ADAMS/Solver Primer

Ann Arbor
August, 2004

Dan Negrut
Dan.Negrut@mscsoftware.com

Andrew Dyer
Andrew.Dyer@mscsoftware.com

1. Introduction

The purpose of this document is to introduce the ADAMS user to the theory that underlies the ADAMS/Solver. A basic understanding of the theory and the ways in which the solver works can help the user make good modeling decisions, or choose a set of simulation parameters that will lead to faster and more reliable simulations.

This document provides in section 1.1 a very brief introduction to the Newton-Raphson method for the solution of non-linear systems. Virtually all analysis modes in ADAMS use this algorithm, and the user is strongly encouraged to understand the basics. This is essential for understanding the informational and debugging messages that ADAMS/Solver provides, why an equilibrium analysis is challenging, how changing the Jacobian evaluation pattern is going to impact the convergence properties of the solver, and list goes on.

In the first part of the document (sections 2 through 6) the focus is on equation formulation and solution. The reader is introduced to the concept of generalized coordinates, as well as the equations that govern the most representative types of analyses that can be performed by the ADAMS/Solver; i.e., Initial Condition Analysis, Kinematic Analysis, Dynamics Analysis, and Static/Quasi-static Analysis. These equation sets are rather abstract, and therefore this discussion is followed in section 7 by a simple test case that reinforces in a simplified two-dimensional framework the process of deriving the equations for each analysis type.

Section 8 discusses ways in which an ADAMS/Solver simulation can be made to run faster and more robustly. Subsections 8.1 and 8.2 are focused on dynamics and statics simulations as they are typically more challenging. The discussion will cover relevant parameters associated with the INTEGRATOR and EQUILIBRIUM statements. Subsection 8.3 provides a set of recommendations and good practices for ADAMS modeling. Subsection 8.4 discusses the issue of validating your results in ADAMS, while subsection 8.5 explains how the user can get useful debug information from ADAMS/Solver.

1.1. Solving Nonlinear Equations. The Newton Method

The Newton Method is widely used by ADAMS/Solver and is an important numerical algorithm to understand in order to produce models that lead to robust and fast simulations. In one dimension, the Newton-Raphson algorithm finds the root x^* of a *non-linear* equation

$$f(x) = 0 \tag{1}$$

where the function $f : \mathbb{R} \to \mathbb{R}$ is assumed to be differentiable. If an initial approximation $x^{(0)}$ of the root is provided, a new configuration $x^{(1)}$, hopefully closer to the root x^* is computed as

$$x^{(1)} = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})} \tag{2}$$

where $f'()$ is the derivative of the function with respect to the variables on which it depends.

This strategy of updating the value of x is obtained by linearizing the function f at the point $x^{(0)}$. Thus,

$$f(x) \approx f(x^{(0)}) + f'(x^{(0)})(x - x^{(0)}) \tag{3}$$

Rather than solving $f(x) = 0$, the root of the right hand side expression in Eq.(3) is sought. This leads to a *linear* equation, and its root is denoted by $x^{(1)}$:

$$f(x^{(0)}) + f'(x^{(0)})(x^{(1)} - x^{(0)}) = 0 \;\Rightarrow\; x^{(1)} = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})}$$

which is the configuration update defined in Eq.(2). The algorithm continues by setting $x^{(0)} \leftarrow x^{(1)}$ and performing another iteration as indicated in Eq.(2), and illustrated in Figure 1.

The method just described is called Newton-Raphson, or sometimes simply Newton. In general, this approach leads to a quadratic rate of convergence to the root x^* provided the iterative process is started close enough to the root. The drawback of the method is that as indicated in Eq.(2), the new update $x^{(1)}$ requires the computation of the derivative both the function $f(x^{(0)})$ and the derivative $f'(x^{(0)})$. At the cost of one function and one derivative evaluation per iteration, this method is considered to be rather expensive. An alternative is to only update the derivative once in a while, and to recycle the derivative computed at an older configuration for several iterations. This method is called Newton-like, and the convergence rate is linear.

Referring to the ADAMS/Solver documentation, consider the INTEGRATOR statement. One of its attributes is the PATTERN setting, which enables the user to dictate how often the derivative is to be updated. Computing the derivative (also called the Jacobian), is in general an expensive operation and should be done as seldom as possible. On the other hand, if it's done too seldom, the speed of convergence will suffer as there will be degradation from quadratic (for Newton-Raphson), to linear rate of convergence (for Newton-like methods). Figure 1 shows a geometric interpretation of the Newton-Raphson method: starting with the configuration $x^{(0)}$, the nonlinear function is locally approximated by a straight line. The new configuration $x^{(1)}$ is taken to be the place where the straight line intersects the x-axis (the solution of the linear equation of Eq.(3)). Moving on, the new tangent is determined; i.e., a new derivative $f(x^{(1)})$ is computed, and the new configuration $x^{(2)}$ is obtained. The Newton-Raphson iteration sequence is thus carried out as:

$$x^{(1)} = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})}$$

$$x^{(2)} = x^{(1)} - \frac{f(x^{(1)})}{f'(x^{(1)})}$$

$$x^{(3)} = x^{(2)} - \frac{f(x^{(2)})}{f'(x^{(2)})}$$

…

…until either the change in the configuration, $\frac{f(x^{(n)})}{f'(x^{(n)})}$, or the value of the of the function, $f(x^{(n)})$, is below a specified threshold (either exit criteria could be used, in general).

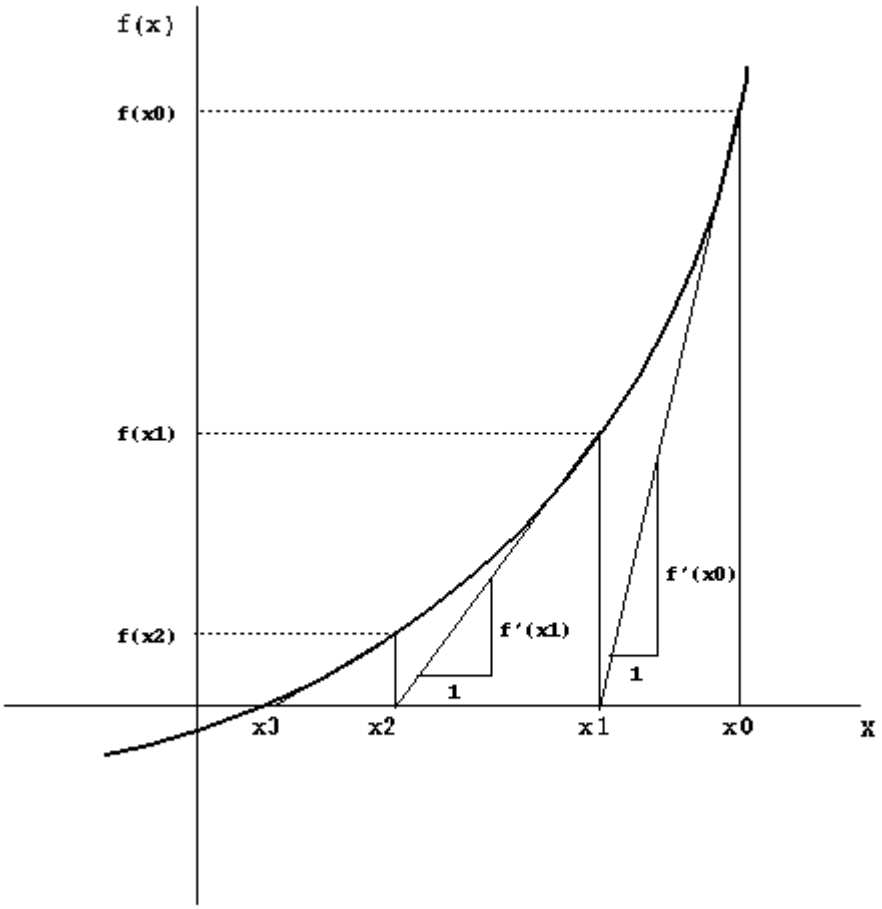


Figure 1. Newton-Raphson Method

Note in Figure 1 that the computed configurations get closer and closer to the point where the non-linear function $f(x)$ intersects the horizontal axis, that is it gets closer and closer to the solution of $f(x) = 0$.

In Figure 2, the Newton-like method is illustrated by adding dotted lines to the solid Newton-Raphson lines from Figure 1. The dotted line represents the slope of the curve as computed at the point $x^{(0)}$, which is used to find not only $x^{(1)}$ (as is the case with the Newton-Raphson method), but also $x^{(2)}$, $x^{(3)}$, and $x^{(4)}$. Note that the slope is re-evaluated at $x^{(4)}$ and the iterative process continues with finding $x^{(5)}$.

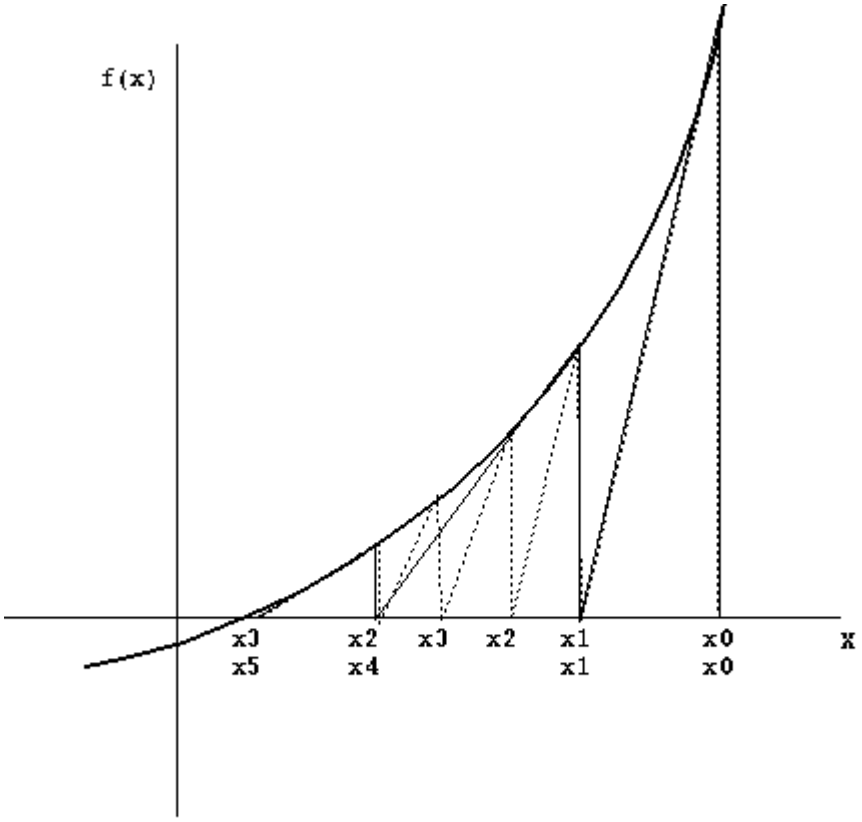


Figure 2. Newton-like Method

The interesting thing to notice is that the slope of the line stays the same for four consecutive iterations; in ADAMS syntax, this would be equivalent to specifying a PATTERN=T:F:F:F, (this is, by the way, the default setting for this attribute in the ADAMS/Solver INTEGRATOR statement).

$$x^{(1)} = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})}$$

$$\dots$$

$$x^{(4)} = x^{(3)} - \frac{f(x^{(3)})}{f'(x^{(0)})}$$

$$x^{(5)} = x^{(4)} - \frac{f(x^{(4)})}{f'(x^{(4)})}$$

$$\dots$$

As suggested by Figure 2, the Newton-Raphson algorithm (represented with solid line) takes two iterations to achieve nearly the same value obtained by the Newton-like approach after four iterations. This is a nice way to see the superior convergence rate associated with the Newton-Raphson method. However, as pointed our earlier, the efficacy of this convergence rate must be balanced by the expensive of evaluating the Jacobian, especially when the dimension of the problem is larger than one. This situation that is discussed in greater detail below.

In general, ADAMS/Solver deals with system of nonlinear equations in multiple unknowns. Thus, the root *x* from the simple example above is replaced by the *n*-dimensional vector, $\mathbf{q} \in \mathbb{R}^n$, and the system that needs to be solved assumes the form

$$\mathbf{f}(\mathbf{q}) = \begin{bmatrix} f_1(\mathbf{q}) \\ f_2(\mathbf{q}) \\ \vdots \\ f_{n-1}(\mathbf{q}) \\ f_n(\mathbf{q}) \end{bmatrix} = \mathbf{0} \qquad (4)$$

Qualitatively, the approach is identical to the one-dimensional case: the non-linear function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is locally linearized at a point $\mathbf{q}^{(0)}$, and the linear approximation is equated to zero. The resulting linear system is solved for a new value $\mathbf{q}^{(1)}$ that is hopefully closer to the root $\mathbf{q}^*$. The linearization of the function produces

$$\mathbf{f}(\mathbf{q}) \approx \mathbf{f}(\mathbf{q}^{(0)}) + \mathbf{F}(\mathbf{q}^{(0)})(\mathbf{q} - \mathbf{q}^{(0)})$$

where $\mathbf{F}(\mathbf{q}^{(0)})$, the Jacobian at $\mathbf{q}^{(0)}$, is defined as:

$$\mathbf{F}(\mathbf{q}^{(0)}) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{q}}\right|_{\mathbf{q}=\mathbf{q}^{(0)}} = \left[\frac{\partial f_i}{\partial q_j}\right]_{\mathbf{q}=\mathbf{q}^{(0)}}$$

The new configuration $\mathbf{q}^{(1)}$ is computed by solving the linear problem:

$$\mathbf{f}(\mathbf{q}^{(0)}) + \mathbf{F}(\mathbf{q}^{(0)})(\mathbf{q}^{(1)} - \mathbf{q}^{(0)}) = \mathbf{0}$$

which leads to:

$$\mathbf{q}^{(1)} = \mathbf{q}^{(0)} - \left[\mathbf{F}(\mathbf{q}^{(0)})\right]^{-1} \mathbf{f}(\mathbf{q}^{(0)}) \tag{5}$$

..for the multi-dimensional case. This is the analog of Eq.(2).

2. Definitions, Notations, Conventions

2.1. Generalized Coordinates used in ADAMS

ADAMS/Solver is used to simulate the time evolution of a mechanical system. At each moment of time, ADAMS/Solver is capable of indicating the position, orientation, and speeds associated with each part in the model. The position and orientation of a part is monitored by means of what is called generalized coordinates. The choice of generalized coordinates is not unique, for instance one could choose Cartesian coordinates, or spherical coordinates. Likewise, one could choose either global or relative coordinates, in which the configuration of a part (position, orientation, speeds) is defined relative to the origin or another part, respectively. The key observation is that once a set of generalized coordinates is selected it is expected that it will uniquely define the configuration of each part of the system at a given instance of time.

In this context, in ADAMS/Solver, the position of a rigid body is defined by three Cartesian coordinates x, y, and z.

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{6}$$

The orientation of a rigid body is defined by a set of three Euler angles that correspond to the 3-1-3 sequence rotation: ψ, θ, and ϕ, respectively. These three angles are stored in an array

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \psi \\ \phi \\ \theta \end{bmatrix} \tag{7}$$

The set of generalized coordinates associated with rigid body i in ADAMS is denoted in what follows by

$$\mathbf{q}_i = \begin{bmatrix} \mathbf{p}_i \\ \boldsymbol{\varepsilon}_i \end{bmatrix} \tag{8}$$

Based on this choice of generalized coordinates, the body longitudinal and angular velocity are obtained as

$$\mathbf{u} = \dot{\mathbf{p}} \tag{9}$$

$$\bar{\boldsymbol{\omega}} = \mathbf{B}\dot{\boldsymbol{\varepsilon}} \equiv \mathbf{B}\boldsymbol{\zeta} \tag{10}$$

where

$$\mathbf{B} = \begin{bmatrix} \sin\phi\sin\theta & 0 & \cos\phi \\ \cos\phi\sin\theta & 0 & -\sin\phi \\ \cos\theta & 1 & 0 \end{bmatrix} \tag{11}$$

and $\bar{\boldsymbol{\omega}}$ is the body angular velocity expressed in the body-fixed coordinate system. Equation (11) is important as it defines the relationship between the angular velocity of the body (an intrinsic characteristic of the body), and the choice of generalized coordinates (which, as mentioned earlier, can be chosen in a variety of ways). Finally, note the relationship between the time derivative of the body orientation matrix $\mathbf{A}$ and angular velocity $\bar{\boldsymbol{\omega}}$:

$$\dot{\mathbf{A}} = \mathbf{A}\tilde{\bar{\boldsymbol{\omega}}} \tag{12}$$

where the orientation matrix $\mathbf{A}$ is defined in terms of the 3-1-3 Euler rotation sequence ψ, θ, and φ as:

$$\mathbf{A} = \begin{bmatrix} \cos\psi\cos\phi - \sin\psi\cos\theta\sin\phi & -\cos\psi\sin\phi - \sin\psi\cos\theta\cos\phi & \sin\psi\sin\theta \\ \sin\psi\cos\phi + \cos\psi\cos\theta\sin\phi & -\sin\psi\sin\phi + \cos\psi\cos\theta\cos\phi & -\cos\psi\sin\theta \\ \sin\theta\sin\phi & \sin\theta\cos\phi & \cos\theta \end{bmatrix}$$

…which is used to compute the derivatives for rotating bodies in three-dimensional space. Note that "~" represents the skew-symmetric operator applied to a vector quantity. In general, for a vector $\mathbf{a}^T = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}$,

$$\tilde{\mathbf{a}} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

Also, a vector quantity marked with an over-bar indicates that the vector is expressed in a local body reference frame.

