot{\eta}_i + 2\zeta_i \omega_i \dot{\eta}_i + \omega_i^2 \eta_i = F_i . \tag{2.3-4}$$

The potential energy or strain energy may be written in terms of the diagonalized stiffness matrix

$$V = \frac{1}{2} \eta^T K \eta \tag{2.3-5}$$

Notice that the potential energy function is a function of the generalized modal coordinates η.

The augmented potential energy function used in Lagrange's quasi-coordinates eqns. (2.1-5) through (2.1-7) included constraints with Lagrange multipliers. The only constraint equation associated with the unconstrained single flex body model is the unity magnitude quaternion constraint. The augmented potential energy function can now be written as

$$V^* = V(q) - \lambda^T \phi(q)$$

$$V^* = \frac{1}{2}\eta^T K \eta^T - \lambda_1 (\varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 + \varepsilon_4^2 - 1) \qquad (2.3\text{-}6)$$

where λ_1 is the Lagrange multiplier.

2.4 Equations of Motion

The kinetic energy function (2.2-5) and the augmented potential energy function (2.3-6) may now be incorporated into Lagrange's quasi-coordinate equations (2.1-5) through (2.1-7). The body reference point (i.e., point O in Fig. 2.2-1) is assumed to be located at the body center of mass in order to simplify the equations somewhat. After lengthy algebraic manipulations, the "full-up" nonlinear EOM can be expressed in the following "shorthand" notation

$$\begin{bmatrix} M & -\widetilde{\Gamma_0 \eta} & \Gamma_0 \\ \widetilde{\Gamma_0 \eta} & \widetilde{II} & \Gamma_1 + \Gamma_2 \\ \Gamma_0^T & \Gamma_1^T + \Gamma_2^T & M \end{bmatrix} \begin{Bmatrix} \ddot{P}_{CM}^B \\ \dot{\omega}^B \\ \ddot{\eta} \end{Bmatrix} = F_L + F_{NL} + F_{APPL}$$

$$= \begin{Bmatrix} -\omega^B \times (M \dot{P}_{CM}^B + 2\Gamma_0 \dot{\eta}) - \omega^B \times (\omega^B \times \Gamma_0 \eta) + \sum_m^{APPL} F_m^B \\ -\dot{II}\,\omega^B - \omega^B \times II\,\omega^B - \Gamma_0 \eta \times (\omega^B \times \dot{P}_{CM}^B) - \omega^B \times (\Gamma_1 + \Gamma_2)\dot{\eta} + \sum_n^{APPL} T_n^B \\ -\omega_k^2 \eta - 2\zeta_k \omega_k \dot{\eta} + \dot{P}_{CM}^B \cdot (\omega^B \times \Gamma_0) + \sum_m^{APPL} \Phi_m^T F_m^B + \sum_n^{APPL} \Psi_n^T T_n^B \\ -2\,\omega^B \cdot \dot{\Gamma}_2 + \frac{1}{2}\omega^{BT}\left[2 I_{1r} + \sum_j^{nm} (I_{2jr} + I_{2rj})\eta_j\right]\omega^B \end{Bmatrix} \qquad (2.4\text{-}1)$$

where parts = number of mass particles

appl = number of externally applied forces or torques.

$$\mathbf{M} = \begin{bmatrix} M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & M \end{bmatrix} \qquad M = \sum_{i}^{parts} \begin{bmatrix} \Phi_i \\ \Psi_i \end{bmatrix} \begin{bmatrix} M_i & 0 \\ 0 & I_i \end{bmatrix} \begin{bmatrix} \Phi_i \\ \Psi_i \end{bmatrix}$$

$$\Gamma_0 = [\Gamma_{01}\ \Gamma_{02}\ \ldots\ \Gamma_{0n}] \qquad \Gamma_1 = [\Gamma_{11}\ \Gamma_{12}\ \ldots\ \Gamma_{1n}]$$

$$\Gamma_2 = \sum_{j}^{nm} \eta_j [\Gamma_{2j1}\ \Gamma_{2j2}\ \ldots\ \Gamma_{2jn}] \qquad \dot{\Gamma}_2 = \sum_{j}^{nm} \dot{\eta}_j [\Gamma_{2j1}\ \Gamma_{2j2}\ \ldots\ \Gamma_{2jn}]$$

$$\Phi_i = [\Phi_{i1}\ \Phi_{i2}\ \ldots\ \Phi_{in}] \qquad \Psi_i = [\Psi_{i1}\ \Psi_{i2}\ \ldots\ \Psi_{in}]$$

$$II = I_{CM} + (I_{1k} + I_{1k}^T)\,\eta_k + \frac{1}{2}(I_{2jk} + I_{2kj})\,\eta_j\,\eta_k$$

$$\dot{II} = (I_{1k} + I_{1k}^T)\,\dot{\eta}_k + \frac{1}{2}(I_{2jk} + I_{2kj})\,\dot{\eta}_j\,\eta_k + \frac{1}{2}(I_{2jk} + I_{2kj})\,\eta_j\,\dot{\eta}_k \qquad (2.4\text{-}2)$$

"Standard" Modal Integrals:

$$\Gamma_{0k} = \sum_{i}^{parts} m_i\,\Phi_{ik}$$

$$\Gamma_{1k} = \sum_{i}^{parts} (m_i\,\rho_i \times \Phi_{ik} + I_i\,\Psi_{ik}) \qquad (2.4\text{-}3)$$

Higher Order Modal Integrals:

$$\Gamma_{2jk} = \sum_{i}^{parts} m_i\,\Phi_{ij} \times \Phi_{ik}$$

$$I_{1k} = \sum_{i}^{parts} m_i[(\rho_i \cdot \Phi_{ik})\,1 - \Phi_{ik}\,\rho_i^T)]$$

$$I_{2jk} = \sum_{i}^{parts} m_i[(\Phi_{ij}^T\,\Phi_{ik})\,1 - \Phi_{ij}\,\Phi_{ik}^T)] \qquad (2.4\text{-}4)$$

where parts = number of mass particles.

Notice that for the "full-up" nonlinear case the mass matrix has time varying elements; this implies that periodic updates of the time varying elements are required in the flexible body math model.

Examination of the "full-up" nonlinear EOM reveals that the equations can be written in levels of increasing simplicity by neglecting certain higher order effects. First, neglecting the higher order modal integral terms allows the EOM to be written as

$$\begin{bmatrix} M & -\widetilde{\Gamma_0 \eta} & \Gamma_0 \\ \widetilde{\Gamma_0 \eta} & I_{CM} & \Gamma_1 \\ \Gamma_0^T & \Gamma_1^T & M \end{bmatrix} \begin{Bmatrix} \ddot{P}^B_{CM} \\ \dot{\omega}^B \\ \ddot{\eta} \end{Bmatrix} =$$

$$\begin{Bmatrix} - \omega^B \times (M \dot{P}^B_{CM} + 2 \Gamma_0 \dot{\eta}) - \omega^B \times (\omega^B \times \Gamma_0 \eta) + \sum_m^{APPL} F^B_m \\ - \omega^B \times I_{CM} \omega^B - \Gamma_0 \eta \times (\omega^B \times \dot{P}^B_{CM}) - \omega^B \times (\Gamma_1 \dot{\eta}) + \sum_n^{APPL} T^B_n \\ - \omega_k^2 \eta - 2 \zeta_k \omega_k \dot{\eta} + \dot{P}^B_{CM} \cdot (\omega^B \times \Gamma_0) + \sum_m^{APPL} \Phi^T_m F^B_m + \sum_n^{APPL} \Psi^T_n T^B_n \end{Bmatrix} \quad (2.4\text{-}5)$$

Eliminating the higher order modal integrals reduces the computations per simulation cycle. Notice in eqn. (2.4-5) that if the "standard" modal integrals were zero (i.e., free-free modes assumption discussed below), then neglecting the higher order modal integrals would eliminate the time varying terms from the mass matrix; this would allow a "one-time" computation of the mass matrix in the initialization and a significant decrease in simulation cycle time.

Secondly, elimination of the remaining nonlinear terms (i.e., includes elimination of the higher order modal integrals) on the right hand side of the EOM leads to another level of simplification

$$\begin{bmatrix} \mathbf{M} & -\widetilde{\Gamma_0 \eta} & \Gamma_0 \\ \widetilde{\Gamma_0 \eta} & \mathbf{I}_{CM} & \Gamma_1 \\ \Gamma_0^T & \Gamma_1^T & M \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{P}}_{CM}^B \\ \dot{\omega}^B \\ \ddot{\eta} \end{Bmatrix} = \begin{Bmatrix} \sum_m^{APPL} \mathbf{F}_m^B \\ \sum_n^{APPL} \mathbf{T}_n^B \\ -\omega_k^2 \eta - 2\zeta_k \omega_k \dot{\eta} + \sum_m^{APPL} \Phi_m^T \mathbf{F}_m^B + \sum_n^{APPL} \Psi_n^T \mathbf{T}_n^B \end{Bmatrix} \tag{2.4-6}$$

Eliminating the nonlinear terms on the right hand side greatly reduces the computational requirements per simulation cycle and again decreases cycle time.

Another simplification level results from the use of free-free modes for the unconstrained flexible body. The "standard" modal integral terms are identically zero when free-free modes are used. The result is a significant simplification of the mass matrix (and the right hand side if the previous simplification was not invoked)

$$\begin{bmatrix} \mathbf{M} & 0 & 0 \\ 0 & \mathbf{I}_{CM} & 0 \\ 0 & 0 & M \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{P}}_{CM}^B \\ \dot{\omega}^B \\ \ddot{\eta} \end{Bmatrix} = \begin{Bmatrix} \sum_m^{APPL} \mathbf{F}_m^B \\ \sum_n^{APPL} \mathbf{T}_n^B \\ -\omega_k^2 \eta - 2\zeta_k \omega_k \dot{\eta} + \sum_m^{APPL} \Phi_m^T \mathbf{F}_m^B + \sum_n^{APPL} \Psi_n^T \mathbf{T}_n^B \end{Bmatrix} \tag{2.4-7}$$

The first and second simplifications resulting in eqns (2.4-5) and (2.4-6) are not prerequisites for applying the free-free mode simplification. However, using the free-free mode simplification in conjunction with eliminating the higher order modal integral terms produces a constant mass matrix which can be computed once in the initialization. Notice that the translational, rotational, and flexibility equations are completely decoupled with respect to one another when the following simplifications are applied: (1) neglect higher order mass integral terms, (2) neglect remaining right hand side nonlinear terms, and (3) utilize free-free modes.

NOTE: It is important to invoke the free-free mode option explicitly in the math model configuration data when free-free modes are being used to characterize body flexibility. Invoking the free-free option eliminates unnecessary computations that will otherwise be performed with standard modal integrals which are zero (or very small due to numerical rounding).

A final simplification of the EOM results when no flex modes are included (i.e., rigid body). In this case, the EOM reduce to the well known linear form of the Newton-Euler equations for translation and rotation

$$\begin{bmatrix} \mathbf{M} & 0 \\ 0 & \mathbf{I}_{CM} \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{P}}^{B}_{CM} \\ \dot{\boldsymbol{\omega}}^{B} \end{Bmatrix} = \begin{Bmatrix} \sum_{m}^{APPL} \mathbf{F}^{B}_{m} \\ \sum_{n}^{APPL} \mathbf{T}^{B}_{n} \end{Bmatrix} \qquad (2.4\text{-}8)$$

All of the above mentioned EOM complexity levels are made available to the user of the two-flex body math model. The user will configure the math model to the desired complexity level through the use of flags and parameters in ASCII input data files.

2.5 Equations of Motion Solution

Regardless of the level of complexity, the equations of motion can be expressed in the general form

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \qquad (2.5\text{-}1)$$

where **A** = system mass matrix
x = quasi-coordinate vector derivative
b = right hand side vector.

The mass matrix (**A**) and the right hand side vector (**b**) are computed (or known) at each point in time. The objective is to solve the linear equation set for the quasi-coordinate vector derivative (**x**) such that numerical integration can be performed to propagate the system states.

Recall that depending on the EOM complexity options selected by the user, the system mass matrix may or may not vary with time. If the mass matrix is not time varying (i.e., constant elements) then a "one-time" solution approach to the linear equation set of (2.5-1) is desired. This would allow a portion of the computations associated with solving (2.5-1) to be performed once in an initialization routine and, thus, reduce the math model cycle time.

A numerical manipulation method called LU (lower-upper) decomposition was selected for solving eqn. (2.5-1). The LU decomposition decomposes the mass matrix into a product of a lower triangular matrix (**L**) and an upper triangular matrix (**U**).

$$\mathbf{A} \cdot \mathbf{x} = (\mathbf{L} \cdot \mathbf{U})\ \mathbf{x} = \mathbf{L} \cdot (\mathbf{U} \cdot \mathbf{x}) = \mathbf{b}$$

$$\mathbf{L} = \begin{bmatrix} \alpha_{11} & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 \\ \alpha_{n1} & \alpha_{n2} & \alpha_{nn} \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{1n} \\ 0 & \beta_{22} & \beta_{2n} \\ 0 & 0 & \beta_{nn} \end{bmatrix} \qquad (2.5\text{-}2)$$

The decomposition is based on a numerical manipulation technique called Crout's algorithm. Once the mass matrix has been decomposed, the problem of solving for the desired vector **x** is broken into two smaller linear equation problems which can be solved using backwards and forwards substitution

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b} \quad \text{(forward)} \tag{2.5-3}$$

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y} \quad \text{(backward)} \tag{2.5-4}$$

Eqn. (2.5-3) can be solved using forward substitution since **L** is lower triangular; eqn. (2.5-4) can be solved using backward substitution since **U** is upper triangular.

Notice that the LU decomposition of the mass matrix can be used to solve for **x** with any right hand side **b**. Thus, for a constant mass matrix the LU decomposition needs only to be computed once during initialization; given a new right hand side vector **b** during each simulation cycle, only the forward/backward substitution operations associated with eqns. (2.5-3) and (2.5-4) are required to solve for **x**. Another possible solution method is to compute the mass matrix inverse $\mathbf{A}^{-1}$. For the constant mass matrix case only a matrix-vector multiplication (given the computed **b** vector) would be required during each cycle to solve for **x**

$$\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b}. \tag{2.5-4}$$

However, for the time varying mass matrix case the LU decomposition (performed each cycle) is numerically more efficient than computing the matrix inverse.

3.0 CONTACT FORCE / MOMENT TRANSFORMATION

During hardware-in-the-loop (HWIL) simulations, the math model requires input from the force/moment (F/M) sensor located on the ceiling of the MTB six DOF facility. The sensor location is assumed to correspond to some arbitrary position on the target body. Transformations are required in order to determine the appropriate force/moment applied to the docking node of each body. The body docking node is defined as the single point on the body through which all contact forces and moments are applied to the body.

The F/M sensor measures the total forces and torques applied to the target body due to hardware contact between the target and chase bodies. It is assumed that a rigid link exists in the target body between the F/M sensor and the docking mechanism; therefore, all contact forces and moments, from all points of contact, are detected by the F/M sensor.

It is also assumed that the docking mechanism mass (for each body) is much smaller than the body masses. This assumption implies that the inertial forces acting on the moving parts of the docking mechanisms may be neglected. In this case, the contact forces and moments between the two bodies are equal and opposite for collocated docking nodes.

Figure 3.0-1 defines the position vectors from the F/M sensor to the target docking node (D1) and to the chase docking node (D2). Due to the rigid link assumption, $\mathbf{P}_{D1S1}$ remains constant in a frame attached to either the sensor location or the target docking node. However, $\mathbf{P}_{D2S1}$ includes the effects of deformation of both the sensor location (or D_1 since they are rigidly connected) and the chase docking node. $\mathbf{F}_{S1}$ and $\mathbf{M}_{S1}$ are defined as the sensed force and moment. $\mathbf{F}_{D1}$, $\mathbf{M}_{D1}$, $\mathbf{F}_{D2}$ and $\mathbf{M}_{D2}$ are defined as the contact force and moment applied at the target docking node and chase docking node, respectively.

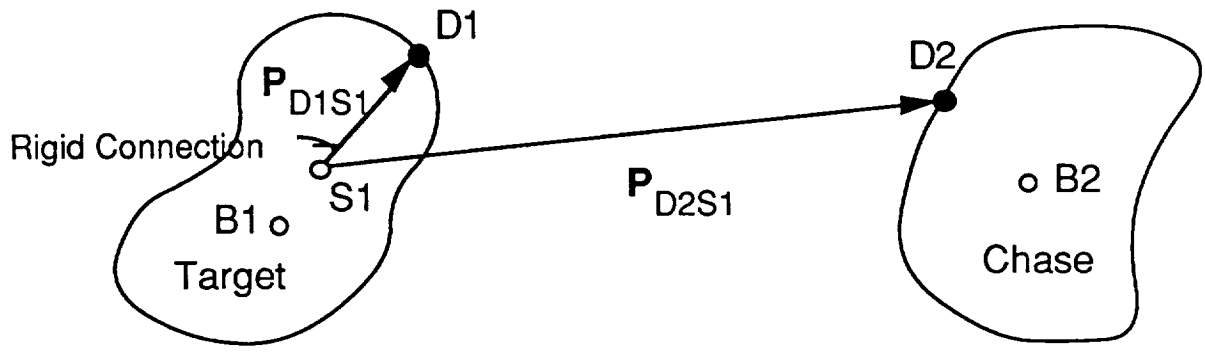


Figure 3.0-1 F/M Sensor to Docking Node Position Vector Definitions.

Due to the rigid link assumption, the forces can be transferred directly from the sensor location to the docking nodes as

$$\mathbf{F}_{D1} = -\mathbf{F}_{D2} = \mathbf{F}_{S1} \qquad (3.0\text{-}1)$$

The moment transformations include the effect of non collocated docking nodes and are expressed as

$$\mathbf{M}_{D1} = \mathbf{M}_{S1} - \mathbf{P}_{D1S1} \times \mathbf{F}_{D1} \qquad (3.0\text{-}2)$$

$$\mathbf{M}_{D2} = -\mathbf{M}_{S1} - \mathbf{P}_{D2S1} \times \mathbf{F}_{D2} \qquad (3.0\text{-}3)$$

Notice that if the docking nodes are collocated (i.e., $\mathbf{P}_{D2S1} = \mathbf{P}_{D1S1}$) the contact moments are also equal and opposite.

4.0 RELATIVE BODY MOTION

The role of the two-flex body math model is to simulate the dynamic response of the two bodies when acted upon by contact, control, and any other modeled external forces and torques. The most important output from the math model is the relative motion between the target and chase docking nodes. This data is used to generate the commands which drive the six DOF motion system.

The transformation matrix describing the orientation of the chase docking frame with respect to the target docking frame must include the effects of flexibility. This is most easily expressed by defining undeformed node frames (i.e., rigid body node frames) for use as intermediate transform orientations. Following this approach, the 3 x 3 transformation matrix from the D2 to the D1 node frame may be written as

$$[D1\ \ D2] = [I\ \ D1]^T [I\ \ D2] \qquad (4.0\text{-}1)$$

$$= [D1_0\ \ D1]^T\ [B1\ \ D1_0]^T\ [I\ \ B1]^T\ [I\ \ B2]\ [B2\ \ D2_0]\ [D2_0\ \ D2]$$

where $D1_0$ and $D2_0$ are undeformed docking node coordinate frames and I represents the inertial reference frame. The transformation from the deformed node frame to the undeformed node frame can be expressed in terms of the infinitesimal rotations about each axis

$$[D1_0\ \ D1] = [R_{x,d\Theta_x}\ \ R_{y,d\Theta_y}\ \ R_{z,d\Theta_z}] = [1 + \widetilde{d\Theta}_{D1}]. \qquad (4.0\text{-}2)$$

Figure 4.0-1 defines the vectors which are used in the equations describing the relative motion. The vectors representing deformation are not shown explicitly in Fig. 4.0-1. The position of the chase docking node with respect to the target docking node may be expressed as

$$\mathbf{P}_{D2D1} = \mathbf{P}_{B2I} + \mathbf{P}_{D2_0B2} + \delta_{D2D2_0} - \mathbf{P}_{B1I} - \mathbf{P}_{D1_0B1} - \delta_{D1D1_0} \quad (4.0\text{-}3)$$

where the δ vectors represent the effects of flexibility. The result is a vector and may be expressed in any coordinate frame.

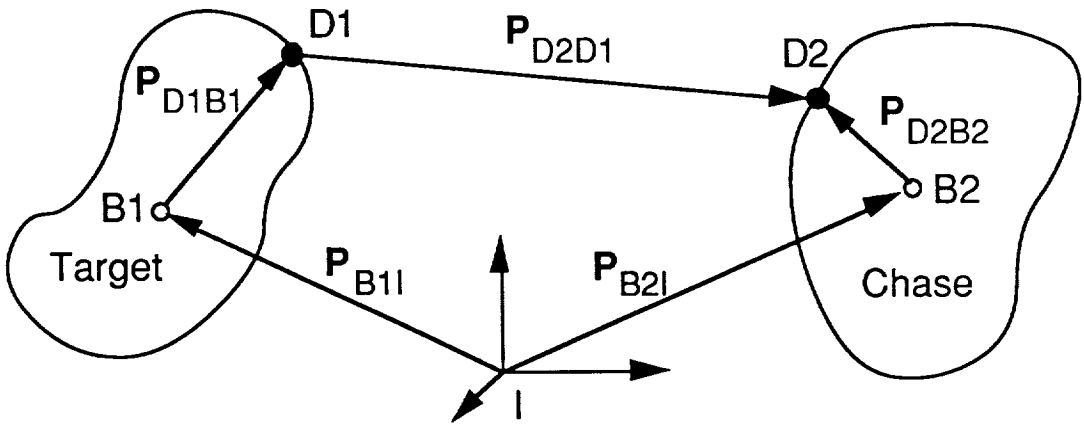


Figure 4.0-1 Relative Body Position Vector Definitions.

The relative velocity between docking nodes is defined as the velocity of the chase docking node as seen by an observer moving with the target docking node. Therefore, the derivative is taken wrt the target docking node frame. Equation (4.0-4) is the well known relationship for the time derivative of a vector viewed from two frames rotating at different angular rates.

$$(\dot{\mathbf{R}})_A = (\dot{\mathbf{R}})_B + \omega_{BA} \times \mathbf{R} \quad (4.0\text{-}4)$$

where $(\dot{\mathbf{R}})_A$ = time derivative of an arbitrary vector **R** wrt A frame

$(\dot{\mathbf{R}})_B$ = time derivative of the vector **R** wrt B frame

ω_{BA} = angular velocity of B frame wrt A frame.

Using the vector definitions in Fig. 4.0-1 and applying eqn. (4.0-4) results in the expression for the time derivative of the position of D2 wrt D1 taken wrt the D1 frame.

$$\left(\dot{\mathbf{P}}_{D2D1}\right)_{D1} = \dot{\mathbf{P}}_{B2I} - \dot{\mathbf{P}}_{B1I} + \boldsymbol{\omega}_{B2I} \times \left(\mathbf{P}_{D2_0B2} + \sum_{j} \boldsymbol{\Phi}_{D2j}\, \eta_j \right) + \sum_{j}^{nm} \boldsymbol{\Phi}_{D2j}\, \dot{\eta}_j$$

$$- \boldsymbol{\omega}_{B1I} \times \left(\mathbf{P}_{D1_0B1} + \sum_{j}^{nm} \boldsymbol{\Phi}_{D1j}\, \eta_j \right) - \sum_{j}^{nm} \boldsymbol{\Phi}_{D1j}\, \dot{\eta}_j$$

$$- \left(\boldsymbol{\omega}_{B1I} + \sum_{j}^{nm} \boldsymbol{\Psi}_{D1j}\, \dot{\eta}_j + \boldsymbol{\omega}_{B1I} \times \sum_{j}^{nm} \boldsymbol{\Psi}_{D1j}\, \dot{\eta}_j \right) \times \mathbf{P}_{D2D1} \qquad (4.0\text{-}5)$$

The deformation vectors are represented by their respective modal expansions in this expression.

The relative angular velocity of the chase docking frame wrt the target docking frame is given in eqn. (4.0-6).

$$\boldsymbol{\omega}_{D2D1} = \boldsymbol{\omega}_{B2I} + \sum_{j}^{nm} \boldsymbol{\Psi}_{D2j}\, \dot{\eta}_j + \boldsymbol{\omega}_{B2I} \times \left(\sum_{j}^{nm} \boldsymbol{\Psi}_{D2j}\, \eta_j \right) - \boldsymbol{\omega}_{B1I}$$

$$- \sum_{j}^{nm} \boldsymbol{\Psi}_{D1j}\, \dot{\eta}_j - \boldsymbol{\omega}_{B1I} \times \left(\sum_{j}^{nm} \boldsymbol{\Psi}_{D1j}\, \eta_j \right) \qquad (4.0\text{-}6)$$

Again, the rotational deformation vectors are expressed as a modal expansion.

5.0 SIMULATION DESCRIPTION

The computer code which implements the MTB flexible body math model is coded in FORTRAN 77 and all floating point numbers are maintained with double precision accuracy. A listing of the computer code, along with input data files, is contained in Appendix A. The math model is coded in a modular form to facilitate debugging and modifications. The math model is controlled by the driver subroutine "DYNAMIC" which is called by the host simulation. This driver subroutine and the common declarations in the include file "INTFAC.INC" serve as the interface to the host simulation.

The following nomenclature is used for specific data extracted from coordinate vectors. Position vectors begin with the letter P. Translational velocity vectors begin with the letters PD. The vector P_A_B_C is defined as the position vector from point B to point A expressed in the C frame coordinates; PD_A_B_C is the time derivative of P_A_B_C. Angular velocity vectors begin with the letter W and derivatives follow the pattern described above. Transformation matrices begin with the letter T. The transformation T_A_B is a 3 x 3 matrix which transforms vectors from the B frame to the A frame. The labels used to define bodies, docking nodes, etc., in Figures 3.0-1 and 4.0-1 are also used in the simulation code.

5.1 Major Subroutines

Figure 5.1-1 is a flow diagram of the MTB two flex body math model. A general description of each of the major subroutines and include files is given in Table 5.1-1 and Table 5.1-2, respectively. The first pass through the model initializes the simulation variables. Data from the force/moment sensor is supplied to the subroutine "FMTRANS" through common declarations in the include file "INTFAC.INC". The relative motion data is made available to the host simulation through common declarations in the include file "INTFAC.INC".

