DYNAMICS OF A ROTATING BEAM WITH INTERMITTENT CONTACT

by

Mary Ann Ieropoli
B.Eng (Aerospace Engineering),
Ryerson University (Canada), 2004.

A thesis
Presented to Ryerson University
In partial fulfillment of the
Requirements for the degree of
Masters of Applied Science
In the program of
Mechanical Engineering

Toronto, Ontario, Canada, 2007

Author's Declaration

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Mary Ann Ieropoli

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Mary Ann Ieropoli

Abstract

DYNAMICS OF A ROTATING BEAM WITH INTERMITTENT CONTACT

Mary Ann Ieropoli, 2007
Masters of Applied Science
in the program of
Mechanical Engineering
Ryerson University

A flexible beam that is attached to a rotating hub, and whose tip encounters intermittent contact with a flat rigid surface is modelled. The beam is modelled using Euler-Bernoulli beam theory. Lagrange Equations are used to develop the system governing equations of motion, impact is modelled using the momentum balance method and contact is represented via a Lagrange multiplier and Coulomb friction. The model does not allow penetration of the surface to occur by enforcing a geometric constraint throughout contact. Both flexible and rigid initial beam assumptions before impact were analyzed. The effects of angular velocity, depth of penetration and the friction coefficient were examined. A numerical algorithm is outlined and Matlab software is used to implement the procedure. The results show some compliance with expected trends and they show smoother transition from unconstrained to constrained motion for the flexible initial beam configuration compared to the rigid configuration.

Acknowledgements

I would like to thank my co-supervisors Dr. Donatus Oguamanam and Dr. Glenn Heppler for all their guidance and insights. Dr. Oguamanam, I appreciate all the time and effort put into teaching me and helping me to develop my research skills. Also, thank you for your patience and support.

I would like to thank my family and friends for all their encouragement and support. Your understanding, patience and help are greatly appreciated.

Table of Contents

List of Figures

List of Tables

Nomenclature

Variable	Definition	Units
A	Cross-sectional area	m^2
A	Coordinate transformation matrix	--
BDF	Backwards Differentiation Formula	--
C	Constraint function dependent on space and time	m
D	Structurally-induced damping matrix	$kg(m^2)/s$
d	Normal distance between hub centre and flat rigid surface	m
E	Modulus of elasticity	N/m^2
e	Coefficient of restitution	--
$\hat{e}_r$	Unit vector along x-axis	--
$\hat{e}_\theta$	Unit vector along y-axis	--
$\hat{e}_n$	Unit vector along z-axis	--
F	Friction force vector	N
G	Gyroscopic-induced damping matrix (depending on context)	$kg(m^2)$
G	Vector of the partial derivative of the constraint function with respect to the generalized coordinates (depending on context)	--
gradC	Vector of the partial derivative of the constraint function with respect to the generalized coordinates	--
h	Step-size	s
I	Cross-sectional moment of inertia	m^4
$\hat{i}$	Unit vector along X-axis	--

Variable	Definition	Units
$\hat{j}$	Unit vector along Y-axis	--
$\hat{k}$	Unit vector along Z-axis	--
L	Length of beam	m
M	Inertial/Mass matrix from equations of motion	kg(m^2)
M	Bending moment	Nm
m	Mass	kg
N	Generalized normal force vector	N
NR	Newton-Raphson	--
P	Impulsive Force	N
Q	Non-conservative generalized force vector	N
$\mathbf{Q}_f$	Generalized friction force vector	N
Q	Shearing force	N
q	Generalized coordinate vector	--
$\dot{\mathbf{q}}$	Generalized velocity vector	1/s
$\ddot{\mathbf{q}}$	Generalized acceleration vector	1/s^2
$\Delta\dot{\mathbf{q}}$	Jump in generalized velocity vector	1/s
R	Global position vector	m
r	Local beam position vector	m
$\mathbf{r}_p$	Local beam position vector for x and y axes	m
r_h	Radius of hub	m

Variable	Definition	Units
$\dot{r}_i$	Solution to the characteristic equation multiplied by beam length of the ith mode	m^{-1}
rpm	Revolutions per minute	rev/min
$\mathbf{S}$	Vector containing all eigenfunctions	--
S_i	ith mode eigenfunction	--
s	Point along beam	m
T	Kinetic energy	J or $kg(m/s)^2$
t	Time	s
U	Potential energy	J or $kg(m/s)^2$
$\mathbf{u}$	Generalized velocity vector	m/s
$\mathbf{u}^p$	Predicted generalized velocity vector	m/s
$\mathbf{V}$	Global velocity vector	m/s
v	Normal component of beam tip velocity	m/s
X, Y, Z	Global axes	--
x, y, z	Local axes	--
$\mathbf{x}$	Generalized coordinate vector	m
$\mathbf{x}^p$	Predicted generalized coordinate vector	m
β_i	Solution to the characteristic equation of the ith mode	m^{-2}
θ	Angle of rotation	rad
κ	Curvature of beam	m^{-1}
λ	Lagrange multiplier	N

Variable	Definition	Units
μ	Friction coefficient	--
ξ	Non-dimensional beam length	--
ρ	Density of beam	kg/m^3
σ_i	Eigenfunction coefficient of the ith mode	--
ψ	Contains all unconstrained matrices and vectors except for $\mathbf{M}\ddot{\mathbf{q}}$	N
Φ	Matrix of the partial derivative of the position vector in the rotating frame with respect to the generalized coordinates.	m
ϕ	Angle of tangent from OX-axis to the beam	rad
ω	Angular velocity	rad/s

Chapter One: Introduction

Rotating beams play a major role in mechanical and aerospace applications. The solutions to problems encountered in robotics, engines, the use of helicopter rotors, etc., depend on a good understanding of the dynamics at work. Further complications in solving problems related to these rotating beams can arise when attempting to model impact or continuous contact. Controlling contact is very important, whether making a robotic arm follow a certain path with a particular amount of force, using wire bristles to polish a part, or simply trying to lessen its adverse effects. It aids in making a system more efficient, enhancing performance and minimizing expenses.

The first step to accomplishing this is to understand the behaviour of the system. This thesis attempts to model and analyze a flexible beam attached to a rotating hub, whose tip encounters intermittent rubbing with a flat rigid surface. The model does not permit the penetration of the rigid surface to occur by enforcing a geometric constraint throughout contact. Using Euler-Bernoulli beam theory, a half rotation causes the system to encounter free rotation (or unconstrained motion), impact, constrained motion on the rigid surface interspersed with brief moments of unconstrained motion, and finally, return to unconstrained motion.

Allowing the beam only to deform, the equations of motion are developed using Lagrange's equations. The coefficients of the equations of motion vary with time, and therefore have to be calculated at every time step. To reduce the amount of calculation, the volume integrals are calculated using Gaussian quadrature. An explicit fifth-order Runge-Kutta method is used to solve for the generalized coordinates and velocities in unconstrained motion. When the system goes from the unconstrained motion phase to the constrained motion phase, impact parameters are calculated using the momentum balance method.

Constrained motion is more difficult to model. Accurately calculating the exchange of energy that occurs when two bodies come into contact is one of the major challenges. Also, friction is a complex occurrence that is often encountered in such applications as the constrained motions of robots or the rubbing of engine components. There are many ways in which engineers attempt to model friction and other forces associated with contact. These usually include using Hertzian contact models, or using Coulomb friction combined with an assumed loading profile or combined with Lagrange multipliers. The latter method is used in this thesis along with a predictor-corrector numerical algorithm.

The transitions between the different types of motion also need to be considered. The integrity of the model depends on being able to combine all the stages in a continuous fashion. This is accomplished by making the most recent generalized quantities of the previous motion the initial conditions for the motion that is to follow. Once this is complete, the parameters angular speed, depth of penetration (if the constraint is not present), friction coefficient and beam configuration at initial impact will be varied to examine the system behaviour. The term 'depth of penetration' is a misnomer though it is consistent with the literature. It is the difference between the normal distance between the hub centre and the rigid surface and the summation of the hub radius and the length of the beam. It is represented in terms of the beam length.

The thesis is organized as follows. Chapter two presents a literature survey. Chapter three describes the system. This is followed by the derivation of the system equations of motion in chapter four. The system is solved using Matlab software that includes coding of Butcher's fifth-order Runge-Kutta method and a numerical method developed by Gear *et al.* [1]. The fifth chapter presents the simulation results. Chapter six elaborates more on the results and other issues encountered, including observed discrepancies in literature. Finally, concluding statements and suggestions for future work are presented in Chapter seven.

Chapter Two: Literature Review

In the reviewed literature, the topics that were encountered many times over included the modelling of flexibility, impact and contact, and axial stiffening. Most theory is based on rigid bodies, due to simplicity. As machine parts and products began to become lighter and faster, rigid body theory became insufficient to fully investigate these components. Therefore, the theory needed to be extended to flexible parts, and examining the implications and errors induced through rigid body theory on flexible components became essential. Also, impact and contact are very complex subjects and therefore require extensive investigation. Axial stiffening was often mentioned because it is an occurrence usually associated with fast rotating structures and ignoring it can introduce substantial error.

2.1 Modelling Flexibility

The most popular way to model a flexible beam is to describe displacement fields using eigenfunctions equations derived from the mode shapes of a non-rotating beam. The summation of these eigenfunctions is used with an approximate method such as the Galerkin method or the assumed modes method. Since the mode shapes are those of a non-rotating beam, the eigenfunctions themselves are approximate. Bellezza *et al.* [2] developed exact eigenfunctions for a linear problem by adding an extra term to the classic solution such that:

$$\psi_c(x) = A\sin(\beta x) + B\cos(\beta x) + C\sinh(\beta x) + D\cosh(\beta x) + Fx \tag{2-1}$$

Ψ_c represents the eigenfunction for a pseudo-clamped beam (the clamped end is rotating), A to F represent arbitrary values and β can be calculated from the characteristic equation. The theoretical results obtained were in very good agreement with experimental values.

Christoforou and Yigit [3] examined the effect of flexibility on low velocity impact response. It was shown that, depending on the nature of the impactor and target, three non-dimensional parameters can completely describe the response. They are: the normalized impact velocity, the relative stiffness of the impactor to the target, and the loss factor, which represents the energy lost by the impactor to the target during contact. From this, the maximum normalized force can be predicted without running a simulation. This is useful for designing impact loading or choosing similar simplified models that can have experimental results scaled to the true problem.

In Gilardi and Sharf [4], a description of the idea of the coefficient of restitution, which is utilized often in many investigations, was given with examples of how it is applied in different models. Care needs to be taken when using the coefficient of restitution for flexible bodies because the coefficient is derived using rigid body assumptions. Yigit *et al.* [5], [6] and Yigit [7] deal with the effect of flexibility on impact response when using this coefficient. All three papers used Euler-Bernoulli beam theory and extended Hamilton's principle to develop the system equations of motion, and the Galerkin method to find approximate solutions. Included in Yigit *et al.* [5] is a one-degree-of-freedom system example that showed that the effect of flexibility on the coefficient of restitution is so small that the rigid body theory assumptions are adequate. Hatman *et al.* [8], [9] used this conclusion to justify using this coefficient for their flexible systems.

Yigit [7] specifically compares rigid and flexible bodies by using a Hertzian contact model. The coefficient of restitution is used to obtain a suitable damping factor, even though major differences are found between the response characteristics of the two types of bodies. They include that the energy loss due to impact in the flexible case is much lower, that the change in angular velocity is much smoother for the flexible case, and that multiple contacts occur for the flexible case. The impulse due to impact is significantly higher for the rigid case. The author concluded that as long as some elastic motion can be excited by impact, the responses of flexible and rigid bodies differed significantly.

2.2 Modelling Impact and Contact

Gilardi and Sharf [4] presented a general literature survey for contact dynamics modelling, and it explained different methodologies for solving impact and contact problems. A popular approach to modelling impact problems is the momentum balance method in which the momentum does not change over the impact process because impact is assumed to be instantaneous, and therefore, body positions do not change. Yigit *et al.* [5], [6] and Palas *et al.* [10] used this concept along with the assumed mode method. The objective in Palas *et al.* [10]

was to determine if the use of these two methods is plausible for solving transverse impact problems involving constraints. Through establishing relationships between the generalized impulse, jump discontinuities in the velocity vector and reaction forces, impact parameters, and the number of elastic degrees of freedom, it was shown that, as long as the number of modes used is sufficiently high, the use of the impulse momentum equations and the coefficient of restitution are effective. The disadvantage of using the momentum balance method is that force histories cannot be obtained because impact is viewed as instantaneous and therefore contact duration is ignored.

Yigit [7] avoids this disadvantage by using a Hertzian contact model. This option, however, has some disadvantages: the damping force at the beginning of impact is not zero as it should be, the sum of the damping and spring forces during restitution can be negative, and the damping force reaches its maximum value when the relative displacement equals zero.

Hatman *et al.* [8] and [9] use the momentum balance method to model the initial impact and to calculate the jump in the system generalized velocities. The rest of the contact is modelled using Lagrange multipliers. This method avoids the damping boundary inconsistencies found with a Hertzian contact model and is able to provide a contact force history. Fung *et al.* [11] and Andersen *et al.* [12] also used the idea of Lagrange multipliers much like Hatman *et al.* did. Excluding the work done with piezomotors in the former two, the ideas and methodology for modelling friction are very similar. Andersen argued, however, that the quality of the results given by this method was unacceptable for his problem and suggested the inclusion of a contact stiffness to improve them. Refs. [13-15] also used this method.

Matsuno and Kasai [14] used Hamilton's principle, Euler-Bernoulli beam theory and the Lagrange multiplier method to model a constrained flexible one-link arm. The inclusion of friction due to the constraining surface introduces a non-homogenous term in the equations of motion. The authors concluded that determining the beam deflection via non-rotating cantilever beam mode shapes was not sufficiently accurate. Therefore, a change of variables was made to the transverse displacement to derive the homogeneous equivalent of the governing equations. Since the modified transverse displacement variable contained the friction term, a new eigenfunction modal model was derived that included both the geometric boundary conditions and the boundary conditions that were induced by the external friction force. The focus of the paper was the control of this system, and hence is out of the scope of this thesis. Ref. [15] extended the study to include a non-symmetric rigid tip mass.

Fung and Chang [13] commit to solely deriving the non-linear dynamic equations for single and double-link flexible systems using the Timoshenko and Euler beam models, the simple flexure model and the rigid body model via Hamiltonian's principle. The system used is a constrained flexible arm with a tip mass and any friction included is modelled using Lagrange multipliers.

Another approach is used in Sinha [16] to examine the dynamic response of a rotating Timoshenko beam whose tip rubs against a rigid casing. In this application, contact is modelled as a periodic pulse loading. The rubbing force acts along the beam axis, in compression, while the shear force due to the Coulomb friction is split between two axes: in the blade tangential axis, opposite the direction of motion, and in the blade normal axis as a point load in the beam lateral direction, using the Dirac delta function. This procedure requires that the contact force time and magnitude be supplied, rather than obtained. This makes the approach not useful for this thesis' purpose, though, the derivation is useful and the results are insightful.

The finite element method is another option used to solve dynamic models involving impact. Yang *et al.* [17] use the extended Hamilton's principle, Euler-Bernoulli beam theory and the finite element method to develop equations of motion for a flexible beam. Hsu *et al.* [18] prove that this method along with the generalized impulse method can be used to study the propagation of transverse waves due to impact in mechanical or structural systems.

In this thesis, the use of Lagrange multipliers to model contact is undertaken because a contact force history can be obtained without making any assumptions of the contact profile. Also, the coefficient of restitution, which is found to be valid for flexible structures, is included. The momentum balance method is used only for initial impact. Since the contact time of the beam is much longer than the instant of impact in this problem, this method is appropriate for the transition calculations when going from unconstrained to constrained motion.

2.3 Modelling Axial Stiffening

Another prevalent issue involves modelling the effects of axial stiffening due to rotation. It was mentioned in Refs. [5], [7], [8], and [17] that this occurrence must be included to avoid a dynamic softening effect. Hatman *et al.* [8] showed that including the longitudinal displacements was necessary to achieving this. In the problem description, however, the beam is only allowed to bend and the inextensibility assumption is used due to the differences in bending and stretching stiffness of a tiny wire filament. Therefore, the authors decided to include the longitudinal displacements within the calculation of the transverse displacements by defining the arc length of the clamping point and an arbitrary point on the beam as invariant. In Refs. [5], [7] and [17], however, a term is added to the potential energy, which represents the work done by the axial deformation.

Al-Qaisia and Al-Bedoor [19] compared these two methods along with a model that combines the two options and a model that does not account for the stiffening. Due to the consistency of the results for the potential energy option, this method was thought to be the best. This option showed the hardening effect of the centrifugal forces when the system coupled-dynamic response, the non-linear natural frequencies behaviour, and the system frequency response were analyzed. The kinetic energy method was seen as the next best option because some softening behaviour was found in the non-linear natural frequencies behaviour and in the frequency response analysis. The combination model performed the worst of the three options accounting for the stiffening. Since this thesis centres heavily on Hatman et al.'s work [8] and [9], the axial stiffening will be accounted for via the kinetic energy.

Chapter Three: Description of the System

Figure 1 shows the coordinate systems and the beam displacement of the system of interest. The global right-handed coordinate system is denoted by X, Y, and Z with unit vectors along each axis denoted by $\hat{i}$, $\hat{j}$, and $\hat{k}$, respectively. The local (or body-fixed) beam coordinate system is denoted by x, y, and z with unit vectors along each axis denoted by $\hat{e}_r$, $\hat{e}_\theta$, and $\hat{e}_n$, respectively. The positive x-axis runs radially from the origin along the beam. The y-axis runs normal to the beam in the direction of transverse deformation. The z-axis is the resultant of the cross product of $\hat{e}_r$ and $\hat{e}_\theta$. The beam will only undergo bending in the transverse direction and out-of-plane motion is not allowed. Any axial shortening due to rotation will be accounted for in the kinetic energy via the transverse eigenfunctions by defining the arc length of the clamping point and an arbitrary point on the beam as invariant [8]. This is

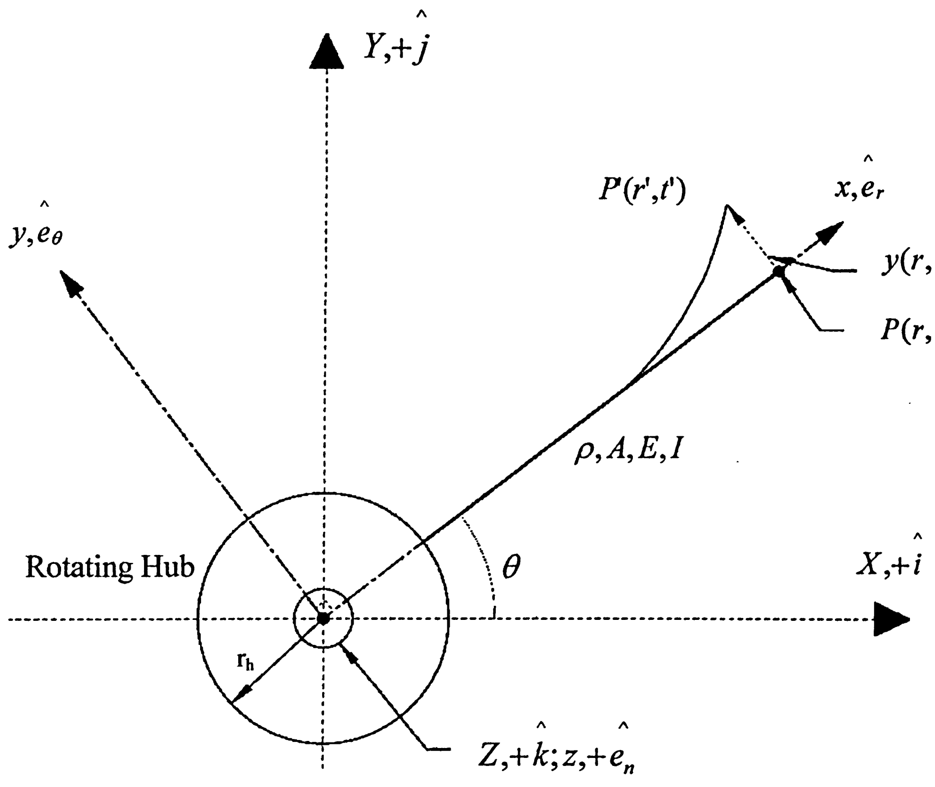

Figure 1: Coordinate systems and beam displacement.

consistent with the inextensibility assumption; the derivation of which can be found in Al-Qaisia and Al-Bedoor [19].

Variables 'A', 'ρ', 'E' and 'I' represent the beam cross-sectional area, the material density, the Young's modulus and the area moment of inertia, respectively. Angle 'θ' represents the angle between the beam and the positive X-axis.

Figure 2 is the system process schematic. The length of the beam is represented by 'L', and the hub radius by 'r_h'. 'd' is the normal distance between the fixed centre of the rotating hub and the rigid surface.

The global X-axis is positive in the downward direction from the origin, O. The Y-axis is positive in the direction on the right-hand side (RHS) of the origin, while the Z-axis is positive coming out of the page, in accordance with the right-hand thumb rule. At the beginning of the simulation, the beam is assumed straight and coincident with the Y-axis in the negative $\hat{j}$ direction and having a constant angular velocity, denoted as 'ω'; which rotates about the global Z-axis. The simulation is stopped at an angle at which contact between the beam tip and the flat rigid surface is lost.

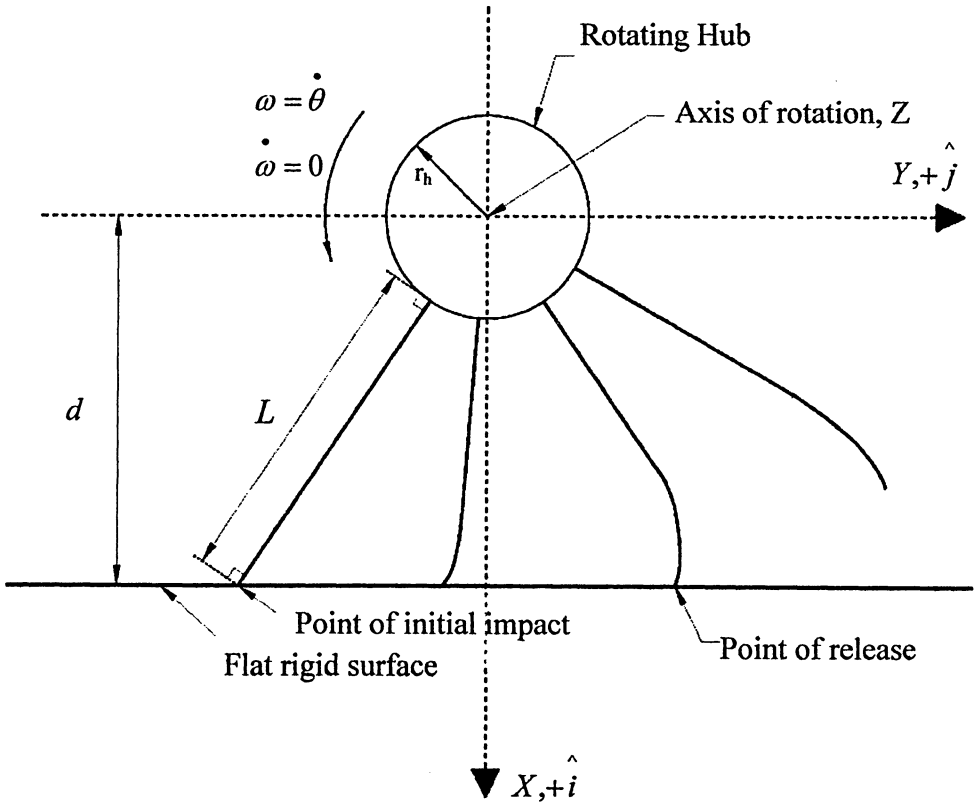

Figure 2: System process schematic.

Chapter Four: Governing Equations of the System and Numerical Algorithm

This chapter develops the governing equations of motion for the system described in the previous chapter. Lagrange equations are used to model system energy, while Euler-Bernoulli beam theory and the mode shapes of a non-rotating cantilever beam are used to describe beam displacement. Impact is modelled using the momentum balance method and contact is represented via the Lagrange multiplier and Coulomb friction. Hatman *et al.* [8] and [9] have already derived these equations and the purpose of including this section is for completeness and to fill in any blanks where the transition from one step to another was not immediately obvious. Also, modifications to the numerical algorithm are noted.