For an entire mechanical system models containing nb bodies, the vector:

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_1^{\mathrm{T}} & \mathbf{q}_2^{\mathrm{T}} & \cdots & \mathbf{q}_{nb}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}^{\mathrm{T}} \tag{13}$$

with $n = 6 \cdot nb$ will describe at a given time the position and orientation of each body in the system.

2.2. Joints in ADAMS

Joints in ADAMS are regarded as constraints that act among some of the coordinates q_1 through q_n of Eq.(13). From a mathematical perspective, such a constraint assumes the expression:

$$\Phi(\mathbf{q}) = 0 \tag{14}$$

…which is simply an algebraic constraint.

For example, a revolute joint acting between two bodies would induce a set of five constraints to allow one degree of freedom between the two bodies connected by this joint.

The collection of all constraints induced by the joints present in the model is denoted by $\mathbf{\Phi}$:

$$\mathbf{\Phi}(\mathbf{q}) = \begin{bmatrix} \mathbf{\Phi}_1^{\mathrm{T}}(\mathbf{q}) & \mathbf{\Phi}_2^{\mathrm{T}}(\mathbf{q}) & \cdots & \mathbf{\Phi}_{nj}^{\mathrm{T}}(\mathbf{q}) \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \Phi_1(\mathbf{q}) & \Phi_2(\mathbf{q}) & \cdots & \Phi_m(\mathbf{q}) \end{bmatrix}^{\mathrm{T}} \tag{15}$$

where nj is the number of joints in the model, and m is the sum of the number of constraints induced by all joints. Note that $\mathbf{q} \in \mathbf{R}^n$, while $\mathbf{\Phi} \in \mathbf{R}^m$. Typically, $m < n$; i.e., the number of generalized coordinates is larger than the number of constraints they must satisfy.

By taking one time derivative of the position kinematic constraint equations of Eq.(15), the velocity kinematic constraint equations are obtained as

$$\mathbf{\Phi}_{\mathbf{q}} \dot{\mathbf{q}} = \mathbf{0} \tag{16}$$

By taking yet another time derivative of Eq.(16), the acceleration kinematic constraint equations are obtained as

$$\mathbf{\Phi}_{\mathbf{q}} \ddot{\mathbf{q}} = -\left(\mathbf{\Phi}_{\mathbf{q}} \dot{\mathbf{q}}\right)_{\mathbf{q}} \dot{\mathbf{q}} \equiv \boldsymbol{\tau} \tag{17}$$

Equations (15) through (17) can be seen as conditions that the generalized coordinates array $\mathbf{q}$ along with its first and second time derivatives must satisfy. This is to ensure

that the evolution of the mechanical system makes sense; i.e., the mechanism is assembled, and the parts move such that the constraints imposed by joints are obeyed at every time.

2.3. Motions in ADAMS

From a mathematical perspective, motions indicate that a generalized coordinate of the system, or an expression depending on generalized coordinates, explicitly depends on time. As an example, consider a simple pendulum connected to ground through a revolute joint. A motion might impose that the angle associated with its unique degree of freedom will change in time like $\alpha = \sin(10\pi \cdot t)$.

Generally, a motion is represented as a time dependent constraint equation:

$$\Phi(\mathbf{q}, t) = \mathbf{0} \tag{18}$$

Revisiting the definition of the position, velocity, and acceleration kinematic constraint equations, both joints and motion constraints may, in general, be written as:

$$\mathbf{\Phi}(\mathbf{q}, t) = \mathbf{0} \tag{19}$$

$$\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}, t) \cdot \dot{\mathbf{q}} = -\mathbf{\Phi}_{t}(\mathbf{q}, t) \tag{20}$$

$$\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}, t) \cdot \ddot{\mathbf{q}} = -\left(\mathbf{\Phi}_{\mathbf{q}} \dot{\mathbf{q}}\right)_{\mathbf{q}} \dot{\mathbf{q}} - 2\mathbf{\Phi}_{\mathbf{q}t} \dot{\mathbf{q}} - \mathbf{\Phi}_{tt}(\mathbf{q}, t) \tag{21}$$

Equations (20) and (21) are obtained by taking one and respectively two time derivatives of the position kinematic constraint equation of Eq.(19). A set of generalized coordinates is said to be consistent, if it satisfies the position kinematic constraint equations. Likewise, a set of generalized velocities is considered consistent if, for a consistent position configuration, $\dot{\mathbf{q}}$ satisfies the velocity kinematic constraint equations of Eq.(20).

3. Initial Condition Analysis

Initial Condition (IC) Analysis is concerned with determining a consistent configuration for the mechanical system model at the beginning of the simulation. During IC analysis, the mechanism must be *assembled* and the velocities of the parts in the mechanism must be *consistent*.

To be assembled, the generalized coordinates $\mathbf{q}$ must satisfy all constraint equations at time t_0 . ; i. e.,

$$\mathbf{\Phi}\left(\mathbf{q}, t_0\right) = \mathbf{0} \tag{22}$$

while for the generalized velocities to be consistent, they must satisfy the velocity kinematic constraint equation

$$\mathbf{\Phi}_{\mathbf{q}}\left(\mathbf{q}, t\right) \cdot \dot{\mathbf{q}} = -\mathbf{\Phi}_{t}\left(\mathbf{q}, t\right) \tag{23}$$

3.1. Position Initial Condition Analysis

During IC analysis, the user might want to fix the value of some of the generalized coordinates (q_{i_1} , q_{i_2} , etc.), from the generalized coordinate array, Eq.(13). In other words, if the user prescribes the position of a body in the system to be $q_{13} = 0.9$, $q_{14} = -1.0$, $q_{16} = 0$, the solver should assemble the mechanism and, at the same time, do its best to satisfy the prescribed conditions. This IC analysis is solved in ADAMS via an optimization approach. The constrained optimization problem solved minimizes the cost function

$$f\left(q_1, \ldots, q_n\right) = \frac{1}{2} w_1 \left(q_1 - q_1^0\right)^2 + \ldots + \frac{1}{2} w_n \left(q_n - q_n^0\right)^2 \tag{24}$$

subject to the constraint equations $\mathbf{\Phi}\left(\mathbf{q}, t_0\right) = \mathbf{0}$. In Eq.(24), the values w_i are weight factors, while q_i^0 can be regarded as the initial generalized coordinates inducing an initial configuration of the system. Notice that this configuration $\mathbf{q}^0 = \begin{bmatrix} q_1^0 & q_2^0 & \ldots & q_n^0 \end{bmatrix}^{\mathrm{T}}$ need not be consistent (i.e., it may not satisfy the constraints).

The user may prescribe that some of the entries in the $\mathbf{q}^0$ array; i.e., $q_{i_1}^0$, $q_{i_2}^0$, etc., are to be regarded "exact". As it will be justified shortly, the corresponding weights w_{i_1}, w_{i_2}, etc., will be given large values; i.e., values like 10^{10}. The remaining weights, namely the ones corresponding to generalized coordinates q_i^0 that the user did not specify are given values of 1. With this approach, the constrained optimization problem solution will keep the user-imposed IC values almost unchanged, while adjusting the "free-to-change" generalized coordinates. Notice that "the solution of the problem" means minimizing the cost function while satisfying the constraints.

In matrix notation, the constrained optimization problem reads:

For $\mathbf{q} \in \mathbf{R}^n$, minimize

$$f(\mathbf{q}) = \frac{1}{2}\left(\mathbf{q}-\mathbf{q}^0\right)^{\mathrm{T}} \mathbf{W}\left(\mathbf{q}-\mathbf{q}^0\right) \tag{25}$$

subject to

$$\mathbf{\Phi}\left(\mathbf{q}, t_0\right) = \mathbf{0} \tag{26}$$

In Eq.(25), $\mathbf{W}$ is a diagonal matrix of weights,

$$\mathbf{W} = diag\left(w_1, w_2, \ldots, w_n\right) \tag{27}$$

while the constraints of Eq.(26) that must be satisfied in the optimization problem are exactly the position kinematic constraints of Eq.(19).

ADAMS approximates the non-convex optimization problem of Eqs.(25) and (26) by a succession of convex problems that are guaranteed to have a solution, which can potentially be found in one iteration. Thus, the set of non-linear constraint equations of Eq.(26) is linearized in the vicinity of $\mathbf{q}^0$

$$\mathbf{\Phi}\left(\mathbf{q}, t_0\right) = \mathbf{\Phi}\left(\mathbf{q}^0, t_0\right) + \mathbf{\Phi}_{\mathbf{q}}\left(\mathbf{q}^0, t_0\right)\left(\mathbf{q}-\mathbf{q}^0\right) \tag{28}$$

With Equation (28) replacing Eq.(26) and the notation $\mathbf{d} \equiv \mathbf{q}-\mathbf{q}^0$ the now convex optimization problem reads

Minimize $$f(\mathbf{d}) = \frac{1}{2}\mathbf{d}^{\mathrm{T}}\mathbf{W}\mathbf{d} \tag{29}$$

Subject to $$\boldsymbol{\Phi}\left(\mathbf{q}^0, t_0\right) + \boldsymbol{\Phi}_{\mathbf{q}}\left(\mathbf{q}^0, t_0\right)\mathbf{d} = \mathbf{0} \tag{30}$$

To solve the convex constrained optimization problem of Eqs.(29) and (30), define the optimization Lagrangian:

$$F\left(\mathbf{d}, \boldsymbol{\lambda}\right) = f\left(\mathbf{d}\right) + \boldsymbol{\lambda}^{\mathrm{T}}\left(\boldsymbol{\Phi}\left(\mathbf{q}^0\right) + \boldsymbol{\Phi}_{\mathbf{q}}\left(\mathbf{q}^0\right)\mathbf{d}\right) \tag{31}$$

The optimality conditions for this problem are

$$\begin{aligned} \left(\frac{\partial F}{\partial \mathbf{d}}\right)^{\mathrm{T}} &= \mathbf{0} \\ \left(\frac{\partial F}{\partial \boldsymbol{\lambda}}\right)^{\mathrm{T}} &= \mathbf{0} \end{aligned} \tag{32}$$

which lead to the following linear system of equations:

$$\begin{bmatrix} \mathbf{W} & \boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\left(\mathbf{q}^0\right) \\ \boldsymbol{\Phi}_{\mathbf{q}}\left(\mathbf{q}^0\right) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{d} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ -\boldsymbol{\Phi}\left(\mathbf{q}^0\right) \end{bmatrix} \tag{33}$$

Based on Eq.(33), ADAMS computes the value of $\mathbf{d}$, and given $\mathbf{q}^0$ computes the solution of the convex optimization problem as

$$\mathbf{q} = \mathbf{q}^0 + \mathbf{d} \tag{34}$$

The configuration induced by the new set of generalized coordinates obtained as in Eq.(34) corresponds to the linearized problem of Eqs.(29) and (30). Therefore, while the solution satisfies the conditions of Eq.(30), it might be that it does not satisfy the original non-linear system constraint equations induced by the system's joints as in Eq.(26). If this is the case, then the configuration $\mathbf{q}$ just obtained is set to be the new $\mathbf{q}^0$, and another iteration, starting with the linearization of Eq.(28), is carried out.

Typically, after a couple of iterations the solution of the linear convex optimization problem will satisfy the non-linear constraint equations of the mechanical system. The approach fails when the linearization in Eq.(28) is a poor approximation of the non-linear manifold solution of the original constraint equations. However, even when the manifold is highly non-linear, the solution sequence will converge if the starting point $\mathbf{q}^0$ is close enough to the final solution. For this reason, it is essential for the algorithm to have a good starting point.

To gain an understanding of how the weights w_i help to keep the user defined initial conditions at their prescribed values, consider a case with only one constraint. Likewise, assume that the system has two generalized coordinates x and y, and that the constraint equation they must satisfy is $\Phi(x, y) = x^2 - y = 0$. Obviously this constraint equation is satisfied by an infinite number of pairs like (2,4), (2.5, 6.25), etc., but in this simple case the user would like to specify that the value of x is $x^0 = 1$, while the value y^0 is free to change. As the value for y^0 is not prescribed, assume our initial guess takes $y^0 = 6$. Since the user prescribed a value for x, the weight associated with this generalized coordinate is large; i.e., $w_1 = 10^{10}$. Since there is no condition imposed on the second generalized coordinate, $w_2 = 1$. Notice that "prescribed" in the discussion above refers to coordinates specified to be *exact* in the ADAMS modeling language (as in the ADAMS *.adm* file).

With this, the matrix of Eq.(33) assumes the form

$$\begin{bmatrix} 10^{10} & 0 & 2 \\ 0 & 1 & -1 \\ 2 & -1 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$$

Then $d_1 = 10^{-9}$, $d_2 = -5$, $\lambda = -5$, and

$$x_{IC} = 1 + 10^{-9} \approx 1$$
$$y_{IC} = 6 - 5 = 1$$

As can be easily verified, $\Phi\left(x_{IC}, y_{IC}\right) = 2 \cdot 10^{-9} + 10^{-18}$. The non-linear constraint equation is very well satisfied, and the correction applied in the user prescribed initial condition x is negligible (order 10^{-9}). In this manner, the weights w_i bias the solution toward changes in one subset of generalized coordinates rather than another.

It is worth noting that during position IC analysis ADAMS checks the compatibility of the constraint equations induced by the joints and motions present in the model. During

this constraint analysis, some of the constraint equations might turn out to be redundant. The benign case is when a redundant constraint is consistent. An example of such a case is when ADAMS is presented with one constraint equation that looks like:

$$x^2 - y = 0 \tag{35}$$

as well as a different constraint equation that reads like:

$$2x^2 - 2y = 0 \tag{36}$$

The constraint in Eq.(36) does not add anything to the picture; when the first equation is satisfied the second one is automatically satisfied too. Thus, the second equation is redundant, but consistent, and throughout the simulation ADAMS will monitor this equation to make sure that the redundant constraint continues to be consistent.

The solver will fail if the previous constraint equation is replaced by:

$$2x^2 - 2y = 1 \tag{37}$$

Equations (35) and (37) cannot be simultaneously satisfied for any $(x, y) \in \mathbf{R}^2$.

When incompatible redundant constraint equations are found, ADAMS/Solver will carry out an LU factorization with full pivoting and inform the user about encountering this situation. ADAMS will stop the simulation upon finding incompatible redundant constraints because from a modeling perspective, there is something qualitatively wrong with the system being simulated.

Redundant constraints are usually encountered when too many joints are used to model the mechanical system and the number of constraint equations generated by these joints exceeds the number of generalized coordinates of the model. In what follows, two or more constraint equations will be called independent if they are not redundant.

It is highly advised for the ADAMS modeler to eliminate redundant constraints. While sometimes the redundant constraint(s) can be benign, they can often have detrimental effects on the simulation as the reaction forces may not be calculated as intended, causing such things as poor simulation performance, unexpected eigenvectors from ADAMS/Linear, etc. More information on this can be found in Section 8.

3.2. Velocity Initial Condition Analysis

The velocity IC analysis is a direct and simple application of the algorithm employed for the position IC analysis. It is direct because it is applied exactly as presented before, and it is simple because the constraint equations that need to be satisfied are already in a linear form. Therefore, there is no need to linearize them as was the case with the

position constraint equations (see Eq.(28)) and the solution is guaranteed to be found in one iteration. Thus, the convex constrained optimization problem solved to retrieve the initial velocities $\dot{q}_i$ minimizes the cost function

$$f\left(\dot{q}_1,\ldots,\dot{q}_n\right)=\frac{1}{2}\left(\dot{\mathbf{q}}-\dot{\mathbf{q}}_0\right)^{\mathrm{T}}\mathbf{W}\left(\dot{\mathbf{q}}-\dot{\mathbf{q}}_0\right) \tag{38}$$

subject to the linear velocity kinematic constraint equations of Eq.(20):

$$\mathbf{\Phi}_{\mathbf{q}}\left(\mathbf{q},t_0\right)\cdot\dot{\mathbf{q}}+\mathbf{\Phi}_t\left(\mathbf{q},t_0\right)=\mathbf{0} \tag{39}$$

As in the displacement IC analysis, the weight diagonal matrix $\mathbf{W}$ has several very large positive entries that ensure that the user-prescribed initial velocities are not changed significantly by the optimization algorithm. From here on, the same procedure previously used for position IC analysis is employed. Note that, because of the linearity of the velocity kinematic constraint equations the algorithm is guaranteed to converge in one iteration.