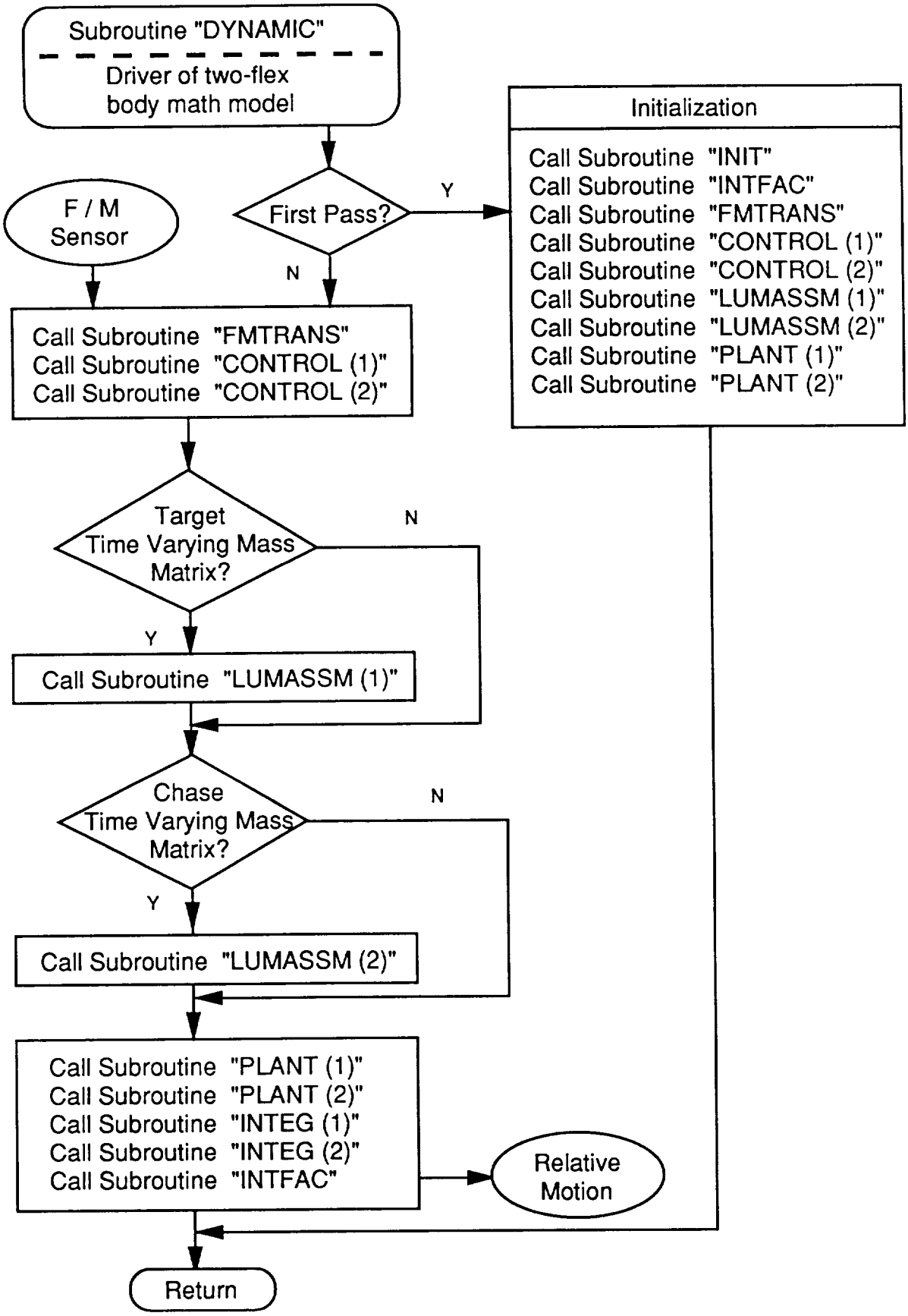


Figure 5.1-1 Two Flexible Body Math Model Flow Diagram.

Table 5.1-1 Two Flex Body Math Model Subroutines.

Subroutine	Description
DYNAMIC	Driver of two-flex body dynamics model; called by main routine.
INIT	Initialization of mass properties, geometry, state vectors, flexibility data, etc.
FMTRANS	Computes contact forces/torques at docking port nodes based on force/torque sensor data.
CONTROL (NB)	Computes control forces/torques applied to NB^{th} body.
LUMASSM (NB)	Computes mass matrix of NB^{th} body in lower-upper decomposed form.
PLANT (NB)	Computes right hand side terms (e.g., damping, stiffness, forcing functions, etc.) of equations of motion and solves **A x=b** for NB^{th} body.
INTEG (NB)	Integrates derivatives of quasi and generalized coordinate vectors and updates generalized coordinate vector derivative for NB^{th} body.
INTFAC	Updates relative transformations, position vectors and rate vectors with flexibility included.

Table 5.1-2 Two Flex Body Simulation Include Files.

INCLUDE FILE	DESCRIPTION
BODY.INC	Variable/parameter declarations and common specifications for mass properties, coordinate vectors, flexibility, etc. for both bodies; also simulation control variables.
INTFAC.INC	Variable declarations and common specifications for interfacing to main routine; includes all necessary relative motion data.

5.2 Input Data

The MTB flex body simulation has three ASCII input data files. Table 5.2-1 gives a general description of the contents of these input data files. A computer listing of example input data files is contained in Appendix A.

The ASCII input data may be modified using any available editor (e.g., the VI editor on the UNIX based ALLIANT computer system). The existing input files contain comments (see example files in Appendix A) which help to define the input data. Most of the input data is well described by the comments. The EOM complexity options are defined with the following acronyms:

EOM Option	Description
EHOMIT	Eliminate higher order modal integral terms
ERNLT	Eliminate remaining right hand side nonlinear terms Note: Enabling ENRLT encompasses EHOMIT regardless of the EHOMIT option status.
FREE-FREE	Free-Free modes

The EOM complexity options are declared as logical variables. Therefore, ".T." is TRUE and enables the option; ".F." is FALSE and disables the option.

The pre-processed flex data (see Table 5.2-1) forms the bulk of the required flexibility data and is generated by the flexible body data pre-processing algorithm discussed in Section 5.3. The remaining body data can be generated using a file editor and then concatenated with the output of the pre-processing algorithm to produce the whole input data file.

Currently, all the input data specified in Table 5.2-1 is read regardless of the EOM options. For example, enabling the EHOM-IT option does not eliminate the need for higher order modal integral data in the respective body input data file even though the data is not used in the simulation. This follows from the fact that the

Table 5.2-1 Two Flex Body ASCII Input Data Files.

INPUT FILE	DESCRIPTION	
MTBSIM.INP	Simulation control data:	Integration step size
		Screen output step size
		Data file output step size
		Stop time
		Integration method flag
BODY1.INP	Target body data:	Mass
		Inertia matrix
		Initial P_B_I_I
		Initial PD_B_I_I
		Initial w_B_I_B
		[I B] Euler rotation sequence
		[I B] Euler rotation angles
		Number of nodes; dock node num
		Node geometry
		*P_D1_S1_S1
		[B1 $D1_0$] Transformation
		*[D1 S1] Transformation
		EOM complexity options
	Flexibility Data:	Number of flexible modes
		Mode frequencies
		Modal damping
		Initial generalized modal coords.
		Initial gen modal coord derivatives
	Pre-Processed Flex Data:	Generalized mass matrix
		Translational modal gains
		Rotational modal gains
		Standard modal integrals
		Higher order modal integrals
BODY2.INP	Chase body data:	Same as target body data omitting *

flexible body data pre-processing routine (Section 5.3) computes and outputs all modal integral terms. An advantage of this approach is that the input data files (for a given body) have a "standard" structure (i.e., same contents).

5.3 Flexible Body Data Pre-Processing Algorithm

Flexible body simulations require a great deal more input data than pure rigid body simulations. Typically, the simulation input data (e.g., modal integrals and generalized mass matrix) is computed from the output of a finite element model (FEM). To assist users of the MTB flexible body simulation, Control Dynamics coded a flexible body data pre-processing algorithm. This algorithm acts as a post-processor for FEM data and a pre-processor for the MTB flexible body simulation. Figure 5.3-1 is a block diagram which illustrates the role of the algorithm.

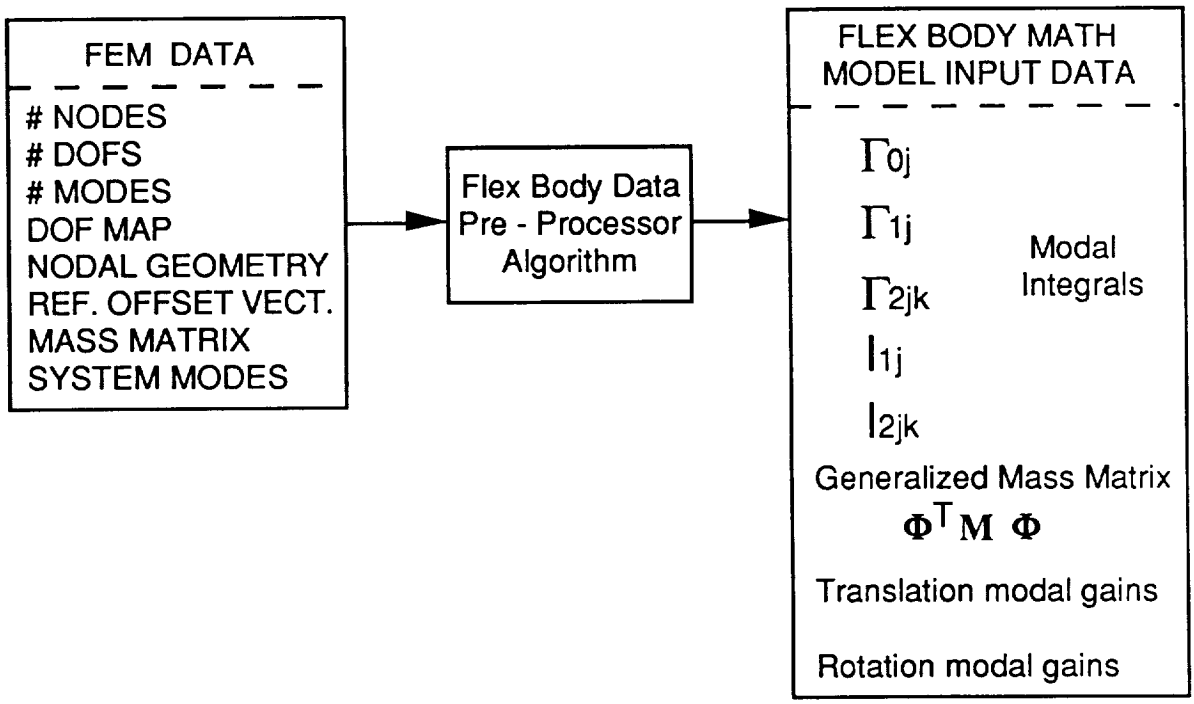


Figure 5.3-1 Flexible Body Data Pre-Processing Algorithm.

Table 5.3-1 describes the flexible body data required in the ASCII input file for the pre-processing algorithm. This data is read using free format (see pre-processor code listing in Appendix C). The output (see flexibility data in Table 5.2-1) is written to the user specified file in a format consistent with the inputs to the MTB math model simulation. The output file from the pre-processor contains the data denoted as pre-processed flex data in Table 5.2-1.

Table 5.3-1 Flexible Body Data Pre-Processor Input.

Input Description	Explicit Input Data
Degree of Freedom Map	# nodes, # DOF, # modes
	Blank line
	node num (i), DOF type (i)
	Blank line
Node Geometry Table	*Offset vector
	Node position vector (i)
	Blank line
Mass Matrix	Mass_mat (# DOF, # DOF)
	Blank line
Modal Matrix	Phi (# DOF, # modes)

*A displacement vector subtracted from each node position vector which allows the body reference point to be relocated.

6.0 SIMULATION VERIFICATION

The math model simulation was verified in a series of progressive steps on a PC-386 computer at Control Dynamics Company prior to installation on the host computer . This section of the report documents the various tests used to verify the math model simulation. Time domain results for all of the various simulation component tests are not presented; only the results from an "all-encompassing" two flexible body comparison to the multi-flex body code TREETOPS is shown. The "full-up" flexible body verification run presented in section 6.4 was repeated after the model was installed on the host computer to ensure proper operation of the model in the MTB facility. This "full-up" run exercises all aspects of the simulation and, therefore, eliminates the need to reverify the simulation in a component manner on the host computer.

6.1 Equations of Motion Solution Verification

The LU decomposition technique used to solve for the derivative of the quasi-coordinate vector is described in Section 2.5 of this report. Since the LU decomposition and forward/backward substitution algorithms were coded to solve a general linear set of equations $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ (i.e., the symmetry property of the mass matrix was not considered), the algorithm may be verified using any arbitrary deterministic system with a square matrix **A**. The LU decomposition algorithms were used to solve several linear systems of various size. The same systems were then solved using the well established computer analysis tool called MATLAB. The solution vectors (**x**) were compared to ensure proper operation of the EOM solution technique.

6.2 Rigid Body Verification

The rigid body portion of the math model was tested by two means. First, each degree of freedom (i.e., 3 translations and 3 rotations per body) was tested independently through the application of a constant force/moment. The results were verified by hand calculations.

Secondly, the simple two rigid body system shown in Figure 6.2-1 was used to test portions of the relative vehicle motion calculations and the contact force/moment transformation. For the system in Fig. 6.2-1, an omnidirectional spring of stiffness K and undeformed length r was assumed at the target docking node in order to simulate contact forces. Various initial conditions on the chase vehicle were used to induce contact between the docking ports in each translational axis. The resulting rigid body motion was verified through hand calculations. Also, forces and torques representing contact were input as "sensed" values to simulate data from the F/M sensor in the MTB facility. The math model operated on these "sensed" values to produce contact forces and torques at the body docking nodes. Again, the results were validated using hand calculations.

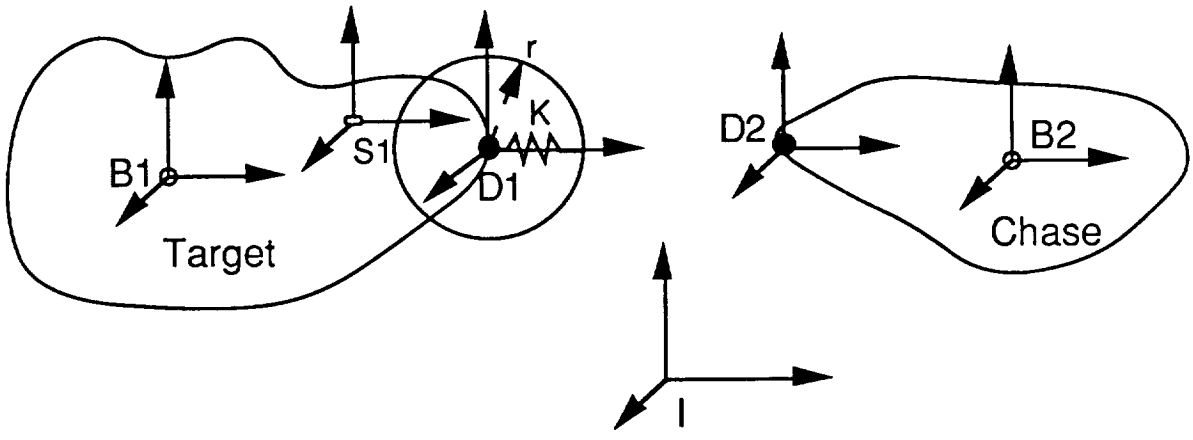
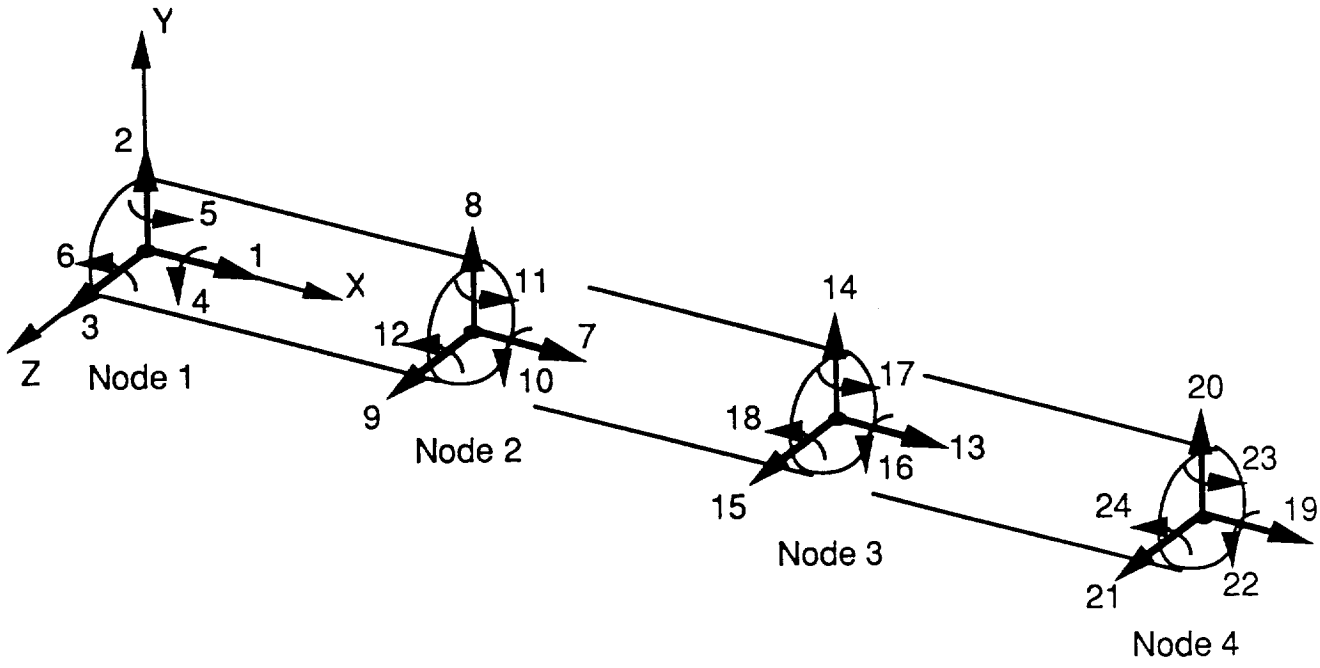


Figure 6.2-1 Two Rigid Body Contact Test Case.

6.3 Flexible Beam Model

The flexible body math model was tested using a free-free flexible beam model created in the structural computer code SMIS (Structures and Matrix Interpretive System). The beam geometry and properties are shown in Figure 6.3-1.



Length = 24.78 m
Radius = 0.1 m
Density = 8410 kg/m^3
Elasticity = 100 x 10^9 Nt/m^2
Rigidity = 30 x 10^9 Nt/m^2

$$ICM = \begin{bmatrix} 32.74 & 0 & 0 \\ 0 & 335K & 0 \\ 0 & 0 & 335K \end{bmatrix} \text{ kg - } m^2$$

Figure 6.3-1 Properties of Free-Free Flexible Test Beam.

The beam was sized to yield a first bending mode near 1 Hz. Table 6.3-1 lists the first four flexible mode frequencies from the SMIS model of the beam described in Figure 6.3-1.

Table 6.3-1 First Four Bending Modes of Free-Free Flexible Test Beam.

Mode Number	Frequency (Hz)	Description
1	1.003	Z Translation/Y Rotation
2	1.003	Y Translation/Z Rotation
3	2.770	Z Translation/Y Rotation
4	2.770	Y Translation/Z Rotation

The SMIS flexible beam model (i.e., mass matrix, mode shapes, DOF map, node geometry, etc.) was processed by the flexible body data processing algorithm discussed in Section 5.3 The processor produced the generalized mass matrix, modal gains, and modal integrals required for the MTB flexible body input data files (See Section 5.2).

6.4 Flexible Body Verification

The flexible beam model was implemented in two other flexible body simulations for comparisons to the MTB math model. Table 6.4-1 summarizes the important differences among the simulations. First, the model was implemented in an in-house (CDy) flex body simulation based on Lagrange's quasi-coordinate EOM (see Section 2.0), but coded independently. This simulation comparison facilitated rapid debugging since the terms in the equations of motion had a "one-to-one" correspondence. Secondly, the flexible beam model was implemented in the multi-flex body simulation called TREETOPS. TREETOPS is based on Kane's equations and gives a totally independent verification of the MTB math model. The majority of the data in TREETOPS is maintained as single precision. Notice also (from Table 6.4-1) that both test simulations use a different integration method than the MTB simulation. Only a comparison between the MTB math model and

TREETOPS is presented in this report.

Table 6.4-1 Simulation Comparison.

Simulation	Precision	Integration Method(s)
MTB Math Model	Double	HWIL Schemes
CDy In-House	Double	4-Pass Runge-Kutta
TREETOPS	Single	4-Pass Runge-Kutta

Several flexible body test cases were investigated using different "contact" forcing functions and body configurations. Only one "full-up" two flexible body comparison to TREETOPS is presented here; this run exercises "all" features of the MTB flexible body math model.

Figure 6.4-1 illustrates the system configuration and contact force/moment application points for the flexible body verification run. Equations (6.4-1) and (6.4-2) define the "contact" forcing function applied to the target docking node (D1). Notice

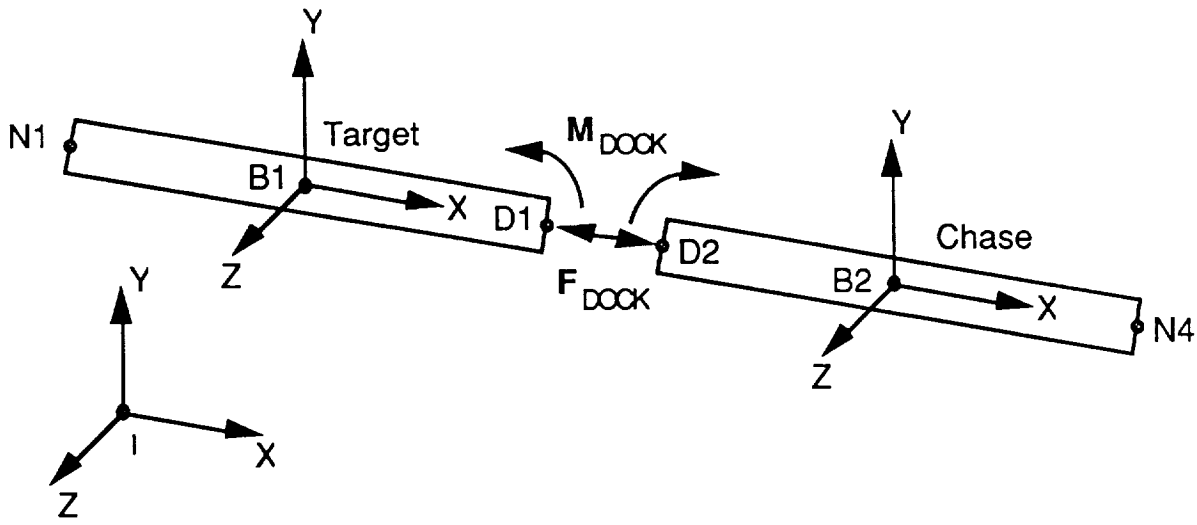


Figure 6.4-1 Two Flexible Beam Test Case.

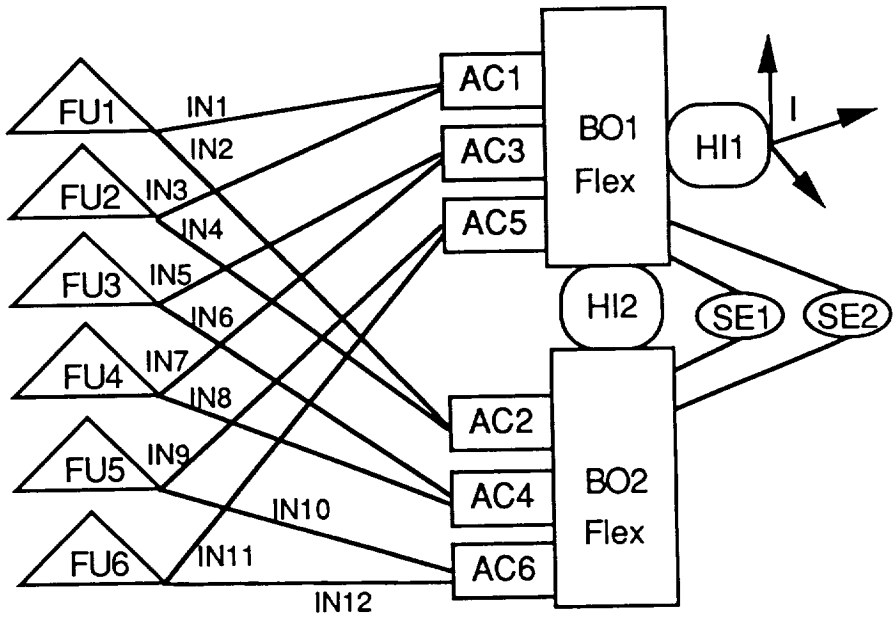
$$\begin{Bmatrix} F_{DOCK} \\ M_{DOCK} \end{Bmatrix} = \begin{Bmatrix} \begin{pmatrix} -3K \\ 2K \\ 0 \end{pmatrix} \text{Nt.} \\ \begin{pmatrix} 0 \\ 15K \\ 0 \end{pmatrix} \text{Nt-m.} \end{Bmatrix} \quad 0 \le t < 0.02 \text{ sec.} \tag{6.4-1}$$

$$\begin{Bmatrix} F_{DOCK} \\ M_{DOCK} \end{Bmatrix} = \begin{Bmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \text{Nt.} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \text{Nt-m.} \end{Bmatrix} \quad t \ge 0.02 \text{ sec.} \tag{6.4-2}$$

that the "contact" forcing function is applied over a short time period (impulsive loading) which is characteristic of actual contact.

Figure 6.4-2 is a TREETOPS block representation of the two flexible beam test case shown in Figure 6.4-1 with the forcing functions defined in eqns. (6.4-1) and (6.4-2). The function generators are step functions which produce the forcing functions; position and negative step functions are used to "turn-on" and "turn-off" the forces and moments. Actuators 1, 2, 3, and 4 are reaction jets which apply a force at a given node; these actuators implement the equal and opposite "contact" forces at the body docking nodes. Actuators 5 and 6 are moments actuators which implement the equal and opposite "contact" moments. In TREETOPS, the axis of application of these actuators changes in the body fixed frame (i.e., undeformed body frame) due to body flexing effects. Body 1 and 2 are both three mode flexible models of the beam shown in Figure 6.3-1. Hinges 1 and 2 are configured with zero translational and rotational stiffness to allow the bodies to respond as "free-floaters" (i.e., unconstrained bodies). Sensor 1 is an inertial position vector sensor which outputs the position of the chase docking node (D2) with respect to the target docking node (D1). Sensor 2 is an inertial velocity sensor which outputs the velocity of D2 with respect to D1. The TREETOPS simulation used Runge-Kutta

integration with a step size of 0.01 seconds; data was also written out every 0.01 seconds.