4.1 Position, Velocity, and Kinetic Energy Terms

The local and global position vectors of the system described in chapter three are represented by **r** and **R**, respectively:

$$\mathbf{r} = \{x, y, z\}^T \tag{4-1}$$

$$\mathbf{R} = \{X, Y, Z\}^T \tag{4-2}$$

The global position vector can be calculated using the transformation matrix **A**, where θ is the angular displacement of the rotating beam:

$$\mathbf{R} = \mathbf{A} \bullet \mathbf{r} \tag{4-3}$$

$$\mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4-4}$$

Velocity is obtained by differentiating Eqn. (4-3) to get:

$$\mathbf{V} = \dot{\mathbf{R}} = \dot{\mathbf{A}}\,\mathbf{r} + \mathbf{A}\,\dot{\mathbf{r}} \tag{4-5}$$

where the over-dot implies differentiation with respect to time. The kinetic energy of the beam is then expressed as:

$$T = \frac{1}{2}\int_v \dot{\mathbf{R}}^T \dot{\mathbf{R}}\, dm \tag{4-6}$$

$$T = \frac{1}{2}\int_v \left(\mathbf{r}^T \dot{\mathbf{A}}^T + \dot{\mathbf{r}}^T \mathbf{A}^T\right)\left(\dot{\mathbf{A}}\mathbf{r} + \mathbf{A}\dot{\mathbf{r}}\right) dm \tag{4-7}$$

$$T = \frac{1}{2}\int_v \left(\mathbf{r}^T \dot{\mathbf{A}}^T \dot{\mathbf{A}}\mathbf{r} + 2\dot{\mathbf{r}}^T \mathbf{A}^T \dot{\mathbf{A}}\mathbf{r} + \dot{\mathbf{r}}^T \mathbf{A}^T\mathbf{A}\dot{\mathbf{r}}\right) dm \tag{4-8}$$

Let $\omega = \dot{\theta}$; the angular velocity, and the transformation multiplications are as follows:

$$\dot{\mathbf{A}}^T \dot{\mathbf{A}} = \omega^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{A}^T \dot{\mathbf{A}} = \omega \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{A}^T \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4-9a-c}$$

The time derivative of the coordinate vector in the moving frame is expressed as:

$$\dot{\mathbf{r}} = \frac{\partial \mathbf{r}}{\partial q_i}\dot{q}_i = \Phi \dot{\mathbf{q}} \tag{4-10}$$

where,

$$\Phi = \begin{bmatrix} \dfrac{\partial x}{\partial q_1} & \cdots & \dfrac{\partial x}{\partial q_k} & \cdots & \dfrac{\partial x}{\partial q_n} \\ \dfrac{\partial y}{\partial q_1} & \cdots & \dfrac{\partial y}{\partial q_k} & \cdots & \dfrac{\partial y}{\partial q_n} \\ \dfrac{\partial z}{\partial q_1} & \cdots & \dfrac{\partial z}{\partial q_k} & \cdots & \dfrac{\partial z}{\partial q_n} \end{bmatrix} \tag{4-11}$$

and the vector of the time derivative of the generalized displacement with respect to time is:

$$\dot{\mathbf{q}} = \left\{\dot{q}_1, \dot{q}_2, \ldots \dot{q}_n\right\}^T \tag{4-12}$$

The resulting kinetic energy term is:

$$T = \frac{1}{2}\int_v \left(\omega^2 \mathbf{r}_p^T \mathbf{r}_p + 2\omega \dot{\mathbf{q}}^T \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_p + \dot{\mathbf{q}}^T \Phi^T \Phi \dot{\mathbf{q}}\right) dm \tag{4-13}$$

where $\mathbf{r}_p = \{x, y\}^T$. This notation is developed because the third row and the third column of the first two transformation multiplications are all zeroes. The following equations are developed for use in the Lagrange equation:

$$\frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial T}{\partial \mathbf{q}} + \frac{\partial U}{\partial \mathbf{q}} = \mathbf{Q} \tag{4-14}$$

where T and U are the kinetic and potential energies, respectively, $\mathbf{Q}$ is the non-conservative generalized force, and $\mathbf{q}$ is the generalized displacement vector.

The energy term derivation continues:

$$\frac{\partial T}{\partial \mathbf{q}} = \frac{1}{2}\int_v \left(\omega^2 \frac{\partial}{\partial \mathbf{q}}\left(\mathbf{r}_p^T \mathbf{r}_p\right) + 2\omega \frac{\partial}{\partial \mathbf{q}}\left\{\dot{\mathbf{q}}^T \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_p\right\} + \frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \Phi^T \Phi \dot{\mathbf{q}}\right)\right) dm \tag{4-15}$$

$$\frac{\partial T}{\partial \dot{\mathbf{q}}} = \frac{1}{2}\int_v \left(2\omega \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_p + 2\Phi^T \Phi \dot{\mathbf{q}}\right) dm \tag{4-16}$$

$$\frac{\partial T}{\partial \dot{\mathbf{q}}} = \int_v \left(\omega \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_p^T + \Phi^T \Phi \dot{\mathbf{q}}\right) dm \tag{4-17}$$

The mass matrix **M** is expressed as:

$$\mathbf{M} = \int_v \partial\mathbf{r}^T \partial\mathbf{r}\, dm \tag{4-18}$$

Using the above definition, the time derivative of equation (4-17) is:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\mathbf{q}}}\right) = \int_v \left(\omega \frac{d}{dt}\left(\Phi_p^T\right)\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\mathbf{r}_p^T + \omega\Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\Phi_p \dot{\mathbf{q}}\right) dm + \dot{\mathbf{M}}\dot{\mathbf{q}} + \mathbf{M}\ddot{\mathbf{q}} \tag{4-19}$$

Therefore,

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial T}{\partial \mathbf{q}} =$$

$$= \mathbf{M}\ddot{\mathbf{q}} + \dot{\mathbf{M}}\dot{\mathbf{q}} + \int_v \left(\omega \frac{d}{dt}\left(\Phi_p^T\right)\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\mathbf{r}_p^T + \omega\Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\Phi_p \dot{\mathbf{q}}\right.$$

$$\left. - \frac{1}{2}\omega^2 \frac{\partial}{\partial \mathbf{q}}\left(\mathbf{r}_p^T \mathbf{r}_p\right) - \omega \frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\mathbf{r}_p\right) - \frac{1}{2}\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \Phi^T \Phi \dot{\mathbf{q}}\right)\right) dm \tag{4-20}$$

$$= \mathbf{M}\ddot{\mathbf{q}} + \dot{\mathbf{M}}\dot{\mathbf{q}} - \frac{1}{2}\int_v \frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \Phi^T \Phi \dot{\mathbf{q}}\right) dm - \frac{1}{2}\int_v \omega^2 \frac{\partial}{\partial \mathbf{q}}\left(\mathbf{r}_p^T \mathbf{r}_p\right) dm$$

$$+ \int_v \left(\omega \frac{d}{dt}\left(\Phi_p^T\right)\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\mathbf{r}_p^T - \omega \frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\mathbf{r}_p\right)\right) dm + \int_v \omega\Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\Phi_p \dot{\mathbf{q}}\, dm \tag{4-21}$$

To simplify the equations, some equations are manipulated. The second term in the above equation is expanded:

$$\dot{\mathbf{M}}\dot{\mathbf{q}} = \frac{d}{dt}\int_v \Phi^T \Phi dm \dot{\mathbf{q}} \tag{4-22}$$

$$\dot{\mathbf{M}}\dot{\mathbf{q}} = \int_v \frac{d}{dt}(\Phi^T)\Phi dm \dot{\mathbf{q}} + \int_v \Phi^T \frac{d}{dt}(\Phi) dm \dot{\mathbf{q}} \tag{4-23}$$

$$\dot{\mathbf{M}}\dot{\mathbf{q}} = \int_v \frac{d\mathbf{q}}{dt}^T \frac{\partial(\Phi^T)}{\partial \mathbf{q}} \Phi dm \dot{\mathbf{q}} + \int_v \Phi^T \frac{d}{dt}(\Phi) dm \dot{\mathbf{q}} \tag{4-24}$$

$$\dot{\mathbf{M}}\dot{\mathbf{q}} = \frac{1}{2}\int_v \frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \Phi^T \Phi \dot{\mathbf{q}}\right) dm + \int_v \Phi^T \frac{d}{dt}(\Phi) dm \dot{\mathbf{q}} \tag{4-25}$$

$$\dot{\mathbf{M}}\dot{\mathbf{q}} - \frac{1}{2}\int_v \frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \Phi^T \Phi \dot{\mathbf{q}}\right) dm = \int_v \Phi^T \frac{d}{dt}(\Phi) dm \dot{\mathbf{q}} \tag{4-26}$$

And then the sixth term of equation (4-21) is expanded:

$$\omega \frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_p \right) = \omega \frac{\partial}{\partial \mathbf{q}} \frac{d\mathbf{q}}{dt} \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_p + \omega \dot{\mathbf{q}}^T \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \Phi_p \tag{4-27}$$

$$= \omega \frac{d}{dt}\left(\Phi_p^T\right) \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_p + \omega \Phi_p^T \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \Phi_p \dot{\mathbf{q}} \tag{4-28}$$

$$\omega \frac{d}{dt}\left(\Phi_p^T\right) \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_p - \omega \frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_p \right) = \omega \Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \Phi_p \dot{\mathbf{q}} \tag{4-29}$$

Also, the fourth term of equation (4-21) is rewritten as:

$$-\frac{1}{2}\omega^2 \int_v \frac{\partial}{\partial \mathbf{q}}\left(\mathbf{r}_p^T \mathbf{r}_p\right) dm = -\omega^2 \int_v r_p^T \frac{\partial}{\partial \mathbf{q}}\left(r_p\right) dm \tag{4-30}$$

$$-\omega^2 \int_v r_p^T \frac{\partial}{\partial \mathbf{q}}\left(r_p\right) dm = -\omega^2 \int_v [x \quad y] \begin{bmatrix} \frac{\partial x}{\partial \mathbf{q}} \\ \frac{\partial y}{\partial \mathbf{q}} \end{bmatrix} dm = -\omega^2 \int_v \left(x \frac{\partial x}{\partial \mathbf{q}} + y \frac{\partial y}{\partial \mathbf{q}}\right) dm \tag{4-31}$$

Therefore:

$$\frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial T}{\partial \mathbf{q}} = \mathbf{M}\ddot{\mathbf{q}} + 2\omega\left(\int_v \left(\Phi_p^T\right)\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \Phi_p dm\right)\dot{\mathbf{q}}$$

$$+\left(\int_v \left[\Phi^T \frac{\partial}{\partial t}(\Phi)\right] dm\right)\dot{\mathbf{q}} - \omega^2 \int_v \left(x \frac{\partial x}{\partial \mathbf{q}} + y \frac{\partial y}{\partial \mathbf{q}}\right) dm \tag{4-32}$$

Let **G** and **D** denote the following matrices:

$$\mathbf{G} = \int_v \left(\Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \Phi_p\right) dm \tag{4-33}$$

and

$$\mathbf{D} = \int_v \left(\Phi^T \frac{\partial}{\partial t}(\Phi)\right) dm \tag{4-34}$$

Matrix **D** becomes all zeroes if the functions that describe the coordinates are linear with respect to the generalized displacement. As can be seen in the following section, this is not the case, because the local x coordinate is quadratic with respect to the generalized displacement. Finally, the kinetic energy terms in the Lagrange equation are derived:

$$\frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial T}{\partial \mathbf{q}} = \mathbf{M}\ddot{\mathbf{q}} + 2\omega\mathbf{G}\dot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} - \omega^2 \int_v \left(x \frac{\partial x}{\partial \mathbf{q}} + y \frac{\partial y}{\partial \mathbf{q}}\right) dm \tag{4-35}$$

4.2 Generalized Coordinates and Further Equation Manipulation

The coordinates are represented as:

$$x = r_h + \int_0^s dx \tag{4-36}$$

where r$_h$ is the hub radius and s is any point along the beam.

But $ds = \sqrt{dx^2 + dy^2}$, hence $dx^2 = ds^2 - dy^2$. It follows that:

$$dx = \sqrt{ds^2 - dy^2} = \sqrt{1-\left(\frac{dy}{ds}\right)^2}\, ds \tag{4-37}$$

Using the binomial series [20]:

$$\sqrt{1-\left(\frac{dy}{ds}\right)^2} \cong \left(1-\frac{1}{2}\left(\frac{dy}{ds}\right)^2\right) \tag{4-38}$$

Therefore,

$$x(s) = r_h + \int_0^s \left(1-\frac{1}{2}\left(\frac{dy}{ds}\right)^2\right) ds = r_h + s - \frac{1}{2}\int_0^s \left(\frac{dy}{ds}\right)^2 ds \tag{4-39}$$

To find the transverse direction displacement field, the cantilever modes are derived. The boundary conditions for a cantilever beam are as follows [21]:

The clamped end: deflection $y(0,t) = 0$, and slope of deflection $\frac{\partial y(0,t)}{\partial s} = 0$ (4-40-1)

The free end: bending moment $M(L,t) = EI(L)\frac{\partial^2 y(L,t)}{\partial s^2} = 0$, (4-42) and shearing force $Q(L,t) = -\frac{\partial}{\partial s}\left[EI(L)\frac{\partial^2 y(L,t)}{\partial s^2}\right] = 0$ (4-43)

Using the general eigenfunction equation [22],

$$S_i(s) = \cosh\beta_i s - \cos\beta_i s - \sigma_i(\sinh\beta_i s - \sin\beta_i s) \tag{4-44}$$

where,

$$\sigma_i = \frac{(\sinh\beta_i L - \sin\beta_i L)}{(\cosh\beta_i L + \cos\beta_i L)} \tag{4-45}$$

and L is the length of the beam. The β_i 's can be found by solving the characteristic equation:

$$\cos\beta_i L \cosh\beta_i L = -1 \tag{4-46}$$

and are related to the natural frequencies of the beam, $(\omega_n)_i$, via the expression:

$$\beta_i^2 = (\omega_n)_i\sqrt{\frac{\rho A}{EI}} \tag{4-47}$$

The generalized coordinates become:

$$x(s) = r_h + s - \frac{1}{2}\mathbf{q}^T[\mathbf{ISP}(s)]_{ij}\mathbf{q} \tag{4-48}$$

and

$$y(s) = [\mathbf{S}(s)]^T\mathbf{q} \tag{4-49}$$

where,

$$[\mathbf{ISP}(s)]_{ij} = \int_0^s\left(\frac{d\mathbf{S}_i(s)}{ds}\right)\left(\frac{d\mathbf{S}_j(s)}{ds}\right)ds \tag{4-50}$$

and

$$\mathbf{S}(s) = \{S_1(s), S_2(s), S_3(s) \ldots S_n(s)\}^T \tag{4-51}$$

where 'n' is the number of modes being used. Letting $r_i = \beta_i L$ and $\xi = s/L$, where ξ is a dimensionless point along the beam measured from where the beam is attached to the rotating hub.

$$[\mathbf{ISP}(\xi)]_{ij} = L[\mathbf{ISP}(s)]_{ij} = \int_0^{\xi} \left(\frac{d\mathbf{S}_i(\xi)}{d\xi} \right) \left(\frac{d\mathbf{S}_j(\xi)}{d\xi} \right) d\xi \tag{4-52}$$

$$S_i(\xi) = \cosh r_i\xi - \cos r_i\xi - \sigma_i(\sinh r_i\xi - \sin r_i\xi) \tag{4-53}$$

and,

$$\sigma_i = \frac{(\sinh r_i - \sin r_i)}{(\cosh r_i + \cos r_i)} \tag{4-54}$$

The local coordinates then become:

$$x(\xi) = r_h + \xi L - \frac{1}{2L} \mathbf{q}^T [\mathbf{ISP}(\xi)]_{ij} \mathbf{q} \tag{4-55}$$

and

$$y(\xi) = [\mathbf{S}(\xi)]^T \mathbf{q} \tag{4-56}$$

Now, the partial derivative of the local coordinates with respect to the generalized coordinates can be expressed as:

$$\Phi(\xi) = \begin{bmatrix} -(1/L)\mathbf{q}^T[\mathbf{ISP}(\xi)] \\ \mathbf{S}^T(\xi) \\ 0 \end{bmatrix} \tag{4-57}$$

Using the above notation, the inertia matrix becomes:

$$\mathbf{M} = \int_0^L \Phi^T(s)\Phi(s)\rho A ds \tag{4-58}$$

$$\mathbf{M} = \rho AL\int_0^1 \Phi^T(\xi)\Phi(\xi)d\xi = \rho AL\int_0^1 \begin{bmatrix} -\left(\frac{1}{L}\right)[\mathbf{ISP}(\xi)]^T\mathbf{q} & \mathbf{S}^T(\xi) & 0 \end{bmatrix} \begin{bmatrix} -\left(\frac{1}{L}\right)\mathbf{q}^T[\mathbf{ISP}(\xi)] \\ \mathbf{S}^T(\xi) \\ 0 \end{bmatrix} d\xi \tag{4-59}$$

$$\mathbf{M} = \frac{\rho A}{L}\int_0^1 [\mathbf{ISP}(\xi)]^T\mathbf{q}\mathbf{q}^T[\mathbf{ISP}(\xi)]d\xi + \rho AL\int_0^1 \mathbf{S}(\xi)\mathbf{S}^T(\xi)d\xi \tag{4-60}$$

where ρ is the beam density, and A is the beam cross-sectional area. **D**, the deformation induced damping matrix, becomes:

$$\mathbf{D} = \int_v \left(\Phi^T \frac{\partial}{\partial t}(\Phi)\right)dm \tag{4-61}$$

$$\mathbf{D} = \rho AL\int_0^1 \begin{bmatrix} \left(\frac{-1}{L}\right)[\mathbf{ISP}(\xi)]^T\mathbf{q} & \mathbf{S}(\xi) & 0 \end{bmatrix} \begin{bmatrix} \left(\frac{-1}{L}\right)[\mathbf{ISP}(\xi)]\dot{\mathbf{q}}^T \\ 0 \\ 0 \end{bmatrix} d\xi \tag{4-62}$$

$$\mathbf{D} = \frac{\rho A}{L}\int_0^1 [\mathbf{ISP}(\xi)]^T\mathbf{q}\dot{\mathbf{q}}^T[\mathbf{ISP}(\xi)]d\xi \tag{4-63}$$

G, the gyroscopic damping matrix, becomes:

$$\mathbf{G} = \int_v \left(\Phi_p^T \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \Phi_p\right)dm \tag{4-64}$$

$$\mathbf{G} = \rho AL\int_0^1\left(\begin{bmatrix} -\left(\frac{1}{L}\right)[\mathbf{ISP}(\xi)]^T\mathbf{q} & \mathbf{S}(\xi)\end{bmatrix}\begin{bmatrix} 0 & -1 \\ 1 & 0\end{bmatrix}\begin{bmatrix} -\left(\frac{1}{L}\right)\mathbf{q}^T[\mathbf{ISP}(\xi)] \\ \mathbf{S}^T(\xi)\end{bmatrix}\right)d\xi \tag{4-65}$$

$$\mathbf{G} = \rho AL\int_0^1\left(\begin{bmatrix} -\left(\frac{1}{L}\right)[\mathbf{ISP}(\xi)]^T\mathbf{q} & \mathbf{S}(\xi)\end{bmatrix}\begin{bmatrix} -\mathbf{S}^T(\xi) \\ -\left(\frac{1}{L}\right)\mathbf{q}^T[\mathbf{ISP}(\xi)]\end{bmatrix}\right)d\xi \tag{4-66}$$

$$\mathbf{G} = \rho AL\int_0^1\left(\left(\frac{1}{L}\right)[\mathbf{ISP}(\xi)]^T\mathbf{q}\mathbf{S}^T(\xi) - \left(\frac{1}{L}\right)\mathbf{S}(\xi)\mathbf{q}^T[\mathbf{ISP}(\xi)]\right)d\xi \tag{4-67}$$

$$\mathbf{G} = \rho A\int_0^1\left([\mathbf{ISP}(\xi)]^T\mathbf{q}\mathbf{S}^T(\xi) - \mathbf{S}(\xi)\mathbf{q}^T[\mathbf{ISP}(\xi)]\right)d\xi \tag{4-68}$$

Also,

$$-\omega^2\int_v\left(x\frac{\partial x}{\partial \mathbf{q}} + y\frac{\partial y}{\partial \mathbf{q}}\right)dm =$$

$$-\rho AL\omega^2\left(\int_0^1\left(r_h + \xi L - \frac{1}{2L}\mathbf{q}^T[\mathbf{ISP}(\xi)]_{ij}\mathbf{q}\right)\left(-\frac{1}{L}\mathbf{q}^T[\mathbf{ISP}(\xi)]\right)d\xi + \int_1^1\left(\mathbf{q}^T\mathbf{S}(\xi)\right)\left(\mathbf{S}^T(\xi)\right)d\xi\right) \tag{4-69}$$

$$= -\rho AL\omega^2\left(-\int_0^1\frac{r_h}{L}\mathbf{q}^T[\mathbf{ISP}(\xi)]d\xi - \int_0^1\xi\mathbf{q}^T[\mathbf{ISP}(\xi)]d\xi\right.$$

$$\left. + \frac{1}{2L^2}\int_0^1\left(\mathbf{q}^T[\mathbf{ISP}(\xi)]\mathbf{q}\right)\left(\mathbf{q}^T[\mathbf{ISP}(\xi)]\right)d\xi + \int_0^1\mathbf{q}^T\mathbf{S}(\xi)\mathbf{S}^T(\xi)d\xi\right) \tag{4-70}$$

4.3 Potential Energy Term

The potential energy is found by integrating the bending moment over the angle of the tangent to the beam. Only the bending part is considered because the inextensibility assumption is in use [8].

$$U = \tfrac{1}{2}\int_0^L M(s)d\phi = \tfrac{1}{2}\int_0^L M(s)\kappa(s)ds \tag{4-71}$$

where M(s) is the bending moment, κ(s) is the curvature of the beam, and ϕ is the angle of the tangent to the beam with the OX-axis. According to Euler-Bernoulli beam theory, "the bending moment in every cross section is proportional to the curvature" [8]:

$$U = \tfrac{1}{2}\int_0^L EI\kappa^2(s)ds \tag{4-72}$$

When curvature is being expressed in terms of arc length, it can be defined as [23]:

$$\kappa(s) = \|\mathbf{r}''\| = \left\|\frac{d^2\mathbf{r}}{ds^2}\right\| \tag{4-73}$$

Therefore, the potential energy is:

$$U = \frac{1}{2}EI\int_0^L\left(\left(\frac{d^2x}{ds^2}\right)^2 + \left(\frac{d^2y}{ds^2}\right)^2\right)ds = \frac{1}{2}\frac{EI}{L^3}\int_0^1\left(\left(\frac{d^2x}{d\xi^2}\right)^2 + \left(\frac{d^2y}{d\xi^2}\right)^2\right)d\xi \tag{4-74}$$

$$\frac{\partial U}{\partial \mathbf{q}} = \frac{EI}{L^3}\int_0^1\left(\frac{d^2x}{d\xi^2}\bullet\frac{d}{d\mathbf{q}}\left(\frac{d^2x}{d\xi^2}\right) + \frac{d^2y}{d\xi^2}\bullet\frac{d}{d\mathbf{q}}\left(\frac{d^2y}{d\xi^2}\right)\right)d\xi \tag{4-75}$$

Let

$$[\mathbf{SP}(\xi)]_{ij} = \int_0^\xi\left(\frac{d\mathbf{S}_i(\xi)}{d\xi}\right)\left(\frac{d\mathbf{S}_j(\xi)}{d\xi}\right)d\xi \tag{4-76}$$

and

$$[\mathbf{SPD}(\xi)]_{ij} = \frac{d}{d\xi}\left([\mathbf{SP}(\xi)]_{ij}\right) \tag{4-77}$$

Calculating the terms in equation (4-75):

$$\frac{d^2x(\xi)}{d\xi^2} = -\frac{1}{2L}\mathbf{q}^T \frac{d}{d\xi}\left([\mathbf{SP}(\xi)]_{ij}\right)\mathbf{q} = -\frac{1}{2L}\mathbf{q}^T\left([\mathbf{SPD}(\xi)]_{ij}\right)\mathbf{q} \tag{4-78}$$

$$\frac{d^2y(\xi)}{d\xi^2} = \mathbf{q}^T \frac{d^2\mathbf{S}(\xi)}{d\xi^2} \tag{4-79}$$

Finally, the potential energy term is:

$$\frac{\partial U}{\partial \mathbf{q}} = \frac{EI}{L^3}\int_0^1 \left(\left(-\left(\frac{1}{2L}\right)\mathbf{q}^T[\mathbf{SPD}(\xi)]\mathbf{q}\right)\left(-\left(\frac{1}{L}\right)\mathbf{q}^T[\mathbf{SPD}(\xi)]\right) + \left(\mathbf{q}^T\frac{d^2\mathbf{S}(\xi)}{d\xi^2}\right)\left(\frac{d^2[\mathbf{S}(\xi)]^T}{d\xi^2}\right)\right)d\xi \tag{4-80}$$

Let $\Psi\left(\mathbf{q},\dot{\mathbf{q}},t\right)$ equal all the energy terms except for the inertia matrix:

$$\Psi\left(\mathbf{q},\dot{\mathbf{q}},t\right) = 2\omega\mathbf{G}\,\dot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} - \omega^2\int_v\left(x\frac{\partial x}{\partial \mathbf{q}} + y\frac{\partial y}{\partial \mathbf{q}}\right)dm + \frac{\partial U}{\partial \mathbf{q}} \tag{4-81}$$

The equation of motion for the unconstrained portion of the process is:

$$\mathbf{M}\,\ddot{\mathbf{q}} + \Psi\left(\mathbf{q},\dot{\mathbf{q}},t\right) = 0 \tag{4-82}$$

4.4 Surface Contact Constrained Motion Terms

The constraint causes the global X term to be equal to the distance between the centre of the hub and the rigid surface.

$$X = x(L)\cos\theta - y(L)\sin\theta = d \tag{4-83}$$

Therefore, the constraint equation is defined as:

$$C(\mathbf{q},t) = x(\xi = 1)\cos\theta - y(\xi = 1)\sin\theta - d = 0 \tag{4-84}$$

$$C(\mathbf{q},t) = (r_h + L - (1/2L)\mathbf{q}^T[\mathbf{ISP}(1)]\mathbf{q})\cos\theta - \mathbf{q}^T\mathbf{S}(1)\sin\theta - d = 0 \tag{4-85}$$

The two terms that describe the constrained motion are the generalized normal force and the generalized friction force. The generalized normal force is expressed in terms of a Lagrange multiplier multiplied with the gradient of the constraint manifold (with respect to the generalized coordinates).

$$\mathbf{N} = \lambda grad\mathbf{C} \tag{4-86}$$

where **N** is the normal force applied to the beam in order to maintain the constraint condition. The magnitude of **N** represents the normal force needed to enforce the constraint on the system, while the sign of the Lagrange multiplier gives the direction of the force along the normal to the constraint manifold.