An exception to the linearity of this problem is if the ADAMS user includes a GCON (generalized constraint) statement which defines a non-linear velocity constraint. In this case, obviously a non-linear solver is used to generate the consistent set of IC's.

3.3. Force and Acceleration Initial Condition Analysis

In the absence of friction forces, the acceleration IC analysis requires the solution of the linear system assembled from the equations of motion (EOM) and the acceleration kinematic constraint equations of Eq.(17). The resulting system has the form:

$$\begin{bmatrix}\mathbf{M} & \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\left(\mathbf{q}^0\right)\\ \mathbf{\Phi}_{\mathbf{q}}\left(\mathbf{q}^0\right) & \mathbf{0}\end{bmatrix}\begin{bmatrix}\ddot{\mathbf{q}}\\ \boldsymbol{\lambda}\end{bmatrix}=\begin{bmatrix}\mathbf{F}\\ \boldsymbol{\tau}\end{bmatrix} \tag{40}$$

where $\mathbf{M}$ is the generalized mass matrix. Note that this is a linear system, and the iterative process involved typically converges in one iteration. The user cannot directly prescribe any initial acceleration for the force/acceleration IC analysis. The reaction force and initial accelerations are evaluated based on the computed initial position, initial velocity, and the applied force acting on the system at the initial time. For a more detailed explanation of how the EOM are obtained (the first row in the Eq.(40)), see the Section on Dynamic Analysis in ADAMS.

Besides $\ddot{\mathbf{q}}$, the solution of the linear system above also provides the Lagrange multipliers $\boldsymbol{\lambda}$. The constraint force and torque induced by joint j on body i are computed as:

$$\mathbf{F}^{\mathrm{C}}=-\left(\frac{\partial\dot{\mathbf{\Phi}}^{(j)}}{\mathbf{v}_i}\right)^{\mathrm{T}}\boldsymbol{\lambda}^{(j)} \tag{41}$$

$$\mathbf{T}^{\mathrm{C}} = -\left(\frac{\partial \dot{\mathbf{\Phi}}^{(j)}}{\boldsymbol{\omega}_i} \right)^{\mathrm{T}} \boldsymbol{\lambda}^{(j)} \tag{42}$$

In Eqs.(41) and (42) the superscript C indicates that the quantities are expressed in a Cartesian coordinate system; $\mathbf{v}_i$ is the Cartesian velocity of body i; $\boldsymbol{\omega}_i$ is the global angular velocity; $\mathbf{\Phi}^{(j)}$ represents the set of constraint equations associated with joint j. An explanation of why the reaction forces expressed in the global reference frame are computed as indicated in Eqs.(41) and (42) is beyond the scope of this discussion, but it suffices to say that they are obtained by projecting the Lagrange multipliers $\boldsymbol{\lambda}^{(j)}$ along the Cartesian translational and rotational velocities $\mathbf{v}_i$, and $\boldsymbol{\omega}_i$ of body i, respectively.

4. Kinematic Analysis

Typically, for a kinematic analysis to be carried out a number of independent constraint equations equal to the number of generalized coordinates in the model must be prescribed. For the mechanism to actually change its configuration in time, some of these constraints must be motions; i.e., they depend on time.

4.1. Position-Level Kinematic Analysis

Given the position of the system at time t_0, the problem here is to determine the position at time $t_1 > t_0$. Because of the non-linear nature of the constraint equations in Eq.(19), a Newton-Raphson iterative method is used in ADAMS to compute $\mathbf{q}_1$ at time t_1. To understand how this method works and what its limitations are first note that it is obtained from a Taylor-expansion-based linearization of the non-linear constraint equations:

$$\mathbf{\Phi}(\mathbf{q}_1, t_1) = \mathbf{\Phi}(\mathbf{q}_0, t_1) + \mathbf{\Phi}_\mathbf{q}(\mathbf{q}_0, t_1)(\mathbf{q}_1 - \mathbf{q}_0) \tag{43}$$

Since the number of constraints is equal to the number of generalized coordinates, the matrix $\mathbf{\Phi}_\mathbf{q}(\mathbf{q}_0, t_1)$ is square. As the constraint equations were assumed to be independent, this matrix is also invertible. Based on an explicit integrator (e.g., Forward-Euler) an initial starting configuration $\mathbf{q}_1^{(0)}$ is determined, and the iterative algorithm proceeds at each iteration $j \geq 0$ by finding the correction $\mathbf{\Delta}^{(j)}$

$$\mathbf{\Phi}_\mathbf{q}(\mathbf{q}_0, t_1)\mathbf{\Delta}^{(j)} = -\mathbf{\Phi}\left(\mathbf{q}_1^{(j)}, t_1\right) \tag{44}$$

Then, $\mathbf{q}_1^{(j+1)} = \mathbf{q}_1^{(j)} + \mathbf{\Delta}^{(j)}$, and the iterative process is repeated until the correction $\mathbf{\Delta}^{(j)}$ and/or the residual $\mathbf{\Phi}\left(\mathbf{q}_1^{(j)}, t_1\right)$ become small enough.

As with position IC analysis, ADAMS can fail to find $\mathbf{q}_1$ if the linearization at the initial guess turns out to be a poor approximation of the non-linear manifold. In these situations, the remedy lies in decreasing the simulation step-size, causing $\mathbf{q}_1$ to lie closer on the manifold to the last consistent configuration, $\mathbf{q}_0$.

4.2. Velocity-Level Kinematic Analysis

Velocity kinematic analysis is straightforward, as the velocity kinematic constraint equations are linear in velocity. With $\mathbf{q}_1$ already available from the position kinematic

analysis, the non-singular matrix $\boldsymbol{\Phi}_{\mathbf{q}}\left(\mathbf{q}_1, t_1\right)$ is evaluated and the linear system of Eq.(20) is solved for the new velocity $\mathbf{q}_1$.

4.3. Acceleration Level Kinematic Analysis

Acceleration kinematic analysis is immediate, as at time t_1 it is found as the solution of the linear system of Eq.(21). Notice that the same matrix that is factored for velocity kinematic analysis is used for a forward/backward substitution sequence to solve for the generalized accelerations $\ddot{\mathbf{q}}$.

Once $\ddot{\mathbf{q}}$ is available, the Lagrange multipliers associated with the set of constraints acting on the system are computed as the solution of the linear system

$$\boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{F} - \mathbf{M}\ddot{\mathbf{q}} \tag{45}$$

This equation is identical to the first row of the linear system of Eq.(40) and in fact represents precisely the equations of motion. More information on how Eq.(45) is obtained is provided in the next Section.

5. Dynamic Analysis

5.1. Nomenclature, Conventions, Definitions.

In addition to the definitions and notations introduced at the beginning of this document, the following quantities will be used in formulating the rigid body equations of motion.

$\mathbf{M}$ - generalized mass matrix

$\bar{\mathbf{J}}$ - generalized inertia matrix expressed about the principal local reference frame

K - kinetic energy, defined as

$$K = \frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathbf{M}\mathbf{u} + \frac{1}{2}\bar{\boldsymbol{\omega}}^{\mathrm{T}}\bar{\mathbf{J}}\bar{\boldsymbol{\omega}} \tag{46}$$

$\boldsymbol{\lambda} \in \mathbf{R}^m$ - array of Lagrange multipliers. The number m of Lagrange multipliers is given by the number of constraint equations induced by joints connecting a body to other bodies in the system.

$\mathbb{F}(\mathbf{q}, \dot{\mathbf{q}}, t) = \begin{bmatrix} \mathbf{f} \\ \bar{\mathbf{n}} \end{bmatrix} \in \mathbf{R}^6$ - the vector of applied forces; $\mathbf{f}$ is the vector of applied forces expressed in the global reference frame, while $\bar{\mathbf{n}}$ represent the applied torque expressed in the local Cartesian reference frame

$\mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}}, t) \in \mathbf{R}^6$ - the generalized force acting on the body. Obtained by projecting the applied force $\mathbb{F}$ upon the generalized coordinates. Typically,

$$\mathbf{Q} = \begin{bmatrix} \left(\boldsymbol{\Pi}^{\mathrm{P}}\right)^{\mathrm{T}} \mathbf{f} \\ \left(\boldsymbol{\Pi}^{\mathrm{R}}\right)^{\mathrm{T}} \bar{\mathbf{n}} \end{bmatrix} \tag{47}$$

where with $\mathbf{v}^{\mathbf{P}}$ being the velocity of the point of application $\mathbf{P}$ of the external force $\mathbb{F}$, the projection operators are computed like

$$\boldsymbol{\Pi}^{\mathbf{P}} = \frac{\partial \mathbf{v}^{\mathbf{P}}}{\partial \mathbf{u}} \tag{48}$$

$$\boldsymbol{\Pi}^{\mathbf{R}} = \frac{\partial \bar{\boldsymbol{\omega}}}{\partial \boldsymbol{\zeta}} \tag{49}$$

5.2. Formulation of Equation Of Motion in ADAMS

The Lagrange formulation of the equations of motion leads to the following second order differential equations

$$\frac{d}{dt}\left[\left(\frac{\partial K}{\partial \dot{\mathbf{q}}}\right)^{\mathrm{T}}\right]-\left(\frac{\partial K}{\partial \mathbf{q}}\right)^{\mathrm{T}}+\boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\lambda}=\mathbf{Q} \tag{50}$$

Considering the choice of generalized coordinates in ADAMS; i.e., the definition of $\mathbf{q}$ as in Eq.(8), Eq.(50) is rewritten for a rigid body as

$$\frac{d}{dt}\begin{bmatrix}\left(\frac{\partial K}{\partial \mathbf{u}}\right)^{\mathrm{T}}\\ \left(\frac{\partial K}{\partial \boldsymbol{\zeta}}\right)^{\mathrm{T}}\end{bmatrix}-\begin{bmatrix}\left(\frac{\partial K}{\partial \mathbf{p}}\right)^{\mathrm{T}}\\ \left(\frac{\partial K}{\partial \boldsymbol{\varepsilon}}\right)^{\mathrm{T}}\end{bmatrix}+\begin{bmatrix}\boldsymbol{\Phi}_{\mathbf{p}}^{\mathrm{T}}\boldsymbol{\lambda}\\ \boldsymbol{\Phi}_{\boldsymbol{\varepsilon}}^{\mathrm{T}}\boldsymbol{\lambda}\end{bmatrix}=\begin{bmatrix}\left(\boldsymbol{\Pi}^{\mathbf{P}}\right)^{\mathrm{T}}\mathbf{f}\\ \left(\boldsymbol{\Pi}^{\mathbf{R}}\right)^{\mathrm{T}}\bar{\mathbf{n}}\end{bmatrix} \tag{51}$$

It is worth pointing out that when dealing with a full system of rigid bodies connected through joints, the system Equation Of Motions (EOM) are obtained by simply stacking together the EOM for the bodies in the system.
Since

$$\frac{d}{dt}\left(\frac{\partial K}{\partial \mathbf{u}}\right)^{\mathrm{T}}=\mathbf{M}\dot{\mathbf{u}} \tag{52}$$

$$\left(\frac{\partial K}{\partial \mathbf{p}}\right)^{\mathrm{T}}=\mathbf{0} \tag{53}$$

with the angular momenta defined as

$$\boldsymbol{\Gamma}\equiv\frac{\partial K}{\partial \boldsymbol{\zeta}}=\mathbf{B}^{\mathrm{T}}\bar{\mathbf{J}}\mathbf{B}\boldsymbol{\zeta} \tag{54}$$

the EOM of Eq.(51) are reformulated in ADAMS as

$$\begin{aligned}\mathbf{M}\dot{\mathbf{u}}+\boldsymbol{\Phi}_{\mathbf{p}}^{\mathrm{T}}\boldsymbol{\lambda}&=\left(\boldsymbol{\Pi}^{\mathbf{P}}\right)^{\mathrm{T}}\mathbf{f}\\ \dot{\boldsymbol{\Gamma}}-\frac{\partial K}{\partial \boldsymbol{\varepsilon}}+\boldsymbol{\Phi}_{\boldsymbol{\varepsilon}}^{\mathrm{T}}\boldsymbol{\lambda}&=\left(\boldsymbol{\Pi}^{\mathbf{R}}\right)^{\mathrm{T}}\bar{\mathbf{n}}\end{aligned} \tag{55}$$

The first order differential equations above are called in what follows kinetic differential equations, and they indicate how external forces determine the time variation of the translational and angular momenta.
Finally, the time variation of the generalized coordinates is related to the translational and angular momenta by means of the kinematic differential equations. By assembling the kinetic and kinematic differential equations ADAMS generates a set of 15 equations for

each rigid body that provide the information necessary to find a numerical solution for the dynamic analysis of a mechanical system. These equations are as follows:

$$\mathbf{M}\dot{\mathbf{u}} + \boldsymbol{\Phi}_{\mathbf{p}}^{\mathrm{T}}\boldsymbol{\lambda} - \left(\boldsymbol{\Pi}^{\mathbf{P}}\right)^{\mathrm{T}}\mathbf{f} = \mathbf{0} \tag{56}$$

$$\boldsymbol{\Gamma} - \mathbf{B}^{\mathrm{T}}\bar{\mathbf{J}}\mathbf{B}\boldsymbol{\zeta} = \mathbf{0} \tag{57}$$

$$\dot{\boldsymbol{\Gamma}} - \frac{\partial K}{\partial \boldsymbol{\varepsilon}} + \boldsymbol{\Phi}_{\boldsymbol{\varepsilon}}^{\mathrm{T}}\boldsymbol{\lambda} - \left(\boldsymbol{\Pi}^{\mathbf{R}}\right)^{\mathrm{T}}\bar{\mathbf{n}} = \mathbf{0} \tag{58}$$

$$\dot{\mathbf{p}} - \mathbf{u} = \mathbf{0} \tag{59}$$

$$\dot{\boldsymbol{\varepsilon}} - \boldsymbol{\zeta} = \mathbf{0} \tag{60}$$

5.3. Numerical Solution for Dynamic Analysis. Jacobian Computation.

Equations (56) through (60) indicate how the generalized coordinates, reaction forces, and applied forces variables change in time. What is missing in this picture is the fact that the solution of this system of differential equations must also satisfy the kinematic constraint equations of Eqs.(19) through (21). From a numerical standpoint, this is what makes the dynamic analysis of a mechanical system challenging.

There is a multitude of methods for solving the assembly of differential + constraint equations. This document is not concerned with these methods and it only provides a glimpse at what ADAMS does to address this problem. In this context it is worth mentioning that this assembly of differential and constraint equations forms what is called a set of Differential-Algebraic Equations (DAE). A DAE has an index associated with it (index defined as the number of times the DAE's must be differentiated to get the system into ODE's), and rule goes that the higher the index, the more challenging the numerical solution of the DAE becomes. In particular, the DAE induced by the dynamic analysis problem in mechanical system simulation has index 3, which is considered high. In the ADAMS solver there are two more reliable methods for solution. The most common one is a direct index 3 DAE solver, in which associated to the differential equations induced by (56) through (60) are the position kinematic constraint equations of Eq.(19). This is how the solver GSTIFF-I3 works in ADAMS.

A second, more refined algorithm reduces the original index 3 problem to an analytically equivalent yet numerically different index 2 DAE problem. Thus, instead of considering the position, the velocity level kinematic constraint equations of Eq.(20) are solved for along with the kinematic differential equations. In the ADAMS solver, this algorithm is called SI2, and while typically slower than the index 3 approach it turns out to be more accurate and robust.

In what follows the index 3 approach is presented in a reasonable amount of detail. In an attempt to keep the presentation simple, the index 3 DAE will be integrated via an order 1 implicit integration formula, which converts the DAE's into a set of algebraic equations. This formula is the backward Euler formula - a one-step, A-stable algorithm that qualitatively captures all the relevant details characteristic to higher order methods.

Backward Euler integration formula replaces the derivative $\dot{\mathbf{y}}_1$ at time t_1 with

$$\dot{\mathbf{y}}_1 = \frac{1}{h}\mathbf{y}_1 - \frac{1}{h}\mathbf{y}_0 \tag{61}$$

Based on Eq.(61), an Initial Value Problem (IVP) $\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}, t)$, $\mathbf{y}(t_0) = \mathbf{y}_0$ is solved by finding $\mathbf{y}(t_1)$ at time $t_1 > t_0$ as the solution $\mathbf{y}_1$ of the discretization algebraic non-linear system

$$\frac{1}{h}\mathbf{y}_1 - \frac{1}{h}\mathbf{y}_0 - \mathbf{g}(t_1, \mathbf{y}_1) = \mathbf{0} \tag{62}$$

The system of equations in Eq.(62) is called a "discretization system" since the derivative in the original IVP problem was "discretized" using the integration formula of Eq.(61). Since almost always the function $\mathbf{g}$ is non-linear, a non-linear algebraic system needs to be solved to retrieve $\mathbf{y}_1$. This is done in ADAMS by using a Newton-Raphson type iterative algorithm.