AC = Actuator
BO = Body
FU = Function Generator
HI = Hinge
IN = Interconnect
SE = Sensor

Figure 6.4-2 TREETOPS Block Representation of Two Flexible Beam Test Case.

The "contact" forcing function defined in eqns (6.4-1) and (6.4-2) was implemented in the MTB math model by computing an equivalent "sensed" force and moment (i.e., at some arbitrary F/M sensor location on the target body). This "sensed" force and moment was then used as if the data came from a physical sensing device; the "sensed" force and moment was transformed to the appropriate body docking nodes and applied as an external forcing function. The MTB was initialized such that the docking nodes were collocated at time zero. Referring to Section 3.0 of this report, the "contact" forces will be equal and opposite as in the

TREETOPS simulation. However, the "contact" moments will be equal and opposite only for collocated docking nodes; the TREETOPS simulation of this two flexible beam problem will apply equal and opposite "contact" moments over the entire duration of the forcing function. The results should still be comparable since the "contact" forcing function is of short duration; thus, the relative displacement of the docking nodes should be small during the forcing period.

In the MTB simulation, both the target and chase bodies are three mode flexible models of the beam shown in Figure 6.3-1. The modal damping was zero and the higher order modal integral terms were neglected; TREETOPS has no direct way of including non-zero modal damping and requires the use of a COSMIC NASTRAN FEM model to include higher order modal integrals. The MTB simulation was configured to use Adams (3-1)/2 integrations with a step size of 0.002 seconds. The smaller step size (factor of 5 lower than TREETOPS step size) should give results comparable to the multi-pass Runge-Kutta scheme.

The results of the MTB and TREETOPS simulation runs will now be presented. Since both bodies are based on the same generic flexible body model, the presented data focuses on the target body and the relative motion data.

Figure 6.4-3 shows the time response of the "contact" forcing functions applied to the target docking node. These correspond to equations (6.4-1) and (6.4-2). Figure 6.4-4 contains both the MTB and TREETOPS results for the target body center of mass (CM) velocity vector expressed in the inertial frame. The corresponding curves are plotted with the same line style and the plot scale does not allow the observer to differentiate between comparable curves. Figure 6.4-5 shows the vector difference between the TREETOPS and the MTB velocity vectors for the target body. The remaining data will be presented in the format of Figures 6.4-4 and 6.4-5; that is, the comparable MTB and TREETOPS data (usually a 3 element vector) will be plotted on the same graph at the top of the page and the

vector difference (TREETOPS - MTB) will be shown in the next figure at the bottom of the page.

Figures 6.4-6 and 6.4-7 contain results for the target body CM position expressed in the inertial frame. Figures 6.4-8 and 6.4-9 show the target body angular velocity vector data. There is significant difference between the MTB and the TREETOPS results for the angular velocity about the body X-axis. The moment arm associated with the torque about the X-axis which produces this angular velocity arises solely from body flexing. The "contact" moment about the body y-axis displaces the target docking node in the negative Z direction. The "contact" force along the positive Y direction, in conjunction with the Z displacement, produces the torque about the X-axis. Investigations into the different results showed that the X-axis angular velocity is sensitive to modal damping; this follows since the moment arises solely from flexibility. In turn, the X-axis angular velocity is also sensitive to the method of integration; different numerical integration methods induce different levels of numerical damping. For example, repeating the MTB simulation run using rectangular integration produces an X-axis angular velocity value much closer to the TREETOPS result. The conclusion is that the difference in integration methods, combined with the effects of single versus double precision and the sensitivity of the X-axis angular velocity in this particular case, is the source of the differing results.

Figures 6.4-10 and 6.4-11 contain the generalized modal coordinate derivative data for the target body. Figures 6.4-12 and 6.4-13 show the generalized modal coordinate data for the target body. Figures 6.4-14 and 6.4-15 show the translational deformation data associated with the target docking node. This data is expressed in the target body frame and represents deformation with respect to the rigid body. Figures 6.4-16 and 6.4-17 contain data for the rotational deformation of the target docking node with respect to the undeformed position.

Figures 6.4-18 and 6.4-19 show the results for the relative translational velocity of the chase docking node with respect to the target docking node expressed in the inertial frame. Figures 6.4-20 and 6.4-21 contain data for the relative docking node position vector (D2 with respect to D1), expressed in the inertial frame. Notice in general that the difference between the MTB and TREETOPS results appears to be growing with increasing time. This can be attributed to the difference in numerical damping associated with the respective integration techniques and the effects of single versus double precision simulation variables.

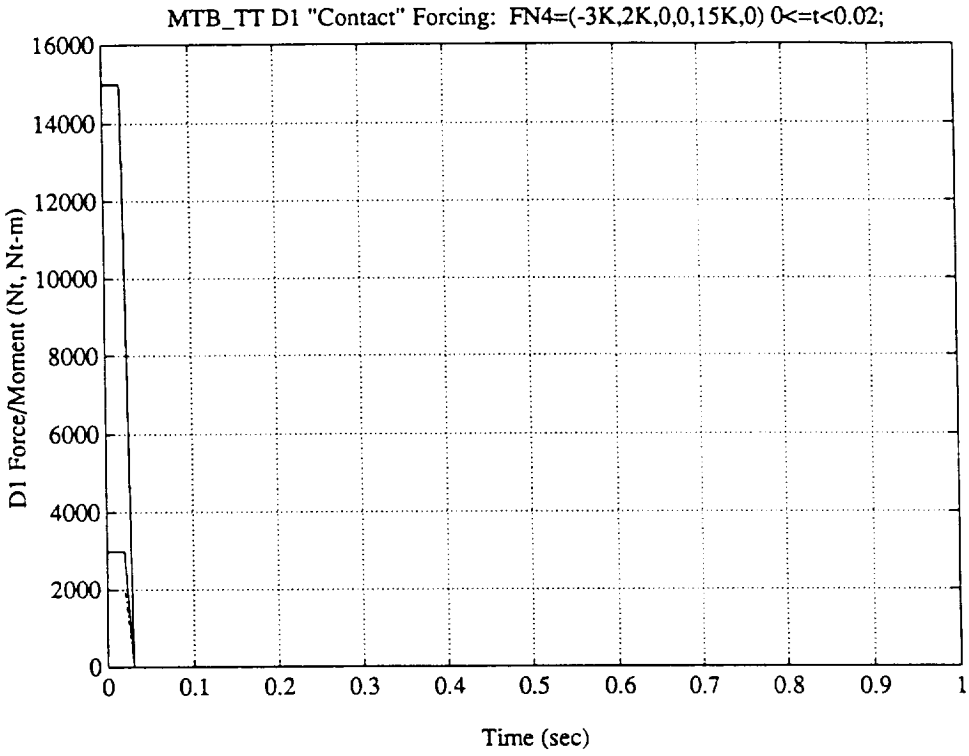


Figure 6.4-3 Target Docking Node "Contact" Forcing Functions.

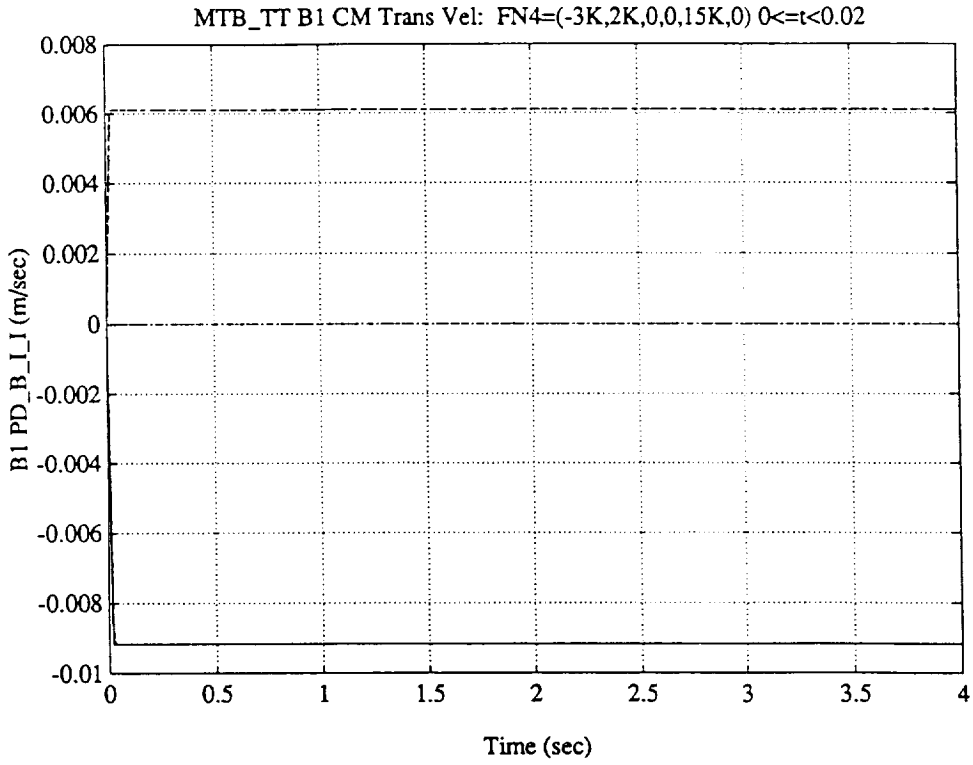


Figure 6.4-4 Target Body CM Translational Velocity.

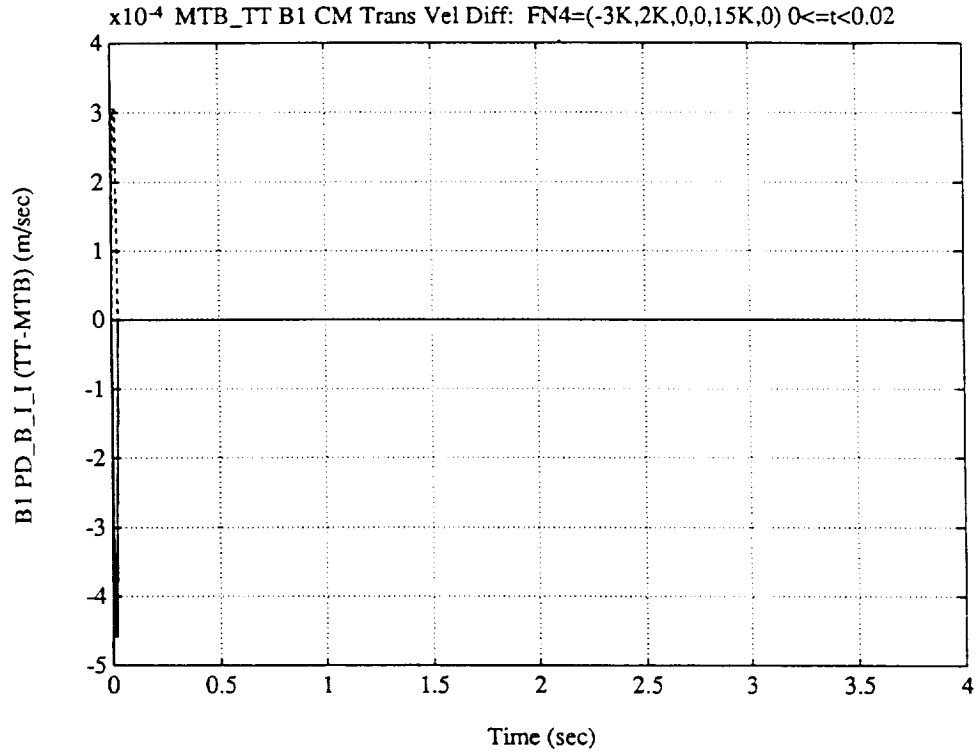


Figure 6.4-5 TREETOPS - MTB Target Body CM Translational Velocity.

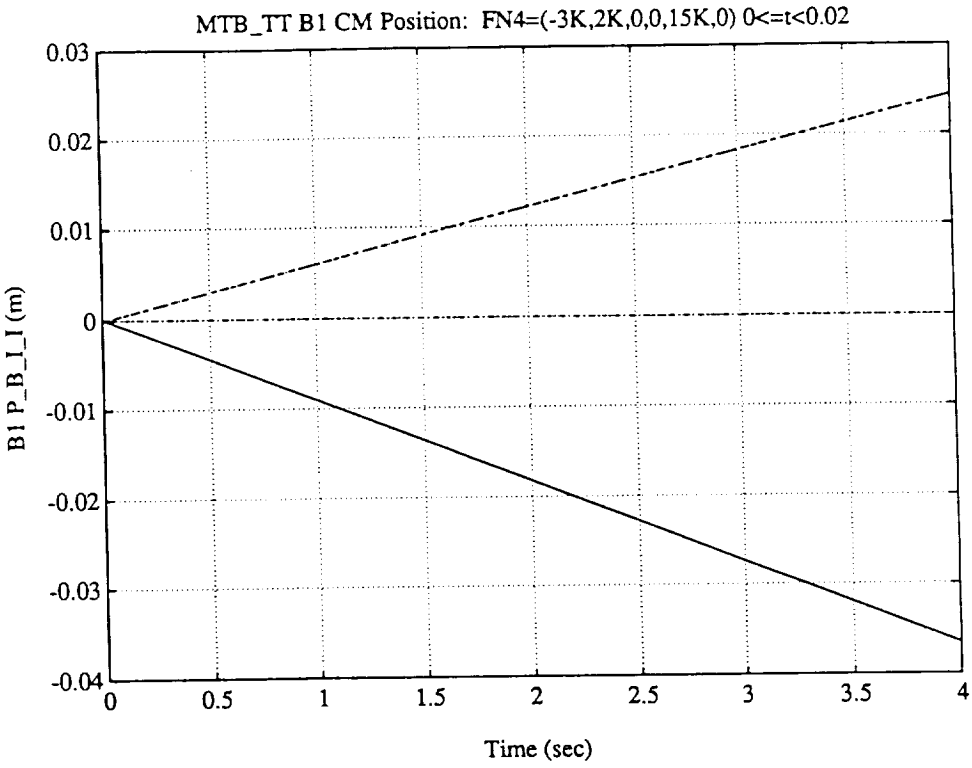


Figure 6.4-6 Target Body CM Position.

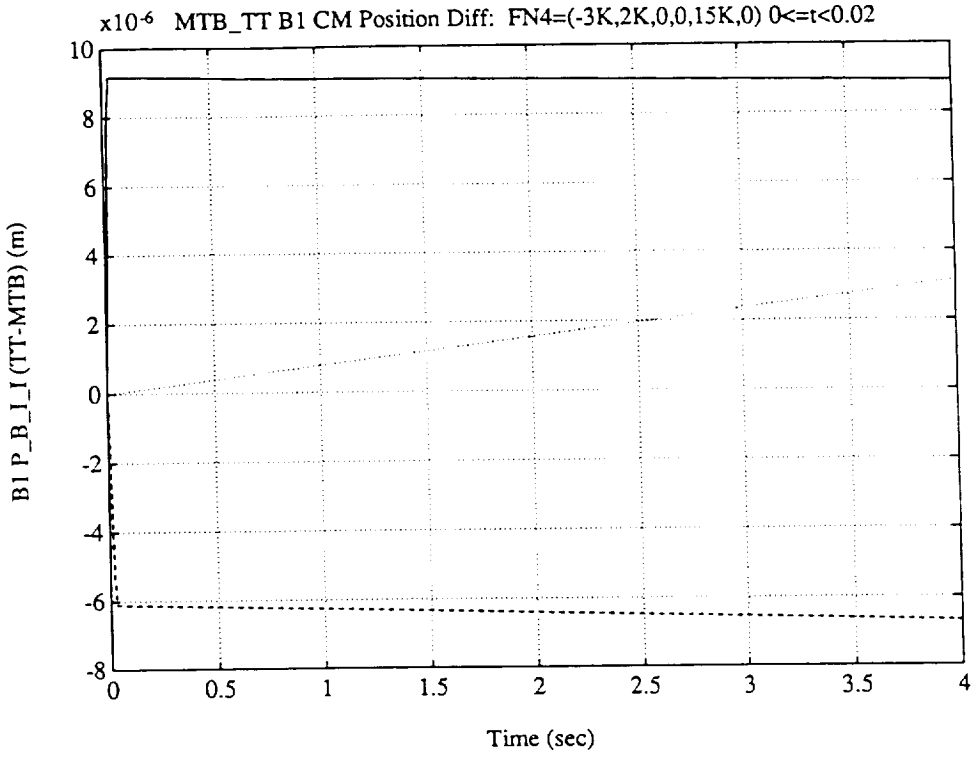


Figure 6.4-7 TREETOPS - MTB Target Body CM Position.

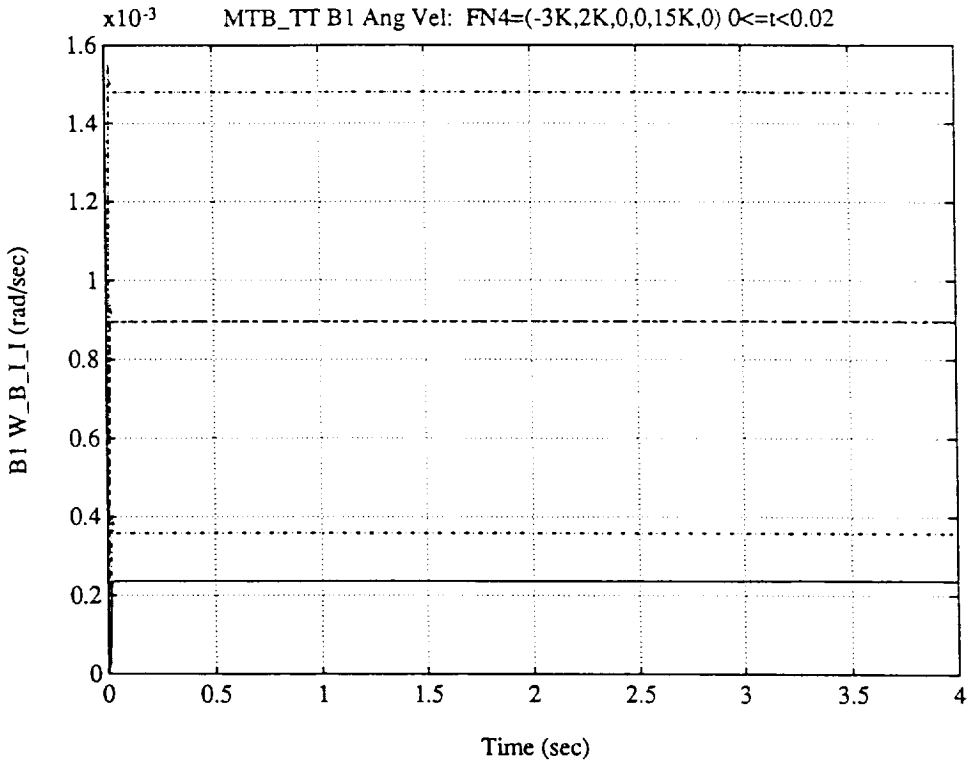


Figure 6.4-8 Target Body Angular Velocity.

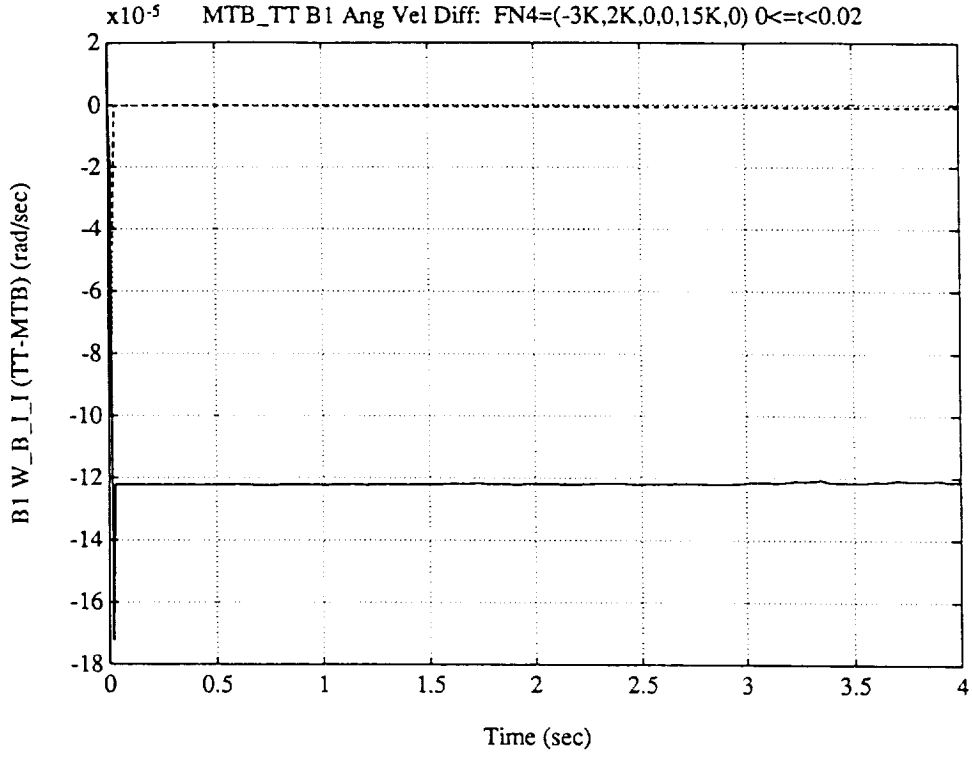


Figure 6.4-9 TREETOPS - MTB Target Body Angular Velocity.

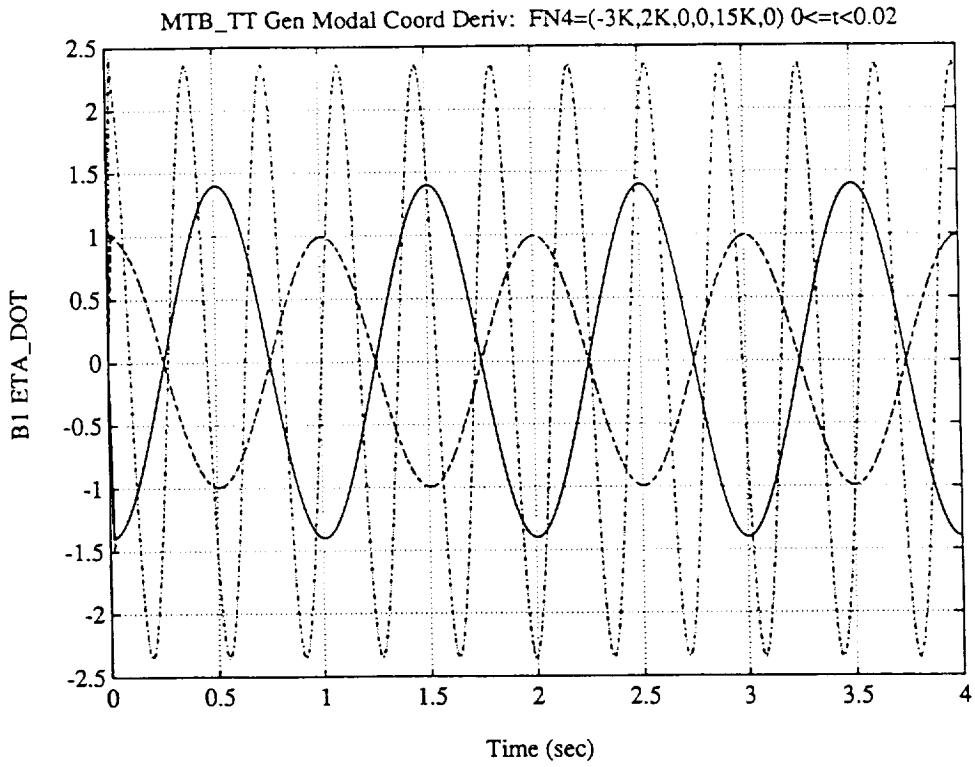


Figure 6.4-10 Target Body Generalized Modal Coordinate Derivatives.

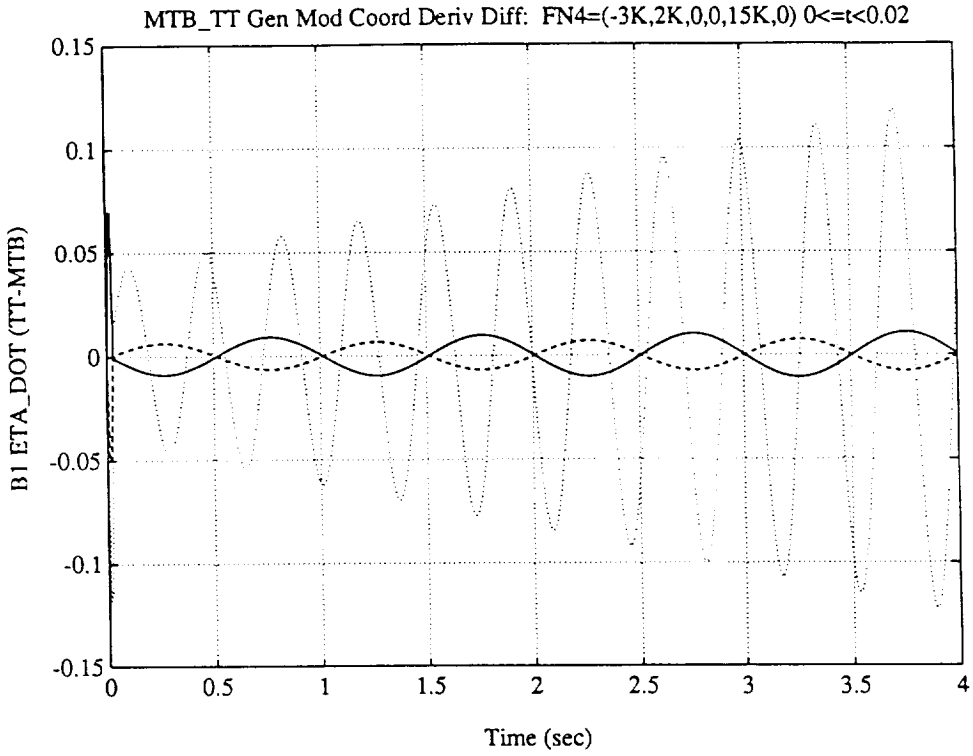


Figure 6.4-11 TREETOPS - MTB Target Body Generalized Modal Coordinate Derivatives.