The generalized friction force is expressed as:

$$\mathbf{Q}_f = \mathbf{F} \bullet \frac{d\mathbf{R}_f}{dq} \tag{4-87}$$

where '**F**' is the friction force and the subscript 'f' refers to the tip of the beam. Using Lagrange's relations of the first kind, the following is true:

$$\frac{d\mathbf{R}_f}{dq} = \frac{d\mathbf{V}_f}{d\dot{q}} \tag{4-88}$$

Therefore,

$$\mathbf{Q}_f = \mathbf{F} \bullet \frac{d\mathbf{V}_f}{dq} \tag{4-89}$$

The friction force is defined by Coulomb's law, and is expressed as:

$$\mathbf{F} = -\mu|\mathbf{N}|\frac{\mathbf{V}_f}{|\mathbf{V}_f|} \tag{4-90}$$

where μ is the coefficient of friction. Manipulating the generalized force vector:

$$\mathbf{Q}_f = -\mu|\mathbf{N}|\frac{\mathbf{V}_f}{|\mathbf{V}_f|}\frac{d\mathbf{V}_f}{d\dot{q}} = -\frac{\mu}{2}\frac{|\mathbf{N}|}{|\mathbf{V}_f|}\frac{d(\mathbf{V}_f^2)}{d\dot{q}} = -\frac{\mu}{2}\frac{|\mathbf{N}|}{\sqrt{\mathbf{V}_f^T\mathbf{V}_f}}\frac{d(\mathbf{V}_f^T\mathbf{V}_f)}{d\dot{q}} \tag{4-91}$$

Substituting Eqn. (4-86) into Eqn. (4-91), the result is:

$$\mathbf{Q}_f = -|\lambda|\frac{\mu}{2}\frac{|grad\mathbf{C}|}{\sqrt{\mathbf{V}_f^T\mathbf{V}_f}}\frac{d(\mathbf{V}_f^T\mathbf{V}_f)}{d\dot{q}} = |\lambda|\mathbf{Q}_f^* \tag{4-92}$$

where,

$$\mathbf{Q}_f^* = -\frac{\mu}{2}\frac{|grad\mathbf{C}|}{\sqrt{\mathbf{V}_f^T\mathbf{V}_f}}\frac{d(\mathbf{V}_f^T\mathbf{V}_f)}{d\dot{q}} \tag{4-93}$$

Since

$$\mathbf{V}_f = \dot{\mathbf{R}}_f = \dot{\mathbf{A}}\,\mathbf{r}_f + \mathbf{A}\,\dot{\mathbf{r}}_f \tag{4-94}$$

has the same form as the velocity used for the kinetic energy, the square of the velocity will be very similar to the kinetic energy derivation.

$$\mathbf{V}_f^T\mathbf{V}_f = \omega^2\mathbf{r}_f^T\begin{bmatrix}1&0&0\\0&1&0\\0&0&0\end{bmatrix}\mathbf{r}_f + 2\omega\dot{\mathbf{q}}^T\,\Phi^T{}_{(\xi=1)}\begin{bmatrix}0&-1&0\\1&0&0\\0&0&0\end{bmatrix}\mathbf{r}_f + \dot{\mathbf{q}}^T\,\Phi^T{}_{(\xi=1)}\Phi_{(\xi=1)}\,\dot{\mathbf{q}}$$

(4-95)

$$\frac{d}{d\dot{\mathbf{q}}}\left(\mathbf{V}_f^T\mathbf{V}_f\right) = 2\omega\Phi^T{}_{(\xi=1)}\begin{bmatrix}0&-1&0\\1&0&0\\0&0&0\end{bmatrix}\mathbf{r}_f + 2\Phi^T{}_{(\xi=1)}\Phi_{(\xi=1)}\,\dot{\mathbf{q}} \qquad (4\text{-}96)$$

The constrained equation of motion thus far is:

$$\mathbf{M}\ddot{\mathbf{q}} + \Psi\left(\mathbf{q},\dot{\mathbf{q}},t\right) = \lambda grad\mathbf{C} + |\lambda|\mathbf{Q}_f^* \qquad (4\text{-}97)$$

4.5 Impact

The constrained system can now be described using the following set of equations:

$$\mathbf{M}\ddot{\mathbf{q}} + \Psi\left(\mathbf{q},\dot{\mathbf{q}},t\right) = \lambda grad\mathbf{C} + |\lambda|\mathbf{Q}_f^* \qquad C(\mathbf{q},t) = 0 \qquad (4\text{-}98a,b)$$

Referring to Eqn. (4-14) and examining equations (4-35), and (4-80-4-82) it is apparent that the RHS of Eqn. (4-98a) is the derived equivalent of the RHS of Eqn. (4-14) for this particular system. Integrating the equation of motion at impact, the equation becomes:

$$\int_{t_0}^{t_o+\tau}\frac{d}{dt}\left(\frac{\partial T}{\partial\dot{\mathbf{q}}}\right)dt - \int_{t_0}^{t_o+\tau}\frac{\partial T}{\partial\mathbf{q}}dt + \int_{t_0}^{t_o+\tau}\frac{\partial U}{\partial\mathbf{q}}dt = \int_{t_0}^{t_o+\tau}\lambda grad\mathbf{C}dt + \int_{t_0}^{t_o+\tau}|\lambda|\mathbf{Q}_f^*dt \qquad (4\text{-}99)$$

Since the position of the bristle is assumed constant during the duration of impact, d**q**=0. Also, it is assumed that the force does not do work because the beam tip does not move. Therefore, the second and third terms on the left-hand side and the last term on the right-hand side of the equation are set equal to zero. The equation is then reduced to:

$$\int_{t_0}^{t_o+\tau} \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\mathbf{q}}}\right) dt = \int_{t_0}^{t_o+\tau} \lambda grad\mathbf{C} dt \tag{4-100}$$

The left-hand side can be further reduced to:

$$\int_{t_0}^{t_o+\tau} \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\mathbf{q}}}\right) dt = \mathbf{M}\int_{t_0}^{t_o+\tau} \ddot{\mathbf{q}}\, dt = \mathbf{M}\int_{0}^{\tau} \frac{d\dot{\mathbf{q}}}{dt} dt = \mathbf{M}\Delta\dot{\mathbf{q}} \tag{4-101}$$

where $\Delta\dot{\mathbf{q}}$ is the jump discontinuity in velocity. It is the equivalent of subtracting the pre-impact generalized velocity from the post-impact generalized velocity. The right hand-side can be written as:

$$\int_{t_0}^{t_o+\tau} \lambda grad\mathbf{C} dt = grad\mathbf{C}\int_{0}^{\tau} \lambda dt = Pgrad\mathbf{C} \tag{4-102}$$

where P is the impulsive force. Therefore, the resulting momentum equation is:

$$\mathbf{M}_o\Delta\dot{\mathbf{q}} = Pgrad\mathbf{C}_o \tag{4-103}$$

where subscript 'o' denotes the initial configuration and $\Delta\dot{\mathbf{q}}$ is the change in generalized velocity. This particular term is also called the 'jump velocity' because it is discontinuous.

The coefficient of restitution is defined as the negative ratio of the post-impact velocity to the pre-impact velocity:

$$e = -\frac{\dot{\mathbf{q}}_{(+0)}}{\dot{\mathbf{q}}_{(-0)}} \tag{4-104}$$

Since the generalized velocity after impact is unknown, it is necessary to develop an equation for the jump velocity in terms of the known parameters: the generalized velocity before impact and the generalized coordinates: (The derivation of the following equation can be found in Appendix A)

$$grad\mathbf{C}_0^T \Delta \dot{\mathbf{q}} = -(1+e)grad\mathbf{C}_0^T\left(\dot{\mathbf{q}}_{(-0)}\right) - (1+e)\frac{\partial \mathbf{C}_0}{\partial t} \tag{4-105}$$

The impulse force, P, can be deduced from Eqns. (4-103) and (4-105), which can then be used to obtain the jump velocity expression:

$$\Delta \dot{\mathbf{q}} = \frac{-(1+e)grad\mathbf{C}_0^T\left(\dot{\mathbf{q}}_{(-0)}\right) - (1+e)\frac{\partial \mathbf{C}_0}{\partial t}}{grad\mathbf{C}_0^T \mathbf{M}_0^{-1} grad\mathbf{C}_0}\left(\mathbf{M}_0^{-1} grad\mathbf{C}_0\right) \tag{4-106}$$

Using Yigit *et al.*'s [5] definition of the coefficient of restitution:

$$e = 0.501 v^{-(1/4)} \tag{4-107}$$

where v is the normal component of the tip velocity of the beam.

4.6 Numerical Routine for Constrained Motion

There are many numerical methods available in the literature; however, choosing an applicable one can be a challenge. The system defined by (4-98) has an index of three [1]. The index of a system is defined as "the number of times that the algebraic equations of the system must be differentiated in order to obtain a standard form of ordinary differential equation (ODE) system" [24]. Index three systems are difficult to solve, and, therefore, it has been suggested that they be transformed into an index two system to be solved. This is accomplished by differentiating the constraint equation twice and solving the Lagrange multiplier in terms of the generalized coordinates and velocities. The second time derivative of the constraint equation is:

$$\ddot{C} = \mathbf{G}^T \ddot{\mathbf{q}} + \dot{\mathbf{q}}^T \frac{\partial C}{\partial \mathbf{q}^T} \dot{\mathbf{q}} + 2\dot{\mathbf{q}}^T \frac{\partial^2 C}{\partial \mathbf{q} \partial t} + \frac{\partial^2 C}{\partial t^2} = 0 \tag{4-108}$$

Solving for $\ddot{\mathbf{q}}$, substituting it into Eqn. (4-98a) and then collecting the terms containing the Lagrange multiplier to one side, the following equation is obtained:

$$\mathbf{G}^T\mathbf{M}^{-1}\mathbf{G}\lambda + \mathbf{G}^T\mathbf{M}^{-1}\mathbf{Q}_f^{*}|\lambda| = \mathbf{G}^T\mathbf{M}^{-1}\Psi - \dot{\mathbf{q}}^T \frac{\partial C}{\partial \mathbf{q}^T}\dot{\mathbf{q}} - 2\dot{\mathbf{q}}^T \frac{\partial^2 C}{\partial \mathbf{q}\partial t} - \frac{\partial^2 C}{\partial t^2} \tag{4-109}$$

Obtaining λ from Eqn. (4-109) and inserting it into Eqn. (4-98a) can allow for the resulting equation to be solved using a Runge-Kutta method. This, however, was found to be unstable. Hatman *et al.* [9] attribute this instability to the fact that having the constraint equation and it's first two derivatives equal zero means that the value of C is always equal to zero. Numerically, this does not happen and solution drift occurs. A way to resolve this problem was proposed [25]. However, it was also found that drift still occurs if the constraint equation is explicitly dependent on time [26]. Therefore, Eqns. (4-108-4-109) are used just to predict the generalized coordinates and velocities in the predictor-corrector algorithm presented next.

To avoid confusion, this algorithm is kept separate from all the other equations being derived thus far by denoting the generalized coordinates by a bold lowercase '**x**' and denoting the generalized velocities with a bold lowercase '**u**'. Thus, the following equations represent the present system equations of motion:

$$\dot{\mathbf{x}} = \mathbf{u}, \qquad \mathbf{M}\dot{\mathbf{u}} = -\Psi(\mathbf{x},\mathbf{u},t) + \lambda\mathbf{G} + |\lambda|\mathbf{Q}_f^{*}, \qquad C(\mathbf{x},t) = 0 \tag{4-110a-c}$$

Using the algorithm proposed by Gear *et al.* [1], a new index two system needs to be developed, whose solution is also a solution of the index three system. As suggested in [1], a term will be added to Eqn. (4-110a) that will introduce nonzero projections of the time derivative of the generalized coordinate vector along the gradient. Also, another constraint is added that makes the total time derivative of the constraint equation equal to zero. This is to mathematically ensure that the constraint equals zero [9]. The system described takes the following form:

$$\dot{\mathbf{x}} = \mathbf{u} + \delta\mathbf{G}, \qquad \mathbf{M}\dot{\mathbf{u}} = -\Psi(\mathbf{x},\mathbf{u},t) + \lambda\mathbf{G} + |\lambda|\,\mathbf{Q}_f^{*}, \qquad \text{(4-111a-b)}$$

$$C(\mathbf{x},t) = 0, \qquad \mathbf{G}^T \bullet \mathbf{u} + \frac{\partial C}{\partial t} = 0 \qquad \text{(4-111c-d)}$$

The generalized coordinates and velocities, the Lagrange multiplier, and δ are the unknowns. The solution to system Eqns. (4-111) is the same as the solution of system Eqns. (4-110) when δ is equal to zero.

Before presenting the algorithm, it is important to be able to solve an equation for a variable, if both the variable and the variable's magnitude are present. From Eqn. (4-109), it is difficult to factor out λ because its sign is unknown. However, the sign of λ is the same whether or not friction is present in the system [6], [12]. Therefore, if the friction term disappears, λ can be solved for. The sign of this solution can be used to find the real λ that exists when friction is present. Going back to Eqn. (4-110), $|\lambda|$ will be replaced with $\lambda sign(\lambda)$, and finally, the actual λ can be isolated.

The following is the implementation of the predictor-corrector algorithm (which is a modified version of Hatman *et al.*'s work [9]):

Step (1): When contact first occurs, assume that the system has reached time t_{n-1} and values $\mathbf{x}_{n-1}$ and $\mathbf{u}_{n-1}$. Then use Eqn. (4-109) to make a prediction for lambda, λ, as explained above. Then do a Runge-Kutta iteration with this new λ, and t_{n-1}, $\mathbf{x}_{n-1}$, and $\mathbf{u}_{n-1}$ for the constrained system.

Step (2): Correct the predicted value for **x**, by solving the system formed by equations (4-111a, c) and by setting δ equal to zero. This is done by using the following backward differentiation formula (BDF) for the time derivative of the generalized coordinates:

$$\dot{\mathbf{x}} = (\mathbf{x}_n - \mathbf{x}_{n-1})/h \tag{4-112}$$

where h is the constant time step. The Newton-Raphson iterative method is used to solve for the unknown δ term. Performing only one corrector iteration, and starting with δ equal to zero:

$$\delta_i = -\frac{C(x_{n-1} + h\mathbf{u}_n^p)}{\mathbf{G}^T(x_{n-1} + h\mathbf{u}_n^p) \bullet \mathbf{G}(x_{n-1} + h\mathbf{u}_n^p)} \tag{4-113}$$

The new value for **x** can be computed as:

$$\mathbf{x}_n = \mathbf{x}_{n-1} + h\mathbf{u}_n^p - \frac{C(x_{n-1} + h\mathbf{u}_n^p)}{\mathbf{G}^T(x_{n-1} + h\mathbf{u}_n^p) \bullet \mathbf{G}(x_{n-1} + h\mathbf{u}_n^p)} \mathbf{G}(x_{n-1} + h\mathbf{u}_n^p, t_n) \tag{4-114}$$

(See Section 6.2 for the full derivation).

Step (3): The generalized velocities, **u**, can be corrected by solving the system formed by equations (4-111 b, d). Using the same BDF mentioned in Step (2) for the time derivative of **u**, the result is:

$$\mathbf{u}_n = \mathbf{u}_{n-1} - h\mathbf{M}^{-1}(\mathbf{x}_n, t_n) \bullet \Psi(\mathbf{x}_n, \mathbf{u}_n^p, t_n) + h\lambda_n \mathbf{M}^{-1}(\mathbf{x}_n, t_n) \bullet \mathbf{G}(\mathbf{x}_n, t_n)$$

$$+ h|\lambda_n| \mathbf{M}^{-1}(\mathbf{x}_n, t_n) \bullet \mathbf{Q}_f^*(\mathbf{x}_n, \mathbf{u}_n^p, t_n) \tag{4-115}$$

where λ_n is the value of the Lagrange multiplier at the time moment t_n. It is currently an unknown value, but it can be calculated by substituting the above expression for $\mathbf{u}_n$ in equation (4-115):

$$h\lambda_n \mathbf{G}^T(\mathbf{x}_n, t_n) \bullet \mathbf{M}^{-1}(\mathbf{x}_n, t_n) \bullet \mathbf{G}(\mathbf{x}_n, t_n) + h|\lambda_n| \mathbf{G}^T(\mathbf{x}_n, t_n) \bullet \mathbf{M}^{-1}(\mathbf{x}_n, t_n) \mathbf{Q}_f^*(\mathbf{x}_n, \mathbf{u}_n^p, t_n)$$

$$= h\mathbf{G}^T(\mathbf{x}_n, t_n) \bullet \mathbf{M}^{-1}(\mathbf{x}_n, t_n) \bullet \Psi(\mathbf{x}_n, \mathbf{u}_n^p, t_n) - \mathbf{G}^T(\mathbf{x}_n, t_n) \bullet \mathbf{u}_{n-1} - \frac{\partial C}{\partial t} \tag{4-116}$$

where,

$$\frac{\partial C}{\partial t} = -\mathbf{G}^T \mathbf{u}_n^p \tag{4-117}$$

Note that Eqn. (4-117) must be solved as outlined above because of the presence of the magnitude of λ_n.

Step (4): Let $\mathbf{x}_n^p = \mathbf{x}_n$ and $\mathbf{u}_n^p = \mathbf{u}_n$, and repeat steps (2) and (3) until the value of λ_n converges. Hatman *et al.* [9] found that five repetitions were usually sufficient if the time step was small enough. Using this modified routine, anywhere from two to four repetitions was the norm.

Step (5): Go to the next time step.

4.7 Algorithm for Entire Problem

The algorithms for the rigid and flexible cases slightly differ to save computational time. For the latter, even the first unconstrained motion must be computed with a small time step to avoid large errors due to the variable coefficient system equations. For the rigid case, however, the time at which contact is first made can be computed using simple geometry and the angular velocity. This is only possible for the first time Step (1) is completed. The generalized vectors are both equal to zero, for this case, until starting Step (2).

Step (1): For the very first time this step is carried out, the beam begins coincident with the global Y-axis with its tip pointing towards the negative $\hat{j}$ direction. For the first rigid case calculation, find the time contact occurs using geometry and the angular velocity. For the flexible case, run the Runge-Kutta routine until the constraint equation becomes equal to zero or if it becomes positive. If it is positive, do a linear interpolation over the last time step to find out the time moment C(**q**,t) is equal to zero. Using this time, update the solution for the generalized vectors. The resulting generalized vectors become the initial conditions for the impact phase of the system motion.

Step (2): When the beam first comes into contact with the flat surface, the jump velocity vector can be calculated from Eqn. (4-106) and added to the latest generalized velocity vector. The generalized coordinate vector remains the same because there is no tip displacement at the instant of impact. These generalized vectors are now the initial conditions for the constrained motion phase.

Step (3): Predict the new generalized co-ordinates and velocities using one Runge-Kutta iteration and perform the correction using steps (1) to (5) in Section 4.6.

Step (4): Take the sign of the Lagrange multiplier.

Step (4a): If the Lagrange multiplier is negative or zero, the beam is in contact with the surface. Update the solution with the values of the generalized co-ordinates and velocities resulting from the corrector iteration, and the value of the Lagrange multiplier. Return to step (3).

Step (4b): If the Lagrange multiplier is positive, the beam tip has lost contact. Do a linear interpolation over the last time step to find out the instant when the Lagrange multiplier becomes zero.

If this is the Lagrange multiplier that occurred when contact was first made, i.e., there was no previous negative multiplier, the system has gone into bounce back (where the jump velocity is big enough to change the direction of the beam tip) and contact has been lost. The interpolation can still be carried through because the result will be equivalent to the time and generalized vectors that occur at the end of impact and a Lagrange multiplier equal to zero. This is valid because the positive multiplier does not have any real meaning.

Now, update the solution with the time when the Lagrange multiplier becomes zero, the corresponding (interpolated) values for the generalized co-ordinates and velocities and a zero value for the Lagrange multiplier. Do an unconstrained motion prediction keeping the same small time step and evaluate the tip distance.

Step (5): Take the sign of the tip distance.

Step (5a): If the sign of the tip distance is positive, it means that the tip of the beam penetrates the rigid surface. Since this is not possible, no solution update is made. Perform a number of unconstrained system Runge-Kutta iterations until the tip distance becomes negative [12]. Update this last solution and continue on to Step (6).

Step (5b): If the tip distance is negative, the motion is now unconstrained. Update the solution with the predicted values for the generalized co-ordinates and velocities, and a zero value for the Lagrange multiplier. Continue on to Step (6).

Step (6): This step is similar to Step (1); however, to avoid confusion, it will be restated. The system has entered into unconstrained motion once again. The initial conditions for this phase are the last updated results from Step (5). Keep doing unconstrained system Runge-Kutta iterations until $C(\mathbf{q},t)$ is greater than or equal to zero. At this point, do a linear interpolation over the last time step to find the instant when $C(\mathbf{q},t)$ is equal to zero. Update the solution and return to Step (2).

Since the time that the beam tip and the flat rigid surface have lost all contact is unknown, allow the simulation to run until the clamped side of the beam has reached such an angle that the tip could not possibly still be in contact. Trial and error can be used to find an appropriate angle. Also, it is helpful to watch the global position of the beam tip as the simulation nears the angle to make sure that the values are far enough away form the rigid surface. This thesis uses an angle of 90° to the positive X-axis.

Figure 3 contains a flow chart that describes the algorithm for the entire problem. Refer to Appendix B for the Matlab program codes associated with the algorithms presented.

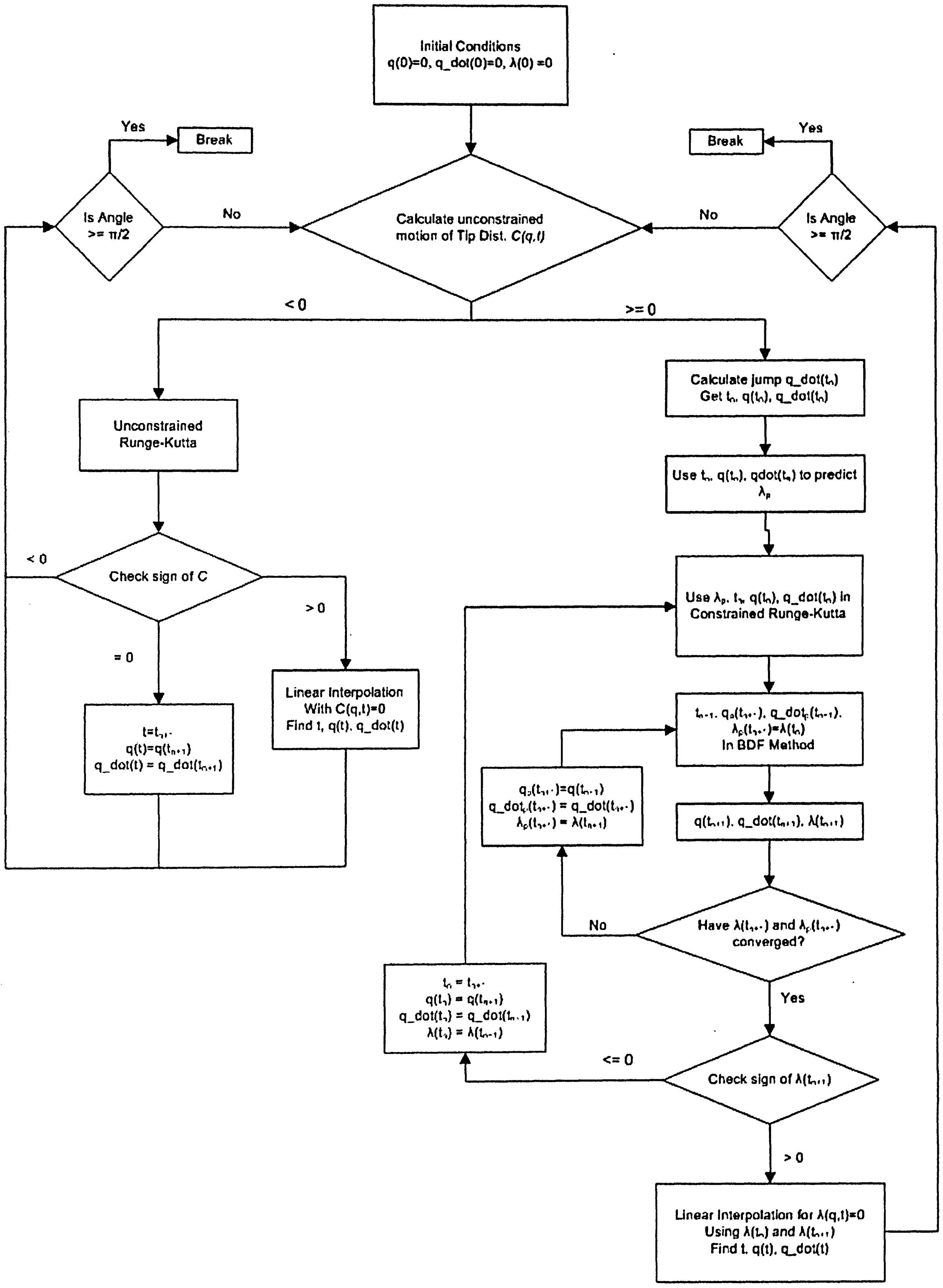
Initial Conditions
q(0)=0, q_dot(0)=0, λ(0) =0
Yes
Break
Is Angle >= π/2
No
Calculate unconstrained motion of Tip Dist. C(q,t)
No
Is Angle >= π/2
Yes
Break
< 0
>= 0
Unconstrained Runge-Kutta
Calculate jump q_dot(tₙ) Get tₙ, q(tₙ), q_dot(tₙ)
Use tₙ, q(tₙ), qdot(tₙ) to predict λₚ
< 0
Check sign of C
> 0
Use λₚ, tₙ, q(tₙ), q_dot(tₙ) in Constrained Runge-Kutta
= 0
t=tₙ₊₁
q(t)=q(tₙ₊₁)
q_dot(t) = q_dot(tₙ₊₁)
Linear Interpolation With C(q,t)=0
Find t, q(t), q_dot(t)
tₙ₊₁, qₚ(tₙ₊₁), q_dotₚ(tₙ₊₁), λₚ(tₙ₊₁)=λ(tₙ)
In BDF Method
qₚ(tₙ₊₁)=q(tₙ₊₁)
q_dotₚ(tₙ₊₁) = q_dot(tₙ₊₁)
λₚ(tₙ₊₁) = λ(tₙ₊₁)
q(tₙ₊₁), q_dot(tₙ₊₁), λ(tₙ₊₁)
No
Have λ(tₙ₊₁) and λₚ(tₙ₊₁) converged?
Yes
tₙ = tₙ₊₁
q(tₙ) = q(tₙ₊₁)
q_dot(tₙ) = q_dot(tₙ₊₁)
λ(tₙ) = λ(tₙ₊₁)
<= 0
Check sign of λ(tₙ₊₁)
> 0
Linear Interpolation for λ(q,t)=0
Using λ(tₙ) and λ(tₙ₊₁)
Find t, q(t), q_dot(t)

Figure 3: Algorithm flow chart for entire problem.