Based on the implicit Euler discretization formula introduced above, all the first order time derivatives that appear in the equations of motion in Eqs.(56) through (60) are discretized to produce a set of algebraic non-linear equations. In the index 3 approach, the position kinematic constraint equations are appended to these equations, along with the force function definition $\mathbf{F}$ and $\mathbf{T}$. This appending of the force functions is done for the sole purpose of increasing the number of unknowns and thus inducing a larger but yet sparser Jacobian matrix. Thus, after the implicit Euler based discretization Eqs.(19), and (56) through (60) along with the force/torque definition equations assume the following form:

$$
\begin{aligned}
\frac{1}{h}\mathbf{M}\mathbf{u}-\frac{1}{h}\mathbf{M}\mathbf{u}_0+\boldsymbol{\Phi}_{\mathbf{p}}^{\mathrm{T}}\boldsymbol{\lambda}-\left(\Pi^{\mathbf{P}}\right)^{\mathrm{T}}\mathbf{f}&=\mathbf{0}\\
\boldsymbol{\Gamma}-\mathbf{B}^{\mathrm{T}}\bar{\mathbf{J}}\mathbf{B}\boldsymbol{\zeta}&=\mathbf{0}\\
\frac{1}{h}\boldsymbol{\Gamma}-\frac{1}{h}\boldsymbol{\Gamma}_0-\left(\frac{\partial K}{\partial\boldsymbol{\varepsilon}}\right)^{\mathrm{T}}+\boldsymbol{\Phi}_{\boldsymbol{\varepsilon}}^{\mathrm{T}}\boldsymbol{\lambda}-\left(\Pi^{\mathbf{R}}\right)^{\mathrm{T}}\bar{\mathbf{n}}&=\mathbf{0}\\
\frac{1}{h}\mathbf{p}-\frac{\mathbf{1}}{h}\mathbf{p}_0-\mathbf{u}&=\mathbf{0}\\
\frac{1}{h}\boldsymbol{\varepsilon}-\frac{\mathbf{1}}{h}\boldsymbol{\varepsilon}_0-\boldsymbol{\zeta}&=\mathbf{0}\\
\boldsymbol{\Phi}\left(\mathbf{p},\boldsymbol{\varepsilon},t_1\right)&=\mathbf{0}\\
\mathbf{f}-\mathbf{F}\left(\mathbf{u},\boldsymbol{\zeta},\mathbf{p},\boldsymbol{\varepsilon},\mathbf{f},\bar{\mathbf{n}},t_1\right)&=\mathbf{0}\\
\bar{\mathbf{n}}-\mathbf{T}\left(\mathbf{u},\boldsymbol{\zeta},\mathbf{p},\boldsymbol{\varepsilon},\mathbf{f},\bar{\mathbf{n}},t_1\right)&=\mathbf{0}
\end{aligned}
\tag{63}
$$

The unknowns in this non-linear system are $\mathbf{u}$, $\boldsymbol{\Gamma}$, $\boldsymbol{\zeta}$, $\mathbf{p}$, $\boldsymbol{\varepsilon}$, $\boldsymbol{\lambda}$, $\mathbf{f}$, $\bar{\mathbf{n}}$. The subscript 1 indicating the time step was dropped for convenience.

Introducing the array

$$
\mathbf{y}=\begin{bmatrix}\mathbf{u}\\ \boldsymbol{\Gamma}\\ \boldsymbol{\zeta}\\ \mathbf{p}\\ \boldsymbol{\varepsilon}\\ \boldsymbol{\lambda}\\ \mathbf{f}\\ \bar{\mathbf{n}}\end{bmatrix}
\tag{64}
$$

the non-linear system of Eq.(63) is rewritten as

$$
\boldsymbol{\Psi}\left(\mathbf{y}\right)=\mathbf{0}
\tag{65}
$$

A Newton-Raphson type algorithm finds the solution of this system. First a prediction $\mathbf{y}^{(0)}$ of the solution is computed, typically by using a predictor constructed around an explicit integrator. Once an initial guess of the solution is provided, iterations

$$
\begin{aligned}
\boldsymbol{\Psi}_{\mathrm{y}}\left(\mathbf{y}_0\right)\boldsymbol{\Delta}^{(j)}&=-\boldsymbol{\Psi}\left(\mathbf{y}^{(j)}\right)\\
\mathbf{y}^{(j+1)}&=\mathbf{y}^{(j)}+\boldsymbol{\Delta}^{(j)}
\end{aligned}
\tag{66}
$$

are carried out until the correction $\boldsymbol{\Delta}^{(j)}$ are small enough (note that the residual $\boldsymbol{\Psi}\left(\mathbf{y}^{(j)}\right)$ *could* potentially be used as an exit criteria, but is not).

The Newton-Raphson method requires the computation of the Jacobian $\mathbf{\Psi}_{\mathbf{y}}(\mathbf{y}_0)$, which is obtained from Eq.(63). With the notation introduced in Eq.(64), the expression of the Jacobian $\mathbf{\Psi}_{\mathbf{y}}(\mathbf{y}_0)$ is provided in the Appendix.

The same remarks made in conjunction with the iterative Newton-Raphson algorithm used for IC analysis and for Kinematic analysis are applicable here. Thus, if the initial guess; i.e., the predicted value of $\mathbf{y}^{(0)}$ is too far away from the solution, the iterative process might fail to converge. This is more likely to happen with dynamic analysis than with other types of analysis, as it is clear that the system that needs to be solved at each integration step is highly non-linear.

If the iterative process fails, the integration step-size is decreased and another step is attempted. ADAMS users are familiar in this context with messages informing them that the step-size was decreased too much, and yet the convergence was not attained. Getting such a message is a bad omen, as typically the user will have to revisit the model, make modifications in the simulation defining parameters, or to try a different integrator like SI2, for example. More details on this can be found in Section 8.

Finally, although the discretization formula used to convey the dynamic analysis solution message was backward Euler it conceptually captures the essence of the ADAMS solution sequence. ADAMS typically uses higher order integrators whenever the signals sent over by the problem being solved suggest that this would improve performance. The expression of the integration Jacobian is qualitatively the same, with very minor and insignificant changes – for example the denominator of the fraction $1/h$ would become $1/(h\beta)$, where β is an integration formula specific coefficient. What is important to remember here is that the use of a more sophisticated integration formula serves in the end the same purpose, namely to replace a first order time derivative with a linear combination of future and past values of the unknown, which is what backward Euler formula does in a very basic way through Eq.(62).

6. Statics Analysis

The ADAMS/SolverF77 supports two methods for finding the equilibrium configuration of a mechanical system: the STATIC approach, and the DYNAMIC approach (see the ADAMS/Solver manual, the EQUILIBRIUM statement/command discussion for more details). The ADAMS/SolverC++ currently only supports the STATIC approach.

6.1. The STATIC approach

From an algorithmic perspective the STATIC approach is based on transforming the equilibrium problem into an equivalent problem that requires the solution of a non-linear system of algebraic equations. As such, this method employs a Newton-Raphson algorithm to find the solution of the non-linear system of equations.

The key observation in the STATIC approach is that at equilibrium, while all the typical equations (equations of motion, constraint equations, DIFF equations, force definition equations, etc.) must be satisfied, the time derivative of any quantity that appears in these equations should be zero (at equilibrium, there is no change in time in the value of a variable, and therefore its derivative should be zero). Based on this observation, considering Eqs.(56) through (60) and setting all the derivatives to zero the following set of *algebraic* equations are obtained:

$$
\begin{aligned}
\boldsymbol{\Phi}_{\mathbf{p}}^{\mathrm{T}}\boldsymbol{\lambda}-\left(\mathbf{A}^{\mathbf{P}}\right)^{\mathrm{T}}\mathbf{f}&=\mathbf{0}\\
\boldsymbol{\Gamma}-\mathbf{B}^{\mathrm{T}}\bar{\mathbf{J}}\mathbf{B}\boldsymbol{\zeta}&=\mathbf{0}\\
-\left(\frac{\partial\mathbf{K}}{\partial\boldsymbol{\varepsilon}}\right)^{\mathrm{T}}+\boldsymbol{\Phi}_{\boldsymbol{\varepsilon}}^{\mathrm{T}}\boldsymbol{\lambda}-\left(\mathbf{A}^{\mathbf{R}}\right)^{\mathrm{T}}\bar{\mathbf{n}}&=\mathbf{0}\\
\mathbf{u}&=\mathbf{0}\\
\boldsymbol{\zeta}&=\mathbf{0}\\
\boldsymbol{\Phi}\left(\mathbf{p},\boldsymbol{\varepsilon},t_0\right)&=\mathbf{0}\\
\mathbf{f}-\mathbf{F}\left(\mathbf{u},\boldsymbol{\zeta},\mathbf{p},\boldsymbol{\varepsilon},\mathbf{f},\bar{\mathbf{n}},\mathbf{x},t_0\right)&=\mathbf{0}\\
\bar{\mathbf{n}}-\mathbf{T}\left(\mathbf{u},\boldsymbol{\zeta},\mathbf{p},\boldsymbol{\varepsilon},\mathbf{f},\bar{\mathbf{n}},\mathbf{x},t_0\right)&=\mathbf{0}\\
\mathbf{d}(\mathbf{u},\boldsymbol{\zeta},\mathbf{p},\boldsymbol{\varepsilon},\mathbf{f},\bar{\mathbf{n}},\mathbf{x},t_0)&=\mathbf{0}
\end{aligned}
\qquad (67)
$$

Note that unlike Eqs. (56) through (60), this set of non-linear equations was modified to include the ADAMS DIFF elements, which for the purpose of this discussion were assumed to have the form

$$\dot{\mathbf{x}}-\mathbf{d}(\mathbf{u},\boldsymbol{\zeta},\mathbf{p},\boldsymbol{\varepsilon},\mathbf{f},\bar{\mathbf{n}},\mathbf{x},t)=\mathbf{0}$$

Here $\mathbf{x}$ is the state associated with the model DIFFs, and for simplicity the DIFFs were assumed in explicit form. Setting $\dot{\mathbf{x}}=\mathbf{0}$ in the definition of the DIFFs leads to the last equation in Eq.(67). It is important to underline that the DIFFs should have been present

in the dynamic analysis discussion in section 5.2 as well, and they were omitted there only to keep the presentation simple.

In the STATIC approach, an equilibrium configuration

$$\mathbf{y}^T = [\mathbf{u} \quad \boldsymbol{\Gamma} \quad \boldsymbol{\zeta} \quad \mathbf{p} \quad \boldsymbol{\varepsilon} \quad \boldsymbol{\lambda} \quad \mathbf{f} \quad \bar{\mathbf{n}} \quad \mathbf{x}]$$

is found as the solution of the algebraic non-linear system of Eq.(67), and to this end the Newton-Raphson algorithm of section 1.1 is used. Recall that for this algorithm to work, the initial starting point of the iteration sequence should be close enough to the solution. This is precisely what makes an equilibrium analysis challenging. Unless the user ensures that the initial mechanical system configuration is close to equilibrium, the algorithm might fail.

NOTE: The numerical solution of the dynamic analysis discussed in detail in section 5.3 also requires the solution of a similar non-linear system during the integration corrector stage. The situation is more tractable in the dynamic analysis case due to the predictor employed by the integrator. It is one of the roles of the predictor to produce a good initial starting point for the Newton-Raphson algorithm. The convenience of a predictor is not available in the STATIC approach, and therefore the user has to either (a) provide a good starting point by setting up the model to be in a configuration close to equilibrium, or (b) change the EQUILIBRIUM parameters to control the convergence of the Newton-Raphson method (see sections 8.2 and 8.3 for a discussion in this sense)

6.2. The DYNAMIC Approach

The DYNAMIC approach is only available in ADAMS/SolverF77, and it is used less frequently. The idea is that rather than solving a non-linear system to find the equilibrium configuration (as in the STATIC approach), the Solver falls back on the integrator to find the equilibrium configuration. ”Virtual” damping is artificially added to the system, and the assumption is that in a finite amount of time the system will settle in an equilibrium configuration due to energy dissipation resulting from the “virtual” and numerical damping associated with the approach. The reader is referred to section 5 and the ADAMS/Solver Manual for details about how a dynamic analysis is carried out.

The DYNAMIC approach in ADAMS/SolverF77 is not the first choice for finding an equilibrium configuration. However, the user is encouraged to fall back on it when the STATIC, which is the preferred approach, fails to find an equilibrium configuration.

6.3. The *STATIC_HOLD* Attribute

STATIC_HOLD is exclusively an attribute of the DIFF statement/command, and it is irrelevant in any analysis mode except statics. Consider for example the definition of the following ADAMS DIFF:

```
DIFF/10, IC=2.0, STATIC_HOLD, FUNCTION=DIFF(10)-DX(23,11)
```

In this example DX represents the distance along the X-axis between markers 23 and 11 (assumed to be defined somewhere else in the model). If the STATIC_HOLD is not present in the above definition, during an equilibrium analysis the Solver will adjust the value of DIFF(10), that is, the state associated with the DIFF such that the expression DIFF(10)-DX(23,11) will become zero. This is because at equilibrium, the time derivative of any quantity should be zero, and therefore DIFF1(10), which according to the DIFF definition is equal to DIFF(10)-DX(23,11), should be zero. Thus, the value IC=2.0 will be overridden by the solver, and the actual value of the state associated with the DIFF after the statics analysis; i.e., the true IC value will be DX(23,11). Sometimes this is what the user wants, but sometimes keeping the value of IC to what was originally set is preferable. This is precisely what STATIC_HOLD does. Under these circumstances, for all purposes the equilibrium analysis proceeds as though the DIFF element is not present in the model. Furthermore, whenever the quantity DIFF(10) is referenced during the static analysis by another modeling element, for instance a force definition, the value returned would be that indicated by the IC (2.0 in our example), while the quantity DIFF1(10) will be zero. The key observation when STATIC_HOLD is present is that even if the Solver indicates that the static analysis converged, *the residual in satisfying the DIFF equation is not zero*. In our case, the residual would be 2.0-DX(23,11), which would be zero only by chance.

Finally, if the STATIC_HOLD attribute is not present in the definition of a DIFF, when the static analysis will have converged the value of the state associated with the DIFF would not be 2.0, but precisely the value DX(23,11). The function DIFF1(10) would evaluate to 0.0, and there would be a small residual in satisfying the DIFF, but it would be at the most as large as the one specified in the IMBALANCE setting associated with the EQUILIBRIUM statement/command.

7. A Simple Pendulum Example

7.1. The Simple Pendulum Model

A simple, two-dimensional pendulum model is chosen as an example, since the ideas for the two-dimensional model are representative of the three-dimensional space, while keeping the equations as simple as possible.

The model is contains one rigid part, the body of the pendulum, and ground. A pin, or revolute, joint connects the pendulum body to ground.

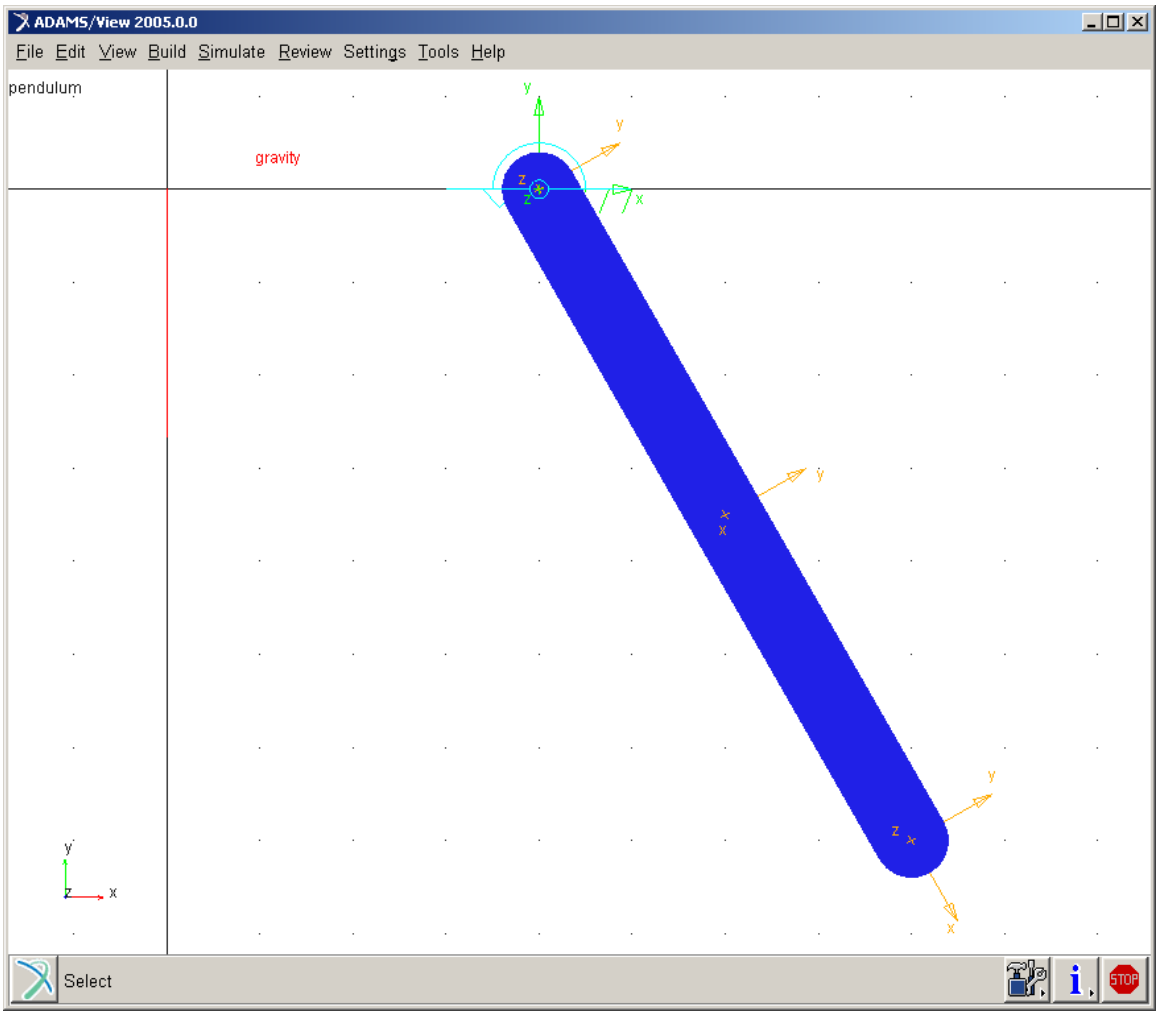


Figure 3

7.2. Initial Condition Analysis

Recall that for an Initial Condition analysis, we are assembling the parts and constraints such that all of the constraints are satisfied for both positions and velocities, as well as determining the initial accelerations and reaction forces in the model. In this example, we will look at the constrained optimization problem for solving the position initial conditions. First, we need to develop the constraint equations so that we can determine a consistent set of generalized coordinates.