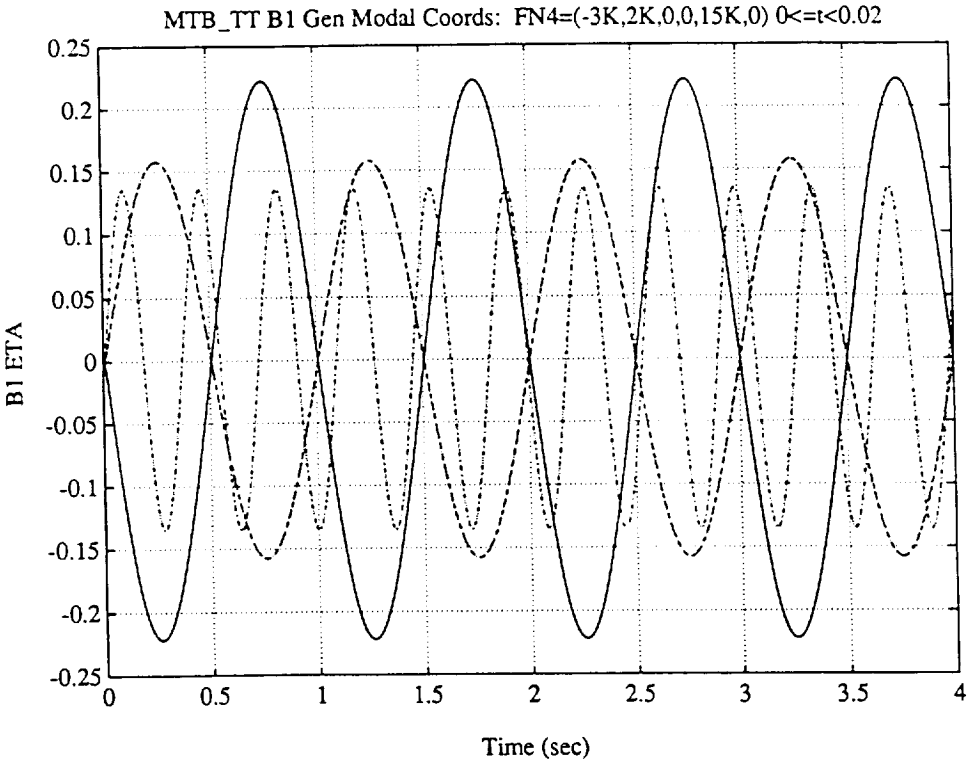


Figure 6.4-12 Target Body Generalized Modal Coordinates.

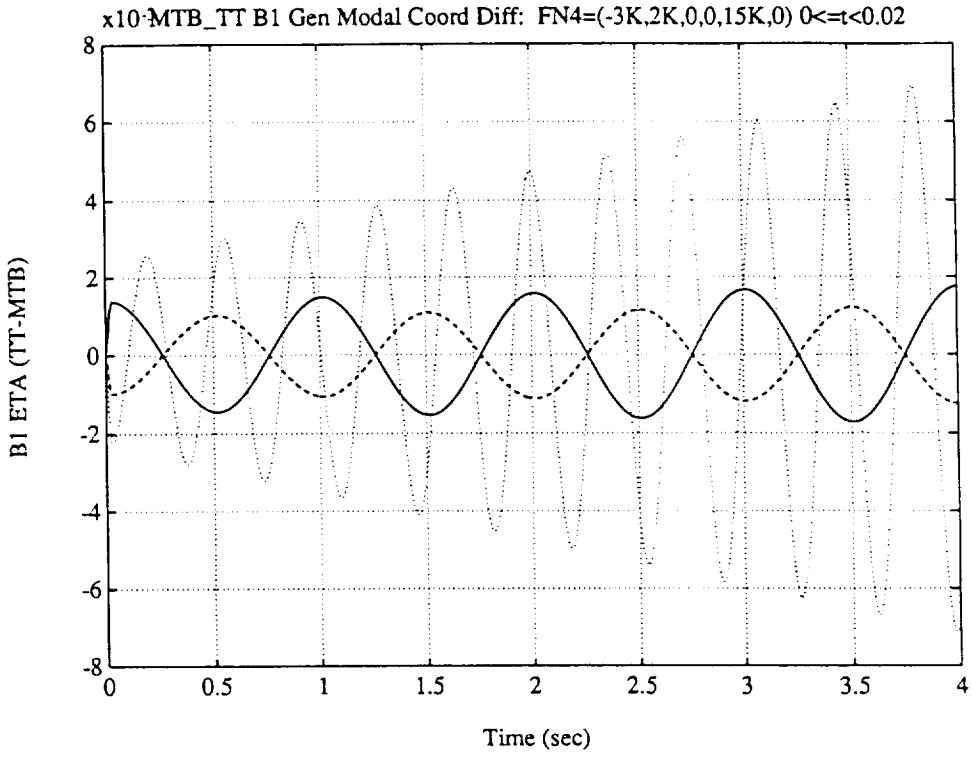


Figure 6.4-13 TREETOPS - MTB Target Body Generalized Modal Coordinates.

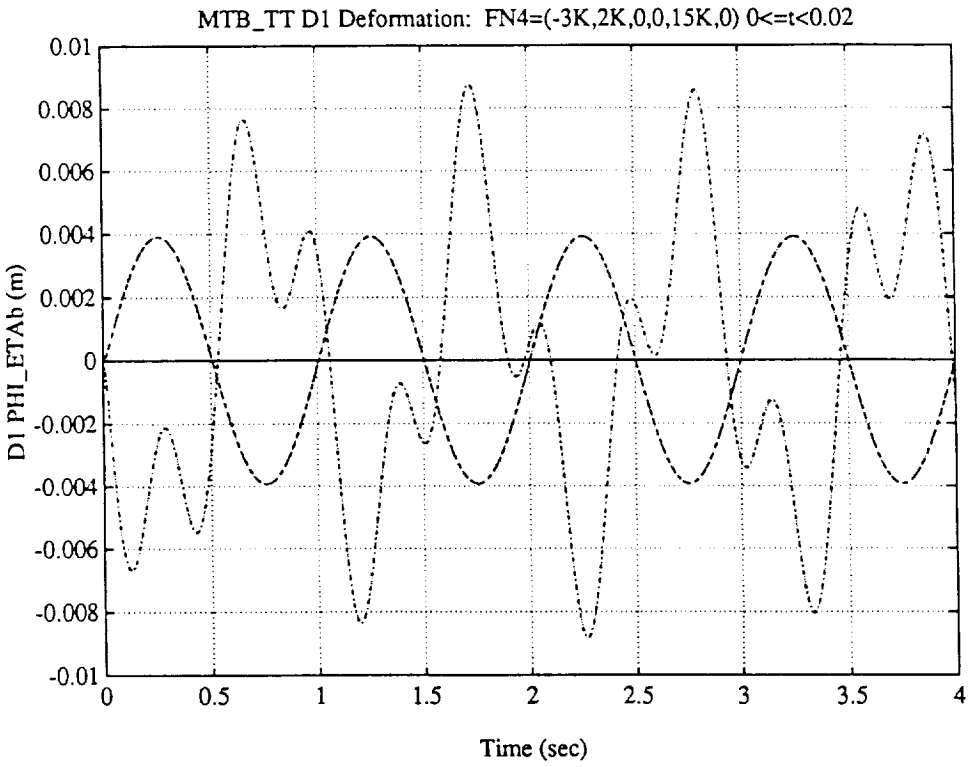


Figure 6.4-14 Target Body Docking Node Translational Deformation.

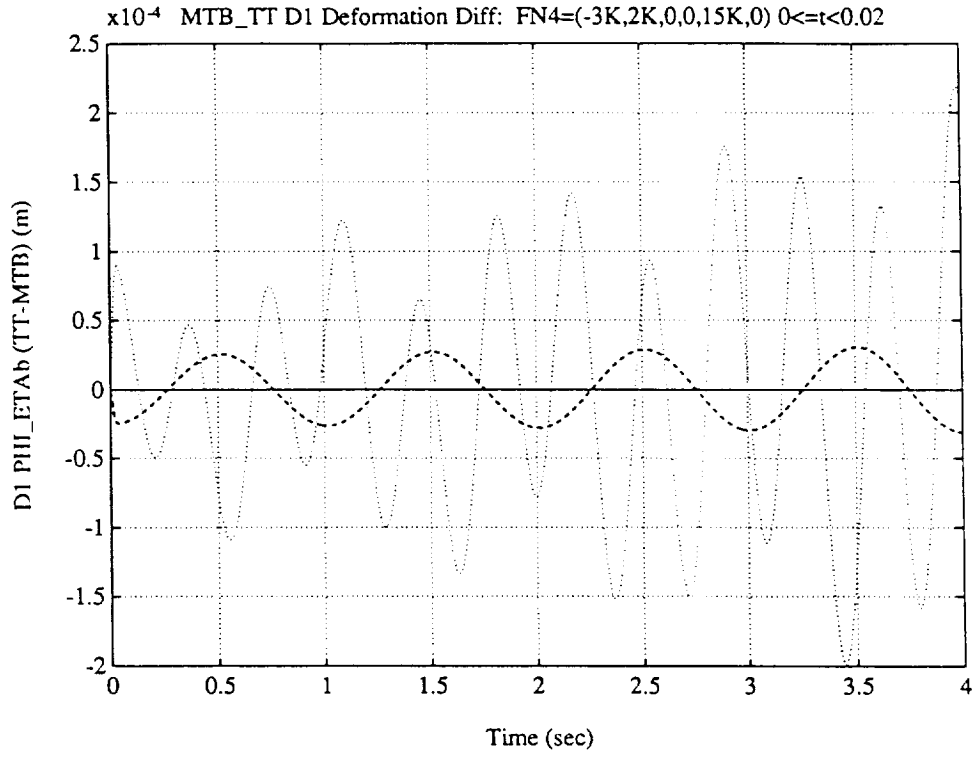


Figure 6.4-15 TREETOPS - MTB Target Body Docking Node Translational Deformation.

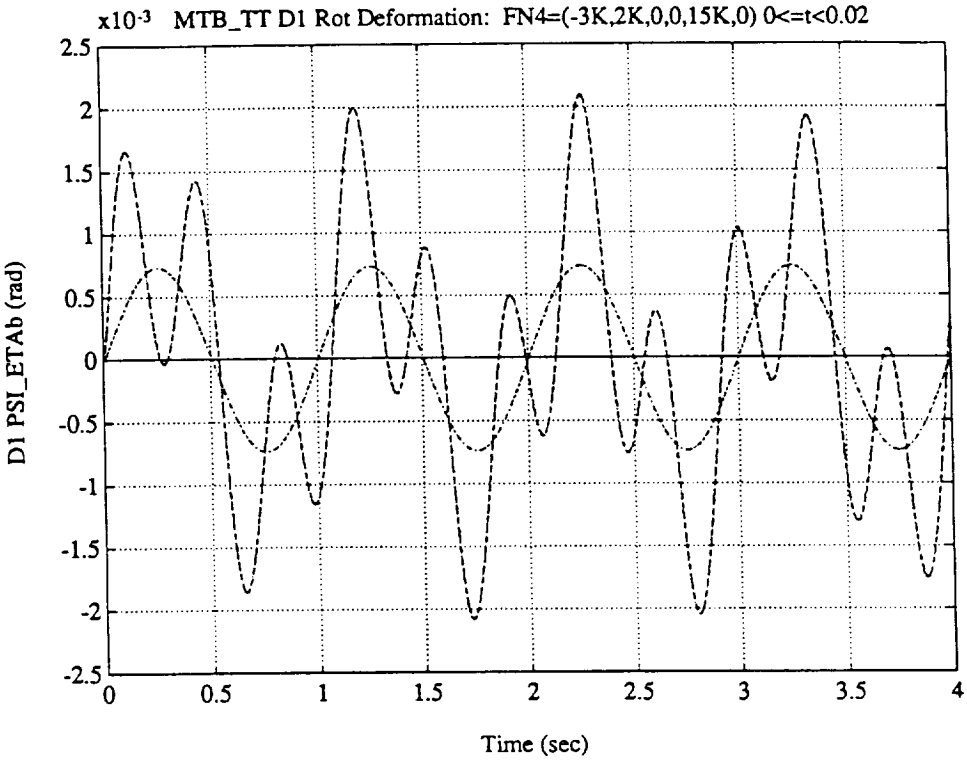


Figure 6.4-16 Target Body Docking Node Rotational Deformation.

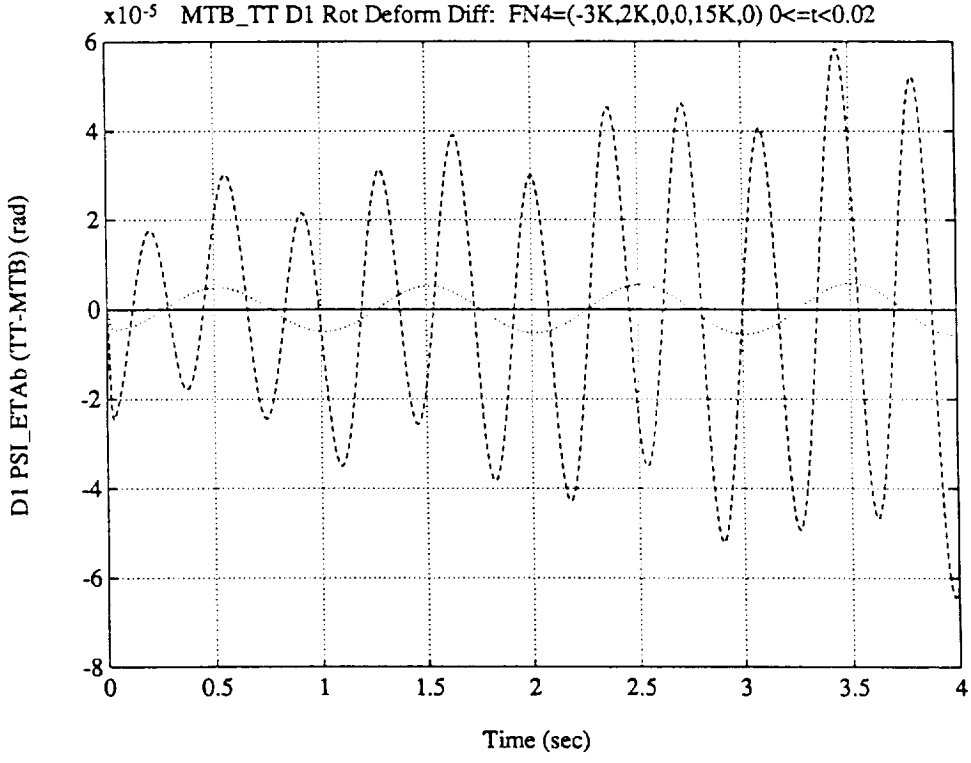


Figure 6.4-17 TREETOPS - MTB Target Body Docking Node Rotational Deformation.

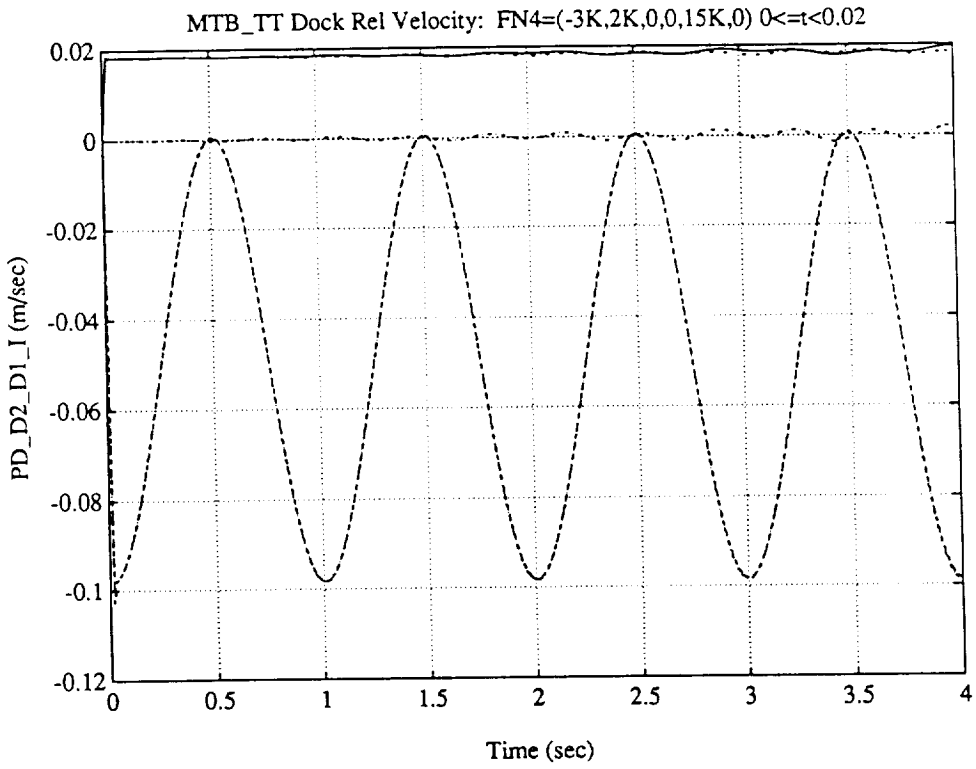


Figure 6.4-18 Docking Node Relative Translational Velocity.

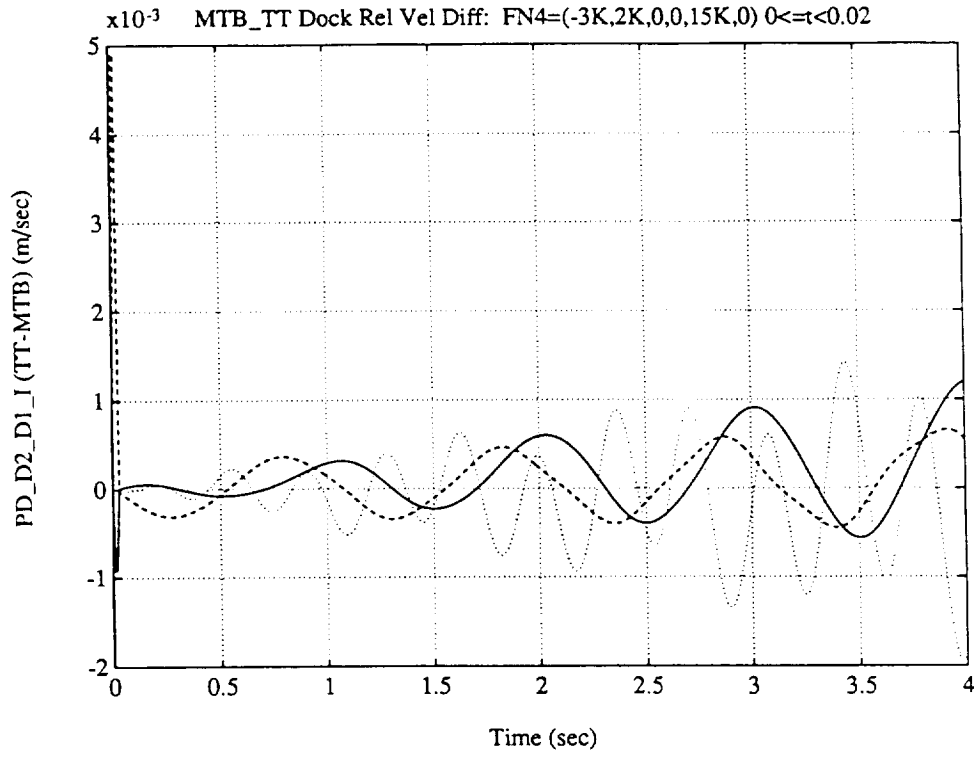


Figure 6.4-19 TREETOPS - MTB Docking Node Relative Translational Velocity.

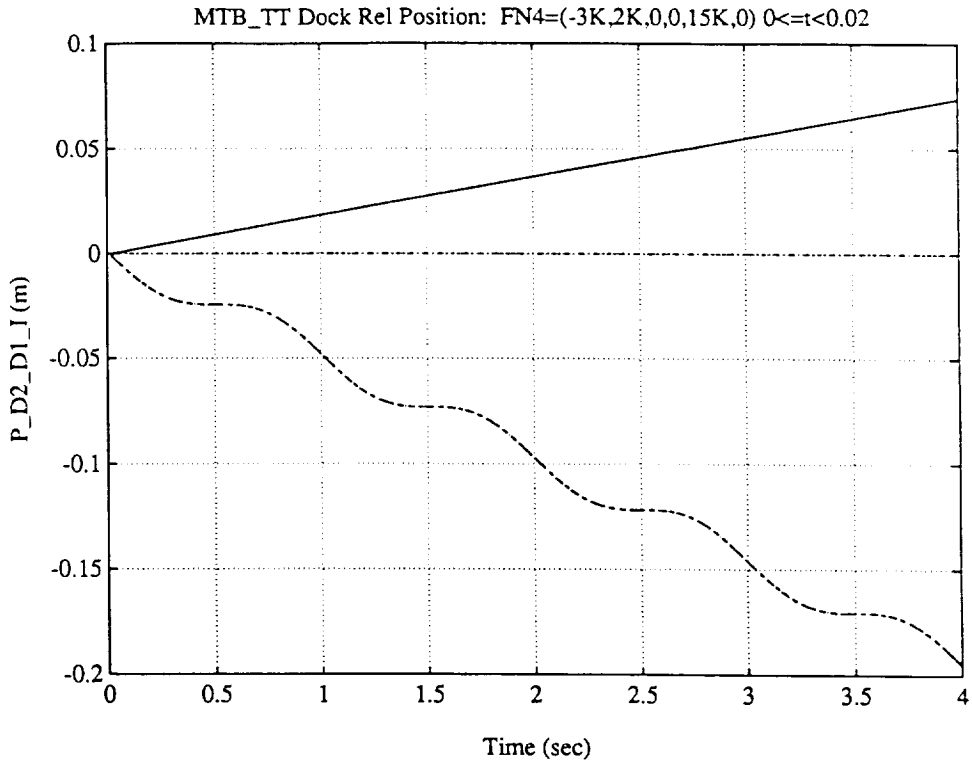


Figure 6.4-20 Docking Node Relative Position.

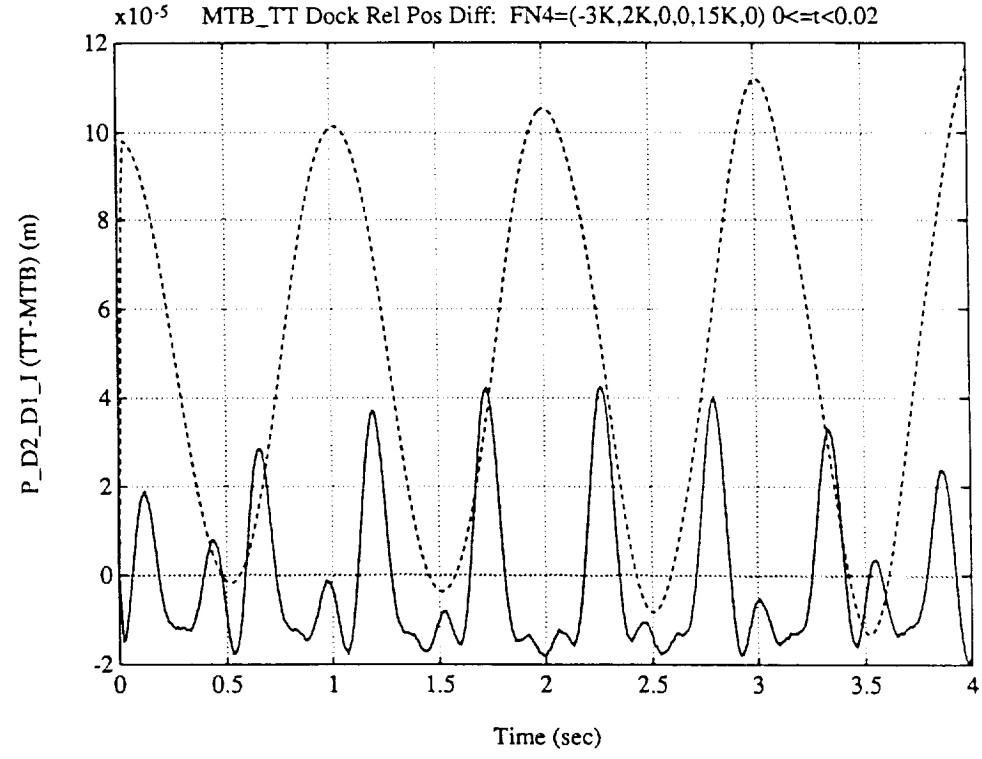


Figure 6.4-21 TREETOPS - MTB Docking Node Relative Position.

7.0 SUMMARY AND CONCLUSIONS

Per the contract objective, a generalized flexible body math model has been developed to further enhance the MTB Facility capabilities. The math model assumes that the flexible modes which characterize body bending will be orthonormal modes. This assumption allows the bending modes to be characterized by the standard second order equation

$$\ddot{\eta}_i + 2\,\zeta_i\,\omega_i\,\dot{\eta}_i + \omega_i^2\,\eta_i = F_i$$

as requested in the statement of work. Only minor modifications to the math model would be required to handle other less common mode types (e.g., non-orthogonal modes which typically result from sub-structure coupling techniques such as Craig-Bampton component mode synthesis).

The model was coded in FORTRAN 77 utilizing double precision floating point numbers. The math model code offers the user several levels of complexity for the equations of motion and three hardware-in-the-loop numerical integration options. Due to the increased volume of data over pure rigid body runs, a flexible body data pre-processing algorithm has been coded to generate simulation input data from FEM output data. The code has been installed and verified on the Alliant computer in the MTB facility.

In the future, the flexible body math model with be incorporated into the host simulation. The cycle times associated with various configurations (i.e., various EOM complexity levels) will need to be determined. Modifications to the flexible body math model may be required in order to reduce the simulation cycle time.