Chapter Five: Numerical Simulations

This chapter presents the simulation results. The results were obtained using a computer code written for a Matlab platform. Five modes are used and the numerical integrations are implemented by the Gaussian Quadrature method, with sixteen abscissas and weight coefficients [27]. Unlike the approach in Ref. [6], a constant time-step was used for the unconstrained motion. This is because the initial beam generalized coordinates and velocities are known to equal zero for the rigid beam initial configuration. Only the time to reach impact was needed to be calculated. Once the generalized coordinates and velocities had values, which happens for unconstrained motion within the contact region, or for an initial flexible beam configuration, it was necessary to use a small time step to avoid large numerical errors. These errors arise because both the generalized values and the coefficient matrices in the equations of motion are time dependent. The largest time-step for any type of motion is 1 μs. Once inside the constrained motion, the time-step is decreased further, if necessary, to better handle the intermittent changes in types of motion.

Table 5-1 lists the dimensions that are common to all simulations. The mechanical properties of steel are obtained from Ref. [11] because they are unspecified in Ref. [6].

Table 5-1: Dimensions used for simulations.

Parameter	Value	Units
Length of beam	0.0635	m
Radius of hub	0.0635	m
Beam cross-section diameter	5.08E-4	m
Density of steel	7800	kg/m^3
Elastic modulus of steel	2.2E11	Pa

The normal distance between the hub centre and the flat rigid surface is denoted by 'd'. The depth of penetration is the difference between 'd' and the summation of the hub radius and the length of the beam. It is expressed as a percentage of the beam length. The values of the varying parameters, see Table 5-2, are those of Ref. [6] and are chosen because these are the typical ranges of values that a steel wire brush undergoes.

Next, the following graphs are obtained from the simulations. In the figures, the ordinate axis (i.e., the y-axis) represents the normal force needed to enforce the geometrical constraint, and these values are calculated as the magnitude of the Lagrange multiplier multiplied by the

Table 5-2: Changing parameters.

Parameter	Values	Units
Angular velocity	550, 1000, 2000, 3000	rpm
Depth of penetration	2, 4, 6, 8	percent
Friction coefficient	0.1, 0.3, 0.4, 0.5, 0.7	--
Initial generalized velocity per mode	0.1	--

magnitude of the gradient of the constrained manifold with respect to the generalized coordinates (see Eqn. (4-86)). The abscissas in the figures represent the global Y-axis values of the system, as outlined in chapter three. Therefore, the value zero represents the global Y coordinate where the hub centre is located.

Figures 4 - 7 show the effect that changing angular velocities and depths of penetration have on the contact process of a single wire filament. The value of the friction coefficient used for all these simulations is 0.4. For Figure 4, 550 rpm was used with a range of depth of penetrations. Since the beam initial configuration at impact is of a rigid beam, the point of first

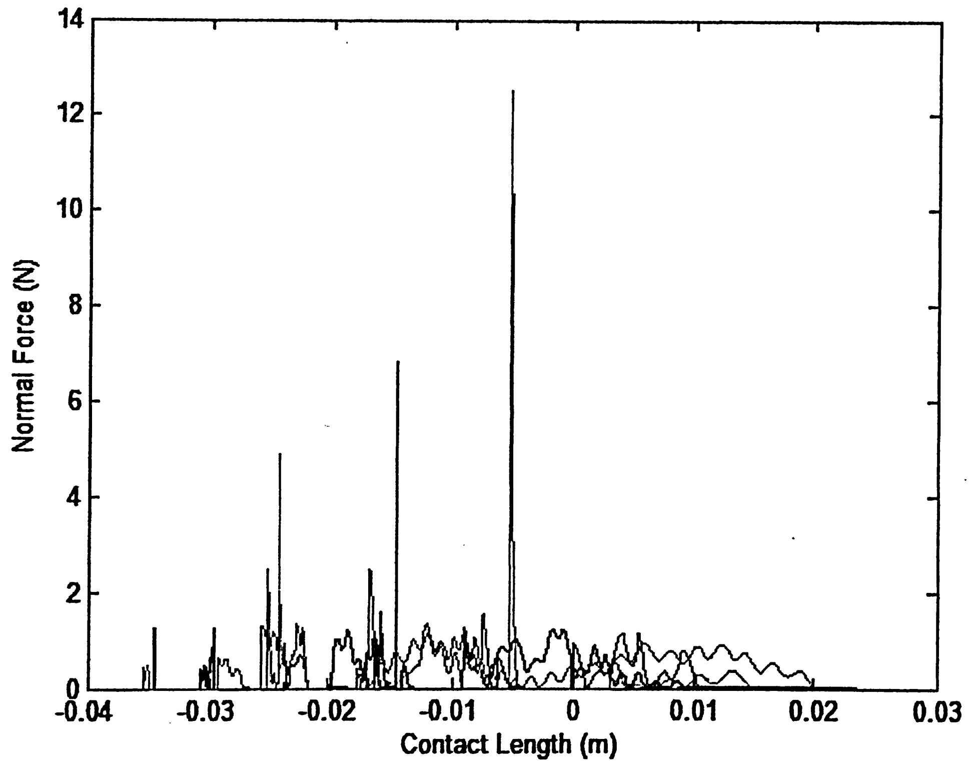

Figure 4: Normal force profiles for 550 rpm hub speed and depth of penetration of 2% (———), 4% (———), 6% (———), 8% (———).

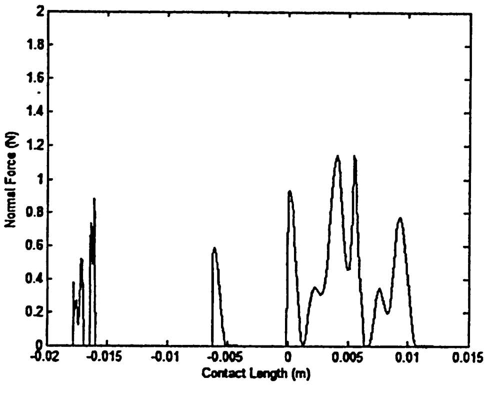

Figure 4(a): 2% depth penetration.

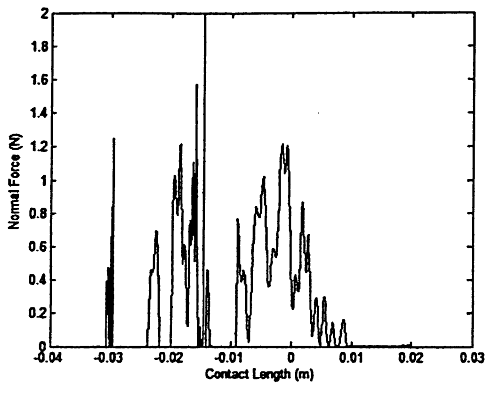

Figure 4(c): 6% depth penetration.

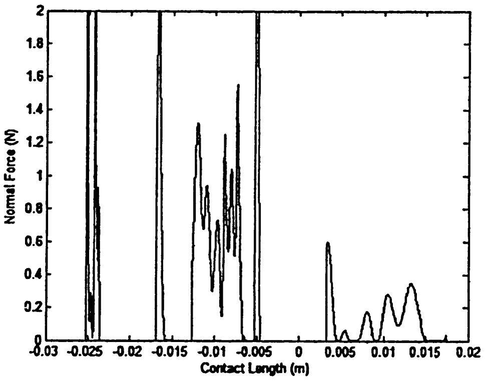

Figure 4(b): 4% depth penetration.

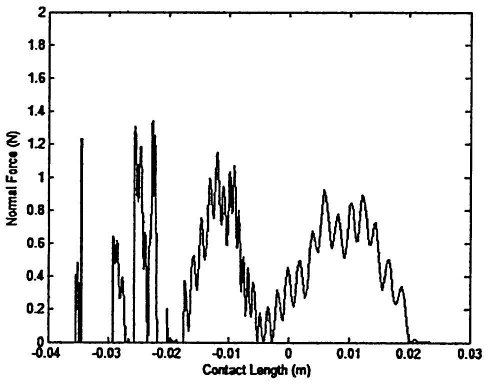

Figure 4(d): 8% depth penetration.

contact can be calculated geometrically. Looking at the figure and using the appropriate dimensions, it is evident that the initial impacts are occurring where they are supposed to be. A trend in how the depth of penetration affects the process cannot be found. Four percent penetration causes the largest forces, followed by six, eight and then two percent.

Figure 5 does the same comparison for a constant angular velocity of 3000 rpm. Looking at the two percent case, contact is beginning later than expected. Observing the simulations as they run, there is a noticeable occurrence where, after the initial impact occurs, the beam tip moves in the opposite direction to the original motion for a few moments, and then continues back in the original direction. This will be referred to as 'bounce back'. When the second impact occurs, the jump in velocity is not large enough to change the direction of the

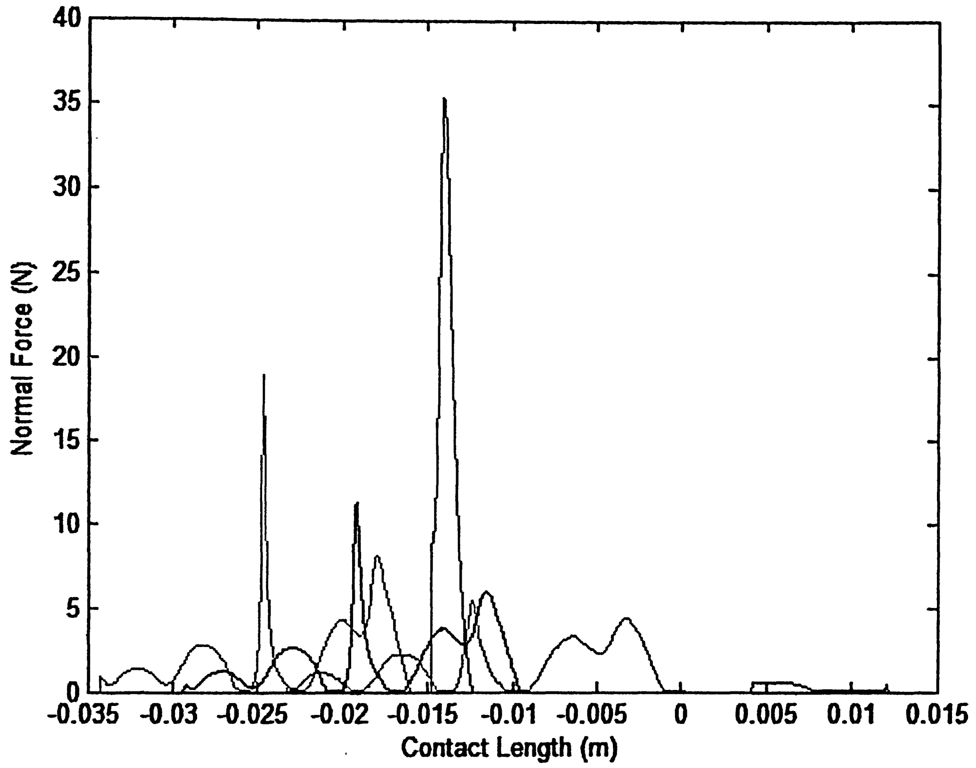

Figure 5: Normal force profiles for 3000 rpm hub speed and depth of penetration of 2% (———), 4% (———), 6% (———), 8% (———).

beam tip, so the beam continues the contact phase. Bounce back actually occurs in the 550 rpm case. However, it is so small that it is not noticeable in the graphed results. In Figure 5, there is not one trend that can describe the effect of the depth of penetration on the system process. It is expected that the faster the system, the larger the normal forces. However, if the two percent simulation is removed, the expected trend can be seen with the remaining results. The interesting observation about these plots is that they are very similar, except in magnitude and where they begin contact.

Figure 6 illustrates how the change in angular velocity affects a system with a two percent penetration. The graph is intuitively expected to show that the normal force increases with increasing angular velocity while the contact length decreases with increasing angular velocity.

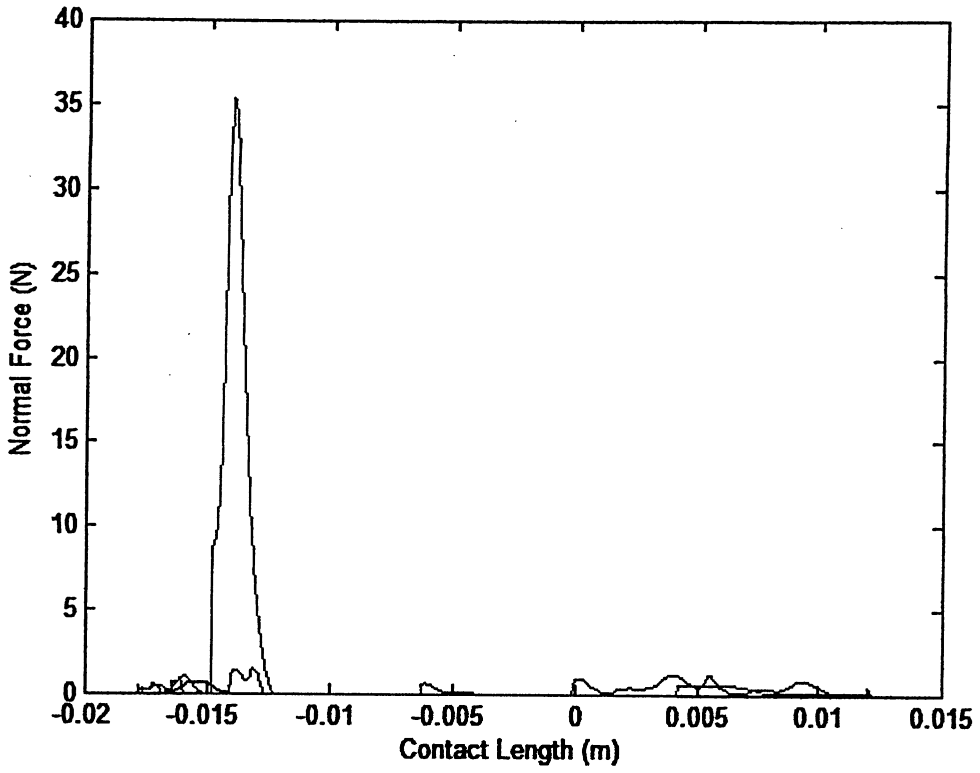

Figure 6: Normal force profiles for depth of penetration of 2% and hub speed of 550 rpm (———), 1000 rpm (———), 2000 rpm (———), 3000 rpm (———).

The 1000 rpm case has larger forces than the 2000 rpm case, but the other results are in accord with intuition. Also, the 3000 rpm case appears to have more contact length than the 1000 and 2000 rpm cases. The trend does, however, hold true for cases 550 to 2000 rpm.

Figure 7 does the same comparison as Figure 6, except for an eight percent penetration. The trend of larger normal forces is upheld. Also, the trend of longer contact lengths for smaller angular velocities is found for rpm values of 550 to 2000. The contact length for the 3000 rpm case is found between 1000 and 2000 rpm.

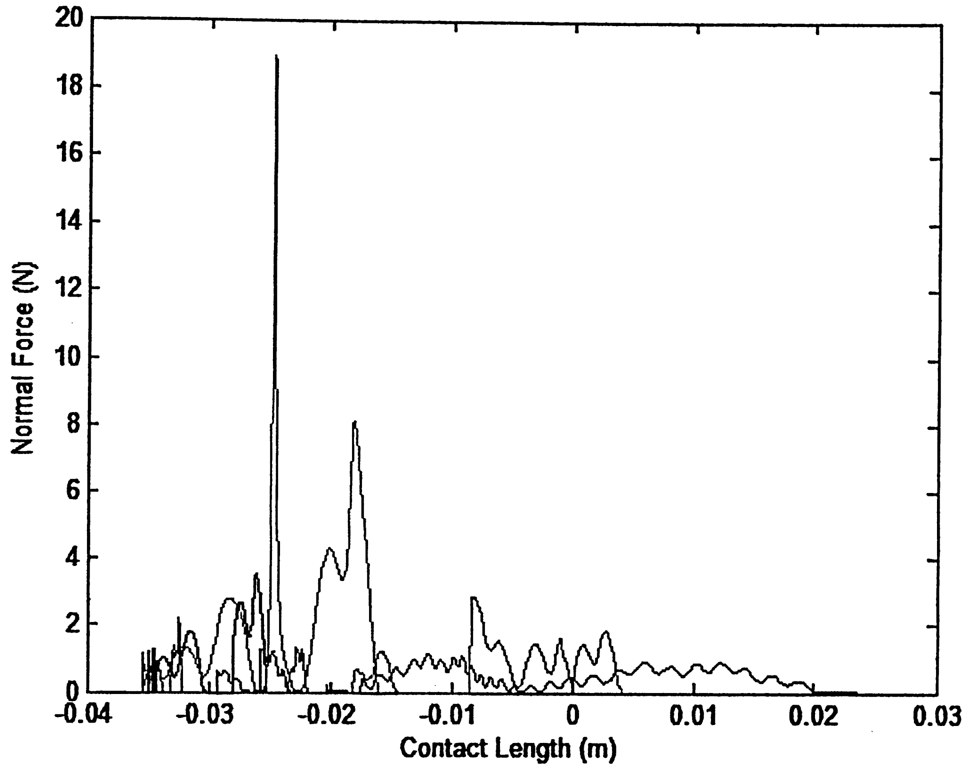

Figure 7: Normal force profiles for depth of penetration of 8% and hub speed of 550 rpm (———), 1000 rpm (———), 2000 rpm (———), 3000 rpm (———).

Figures 8 - 11 show the effect that the coefficient of friction has on the system results. It is expected that the profiles of the normal force against contact length will not change much in shape, just in the magnitude of the normal force. Figure 8 depicts results for a system with a hub speed of 550 rpm and a two percent penetration. It may be counter-intuitive to expect that the normal forces are larger for smaller coefficients of friction; however, if the friction force is reduced (with a smaller coefficient), the normal force must increase to keep the tip on the rigid surface [6]. The trend described here is followed in Figure 8 for the first part of the process, however, the pattern begins to unravel, especially with the coefficient equal to 0.3.

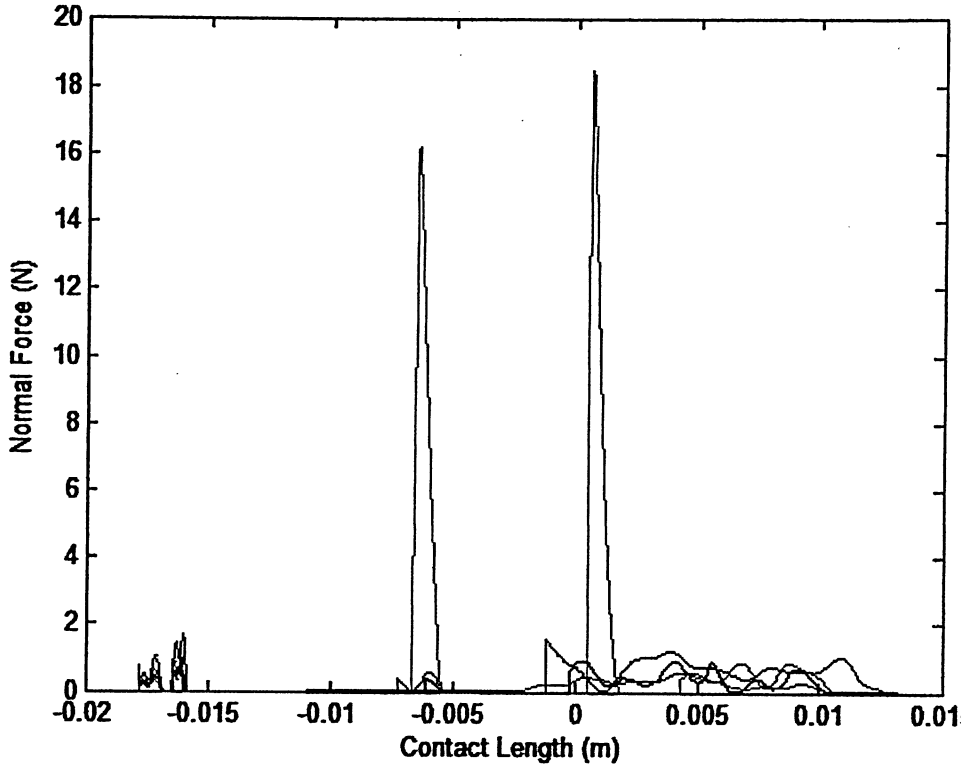

Figure 8: Normal force profiles for depth of penetration of 2%, hub speed of 550 rpm, and friction coefficient values of 0.1 (———), 0.3 (———), 0.5 (———), 0.7 (———).

The corresponding plots for the eight percent penetration are depicted in Figure 9. Again the trend described above is followed very closely for the first part of the process, then its pattern appears to unravel, and then again appears to follow the trend, albeit not as closely as the very first part. Figure 10 has results for a system with a hub speed of 3000 rpm and a two percent penetration. Basic profile shapes can be observed and only some coefficients follow the trend at different locations. Figure 11 is a result of 3000 rpm angular velocity and eight percent penetration. The same tendencies found in the previous three graphs can also be seen here.

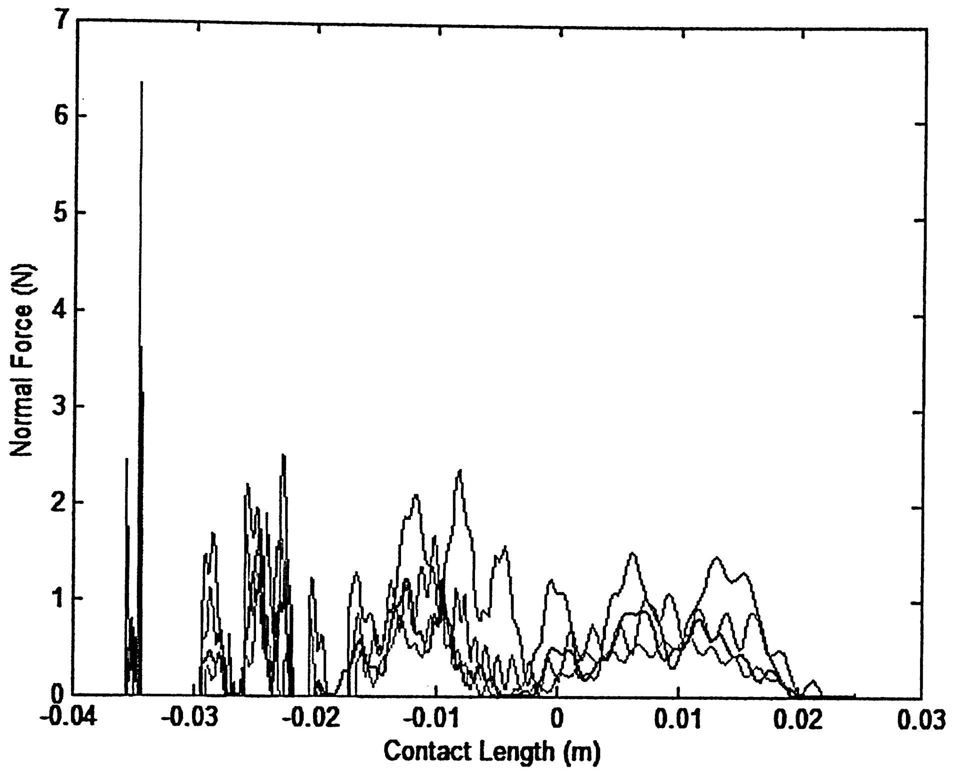

Figure 9: Normal force profiles for depth of penetration of 8%, hub speed of 550 rpm, and friction coefficient values of 0.1 (———), 0.3 (———), 0.5 (———), 0.7 (———).

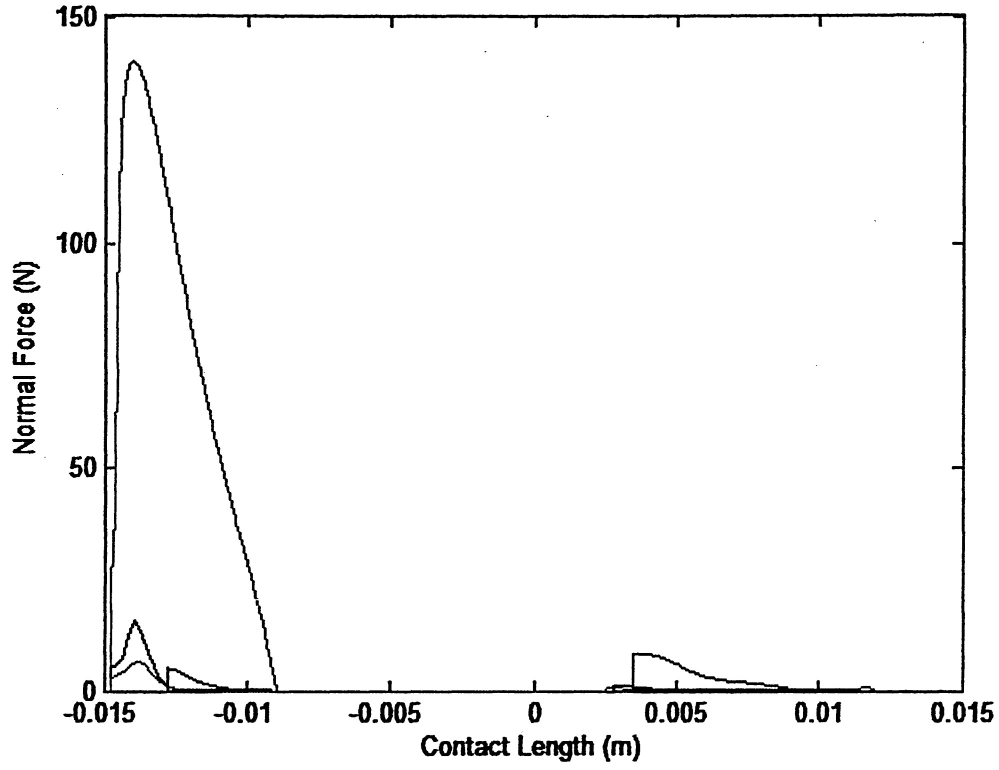

Figure 10: Normal force profiles for depth of penetration of 2%, hub speed of 3000 rpm, and friction coefficient values of 0.1 (———), 0.3 (———), 0.5 (———), 0.7 (———).