For our two-dimensional pendulum, we currently have only a revolute joint to provide the constraint equations. The revolute joint attaches the body of the pendulum to ground, and ensures that:

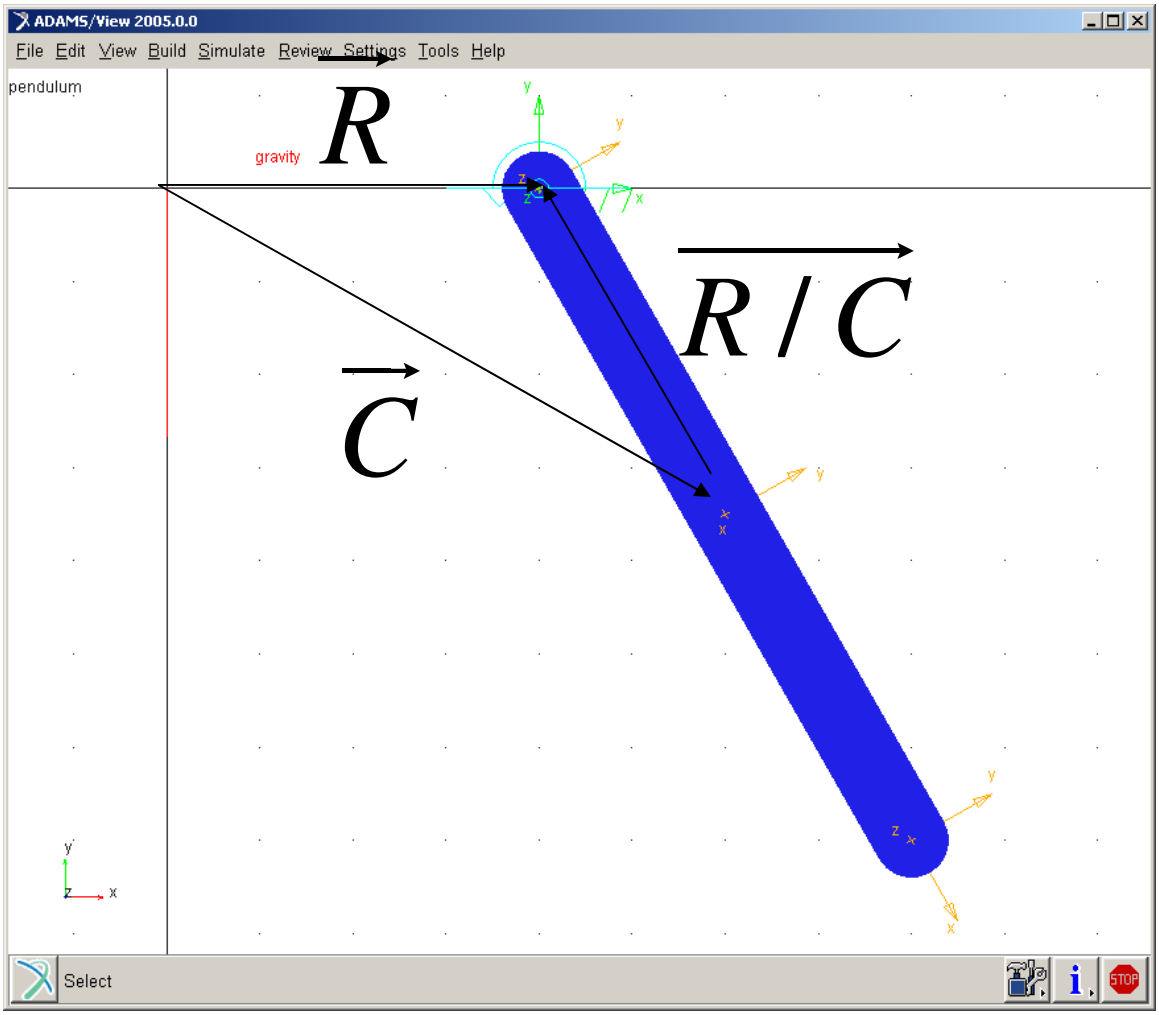


Figure 4

$$\vec{R} = \vec{C} + \overrightarrow{R/C}$$

…or, equivalently:

$$\vec{C} + \overrightarrow{R/C} - \vec{R} = 0$$

Where:

- The constant *R* is a vector which locates the pin (revolute joint) in the ground reference frame
- The variable *C* is a vector which locates the center of mass of the pendulum in the ground reference frame
- The variable *R/C* is the vector from the center-of-mass of the pendulum to the pin (revolute joint) attaching it to ground.

The condition imposed by the mechanism can be rewritten in vector form as the constraint equation:

$$\Phi(x, y, \theta) = (x + R/C_1 - R_1)\hat{i} + (y + R/C_2 - R_2)\hat{j} = 0$$

$$R/C_1 = -l\cos\theta$$
$$R/C_2 = -l\sin\theta$$

…where $\hat{i}$ and $\hat{j}$ are unit vectors along the x and y coordinate axes in the global (ground) reference frame. We can, alternatively, express this vector constraint in matrix notation:

$$\Phi(x, y, \theta) = \begin{pmatrix} x + R/C_1 - R_1 \\ y + R/C_2 - R_2 \end{pmatrix} = 0$$

The Jacobian matrix for Φ is the 2 by 3 array:

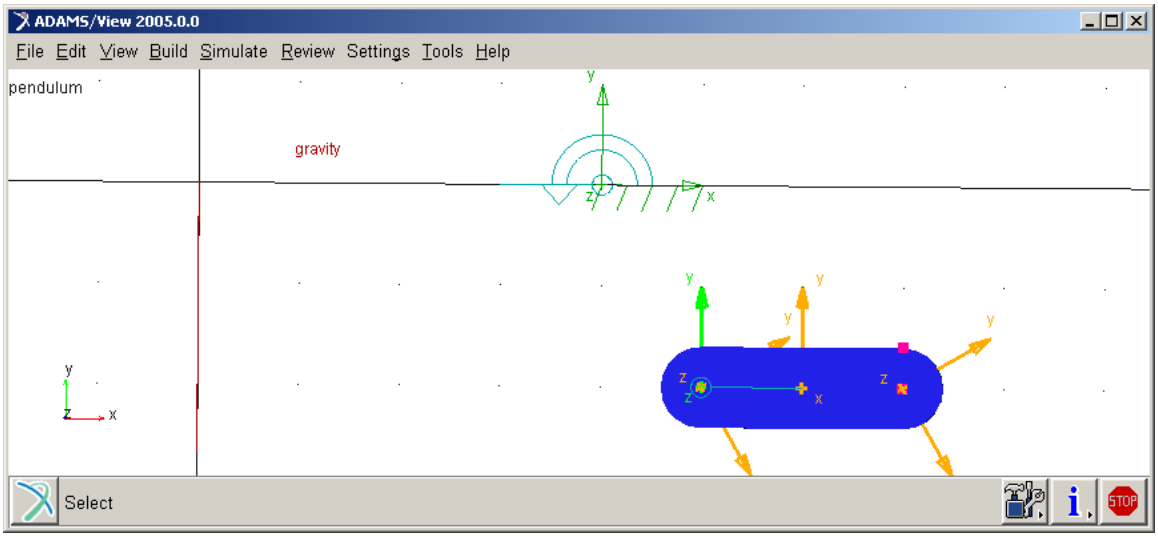
$$\Phi_q = \begin{pmatrix} 1 & 0 & l\sin\theta \\ 0 & 1 & -l\cos\theta \end{pmatrix} = 0$$

Let's now formulate the constrained optimization problem for our constraint equations:

$$\begin{bmatrix} w_1 & 0 & 0 & 1 & 0 \\ 0 & w_2 & 0 & 0 & 1 \\ 0 & 0 & w_3 & l\sin\theta^0 & -l\cos\theta^0 \\ 1 & 0 & l\sin\theta^0 & 0 & 0 \\ 0 & 1 & -l\cos\theta^0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ x^0 - l\cos\theta^0 - R_1 \\ y^0 - l\sin\theta^0 - R_2 \end{bmatrix}$$

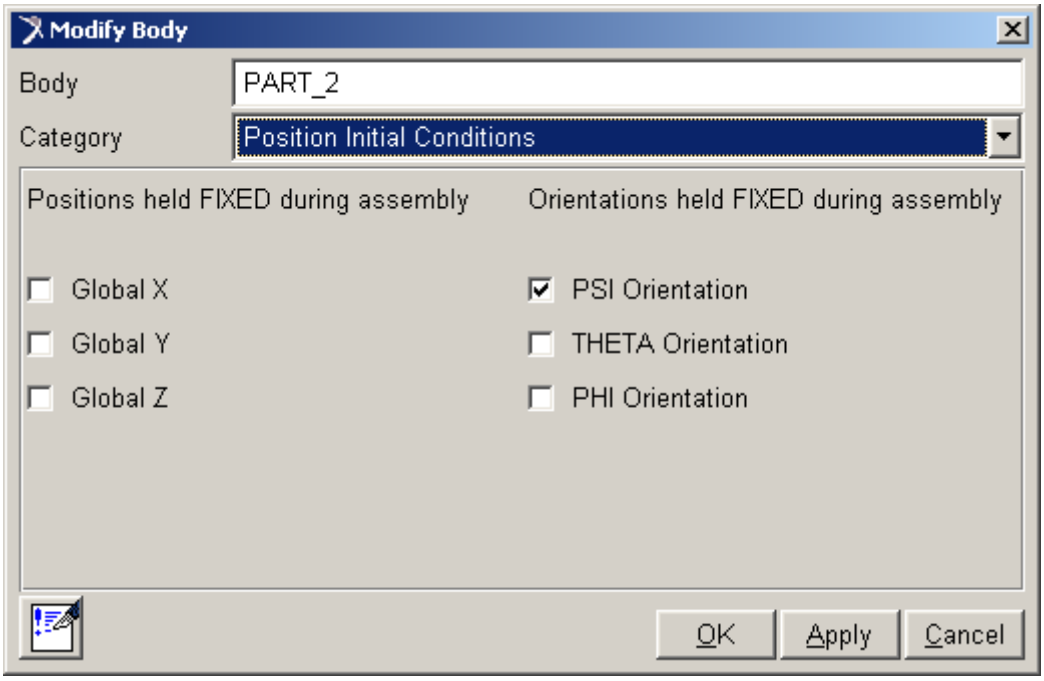
For example, let's say we have a broken joint, and the pendulum is horizontal, length *2l=2*, and we have decided to fix the orientation such that:

$$\theta^0 = 0^\circ\,(horizontal)$$
$$w_3 = 10^{10}\,(fixed)$$
$$\left.\begin{aligned} R_1 &= 4 \\ x^0 &= 6 \end{aligned}\right\} = broken$$
$$\left.\begin{aligned} R_2 &= 0 \\ y^0 &= -2 \end{aligned}\right\} = broken$$
$$w_1 = w_2 = 1(free)$$
$$l = 1$$



In other words, our pendulum is horizontal, and the pendulum center of mass position *(x,y)* is at (6,-2), while the revolute joint location is at (4,0). This puts the end of the pendulum of length *2l* at (5,-2), which is not coincident with the revolute location, as required to make the constraint consistent.

Notice that we have put a large weight for the orientation of the pendulum, which is achieved by "fixing" orientation like so:



(Note that because we are building a two-dimensional problem in ADAMS/View, the THETA above is not the same as the one in our two-dimensional model).

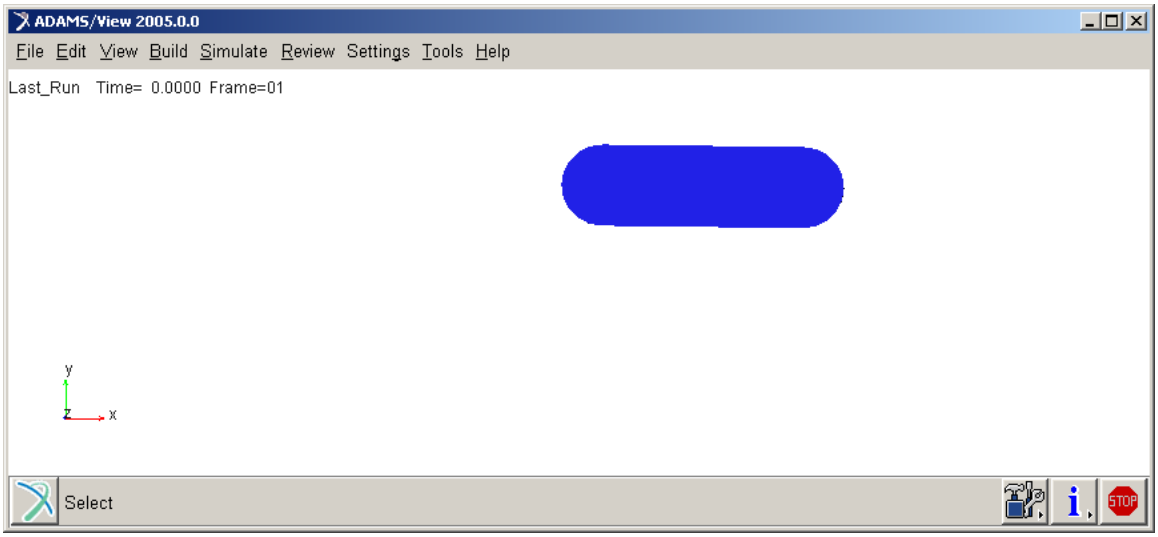
Substituting the initial configuration into our optimization problem produces:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 10^{10} & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 6-1-4=1 \\ -2-0-0=-2 \end{bmatrix}$$

By inspection, you can see that:

$$d_1 = -\lambda_1 = -1$$
$$d_2 = -\lambda_2 = 2$$
$$d_3 = \frac{\lambda_2}{10^{10}} = -2^{-10}$$
$$x^1 = x^0 + d_1 = 6 + (-1) = 5$$
$$y^1 = y^0 + d_2 = -2 + 2 = 0$$
$$\theta^1 = \theta^0 + d_3 = 0 + \left(-2^{-10}\right) \approx 0^\circ$$

This puts our pendulum center of mass position *(x,y)*, at (5,0) at 0 degrees orientation (horizontal). And, since the half-length of the pendulum (*l*) is one, this is the correct location given our revolute joint is at (4,0).



7.3. Dynamic Analysis

For the dynamic analysis of a multi-body system, the inertial forces, the constraining forces, the potential forces, and any externally applied forces must be kept in equilibrium. The Euler-Lagrange equations are used by ADAMS/Solver to generate the equations of motion and, as listed previously, are the following:

$$\frac{d}{dt}\left[\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right)^{\mathrm{T}}\right]-\left(\frac{\partial L}{\partial \mathbf{q}}\right)^{\mathrm{T}}+\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda=\mathbf{Q} \qquad (68)$$

q is the column matrix of n generalized coordinates of the rigid bodies which describe the configuration of the system at any given instant in time. *L* defines the Lagrangian, which is the difference between the kinetic energy (*T*) of the mechanical system and the potential energy (*V*).

The first expression represents the inertial forces; the second expression represents the potential forces; the third expression represents the constraint (i.e., joints and motions) forces, and Q represents the externally applied forces.