APPENDIX A

MATH MODEL SIMULATION CODE

ASCII Input File: MTBSIM.INP

```
0.002, 500, 5          Integ step size (sec); dt per CRT; dt per write
2.                     !Stop time
2                      !INT:  1=Euler (Adams1), 2=Adams3-1, 3=Adams23-16+5
```

ASCII Input File: BODY1.INP

```
6547.                  ! mass (c-u);  Free-Free Brass Beam Model
32.74, 0., 0.          ! inertia matrix (c-u); symmetric
0., 335000., 0.
0., 0., 335000.
0., 0., 0.             ! P_B_I_I (length c-u); SAME AS GSV(1-3)
0., 0., 0.             ! PD_B_I_B  (rate c-u); SAME AS QSV(1-3)
0., 0., 0.             ! W_B_I_B  (deg/sec); SAME AS QSV(4-6)
1, 2, 3                ! I <-- BODY:  Euler rot seq (1=x, 2=y, 3=z)
0., 0., 0.             ! I <-- BODY:  Euler angles (deg)
4, 4                   ! # nodes, NODE_DOCK; BODY origin at CM
-12.39, 0., 0.         ! node 1 in BODY
-4.13, 0., 0.          ! node 2 in BODY
4.13, 0., 0.           ! node 3 in BODY
2.39, 0., 0.           ! node 4
0., 0., 0.             ! P_D1_S1_S1
1., 0., 0.             ! T_B1_D1O
0., 1., 0.
0., 0., 1.
1., 0., 0.             ! T_D1_S1
0., 1., 0.
0., 0., 1.
.T.,.F.,.T.            ! EOM: 1)T=EHOMIT; 2)T=(1)+ERNLT 3)T=FREE-FREE
3                      ! # modes
6.2993, 6.2993, 17.4063   ! modal freqs (rad/sec)
0.0, 0.0, 0.0             ! modal damping (zeta_i)
0., 0., 0.                 modal coords:  eta(NM); GSV(>7)
0., 0., 0.                ! deriv modal coords:  etadot(NM); QSV(>6)
  1.000000128348988   0.000000000E+000  -6.931270607381940E-018  !Gen mass
  0.00000000E+000     1.000000128348988   0.0000000000000E+000
 -5.143184055728112E-017   0.0000000E+000    1.000000170771713
   .0000000E+00  .0000000E+00  .2485285E-01    ! Translational Modal Gains:  Phi
```

```
.0000000E+00  .2485285E-01  .0000000E+00
.0000000E+00  .0000000E+00  .2462543E-01
.0000000E+00  .0000000E+00 -.9216621E-02
.0000000E+00 -.9216621E-02  .0000000E+00
.0000000E+00  .0000000E+00 -.1633864E-01
.0000000E+00  .0000000E+00 -.9216621E-02
.0000000E+00 -.9216621E-02  .0000000E+00
.0000000E+00  .0000000E+00  .1633864E-01
.0000000E+00  .0000000E+00  .2485285E-01
.0000000E+00  .2485285E-01  .0000000E+00
.0000000E+00  .0000000E+00 -.2462543E-01
.0000000E+00  .4663153E-02  .0000000E+00   ! Rotational Modal Gains:  Psi
.0000000E+00  .0000000E+00 -.4663153E-02
.0000000E+00  .7881236E-02  .0000000E+00
.0000000E+00  .2742354E-02  .0000000E+00
.0000000E+00  .0000000E+00 -.2742354E-02
.0000000E+00 -.1159555E-02  .0000000E+00
.0000000E+00 -.2742354E-02  .0000000E+00
.0000000E+00  .0000000E+00  .2742354E-02
.0000000E+00 -.1159555E-02  .0000000E+00
.0000000E+00 -.4663153E-02  .0000000E+00
.0000000E+00  .0000000E+00  .4663153E-02
.0000000E+00  .7881236E-02  .0000000E+00
.0000000E+00  .0000000E+00 -.5443560E-05   ! Gamma_0
.0000000E+00 -.5443560E-05  .0000000E+00
.0000000E+00  .0000000E+00  .3552714E-14
.0000000E+00 -.3552714E-13  .0000000E+00   ! Gamma_1
.0000000E+00  .0000000E+00  .3552714E-13
.0000000E+00 -.2119259E-04  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00   ! Gamma_2jk
-.1000715E+01  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
 .1000715E+01  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
 .5551115E-16  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
-.5551115E-16  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00   ! I_1k
 .0000000E+00  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
 .0000000E+00  .0000000E+00  .0000000E+00
 .4752610E+00  .0000000E+00  .0000000E+00
 .1000715E+01  .0000000E+00  .0000000E+00   ! I_2jk
 .0000000E+00  .1000715E+01  .0000000E+00
```

```
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00 -.1000715E+01  .0000000E+00
.5551115E-16  .0000000E+00  .0000000E+00
.0000000E+00  .5551115E-16  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00  .0000000E+00 -.1000715E+01
.0000000E+00  .0000000E+00  .0000000E+00
.1000715E+01  .0000000E+00  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00  .0000000E+00  .1000715E+01
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00  .0000000E+00 -.5551115E-16
.0000000E+00  .0000000E+00  .0000000E+00
-.2775558E-16  .0000000E+00  .0000000E+00
.0000000E+00 -.2775558E-16  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00
.0000000E+00  .2775558E-16  .0000000E+00
.1013732E+01  .0000000E+00  .0000000E+00
.0000000E+00  .1013732E+01  .0000000E+00
.0000000E+00  .0000000E+00  .0000000E+00
```

ASCII Input File: BODY2.INP

Same structure as BODY1.INP omitting P_D1_S1_S1 and T_D1_S1 data since the F/M sensor is located only on the target body.

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc Filename:        BODY.INC
cc
cc Function:        Include file for variable/common declarations
cc                  relating to body data.
cc
cc Source:          JC     Date: 2/91
cc
cc Comments:        Double precision
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      INTEGER MNM                          !MAX # OF MODES PER BODY
      INTEGER MNN                          !MAX # OF NODES PER BODY
      INTEGER NBODES                       !# OF BODIES
      PARAMETER (MNM = 10, MNN = 10, NBODES = 2)
      INTEGER NODE_DOCK(NBODES)  !BODY DOCKING PORT NODE NUM
      INTEGER NMODES(NBODES)     !# FLEX MODES PER BODY
      INTEGER NNODES(NBODES)     !# NODES PER BODY
      INTEGER INDX(NBODES,6+MNM) !REC OF LUDCMP ROW PERMUTATIONS

cc    Simulation options vars
      LOGICAL*2 EOM(NBODES,3)    !OPTIONS FOR EOM COMPLEXITY
      INTEGER INT                !INTEGRATION OPTIONS
      REAL*8  TIME               !SIM TIME (SEC)
      REAL*8  DT                 !INTEGRATION STEP SIZE (SEC)

cc    Mass properties/modal data vars
      REAL*8 MASS(NBODES)                  !BODY MASS
      REAL*8 ICM(NBODES,3,3)               !INERTIA MAT ABOUT BODY CM
      REAL*8 GMASS(NBODES,MNM,MNM)         !GEN MASS MATRIX PER BODY
      REAL*8 NODES(NBODES,MNN,3)           !NODE LOCATIONS PER BODY IN
                                           !       BODY

      REAL*8 MOD_FREQ(NBODES,MNM)          !MODAL FREQUENCIES PER BODY
      REAL*8 MOD_ZETA(NBODES,MNM)          !MODAL DAMPING PER BODY
      REAL*8 PHI(NBODES,MNN,MNM,3)         !TRANS MODAL GAINS
      REAL*8 PSI(NBODES,MNN,MNM,3)         !ROT MODAL GAINS

      REAL*8  PHIETA(NBODES,MNN,3)         !SUM OF PHI*ETA PER NODE
      REAL*8  PHIETAD(NBODES,MNN,3)        !SUM OF PHI*ETAD PER NODE
      REAL*8  PSIETA(NBODES,MNN,3)         !SUM OF PSI*ETA PER NODE
      REAL*8  PSIETAD(NBODES,MNN,3)        !SUM OF PSI*ETAD PER NODE

cc    Mass Integral Terms
      REAL*8 GAM_0(NBODES,3,MNM)           !ARR OF GAM_0 MASS INTEGS
      REAL*8 GAM_1(NBODES,3,MNM)           !ARR OF GAM_1 MASS INTEGS
      REAL*8 GAM_2JK(NBODES,MNM,MNM,3) !ARR OF GAM_2JK MASS INTS
```

```
      REAL*8 I_1K(NBODES,MNM,3,3)          !ARR OF 3X3 I_1K MASS INTS
      REAL*8 I_2JK(NBODES,MNM,MNM,3,3)     !ARR OF 3X3 I_2JK MASS INTS
cc
      REAL*8 LU_MASSM(NBODES,6+MNM,6+MNM)        !BODY #NB LUDCMP
                                                 !     MASS MAT

cc External body forces/moments
      REAL*8  FN_BODY(NBODES,MNN,3)        !FORCES APPLIED AT EACH NODE
      REAL*8  MN_BODY(NBODES,MNN,3)        !MOMENTS APPL AT EACH NODE

cc Quasi-coord (with angular body rates) data
      REAL*8  QSV(NBODES,6+MNM)    !       QUASI COORD VECTOR
      REAL*8  QSVD(NBODES,6+MNM)           !QUASI VECTOR DERIV
      REAL*8  QSVDO_1(NBODES,6+MNM)        !1 OLD QUASI VECTOR DERIV
      REAL*8  QSVDO_2(NBODES,6+MNM)        !2 OLD QUASI VECTOR DERIV

cc Generalized coord (quaternion for orientation) data
      REAL*8  GSV(NBODES,7+MNM)            !GEN COORD VECTOR
      REAL*8  GSVD(NBODES,7+MNM)           !GEN VECTOR DERIV
      REAL*8  GSVDO_1(NBODES,7+MNM)        !1 OLD GEN VECTOR DERIV
      REAL*8  GSVDO_2(NBODES,7+MNM) !      2 OLD GEN VECTOR DERIV

ccccccccccccccccccccccc BODY #1 - TARGET ccccccccccccccccccccc
cc Position/rate vectors
      REAL*8  P_D1O_B1_B1(3)    !UNDEFORM D1 POS VECT WRT B1 IN B1 (L)
      REAL*8  P_D1_S1_S1(3)     !ASSUMED CONST POS D1 WRT S1 IN S1 (L)

cc Transformations/quaternions
      REAL*8  T_B1_D1O(3,3)     !UNDEFORMED D1 <-- B1 TRANS
      REAL*8  T_D1_S1(3,3)      !ASSUMED CONST TRANS D1 <--S1

ccccccccccccccccccccccc BODY #2 - TARGET ccccccccccccccccccccc
cc Position/rate vectors
      REAL*8  P_D2O_B2_B2(3)    !UNDEFORM D2 POS VECT WRT B2 IN B2 (L)

cc Transformations/quaternions
      REAL*8  T_B2_D2O(3,3)     !UNDEFORMED D2 <-- B2 TRANS

ccccccccccccccccccccccccccc COMMONS ccccccccccccccccccccccccccc
cc Simulation options
      COMMON / OPTIONS / EOM, INT

cc Simulation control related commons
      COMMON / SIMCON  / TIME, DT

cc Mass properties related commons
      COMMON / RIGID   / MASS, ICM, NNODES, NODES
      COMMON / FLEX    / GMASS
      COMMON / MASSINT / GAM_0, GAM_1, GAM_2JK, I_1K, I_2JK
```

```
      COMMON / LHS    / LU_MASSM, INDX

cc  Modal data related commons
      COMMON / MODES  / NMODES, NODE_DOCK, PHI, PSI,
     &                  MOD_FREQ, MOD_ZETA, PHIETA, PHIETAD,
     &                  PSIETA, PSIETAD

cc  External body forces/moments per body node commons
      COMMON / NODEFT / FN_BODY, MN_BODY

cc  Coord vector related commons
      COMMON / QSTATE / QSV, QSVD, QSVDO_1, QSVDO_2
      COMMON / GSTATE / GSV, GSVD, GSVDO_1, GSVDO_2

cc  Body #1 related commons
      COMMON / VECT1  / P_D1O_B1_B1, P_D1_S1_S1
      COMMON / TRANS1 / T_B1_D1O, T_D1_S1

cc  Body #2 related commons
      COMMON / VECT2  / P_D2O_B2_B2
      COMMON / TRANS2 / T_B2_D2O
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc Filename:        INTFAC.INC
cc
cc Function:        Include file for variable/common declarations
cc                  of variables necessary to interface to NASA's
cc                   code.
cc
cc Source:          JC     Date: 3/91
cc
cc Comments:        Double precision
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      REAL*8 T_B1_I(3,3)            !UPDATED IN INTFAC
      REAL*8 T_B2_I(3,3)            !UPDATED IN INTFAC

      REAL*8 T_D1_D2(3,3)           !UPDATED IN INTFAC
      REAL*8 P_D2_D1_D1(3)          !UPDATED IN INTFAC
      REAL*8 PD_D2_D1_D1(3)         UPDATED IN INTFAC
      REAL*8 W_D2_D1_D2(3)          !UPDATED IN INTFAC
      REAL*8 T_B1_B2(3,3)           !UPDATED IN INTFAC
      REAL*8 T_B2_B1(3,3)           UPDATED IN INTFAC
      REAL*8 P_B2_B1_B1(3)          !UPDATED IN INTFAC
      REAL*8 T_D1_B1(3,3)           !UPDATED IN INTFAC
      REAL*8 T_D2_B2(3,3)           !UPDATED IN INTFAC
      REAL*8 P_D1_B1_B1(3)          !UPDATED IN INTFAC
      REAL*8 P_D2_B2_B2(3)          !UPDATED IN INTFAC
      REAL*8 FS1_S1(3)              !FROM BODY #1 F/M SENSOR
      REAL*8 MS1_S1(3)              !FROM BODY #1 F/M SENSOR

      COMMON / BODY_I   / T_B1_I, T_B2_I
      COMMON / REL_DOCK  / T_D1_D2, P_D2_D1_D1, PD_D2_D1_D1,
     &              W_D2_D1_D2
      COMMON / BODY_BODY / T_B1_B2, T_B2_B1, P_B2_B1_B1
      COMMON / BODY_DOCK / T_D1_B1, T_D2_B2, P_D1_B1_B1,
     &              P_D2_B2_B2
      COMMON / FMSENSOR  / FS1_S1, MS1_S1
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      PROGRAM TESTMTBF
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc  Function:        Test driver for checkout of the NAMTB flex
cc                   body simulation; formatted MATLAB output.
cc
cc  Source:          JC     Date: 3/91
cc
cc  Comments:        Double precision
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      IMPLICIT NONE

cc  Include files
      INCLUDE 'BODY.INC'
      INCLUDE 'INTFAC.INC'

cc  Locals
      INTEGER  I, NB
      INTEGER  NCRT, CRT, NWRIT, WRIT
      INTEGER  SEQ(3)
      REAL*8  TSTOP
      REAL*8  TEMP3_1(3), TEMP3_2(3), TEMP3_3(3)
      REAL*8  TEMP3_4(3), TEMP3_5(3), W_B_I_I(3), EA(3)
      REAL*8  TEMP4_1(4), TEMP4_2(4)
      REAL*8  T_I_B1(3,3), DCM(3,3), T_I_D1(3,3), T_B1_D1(3,3)
      REAL*8  T_I_B2(3,3)

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc  Initialization
      DATA  TIME, NCRT / 0.0, 0 /
      DATA SEQ / 1, 2, 3 /

cc  First pass sim initialization
      CALL DYNAMIC

      OPEN(UNIT=13,FILE='MTBSIM.INP',FORM='FORMATTED',STATUS='OLD')

      OPEN(UNIT=41,FILE='MTB1.MAT',FORM='FORMATTED')
      OPEN(UNIT=42,FILE='MTB2.MAT',FORM='FORMATTED')

      WRITE(6,*)
      READ(13,*)  DT, CRT, WRIT
      NWRIT = WRIT
      NCRT = CRT
      READ(13,*)  TSTOP
      READ(13,*)  INT
```

```
    CLOSE(13)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

cc  Top of Main Loop
10   CONTINUE

cc  Ouput to CRT
    IF( NCRT .EQ. CRT) THEN
     NCRT = 0
     WRITE(6,*) 'SIM TIME (sec):  ',TIME
     WRITE(6,*) 'T_B1_I: ',(T_B1_I(1,I),I=1,3)
     WRITE(6,*) 'T_B1_I: ',(T_B1_I(2,I),I=1,3)
     WRITE(6,*) 'T_B1_I: ',(T_B1_I(3,I),I=1,3)
     WRITE(6,*)
     WRITE(6,*) 'T_B2_I: ',(T_B2_I(1,I),I=1,3)
     WRITE(6,*) 'T_B2_I: ',(T_B2_I(2,I),I=1,3)
     WRITE(6,*) 'T_B2_I: ',(T_B2_I(3,I),I=1,3)
     WRITE(6,*)
    ENDIF
    NCRT = NCRT + 1

cc  Unformatted data output to *.mat files
    IF(NWRIT .EQ. WRIT) THEN
     NWRIT = 0

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
cc  PROCESS OUTPUT DATA
        DO 50 I=1,3
         TEMP4_1(I) = GSV(1,3+I)
         TEMP4_2(I) = GSV(2,3+I)
         TEMP3_3(I) = QSV(1,3+I)
         TEMP3_4(I) = QSV(2,3+I)
50      CONTINUE
        TEMP4_1(4) = GSV(1,7)
        TEMP4_2(4) = GSV(2,7)

        CALL Q_T_DCM(TEMP4_1, DCM)
        CALL DCM_T_EA(DCM, SEQ, EA)

        CALL D_MTRANS(T_B1_I, T_I_B1, 3, 3)
        CALL D_MMUL(T_I_B1, TEMP3_3, W_B_I_I, 3, 3, 1)

cc  TREETOPS comparable data for BRASS3_2 (two body "contact") run
         CALL D_MTRANS(T_D1_B1, T_B1_D1, 3, 3)
         CALL D_MTRANS(T_B1_I, T_I_B1, 3, 3)
         CALL D_MMUL(T_I_B1, T_B1_D1, T_I_D1, 3, 3, 3)
         CALL D_MMUL(T_I_D1, P_D2_D1_D1, TEMP3_1, 3, 3, 1)
         CALL D_MMUL(T_I_D1, PD_D2_D1_D1, TEMP3_2, 3, 3, 1)
```

```
      NB=1
      WRITE(41,444) TIME, (QSV(NB,I),I=1,6+NMODES(NB)),
     &     (GSV(NB,I),I=1,3), (EA(I),I=1,3), (GSV(NB,I),
     &     I=8,7+NMODES(NB)), (PHIETA(NB,4,I),I=1,3),
     &     (PHIETAD(NB,4,I),I=1,3), (PSIETA(NB,4,I),I=1,3),
     &     (PSIETAD(NB,4,I),I=1,3), (TEMP3_1(I),I=1,3),
     &     (TEMP3_2(I),I=1,3), (GSVD(NB,I),I=1,3),
     &     (W_B_I_I(I),I=1,3)

cc  Body 2
      CALL Q_T_DCM(TEMP4_1, DCM)
      CALL DCM_T_EA(DCM, SEQ, EA)

      CALL D_MTRANS(T_B2_I, T_I_B2, 3, 3)
      CALL D_MMUL(T_I_B2, TEMP3_4, W_B_I_I, 3, 3, 1)

      NB = 2
      WRITE(42,444) TIME, (QSV(NB,I),I=1,6+NMODES(NB)),
     &     (GSV(NB,I),I=1,3), (EA(I),I=1,3), (GSV(NB,I),
     &     I=8,7+NMODES(NB)), (PHIETA(NB,4,I),I=1,3),
     &     (PHIETAD(NB,4,I),I=1,3), (PSIETA(NB,4,I),I=1,3),
     &     (PSIETAD(NB,4,I),I=1,3),
     &     (GSVD(NB,I),I=1,3),
     &     (W_B_I_I(I),I=1,3)

    ENDIF
    NWRIT = NWRIT + 1

444  FORMAT(2X,50E15.7)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    CALL DYNAMIC

cc  Adjust step size on final step
    IF( (TIME + DT) .GT. TSTOP .AND. (TIME + DT) .LT.
   &   (TSTOP + DT/2) ) THEN
     DT = TSTOP - TIME
     NWRIT = WRIT
     NCRT = CRT
    ENDIF

cc  Increment sim time and check for end
    TIME = TIME + DT
    IF(TIME .LE. TSTOP) GOTO 10

    CLOSE(41)
    CLOSE(42)

cc  Create ouput data label files
     open(unit=41, file='mtb1.lab')
```

```
write(41,500) 1,'Time    '
write(41,500) 2,'PD_B_I_B '
write(41,500) 3,'PD_B_I_B '
write(41,500) 4,'PD_B_I_B '
write(41,500) 5,'W_B_I_B  '
write(41,500) 6,'W_B_I_B  '
write(41,500) 7,'W_B_I_B  '
write(41,500) 8,'ETA_D(1) '
write(41,500) 9,'ETA_D(2) '
write(41,500) 10,'ETA_D(3) '
write(41,500) 11,'P_B_I_I  '
write(41,500) 12,'P_B_I_I  '
write(41,500) 13,'P_B_I_I  '
write(41,500) 14,'B_I EA(1)'
write(41,500) 15,'B_I EA(2)'
write(41,500) 16,'B_I EA(3)'
write(41,500) 17,'ETA(1)   '
write(41,500) 18,'ETA(2)   '
write(41,500) 19,'ETA(3)   '
write(41,500) 20,'PHIETA4x '
write(41,500) 21,'PHIETA4y '
write(41,500) 22,'PHIETA4z '
write(41,500) 23,'PHIETAD4x'
write(41,500) 24,'PHIETAD4y'
write(41,500) 25,'PHIETAD4z'
write(41,500) 26,'PSIETA4x '
write(41,500) 27,'PSIETA4y '
write(41,500) 28,'PSIETA4z '
write(41,500) 29,'PSIETAD4x'
write(41,500) 30,'PSIETAD4y'
write(41,500) 31,'PSIETAD4z'
write(41,500) 32,'P_D2_D1_I'
write(41,500) 33,'P_D2_D1_I'
write(41,500) 34,'P_D2_D1_I'
write(41,500) 35,'PD_D2_D1i'
write(41,500) 36,'PD_D2_D1i'
write(41,500) 37,'PD_D2_D1i'
write(41,500) 38,'PD_B_I_I '
write(41,500) 39,'PD_B_I_I '
write(41,500) 40,'PD_B_I_I '
write(41,500) 41,'W_B_I_I  '
write(41,500) 42,'W_B_I_I  '
write(41,500) 43,'W_B_I_I  '

open(unit=42, file='mtb2.lab')

write(42,500) 1,'Time    '
write(42,500) 2,'PD_B_I_B '
```

```
      write(42,500) 3,'PD_B_I_B '
      write(42,500) 4,'PD_B_I_B '
      write(42,500) 5,'W_B_I_B  '
      write(42,500) 6,'W_B_I_B  '
      write(42,500) 7,'W_B_I_B  '
      write(42,500) 8,'ETA_D(1) '
      write(42,500) 9,'ETA_D(2) '
      write(42,500) 10,'ETA_D(3) '
      write(42,500) 11,'P_B_I_I  '
      write(42,500) 12,'P_B_I_I  '
      write(42,500) 13,'P_B_I_I  '
      write(42,500) 14,'B_I EA(1)'
      write(42,500) 15,'B_I EA(2)'
      write(42,500) 16,'B_I EA(3)'
      write(42,500) 17,'ETA(1)   '
      write(42,500) 18,'ETA(2)   '
      write(42,500) 19,'ETA(3)   '
      write(42,500) 20,'PHIETA4x '
      write(42,500) 21,'PHIETA4y '
      write(42,500) 22,'PHIETA4z '
      write(42,500) 23,'PHIETAD4x'
      write(42,500) 24,'PHIETAD4y'
      write(42,500) 25,'PHIETAD4z'
      write(42,500) 26,'PSIETA4x '
      write(42,500) 27,'PSIETA4y '
      write(42,500) 28,'PSIETA4z '
      write(42,500) 29,'PSIETAD4x'
      write(42,500) 30,'PSIETAD4y'
      write(42,500) 31,'PSIETAD4z'
      write(42,500) 32,'PD_B_I_I '
      write(42,500) 33,'PD_B_I_I '
      write(42,500) 34,'PD_B_I_I '
      write(42,500) 35,'W_B_I_I  '
      write(42,500) 36,'W_B_I_I  '
      write(42,500) 37,'W_B_I_I  '

500   format(4x,i3,2x,a9)

      CLOSE(42)
      CLOSE(42)

      STOP
      END
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE DYNAMIC
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc  Function:          Driver of flexible body contact dynamics model
cc                     for the 6-DOF docking/berthing facility.
cc
cc  Source:  JC        Date:  2-3/91
cc
cc  Comments:          Double precision
cc                     Data via common block declarations
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      IMPLICIT NONE

cc  Include files
      INCLUDE 'BODY.INC'

cc  Locals
      INTEGER  I
      LOGICAL  PASS              ! SUBROUTINE PASS FLAG
      DATA  PASS / .FALSE. /

cc  Initialization
      IF(.NOT. PASS) THEN
        PASS = .TRUE.
        CALL INIT
        CALL INTFAC
        CALL FMTRANS
        CALL CONTROL(1)
c       CALL CONTROL(2)
        CALL LUMASSM(1)
        CALL LUMASSM(2)
        CALL PLANT(1)
        CALL PLANT(2)
        RETURN
      ENDIF

ccccccccccccccccccc  Main Loop  ccccccccccccccccccccccccc

cc  Compute contact forces/torques based on sensor output
      CALL FMTRANS

cc  Compute control system effects
      CALL CONTROL(1)
c     CALL CONTROL(2)

cc  Compute time varying mass matrix for Target and Chase if nec
      IF(.NOT. EOM(1,1) .OR. .NOT. EOM(1,3)) THEN
```

```
      CALL LUMASSM(1)
    ENDIF
    IF(.NOT. EOM(2,1) .OR. .NOT. EOM(2,3)) THEN
      CALL LUMASSM(2)
    ENDIF

cc  Compute right hand side of EOMs for bodies 1 and 2
    CALL PLANT(1)
    CALL PLANT(2)

cc  Integrate EOMs and handle quasi-to-generalized coord.
cc   transformations
    CALL INTEG(1)
    CALL INTEG(2)

cc  Compute body relative data
    CALL INTFAC

    RETURN
    END
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE INIT
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc Function:          Initialization of mass properties, geometry,
cc                    state vectors, etc.
cc
cc Source:  JC     Date: 2/91
cc
cc Comments:          Double precision
cc                    Data input via reads from ASCII input files
cc                    Data distributed via common block declarations
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      IMPLICIT NONE

cc  Include files
      INCLUDE 'BODY.INC'
      INCLUDE 'INTFAC.INC'

cc  Local declarations
      INTEGER  I, J, K, L
      INTEGER  NB, ROT_SEQ(3)
      REAL*8  PI, RTD
      REAL*8  ROT_ANG(3)
      REAL*8  TEMP1_33(3,3), TEMP2_33(3,3), TEMP3_33(3,3), DCM(3,3)
      REAL*8  Q4(4)
      PARAMETER ( PI = 3.141592654d0 )
      PARAMETER ( RTD = 180.d0/PI )

cc  Open pertinent data files
      OPEN(UNIT=11,FILE='BODY1.INP',FORM='FORMATTED',STATUS='OLD')
      OPEN(UNIT=12,FILE='BODY2.INP',FORM='FORMATTED',STATUS='OLD')
      OPEN(UNIT=20,FILE='INIT.OUT',FORM='FORMATTED',STATUS='UNKNOWN')

cc  Read in body #1 (target) data
      NB = 1

      WRITE(20,*)
      WRITE(20,'(10X,"BODY #",I2," INITIALIZATION DATA")') NB
      WRITE(20,*)