Figure 12 differs from the rest of the previous figures because, when initial impact occurs, the beam is not in a rigid configuration. An initial disturbance in the generalized velocity vector will allow the beam to vibrate like a flexible beam before initial impact occurs. The figure represents a system rotating at 550 rpm, with two percent depth of penetration, and a coefficient of friction of 0.4. A value of 0.1 s^{-1} is used once for each mode in five different simulations. The graph is dense; therefore, the modes will be graphed separately in the successive figures, Figures 12(a)-(e), representing the disturbances in modes 1 to 5, respectively. Looking at the legend for Figure 12, when the disturbance is put into the second mode, a large force appears, but will not fully appear in Figure 12(b) due to the range on the vertical axis scale.

Comparing Figure 4(a), which is the same system except with an initially rigid configuration, to Figures 12(a)-(e), it is evident that its profile is more consistent with that of the higher modes. Particularly, when the disturbance is included in modes four and five, Figures

12(d)-(e), the profile appears very similar, except for some slight differences in the magnitude of the normal force. This suggests that the higher modes are being excited in the simulation.

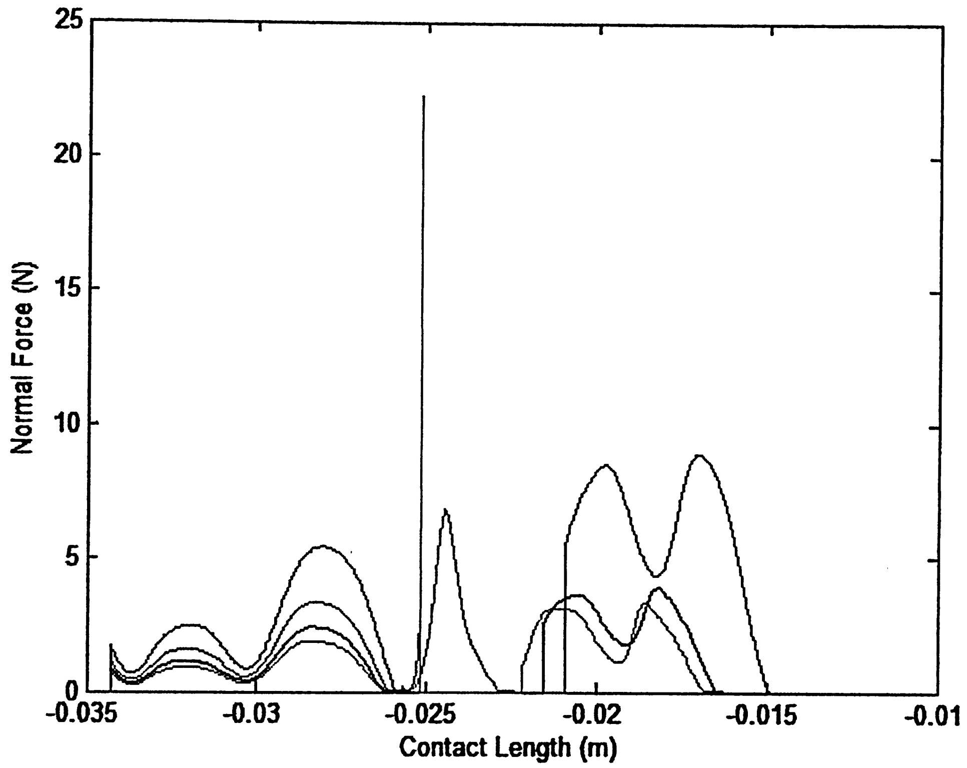

Figure 11: Normal force profiles for depth of penetration of 8%, hub speed of 3000 rpm, and friction coefficient values of 0.1 (———), 0.3 (———), 0.5 (———), 0.7 (———).

Figure 13 shows the global tip position throughout the entire process. However, the ordinate axis is multiplied by a negative unit vector. This is done to place the results in the third and fourth quadrants of a conventional coordinate system. The result gives a better sense of the true physical tip positioning. Therefore, multiply the X-axis position obtained from the graph by –1 to get the true value.

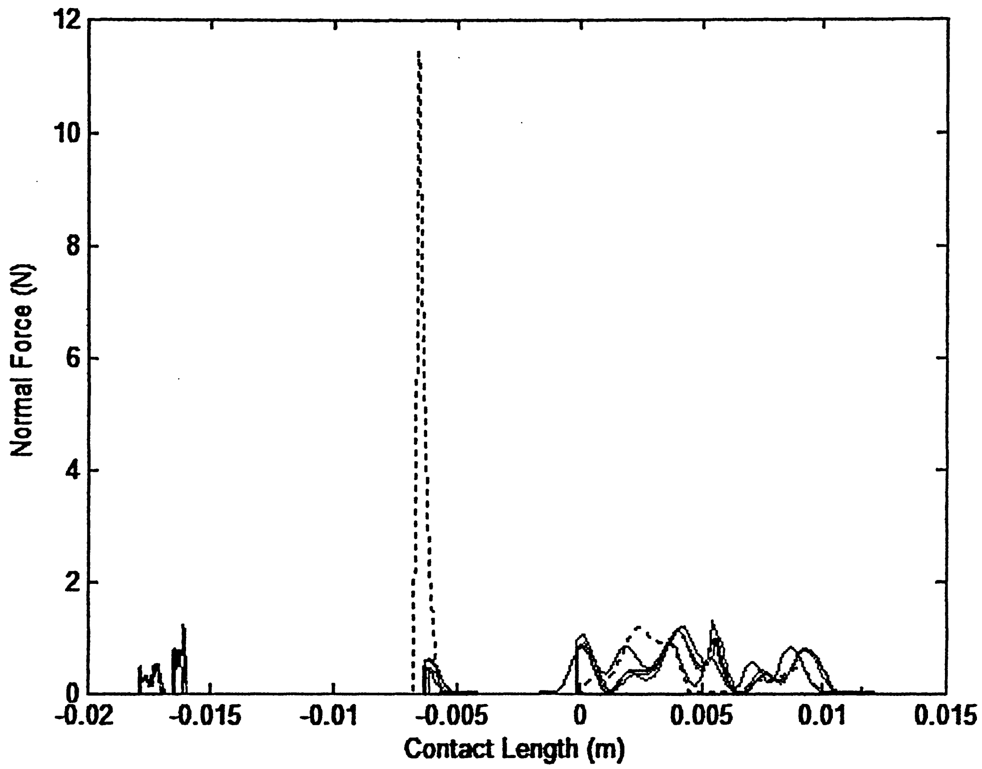

Figure 12: Normal force profiles for depth of penetration of 2%, hub speed of 550 rpm, friction value of 0.4, and flexible beam configuration before initial impact. Mode number with initial generalized velocity disturbance of 0.1 s^{-1}: 1 (———), 2 (·········), 3 (———), 4 (———), 5 (———).

Figure 13 represents the case where the angular velocity is 550 rpm, there is a two percent depth of penetration, and there is a friction coefficient of 0.4; however, the other cases have very similar occurrences. In Figure 13(a), most phases of the process can be seen: the location of the initial impact, approximately at (-0.0179, -0.1257) on the graph, constrained motion, characterized by a flat straight line, with intervals of unconstrained motion, and full unconstrained motion with two unexpected disturbances in beam tip positioning. A close up of these disturbances can be seen in Figure 13(b). This may be due to the excitations of higher mode shapes, or may just be due to numerical errors. Figure 13(c) is a close-up of the contact region, while part (d) is a close-up look at the bounce back that occurs at the time initial impact is over.

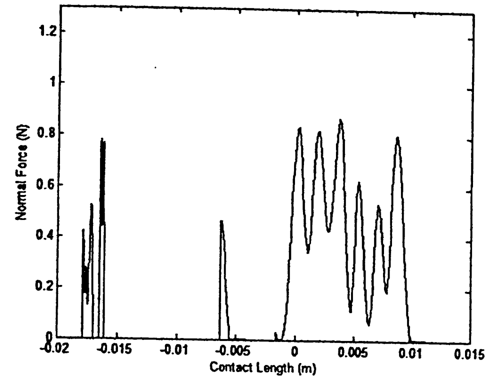

Figure 12(a): Disturbance in mode shape number 1.

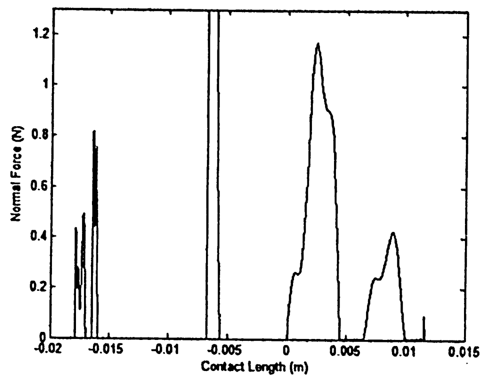

Figure 12(b): Disturbance in mode shape number 2.

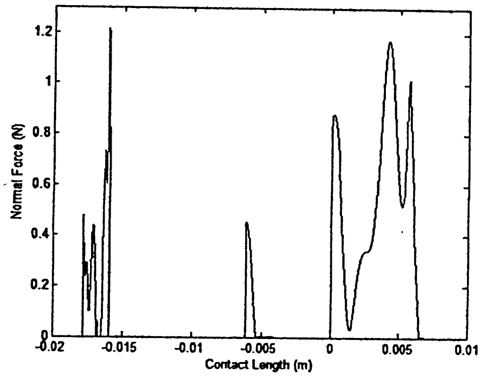

Figure 12(c): Disturbance in mode shape number 3.

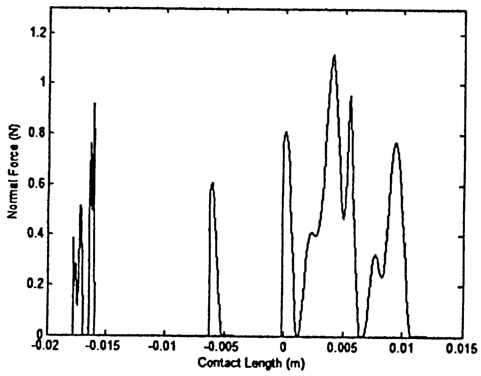

Figure 12(d): Disturbance in mode shape number 4.

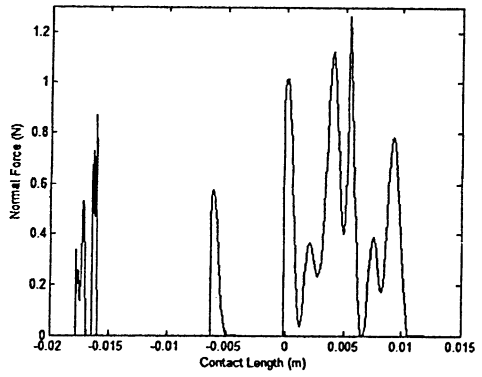

Figure 12(e): Disturbance in mode shape number 5.

According to Ref. [9], the flexibility of a beam causes the transition from unconstrained motion to impact to be numerically more gradual. The transitions are smoother with less change in process parameter values when compared to the rigid beam cases. Therefore, it is important to investigate the magnitude of the changes in velocity when impact occurs, to see if this

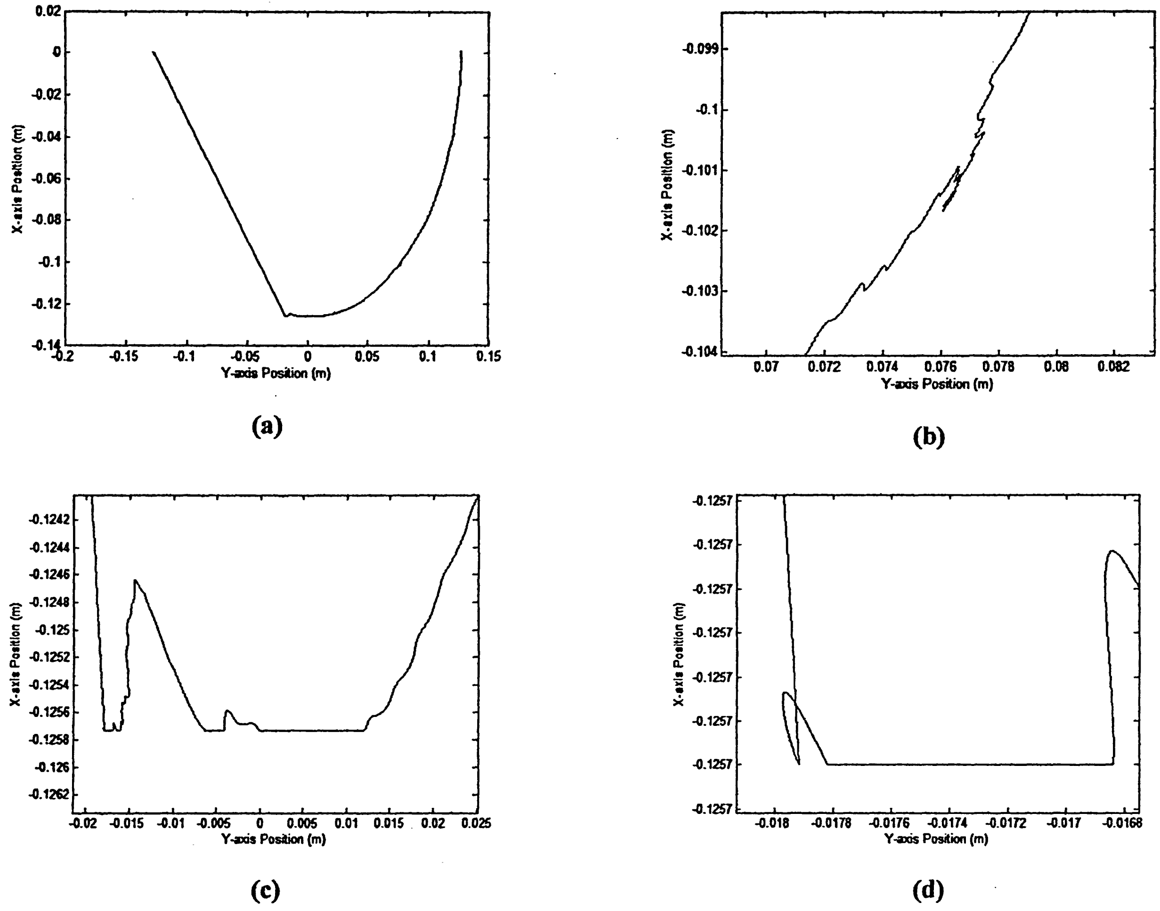

Figure 13: Displacement profile for depth of penetration of 2%, hub speed of 550 rpm and friction value of 0.4.

occurrence is captured. Table 5-3 uses Eqn. (4-5), two percent depth penetration and a friction coefficient of 0.4 to compare the rigid and flexible initial configuration cases. The flexible case is calculated using a disturbance in the first mode of the generalized velocity at the beginning of the simulation.

Several observations can be made, even though the changes are small. The magnitudes of the impact, before and after, and the change in velocity magnitudes are all largest for the rigid cases when compared to the corresponding flexible cases. Within the data of the flexible cases, the magnitudes decrease with increasing initial disturbance of the generalized velocity vector. The decrease in the magnitude of the change in velocities implies that the velocity discontinuity value is smaller for the flexible cases. Intuitively, the smaller the discontinuity, the smoother the transition from the unconstrained phase of motion to the constrained phase of motion is.

Table 5-3: Comparison of tip velocity magnitudes and coefficient of restitution when impact occurs.

Parameters		Beam Tip Velocity (m/s)			Coefficient Of Restitution
rpm	Other	Before Impact $V_{x(-0)}$	After Impact $V_{x(+0)}$	Change in Magnitude ΔV_x	$-\frac{V_{x(+0)}}{V_{x(-0)}}$
550	Rigid $\dot{\mathbf{q}}(1)=0$	1.0319	-0.5129	1.5448	0.4970
	Flexible $\dot{\mathbf{q}}(1)=0.1$	1.0171	-0.5074	1.5245	0.4989
	Flexible $\dot{\mathbf{q}}(1)=1$	0.9371	-0.4772	1.4143	0.5092
	Flexible $\dot{\mathbf{q}}(1)=2$	0.8912	-0.4596	1.3508	0.5157
3000	Rigid $\dot{\mathbf{q}}(1)=0$	5.6283	-1.8307	7.4590	0.3253
	Flexible $\dot{\mathbf{q}}(1)=0.1$	5.6006	-1.8239	7.4245	0.3257
	Flexible $\dot{\mathbf{q}}(1)=1$	5.3422	-1.7605	7.1027	0.3295
	Flexible $\dot{\mathbf{q}}(1)=2$	5.0358	-1.6842	6.7200	0.3344

The coefficient of restitution is slightly smaller for the rigid case than it is for the flexible case. As the initial excitation of the generalized velocity vector increases, so does the size of the coefficient of restitution. This coefficient ranges in value from 0 to 1, from a perfectly plastic collision to a perfectly elastic collision, respectively [4]. From the tabulated results, it appears that the impact that a flexible beam makes with a flat rigid surface is slightly more elastic than when compared to an impact involving a rigid beam. Another observation is that the rate of change in the size of the coefficient of restitution, as the initial excitation is increased, appears to be larger for the slower angular velocity.

Chapter Six: Discussions

The discussion of the results is presented in three sections for brevity. The first section includes a comparison of the graphs in chapter five to those in Hatman *et al.* [9]. This is followed by a comparison of different expressions for the generalized displacement vector of the predictor-corrector algorithm found in the literature. The third section briefly discusses other challenges encountered. These include imposing the second constraint (see Eqn. (4-111d)) and eliminating sharp spikes in the results.

6.1 Comparison to Hatman *et al.*'s results [9]

Further discussion about the obtained results is required because of the lack of consistency between these results and Hatman *et al.*'s results [9]. In their paper, most of the graphs displayed the expected trends discussed in the preceding chapter except for one graph. Their Figure 4, the equivalent to Figure 7 in this thesis, showed that the trend of the faster angular velocity yielding the higher normal force is reversed for the 2000 and 3000 rpm cases. This may indicate that reversals in the expected trend are possible and this may also attest to the complexity of the physics of the system. Unfortunately, there appears to be a mistake in this very figure. Part of the solution to the 550 rpm case is missing. According to the results found in Figure 1 of Hatman *et al.* [9], Figure 4 should have a noticeable normal force profile between the contact length –0.02 and –0.01. This begs the question, 'is the entire graph wrong or is there only that portion missing'? If only that portion is missing, then the trend reversal argument still holds.

Further, bounce back is observed in most of the graphs in Hatman *et al.* [9], especially in Figures 2-4, 7 and 8. These correspond to Figures 5-7, 10 and 11 in this thesis, respectively. All these graphs include the higher rates of rotation tested, and this is where bounce back is large enough to be observed. This may seem to be a trivial observation; however, since Hatman *et al.* [9] do not mention it, it raised a red flag when observing the numerical results being calculated. The fear was that the calculation of a positive lambda, that is supposed to occur when there is no contact but does not represent any real value, would offset the system's true values by calculating irrelevant generalized coordinates and velocities. Since bounce back is also visible in Hatman *et al.* [9], further investigation is not warranted at this point.

6.2 Discrepancies found in literature

This section includes one of the most important issues: there are discrepancies between different published articles on the numerical equation used to solve for the next generalized coordinate in the corrector phase of the constrained motion. When using a BDF of the form of Eqn. (4-114), the term 'x_{n-1}' is expected to appear when solving for 'x_n'. In Hatman *et al.* [9], that term is missing. Fung *et al.* [11], however, have included it while keeping the rest of the equation the same as Hatman *et al.*'s [9]. Andersen *et al.* [12] also include this term; however, the term that contains the extra variable, δ, added to create an index two system has a negative sign in front of it. This is different from both aforementioned papers. Fung *et al.* [11] and Andersen *et al.* [12] both cite Hatman *et al.* [9] but do not allude to the differences in this equation. Below, in Table 6-1, the three equations and Eqn. (4-114), are grouped together for clarity.

Table 6-1: Generalized coordinate vector equation in the corrector phase of different papers.

Reference	Generalized Coordinate Vector Equation
Hatman *et al.* [9]	$\mathbf{x}_n = h\mathbf{u}_n^p + \dfrac{C(\mathbf{x}_n^p,t_n)}{\mathbf{G}^T(\mathbf{x}_n^p,t_n)\bullet\mathbf{G}(\mathbf{x}_n^p,t_n)}\mathbf{G}(\mathbf{x}_n^p,t_n)$
Fung *et al.* [11]	$\mathbf{x}_n = \mathbf{x}_{n-1} + h\mathbf{u}_n^p + \dfrac{C(\mathbf{x}_n^p,t_n)}{\mathbf{G}^T(\mathbf{x}_n^p,t_n)\bullet\mathbf{G}(\mathbf{x}_n^p,t_n)}\mathbf{G}(\mathbf{x}_n^p,t_n)$
Andersen *et al.* [12]	$\mathbf{x}_n = \mathbf{x}_{n-1} + h\mathbf{u}_n^p - \dfrac{C(\mathbf{x}_n^p,t_n)}{\mathbf{G}^T(\mathbf{x}_n^p,t_n)\bullet\mathbf{G}(\mathbf{x}_n^p,t_n)}\mathbf{G}(\mathbf{x}_n^p,t_n)$
Chapter Four, Eqn. (4-114)	$\mathbf{x}_n = \mathbf{x}_{n-1} + h\mathbf{u}_n^p - \dfrac{C(a,t_n)}{\mathbf{G}^T(a,t_n)\bullet\mathbf{G}(a,t_n)}\mathbf{G}(a,t_n)$, where $a = \mathbf{x}_{n-1} + h\mathbf{u}_n^p$

Based on Gear *et al.* [1], which was cited in Hatman *et al.* [9] specifically for its predictor-corrector procedure, there is an equation for $\partial C/\partial v_n$, where 'v_n' equals δ multiplied with the stepsize. This equation is used to solve for the generalized coordinate vector corrector equation. Hatman *et al.* [9], Fung *et al.* [11], and Andersen *et al.* [12] mention the use of a Newton-Raphson (NR) iteration, but do not explicitly show it. Reviewing Gear's equation for

$\partial C/\partial v_n$ [1], it appears that the NR iteration is used to solve for the extra variable, δ, using the constraint equation. Note that the problem presented here has a constraint dependent on both time and space, unlike [1], whose constraint is only dependent on space. The equations presented here reflect this difference.

Using the BDF outlined in the previous chapter, the following equation is obtained:

$$\mathbf{x}_n = \mathbf{x}_{n-1} + h\mathbf{u}_n^p + v_n\mathbf{G} \tag{6-1}$$

where

$$v_n = h\delta \tag{6-2}$$

Now, using the NR equation,

$$v_n = v_{n-1} - \frac{f(v_{n-1},t)}{f'(v_{n-1},t)} \tag{6-3}$$

where

$$f(v_{n-1},t) = C(\mathbf{x}_{n-1} + h\mathbf{u}_n^p + v_{n-1}\mathbf{G},t) \tag{6-4}$$

and letting a = $\mathbf{x}_{n-1} + h\mathbf{u}_n^p + v_{n-1}\mathbf{G}$, the derivative of Eqn. (6-4) with respect to v_n is found:

$$\frac{\partial C}{\partial v_n} = \frac{\partial C(a)}{\partial v_n}\frac{\partial a}{\partial v_n} = \mathbf{G}(a)\frac{\partial a}{\partial v_n} \tag{6-5}$$

$$\frac{\partial a}{\partial v_n} = \mathbf{G}^T(a) + \frac{\partial \mathbf{G}(a)}{\partial v_n}v_n = \mathbf{G}^T(a) + \frac{\partial \mathbf{G}(a)}{\partial a}\frac{\partial a}{\partial v_n}v_n \tag{6-6}$$

$$\left[\mathbf{I} - \frac{\partial \mathbf{G}(a)}{\partial a}v_n\right]\frac{\partial a}{\partial v_n} = \mathbf{G}^T(a) \tag{6-7}$$

$$\frac{\partial a}{\partial v_n} = \left[\mathbf{I} - \frac{\partial \mathbf{G}(a)}{\partial a} v_n \right]^{-1} \mathbf{G}^T(a) \tag{6-8}$$

Substituting Eqn. (6-8) into Eqn. (6-5), the result is:

$$\frac{\partial C}{\partial v_n} = \mathbf{G}(a) \left[\mathbf{I} - \frac{\partial \mathbf{G}(a)}{\partial a} v_n \right]^{-1} \mathbf{G}^T(a) \tag{6-9}$$

Letting $v_{n-1} = 0$, for the first (and only iteration) required, and substituting Eqn. (6-9) into Eqn. (6-3), the final result is:

$$v_n = -\frac{C(x_{n-1} + h\mathbf{u}_n^P)}{\mathbf{G}^T(x_{n-1} + h\mathbf{u}_n^P) \bullet \mathbf{G}(x_{n-1} + h\mathbf{u}_n^P)} \tag{6-10}$$

This leads to an equation with the same structure as that in Andersen *et al.* [12]. Using this logic, however, leads to one difference between all three papers and this thesis. The last term of Eqn. (4-114), is calculated using Eqn. (6-10), not $f(\mathbf{x}_n^P, t)$. This change significantly aided in the convergence of the corrector phase in the coded algorithm.