The general form of the this equation appears a bit difficult to understand as a whole, but examining the components of the equations piece by piece will help in understanding how it generates the equations of motion. The application of the mechanical system consisting of a single moving part will illustrate the general case.

For the two-dimensional pendulum, the generalized coordinates q are:

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

More generally, q contains all of the n coordinates of the bodies that make up a mechanical system. For a two-dimensional system, n = 3N, where N = number of rigid bodies.

The Euler-Lagrange equations above require the Lagrangian to generate the inertial and potential forces:

$$L = T - V$$

For our two-dimensional pendulum, the kinetic and potential energy are, respecitively:

$$T = \tfrac{1}{2}(m\dot{x}^2 + m\dot{y}^2 + I\dot{\theta}^2)$$

$$V = mgy$$

Let's take the Euler-Lagrange equations one term at a time, starting with the left-most term for the inertial forces.

First, notice that $\frac{\partial L}{\partial \dot{\mathbf{q}}}$ is the momentum, and is an n x1 array:

$$\frac{\partial L}{\partial \dot{\mathbf{q}}} = \begin{bmatrix} \frac{\partial L}{\partial \dot{x}} \\ \frac{\partial L}{\partial \dot{y}} \\ \frac{\partial L}{\partial \dot{\theta}} \end{bmatrix} = \begin{bmatrix} m\dot{x} \\ m\dot{y} \\ I\dot{\theta} \end{bmatrix}$$

The inertia forces on the body in the respective directions (q) are then:

$$\frac{d}{dt}\left[\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right)^{\mathrm{T}}\right] = \frac{d}{dt}\begin{bmatrix}\frac{\partial L}{\partial \dot{x}}\\ \frac{\partial L}{\partial \dot{y}}\\ \frac{\partial L}{\partial \dot{\theta}}\end{bmatrix} = \frac{d}{dt}\begin{bmatrix} m\dot{x}\\ m\dot{y}\\ I\dot{\theta}\end{bmatrix} = \begin{bmatrix} m\ddot{x}\\ m\ddot{y}\\ I\ddot{\theta}\end{bmatrix}$$

The second term from the left is $\frac{\partial L}{\partial \mathbf{q}}$, which is an n x 1 array and indicates the sensitivity of the Lagrangian for the mechanical system to one of the coordinates. These are the potential forces; note that ADAMS/Solver includes only the potential force due to gravity in this term – all other potential forces are actually included in Q.

$$\frac{\partial L}{\partial \mathbf{q}} = \begin{bmatrix}\frac{\partial L}{\partial x}\\ \frac{\partial L}{\partial y}\\ \frac{\partial L}{\partial \theta}\end{bmatrix} = \begin{bmatrix} 0\\ mg\\ 0\end{bmatrix}$$

So, what we have thus far is:

$$\begin{bmatrix} m & 0 & 0\\ 0 & m & 0\\ 0 & 0 & I\end{bmatrix}\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{\theta}\end{bmatrix} + \begin{bmatrix} 0\\ mg\\ 0\end{bmatrix} + \Phi_{\mathbf{q}}^{\mathrm{T}}\lambda = \mathbf{Q}$$

This is a system of second order differential equations. Note that this system will be converted to a set of first order differential equations later by introducing velocity variables.

The forces in the translational direction are determined by the first two equations. In the third equation, the inertial load due to the angular acceleration is balanced by the net torque about the center of mass of the pendulum. Since there are no applied forces (Q) in this model, the last part of the Euler-Lagrange equations to determine is the constraint forces.

The Constraint Equations, Φ

The systems of equations that are generated thus far include five unknowns, x, y, θ, as well as the Lagrange multipliers (constraint forces), λ_1 and λ_2. However, we currently only have three equations, and thus the solution to the five unknowns is not completely determined. The additional equations required are the constraint equations, which are provided by joints and motions in mechanical systems. In ADAMS, users can also create general constraints by using the GCON statement with the C++ Solver.

For our two-dimensional pendulum, we currently have only a revolute joint to provide the constraint equations. The revolute joint attaches the body of the pendulum to ground, and ensures that:

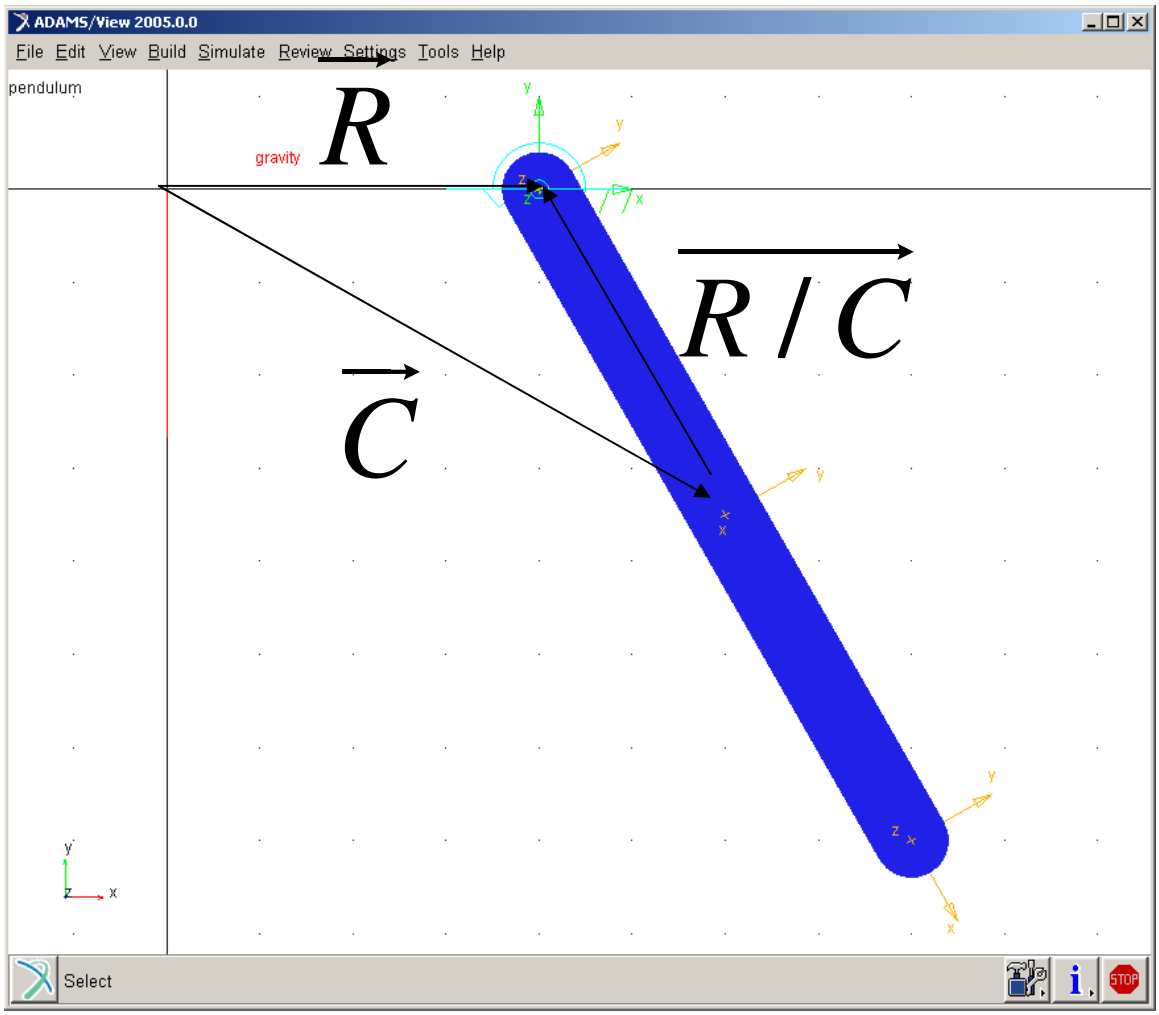


Figure 5

$$\vec{R} = \vec{C} + \overrightarrow{R/C}$$

…or, equivalently:

$$\vec{C} + \overrightarrow{R/C} - \vec{R} = 0$$

Where:

- The constant *R* is a vector which locates the pin (revolute joint) in the ground reference frame
- The variable *C* is a vector which locates the center of mass of the pendulum in the ground reference frame
- The variable *R/C* is the vector from the center-of-mass of the pendulum to the pin (revolute joint) attaching it to ground.

The condition imposed by the mechanism can be rewritten in vector form as the constraint equation:

$$\Phi(x, y, \theta) = (x + R/C_1 - R_1)\hat{i} + (y + R/C_2 - R_2)\hat{j} = 0$$

$$R/C_1 = -l\cos\theta$$
$$R/C_2 = -l\sin\theta$$

…where $\hat{i}$ and $\hat{j}$ are unit vectors along the x and y coordinate axes in the global (ground) reference frame. We can, alternatively, express this vector constraint in matrix notation:

$$\Phi(x, y, \theta) = \begin{pmatrix} x + R/C_1 - R_1 \\ y + R/C_2 - R_2 \end{pmatrix} = 0$$

The Jacobian matrix for Φ is the 2 by 3 array:

$$\Phi_q = \begin{pmatrix} 1 & 0 & l\sin\theta \\ 0 & 1 & -l\cos\theta \end{pmatrix} = 0$$

Associated with each constraint is a Lagrange multiplier that provides the constraint (reaction) forces. Using the constraint equations and the Lagrange multipliers, final term in the Euler-Lagrange equations can be formed:

$$\Phi_q^T \lambda = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ l\sin\theta & -l\cos\theta \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_1 l\sin\theta - \lambda_2 l\cos\theta \end{pmatrix}$$

The first two entries are constraint forces, and the third is the torque.

In summary, the power of the formulation process that uses the Euler-Lagrange equation is illustrated by the conversion of the constraint equations into the coupling terms. The dynamics of the mechanism are governed by:

- Three force-balance equations
- Two constraint equations

Combining the force balance and constraint equations gives us our mechanical pendulum:

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ mg \\ 0 \end{bmatrix} + \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_1 l \sin\theta - \lambda_2 l \cos\theta \end{bmatrix} = \mathbf{0}$$

Or, equivalently, by carrying out the matrix algebra:

$$\left.\begin{array}{c} m\ddot{x} + \lambda_1 \\ m\ddot{y} + \lambda_2 + mg \\ I\ddot{\theta} + \lambda_1 l \sin\theta - \lambda_2 l \cos\theta \\ x - l\cos\theta - R_1 \\ y - l\sin\theta - R_2 \end{array}\right\} = 0$$

This set of DAE's for our initial value problem is well-defined, with five equations and five unknowns.

Conversion to First-order System

The system above contains second-order differential equations. The formulation used by ADAMS/Solver GSTIFF formulation makes use of the method of introducing an intermediate variable for each higher order derivative to reduce the order of the system to first order.

Let

$$\mathbf{U} - \dot{\mathbf{q}} = \mathbf{0}$$

Here the components of $\mathbf{U}$ are:

$$U_x = \dot{x}$$
$$U_y = \dot{y}$$
$$U_\theta = \dot{\theta}$$

These are kinematic differential equations, also known as the velocity equations. Substituting the velocity equations into our system of DAE's for the pendulum yields

$$\left.\begin{array}{c} m\dot{U}_x + \lambda_1 \\ m\dot{U}_y + \lambda_2 + mg \\ I\dot{U}_\theta + \lambda_1 l\sin\theta - \lambda_2 l\cos\theta \\ U_x - \dot{x} \\ U_y - \dot{y} \\ U_\theta - \dot{\theta} \\ x - l\cos\theta - R_1 \\ y - l\sin\theta - R_2 \end{array}\right\} = 0$$

Thus, the second-order system has been reduced to a first order system with the unknowns:

$$\begin{pmatrix} U_x \\ U_y \\ U_\theta \\ x \\ y \\ \theta \\ \lambda_1 \\ \lambda_2 \end{pmatrix}$$

This system is well-posed at this point, but ADAMS/Solver adds set of equations to make the solution more computationally simple and to increase the scarcity of the equations. The angular momentum of the system is calculated explicitly and added to the system of equations, while the translational momentum is not calculated explicitly since it is a relatively easy quantity to compute. Thus, for our pendulum, the angular momentum, *p*, will be introduced to the system of equations as:

$$p - \frac{\partial L}{\partial \dot{q}} = 0$$
$$\Rightarrow p_\theta - I\dot{\theta} = p_\theta - IU_\theta = 0$$

Now, our final set of equations is:

$$\left.\begin{array}{c} m\dot{U}_x + \lambda_1 \\ m\dot{U}_y + \lambda_2 + mg \\ \dot{p}_\theta + \lambda_1 l \sin\theta - \lambda_2 l \cos\theta \\ p - IU_\theta \\ U_x - \dot{x} \\ U_y - \dot{y} \\ U_\theta - \dot{\theta} \\ x - l\cos\theta - R_1 \\ y - l\sin\theta - R_2 \end{array}\right\} = 0$$

…with these unknowns:

$$\begin{pmatrix} U_x \\ U_y \\ U_\theta \\ p_\theta \\ x \\ y \\ \theta \\ \lambda_1 \\ \lambda_2 \end{pmatrix}$$

Note the scarcity in the relationships between the variables in the system of DAE's. It is of considerable importance in the algorithms to invert the matrices for the solution process as well as in forming the analytical derivatives in the C++ Solver (as opposed to the FORTRAN solver which numerically perturbs the system to get the derivatives).

To express this system of DAE's in a more compact form, we can rewrite the equations as:

$$G(Y, \dot{Y}, t) = 0$$

..where *G* represents the column matrix of function of the unknowns and where

$$Y = \begin{pmatrix} U_x \\ U_y \\ U_\theta \\ p_\theta \\ x \\ y \\ \theta \\ \lambda_1 \\ \lambda_2 \end{pmatrix}$$

…is the column matrix of the unknowns. Again, the differential equations in the system are first order, and the system as a whole is nonlinear.

Notice that the system is implicit as opposed to an explicit form where

$$\dot{Y} = f(Y,t)$$

…and a set of ODE's could be solved. For the implicit integration algorithm, the DAE's are transformed into a system of non-linear *algebraic* equations by approximating each derivative in the column matrix $\dot{Y}$ with a backward differentiation formula (BDF) with GSTIFF. For numerically stiff systems (characterized by over-damped "high"-frequency eigenvalues and under-damped "low"-frequency eigenvalues) the BDF methods provide the largest stability region among the multi-step integration methods.

The Jacobian Matrix

One of the most expensive and difficult steps of solving the dynamics problem is in generating and inverting the Jacobian matrix. In this section, we will form the Jacobian matrix to give you a sense of both the structure and individual entries. In later sections, this can help you understand how the model is solved and how problems can occur in the simulation.

We need to convert our DAE's into an algebraic set of equations so that we can use Newton-Raphson to solve the problem. To accomplish this, the backward differentiation formula(s) can be used. The first order BDF is:

$$\dot{q} \approx \frac{q - q_{n-1}}{h}$$

…which defines the numerical relationship between a generalized coordinate and its first derivative (h is the time step) for the first order backward differentiation formula (BDF). For the higher order approximation, the equation above becomes

$$\dot{q} \approx \frac{q - q_{n-1}}{h\beta}$$

…where the coefficient β depends upon the order of the formula.

From the expression above,

$$\frac{\partial \dot{q}}{\partial q} = \frac{1}{h}$$

…for each unknown q in the state vector.

The Newton-Raphson algorithm for the solution of a system of non-linear algebraic equations requires the formulation of the Jacobian matrix for which the entry in the i^{th} row and j^{th} column is the partial derivative of the i^{th} equation with respect to the j^{th} unknown. Thus, after converting our DAE's with our BDF method, the Jacobian matrix is calculated to be:

$$\begin{bmatrix}
\frac{m}{h\beta} & & & & & & & 1 & \\
 & \frac{m}{h\beta} & & & & & & & 1 \\
 & & & \frac{1}{h\beta} & & & (\lambda_1 l\cos\theta + \lambda_2 l\sin\theta) & l\sin\theta & -l\cos\theta \\
 & & -I & 1 & & & & & \\
1 & & & & \frac{1}{h\beta} & & & & \\
 & 1 & & & & \frac{1}{h\beta} & & & \\
 & & 1 & & & & \frac{1}{h\beta} & & \\
 & & & & 1 & & l\sin\theta & & \\
 & & & & & 1 & -l\cos\theta & &
\end{bmatrix}$$

…where the entries that are not filled in are identically zero (here, β = 1 for Backward Euler).

Following the process outlined in Section 5, the Jacobian is evaluated and inverted repeatedly (as specified by PATTERN) to solve the equations of motion. Notice that several terms have the step size, *h*, in the denominator. When the step size is small, these terms become very large, and this produces an ill-conditioned Jacobian. Many options exist for dealing with this scenario and are described in more detail in Section 8.