      READ(11,*) MASS(NB)              !MASS
      WRITE(20,'(2X,"MASS: ",F12.4)')  MASS(NB)

      READ(11,*) (ICM(NB,1,J), J=1,3)    !INERTIA MATRIX CM
      READ(11,*) (ICM(NB,2,J), J=1,3)
      READ(11,*) (ICM(NB,3,J), J=1,3)
      WRITE(20,'(2X,"ICM: ",3F12.4)')  (ICM(NB,1,J),J=1,3)
```

```
      WRITE(20,'(2X,"ICM: ",3F12.4)')  (ICM(NB,2,J),J=1,3)
      WRITE(20,'(2X,"ICM: ",3F12.4)')  (ICM(NB,3,J),J=1,3)
      WRITE(20,*)

      READ(11,*) (GSV(NB,J),J=1,3)     !BODY CM POS WRT I IN I
      READ(11,*) (QSV(NB,J),J=1,3)     !BODY TRANS RATE WRT I IN BODY
      READ(11,*) (QSV(NB,J),J=4,6)     !BODY ANG RATE WRT I IN BODY

      READ(11,*) (ROT_SEQ(J), J=1,3)   !BODY <-- I ROT SEQUENCE
      READ(11,*) (ROT_ANG(J), J=1,3)   !BODY <-- I ROT ANGLES (DEG)

cc  Compute trans from Inertial (I) to Body 1 (target) from
cc   Euler angle sequence (I <-- Body)
      DO 3 J=1,3
        QSV(NB,3+J) = QSV(NB,3+J) / RTD
        ROT_ANG(J) = ROT_ANG(J) / RTD
3     CONTINUE

      CALL D_ROT(ROT_SEQ(1), ROT_ANG(1), TEMP1_33)
      CALL D_ROT(ROT_SEQ(2), ROT_ANG(2), TEMP2_33)
      CALL D_ROT(ROT_SEQ(3), ROT_ANG(3), TEMP3_33)

      CALL D_MMUL(TEMP1_33, TEMP2_33, DCM, 3, 3, 3)
      CALL D_MMUL(DCM, TEMP3_33, TEMP1_33, 3, 3, 3)
      CALL D_MTRANS(TEMP1_33, T_B1_I, 3, 3)

cc  Compute BODY <-- I quaternion and put in generalized state vect
      CALL DCM_T_Q(T_B1_I, Q4)
      CALL D_QNORM(Q4)

      WRITE(20,*) ' T_B1_I: ',(T_B1_I(1,I),I=1,3)
      WRITE(20,*) ' T_B1_I: ',(T_B1_I(2,I),I=1,3)
      WRITE(20,*) ' T_B1_I: ',(T_B1_I(3,I),I=1,3)
      WRITE(20,*)

      DO 5 J=1,4
        GSV(NB,3+J) = Q4(J)
5     CONTINUE
      WRITE(20,'(3X,"ANG RATES CONVERTED TO (RAD/S) IN SVs")')
      WRITE(20,'(2X,"QSV(1->6): ",6F12.5)')  (QSV(NB,J),J=1,6)
      WRITE(20,*)
      WRITE(20,'(2X,"GSV(1->7): ",7F12.5)')  (GSV(NB,J),J=1,7)
      WRITE(20,*)

cc  Body node data
      READ(11,*) NNODES(NB), NODE_DOCK(NB)
      WRITE(20,'(2X,"NNODES: ",I3,8X,"NODE_DOCK: ",I3)')
     &          NNODES(NB), NODE_DOCK(NB)
```

```
      DO 10 I=1,NNODES(NB)
        READ(11,*) (NODES(NB,I,J),J=1,3)   !BODY NB NODES
        WRITE(20,'(2X,"NODE ",I2,": ",10F12.5)')
     &       I, (NODES(NB,I,J),J=1,3)
10    CONTINUE
      WRITE(20,*)

cc    Sensor/docking port location/orientation initialization
      DO 15 I=1,3
        P_D1O_B1_B1(I) = NODES(NB,NODE_DOCK(NB),I)
15    CONTINUE

      READ(11,*) (P_D1_S1_S1(I),I=1,3)

      READ(11,*) (T_B1_D1O(1,I),I=1,3)
      READ(11,*) (T_B1_D1O(2,I),I=1,3)
      READ(11,*) (T_B1_D1O(3,I),I=1,3)

      READ(11,*) (T_D1_S1(1,I),I=1,3)
      READ(11,*) (T_D1_S1(2,I),I=1,3)
      READ(11,*) (T_D1_S1(3,I),I=1,3)

cc    Equations of motion complexity options
      READ(11,*) (EOM(NB,I),I=1,3)       !LOGICAL EOM OPTIONS
      WRITE(20,'(2X,"EOM FLAGS (EHOMIT; ERNLT; FF MODES): ",3L6)')
     &           (EOM(NB,I),I=1,3)

cc    EOM options:  ERNLT implies EHOMIT
      IF( EOM(NB,2) ) EOM(NB,1) = .TRUE.

cc    Flex data
      READ(11,*) NMODES(NB)
      WRITE(20,'(2X,"NMODES: ",I3)') NMODES(NB)

      IF(NMODES(NB) .EQ. 0)  GOTO 200

cc    Modal frequencies (rad/sec)
      READ(11,*) (MOD_FREQ(NB,I),I=1,NMODES(NB))
      WRITE(20,'(2X,"MOD_FREQ: ",10F12.4)') (MOD_FREQ(NB,I),
     &       I=1,NMODES(NB))

cc    Modal damping
      READ(11,*) (MOD_ZETA(NB,I),I=1,NMODES(NB))
      WRITE(20,'(2X,"MOD_ZETA: ",10F12.4)') (MOD_ZETA(NB,I),
     &       I=1,NMODES(NB))

cc    Init modal coords (eta) and modal coord derivs (etad)
      READ(11,*) (GSV(NB,7+I),I=1,NMODES(NB))
      READ(11,*) (QSV(NB,6+I),I=1,NMODES(NB))
```

```
      WRITE(20,'(2X,"ETA: ",10F12.4)') (GSV(NB,7+I),I=1,NMODES(NB))
      WRITE(20,'(2X,"ETAD: ",10F12.4)') (QSV(NB,6+I),I=1,NMODES(NB))
      WRITE(20,*)

cc  Generalized mass matrix
      DO 25 I=1,NMODES(NB)
       READ(11,*) (GMASS(NB,I,J),J=1,NMODES(NB)) !GEN MASS MAT
25    CONTINUE

cc  Translation modal gains
      DO 35 J=1,NNODES(NB)
       DO 30 K=1,NMODES(NB)
        READ(11,*) (PHI(NB,J,K,I),I=1,3)
30     CONTINUE
35    CONTINUE

cc  Rotation modal gains
      DO 45 J=1,NNODES(NB)
       DO 40 K=1,NMODES(NB)
        READ(11,*) (PSI(NB,J,K,I),I=1,3)
40     CONTINUE
45    CONTINUE

cc  Standard mass integral terms
      DO 50 J=1,NMODES(NB)
       READ(11,*) (GAM_0(NB,I,J), I=1,3) !GAM_0 MASS INTEGS
50    CONTINUE

      DO 60 J=1,NMODES(NB)
       READ(11,*) (GAM_1(NB,I,J), I=1,3) !GAM_1 MASS INTEGS
60    CONTINUE

cc  Zero standard MITs for free-free mode option
      IF( EOM(NB,3) ) THEN
       DO 65 J=1,NMODES(NB)
        DO 66 I=1,3
         GAM_0(NB,I,J) = 0.d0
         GAM_1(NB,I,J) = 0.d0
66      CONTINUE
65     CONTINUE
      ENDIF

cc  Higher order mass integral terms
      DO 70 I=1,NMODES(NB)
       DO 80 J=1,NMODES(NB)
        READ(11,*) (GAM_2JK(NB,I,J,K),K=1,3) !GAM_2JK MASS INTEGS
80     CONTINUE
70    CONTINUE
```

```
      DO 90 J=1,NMODES(NB)
       DO 100 K=1,3
        READ(11,*) (I_1K(NB,J,K,L),L=1,3)    !I_1K MASS INTEGS
100     CONTINUE
90    CONTINUE

      DO 110 I=1,NMODES(NB)
       DO 120 J=1,NMODES(NB)
        DO 130 K=1,3
         READ(11,*) (I_2JK(NB,I,J,K,L),L=1,3)  !I_1K MASS INTEGS
130      CONTINUE
120     CONTINUE
110   CONTINUE

200   CONTINUE

cc  Read in body #2 (CHASE) data
      NB = 2

      WRITE(20,*)
      WRITE(20,*)
      WRITE(20,'(10X,"BODY #",I2," INITIALIZATION DATA")') NB
      WRITE(20,*)

      READ(12,*) MASS(NB)                !MASS
      WRITE(20,'(2X,"MASS: ",F12.4)')  MASS(NB)

      READ(12,*) (ICM(NB,1,J), J=1,3)    !INERTIA MATRIX CM
      READ(12,*) (ICM(NB,2,J), J=1,3)
      READ(12,*) (ICM(NB,3,J), J=1,3)
      WRITE(20,'(2X,"ICM: ",3F12.4)')  (ICM(NB,1,J),J=1,3)
      WRITE(20,'(2X,"ICM: ",3F12.4)')  (ICM(NB,2,J),J=1,3)
      WRITE(20,'(2X,"ICM: ",3F12.4)')  (ICM(NB,3,J),J=1,3)
      WRITE(20,*)

      READ(12,*) (GSV(NB,J),J=1,3)     !BODY CM POS WRT I IN I
      READ(12,*) (QSV(NB,J),J=1,3)     !BODY TRANS RATE WRT I IN BODY
      READ(12,*) (QSV(NB,J),J=4,6)     !BODY ANG RATE WRT I IN BODY
      READ(12,*) (ROT_SEQ(J), J=1,3)   !BODY <-- I ROT SEQUENCE
      READ(12,*) (ROT_ANG(J), J=1,3)   !BODY <-- I ROT ANGLES (DEG)

cc  Compute trans from Inertial (I) to Body 2 (CHASE) from
cc   Euler angle sequence (I <-- Body)
      DO 203 J=1,3
       QSV(NB,3+J) = QSV(NB,3+J) / RTD
       ROT_ANG(J) = ROT_ANG(J) / RTD
203   CONTINUE
      CALL D_ROT(ROT_SEQ(1), ROT_ANG(1), TEMP1_33)
      CALL D_ROT(ROT_SEQ(2), ROT_ANG(2), TEMP2_33)
```

```
      CALL D_ROT(ROT_SEQ(3), ROT_ANG(3), TEMP3_33)

      CALL D_MMUL(TEMP1_33, TEMP2_33, DCM, 3, 3, 3)
      CALL D_MMUL(DCM, TEMP3_33, TEMP1_33, 3, 3, 3)
      CALL D_MTRANS(TEMP1_33, T_B2_I, 3, 3)

cc    Compute BODY <-- I quaternion and put in generalized state vect
      CALL DCM_T_Q(T_B2_I, Q4)
      CALL D_QNORM(Q4)

      WRITE(20,*) ' T_B2_I: ',(T_B2_I(1,I),I=1,3)
      WRITE(20,*) ' T_B2_I: ',(T_B2_I(2,I),I=1,3)
      WRITE(20,*) ' T_B2_I: ',(T_B2_I(3,I),I=1,3)
      WRITE(20,*)

      DO 205 J=1,4
        GSV(NB,3+J) = Q4(J)
205   CONTINUE
      WRITE(20,'(3X,"ANG RATES CONVERTED TO (RAD/S) IN SVs")')
      WRITE(20,'(2X,"QSV(1->6): ",6F12.5)')  (QSV(NB,J),J=1,6)
      WRITE(20,*)
      WRITE(20,'(2X,"GSV(1->7): ",7F12.5)')  (GSV(NB,J),J=1,7)
      WRITE(20,*)

cc    Body node data
      READ(12,*) NNODES(NB), NODE_DOCK(NB)
      WRITE(20,'(2X,"NNODES: ",I3,8X,"NODE_DOCK: ",I3)')
     &          NNODES(NB), NODE_DOCK(NB)

      DO 210 I=1,NNODES(NB)
        READ(12,*) (NODES(NB,I,J),J=1,3)   !BODY NB NODES
        WRITE(20,'(2X,"NODE ",I2,": ",10F12.5)')
     &        I, (NODES(NB,I,J),J=1,3)
210   CONTINUE
      WRITE(20,*)

cc    Sensor/docking port location/orientation initialization
      DO 215 I=1,3
        P_D2O_B2_B2(I) = NODES(NB,NODE_DOCK(NB),I)
215   CONTINUE

      READ(12,*) (T_B2_D2O(1,I),I=1,3)
      READ(12,*) (T_B2_D2O(2,I),I=1,3)
      READ(12,*) (T_B2_D2O(3,I),I=1,3)

cc    Equations of motion complexity options
      READ(12,*) (EOM(NB,I),I=1,3)        !LOGICAL EOM OPTIONS
      WRITE(20,'(2X,"EOM FLAGS (EHOMIT; ERNLT; FF MODES): ",3L6)')
     &          (EOM(NB,I),I=1,3)
```

```
cc  EOM options:  ERNLT implies EHOMIT
      IF( EOM(NB,2) ) EOM(NB,1) = .TRUE.

cc  Flex data
      READ(12,*) NMODES(NB)
      WRITE(20,'(2X,"NMODES: ",I3)') NMODES(NB)

      IF(NMODES(NB) .EQ. 0)  GOTO 400

cc  Modal frequencies (rad/sec)
      READ(12,*) (MOD_FREQ(NB,I),I=1,NMODES(NB))
      WRITE(20,'(2X,"MOD_FREQ: ",10F12.4)') (MOD_FREQ(NB,I),
     &         I=1,NMODES(NB))

cc  Modal damping
      READ(12,*) (MOD_ZETA(NB,I),I=1,NMODES(NB))
      WRITE(20,'(2X,"MOD_ZETA: ",10F12.4)') (MOD_ZETA(NB,I),
     &         I=1,NMODES(NB))

cc  Init modal coords (eta) and modal coord derivs (etad)
      READ(12,*) (GSV(NB,7+I),I=1,NMODES(NB))
      READ(12,*) (QSV(NB,6+I),I=1,NMODES(NB))
      WRITE(20,'(2X,"ETA: ",10F12.4)') (GSV(NB,7+I),I=1,NMODES(NB))
      WRITE(20,'(2X,"ETAD: ",10F12.4)') (QSV(NB,6+I),I=1,NMODES(NB))
      WRITE(20,*)

cc  Generalized mass matrix
      DO 245 I=1,NMODES(NB)
        READ(12,*) (GMASS(NB,I,J),J=1,NMODES(NB)) !GEN MASS MAT
245   CONTINUE

cc  Translation modal gains
      DO 225 J=1,NNODES(NB)
        DO 230 K=1,NMODES(NB)
          READ(12,*) (PHI(NB,J,K,I),I=1,3)
230     CONTINUE
225   CONTINUE

cc  Rotation modal gains
      DO 235 J=1,NNODES(NB)
        DO 240 K=1,NMODES(NB)
          READ(12,*) (PSI(NB,J,K,I),I=1,3)
240     CONTINUE
235   CONTINUE

cc  Standard mass integral terms
      DO 250 J=1,NMODES(NB)
        READ(12,*) (GAM_0(NB,I,J), I=1,3) !GAM_0 MASS INTEGS
```

```
250  CONTINUE

     DO 260 J=1,NMODES(NB)
      READ(12,*) (GAM_1(NB,I,J), I=1,3) !GAM_1 MASS INTEGS
260  CONTINUE

cc  Zero standard MITs for free-free mode option
     IF( EOM(NB,3) ) THEN
      DO 265 J=1,NMODES(NB)
       DO 266 I=1,3
        GAM_0(NB,I,J) = 0.d0
        GAM_1(NB,I,J) = 0.d0
266     CONTINUE
265    CONTINUE
     ENDIF

cc  Higher order mass integral terms
     DO 270 I=1,NMODES(NB)
      DO 280 J=1,NMODES(NB)
       READ(12,*) (GAM_2JK(NB,I,J,K),K=1,3) !GAM_2JK MASS INTEGS
280    CONTINUE
270  CONTINUE

     DO 290 J=1,NMODES(NB)
      DO 300 K=1,3
       READ(12,*) (I_1K(NB,J,K,L),L=1,3)    !I_1K MASS INTEGS
300    CONTINUE
290  CONTINUE

     DO 310 I=1,NMODES(NB)
      DO 320 J=1,NMODES(NB)
       DO 330 K=1,3
        READ(12,*) (I_2JK(NB,I,J,K,L),L=1,3)  !I_1K MASS INTEGS
330     CONTINUE
320    CONTINUE
310  CONTINUE

400  CONTINUE

     CLOSE(11)
     CLOSE(12)
     CLOSE(20)

     RETURN
     END
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE FMTRANS
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc  Function:       Computes contact forces/torques at docking port
cc                  nodes in respective body frames based on output
cc                  of body #1 (target) force/moment sensor.
cc
cc  Source:         JC     Date: 3/91
cc
cc  Comments:       Double precision
cc                  Data via common block declarations
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      IMPLICIT NONE

cc  Include files
      INCLUDE 'BODY.INC'
      INCLUDE 'INTFAC.INC'

cc  Locals
      INTEGER  I, J, K
      REAL*8  TEMP1_3(3), TEMP2_3(3)
      REAL*8  FS1_B1(3), FS1_B2(3)
      REAL*8  MD1_S1(3), MD2_S1(3), MD1_B1(3), MD2_B2(3)
      REAL*8  T_B1_S1(3,3), T_B2_S1(3,3)
      REAL*8  P_D2_S1_S1(3)
      REAL*8  F_D1(3), M_D1(3), T_S1_D1(3,3)

ccccccccc   "Contact" Test Case     ccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc  Compute sensed F/M based on comparable TREETOPS inputs
      IF(TIME .LT. 0.0199) THEN
         F_D1(1) = -3000.d0
         F_D1(2) =  2000.D0
         F_D1(3) =  0.D0
         M_D1(1) =  0.D0
         M_D1(2) = 15000.D0
         M_D1(3) =  0.D0

c        FN_BODY(1,4,1) = -3000.D0
c        FN_BODY(1,4,2) = 2000.D0
c        FN_BODY(1,4,3) = 0.D0
c        MN_BODY(1,4,1) = 0.D0
c        MN_BODY(1,4,2) = 15000.D0
c        MN_BODY(1,4,3) = 0.D0

         CALL D_MTRANS(T_D1_S1, T_S1_D1, 3, 3)
         CALL D_MXV(T_S1_D1, F_D1, FS1_S1, 3, 3)
```

```
      CALL D_MXV(T_S1_D1, M_D1, MS1_S1, 3, 3)
      CALL D_VCROSS(P_D1_S1_S1, FS1_S1, TEMP1_3)
      DO I=1,3
        MS1_S1(I) = MS1_S1(I) + TEMP1_3(I)
      ENDDO

    ELSE
      DO I=1,3
        FS1_S1(I) = 0.D0
        MS1_S1(I) = 0.D0
      ENDDO
    ENDIF
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

cc  Transform sensed forces to respective body fixed sytems
cc  Assume rigid connection from sensor to docking port
cc   (i.e., T_D1_S1 remains constant)
    DO 4 I=1,3
      DO 5 K=1,3
        T_B1_S1(I,K) = 0.D0
5     CONTINUE
      DO 6 J=1,3
        T_B1_S1(I,J) = T_B1_S1(I,J) + T_D1_B1(J,I) * T_D1_S1(J,I)
6     CONTINUE
4   CONTINUE

    CALL D_MMUL(T_B2_B1, T_B1_S1, T_B2_S1, 3, 3, 3)
    CALL D_MXV(T_B1_S1, FS1_S1, FS1_B1, 3, 3)
    CALL D_MXV(T_B2_S1, FS1_S1, FS1_B2, 3, 3)

cc  Equal and opposite contact forces; all contact points sensed
cc   by force/moment sensor on body #1
    DO 10 I=1,3
      FN_BODY(1,NODE_DOCK(1),I) = FS1_B1(I)
      FN_BODY(2,NODE_DOCK(2),I) = -FS1_B2(I)
10   CONTINUE

cc  Compute vector from S1 to D1 and D2
    DO 15 I=1,3
      TEMP1_3(I) = 0.D0
      DO 16 J=1,3
        TEMP1_3(I) = TEMP1_3(I) + T_D1_S1(J,I) * P_D2_D1_D1(J)
16    CONTINUE
15  CONTINUE

    DO 20 I=1,3
      P_D2_S1_S1(I) = P_D1_S1_S1(I) + TEMP1_3(I)
20  CONTINUE
```

```
cc  Compute contact moments (at D1 and D2) in respect body coords
      CALL D_VCROSS(P_D1_S1_S1, FS1_S1, TEMP1_3)
      CALL D_VCROSS(P_D2_S1_S1, FS1_S1, TEMP2_3)

      DO 40 I=1,3
        MD1_S1(I) = MS1_S1(I) - TEMP1_3(I)
        MD2_S1(I) = -MS1_S1(I) + TEMP2_3(I)
40    CONTINUE

      CALL D_MXV(T_B1_S1, MD1_S1, MD1_B1, 3, 3)
      CALL D_MXV(T_B2_S1, MD2_S1, MD2_B2, 3, 3)

cc  Add contact moments to node moment vector
      DO 50 I=1,3
        MN_BODY(1,NODE_DOCK(1),I) = MD1_B1(I)
        MN_BODY(2,NODE_DOCK(2),I) = MD2_B2(I)
50    CONTINUE

      RETURN
      END
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE CONTROL(NB)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc Function:        Computes control forces and torques applied to
cc                  bodies in the respective body fixed frames.
cc
cc Source:          JC     Date: 3/91
cc
cc Comments:        Double precision
cc                  Data via common block declarations
cc                  File serves as a place holder when no control
cc                  is implemented
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

cc Include files
      INCLUDE 'BODY.INC'
      INCLUDE 'INTFAC.INC'

cc Locals


      RETURN
      END
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE LUMASSM(NB)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc Function:        Computes mass matrix of body #NB in lower-upper
cc                  decomposed format (i.e., LU decomposition);
cc                  the resulting system can be solved using
cc                  forwards/backwards substitution.
cc
cc Source:          JC     Date: 2/91
cc
cc Comments:        Double precision
cc                  Data via common block declarations
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      IMPLICIT NONE

cc Include files
      INCLUDE 'BODY.INC'

cc Local declarations
      INTEGER I, J, K, L
      INTEGER NB, NM, INDXX(6+MNM)
      REAL*8 MASSM(6+MNM,6+MNM)
      REAL*8 TEMP_3MNM(3,MNM), TEMP_MNM(MNM)
      REAL*8 GAM0_ETA(3), G0_ETA_T(3,3), GAM_2(3,MNM)
      REAL*8 II_1(3,3), II_2(3,3), D

cc Start building lower triangular mass matrix (symmetric)
cc Add diagonal mass submatrix (3x3) assoc with rigid body trans
      MASSM(1,1) = MASS(NB)
      MASSM(2,2) = MASS(NB)
      MASSM(3,3) = MASS(NB)

cc Add rigid body moment of inertia about body CM submatrix (3x3)
cc  to lower triangular mass matrix
      DO 10 I=1,3
        DO 20 J=1,I
          MASSM(3+I,3+J) = ICM(NB,I,J)
20      CONTINUE
10    CONTINUE

cc Check for no flex body modes
      NM = NMODES(NB)
      IF(NM .EQ. 0) GOTO 500

cc Add generalized mass submatrix (NMxNM) assoc with modal
cc  coords to lower triangular mass matrix
      DO 30 I=1,NM
```

```
      DO 40 J=1,I
        MASSM(6+I,6+J) = GMASS(NB,I,J)
40    CONTINUE
30    CONTINUE

cc  Coupling between rigid body translation and rotation
cc   via mass integral terms
cc  Pull gamma0 (3xNM) matrix and eta vector (NMx1) out
      IF( .NOT. EOM(NB,3) ) THEN
         CALL D_ZERO(TEMP_3MNM,3,MNM)

         DO 50 J=1,NM
           DO 60 I=1,3
             TEMP_3MNM(I,J) = GAM_0(NB,I,J)
60         CONTINUE
           TEMP_MNM(J) = GSV(NB,7+J)
50       CONTINUE

cc  Create gamma_0 (3 x MNM) * eta (MNM x 1) tilde matrix
         CALL D_MMUL(TEMP_3MNM,TEMP_MNM,GAM0_ETA,3,MNM,1)
         CALL D_TILDE(GAM0_ETA, G0_ETA_T)
      ELSE
         CALL D_ZERO(G0_ETA_T, 3, 3)
      ENDIF

cc  Add gam0_eta tilde (3x3), gam0 (3xNM), and gam1 (3xNM)
cc   submatrices to lower triangular mass matrix
      DO 70 I=1,3
        DO 80 J=1,3
          MASSM(3+I,J) = G0_ETA_T(I,J)
80      CONTINUE
        DO 90 J=1,NM
          MASSM(6+J,I) = GAM_0(NB,I,J)
          MASSM(6+J,3+I) = GAM_1(NB,I,J)
90      CONTINUE
70    CONTINUE

cc  Check for elimination of higher order mass integral terms
      IF(EOM(NB,1)) GOTO 500

cc  Compute gamma2 (3 x NM) and add to lower triangular mass mat
      DO 100 I=1,3
        DO 110 J=1,NM
          GAM_2(I,J) = 0.0D0
          DO 120 K=1,NM
            GAM_2(I,J) = GAM_2(I,J) + GSV(NB,7+K)*
     &                   GAM_2JK(NB,K,J,I)
120       CONTINUE
          MASSM(6+J,3+I) = MASSM(6+J,3+I) + GAM_2(I,J)
```

```
110    CONTINUE
100  CONTINUE

cc Compute II_1 (3 x 3) and II_2 (3 x 3); HOM-ITs affecting
cc  moment of inertia matrix
    DO 170 I=1,3
     DO 180 J=1,3
      II_1(I,J) = 0.0D0
      II_2(I,J) = 0.0D0
      DO 190 K=1,NM
       II_1(I,J) = II_1(I,J) + GSV(NB,7+K)*
   &            ( I_1K(NB,K,I,J) + I_1K(NB,K,J,I) )
       DO 200 L=1,NM
        II_2(I,J) = II_2(I,J) + 0.5*GSV(NB,7+K)*GSV(NB,7+L)*
   &            ( I_2JK(NB,K,L,I,J) + I_2JK(NB,L,K,I,J) )
200      CONTINUE
190     CONTINUE
180    CONTINUE
170   CONTINUE

cc Add II_1 and II_2 to lower triangular mass matrix
    DO 220 I=1,3
     DO 230 J=1,I
      MASSM(3+I,3+J) = MASSM(3+I,3+J) + II_1(I,J) + II_2(I,J)
230    CONTINUE
220   CONTINUE

cc Now have FUN-WHOM-IT mass matrix

500  CONTINUE

cc Fill in upper triangle of mass matrix using symmetry prop
    DO 400 I=1,6+NM
     DO 420 J=I,6+NM
      MASSM(I,J) = MASSM(J,I)
420    CONTINUE
400   CONTINUE