6.3 Other Issues

Another challenge was to impose the second constraint added when the system was turned into a two-index problem from a three-index problem. Making the total time derivative of the constraint equal to zero by calculating the partial time derivative of the constraint, using the value of $\mathbf{x}_n$, created normal force values on the order of 1E4. This happened because the difference between the actual calculation and the values of zero was too large when used in the calculations for lambda. The other terms in the Lagrange multiplier equation, Eqn. (4-116), are multiplied by the very small step size, forcing this constraint to compensate numerically. One way to approach this problem was to force the partial derivative of the constraint with respect to time to be equal to the gradient of the constraint manifold multiplied by the predicted value of $\mathbf{u}_n$.

While forcing a constraint is counter-productive to finding the corrected values, it is important to note that this second constraint is not fully forced to the zero value. The values of $\mathbf{u}_{n-1}$ and $\mathbf{u}_n^p$ are not necessarily equal; therefore, they still play a role in Eqn. (4-116).

On a quick note, sudden spikes may be found in the simulated data, and this indicates that the step size being used is not sufficiently small. A sudden spike is characterized by a very sharp change in the magnitude of the normal force. In the figures presented in Chapter Five, there appears to be sudden spikes, but this is due to the relative contact length with other portions of the results. All the apparent spikes have been checked to ensure that they are exactly obtained through gradual numerical iterations.

Chapter Seven: Conclusion

The preceding chapters presented the description, solution and discussion of a problem that is composed of a flexible Euler-Bernoulli beam attached to a rotating hub and whose tip experiences intermittent contact. Two assumptions of the beam state before impact were investigated. The first is the rigid beam assumption [8] and [9], and the second is its relaxation to allow flexibility. While some of the results were counter-intuitive, important conclusions can be inferred. These include the evidence of bounce back, and the importance of a smoother transition between types (or stages) of motion. The expected trends of increasing normal forces with increasing angular velocity, with increasing depth of penetration and with decreasing friction coefficient were found to some extent in all the graphed simulations.

The numerical algorithm, specifically, the predictor-corrector process, was most difficult to implement and is suspected of causing the differences encountered between the expected and obtained results. The discrepancies found in the literature made finding the appropriate numerical algorithm for constrained motion difficult; however, adjustments were made to produce a converging algorithm.

Future work is strongly encouraged for this type of problem because of its far-reaching applicability. The solution to this problem is a great foundation because it creates opportunities for different branches of research within these applications. For further extension or expansion of this work, it is suggested that the numerical process first be reviewed. All the anticipated trends were not observed consistently. Hence, a first step would be to determine whether the observations are numerical artifacts or the result of some physical phenomena not captured in the model.

It may also help to use Runge-Kutta methods that take into consideration the variable coefficients in the equations of motion. General explicit Runge-Kutta methods of second order with varying step-size and fifth order, in the literature [9] and in this thesis, respectively, are used to solve this particular problem. All the coefficient matrices, however, change with time. The ordinary differential equation has now become much more complex due to higher-order terms of time.

Once the simulations run with evidence of a sound procedure, the results obtained can contribute to a greater understanding of rotating beams and the contact process. It can be applied to slower moving purposes, such as the control of robotic arms, by removing the axial stiffening effects if it makes the solution easier to calculate. It can also be applied to fast rotating engine parts [16], but without having to provide a loading profile. Also, adding additional wire filaments is another option to investigate. Using other beam theories or changing the way the axial stiffening is calculated can also advance the procedure to be used for more complex structures. Finding exact eigenfunctions for this problem is another interesting research option [2].

Another option is to conduct experimental investigations. This would be extremely beneficial because comparing the theoretical to the experimental results would give an indication of whether the model being used is sufficient to capture the system behaviour. A better understanding of the actual physics involved could be realized and this could possibly confirm or challenge the expected trends listed above.

References

1. C.W. Gear, B. Leimkuhler and G. K. Gupta. *Automatic integration of Euler-Lagrange equations with constraints.* Journal of Computational and Applied Mathematics (1985) 12/13, 77-90.
2. F. Bellezza, L. Lanari, G. Ulivi. Exact modeling of the flexible slewing link. IEEE International conference on Robotics and Automation, 1990 Proceedings (1990) 1, 734-739.
3. A.P. Christoforou and A.S. Yigit. *Effect of flexibility on low velocity impact response.* Journal of Sound and Vibration (1998) 217(3), 563-578.
4. G. Gilardi and I. Sharf. *Literature survey of contact dynamics modelling.* Mechanism and Machine Theory (2002) 37, 1213-1239.
5. A.S. Yigit, A.G. Ulsoy and R.A. Scott. *Dynamics of a radially rotating beam with impact, Part 1: Theoretical and Computational Model.* Journal of Vibration and Acoustics (1990) 112, 65-70.
6. A.S. Yigit, A.G. Ulsoy and R.A. Scott. *Dynamics of a radially rotating beam with impact, Part 2: Experimental and simulation results.* Journal of Vibration and Acoustics (1990) 112, 71-77.
7. A.S. Yigit. *The effect of flexibility on the impact response of a two-link rigid-flexible manipulator.* Journal of Sound and Vibration (1994) 177(3), 349-361.
8. V.G. Hatman, I. Haque and A. Bagchi. *Dynamics of a flexible rotating beam interacting with a flat rigid surface, Part I: Model development.* Journal of Sound and Vibration (1996) 194(5), 653-669.
9. V.G. Hatman, I. Haque and A. Bagchi. *Dynamics of a flexible rotating beam interacting with a flat rigid surface, Part II: Numerical Solution.* Journal of Sound and Vibration (1996) 194(5), 671-683.
10. H. Palas, W.C. Hsu, and A.A. Shabana. *On the use of momentum balance and the assumed modes method in transverse impact problems.* Transactions of the ASME Journal of Vibration and Acoustics. (1992) 114, 364-373.

11. R.F. Fung, C.M. Yao and D.G. Chang. *Dynamic and contact analysis of a bimodal ultrasonic motor.* IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control (1999) 46(1), 47-60.

12. B. Andersen, M. Blanke and J. Helbo. *Two-mode resonator and contact model for standing wave piezomotor.* Proceedings of the 2001 ASME Design Engineering Technical Conferences & Computers and Information in Engineering Conference, Pittsburgh, Pennsylvania, USA, September 9-12, 2001.

13. R.F. Fung and H.C. Chang. *Dynamic modelling of a non-linearly constrained flexible manipulator with a tip mass by Hamilton's principle.* Journal of Sound and Vibration (1998) 216(5), 751-769.

14. F. Matsuno and S. Kasai. *Modeling and robust force control of constrained one-link flexible arms.* Journal of Robotic Systems (1998) 15(8), 447-464.

15. Y. Morita, Y. Kobayashi, H. Kando, F. Matsuno, T. Kanzawa and H.Ukai. *Robust force control of a flexible arm with a nonsymmetric rigid tip body.* Journal of Robotic Systems (2001) 18(5), 221-235.

16. S.K. Sinha. *Non-linear dynamic response of a rotating radial Timoshenko beam with periodic pulse loading at the free-end.* International Journal of Non-Linear Mechanics (2005) 40, 113-149.

17. J.B. Yang, L.J. Jiang, and D.CH. Chen. *Dynamic modelling and control of a rotating Euler-Bernoulli beam.* Journal of Sound and Vibration (2004) 274, 863-875.

18. W.C. Hsu and A.A. Shabana. *Finite element analysis of impact-induced transverse waves in rotating beams.* Journal of Sound and Vibration (1993) 168(2), 355-369.

19. A.A. Al-Qaisia and B.O. Al-Bedoor. *Evaluation of different methods for the consideration of the effect of rotation on the stiffening of rotating beams.* Journal of Sound and Vibration (2005), 531-553.

20. B.H. Edwards, R.P. Hostetler, and R.E. Larson. *Calculus,* 6th Edition. Houghton Mifflin Company. New York, 1998.

21. L. Meirovitch. *Fundamentals of Vibrations.* McGraw Hill. Toronto, 2001.

22. D. Inman. Engineering Vibration. Prentice-Hall, Inc., Toronto, 1996.

23. L. Rade and B. Westergren. *Mathematics Handbook for Science and Engineering,* 4th Edition. Springer-Verlag Berlin Heidelberg. New York, 1999.

24. W. Blajer. *Index of differential-algebraic equations governing the dynamics of constrained mechanical systems.* Applied Mathematical Modelling (1992) 16, 70-77.

25. J. Baumgarte. *Stabilization of constraints and integrals of motion in dynamical systems.* Computational Mathematics and Applied Mechanical Engineering (1976) 1, 1-16.

26. U.M. Ascher and L.R. Petzold. *Stability of computational methods for constrained dynamics systems.* SIAM Journal of Scientific Computing (1993) 14, 95-120.

27. Student Companion Site, Epperson: An Introduction to Numerical Methods and Analysis. *Gaussian Quadrature Weights and Abscissas.* John Wiley & Sons, Inc. (2000-2006), http://bcs.wiley.com/he-bcs/Books?action=weblinks&bcsId=1264&itemId=0471316474 &webLinkURL=http://www3.interscience.wiley.com:8100/legacy/college/epperson/ 0471316474/gauss/gauss.html

28. R.P. Canale and S.C. Chapra. *Numerical Methods for Engineers: With Software and Programming Applications.* (4th ed.) New York: McGraw-Hill, 2002.

Appendix A – Derivation of Eqn. (4-105)

The coefficient of restitution is defined as the negative ratio of the post-impact velocity to the pre-impact velocity. This velocity is the relative normal velocity between the two bodies coming into contact. The derivative of the constraint function with respect to time represents the relative normal velocity:

$$e = -\frac{\dot{C}(\mathbf{q}_{(+0)}, \dot{\mathbf{q}}_{(+0)}, t)}{\dot{C}(\mathbf{q}_{(-0)}, \dot{\mathbf{q}}_{(-0)}, t)} \tag{A-1}$$

Therefore:

$$\dot{C}(\mathbf{q}_{(+0)}, \dot{\mathbf{q}}_{(+0)}, t) = -e\,\dot{C}(\mathbf{q}_{(-0)}, \dot{\mathbf{q}}_{(-0)}, t) \tag{A-2}$$

where,

$$\dot{C}(\mathbf{q}, \dot{\mathbf{q}}, t) = \frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}} + \frac{\partial C}{\partial t} \tag{A-3}$$

It follows:

$$\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(+0)} + \frac{\partial C}{\partial t} = -e\left(\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(-0)} + \frac{\partial C}{\partial t}\right) \tag{A-4}$$

The goal is to find the jump in generalized velocity, which is defined as:

$$\Delta\dot{\mathbf{q}} = \left(\dot{\mathbf{q}}_{(+0)} - \dot{\mathbf{q}}_{(-0)}\right) \tag{A-5}$$

To create this term in Eqn. (A-4), $-\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(-0)}$ will be added to each side:

$$\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(+0)}+\frac{\partial C}{\partial t}-\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(-0)}=-e\left(\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(-0)}+\frac{\partial C}{\partial t}\right)-\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(-0)} \tag{A-6}$$

$$\frac{\partial C}{\partial \mathbf{q}}\Delta\dot{\mathbf{q}}=-e\left(\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(-0)}+\frac{\partial C}{\partial t}\right)-\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(-0)}-\frac{\partial C}{\partial t} \tag{A-7}$$

$$\frac{\partial C}{\partial \mathbf{q}}\Delta\dot{\mathbf{q}}=-(1+e)\frac{\partial C}{\partial \mathbf{q}}\dot{\mathbf{q}}_{(-0)}-(1+e)\frac{\partial C}{\partial t} \tag{A-8}$$

Noting that $\frac{\partial C}{\partial \mathbf{q}}$ is equivalent to *grad*C and using the subscript 'o' to denote values representing initial impact configuration, equation (4-105) is finally obtained:

$$grad\mathbf{C}_0^T\Delta\dot{\mathbf{q}}=-(1+e)grad\mathbf{C}_0^T\left(\dot{\mathbf{q}}_{(-0)}\right)-(1+e)\frac{\partial \mathbf{C}_0}{\partial t} \tag{A-9}$$

Appendix B – Matlab Program Codes

There are fourteen programs that are used to complete the simulations in Chapter five. Here, they are divided into seven sections: the main parent program, the general parameter definition programs, which hold and/or produce variable values for use in other programs, the unconstrained motion programs, the building system matrices program, the jump velocity calculation program, the constrained motion programs and finally, the program that calculates the velocity terms.

B1. Main Parent Program

```
% Filename: rotate.m
%
% Parent program:  (none)
% Child programs:  getvalues.m, XIequalONE.m, Unconstrained.m, jump.m,
%                  Constrained.m, LagMultip.m, TipDistance.m
%
% Purpose: This program simulates the rotation of a circular cross-sectioned, cantilever beam attached to a
%          rotating hub. The beam begins in the horizontal position when it is let go and it eventually
%          comes into contact with a flat rigid surface. It experiences unconstrained motion, impact,
%          constrained motion with interspersed unconstrained motion and finally, it is released into
%          unconstrained motion once again. The program ends when the angle of rotation of the clamped
%          end of the beam reaches a certain value that the tip of the beam has definitely lost contact with
%          the rigid surface. This program builds arrays of the global positioning coordinates, the normal force
%          encountered by the beam, the generalized coordinates and velocities, the value of the constraint
%          equation, and the time at which all this occurs.
%
% Variables: L - length of beam, rh - radius of hub, h - time step
%       AngVel - system angular velocity
%       ISP_one - ISP matrix computed for the tip of the beam
%       S_one - S (mode shape) vector computed for the tip of the beam
%       d - normal distance between the centre of the hub and the flat rigid surface
%       cnt - counter used to build arrays
%       flag - changes value when the initial impact of the rigid case is over to control how time is advanced in the
%           unconstrained phase
%       flag1 - changes value to stop program when the angle of the clamped end exceeds the maximum value
%       flag2 - changes value to supress Lagrange multiplier prediction when not needed in Constrained.m
%       flag3 - changes value if the program should run TipDistance.m
%       q - generalized coordinates vector
%       q_dot - generalized velocity vector
%       t - time, nlambbda - Lagrange multiplier
%       angle - angle of the clamped end of beam to the positive X-axis
%       X - global X-axis coordinate, Y - global Y-axis coordinate
%       C - constraint function
%       prevLAM - keeps the previous Lagrange multiplier value
%       N - the magnitude of the partial derivative of the constraint equation with respect to the generalized
%          coordinates
%       a, b, f - variables that hold the latest generalized coordinates,generalized velocities and time to be
%           used in TipDistance.m.
%          TipDist - the distance between the tip of the beam and the flat surface (equivalent to C)

clear;
clc;

global L AngVel rh h ISP_one S_one d cnt flag flag1 flag2 flag3

% Calling functions getvalues.m, XIequalONE.m to set variables needed for the rest of the program.
[q, q_dot, t, nlambda] = getvalues;
[ISP_one, S_one] = XIequalONE;

% Beginning arrays at cnt = 1, angle = -90 degrees;
angle = -(pi/2) + AngVel*t(cnt);
X(cnt) = cos(angle)*(rh + L - (0.5/L)*q(:,cnt)'*ISP_one*q(:,cnt)) - sin(angle)*q(:,cnt)'*S_one;
Y(cnt) = sin(angle)*(rh + L - (0.5/L)*q(:,cnt)'*ISP_one*q(:,cnt)) + cos(angle)*q(:,cnt)'*S_one;
C(cnt) = X(cnt) - d;
pam(cnt) = 0;

% This while loop keeps the program running for any value of C and can only be stopped when the variable angle is
% greater than or equal to +90 degrees.

while C(cnt) >= 0 || C(cnt) < 0

  % This if statement runs when the value of C(cnt) is less than zero.

  if C(cnt) < 0

    % This while loop runs as long as C(cnt) remains less than zero and calls Unconstrained.m to run one Runge-
    % Kutta iteration for the unconstrained phase of motion for each loop.

    while C(cnt) < 0

      [q(:,cnt), q_dot(:,cnt), t(cnt), X(cnt), Y(cnt), C(cnt), pam(cnt)] = Unconstrained(q(:,cnt), q_dot(:,cnt), t(cnt));
      fprintf('angle = %g\t\tC = %g\t\tX = %g\t\tY = %g\n', -(pi/2) + AngVel*t(cnt), C(cnt), X(cnt), Y(cnt));

      % When 'angle' is greater than or equal to +90 degrees, flag1 is made equal to zero and this breaks the
      % program out of the 'while C(cnt) < 0' loop.

      if flag1 == 0;
        break
      end

    end    % End of 'while C(cnt) < 0' loop.

  % The last value of C of the 'while C(cnt) < 0' loop is positive so the following 'if' statement does linear
  % interpolation to find when contact first occurs (when C(cnt) = 0).

    if C(cnt) > 0

      t(cnt) = t(cnt) - C(cnt)*(t(cnt-1) - t(cnt))/(C(cnt-1) - C(cnt));
      q(:,cnt) = q(:,cnt) - C(cnt)*(q(:,cnt-1) - q(:,cnt))/(C(cnt-1) - C(cnt));
      q_dot(:,cnt) = q_dot(:,cnt) - C(cnt)*(q_dot(:,cnt-1) - q_dot(:,cnt))/(C(cnt-1) - C(cnt));

      % Solution is updated without advancing the counter because these values are needed to overwrite the last
      % values.
```

```
            angle = -(pi/2) + AngVel*t(cnt);
            X(cnt) = cos(angle)*(rh + L - (0.5/L)*q(:,cnt)'*ISP_one*q(:,cnt)) - sin(angle)*q(:,cnt)'*S_one;
            Y(cnt) = sin(angle)*(rh + L - (0.5/L)*q(:,cnt)'*ISP_one*q(:,cnt)) + cos(angle)*q(:,cnt)'*S_one;
            C(cnt) = X(cnt) - d;
            pam(cnt) = 0;

        end    % End of 'if C(cnt) > 0' statement.

        nlambda = 0;    % Lagrange multiplier is reset to zero.

        % When 'angle' is greater than or equal to +90 degrees, flag1 is made equal to zero and this breaks the program
        % out of the 'if C(cnt) < 0' loop.
        if angle >= pi/2
            flag1 = 0;
            break
        end

    elseif C(cnt) >= 0      % Corresponds to the 'if C(cnt) < 0' statement.

        % When this loop is first entered, the beam is coming into contact with the flat surface. First, impact occurs, so
        % the jump.m function is called to calculate the jump velocity vector. Since time and position do not change,
        % time, the generalized coordinates, the global coordinates, the constraint function and the normal force all
        % remain the same.

            cnt = cnt + 1; t(cnt) = t(cnt-1); q(:,cnt) = q(:,cnt-1);
            X(cnt) = X(cnt-1); Y(cnt) = Y(cnt-1);
            C(cnt) = C(cnt-1); pam(cnt) = 0;

            [q_dot(:,cnt)] = jump(q(:,cnt), q_dot(:,cnt-1), t(cnt));

        % This while loop runs as long as C(cnt) remains greater than or equal to zero. This represents the constrained
        % phase of motion and calls Constrained.m to predict and correct the generalized vectors and the Lagrange
        % multiplier values. LagMultip.m and TipDistance.m are called to direct the program into the next appropriate
        % phase of motion.

        while C(cnt) >= 0

            h = 1e-7;
            fprintf('\n\n');
            prevLAM = nlambda;

            % This 'if' statement changes the stepsize if the value of the Lagrange multiplier becomes very small.
            % Changing the size allows the program to avoid spikes that occur when the results are not accurate enough
            % (the Lagrange multiplier oscillates for long periods of time), and allows it to run faster when this extra
            % accuracy is not needed.

            if abs(nlambda) > 0 && abs(nlambda) < 1e-4
                h = 1e-8;
            end

            % Constrained.m predicts and corrects the generalized vectors and the Lagrange multiplier values for the
            % constrained motion.

            [nx, nu, ti, nlambda, N] = Constrained(q, q_dot, t(cnt), nlambda);
            fprintf('\n\n');
        % LagMultip.m updates the solution and uses the sign of the corrected nlambda to direct the program into
        % the correct phase of motion. If the sign is negative, constrained motion continues. If the sign is positive,
        % the motion becomes unconstrained and the program continues to TipDistance.m.

        [q(:,cnt), q_dot(:,cnt), t(cnt), X(cnt), Y(cnt), C(cnt), pam(cnt), TipDist, a, b, f] = LagMultip(q, q_dot, t, ti,
        nlambda, nx, nu, N, prevLAM);

        % If the program is directed to TipDistance.m, the sign of the variable TipDist directs the program to the
        % unconstrained phase of motion with the correct values for the latest solution.  If the sign is negative, the
        % solution is updated right away and the program goes into unconstrained motion. If the sign is positive,
        % Runge-Kutta iterations of the unconstrained system are run until the TipDist sign becomes negative, and
        % the latest values are used to update the solution.

        if flag3 == 1
          [q(:,cnt), q_dot(:,cnt), t(cnt), X(cnt), Y(cnt), C(cnt), pam(cnt)] = TipDistance(a, b, f, TipDist);
        end

        if flag1 == 0;
          break
        end

      end       % End of 'while C(cnt) >= 0' loop.

      % The step-size is made smaller so the unconstrained motion runs faster.
      h = 1e-6;

      % flag2 is reassigned so it can be used the next time it is needed in Constrained.m.
      flag2 = 1;

        if angle >= pi/2
          flag1 = 0;
          break
        end

    end    % End of 'if C(cnt) < 0' statement.

    % This 'if' statement allows the program to stop completely.

    if flag1 == 0
      break
    end

  end    % End of 'while C(cnt) >= 0 || C(cnt) < 0' loop.
```