7.4. Kinematic Analysis

The motion of the simple pendulum described previously is approximately sinusoidal only for very low-amplitude oscillatory motion. We can, for the sake of this discussion, impose a restriction that the motion be exactly sinusoidal. By controlling the horizontal (or vertical) motion of the center of mass, we are imposing another constraint upon the system. With an additional constraint, the remaining single degree of freedom is removed and the motion of the constrained pendulum becomes purely kinematic. Another way of saying this is that since we have three degrees of freedom and three independent constraint equations, the motion of the pendulum is completely prescribed by the constraint equations.

Sinusoidal motion could be achieved mechanically by attaching a slotted rod, constrained to slide back and forth, to a pin at the center of mass. The connection between the rod and the body of the pendulum would permit motion in the vertical direction while enforcing sinusoidal motion in the horizontal direction. We will achieve the effect mathematically by adding a constraint equation between ground and the center of mass of the pendulum.

The constraint equation for the motion-generator is

$$x - R_1 = C \sin \alpha t$$

..where the line of action is along the x-axis, and $\vec{R}$ is the location of the revolute joint for the pendulum in the ground reference frame.

With the addition of this constraint equation (motion) to our system, the new set of equations:

$$\left.\begin{array}{c} m\dot{U}_x + \lambda_1 \\ m\dot{U}_y + \lambda_2 + mg \\ \dot{p}_\theta + \lambda_1 l \sin\theta - \lambda_2 l \cos\theta \\ p - IU_\theta \\ U_x - \dot{x} \\ U_y - \dot{y} \\ U_\theta - \dot{\theta} \\ x - l\cos\theta - R_1 \\ y - l\sin\theta - R_2 \\ x - R_1 = C\sin\alpha t \end{array}\right\} = 0$$

The addition of another constraint equation to the DAE's has brought a fundamental change to the nature of the problem. Again, notice that the number of generalized coordinated equals the number of independent constraint equations, and thus the dynamics problem is now a kinematic problem characterized by zero degrees of freedom.

Since the motion of the system is completely described throughout all time, adding forces and torques to the system will not affect the *motion* of this system. Adding forces and torques will change the required forces for the constraint equations to impose this motion, however.

The Jacobian Matrix

The motion of a kinematics problem is completely defined by the constraint equations in the system. Although it is still possible to form the Jacobian matrix and integrate the full system of equations, it is no longer necessary. We only need to solve the three by three system of non-linear algebraic constraint equations, that is, the final three equations in the system above.

Beginning with an initial estimate of:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

…the Newton-Raphson algorithm will be used to solve for the position of the mechanism at any given time, *t*. Note that the initial estimate for the first time step is the design position, and the second time step is the solution from the first time step. However, for every time step afterward, a linear prediction based on the last two solutions is used to come up with the first estimate for Newton-Raphson.

The Jacobian matrix for our kinematic system is:

$$\begin{bmatrix} 1 & 0 & l\sin\theta \\ 0 & 1 & -l\cos\theta \\ 1 & 0 & 0 \end{bmatrix}$$

The kinematic solution generates values only for the displacement variables. To compute values for the remaining elements of the state vector Y, the constraint equations are differentiated once and evaluated with the computed values of the variables to obtain the velocities. Next, the equations are differentiated and evaluated a second time to obtain the accelerations. Finally, the force balance equations are used to compute values for the constraint forces.

7.5. Static Analysis

We can generate the set of static equations for the simple pendulum by starting with the dynamics equations we formed previously. In order to find a static solution, our pendulum must have at least one degree of freedom, and thus our kinematic problem in the last section cannot find a static solution. What needs to be done to form our static problem is to set all time-varying terms to zero since our objective is to find a set of states where the system does not change with time.

So, starting from our dynamics equations:

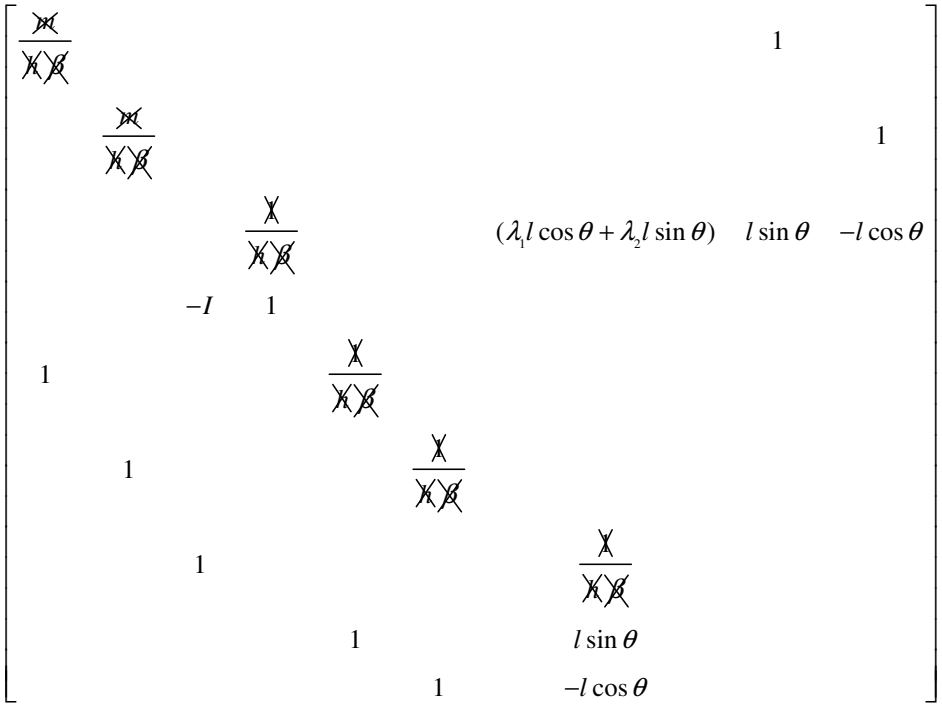
$$\left.\begin{array}{c} m\dot{U}_x + \lambda_1 \\ m\dot{U}_y + \lambda_2 + mg \\ \dot{p}_\theta + \lambda_1 l \sin\theta - \lambda_2 l \cos\theta \\ p - IU_\theta \\ U_x - \dot{x} \\ U_y - \dot{y} \\ U_\theta - \dot{\theta} \\ x - l\cos\theta - R_1 \\ y - l\sin\theta - R_2 \end{array}\right\} = 0$$

Remove all derivatives leaves:

$$\left.\begin{array}{c} m\cancel{\dot{U}_x} + \lambda_1 \\ m\cancel{\dot{U}_y} + \lambda_2 + mg \\ \cancel{\dot{p}_\theta} + \lambda_1 l \sin\theta - \lambda_2 l \cos\theta \\ p - IU_\theta \\ U_x - \cancel{\dot{x}} \\ U_y - \cancel{\dot{y}} \\ U_\theta - \cancel{\dot{\theta}} \\ x - l\cos\theta - R_1 \\ y - l\sin\theta - R_2 \end{array}\right\} = 0$$

In turn, the Jacobian is used to effectively steer Newton-Raphson to a static configuration by adjusting the generalized coordinates (i.e., positions/orientation) and reaction forces.

The Jacobian Matrix



Keep in mind that there as static uses Newton-Raphson to find the equilibrium solution, it makes no consideration to find a minimum energy configuration. Thus, it is possible to find an equilibrium position where the pendulum is straight down (stable) or straight up (unstable). Use of ADAMS/Linear to look at the eigenvalues of the system would reveal positive eigenvalues when the pendulum is straight up, and negative eigenvalues when down. One option is to use the EQUILIBRIUM/DYNAMIC, which allows an exit criteria for equilibrium based on kinetic energy of the system.

8. Ways and Means for Improving your MSC.ADAMS Simulation

8.1. Settings in the INTEGRATOR Statement

This section discusses some of the INTEGRATOR statement attributes that have proven to be the most effective in changing the behavior of the integrator in ADAMS/Solver.

a) By far the most used integrator in ADAMS is GSTIFF. It comes in at least two flavors, the I3, and the SI2 (the ADAMS/SolverFortran has I1 as well). The I3 GSTIFF formulation has been around for more than two decades. As such, it has experienced many improvements and it is now considered to be the most robust integrator. It doesn't mean however that there aren't models that will not run with GSTIFF but run with WSTIFF. In fact in case you are not successful to run your simulation GSTIFF, you should try to use WSTIFF, or (in ADAMS/SolverC++) the new HHT integrator.

b) Speaking of the SI2 versus the I3 GSTIFF integrators, it is worth noting that the latter is typically less picky when it comes to discontinuities handling. The SI2 integrator gives nicer and smoother results, but they usually come at a higher CPU time, unless you relax the ERROR setting when compared to the value that is used for the same model simulated with GSTIFF. A good starting point for ERROR when using SI2 is 10 times the ERROR for I3.

c) The HINIT setting comes handy if you know that at the beginning of the simulation there are high transients. Don't by shy about setting HINIT to small values; if there are no high transients, the integrator will quickly increase the value of the integration step-size. However, if there are high transients, a small value of HINIT, e.g. 1.E-7, might help the integrator prevent rejected or failed time-steps in the beginning of the simulation

d) The HMAX setting is useful if you want to

d1) get the integrator to basically run at constant step-size (set a small value for HMAX, and a large value for ERROR)

d2) prevent the integrator from skipping over very short events (like contact between two thin skin geometries)

e) ERROR is the parameter that allows you to indicate how accurate you want the results to be. Always remember that ERROR means different things for different integrators. It is only by using an integrator for a while in conjunction with your problem that you will get an intuition about what ERROR setting strikes the right compromise between accuracy and efficiency. Too loose of a value might produce results that towards the end of the simulation become very inaccurate, or sometimes it can lead to outright simulation failure. Too strict of a tolerance can result in large simulation CPU times and/or simulation failure (if the integrator tries to reduce the step-size to very small values to meet the accuracy demand).

f) The INTERPOLATE attribute when set to ON can improve the quality of the results by preventing some of the spikes. Typically, switching to

INTEROPLATE=ON in GSTIFF slightly slows down the simulation. The HHT integrator in ADAMS/SolverC++ only marginally slows down when in interpolate mode.

g) MAXIT is an attribute not recommended to change unless you have a very good reason to do so. One thing that has been reported to be useful was to set MAXIT=7, and PATTERN=F:F. By doing this, the Solver only re-evaluates the integration Jacobian when the corrector fails to converge, which happens after 7 iterations in this case. This has resulted in fast simulations for models that are close to linear, or which do not change much their configuration in time.

h) If you set PATTERN=T, then the jacobian is evaluated at each iteration (very expensive!!!), but the rate of convergence is theoretically quadratic since the resulting method is Newton-Raphson (see section 1.1). On the other hand, if PATTERN=F:F, then the jacobian is only evaluated when the corrector fails to converge. Note one important difference between the FORTRAN and C++ solvers. If PATTERN=F in the C++ solver, that means that the integrator is given control how often to evaluate the Jacobian. Remember that doing it too often is in general expensive; doing it too seldom might lead to convergence failure. The C++ Solver tries to reach a compromise with an adaptive Jacobian pattern, and as indicated this strategy kicks in by setting PATTERN=F. To conclude, in C++ Solver, you can obtain the FORTRAN PATTERN=F behavior by setting PATTERN=F:F.

i) The attribute KMAX is not something that is usually modified. It has been noticed that for models that have very high mechanical stiffness, such as models with flexible bodies, setting KMAX=2 sometimes results in shorter simulation times.

j) The CORRECTOR attribute might come handy for models where the corrector has a hard time converging the Newton-Raphson process. By choosing the MODIFIED corrector the integrator will accept more readily a system configuration as being “converged”. This is useful for models with contacts and discontinuities. Use this feature with care, since this approach can sometimes lead to results that are different than the actual solution.

8.2. Settings in the EQUILIBRIUM Statement

Static analyses in ADAMS are mostly carried out using the STATIC rather than the DYNAMIC approach, and the focus of the discussion below is reflecting this bias. It should be pointed out that finding an equilibrium configuration is a challenging task, particularly when the initial configuration of the system if far from the equilibrium configuration. When the STATIC approach is used, the equilibrium configuration is obtained as a solution of a non-linear system, and therefore if the initial guess is far from the solution the Newton-Raphson algorithm diverges (see section 1.1)

As a general piece of advice, always consider turning on the debug information during the static analysis (having DEBUG/EPRINT in your command file before the static simulation turns on the debug information; you can turn it off with DEBUG/NOEPRINT afterwards; see section 8.5.1 for more information on DEBUG/EPRINT).

a) ALIMT is one way to "tame" large corrections in the Newton-Raphson iterations (you can verify how large the Newton-Raphson corrections are by inspecting the DEBUG/EPRING output). Every single time the largest angle (orientation) correction that is suggested by the Newton-Raphson algorithm exceeds the value ALIMIT, *all* the corrections are scaled by a value that renders the largest orientation correction equal to ALIMIT. As an example, consider that ALIMIT=30D (which is actually the default value), and the first correction in the Newton-Raphson algorithm suggests an angle correction of 45D. If this is the largest angular correction, all the corrections will be scaled by a factor 1.5 (=45D/30D). Therefore, indirectly, *all* the corrections (including translational corrections), are scaled down.
   As a rule of thumb, if the static analysis fails, try setting ALIMIT=10D, or even to a smaller value. However, you should also increase MAXIT, since as you restrict the solver to small corrections, you will have to allow for more iterations to reach the final equilibrium configuration
b) ERROR regards the correction convergence threshold; in other words, the static analysis will not be considered successful if the most recent Newton-Raphson iteration produces a state correction that is larger in magnitude than ERROR.
c) IMBALANCE regards the residual in satisfying the non-linear equations that need to be satisfied for equilibrium (see section 6). If your model has a hard time finding an equilibrium, try the "funnel" approach: first run a static analysis with lax ERROR and IMBALANCE values, a large value for STABILITY, say 1.0 (see below), a conservative value for ALIMIT (say 10D), and allow for a large number of iterations, say MAXIT=250. If this attempt succeeds, decrease ERROR, IMBALANCE, and STABILITY by a factor of 10, and run a second static analysis. If this is successful, tighten even further the simulation settings. The idea is that you run a succession of static analyses with more and more conservative settings; thus, rather than finding the equilibrium configuration in one shot, you use a "funnel" approach to finding it.
d) MAXIT indicates the maximum number of Newton-Raphson iterations allowed to find an equilibrium configuration. Allow large values for MAXIT, particularly so if you have changed the default values for ALIMIT (by decreasing its value), or STABILITY (by increasing it). Note that a very large value for MAXIT does not hurt. Have DEBUG/EPRINT on, and watch on the screen how your residuals and corrections hopefully decrease with each iteration. Having a large MAXIT will increase your chances of success
e) PATTERN is by default set to "true", therefore the Jacobian is evaluated at each iteration. This makes the algorithm a true Newton-Raphson method (see section 1.1). This setting has not proved to be very helpful if changed, and the CPU penalty of evaluating the Jacobian at each iteration is not prohibitive, since in general statics analysis do not take long simulation times.
f) STABILITY is a factor that can greatly impact the success or failure of your static analysis. An analogy can be made between the roles that the STABILITY and the integration step-size play in the static, and dynamic analysis, respectively. Having a very small value for the STABILITY attribute would be equivalent to

having a very large step-size in the integrator. Especially for challenging models that fail to find equilibrium, setting a larger value for STABILITY effectively makes the static algorithm be less aggressive. Recall the "funnel" approach discussed above (see c) above). In the beginning take large STABILITY values, and when you get closer to the solution be more aggressive and take smaller values. A large STABILITY value is in the range 0.1-5. A small value is 1.E-5 (which is the default).
Note: when experimenting with the STABILITY setting always turn on DEBUG/EPRINT to be able to judge how this attribute is influencing the convergence of the static analysis for your model.

g) TLIMIT qualitatively plays the same role that ALIMIT does, except that it monitors the corrections in the positions rather than orientations. It has been noted that in general changing the default setting of this attribute does not have a great impact on the robustness of a static analysis, or at least not as much as changing the ALIMIT attribute.

8.3. Rules of Thumb for Creating Robust Models in ADAMS

Below are presented a series of "rules of thumb" or "best practices" for modeling that will lead to more robust and faster ADAMS simulations.

1. Avoid *discontinuities.*

This is the most important advice that can be provided. It cannot be stressed enough that by far, discontinuities are the root cause of most simulation problems in ADAMS. Avoid them. Examples of discontinuous functions are MIN, MAX, DIM, MOD, IF, etc. Discontinuous displacement, velocity, or acceleration motions cause corrector and integrator failures, and produce large integration errors. Likewise, discontinuous forces cause corrector failures.

To understand the reason why discontinuities are causing problems, recall that an integrator like GSTIFF is a variable order integrator. The order of the integration formula used by GSTIFF can be anywhere between one and six, and always, the higher the order the better the results will be. The user does not have control over how the integrator chooses its order since this is something determined inside the integrator based on some optimality conditions. However, the user can help the integrator in succeeding to choose high orders (five or six) by making sure that there are no discontinuities in the model. The rule is that for the integrator to work at order p (where p for GSTIFF is between one and six), all the functions appearing in the model (motions, forces, variables, etc.) should have at least $p+1$ continuous derivatives. Thus, for instance if you hope to have a simulation during which GSTIFF works at order six, then you need to have all the functions in the model have at least seven continuous derivatives.