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
c    WRITE(6,*)
c    WRITE(6,*) '  MASSM:'
c    DO 425 I=1,6+NM
c      WRITE(6,'(1X,10E12.5)') (MASSM(I,J),J=1,6+NM)
c425   CONTINUE
c    WRITE(6,*)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

cc Compute lower-upper decomposition of mass matrix
     CALL LUDCMP(MASSM,6+NM,6+MNM,INDXX,D)
```

```
cc  Record in memory
      DO 510 I=1,6+NM
       INDX(NB,I) = INDXX(I)
       DO 520 J=1,6+NM
        LU_MASSM(NB,I,J) = MASSM(I,J)
520     CONTINUE
510   CONTINUE

      RETURN
      END
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PLANT(NB)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc Function:                Computes right hand side of equations of motion.
cc                          These terms represent the effects of damping,
cc                          stiffness, and forcing functions.
cc
cc Source:                  JC     Date: 2/91
cc
cc Comments:        Double precision
cc                          Data via common block declarations
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      IMPLICIT NONE

cc Include files
      INCLUDE 'BODY.INC'

cc Local declarations
      INTEGER I, J, K, L, M, N
      INTEGER NB, NM, INDXX(6+MNM)
      REAL*8 MASSM(6+MNM,6+MNM)
      REAL*8 TEMP1_MNM3(MNM,3), TEMP2_MNM3(MNM,3)
      REAL*8 TEMP1_3MNM(3,MNM), TEMP2_3MNM(3,MNM)
      REAL*8 TEMP1_3(3), TEMP2_3(3), TEMP3_3(3), TEMP4_3(3)
      REAL*8 TEMP5_3(3), TEMP6_3(3), TEMP_33(3,3)
      REAL*8 TEMP1_MNM(MNM), TEMP2_MNM(MNM)
      REAL*8 GAM0_ETA(3), GAM0_ETAD(3)
      REAL*8 GAM_2D(3,MNM), GAM_2(3,MNM)
      REAL*8 RHS(6+MNM), FNM(MNM), TNM(MNM)
      REAL*8 F_BODY(3), T_BODY(3)
      REAL*8 WB(3)
      REAL*8 II_1(3,3), II_2(3,3), II_1D(3,3), II_2D(3,3)
      REAL*8 II(3,3), IID(3,3)
      REAL*8 D_DOT

cc Rigid body translational/rotational forcing functions
cc Based on nodes locations defined over each body
cc Mult forces, pure torques, and POS x FORCE torques due to
cc  node forces are summed into a single force/torque at CM.
      CALL D_ZERO(F_BODY, 3, 1)
      CALL D_ZERO(T_BODY, 3, 1)
      CALL D_ZERO(RHS, 6+MNM, 1)
      NM = NMODES(NB)

cc Compute torque at CM due to POS x FORCE of node forces
cc Include flexibility effects on node position vectors
      DO 10 N=1,NNODES(NB)
```

```
      DO 15 J=1,3
        TEMP1_3(J) = NODES(NB,N,J) + PHIETA(NB,N,J)
        TEMP2_3(J) = FN_BODY(NB,N,J)
15    CONTINUE
      CALL D_VCROSS(TEMP1_3, TEMP2_3, TEMP3_3)
      DO 20 J=1,3
        F_BODY(J) = F_BODY(J) + FN_BODY(NB,N,J)
        T_BODY(J) = T_BODY(J) + MN_BODY(NB,N,J) + TEMP3_3(J)
20    CONTINUE
10    CONTINUE

cc    Build right hand side terms; pull out body ang vel vector
      DO 30 J=1,3
        RHS(J) = F_BODY(J)
        RHS(3+J) = T_BODY(J)
        WB(J) = QSV(NB,3+J)
30    CONTINUE

cc    Check for no flex body modes
      IF(NM .EQ. 0) GOTO 500

cc    Compute EHOMIT + ERNLT flex RHS terms
cc    Generalized flex eqns forcing functions
      DO 40 N=1,NNODES(NB)
        DO 50 M=1,NM
          DO 60 I=1,3
            TEMP1_MNM3(M,I) = PHI(NB,N,M,I)
            TEMP2_MNM3(M,I) = PSI(NB,N,M,I)
60        CONTINUE
50      CONTINUE
        DO 70 I=1,3
          TEMP1_3(I) = FN_BODY(NB,N,I)
          TEMP2_3(I) = MN_BODY(NB,N,I)
70      CONTINUE
        CALL D_MMUL(TEMP1_MNM3, TEMP1_3, FNM, MNM, 3, 1)
        CALL D_MMUL(TEMP2_MNM3, TEMP2_3, TNM, MNM, 3, 1)
        DO 80 M=1,NM
          RHS(6+M) = RHS(6+M) + FNM(M) + TNM(M)
80      CONTINUE
40    CONTINUE

cc    Strain energy and damping terms
      DO 90 M=1,NM
         RHS(6+M) = RHS(6+M) - GSV(NB,7+M) * MOD_FREQ(NB,M)**2 -
     &        2.D0 * MOD_ZETA(NB,M) * MOD_FREQ(NB,M) * GSVD(NB,7+M)
90    CONTINUE

cc    Check for EHOMIT + ERNLT option; ERNLT encompasses EHOMIT
      IF(EOM(NB,2)) GOTO 500
```

```
cc  Nonlinear translational RHS terms
cc  Pull out gamma0 (3xNM) matrix, gamm1 (3xNM),
cc   eta vector (NMx1), and etad vector (NMx1)
      CALL D_ZERO(TEMP1_3MNM,3,MNM)
      CALL D_ZERO(TEMP2_3MNM,3,MNM)

      DO 100 J=1,NM
       DO 110 I=1,3
        TEMP1_3MNM(I,J) = GAM_0(NB,I,J)
        TEMP2_3MNM(I,J) = GAM_1(NB,I,J)
110     CONTINUE
       TEMP1_MNM(J) = GSV(NB,7+J)
       TEMP2_MNM(J) = QSV(NB,6+J)
100   CONTINUE

cc  Create gamma_0 (3 x MNM) * eta (MNM x 1) vector
cc  Compute wb x (wb x gam0*eta)
      IF( .NOT. EOM(NB,3) ) THEN
         CALL D_MMUL(TEMP1_3MNM, TEMP1_MNM, GAM0_ETA, 3, MNM, 1)
         CALL D_VCROSS(WB, GAM0_ETA, TEMP1_3)
         CALL D_VCROSS(WB, TEMP1_3, TEMP2_3)
      ELSE
         TEMP2_3(1) = 0.D0
         TEMP2_3(2) = 0.D0
         TEMP2_3(3) = 0.D0
      ENDIF

cc  Compute wb x (mass*Rdcmb + 2*gam0*etad)
      IF( .NOT. EOM(NB,3) ) THEN
         CALL D_MMUL(TEMP1_3MNM, TEMP2_MNM, GAM0_ETAD, 3, MNM, 1)
      ELSE
         GAM0_ETAD(1) = 0.D0
         GAM0_ETAD(2) = 0.D0
         GAM0_ETAD(3) = 0.D0
      ENDIF

      DO 120 I=1,3
       TEMP1_3(I) = MASS(NB) * QSV(NB,I) +
     &          2.D0 * GAM0_ETAD(I)
120   CONTINUE

      CALL D_VCROSS(WB, TEMP1_3, TEMP3_3)

cc  Add translational terms to RHS
      DO 130 I=1,3
       RHS(I) = RHS(I) - TEMP2_3(I) - TEMP3_3(I)
       TEMP1_3(I) = QSV(NB,I)
130   CONTINUE
```

```
cc  Nonlinear rotational RHS terms
cc  wb x (gam1 * etad)
    IF( .NOT. EOM(NB,3) ) THEN
       CALL D_MMUL(TEMP2_3MNM, TEMP2_MNM, TEMP2_3, 3, MNM, 1)
       CALL D_VCROSS(WB, TEMP2_3, TEMP3_3)

cc  gam0*eta x ( wb x Rdcmb)
       CALL D_VCROSS(WB, TEMP1_3, TEMP4_3)
       CALL D_VCROSS(GAM0_ETA, TEMP4_3, TEMP5_3)
    ELSE
       DO 135 I=1,3
        TEMP3_3(I) = 0.D0
        TEMP5_3(I) = 0.D0
135    CONTINUE
    ENDIF

cc  wb x (ICM*wb)
    IF(EOM(NB,1)) THEN
     DO 140 I=1,3
      DO 150 J=1,3
       TEMP_33(I,J) = ICM(NB,I,J)
150     CONTINUE
140    CONTINUE

     CALL D_MMUL(TEMP_33, WB, TEMP6_3, 3, 3, 1)
     CALL D_VCROSS(WB, TEMP6_3, TEMP4_3)
    ELSE
       TEMP4_3(1) = 0.D0
       TEMP4_3(2) = 0.D0
       TEMP4_3(3) = 0.D0
    ENDIF

cc  Add nonlinear rotational terms to RHS
    DO 160 I=1,3
     RHS(3+I) = RHS(3+I) - TEMP3_3(I) - TEMP4_3(I) -
   &          TEMP5_3(I)
160  CONTINUE

cc  Nonlinear RHS terms in flex eqns
cc   Rdcmb . (wb x gam0) ; performed on sub vector basis in arrays
    IF( .NOT. EOM(NB,3) ) THEN
       DO 170 M=1,NM
        DO 180 I=1,3
         TEMP2_3(I) = GAM_0(NB,I,M)
180     CONTINUE
        CALL D_VCROSS(WB, TEMP2_3, TEMP3_3)

cc  Add to RHS
```

```
        RHS(6+M) = RHS(6+M) + D_DOT(TEMP1_3, TEMP3_3, 3)
170   CONTINUE
      ENDIF

cc  Check for elimination of higher order mass integral terms
      IF(EOM(NB,1)) GOTO 500

cc  HOMITs on RHS of rotational eqns
cc  Compute II (3 x 3); moment of inertia matrix including
cc   effects of HOMITs
cc  Compute IID (derivative of II) (3 x 3)
      DO 200 I=1,3
        DO 210 J=1,3
          II_1(I,J) = 0.D0
          II_2(I,J) = 0.D0
          II_1D(I,J) = 0.D0
          II_2D(I,J) = 0.D0
          DO 220 K=1,NM
            II_1(I,J) = II_1(I,J) + GSV(NB,7+K)*
     &                  ( I_1K(NB,K,I,J) + I_1K(NB,K,J,I) )
            II_1D(I,J) = II_1D(I,J) + QSV(NB,6+K)*
     &                  ( I_1K(NB,K,I,J) + I_1K(NB,K,J,I) )
            DO 230 L=1,NM
              II_2(I,J) = II_2(I,J) + 0.5*GSV(NB,7+K)*GSV(NB,7+L)*
     &                ( I_2JK(NB,K,L,I,J) + I_2JK(NB,L,K,I,J) )
              II_2D(I,J) = II_2D(I,J) + 0.5*QSV(NB,6+K)*GSV(NB,7+L)*
     &                ( I_2JK(NB,K,L,I,J) + I_2JK(NB,L,K,I,J) ) +
     &                 0.5*GSV(NB,7+K)*QSV(NB,6+L)*
     &                ( I_2JK(NB,K,L,I,J) + I_2JK(NB,L,K,I,J) )
230         CONTINUE
220       CONTINUE
          II(I,J) = ICM(NB,I,J) + II_1(I,J) + II_2(I,J)
          IID(I,J) = II_1D(I,J) + II_2D(I,J)
210     CONTINUE
200   CONTINUE

cc  Compute  wb x (II * wb)
      CALL D_MMUL(II, WB, TEMP2_3, 3, 3, 1)
      CALL D_VCROSS(WB, TEMP2_3, TEMP1_3)

cc  Compute  IID * wb
      CALL D_MMUL(IID, WB, TEMP2_3, 3, 3, 1)

cc  Compute  wb x (gam2 * etad)
      DO 240 I=1,3
        DO 250 J=1,NM
          GAM_2(I,J) = 0.D0
          DO 260 K=1,NM
            GAM_2(I,J) = GAM_2(I,J) + GSV(NB,7+K)*
```

```
   &              GAM_2JK(NB,K,J,I)
260     CONTINUE
250    CONTINUE
240  CONTINUE

     CALL D_MMUL(GAM_2, TEMP2_MNM, TEMP3_3, 3, MNM, 1)
     CALL D_VCROSS(WB, TEMP3_3, TEMP4_3)

cc  Add rotational HOMITs to RHS
     DO 270 I=1,3
      RHS(3+I) = RHS(3+I) - TEMP1_3(I) - TEMP2_3(I) -
   &          TEMP4_3(I)

cc  Compute gam_2 dot (3 x NM) for flex eqns
      DO 280 J=1,NM
       GAM_2D(I,J) = 0.D0
       DO 290 K=1,NM
        GAM_2D(I,J) = GAM_2D(I,J) + QSV(NB,6+K)*
   &              GAM_2JK(NB,K,J,I)
290     CONTINUE
280    CONTINUE
270  CONTINUE

cc  Compute  -2*wb . gam_2dot on array sub vector basis
     DO 300 M=1,NM
      DO 310 I=1,3
       TEMP1_3(I) = GAM_2D(I,M)
310    CONTINUE

      RHS(6+M) = RHS(6+M) - 2.D0 * D_DOT(WB, TEMP1_3, 3)

cc  Pull out each I_1k 3x3 submatrix, compute  wb'*I1m*wb,
cc   and add to RHS
      DO 320 I=1,3
       DO 330 J=1,3
        TEMP_33(I,J) = I_1K(NB, M, I, J)
330     CONTINUE
320    CONTINUE

      CALL D_MMUL(TEMP_33, WB, TEMP2_3, 3, 3, 1)
      RHS(6+M) = RHS(6+M) + D_DOT(WB, TEMP2_3, 3)

cc  Compute col vect over I: 0.5 * wb' * sumj(I_2jI + I_2Ij) * wb
      CALL D_ZERO(TEMP_33, 3, 3)

      DO 340 K=1,NM
       DO 350 I=1,3
        DO 360 J=1,3
         TEMP_33(I,J) = TEMP_33(I,J) + GSV(NB,7+K) *
```

```
   &            ( I_2JK(NB,K,M,I,J) + I_2JK(NB,M,K,I,J) )
360       CONTINUE
350      CONTINUE
340     CONTINUE

      CALL D_MMUL(TEMP_33, WB, TEMP3_3, 3, 3, 1)
      RHS(6+M) = RHS(6+M) + 0.5D0 * D_DOT(WB, TEMP3_3, 3)
300   CONTINUE

cc  Solve system for deriv of quasi coord vector

500   CONTINUE

cc  Extract appropriate upper triangular mass matrix
      DO 510 I=1,6+NM
       DO 520 J=1,6+NM
        MASSM(I,J) = LU_MASSM(NB,I,J)
520     CONTINUE
510   CONTINUE

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
c     WRITE(6,*)
c     WRITE(6,*) ' RHS: '
c     WRITE(6,'(1X,10E12.5)')  (RHS(I),I=1,6+NM)
c     WRITE(6,*)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

cc  Solve upper triangular system (A*x = B) using back sub
      DO 530 J=1,6+NM
       INDXX(J) = INDX(NB,J)
530   CONTINUE

      CALL LUBKSB(MASSM, 6+NM, 6+MNM, INDXX, RHS)

cc  Store quasi-coord state vector derivative
      DO 550 I=1,6+NM
       QSVD(NB,I) = RHS(I)
550   CONTINUE

      RETURN
      END
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE INTEG(NB)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc  Function:        Integrates derivatives of quasi-coord state
cc                   vector (W); transforms (W) to generalized
cc                   coord state vector derivative (qdot) and
cc                   integrates qdot to produce q.  Normalizes
cc                   quaternion in generalized coord vector.
cc
cc              Four integration options based on integer INT:
cc                   INT = 1  -->  Euler or Adams 1
cc                   INT = 2  -->  Adams (3 - 1) / 2
cc                   INT = 3  -->  Adams (23 - 16 + 5) / 12
cc
cc  Source:          JC     Date: 2/91
cc
cc  Comments:        Double precision
cc                   Data via common block declarations
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      IMPLICIT NONE

cc  Include files
      INCLUDE 'BODY.INC'

cc  Local declarations
      INTEGER I
      INTEGER NB, NM
      REAL*8  TEMP1_3(3), TEMP2_3(3), TEMP_4(4), Q4(4)
      REAL*8  WB(3), ET(4,3), TEMP1_33(3,3), TEMP2_33(3,3)

cc  Initialization
      NM = NMODES(NB)

cc  Transform quasi-coord state vector (QSV) to deriv of gen coord
cc   state vector (GSVD) in sub blocks;  {GSVD} = [beta] * {QSV}

cc  Rigid body translation rates
cc  Extract body vel vector PD_B_I_B, ang vel vector W_B_I_B,
cc   and BODY <-- I quaternion
      DO 20 I=1,3
        TEMP1_3(I) = QSV(NB,I)
        WB(I) = QSV(NB,3+I)
        Q4(I) = GSV(NB,3+I)
20    CONTINUE
      Q4(4) = GSV(NB,7)

cc  Transform PD_B_I_B to PD_B_I_I
```

```
      CALL Q_T_DCM(Q4, TEMP1_33)
      CALL D_MTRANS(TEMP1_33, TEMP2_33, 3, 3)
      CALL D_MMUL(TEMP2_33, TEMP1_3, TEMP2_3, 3, 3, 1)

cc    Build (Q_B_I)dot <-- angular body rate vector trans
      ET(1,1) = Q4(4)
      ET(1,2) = -Q4(3)
      ET(1,3) = Q4(2)
      ET(2,1) = Q4(3)
      ET(2,2) = Q4(4)
      ET(2,3) = -Q4(1)
      ET(3,1) = -Q4(2)
      ET(3,2) = Q4(1)
      ET(3,3) = Q4(4)
      ET(4,1) = -Q4(1)
      ET(4,2) = -Q4(2)
      ET(4,3) = -Q4(3)

cc    Update Q_B_I derivative
      CALL D_MMUL(ET, WB, TEMP_4, 4, 3, 1)

cc    Add to GSVD
      DO 30 I=1,3
        GSVD(NB,I) = TEMP2_3(I)
        GSVD(NB,3+I) = 0.5D0 * TEMP_4(I)
30    CONTINUE
      GSVD(NB,7) = 0.5D0 * TEMP_4(4)

cc    Modal coord trans is an identity matrix
      DO 50 I=1,NM
        GSVD(NB,7+I) = QSV(NB,6+I)
50    CONTINUE

cc    Integrate gen coord state deriv vector based on INT option
cc    Integrate quasi-coord state deriv vector based on INT option
      IF(INT .EQ. 1) THEN
        DO 60 I=1,7+NM
          GSV(NB,I) = GSV(NB,I) + DT*GSVD(NB,I)
60      CONTINUE

        DO 70 I=1,6+NM
          QSV(NB,I) = QSV(NB,I) + DT*QSVD(NB,I)
70      CONTINUE

      ELSEIF(INT .EQ. 2) THEN
        DO 80 I=1,7+NM
          GSV(NB,I) = GSV(NB,I) + (DT/2.D0) * ( 3.D0 * GSVD(NB,I) -
     &                          GSVDO_1(NB,I) )
80      CONTINUE
```

```
      DO 90 I=1,6+NM
       QSV(NB,I) = QSV(NB,I) + (DT/2.D0) * ( 3.D0 * QSVD(NB,I) -
     &                        QSVDO_1(NB,I) )
90    CONTINUE

      ELSEIF(INT .EQ. 3) THEN
       DO 100 I=1,7+NM
        GSV(NB,I) = GSV(NB,I) + (DT/12.D0) * ( 23.D0 * GSVD(NB,I) -
     &          16.D0 * GSVDO_1(NB,I) + 5.D0 * GSVDO_2(NB,I) )
100    CONTINUE

       DO 110 I=1,6+NM
        QSV(NB,I) = QSV(NB,I) + (DT/12.D0) * ( 23.D0 * QSVD(NB,I) -
     &          16.D0 * QSVDO_1(NB,I) + 5.D0 * QSVDO_2(NB,I) )
110    CONTINUE

      ELSE
       WRITE(6,*) '***  INTEG:  Integration Option Invalid  ***'
       STOP
      ENDIF

cc  Normalize Q_B_I and replace in generalized-coord vector
      DO 120 I=1,4
       Q4(I) = GSV(NB,3+I)
120   CONTINUE

      CALL D_QNORM(Q4)

      DO 130 I=1,4
       GSV(NB,3+I) = Q4(I)
130   CONTINUE

cc  Save old state derivative data
      DO 150 I=1,7+NM
       GSVDO_2(NB,I) = GSVDO_1(NB,I)
       GSVDO_1(NB,I) = GSVD(NB,I)
150   CONTINUE

      DO 160 I=1,6+NM
       QSVDO_2(NB,I) = QSVDO_1(NB,I)
       QSVDO_1(NB,I) = QSVD(NB,I)
160   CONTINUE

      RETURN
      END
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE INTFAC
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc  Function:       Updates transformations and relative position
cc                  and rate vectors; includes flexibility effects.
cc
cc  Source:         JC     Date: 3/91
cc
cc  Comments:       Double precision
cc                  Data via common block declarations
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      IMPLICIT NONE

cc  Include files
      INCLUDE 'BODY.INC'
      INCLUDE 'INTFAC.INC'

cc  Locals
      INTEGER  I, J, N, NB, M
      REAL*8  T_I_B1(3,3)
      REAL*8  T_D2_D1(3,3), T_D1O_B1(3,3)
      REAL*8  T_D1_D1O(3,3), T_D2O_D2(3,3)
      REAL*8  TEMP1_33(3,3), TEMP2_33(3,3)
      REAL*8  TEMP1_3(3), TEMP2_3(3), TEMP3_3(3), TEMP4_3(3)
      REAL*8  IDENT_33(3,3), Q4(4)
      REAL*8  W_B1_I_B1(3), W_B2_I_B2(3)

      DATA  IDENT_33 / 1.D0, 0.D0, 0.D0, 0.D0, 1.D0, 0.D0,
     &            0.D0, 0.D0, 1.D0/

cc  Build transformations from undeformed docking port systems
cc   to deformed docking point systems (due to infinitesimal
cc   rotations resulting from flexibility).
      DO 40 NB=1,NBODES

cc  Compute BODY <-- I transformations from generalized coord
cc   vector quaternions
       DO 10 I= 1,4
        Q4(I) = GSV(NB,3+I)
10     CONTINUE

       IF(NB .EQ. 1) THEN
        CALL Q_T_DCM(Q4, T_B1_I)
       ELSEIF(NB .EQ. 2) THEN
        CALL Q_T_DCM(Q4, T_B2_I)
       ENDIF
```

```
cc  Compute flexibility effects at each node
      DO 20 N=1,NNODES(NB)
       DO 30 J=1,3

        PHIETA(NB,N,J) = 0.D0
        PHIETAD(NB,N,J) = 0.D0
        PSIETA(NB,N,J) = 0.D0
        PSIETAD(NB,N,J) = 0.D0

        DO 35 M=1,NMODES(NB)
         PHIETA(NB,N,J) = PHIETA(NB,N,J) + PHI(NB,N,M,J)*
     &              GSV(NB,7+M)
         PHIETAD(NB,N,J) = PHIETAD(NB,N,J) + PHI(NB,N,M,J)*
     &               QSV(NB,6+M)
         PSIETA(NB,N,J) = PSIETA(NB,N,J) + PSI(NB,N,M,J)*
     &              GSV(NB,7+M)
         PSIETAD(NB,N,J) = PSIETAD(NB,N,J) + PSI(NB,N,M,J)*
     &               QSV(NB,6+M)
35       CONTINUE
30      CONTINUE
20     CONTINUE

cc  Compute and add in infinitesimal rotations at DOCKING node
cc   due to flexibility
       TEMP1_3(1) = PSIETA(NB,NODE_DOCK(NB),1)
       TEMP1_3(2) = PSIETA(NB,NODE_DOCK(NB),2)
       TEMP1_3(3) = PSIETA(NB,NODE_DOCK(NB),3)

       CALL D_TILDE(TEMP1_3, TEMP1_33)

       IF(NB .EQ. 1) THEN
         CALL D_ADDM(IDENT_33, TEMP1_33, TEMP2_33, 3, 3)
         CALL D_MTRANS(TEMP2_33, T_D1_D1O, 3, 3)
       ELSEIF(NB .EQ. 2) THEN
         CALL D_ADDM(IDENT_33, TEMP1_33, T_D2O_D2, 3, 3)
       ENDIF

40    CONTINUE

cc  Compute transformation D1 <-- D2
      CALL D_MTRANS(T_B1_D1O, T_D1O_B1, 3, 3)
      CALL D_MMUL(T_D1_D1O, T_D1O_B1, T_D1_B1, 3, 3, 3)
      CALL D_MTRANS(T_B1_I, T_I_B1, 3, 3)
      CALL D_MMUL(T_B2_I, T_I_B1, T_B2_B1, 3, 3, 3)
      CALL D_MTRANS(T_B2_B1, T_B1_B2, 3, 3)
      CALL D_MMUL(T_D1_B1, T_B1_B2, TEMP1_33, 3, 3, 3)
      CALL D_MMUL(T_B2_D2O, T_D2O_D2, TEMP2_33, 3, 3, 3)
      CALL D_MTRANS(TEMP2_33, T_D2_B2, 3, 3)
      CALL D_MMUL(TEMP1_33, TEMP2_33, T_D1_D2, 3, 3, 3)
```

```
      CALL D_MTRANS(T_D1_D2, T_D2_D1, 3, 3)

cc  Compute position vector D2 <-- D1 expressed in D1 frame
      DO 45 I=1,3
       TEMP1_3(I) = GSV(2,I) - GSV(1,I)
45    CONTINUE

      CALL D_MXV(T_B1_I, TEMP1_3, P_B2_B1_B1, 3, 3)

      DO 50 I=1,3
       P_D1_B1_B1(I) = P_D1O_B1_B1(I) + PHIETA(1,NODE_DOCK(1),I)
       TEMP1_3(I) = -P_D1_B1_B1(I) + P_B2_B1_B1(I)
50    CONTINUE