B2. General Parameter Definition Programs

```
% Filename: getvalues.m
%
% Parent program:  rotate.m
% Child programs:  (none)
%
% Purpose: This program contains all initial values needed. Most values are made global, however, the initial
%             generalized coordinates, the initial generalized velocities, initial time and initial nlambda are returned.
```

```
function [q, q_dot, t, nlambda] = getvalues

global Nmds L rh d A rho I AngVel AngAcc E mu zone h ang
global plambda flag flag1 flag2 flag3 cnt Acoeff ri

%%%%CHANGEABLE VARIABLES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rpm = 550;    % revolutions per minute.
pp = 0.02;    % percent penetration: pp*100
mu = 0.4;     % Coefficient of friction.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Nmds = 5;                   % number of modes (5 maximum)
L = 0.0635;                 % length of beam (m)
rh = 0.0635;                % radius of hub (m)
dia = 0.000508;             % cross-section diameter (m)
d = (L+rh)-(pp*L);          % perpendicular distance from centre of hub to wokpiece surface (m)
ang = -acos(d/(L+rh));      % angle at which impact first occurs for rigid body configuration
fprintf('d = %g\nang = %g\n', d, ang);
A = pi*(dia/2)^2;           % cross-sectional area of beam (m^2)
rho = 7800;                 % density of steel (kg/m^3)
I = pi*(dia^4)/64;          % moment of inertia Iz (m^4)
AngVel = rpm*(pi/30);       % angular velocity (rad/s)
E = 2.2e11;                 % elastic modulus of steel (N/m^2)
zone = 0.005;               % width of clearance zone (m)
h = 0.001;                  % time step (s)
plambda = 10000; nlambda = 0;   % initial predicted lambda and lambda

flag = 1;  % flag = 0 when the initial impact is over to control how time is advanced in the unconstrained phase.
           % Make sure to make equal to 0 when studying flexible case!
flag1 = 1; % flag1 = 0 when the maximum angle of rotation is reached.
flag2 = 1; % flag2 = 1 when the Lagrange multiplier needs to be predicted in Constrained.m.
flag3 = 0; % flag3 = 1 when the program must enter the TipDistance.m in the constrained phase of motion.
cnt = 1;   % Counter used to build arrays.

% The ri vector holds the first five solutions to the characteristic equation of a non-rotating cantilever beam
% multiplied by the length of the beam. The Acoeff vector calculates the coefficients for the mode shape
% eigenfunctions.

ri = [1.87510407; 4.69409113; 7.85475744; 10.99554073; 14.13716839];
Acoeff = (sinh(ri) - sin(ri))./(cosh(ri) + cos(ri));

% Initial Conditions
q = zeros(Nmds,cnt);        % generalized coordinates vector array
q_dot = zeros(Nmds,cnt);    % generalized velocity vector array
t = zeros(cnt);             % time array

% q_dot = [0.1;0;0;0;0];    % Used for initial flexible beam configuration

return


% Filename: XIequalONE.m
%
% Parent program:  rotate.m
% Child programs:  (none)
%
```

```
%
% Purpose:  This program returns the matrix ISP and the vector S evaluated at the free end of the beam.  ISP and S
%           have to do with the mode shapes of the beam.
%
% Variables: xi - represents the non-dimensional location along the beam at which ISP and S is calculated for. xi = 1
%                 represents the free end of the beam.
%          i - represents the row of the matrix
%          j - represents the column of the matrix
%          aa - integration of sin(ri(i)*xi)*sin(ri(j)*xi) from 0 to 1
%          bb - integration of sin(ri(i)*xi)*cos(ri(j)*xi) from 0 to 1
%          cc - integration of sin(ri(i)*xi)*sinh(ri(j)*xi) from 0 to 1
%          dd - integration of sin(ri(i)*xi)*cosh(ri(j)*xi) from 0 to 1
%          ee - integration of cos(ri(i)*xi)*sin(ri(j)*xi) from 0 to 1
%          ff - integration of cos(ri(i)*xi)*cos(ri(j)*xi) from 0 to 1
%          gg - integration of cos(ri(i)*xi)*sinh(ri(j)*xi) from 0 to 1
%          hh - integration of cos(ri(i)*xi)*cosh(ri(j)*xi) from 0 to 1
%          ii - integration of sinh(ri(i)*xi)*sin(ri(j)*xi) from 0 to 1
%          jj - integration of sinh(ri(i)*xi)*cos(ri(j)*xi) from 0 to 1
%          kk - integration of sinh(ri(i)*xi)*sinh(ri(j)*xi) from 0 to 1
%          ll - integration of sinh(ri(i)*xi)*cosh(ri(j)*xi) from 0 to 1
%          mm - integration of cosh(ri(i)*xi)*sin(ri(j)*xi) from 0 to 1
%          nn - integration of cosh(ri(i)*xi)*cos(ri(j)*xi) from 0 to 1
%          oo - integration of cosh(ri(i)*xi)*sinh(ri(j)*xi) from 0 to 1
%          pp - integration of cosh(ri(i)*xi)*cosh(ri(j)*xi) from 0 to 1

function [ISP_one, S_one] = XIequalONE

global Nmds Acoeff ri

xi = 1;

for i = 1:Nmds

  for j = 1:Nmds

    if i == j

      aa = (-cos(ri(i)*xi)*sin(ri(i)*xi) + ri(i)*xi)/(2*ri(i));

      bb = (sin(ri(i)*xi))^2/(2*ri(i));

      cc = (-cos(ri(i)*xi)*sinh(ri(i)*xi) + sin(ri(i)*xi)*cosh(ri(i)*xi))/(2*ri(i));

      dd = -(-1 + cos(ri(i)*xi)*cosh(ri(i)*xi) - sin(ri(i)*xi)*sinh(ri(i)*xi))/(2*ri(i));

      ee = bb;

      ff = (cos(ri(i)*xi)*sin(ri(i)*xi) + ri(i)*xi)/(2*ri(i));

      gg = (-1 + cos(ri(i)*xi)*cosh(ri(i)*xi) + sin(ri(i)*xi)*sinh(ri(i)*xi))/(2*ri(i));

      hh = (cos(ri(i)*xi)*sinh(ri(i)*xi) + sin(ri(i)*xi)*cosh(ri(i)*xi))/(2*ri(i));

      ii = cc;

      jj = gg;
    kk = -(-cosh(ri(i)*xi)*sinh(ri(i)*xi) + ri(i)*xi)/(2*ri(i));

    ll = (-1 + (cosh(ri(i)*xi))^2)/(2*ri(i));

    mm = dd;

    nn = hh;

    oo = ll;

    pp = (cosh(ri(i)*xi)*sinh(ri(i)*xi) + ri(i)*xi)/(2*ri(i));

else

    aa = (sin((ri(i)-ri(j))*xi)*ri(i) + sin((ri(i)-ri(j))*xi)*ri(j) - ...
      sin((ri(i)+ri(j))*xi)*ri(i) + sin((ri(i)+ri(j))*xi)*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

    bb = -(-2*ri(i) + cos((ri(i)+ri(j))*xi)*ri(i) - cos((ri(i)+ri(j))*xi)*ri(j) + ...
      cos((ri(i)-ri(j))*xi)*ri(i) + cos((ri(i)-ri(j))*xi)*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

    cc = -(ri(i)*sinh(ri(j)*xi)*cos(ri(i)*xi) - ri(j)*cosh(ri(j)*xi)*sin(ri(i)*xi))/((ri(j)^2) + (ri(i)^2));

    dd = -(-ri(i) + ri(i)*cosh(ri(j)*xi)*cos(ri(i)*xi) - ri(j)*sinh(ri(j)*xi)*sin(ri(i)*xi))/((ri(j)^2) + (ri(i)^2));

    ee = -(-2*ri(j) + cos((ri(j)+ri(i))*xi)*ri(j) - cos((ri(j)+ri(i))*xi)*ri(i) + ...
      cos((ri(j)-ri(i))*xi)*ri(j) + cos((ri(j)-ri(i))*xi)*ri(i))/(2*(ri(j)-ri(i))*(ri(j)+ri(i)));

    ff = (sin((ri(i)-ri(j))*xi)*ri(i) + sin((ri(i)-ri(j))*xi)*ri(j) + ...
      sin((ri(i)+ri(j))*xi)*ri(i) - sin((ri(i)+ri(j))*xi)*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

    gg = (-ri(j) + ri(j)*cosh(ri(j)*xi)*cos(ri(i)*xi) + ri(i)*sinh(ri(j)*xi)*sin(ri(i)*xi))/((ri(j)^2) + (ri(i)^2));

    hh = (ri(i)*cosh(ri(j)*xi)*sin(ri(i)*xi) + ri(j)*sinh(ri(j)*xi)*cos(ri(i)*xi))/((ri(j)^2) + (ri(i)^2));

    ii = -(ri(j)*sinh(ri(i)*xi)*cos(ri(j)*xi) - ri(i)*cosh(ri(i)*xi)*sin(ri(j)*xi))/((ri(i)^2) + (ri(j)^2));

    jj = (-ri(i) + ri(i)*cosh(ri(i)*xi)*cos(ri(j)*xi) + ri(j)*sinh(ri(i)*xi)*sin(ri(j)*xi))/((ri(i)^2) + (ri(j)^2));

    kk = (sinh((ri(i)+ri(j))*xi)*ri(i) - sinh((ri(i)+ri(j))*xi)*ri(j) - ...
      sinh((ri(i)-ri(j))*xi)*ri(i) - sinh((ri(i)-ri(j))*xi)*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

    ll = (-2*ri(i) + cosh((ri(i)+ri(j))*xi)*ri(i) - cosh((ri(i)+ri(j))*xi)*ri(j) + ...
      cosh((ri(i)-ri(j))*xi)*ri(i) + cosh((ri(i)-ri(j))*xi)*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

    mm = -(-ri(j) + ri(j)*cosh(ri(i)*xi)*cos(ri(j)*xi) - ri(i)*sinh(ri(i)*xi)*sin(ri(j)*xi))/((ri(i)^2) + (ri(j)^2));

    nn = (ri(j)*cosh(ri(i)*xi)*sin(ri(j)*xi) + ri(i)*sinh(ri(i)*xi)*cos(ri(j)*xi))/((ri(i)^2) + (ri(j)^2));

    oo = (-2*ri(j) + cosh((ri(j)+ri(i))*xi)*ri(j) - cosh((ri(j)+ri(i))*xi)*ri(i) + ...
      cosh((ri(j)-ri(i))*xi)*ri(j) + cosh((ri(j)-ri(i))*xi)*ri(i))/(2*(ri(j)-ri(i))*(ri(j)+ri(i)));

    pp = (sinh((ri(i)+ri(j))*xi)*ri(i) - sinh((ri(i)+ri(j))*xi)*ri(j) + ...
      sinh((ri(i)-ri(j))*xi)*ri(i) + sinh((ri(i)-ri(j))*xi)*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

end
```

```
      ISP_one(i,j) = ri(i)*ri(j)*(aa + cc + ii + kk - Acoeff(j)*(dd - bb + ll - jj) - Acoeff(i)*(mm + oo - ee - gg) + ...
         Acoeff(i)*Acoeff(j)*(ff - hh - nn + pp));

   end

   S_one(i,1) = cosh(ri(i)*xi) - cos(ri(i)*xi) - Acoeff(i)*(sinh(ri(i)*xi) - sin(ri(i)*xi));

end

return
```

B3. Unconstrained Motion Programs

```
% Filename: Unconstrained.m
%
% Parent program:  rotate.m
% Child programs:  uncODE.m
%
% Purpose:  This program advances the program through the unconstrained phase of motion once every time it is
%            called.  The inputs are the latest values for the generalized coordinate and velocity vectors, and time.
%            The outputs are the updated generalized coordinates and velocities, time, global coordinate
%             positions, the value of the constraint function, and the normal force.
%
% Variables: h - step size
%         ang - angle at which impact first occurs for rigid body configuration
%         a, b, f - variables that hold the latest generalized coordinates,generalized velocities and time

function [q, q_dot, t, X, Y, C, pam] = Unconstrained(q, q_dot, t)

global L rh AngVel ISP_one S_one d cnt h flag flag1 Nmds ang

h = 1e-6;

% If flag = 1, then the rigid case is being tested and the initial impact has not yet occurred.  When flag = 1, the time
% at which impact occurs is found.  When flag = 0, a Runge-Kutta iteration for the unconstrained system is
% completed via uncODE.m.

if flag == 1

   t = (ang + (pi/2))/AngVel;
   flag = 0;

else

   [a, b, f] = uncODE(q, q_dot, t);
   t = f; q = a; q_dot = b;

end

% The solution is updated and returned.

cnt = cnt + 1;
angle = -(pi/2) + AngVel*t;
X = cos(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) - sin(angle)*q'*S_one;
```

```
Y = sin(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) + cos(angle)*q'*S_one;
C = X - d;
pam = 0;

if angle >= pi/2
   flag1 = 0;
end

return


% Filename: uncODE.m
%
% Parent program:  Unconstrained.m
% Child programs:  sysMatrices.m
%
% Purpose:  This program uses Butcher's fifth-order Runge-Kutta method [28] to calculate the next generalized
%           coordinates and generalized velocities for the unconstrained motion. This program also returns the next
%           time step. The program sysMatrices.m creates the inertia matrix coefficient, and the rest of the
%           unconstrained system equations of motion.
%
% Variables: M - inertia matrix coefficient
%         fi - holds the rest of the unconstrained system equations of motion
%         onek1 to onek6 - k1 to k6 of the Runge-Kutta method for the first equation in the ODE.
%         twok1 to twok6 - k1 to k6 of the Runge-Kutta method for the second equation in the ODE.
%         b2 to b6 - represents the new value for q for equations k2 to k6.
%         c2 to c6 - represents the new value for q_dot for equations k2 to k6.
%
%  The equations of the ODE are (1) q, (2) q_dot.
%  The derivations become (1) q_dot, and (2) q_(double dot) = M^-1*(fi)

function [q, q_dot, t] = uncODE(q, q_dot, t)

global h

% Finding K1 for both equations.
[M1, fi1] = sysMatrices(q,q_dot);

onek1 = q_dot; twok1 = -(inv(M1))*fi1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K2 for both equations.
b2 = q + (h/4)*onek1;
c2 = q_dot + (h/4)*twok1;
[M2, fi2] = sysMatrices(b2,c2);

onek2 = c2;
twok2 = -(inv(M2))*fi2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K3 for both equations.
b3 = q + (h/8)*onek1 + (h/8)*onek2;
c3 = q_dot + (h/8)*twok1 + (h/8)*twok2;
[M3, fi3] = sysMatrices(b3,c3);
onek3 = c3;
twok3 = -(inv(M3))*fi3;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K4 for both equations.
b4 = q - (h/2)*onek2 + h*onek3;
c4 = q_dot - (h/2)*twok2 + h*twok3;
[M4, fi4] = sysMatrices(b4,c4);

onek4 = c4;
twok4 = -(inv(M4))*fi4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K5 for both equations.
b5 = q + (3*h/16)*onek1 + (9*h/16)*onek4;
c5 = q_dot + (3*h/16)*twok1 + (9*h/16)*twok4;
[M5, fi5] = sysMatrices(b5,c5);

onek5 = c5;
twok5 = -(inv(M5))*fi5;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K6 for both equations.
b6 = q - (3*h/7)*onek1 + (2*h/7)*onek2 + (12*h/7)*onek3 - (12*h/7)*onek4 + (8*h/7)*onek5;
c6 = q_dot - (3*h/7)*twok1 + (2*h/7)*twok2 + (12*h/7)*twok3 - (12*h/7)*twok4 + (8*h/7)*twok5;
[M6, fi6] = sysMatrices(b6,c6);

onek6 = c6;
twok6 = -(inv(M6))*fi6;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

q = q + (h/90)*(7*onek1 + 32*onek3 + 12*onek4 + 32*onek5 + 7*onek6);

q_dot = q_dot + (h/90)*(7*twok1 + 32*twok3 + 12*twok4 + 32*twok5 + 7*twok6);

t = t + h;

return
```

B4. Building System Matrices

```
% Filename: sysMatrices.m
%
% Parent program:   uncCode.m, predictLAM.m, conODE.m, NewRaph.m
% Child programs:   (none)
%
%
% Purpose:  This program returns the inertia matrix and the fi vector defined in Hatman et. al's paper [8].  The fi
%           vector includes all the damping and stiffness matrices multiplied by the appropriate generalized vector.
%           The inputs are the generalized coordinates and velocities. Guassian-Quadrature method with sixteen
%           abscissas and weights is used to calculate the volume integrals of the equations of motion coefficients.
%           All the integral boundaries must be converted from [0 to 1] to [-1 to 1]. The term being integrated is
%           converted to: xi = 0.5*(x + 1), therefore, d(xi) = 0.5*d(x).
%           Matrices ISP and SPD, and vectors S and dSQR_S are created [8]. dSQR_S is made by multiplying the
%           derivative of S, with respect to the position along the beam, by itself.
%
```

```
% Variables: i - represents the row of the matrix
%         j - represents the column of the matrix
%         cnt - counter for running through the Guass-Quadrature weights and abscissas vectors
%         aa - integration of sin(ri(i)*xi)*sin(ri(j)*xi) from -1 to 1
%         bb - integration of sin(ri(i)*xi)*cos(ri(j)*xi) from -1 to 1
%         cc - integration of sin(ri(i)*xi)*sinh(ri(j)*xi) from -1 to 1
%         dd - integration of sin(ri(i)*xi)*cosh(ri(j)*xi) from -1 to 1
%         ee - integration of cos(ri(i)*xi)*sin(ri(j)*xi) from -1 to 1
%         ff - integration of cos(ri(i)*xi)*cos(ri(j)*xi) from -1 to 1
%         gg - integration of cos(ri(i)*xi)*sinh(ri(j)*xi) from -1 to 1
%         hh - integration of cos(ri(i)*xi)*cosh(ri(j)*xi) from -1 to 1
%         ii - integration of sinh(ri(i)*xi)*sin(ri(j)*xi) from -1 to 1
%         jj - integration of sinh(ri(i)*xi)*cos(ri(j)*xi) from -1 to 1
%         kk - integration of sinh(ri(i)*xi)*sinh(ri(j)*xi) from -1 to 1
%         ll - integration of sinh(ri(i)*xi)*cosh(ri(j)*xi) from -1 to 1
%         mm - integration of cosh(ri(i)*xi)*sin(ri(j)*xi) from -1 to 1
%         nn - integration of cosh(ri(i)*xi)*cos(ri(j)*xi) from -1 to 1
%         oo - integration of cosh(ri(i)*xi)*sinh(ri(j)*xi) from -1 to 1
%         pp - integration of cosh(ri(i)*xi)*cosh(ri(j)*xi) from -1 to 1

function [M, fi] = sysMatrices(q, q_dot)

global Nmds Acoeff ri L rho A E I AngVel rh

M = zeros(Nmds, Nmds);     % Mass matrix.
D = zeros(Nmds, Nmds);     % Structurally-induced damping matrix.
G = zeros(Nmds, Nmds);     % Gyroscopically-induced damping matrix.
stif = zeros(Nmds, Nmds);  % Represents the stiffening effect caused by rotation.
d_U = zeros(Nmds, Nmds);   % Potential energy contribution to the stiffening matrix.

% Guass-Quadrature weights (W) and abscissas (n) for n = 16.
W = [0.18945061045507; 0.18945061045507; 0.18260341504492; 0.18260341504492; 0.16915651939500;
0.16915651939500; 0.14959598881658; 0.14959598881658; 0.12462897125553; 0.12462897125553;
0.09515851168249; 0.09515851168249; 0.06225352393865; 0.06225352393865; 0.02715245941175;
0.02715245941175];
n = [0.09501250983764; -0.09501250983764; 0.28160355077926; -0.28160355077926; 0.45801677765723; -
0.45801677765723; 0.61787624440264; -0.61787624440264; 0.75540440835500; -0.75540440835500;
0.86563120238783; -0.86563120238783; 0.94457502307323; -0.94457502307323; 0.98940093499165; -
0.98940093499165];

for cnt = 1:16

  x = n(cnt);

  for i = 1:Nmds

    for j = 1:Nmds

      if i == j

        aa = (-cos(ri(i)*0.5*(x + 1))*sin(ri(i)*0.5*(x + 1)) + ri(i)*0.5*(x + 1))/(2*ri(i));

        bb = (sin(ri(i)*0.5*(x + 1)))^2/(2*ri(i));

        cc = (-cos(ri(i)*0.5*(x + 1))*sinh(ri(i)*0.5*(x + 1)) + sin(ri(i)*0.5*(x + 1))*cosh(ri(i)*0.5*(x +
          1)))/(2*ri(i));
    dd = -(-1 + cos(ri(i)*0.5*(x + 1))*cosh(ri(i)*0.5*(x + 1)) - sin(ri(i)*0.5*(x + 1))*sinh(ri(i)*0.5*(x +
       1)))/(2*ri(i));

    ee = bb;

    ff = (cos(ri(i)*0.5*(x + 1))*sin(ri(i)*0.5*(x + 1)) + ri(i)*0.5*(x + 1))/(2*ri(i));

    gg = (-1 + cos(ri(i)*0.5*(x + 1))*cosh(ri(i)*0.5*(x + 1)) + sin(ri(i)*0.5*(x + 1))*sinh(ri(i)*0.5*(x +
       1)))/(2*ri(i));

    hh = (cos(ri(i)*0.5*(x + 1))*sinh(ri(i)*0.5*(x + 1)) + sin(ri(i)*0.5*(x + 1))*cosh(ri(i)*0.5*(x +
       1)))/(2*ri(i));

    ii = cc;

    jj = gg;

    kk = -(-cosh(ri(i)*0.5*(x + 1))*sinh(ri(i)*0.5*(x + 1)) + ri(i)*0.5*(x + 1))/(2*ri(i));

    ll = (-1 + (cosh(ri(i)*0.5*(x + 1)))^2)/(2*ri(i));

    mm = dd;

    nn = hh;

    oo = ll;

    pp = (cosh(ri(i)*0.5*(x + 1))*sinh(ri(i)*0.5*(x + 1)) + ri(i)*0.5*(x + 1))/(2*ri(i));

else

    aa = (sin((ri(i)-ri(j))*0.5*(x + 1))*ri(i) + sin((ri(i)-ri(j))*0.5*(x + 1))*ri(j) - ...
       sin((ri(i)+ri(j))*0.5*(x + 1))*ri(i) + sin((ri(i)+ri(j))*0.5*(x + 1))*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

    bb = -(-2*ri(i) + cos((ri(i)+ri(j))*0.5*(x + 1))*ri(i) - cos((ri(i)+ri(j))*0.5*(x + 1))*ri(j) + ...
       cos((ri(i)-ri(j))*0.5*(x + 1))*ri(i) + cos((ri(i)-ri(j))*0.5*(x + 1))*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

    cc = -(ri(i)*sinh(ri(j)*0.5*(x + 1))*cos(ri(i)*0.5*(x + 1)) - ri(j)*cosh(ri(j)*0.5*(x + 1))*sin(ri(i)*0.5*(x +
       1)))/((ri(j)^2) + (ri(i)^2));

    dd = -(-ri(i) + ri(i)*cosh(ri(j)*0.5*(x + 1))*cos(ri(i)*0.5*(x + 1)) - ri(j)*sinh(ri(j)*0.5*(x +
       1))*sin(ri(i)*0.5*(x + 1)))/((ri(j)^2) + (ri(i)^2));

    ee = -(-2*ri(j) + cos((ri(j)+ri(i))*0.5*(x + 1))*ri(j) - cos((ri(j)+ri(i))*0.5*(x + 1))*ri(i) + ...
       cos((ri(j)-ri(i))*0.5*(x + 1))*ri(j) + cos((ri(j)-ri(i))*0.5*(x + 1))*ri(i))/(2*(ri(j)-ri(i))*(ri(j)+ri(i)));

    ff = (sin((ri(i)-ri(j))*0.5*(x + 1))*ri(i) + sin((ri(i)-ri(j))*0.5*(x + 1))*ri(j) + ...
       sin((ri(i)+ri(j))*0.5*(x + 1))*ri(i) - sin((ri(i)+ri(j))*0.5*(x + 1))*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

    gg = (-ri(j) + ri(j)*cosh(ri(j)*0.5*(x + 1))*cos(ri(i)*0.5*(x + 1)) + ri(i)*sinh(ri(j)*0.5*(x +
       1))*sin(ri(i)*0.5*(x + 1)))/((ri(j)^2) + (ri(i)^2));

    hh = (ri(i)*cosh(ri(j)*0.5*(x + 1))*sin(ri(i)*0.5*(x + 1)) + ri(j)*sinh(ri(j)*0.5*(x + 1))*cos(ri(i)*0.5*(x +
       1)))/((ri(j)^2) + (ri(i)^2));

    ii = -(ri(j)*sinh(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) - ri(i)*cosh(ri(i)*0.5*(x + 1))*sin(ri(j)*0.5*(x +
        1)))/((ri(i)^2) + (ri(j)^2));

      jj = (-ri(i) + ri(i)*cosh(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) + ri(j)*sinh(ri(i)*0.5*(x +
        1))*sin(ri(j)*0.5*(x + 1)))/((ri(i)^2) + (ri(j)^2));

      kk = (sinh((ri(i)+ri(j))*0.5*(x + 1))*ri(i) - sinh((ri(i)+ri(j))*0.5*(x + 1))*ri(j) - ...
        sinh((ri(i)-ri(j))*0.5*(x + 1))*ri(i) - sinh((ri(i)-ri(j))*0.5*(x + 1))*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

      ll = (-2*ri(i) + cosh((ri(i)+ri(j))*0.5*(x + 1))*ri(i) - cosh((ri(i)+ri(j))*0.5*(x + 1))*ri(j) + ...
        cosh((ri(i)-ri(j))*0.5*(x + 1))*ri(i) + cosh((ri(i)-ri(j))*0.5*(x + 1))*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

      mm = -(-ri(j) + ri(j)*cosh(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) - ri(i)*sinh(ri(i)*0.5*(x +
        1))*sin(ri(j)*0.5*(x + 1)))/((ri(i)^2) + (ri(j)^2));

      nn = (ri(j)*cosh(ri(i)*0.5*(x + 1))*sin(ri(j)*0.5*(x + 1)) + ri(i)*sinh(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x +
        1)))/((ri(i)^2) + (ri(j)^2));

      oo = (-2*ri(j) + cosh((ri(j)+ri(i))*0.5*(x + 1))*ri(j) - cosh((ri(j)+ri(i))*0.5*(x + 1))*ri(i) + ...
        cosh((ri(j)-ri(i))*0.5*(x + 1))*ri(j) + cosh((ri(j)-ri(i))*0.5*(x + 1))*ri(i))/(2*(ri(j)-ri(i))*(ri(j)+ri(i)));

      pp = (sinh((ri(i)+ri(j))*0.5*(x + 1))*ri(i) - sinh((ri(i)+ri(j))*0.5*(x + 1))*ri(j) + ...
        sinh((ri(i)-ri(j))*0.5*(x + 1))*ri(i) + sinh((ri(i)-ri(j))*0.5*(x + 1))*ri(j))/(2*(ri(i)-ri(j))*(ri(i)+ri(j)));

    end

    ISP(i,j) = ri(i)*ri(j)*(aa + cc + ii + kk - Acoeff(j)*(dd - bb + ll - jj) - Acoeff(i)*(mm + oo - ee - gg) + ...
      Acoeff(i)*Acoeff(j)*(ff - hh - nn + pp));

    SPD(i,j) = ri(i)*ri(j)*(ri(i)*cos(ri(i)*0.5*(x + 1))*sin(ri(j)*0.5*(x + 1)) + ...
      ri(j)*sin(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) + ri(i)*cos(ri(i)*0.5*(x + 1))*sinh(ri(j)*0.5*(x + 1)) + ...
      ri(j)*sin(ri(i)*0.5*(x + 1))*cosh(ri(j)*0.5*(x + 1)) + ri(i)*cosh(ri(i)*0.5*(x + 1))*sin(ri(j)*0.5*(x + 1)) + ...
      ri(j)*sinh(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) + ri(i)*cosh(ri(i)*0.5*(x + 1))*sinh(ri(j)*0.5*(x + 1)) +
      ri(j)*sinh(ri(i)*0.5*(x + 1))*cosh(ri(j)*0.5*(x + 1)) - Acoeff(j)*(ri(i)*cos(ri(i)*0.5*(x +
      1))*cosh(ri(j)*0.5*(x + 1)) + ...
      ri(j)*sin(ri(i)*0.5*(x + 1))*sinh(ri(j)*0.5*(x + 1)) - ri(i)*cos(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) + ...
      ri(j)*sin(ri(i)*0.5*(x + 1))*sin(ri(j)*0.5*(x + 1)) + ri(i)*cosh(ri(i)*0.5*(x + 1))*cosh(ri(j)*0.5*(x + 1)) + ...
      ri(j)*sinh(ri(i)*0.5*(x + 1))*sinh(ri(j)*0.5*(x + 1)) - ri(i)*cosh(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) +
      ri(j)*sinh(ri(i)*0.5*(x + 1))*sin(ri(j)*0.5*(x + 1))) - Acoeff(i)*(ri(i)*sinh(ri(i)*0.5*(x +
      1))*sin(ri(j)*0.5*(x + 1)) + ...
      ri(j)*cosh(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) + ri(i)*sinh(ri(i)*0.5*(x + 1))*sinh(ri(j)*0.5*(x + 1)) +
      ri(j)*cosh(ri(i)*0.5*(x + 1))*cosh(ri(j)*0.5*(x + 1)) + ri(i)*sin(ri(i)*0.5*(x + 1))*sin(ri(j)*0.5*(x + 1)) - ...
      ri(j)*cos(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) + ri(i)*sin(ri(i)*0.5*(x + 1))*sinh(ri(j)*0.5*(x + 1)) - ...
      ri(j)*cos(ri(i)*0.5*(x + 1))*cosh(ri(j)*0.5*(x + 1))) + ...
      Acoeff(i)*Acoeff(j)*(ri(i)*sinh(ri(i)*0.5*(x + 1))*cosh(ri(j)*0.5*(x + 1)) + ...
      ri(j)*cosh(ri(i)*0.5*(x + 1))*sinh(ri(j)*0.5*(x + 1)) - ri(i)*sinh(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) +
      ri(j)*cosh(ri(i)*0.5*(x + 1))*sin(ri(j)*0.5*(x + 1)) + ri(i)*sin(ri(i)*0.5*(x + 1))*cosh(ri(j)*0.5*(x + 1)) - ...
      ri(j)*cos(ri(i)*0.5*(x + 1))*sinh(ri(j)*0.5*(x + 1)) - ri(i)*sin(ri(i)*0.5*(x + 1))*cos(ri(j)*0.5*(x + 1)) - ...
      ri(j)*cos(ri(i)*0.5*(x + 1))*sin(ri(j)*0.5*(x + 1))));

  end

  S(i,1) = cosh(ri(i)*0.5*(x + 1)) - cos(ri(i)*0.5*(x + 1)) - ...
    Acoeff(i)*(sinh(ri(i)*0.5*(x + 1)) - sin(ri(i)*0.5*(x + 1)));

  dSQR_S(i,1) = ((ri(i))^2)*(cosh(ri(i)*0.5*(x + 1)) + cos(ri(i)*0.5*(x + 1)) - ...
    Acoeff(i)*(sinh(ri(i)*0.5*(x + 1)) + sin(ri(i)*0.5*(x + 1))));
    end

  % The matrices are added to for every weight.
  M = M + rho*A*W(cnt)*(ISP'*q*q'*ISP)/(2*L);
  D = D + rho*A*W(cnt)*(ISP'*q*q_dot'*ISP)/(2*L);
  G = G + rho*A*W(cnt)*AngVel*(ISP'*q*S' - S*q'*ISP);
  stif = stif - rho*A*L*(AngVel^2)*W(cnt)*(-rh*ISP'/(2*L) - 0.25*(n(cnt)+1)*ISP' + ...
     (q'*ISP*q)*ISP'/(4*L^2));
  d_U = d_U + E*I*W(cnt)*((q'*SPD*q)*SPD/(2*L^2) + dSQR_S*dSQR_S')/(2*L^3);

end

% Adding any terms that include the identity matrix and creating fi.
M = M + rho*A*L*eye(Nmds);
stif = stif - rho*A*L*(AngVel^2)*eye(Nmds);
fi = G*q_dot + D*q_dot + stif*q + d_U*q;

return
```