      CALL D_MXV(T_D1_B1, TEMP1_3, TEMP2_3, 3, 3)

      DO 60 I=1,3
       P_D2_B2_B2(I) = P_D2O_B2_B2(I) + PHIETA(2,NODE_DOCK(2),I)
60    CONTINUE

      CALL D_MXV(T_D2_B2, P_D2_B2_B2, TEMP3_3, 3, 3)
      CALL D_MXV(T_D1_D2, TEMP3_3, TEMP1_3, 3, 3)

      DO 70 I=1,3
       P_D2_D1_D1(I) = TEMP2_3(I) + TEMP1_3(I)
       W_B1_I_B1(I) = QSV(1,3+I)
       W_B2_I_B2(I) = QSV(2,3+I)
       TEMP1_3(I) = PSIETA(1,NODE_DOCK(1),I)
       TEMP2_3(I) = PSIETA(2,NODE_DOCK(2),I)
70    CONTINUE

cc  Compute angular velocity of D2 wrt D1 expressed in D2
      CALL D_VCROSS(W_B1_I_B1, TEMP1_3, TEMP3_3)
      CALL D_VCROSS(W_B2_I_B2, TEMP2_3, TEMP4_3)

      DO 80 I=1,3
       TEMP1_3(I) = -TEMP3_3(I) - W_B1_I_B1(I) -
     &             PSIETAD(1,NODE_DOCK(1),I)
       TEMP2_3(I) = TEMP4_3(I) + W_B2_I_B2(I) +
     &             PSIETAD(2,NODE_DOCK(2),I)
80    CONTINUE

      CALL D_MMUL(T_D2_B2, TEMP2_3, TEMP4_3, 3, 3, 3)
      CALL D_MMUL(T_D1_B1, TEMP1_3, TEMP3_3, 3, 3, 3)
      CALL D_MMUL(T_D2_D1, TEMP3_3, TEMP1_3, 3, 3, 3)

      DO 90 I=1,3
       W_D2_D1_D2(I) = TEMP1_3(I) + TEMP4_3(I)
90    CONTINUE
```

```
cc  Compute derivative of D2 <-- D1 position vector wrt D1
cc   expressed in D1
      CALL D_VCROSS(W_B1_I_B1, P_D1_B1_B1, TEMP3_3)
      CALL D_VCROSS(W_B2_I_B2, P_D2_B2_B2, TEMP4_3)

      DO 110 I=1,3
         TEMP1_3(I) = -TEMP3_3(I) - PHIETAD(1,NODE_DOCK(1),I)
     &           - QSV(1,I)
        TEMP2_3(I) = TEMP4_3(I) + PHIETAD(2,NODE_DOCK(2),I)
     &           + QSV(2,I)
        TEMP4_3(I) = PSIETA(1,NODE_DOCK(1),I)
110   CONTINUE

      CALL D_MXV(T_D1_B1, TEMP1_3, TEMP3_3, 3, 3)
      CALL D_MXV(T_D2_B2, TEMP2_3, TEMP1_3, 3, 3)
      CALL D_MXV(T_D1_D2, TEMP1_3, TEMP2_3, 3, 3)
      CALL D_VCROSS(W_B1_I_B1, TEMP4_3, TEMP1_3)

      DO 120 I=1,3
        TEMP4_3(I) = -W_B1_I_B1(I) - PSIETAD(1,NODE_DOCK(1),I)
     &           - TEMP1_3(I)
120   CONTINUE

      CALL D_MXV(T_D1_B1, TEMP4_3, TEMP1_3, 3, 3)
      CALL D_VCROSS(TEMP1_3, P_D2_D1_D1, TEMP4_3)

      DO 130 I=1,3
        PD_D2_D1_D1(I) = TEMP3_3(I) + TEMP2_3(I) + TEMP4_3(I)
130   CONTINUE

      RETURN
      END
```

APPENDIX B

MATH MODEL GLOBAL VARIABLE DEFINITIONS

DT	-	Integration step size (sec)
EOM(NBODES,3)	-	Equations of motion complexity flags
FN_BODY(NBODES,MNN,3)	-	External forces applied at each body node
FS1_S1(3)	-	Sensed contact forces
GAM_0(NBODES,3,MNM)	-	Array of GAM_0 modal integrals
GAM_1(NBODES,3,MNM)	-	Array of GAM_1 modal integrals
GAM_2JK(NBODES,MNM,MNM,3)	-	Array of GAM_2JK modal integrals
GMASS(NBODES,MNM,MNM)	-	Generalized mass matrices
GSV(NBODES,7+MNM)	-	Generalized coordinate vectors
GSVD(NBODES,7+MNM)	-	Generalized coordinate vector derivatives
GSVDO_1(NBODES,7+MNM)	-	Previous generalized coordinate vector derivatives
GSVDO_2(NBODES,7+MNM)	-	Second previous generalized coordinate vector derivatives
ICM(NBODES,3,3)	-	Body inertia matrices about respective body center of mass
INDX(NBODES,6+MNM)	-	Record of row permutations in LU decomposition
INT	-	Integration option flag
I_1K(NBODES,MNM,3,3)	-	Array of 3X3 I_1K modal integrals
I_2JK(NBODES,MNM,MNM,3,3)	-	Array of 3X3 I_2JK modal integrals
LU_MASSM(NBODES,6+MNM,6+MNM)	-	LU decomposed mass matrices
MASS(NBODES)	-	Body mass
MN_BODY(NBODES,MNN,3)	-	External moments applied at each body node

MNM	-	Maximum number of modes per body
MNN	-	Maximum number of nodes per body
MOD_FREQ(NBODES,MNM)	-	Flexible mode frequencies
MOD_ZETA(NBODES,MNM)	-	Damping per flexible mode
MS1_S1(3)	-	Sensed contact moments
NBODES	-	Number of bodies in simulation
NMODES(NBODES)	-	Number of flexible modes per body
NNODES(NBODES)	-	Number of nodes per body
NODE_DOCK(NBODES)	-	Body docking port node number
NODES(NBODES,MNN,3)	-	Node locations in respective body coordinates
P_B2_B1_B1(3)	-	Position vector of B2 center of mass with respect to B1 center of mass in B1 coordinates
PD_D2_D1_D1(3)	-	Velocity vector of chase docking node with respect to target docking node in D1 coordinates
P_D1O_B1_B1(3)	-	Position vector of undeformed target docking node with respect to target center of mass in B1 coordinates
P_D1_B1_B1(3)	-	Position vector of target docking node with respect to target center of mass in B1 coordinates
P_D1_S1_S1(3)	-	Position vector of target docking node with respect to force/moment sensor in S1 coordinates
P_D2O_B2_B2(3)	-	Position vector of undeformed chase docking node with respect to chase center of mass in B2 coordinates
P_D2_B2_B2(3)	-	Position vector of chase docking node with respect to chase center of mass in B2 coordinates

P_D2_D1_D1(3)	-	Position vector of chase docking node with respect to target docking node in D1 coordinates
PHI(NBODES,MNN,MNM,3)	-	Translational modal gains
PHIETA(NBODES,MNN,3)	-	Node translational deformation in respective body coordinates
PHIETAD(NBODES,MNN,3)	-	Node translational deformation velocity in respective body coordinates
PSI(NBODES,MNN,MNM,3)	-	Rotational modal gains
PSIETA(NBODES,MNN,3)	-	Node rotational deformation in respective body coordinates
PSIETAD(NBODES,MNN,3)	-	Node rotational deformation velocity in respective body coordinates
QSV(NBODES,6+MNM)	-	Quasi-coordinate vectors
QSVD(NBODES,6+MNM)	-	Quasi-coordinate vector derivatives
QSVDO_1(NBODES,6+MNM)	-	Previous quasi-coordinate vector derivatives
QSVDO_2(NBODES,6+MNM)	-	Second previous quasi-coordinate vector derivatives
T_B1_B2(3,3)	-	Transformation from B2 to B1 coordinates
T_B1_D1O(3,3)	-	Transformation from undeformed target docking frame to B1 coordinates
T_B1_I(3,3)	-	Transformation from inertial to B1 coordinates
T_B2_B1(3,3)	-	Transformation from B1 to B2 coordinates
T_B2_D2O(3,3)	-	Transformation from undeformed chase docking frame to B2 coordinates
T_B2_I(3,3)	-	Transformation from inertial to B2 coordinates
T_D1_B1(3,3)	-	Transformation from B1 to D1 coordinates
T_D1_D2(3,3)	-	Transformation from D2 to D1 coordinates

T_D1_S1(3,3)	-	Transformation from force/moment sensor to D1 coordinates
T_D2_B2(3,3)	-	Transformation from B2 to D2 coordinates
TIME	-	Simulation time (sec)
W_D2_D1_D2(3)	-	Angular velocity of chase docking frame with respect to target docking frame in D2 coordinates

APPENDIX C

FLEXIBLE BODY DATA PRE-PROCESSING CODE

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program massint
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc  Purpose:        Compute mass integral terms for a consistent mass
cc                  matrix.  Output modal data for use in flex body sims.
cc
cc  Source:         MC and JC      4/91
cc
cc  Explicit Inputs:
cc    ndof                          - number of degrees of freedom (DOF) of
cc                                    the system
cc    nnode                         - number of nodes of the system
cc    nmode                         - number of modes of the system
cc    idof_type(ndof)               - DOF array from DOF map
cc    inod(ndof)                    - node array from DOF map
cc    mass_mat(ndof,ndof)           - consistent mass matrix
cc    gen_mass(nmode,nmode)          - generalized mass mat (phi'*mass_mat*phi)
cc    phi(ndof,nmode)               - matrix of eigenvectors (mode shapes) of the
cc                                    system
cc    node_mat(nnode,3)             - nodal geometry matrix
cc    r_offset(3)                   - offset vector for nodal geometry matrix
cc
cc  Explicit Outputs:
cc    gen_mass                      - generalized mass matrix  (phi'*M*phi)
cc    gamma_0(3,nmode)              - zeroth mass integral vector
cc    gamma_1(3,nmode)              - first mass integral vector
cc    gamma_2(nmode,nmode,3)        - second mass integral vector
cc    I_1(nmode,3,3)                - first mass integral matrix
cc    I_2(nmode,nmode,3,3)          - second mass integral matrix
cc
cc  Others:
cc    rho(nnode,3)                  - offset nodal geometry matrix
cc    a(ndof,nmode)                 - matrix of the "a_j" vectors, stored column-wise
cc    a_t(nnode,nmode,3)            - subset of the "a" matrix describing the
cc                                    translation of the nodes (x,y,z)
cc    a_a(nnode,nmode,3)            - subset of the "a" matrix describing the
cc                                    rotation of the nodes (theta_x, theta_y, theta_z)
cc    phi_t(nnode,nmode,3)          - subset of the modal matrix describing the
cc                                    translation of the nodes (x,y,z)
cc    psi(nnode,nmode,3)            - subset of the modal matrix describing the
cc                                    rotation of the nodes (theta_x, theta_y, theta_z)
```

```
cc   rho_i(3)                    - vector describing the location of a node relative
cc                                 to an inertial frame of reference
cc   a_t_i(3)                    - vector of the "a_t" matrix at a particular node
cc                                 for a particular node
cc   r_x_a(3)                    - resultant vector of the cross product of "rho_i"
cc                                 and "a_t_i"
cc   a_x_phi(3)                  - resultant vector of the cross product of "a_t_i"
cc                                 and "phi_t_i"
cc   a_dot_phi                   - resultant scalar of the dot product of "a_t_i"
cc                                 and "phi_t_i"
cc   rho_phi_mat(3,3)            - resultant matrix of "rho_i" times "phi_t_i"
cc                                 transpose
cc   a_phi_mat(3,3)              - resultant matrix of "a_i" times "phi_t_i"
cc                                  transpose
cc   rho_dot_a                    - dot product of "rho_i" and "a_t_i"
cc   zeros(3,3)                  - 3x3 matrix of zeros used to initialize the mass
cc                                 integral matrices
cc   ident(3,3)                  - 3x3 identity matrix
cc
cc Comments:      Double precision
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer  ndofmax, nmodemax, nnodemax
      parameter (ndofmax = 40)
      parameter (nmodemax = 40)
      parameter (nnodemax = 10)

cc  Counters
      integer i, j, k, i1, icount, n

cc  Parameters of model
      integer nnode, nmode, ndof

cc  DOF map information
      integer idof_type(ndofmax), inod(ndofmax)

cc  Input variables
      real*8 mass_mat(ndofmax,ndofmax), phi(ndofmax,nmodemax)
      real*8 r_offset(3), node_mat(nnodemax,3)
      real*8 gen_mass(nmodemax,nmodemax), temp(ndofmax,nmodemax)

cc  Offset nodal map variables
      real*8 rho(nnodemax,3), rho_i(3), rho_x_a(3)
      real*8 rho_phi_mat(3,3), rho_dot_a

cc  Modal matrix variables
      real*8 phi_t(nnodemax, nmodemax, 3), psi(nnodemax, nmodemax, 3)
      real*8 phi_t_i(3)
```

```
cc "A" matrix variables
      real*8 a(ndofmax,nmodemax)
      real*8 a_t(nnodemax, nmodemax, 3), a_a(nnodemax, nmodemax, 3)
      real*8 a_t_i(3), a_x_phi(3), a_dot_phi
      real*8 a_phi_mat(3,3)

cc  Mass integral vectors
      real*8 gamma_0(3,nmodemax), gamma_1(3,nmodemax)
      real*8 gamma_2(nmodemax, nmodemax, 3)

cc  First mass integral matrix and associated temporary variables
      real*8 I_1_1(3,3), I_1_2(3,3), I_1_3(3,3), I_1_4(3,3)
      real*8 I_1(nmodemax,3,3), I_1_n_old(3,3)

cc  Second mass integral matrix and associated temporary variables
      real*8 I_2_1(3,3),I_2_2(3,3),I_2_3(3,3),I_2_4(3,3)
      real*8 I_2(nmodemax,nmodemax,3,3),I_2_n_old(3,3)

cc  3x3 zero and identity matrices
      real*8 zeros(3,3), ident(3,3)
      character*15  file1, file2

      data  zeros / 9*0.d0 /
      data  ident /1.d0, 3*0.d0, 1.d0, 3*0.d0, 1.d0/

cc  Open data files
      write(6,*)
      write(6,'(2x,"Enter raw data file name: ",$)')
      read(5,'(a)') file1
      write(6,'(2x,"Enter output data file name: ",$)')
      read(5,'(a)') file2
      write(6,*)

      open(unit=1, file = file1, status = 'old',
     &     form = 'formatted')
      open(unit=2, file = file2, status = 'unknown',
     &     form = 'formatted')

cc  Read DOF map
      read(1,*) nnode,ndof,nmode
      write(6,'(4x,"nnode; ndof; nmode: ",3i5)') nnode,ndof,nmode
      read(1,*)
      do 10 i=1,ndof
        read(1,*) inod(i), idof_type(i)
10    continue

cc  Read nodal geometry matrix
      read(1,*)
```

```
      read(1,*) (r_offset(i),i=1,3)
      do 35 i=1,nnode
        read(1,*) (node_mat(i,j),j=1,3)
35    continue

cc  Read mass matrix
      read(1,*)
      do 20 i=1,ndof
        read(1,*) (mass_mat(i,j),j=1,ndof)
20    continue

      write(6,*) '   Mass matrix read complete'

cc  Read modal (eigenvector) matrix
      read(1,*)
      do 30 i=1,ndof
        read(1,*) (phi(i,j),j=1,nmode)
30    continue

      write(6,*) '   Modal matrix read complete'

cc  Calculate the nodal position vector based on the nodal geometry
cc  matrix and the offset vector
      do 33 i=1,nnode
        do 34 j=1,3
          rho(i,j) = node_mat(i,j) - r_offset(j)
34      continue
33    continue

cc  Calculate generalized mass matrix
      call d_mmul(mass_mat, phi, temp, ndof, ndof, nmode,
     &          ndofmax, ndofmax, nmodemax)

      do 36 i=1,nmode
        do 37 j=1,nmode
          gen_mass(i,j) = 0.d0
          do 38 k=1,ndof
            gen_mass(i,j) = gen_mass(i,j) + phi(k,i) * temp(k,j)
38        continue
37      continue
36    continue

      write(6,*) '   Generalized mass matrix formed'

cc  Calculate the "a" matrix
      do 60 j=1,nmode
        do 40 i=1,ndof
          a(i,j) = 0.d0
          do 50 k=1,ndof
```

```
        a(i,j) = a(i,j) + mass_mat(i,k)*phi(k,j)
50      continue
40     continue
60    continue

      write(6,*) '    A matrix formed'

cc  Divide the "a" matrix into translational and rotational subsets
      do 80 j=1,nmode
        do 70 i=1,ndof
          if(idof_type(i).le.3) then
            a_t(inod(i), j, idof_type(i)) = a(i,j)
          elseif(idof_type(i).ge.4.and.idof_type(i).le.6) then
            a_a(inod(i), j, (idof_type(i)-3)) = a(i,j)
          endif
70       continue
80     continue

cc  Calculate the zeroth mass integral vector
      do 110 j=1,nmode
        do 100 k=1,3
          gamma_0(k,j) = 0.0d0
          do 90 i=1,nnode
            gamma_0(k,j) = gamma_0(k,j) + a_t(i,j,k)
90        continue
100      continue
110    continue

      write(6,*) '    Gamma_0J calculated'

cc  Calculate the first mass integral vector
      do 150 j=1,nmode
        do 130 k=1,3
          gamma_1(k,j) = 0.0d0
          do 140 i=1,nnode
            do 141 n=1,3
              a_t_i(n) = a_t(i,j,n)
              rho_i(n) = rho(i,n)
141         continue
            call d_vcross( rho_i, a_t_i, rho_x_a )
            gamma_1(k,j) = gamma_1(k,j) + a_a(i,j,k) + rho_x_a(k)
140       continue
130     continue
150   continue

      write(6,*) '    Gammo_1J calculated'

cc  Divide the modal matrix into translational and rotational subsets
      do 170 j=1,nmode
```

```
      do 160 i=1,ndof
        if(idof_type(i).le.3) then
          phi_t( inod(i), j, idof_type(i) ) = phi(i,j)
        elseif(idof_type(i).ge.4.and.idof_type(i).le.6) then
          psi( inod(i), j, (idof_type(i)-3) ) = phi(i,j)
        endif
160     continue
170   continue

    write(6,*) '   Separation of trans and rot modes'

cc  Initialize counter
    icount = 1

cc  Calculate the second mass integral vector
    do 230 j=icount,nmode
     do 220 k=icount,nmode
       do 180 i=1,3
         gamma_2(j,k,i) = 0.0d0
180      continue
        do 210 i1 = 1,nnode
         do 181 i=1,3
           a_t_i(i) = a_t(i1,j,i)
           phi_t_i(i) = phi_t(i1,k,i)
181        continue
          call d_vcross( a_t_i, phi_t_i, a_x_phi )
          do 200 i=1,3
            gamma_2(j,k,i) = gamma_2(j,k,i) + a_x_phi(i)
            gamma_2(k,j,i) = -gamma_2(j,k,i)
200         continue
210       continue
220     continue
      icount = icount+1
230   continue

    write(6,*) '   Gamma_2JK calculated'

cc  Calculate the first mass integral matrix
    do 280 k=1,nmode
     call d_fmim( I_1, zeros, k, nmodemax )
     do 270 j=1,nnode
        call d_mf3dm( I_1, I_1_n_old, k, nmodemax )
        call d_vf3dm( a_t, a_t_i, j, k, nnodemax, nmodemax )
        call d_vf3dm( phi_t, phi_t_i, j, k, nnodemax, nmodemax )
        call d_vf2dm( rho, rho_i, j, nnodemax )
        call d_dot( rho_i, a_t_i, rho_dot_a )
        call d_cxm( ident, rho_dot_a, I_1_1, 3, 3 )
        call d_mmul( phi_t_i, rho_i, rho_phi_mat, 3,1,3,3,1,3)
        call d_chsn( rho_phi_mat, I_1_2, 3, 3 )
```

```
        call d_addm( I_1_1, I_1_2, I_1_3, 3, 3 )
        call d_addm( I_1_3, I_1_n_old, I_1_4, 3, 3)
        call d_fmim( I_1, I_1_4, k, nmodemax )
270     continue
280   continue

      write(6,*) '    I_1K calculated'

cc  Calculate the second mass integral matrix
      do 310 j=1,nmode
        do 300 k=1,nmode
          call d_smim( I_2, zeros, j, k, nmodemax )
          do 290 i=1,nnode
            call d_mf4dm( I_2, I_2_n_old, j, k, nmodemax )
            call d_vf3dm( a_t, a_t_i, i, j, nnodemax, nmodemax )
            call d_vf3dm( phi_t, phi_t_i, i, k, nnodemax,
     &                  nmodemax )
            call d_dot( a_t_i, phi_t_i, a_dot_phi )
            call d_cxm( ident, a_dot_phi, I_2_1, 3, 3 )
            call d_mmul( a_t_i, phi_t_i, a_phi_mat, 3, 1, 3, 3, 1, 3)
            call d_chsn( a_phi_mat, I_2_2, 3, 3 )
            call d_addm( I_2_1, I_2_2, I_2_3, 3, 3 )
            call d_addm( I_2_3, I_2_n_old, I_2_4, 3, 3 )
            call d_smim( I_2, I_2_4, j, k, nmodemax )
290         continue
300       continue
310   continue

      write(6,*) '    I_2JK calculated'
      write(6,*)

cc  Write output in format consistent with NAMTB flex body
cc   simulation input

      write(6,'(2x,"Writing output data to: ",a15)') file2

      write(6,*)

      write(6,*) '    Generalized mass matrix'
      do i=1,nmode
        write(2,*)  (gen_mass(i,j),j=1,nmode)
      enddo

      write(6,*) '    Trans and rot modal gains'
      do i=1,nnode
       do j=1,nmode
        write(2,500) (phi_t(i,j,k),k=1,3)
       enddo
      enddo
```

```
      do i=1,nnode
       do j=1,nmode
         write(2,500) (psi(i,j,k),k=1,3)
       enddo
      enddo

      write(6,*) '   Gamma_0J'
      do j=1,nmode
        write(2,500)  (gamma_0(i,j),i=1,3)
      enddo

      write(6,*) '   Gamma_1J'
      do j=1,nmode
        write(2,500)  (gamma_1(i,j),i=1,3)
      enddo

      write(6,*) '   Gamma_2JK'
      do i=1,nmode
       do j=1,nmode
         write(2,500)  (gamma_2(i,j,k),k=1,3)
       enddo
      enddo

      write(6,*) '   I_1K'
      do i=1,nmode
       do j=1,3
         write(2,500) (I_1(i,j,k),k=1,3)
       enddo
      enddo

      write(6,*) '   I_2JK'
      do i=1,nmode
       do i1=1,nmode
        do j=1,3
          write(2,500) (I_2(i,i1,j,k),k=1,3)
        enddo
       enddo
      enddo

      write(6,*)
  500  format(1x,3e15.7)

      close(9)
      close(1)
      stop
      end

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine d_fmim(a,b,k,n)
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc Function:        Sets the value of the first mass integral matrix
cc
cc Source:          MC
cc
cc Explicit Inputs:
cc                  b - 3x3 matrix
cc                  n - number of modes
cc                  k - mode counter
cc
cc Explicit Outputs:
cc                  a - first mass integral matrix nx3x3
cc
cc Comments:        Double precision
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer n,i,j,k
      real*8 a(n,3,3), b(3,3)

      do 2 i=1,3
        do 1 j=1,3
          a(k,i,j) = b(i,j)
1       continue
2     continue

      return
      end

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine d_smim(a,b,k1,k2,n)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc Function:        Sets the value of the second mass integral matrix
cc
cc Source:          MC
cc
cc Explicit Inputs:
cc                  b  - 3x3 matrix
cc                  n  - number of modes
cc                  k1 - mode counter 1
cc                  k2 - mode counter 2
cc
cc Explicit Outputs:
cc                  a - second mass integral matrix nxnx3x3
cc
cc Comments:        Double precision
cc
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer n, i, j, k1, k2
      real*8 a(n,n,3,3), b(3,3)

      do 2 i=1,3
        do 1 j=1,3
          a(k1,k2,i,j) = b(i,j)
1       continue
2     continue

      return
      end

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine d_mf4dm( a,b,i,j,n )
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc  Function:       Separates a 3x3 matrix from a 4D matrix
cc
cc  Source:         MC
cc
cc  Explicit Inputs:
cc                  a - 4-dimensional matrix nxnx3x3
cc                  i - row position
cc                  n - row dimension
cc
cc  Explicit Outputs:
cc                  b - 3x3 matrix
cc
cc  Comments:       Double precision
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer n,i,j,k,k1
      real*8 a(n,n,3,3), b(3,3)

      do 2 k=1,3
        do 1 k1=1,3
          b(k,k1) = a(i,j,k,k1)
1       continue
2     continue

      return
      end

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine d_mf3dm( a,b,i,n )
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
cc
cc Function:        Separates a 3x3 matrix from a 3D matrix
cc
cc Source:          MC
cc
cc Explicit Inputs:
cc                  a - 3-dimensional matrix nx3x3
cc                  i - row position
cc                  n - row dimension
cc
cc Explicit Outputs:
cc                  b - 3x3 matrix
cc
cc Comments:        Double precision
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer n,i,j,k
      real*8 a(n,3,3), b(3,3)

      do 2 k=1,3
        do 1 j=1,3
          b(j,k) = a(i,j,k)
1       continue
2     continue

      return
      end

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine d_vf3dm( a,b,i,j,n,m )
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc Function:        Separates a 3 element vector from a 3D matrix
cc
cc Source:           MC
cc
cc Explicit Inputs:
cc                  a - 3-dimensional matrix nxmx3
cc                  i - row position
cc                  j - column position
cc                  n - row dimension
cc                  m - column dimension
cc
cc Explicit Outputs:
cc                  b - three element vector
cc
cc Comments:        Double precision
cc
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer n,m,i,j,k
      real*8 a(n,m,3), b(3)

      do 1 k=1,3
         b(k) = a(i,j,k)
1     continue

      return
      end

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine d_vf2dm( a,b,i,n )
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
cc  Function:       Separates a 3 element vector from a 2D matrix
cc
cc  Source:         MC
cc
cc  Explicit Inputs:
cc                  a - 2-dimensional matrix nx3
cc                  i - row position
cc                  n - row dimension
cc
cc  Explicit Outputs:
cc                  b - three element vector
cc
cc  Comments:       Double precision
cc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer n,i,k
      real*8 a(n,3), b(3)

      do 1 k=1,3
         b(k) = a(i,k)
1     continue

      return
      end
```