B5. Jump Velocity Calculation

```
% Filename: jump.m
%
% Parent program:  rotate.m
% Child programs:  sysMatrices.m, doubleV.m
%
% Purpose: This program calculates the tip generalized velocity after impact. See Eqn. (4-106). The inputs are the
%          latest generalized coordinate and velocity vectors and the time at which impact occurs. The output is
%          the new generalized velocity vector.
%
% Variables: gradCo - the partial derivative of the constraint equation with respect to the generalized coordinates.
%        dCo_t - the partial derivative of the constraint equation with respect to time.
%        v - beam tip velocity
%        e - coefficient of restitution

function [q_dot] = jump(q, q_dot, t)

global ISP_one S_one AngVel rh L

% This calculates the terms that are needed for Eqn. (4-106).
angle = -(pi/2) + AngVel*t;
gradCo = -((1/L)*cos(angle)*ISP_one'*q + sin(angle)*S_one);
dCo_t = -AngVel*(sin(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) + cos(angle)*q'*S_one);
[M] = sysMatrices(q, q_dot);
inMo = inv(M);

% Calculating the beam tip velocity and the coefficient of restitution.
vx = -AngVel*sin(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) - AngVel*cos(angle)*q'*S_one - ...
   cos(angle)*(1/L)*q'*ISP_one*q_dot - sin(angle)*q_dot'*S_one;
vy = AngVel*cos(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) - AngVel*sin(angle)*q'*S_one - ...
   sin(angle)*(1/L)*q'*ISP_one*q_dot + cos(angle)*q_dot'*S_one;

v = sqrt(vx^2 + vy^2);
e = 0.501*v^(-1/4);
```

```
% Add the generalized velocity input to the jump velocity calculated by Eqn. (4-106).
q_dot = q_dot + ((-(1 + e)*gradCo'*q_dot - (1 + e)*dCo_t)/(gradCo'*inMo*gradCo))*(inMo*gradCo);

return
```

B6. Constrained Motion Programs

```
% Filename: Constrained.m
%
% Parent program:  rotate.m
% Child programs:  predictLAM.m, conODE.m, NewRaph.m
%
% Purpose: This program contains the predictor-corrector algorithm used for the constrained phase of motion. The
%          program predictLAM.m predicts the Lagrange muliplier, the program conODE.m continues with the
%          prediction phase by performing one Runge-Kutta iteration on the constrained system.  NewRaph.m
%          calculates the generalized vectors in the corrector phase. The inputs include the latest generalized
%          coordinates and velocities, the latest time and Lagrange multiplier.  The outputs are the corrected
%          generalized coordinates and velocities, the corrected Lagrange multiplier, and the corresponding time.
%
% Variables: flag2 - made equal to one when the Lagrange multiplier should be predicted.  This happens every time
%                    the system enters into constrained motion. After that, the previous step Lagrange multiplier is
%                    used as the prediction.
%        chek - calculates the percentage difference between the new Lagrange multiplier and the predicted
%               Lagrange multiplier.
%        nlambda - new Lagrange multiplier
%        plambda - predicted Lagrange multiplier
%        nx, px - new and predicted generalized coordinates vector, respectively.
%        nu, pu - new and predicted generalized velocities vector, respectively.

function [nx, nu, ti, nlambda, N] = Constrained(q, q_dot, t, nlambda)

global plambda cnt flag2

% Predictor stage.
if flag2 == 1
  [nlambda] = predictLAM(q(:,cnt), q_dot(:,cnt), t);
  flag2 = 0;
end
[px, pu, ti] = conODE(q(:,cnt), q_dot(:,cnt), t, nlambda);

% Corrector stage (time does not advance).
% This first run of NewRaph.m creates the first corrected values.
[nx, nu, nlambda, N] = NewRaph(px, pu, ti, q, q_dot);

chek = abs((nlambda - plambda)/plambda);

% In this while loop, the above corrected values are reassigned as the predicted values and then these are put into the
% NewRaph.m program again. This runs until the percent change between the new Lagrange multiplier and the
% predicted Lagrange multiplier is less than 1e-5.

while  chek > 1e-5

  px = nx; pu = nu; plambda = nlambda;
  [nx, nu, nlambda, N] = NewRaph(px, pu, ti, q, q_dot);
```

```
    chek = abs((nlambda - plambda)/plambda);

  end

  return


 % Filename: predictLAM.m
 %
 % Parent program:  Constrained.m
 % Child programs:  sysMatrices.m, doubleV.m
 %
 % Purpose: This program predicts the Lagrange multiplier using Eqn. (4-109) using the approach outlined in Section
 %          4.6.  The input is the latest generalized coordinates and velocities and the corresponding time.  The
 %          output is the predicted Lagrange multiplier.  It is named nlambda because it's the newest calculated
 %          value.
 %
 % Variables: mu - coefficient of friction
 %        G - partial derivative of the constraint function with respect to the generalized coordinates.
 %        N - magnitude of G
 %        sqt_VV - the square root of velocity squared.  The velocity is that of the beam tip.
 %        dVV - the derivation of velocity squared.
 %        Qfstar - the generalized frictional force, Eqn. (4-93).

 function [nlambda] = predictLAM(q, q_dot, t)

 global L AngVel ISP_one S_one mu rh

 angle = -(pi/2) + AngVel*t;
 G = -((1/L)*cos(angle)*ISP_one'*q + sin(angle)*S_one);
 [sqt_VV, dVV] = doubleV(q, q_dot);
 N = sqrt(sum(G.^2));

 Qfstar = -0.5*mu*N*dVV/sqt_VV;
 [M, fi] = sysMatrices(q, q_dot);
 iM = inv(M);

 % EQN10 represents the last three terms of Eqn. (4-108)
 EQN10 = q_dot'*(cos(angle)*(1/L)*ISP_one)*q_dot - ...
    2*AngVel*q_dot'*(sin(angle)*(1/L)*ISP_one*q - cos(angle)*S_one);

 % This equation solves for the Lagrange multiplier using Eqn. (4-109) without the term including the magnitude
 % with the Lagrange multiplier.

 lambda = (G'*iM*fi + EQN10)/(G'*iM*G);

 % This equation uses the sign of the above calculated lambda and inserts it into Eqn. (4-109) to solve for the
 % Lagrange multiplier.

 nlambda = (G'*iM*fi + EQN10)/(G'*iM*G + sign(lambda)*G'*iM*Qfstar);

 return


 % Filename: conODE.m
 %
```

```
% Parent program:  Constrained.m
% Child programs:  sysMatrices.m, doubleV.m
%
% Purpose:  This program uses Butcher's fifth-order Runge-Kutta [28] to calculate the next generalized coordinates
%           and generalized velocities for the constrained motion. This program also returns the next time step. The
%           program sysMatrices.m creates the inertia matrix coefficient, and the rest of the constrained system
%           equations of motion.
%
% Variables: M - inertia matrix coefficient
%       fi - holds the rest of the unconstrained system equations of motion
%       angle1 to angle6 - angle of rotation at clamped end of beam
%       G1 to G6 - derivative of the constraint function with respect to the generalized coordinates
%       N1 to N6 - magnitude of G1 to G6
%       sqt_VV1 to sqt_VV6 - the square root of velocity squared.
%       dVV1 to dVV6 - the derivation of velocity squared.
%       Qfstar - the generalized frictional force, Eqn. (4-93).
%       onek1 to onek6 - k1 to k6 of the Runge-Kutta method for the first equation in the ODE.
%       twok1 to twok6 - k1 to k6 of the Runge-Kutta method for the second equation in the ODE.
%       b2 to b6 - represents the new value for q for equations k2 to k6.
%       c2 to c6 - represents the new value for q_dot for equations k2 to k6.
%
% The equations of the ODE are (1) q, (2) q_dot.
% The derivations become (1) q_dot, and (2) q_(double dot) = M^-1*(fi) + M^-1*G*nlambda +
%                                                             M^-1*Qfstar1*abs(nlambda)

function [q, q_dot, t] = conODE(q, q_dot, t, nlambda)

global L AngVel ISP_one S_one mu h

% Finding K1 for both equations.
angle1 = -(pi/2) + AngVel*t;
G1 = -((1/L)*cos(angle1)*ISP_one'*q + sin(angle1)*S_one);
[sqt_VV1, dVV1] = doubleV(q, q_dot);
N1 = sqrt(sum(G1.^2));

Qfstar1 = -0.5*mu*N1*dVV1/sqt_VV1;
[M1, fi1] = sysMatrices(q,q_dot);

onek1 = q_dot;
twok1 = inv(M1)*G1*nlambda + inv(M1)*Qfstar1*abs(nlambda) - inv(M1)*fi1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K2 for both equations.
b2 = q + (h/4)*onek1;
c2 = q_dot + (h/4)*twok1;

angle2 = -(pi/2) + AngVel*(t+(h/4));
G2 = -((1/L)*cos(angle2)*ISP_one'*b2 + sin(angle2)*S_one);
[sqt_VV2, dVV2] = doubleV(b2,c2);
N2 = sqrt(sum(G2.^2));

Qfstar2 = -0.5*mu*N2*dVV2/sqt_VV2;
[M2, fi2] = sysMatrices(b2,c2);

onek2 = c2; % twok2 includes M2, fi2, q = b2, q_dot = c2
twok2 = inv(M2)*G2*nlambda + inv(M2)*Qfstar2*abs(nlambda) - inv(M2)*fi2;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K3 for both equations.
b3 = q + (h/8)*onek1 + (h/8)*onek2;
c3 = q_dot + (h/8)*twok1 + (h/8)*twok2;

angle3 = -(pi/2) + AngVel*(t+(h/4));
G3 = -((1/L)*cos(angle3)*ISP_one'*b3 + sin(angle3)*S_one);
[sqt_VV3, dVV3] = doubleV(b3,c3);
N3 = sqrt(sum(G3.^2));

Qfstar3 = -0.5*mu*N3*dVV3/sqt_VV3;
[M3, fi3] = sysMatrices(b3,c3);

onek3 = c3; % twok3 includes M3, fi3, q = b3, q_dot = c3
twok3 = inv(M3)*G3*nlambda + inv(M3)*Qfstar3*abs(nlambda) - inv(M3)*fi3;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K4 for both equations.
b4 = q - (h/2)*onek2 + h*onek3;
c4 = q_dot - (h/2)*twok2 + h*twok3;

angle4 = -(pi/2) + AngVel*(t+(h/2));
G4 = -((1/L)*cos(angle4)*ISP_one'*b4 + sin(angle4)*S_one);
[sqt_VV4, dVV4] = doubleV(b4,c4);
N4 = sqrt(sum(G4.^2));

Qfstar4 = -0.5*mu*N4*dVV4/sqt_VV4;
[M4, fi4] = sysMatrices(b4,c4);

onek4 = c4; % twok4 includes M4, fi4, q = b4, q_dot = c4
twok4 = inv(M4)*G4*nlambda + inv(M4)*Qfstar4*abs(nlambda) - inv(M4)*fi4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K5 for both equations.
b5 = q + (3*h/16)*onek1 + (9*h/16)*onek4;
c5 = q_dot + (3*h/16)*twok1 + (9*h/16)*twok4;

angle5 = -(pi/2) + AngVel*(t+(3*h/4));
G5 = -((1/L)*cos(angle5)*ISP_one'*b5 + sin(angle5)*S_one);
[sqt_VV5, dVV5] = doubleV(b5,c5);
N5 = sqrt(sum(G5.^2));

Qfstar5 = -0.5*mu*N5*dVV5/sqt_VV5;
[M5, fi5] = sysMatrices(b5,c5);

onek5 = c5; % twok5 includes M5, fi5, q = b5, q_dot = c5
twok5 = inv(M5)*G5*nlambda + inv(M5)*Qfstar5*abs(nlambda) - inv(M5)*fi5;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Finding K6 for both equations.
b6 = q - (3*h/7)*onek1 + (2*h/7)*onek2 + (12*h/7)*onek3 - (12*h/7)*onek4 + (8*h/7)*onek5;
c6 = q_dot - (3*h/7)*twok1 + (2*h/7)*twok2 + (12*h/7)*twok3 - (12*h/7)*twok4 + (8*h/7)*twok5;

angle6 = -(pi/2) + AngVel*(t+h);
G6 = -((1/L)*cos(angle6)*ISP_one'*b6 + sin(angle6)*S_one);
```

```
[sqt_VV6, dVV6] = doubleV(b6,c6);
N6 = sqrt(sum(G6.^2));

Qfstar6 = -0.5*mu*N6*dVV6/sqt_VV6;
[M6, fi6] = sysMatrices(b6,c6);

onek6 = c6; % twok4 includes M6, fi6, q = b6, q_dot = c6
twok6 = inv(M6)*G6*nlambda + inv(M6)*Qfstar6*abs(nlambda) - inv(M6)*fi6;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

q = q + (h/90)*(7*onek1 + 32*onek3 + 12*onek4 + 32*onek5 + 7*onek6);

q_dot = q_dot + (h/90)*(7*twok1 + 32*twok3 + 12*twok4 + 32*twok5 + 7*twok6);

t = t + h;

return


% Filename: NewRaph.m
%
% Parent program:  Constrained.m
% Child programs:  doubleV.m, sysMatrices.m
%
%
% Purpose: This program completes one iteration of the corrector algorithm. It's inputs are the predicted generalized
%          coordinates and velocities, the time, and the generalized coordinates and velocities matrices.  The outputs
%          are the corrected generalized coordinate and velocity vectors, and Lagrange multiplier and the latest N. See
%          Section 4.6.
%
% Variables: angle - angle of rotation at clamped end of beam
%          x2 - generalized coordinate vector used to calculate extra term in second system defined in Eqns. (4-111).
%               See also Eqn. (4-114).
%          C2 - constraint function when q = x2.
%          gradC2 - partial derivative of constraint function with respect to the generalized coordinates when q = x2.
%          delta - extra term in second system defined in Eqns. (4-111). See also Eqn. (4-113).
%          q(:,cnt) - equivalent to x(n-1)
%          q_dot(:,cnt) - equivalent to u(n-1)
%          G - derivative of the constraint function with respect to the generalized coordinates
%          N - magnitude of G
%          sqt_VV - the square root of velocity squared.
%          dVV - the derivation of velocity squared.
%          Qfstar - the generalized frictional force, Eqn. (4-93).
%          dC_t - partial derivative of the constraint function with respect to time.
%          nx, px - new and predicted generalized coordinates vector, respectively.
%          nu, pu - new and predicted generalized velocities vector, respectively.
%          nlambda - new Lagrange multiplier
%          plambda - predicted Lagrange multiplier

function [nx, nu, nlambda, N] = NewRaph(px, pu, ti, q, q_dot)

global L rh AngVel ISP_one S_one d cnt mu h

angle = -(pi/2) + AngVel*ti;
X = cos(angle)*(rh + L - (0.5/L)*px'*ISP_one*px) - sin(angle)*px'*S_one;
Y = sin(angle)*(rh + L - (0.5/L)*px'*ISP_one*px) + cos(angle)*px'*S_one;
```

```
C = X - d;
fprintf('NEWangle = %g\t\tC = %g\t\tX = %e\t\tY = %g\n', angle, C, X, Y);

x2 = q(:,cnt) + h*pu;
C2 = cos(angle)*(rh + L - (0.5/L)*x2'*ISP_one*x2) - sin(angle)*x2'*S_one - d;
gradC2 = -((1/L)*cos(angle)*ISP_one'*x2 + sin(angle)*S_one);
delta = C2/(gradC2'*gradC2);

%dC/dq at xi = 1;
gradC = -((1/L)*cos(angle)*ISP_one'*px + sin(angle)*S_one);

% Next generalized coordinates.
nx = q(:,cnt) + h*pu - delta*gradC2;

G = -((1/L)*cos(angle)*ISP_one'*nx + sin(angle)*S_one);
[sqt_VV, dVV] = doubleV(nx, pu);
N = sqrt(sum(G.^2));

Qfstar = -0.5*mu*N*dVV/sqt_VV;

% This is Eqn. (4-117).
dC_t = -G'*pu;
[M, fi] = sysMatrices(nx,pu);
iM = inv(M);

% This equation solves for the Lagrange multiplier using Eqn. (4-116) without the term including the magnitude
% with the Lagrange multiplier.

lambda = (h*G'*iM*fi - G'*q_dot(:,cnt) - dC_t)/(h*G'*iM*G);

% This equation uses the sign of the above calculated lambda and inserts it into Eqn. (4-116) to solve for the
% Lagrange multiplier.

nlambda = (h*G'*iM*fi - G'*q_dot(:,cnt) - dC_t)/(h*G'*iM*G + h*sign(lambda)*G'*iM*Qfstar);

nu = q_dot(:,cnt) - h*iM*fi + h*nlambda*iM*G + h*abs(nlambda)*iM*Qfstar;
fprintf('nlambda = %g\n', nlambda);

return


% Filename: LagMultip.m
%
% Parent program:  rotate.m
% Child programs:  uncODE.m
%
% Purpose:  This program updates the latest solution of the latest corrector iteration and uses the sign of the
%           corrected Lagrange multiplier to direct the program into the correct phase of motion. If the sign is
%           negative, constrained motion continues. If the sign is positive, the motion becomes unconstrained and
%           the program continues to TipDistance.m. The inputs and outputs are listed in the variables
%
% Variables:
%   Inputs: q - generalized coordinates matrix
%           q_dot - generalized vectors matrix
%           t - time at the last updated solution
%           ti - time at last constrained motion iteration (conODE.m)
%        nlambda - the latest corrected Lagrange multiplier
%        nx - the latest corrected generalized coordinate vector
%        nu - the latest corrected generalized velocity vector
%        N - the latest magnitude of the partial derivative of the constraint function with respect to the generalized
%           coordinates.
%        prevLAM - the previous Lagrange multiplier
%   Outputs are the updated solutions for:
%        q - the generalized coordinate vector
%        q_dot - the generalized velocity vector
%        t - time
%        X, Y - the global beam tip coordinates
%        C - the constraint function
%        pam - the normal force Eqn. (4-86)
%        TipDist - the beam tip distance from the flat rigid surface
%        a, b, f - variables that hold the latest generalized coordinates,generalized velocities and time
%   Others: flag3 - made equal to one when the program must enter the TipDistance.m in the constrained phase of
%                    motion. This happens when the Lagrange multiplier is showing the system not to be in contact,
%                    but the constraint function does not reflect this. Therefore, flag3 sends the program to complete
%                    several unconstrained iterations until the constraint function does reflect this.

function [q, q_dot, t, X, Y, C, pam, TipDist, a, b, f] = LagMultip(q, q_dot, t, ti, nlambda, nx, nu, N, prevLAM)

global cnt AngVel d rh L ISP_one S_one plambda flag1 flag3

% If nlambda converges and is still negative, contact is still occurring and time advances.

if nlambda < 0

   % The solution is updated. a=b=f=0 because the program does not need to go into TipDistance.m, and therefore,
   % their values are irrelevent for the time being.

   cnt = cnt + 1; t = ti;
   q = nx; q_dot = nu; plambda = nlambda;

   angle = -(pi/2) + AngVel*t;
   X = cos(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) - sin(angle)*q'*S_one;
   Y = sin(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) + cos(angle)*q'*S_one;
   C = 0;
   pam = N*abs(nlambda);
   TipDist = C;
   a = 0; b = 0; f = 0;

else

   % If nlambda converges and is positive, a linear interpolation is done to find the moment when nlambda is equal
   % to zero.

   nt = ti - nlambda*(t(cnt) - ti)/(prevLAM - nlambda);
   nq = nx - nlambda*(q(:,cnt) - nx)/(prevLAM - nlambda);
   nq_dot = nu - nlambda*(q_dot(:,cnt) - nu)/(prevLAM - nlambda);

   % The solution is updated. Also, an unconstrained motion Runge-Kutta iteration is done.

   cnt = cnt + 1; t = nt;
   q = nq; q_dot = nq_dot;
   angle = -(pi/2) + AngVel*t; pam = 0;
   X = cos(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) - sin(angle)*q'*S_one;
   Y = sin(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) + cos(angle)*q'*S_one;
   C = X - d;
   [a, b, f] = uncODE(q, q_dot, t);

   % The beam tip distance from the flat rigid surface is calculated.

   angle = -(pi/2) + AngVel*f;
   TipDist = cos(angle)*(rh + L - (0.5/L)*a'*ISP_one*a) - sin(angle)*a'*S_one - d;
   flag3 = 1;

end

if angle >= pi/2
   flag1 = 0;
end

return


% Filename: TipDistance.m
%
% Parent program:  rotate.m
% Child programs:  uncODE.m
%
% Purpose: This program takes the sign of the TipDist variable, and directs the program to the unconstrained phase
%          of motion with the correct values for the latest solution.

function [q, q_dot, t, X, Y, C, pam] = TipDistance(a, b, f, TipDist)

global cnt AngVel d rh L h ISP_one S_one flag1 flag3

% If the sign of TipDist is negative, the solution is updated right away and the program goes into unconstrained
% motion.

if TipDist < 0

   cnt = cnt + 1;
   t = f; q = a; q_dot = b;
   angle = -(pi/2) + AngVel*f;
   X = cos(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) - sin(angle)*q'*S_one;
   Y = sin(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) + cos(angle)*q'*S_one;
   C = TipDist; pam = 0;

else

   % If the sign is positive, Runge-Kutta iterations of the unconstrained system system are run until the TipDist sign
   % becomes negative. Therefore, the while loop runs as long as TipDist is greater than or equal to zero.

   while TipDist >= 0

      h = 1e-6;
      [a, b, f] = uncODE(a, b, f);
      angle = -(pi/2) + AngVel*f;
      TipDist = cos(angle)*(rh + L - (0.5/L)*a'*ISP_one*a) - sin(angle)*a'*S_one - d;
      fprintf('angle = %g\t\tTipDist = %g\t\tX = %g\n', angle, TipDist, TipDist+d);
   end

   % The latest values are used to update the solution.

   cnt = cnt + 1;
   t = f; q = a; q_dot = b;
   angle = -(pi/2) + AngVel*t;
   X = cos(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) - sin(angle)*q'*S_one;
   Y = sin(angle)*(rh + L - (0.5/L)*q'*ISP_one*q) + cos(angle)*q'*S_one;
   C = TipDist; pam = 0;

end

% This flag is set to zero and will be set back to one in LagMultip.m if needed.
flag3 = 0;

if angle >= pi/2
   flag1 = 0;
end

return
```

B7. Calculating Velocity Terms

```
% Filename: doubleV.m
%
% Parent program:  predictLAM.m, NewRaph.m
% Child programs:  (none)
%
% Purpose:  This program calculates the velocity of the beam tip. The inputs are the latest generalized coordinate
%           and velocity vectors. The output is of the square root of velocity squared and the derivative of the
%           velocity squared. See Eqns. (4-95-4-96).
%
% Variables: rf - position vector at the free end.
%         drf - derivative of variable rf with respect to the generalized coordinates
%         VV - velocity squared
%         sqt_VV - the square root of velocity squared.
%         dVV - the derivation of velocity squared.

function [sqt_VV, dVV] = doubleV(q, q_dot)

global L rh AngVel ISP_one S_one Nmds

empty = zeros(1,Nmds);

rf = [rh + L - (0.5*q'*ISP_one*q)/L; q'*S_one; 0];
drf = [-q'*ISP_one/L; S_one'; empty];
m1 = [1 0 0; 0 1 0; 0 0 0];
m2 = [0 -1 0; 1 0 0; 0 0 0];

VV = AngVel^2*rf'*m1*rf + 2*AngVel*q_dot'*drf'*m2*rf + q_dot'*drf'*drf*q_dot;
sqt_VV = sqrt(VV);
dVV = 2*AngVel*drf'*m2*rf + 2*drf'*drf*q_dot;
return
```

@ BL-14-21