If you invest the time and make sure that the system is modeled right, you will be rewarded by a large integration order. The reason why you want a large integration order is that the local integration error is proportional to h^{p+1}, and this quantity multiplied by a scaling factor should be less than the user prescribed ERROR. Since the value of the step-size h is typically less than 1.0, it's easy to see that the higher the order the integrator works at, the larger the step-size h can be and still not violate the user selected ERROR.

In the end, if you can run a simulation at large step-sizes and high order it is guaranteed that your simulation will finish quickly, and the quality of the results is going to be very good.

Finally, please note that even when a function looks smooth, what is important is how many derivatives of the function continue to be smooth. For instance, if a cubic spline is used to represent a motion then the function will have two continuous derivatives. The third derivative will be already discontinuous, and therefore GSTIFF will be prevented to increase the integration order to high values. An easy fix to this problem is to actually specify the motion as a velocity rather than a position level motion. This simple change will potentially increase the GSTIFF integration order by one, a very positive change.

2. Choose the modeling units wisely

A poor choice of modeling units leads in different equations to entries that are vastly different in magnitude. The end result is that during the solution sequence the Solver will encounter matrices that have very large condition numbers. If the situation is not extremely bad, a matrix with large condition number will lead to very poor quality corrections in the Newton-Raphson algorithm for example (see section 1.1) and therefore large number of iterations for convergence. If the condition number is very large, the corrector stage of the solution can fail altogether, or the matrix is identified singular by the linear sparse solver.

To prevent situations like these:

a. Choose units so that model states (displacements and velocities) assume reasonable values. For example, choosing "mm" for displacements of a rocket, which travels thousands of kilometers, is a poor choice.
b. Choose time units appropriate to the phenomena being studied. If duration of dynamic event time is short, consider using MILLISECONDS units.

3. If it's possible, avoid dummy parts (any part with zero or very small mass)

Keep in mind these recommendations:

a) Sometimes dummy parts are useful, but if possible, avoid using them.
b) Avoid connecting dummy parts with compliant connections (BEAM, BUSHING, etc.). If the mass is very small, qualitatively, acceleration = Force/mass = very large number. Thus, under certain circumstances small masses/moments of inertia introduce high frequencies and/or jolts into the system, which has detrimental effects on the solver performance.
c) If you must use dummy parts, then constrain all its degrees of freedom, since with no degrees of freedom for the dummy part, qualitatively, acceleration=Force/mass is not an issue.
d) Dummy parts should be massless, 0.0 (or unassigned), and not 1e-20.

4. Keep an eye on the JOINT elements in the model

In most simulations the use of JOINTs in the model is not leading to robustness issues. However, avoid redundant constraints. ADAMS/Solver tries to eliminate them by

looking at pivots in a constraint jacobian, which are in no particular order. As a result the physical meaning may be disregarded. In general, model that has redundant constraints is an example of poor modeling practices, and it's an indication that the nature of the system being modeled has not been understood. Redundant constraints can cause eigenvectors from ADAMS/Linear to produce unwanted results, as the DOF kept may not be the ones you are interested in for the linear analysis.

5. MOTIONS

Ideally, motions should be defined through functions that are as smooth as possible; i.e., that have a large number of continuous derivatives. A simulation in which all the inputs are smooth will run with high integration orders.

a) Remember the advice number 1 above: keep the expression of the MOTION as smooth as possible, and avoid discontinuities
b) Avoid using SPLINE functions in a MOTION definition (they only have two continuous derivatives).
c) If you have to use a spline function in a motion though, you are better off defining the spline at the velocity rather than position level. For more information see the MOTION statement in ADAMS/Solver User Manual (see also Knowledge Base Article #9752)
d) Avoid defining a MOTION as a function of variables (i.e., states).
e) A cubic spline (CUBSPL) may in general work better on motions than the Akima spline. The derivatives of the Akima are not as nice as those of the cubic spline (they are more useful in forces rather than motions, see Knowledge Base Article #7534).
f) Consider filtering data to smooth out spline data for motions.

6. FORCES

a) Remember advice number 1 above: keep the expression of the force as smooth as possible, and avoid discontinuities.
b) If using data, approximate forces with smooth, continuous spline functions.
c) Make sure velocities are correct in force expressions. For example, in this damping function: $-c \cdot VX(12,25,25)$, the fourth marker is missing. This marker defines the reference frame in which the time derivatives are taken, and this may be important.

7. CONTACTS

a) Contacts should penetrate before statics. Models with impacts should have slight penetration in model position when doing statics.

b) All tires should penetrate the road. Models with tires should have slight penetration in model position when doing statics. For example, if only rear tires penetrate, the static position could be a "handstand".
c) Contact properties are model dependent. See the CONTACT statement documentation and Knowledge Base Article #10170 for a starting point. Adjustment of the properties to match experimental results is expected.
d) For contact models, set the INTEGRATOR attribute HMAX=0.01 (a good initial guess). If problems are encountered, some times a smaller 0.001 values is better. Note that many times problems come from the use of thin shells. If the input geometry is very thin, there is a possibility that one geometry may completely pass through another, resulting in invalid volume of intersection calculations. This can result in missed contacts, pass-through, or generation of unusually high contact force generation
e) It's not a bad idea to set INTEGRATOR/HINIT also, if there is a possibility of sudden movement at the beginning of the simulation. Set HINIT to a small value, e.g. 1.E-7, to give the integrator a chance to ease into the simulation.
f) Don't turn contact friction on (if possible) until everything else is working
g) The contact penetration depth parameter (CONTACT/DMAX) can force the integrator to take extremely small step sizes if it is too low (it should not usualy be less than 0.1 mm or 0.004 in).

8. SUBROUTINES

a) Always use an ADAMS function over a subroutine, if possible. The quality of the partial derivatives is incomparably better when a function for a force for instance, is defined via a statement rather than a user subroutine. Consequently, the quality of the Jacobian in a Newton-Raphson algorithm is going to be better, which typically results in better convergence properties
b) If using ADAMS/SolverC++, you can provide the partial derivatives from within your subroutine in order to improve the overall quality of the Jacobian. The function call SYSPAR is the means through which the user can feed the partials back into the ADAMS/SolverC++. As indicated earlier, better partials result in better Newton-Raphson convergence.
c) If you receive errors in your model, for the purpose of debugging eliminate user-subroutines so as they are not the source of error.
d) Make sure that your compiler is compatible with the current version of MSC.ADAMS.

SIMULATION/INTEGRATORS

a) When applicable, perform an initial static analysis first. Note that a static solution may be more difficult to find than a dynamic solution. If you care only about the dynamic solution and cannot find static equilibrium, then either increase your equilibrium error tolerance, or forget about the static simulation.

b) If GSTIFF won't start, it is most likely a problem with initial conditions. Try to set HINIT to values that are conservative, and thus the integrator will get a chance to slowly ease into the simulation.
c) Read the documentation about the new corrector implemented in ADAMS and use it when having problems with discontinuities. Understand the benefits and drawbacks associated with CORRECTOR=NEW seeting.
d) Don't let the integrator step over important events. Short duration events, such as an impulse, can be captured by setting the maximum step-size HMAX to values that are less than the width of the impulse.
e) Use a conservative value on HMAX combined with a lax ERROR setting if you want to cause the integrator to acts like a fixed-step integrator. There is a consensus that preventing frequent step-size changes improves the quality of the solution.
f) Spikes in results output may arise from changes in step size. Reduce HMAX or try setting HINIT=HMAX. Run with SI2 instead, or use the INTERPOLATE=ON feature.
g) ADAMS/Solver uses a body 313 rotation sequence (psi, theta, phi). When the second rotation angle (theta), becomes zero, the model is in a singular configuration (there is an infinite number of configurations for the psi and phi angles which lead to the same overall orientation; likewise, the time derivative of the orientation angles run into problems in a singular configuration). When the system finds itself into a configuration very close to a singular one, internally the solver changes the orientation of the body principal axes. This operation always requires an integrator restart. As a rule of thumb, if the z-axis of the part principal inertia axis is parallel to z-axis of ground there will be an Euler singularity. If you have prior knowledge about how the orientation of the bodies in the model is going to change time, maybe you can define the axes of the IM marker such that Euler singularities are avoided. Euler singularities lead to simulation slow-down, and spikes in results (since the integrator is restarted at a small step-size and at order one).
h) In the ADAMS/SolverC++, if your model already runs fine and you are after gaining some extra speed-ups, try to set the Jacobian evaluation pattern to F, like PATTERN=F. This will effectively change the strategy of evaluating the Jacobian. Recall that if you don't specify anything the Jacobian evaluation PATTERN is T:F:F:F:T:F:F:F:T:F. However, if you set PATTERN=F, the integrator will take over the task of deciding how often the Jacobian needs to be evaluated (adaptive Jacobian mode). It was noticed that for some models this strategy leads to speed-ups of up to 15%. Note that (i) if you want to actually never have the Jacobian re-evaluated unless a convergence failure occurs (helpful for linear models), you can specify PATTERN=F:F; (ii) the adaptive Jacobian mode works only for GSTIFF (both I3 and SI2) in the ADAMS/SolverC++.
i) In the ADAMS/SolverC++, if your model has high frequencies or discontinuities, you might want to use the new HHT integrator. In our experiences quite often the HHT integrator produced sizeable speed-ups in comparison with GSTIFF I3 and SI2.

SIMULATION/STATICS

Remember that the static analysis if a very challenging analysis, particularly when the initial configuration of the system if far from equilibrium. The equilibrium configuration is typically obtained as a solution of a non-linear system, and therefore if the initial guess is far from the solution the Newton-Raphson algorithm diverges (see section 1.1)

a) Allow MAXIT to assume large values (200 for example). This is not going to affect the speed of convergence, but it will give the Solver quite a few chances to find an equilibrium configuration
b) Unless you start close to the equilibrium configuration use a STABILITY value that is large (0.1 to 2.0, rather than the 1.E-5, which is the default). A large value of the STABILITY parameter is an indication that you are not in a hurry to reach the equilibrium, but you are content with the Solver taking small steps towards the equilibrium configuration. Large values of the STABILITY attribute indicate that you allow the Solver to be very aggressive in making corrections in the model configuration. It is a good idea to experiment with the value of this settings, as in our experience we found it to have quite an effect in the success/failure of the static analysis.
c) Use the attribute ALIMIT to limit the value of the orientation update that the Solver is allowed to take. The default is 30D, but for problematic models more conservative values such as ALIMIT=10D can actually help with the overall convergence. For more challenging models you might want to also change the default value of the TLIMIT attribute, although in our experience this had a smaller impact.

FRICTION

a) ADAMS has its own friction model implemented. However, if you are not satisfied with the existing friction model, new friction models (such as LuGre, elasto-plastic, etc) can be implemented using ADAMS function and user-subroutine support.
b) Note that unlike the ADAMS/Solver C++, the ADAMS/SolverF77 does not allow friction in joints connecting flexible bodies.

DEBUGGING

a) While trying to debug or validate your simulations, always turn on DEBUG/EPRINT. You should only turn off the DEBUG/EPRINT command when you are comfortable with the results and satisfied with the CPU time required to complete a simulation.
b) Understand the output sent out to the user, and see where the Solver has a hard time advancing the simulation. Monitor and correct any warnings sent out by the solver

c) Try to understand the system that you are simulating from a physical standpoint.
d) Use "building blocks" of concepts that worked in the past. Add enhancements to the model using a crawl-walk-run approach.
e) Test with small models to isolate problems.
f) Have graphics for visualizing motion.
g) Look at damping terms as a source of errors. Incorrect sign, missing terms are typical mistakes.
h) Models should have no warnings during simulation (e.g., redundant constraints, spline functions, etc.)
i) Understand numerical methods, and particularly understand your integrator.
j) Look for results which become very large in magnitude; this could indicate a discontinuity

MISCELLANEOUS

a) Avoid very large numbers and very small numbers. Be wary when your model contains numbers like 1e+23 or 1e-20. This is most likely a symptom of a bad units choice (see the UNITS section above), and it typically results in matrices with large condition numbers.
b) Extend the range of spline data beyond the range of need.
c) Don't write function expressions that can potentially divide by zero. The quick and dirty way to do this is to use the MAX function, as in FUNCTION = 8/MAX(0.01,your_function). However, there are more intelligent ways of doing this, since the function MAX is discontinuous, which leads to problems of a different nature (remember the advice number 1 above)
d) Add moderate damping in your model so frequencies can die off. This will speed up your simulation.
e) Avoid very high damping rates. High damping is causing a rapid decay in response, which is difficult for an integrator to follow.
f) Avoid toggles, dual solutions, or bifurcations since they lead to discontinuities.
g) Don't use 1.0 for the exponent in IMPACT or BISTOP functions. This creates a "corner" (i.e. a non-smooth function). Try 2.2 for the exponent instead (note that this parameter depends on units).
h) The ADAMS/SolverC++ has support for 2D parts. A 2D part adds fewer equations to the overall problem and especially for such elements as chains and belts this can results in more robust and faster simulations.

8.4. Validating your Simulation Results

Before checking on the quality of results and ensuring that the results are converged, the assumption is that you have applied the recommended debugging techniques to eliminate all modeling errors (see section 8.5 for debugging support in ADAMS). Likewise, it is assumed that you have followed to the extent possible the rules

of thumb for creating robust ADAMS models (see section 8.3). With these tasks out of the way, set DTOUT (END/STEPS in the output statement/command) to suit your time-sampling simulation requirements. For a correlation study, i.e., correlation with test data, this value is typically driven by the sampling rate of the test data. For a given DTOUT (or END/STEPS combo) take the following annotated steps:

1. Start with an initial error tolerance value for the INTEGRATOR statement. The starting point could be the Solver default value of 1E-3.
2. Build/Run simulation with DEBUG/EPRINT turned on, with the initial estimate of error value.
3. Reduce error by a factor of your choosing - a factor of 10 is commonly used.
4. Build/Run simulation with DEBUG/EPRINT turned on, with the reduced error value.
5. Compare the results between the simulations with original and reduced ERROR values. Check results, particularly displacements first if you are using I3 integrator(s). You will be looking at one of the following two scenarios.
    a. If results don't match, reduce error tolerance further from the current value by a factor of your choosing, and start again from Step 2.

    b. If results match closely, compare (gdiff) message files between the two simulations. Ensure that Solver is "doing more work" (taking more steps) because of the reduced error tolerance value.
    If you are not seeing more integration steps being taken in the simulation with the reduced error tolerance value, then most likely you need to decouple the effects of step-size and integration error control. In this situation, either your output step size DTOUT, or HMAX setting is smaller than the integration step-size that would otherwise be required by your ERROR setting, and is thus controlling the accuracy of your results. To decouple DTOUT (or HMAX) and ERROR, make DTOUT (and/or HMAX), much bigger (fewer steps, bigger steps) and use ERROR to do the study as before until you return to case a) above and find that the reduced error tolerance value did cause solver to “do more work”.

6. At this point, reset your error tolerance to the larger of the ERROR values from the two simulations that proved that you had a converged solution (the simulations that gave the same results). At this point you can be comfortable that you have achieved a converged solution for displacements.
NOTE: Dealing with corrector/integrator problems (will most likely happen), is handled in the ADAMS/Solver documentation, under the INTEGRATOR statement/command section. More information is available in the MSC.ADAMS Knowledge Base at http://support.adams.com/kb, or in section 8.1 of this document.
7. Follow steps 1 through 5 if you want to converge the acceleration results (when using an I3 integrator the accelerations are known to sometimes display spikes, which typically are more difficult to eliminate). In our experience the acceleration spikes can be eliminated or reduced by slowly decreasing the value

of HMAX and/or ERROR, while switching to INTERPOLATE=ON in the INTEGRATOR statement/command. In case this approach does not work you might also want to read the Knowledge Base Article 12400, which presents a slightly different alternative for converging the acceleration results. See also section 8.3 of this document for other suggestions.

8. If both the displacement and accelerations have been converged, it is recommended to run the simulation with SI2 as a last step in this process, the idea being that all integrators should converge to the same solution, albeit with different settings.
   If SI2 will run with the settings determined by steps 1 through 7 for I3, then it will most likely be converged to the same solution. This is because SI2 will converge displacements with a larger setting of ERROR than needed with I3. Typically, velocities will also be converged during the ERROR part of the process (steps 1 through 5) when using SI2. The advantages of SI2 are documented in the ADAMS/Solver documentation, but the main points to be noted are (*i*) SI2 is more accurate for velocities and accelerations, and (*ii*) the Jacobian matrix is very stable at small step sizes (something to keep in mind when you are reducing HMAX with I3). One caveat with SI2 is that motions must be smooth and twice differentiable. Non-smooth motions, which cause infinite accelerations, are a common cause of failures with SI2. As pointed out in section 8.3, you should always strive to provide smooth inputs to your model, be it of force of motion type.

On a final note, keep in mind that sometimes in spite of all your efforts you won't be able to converge the results. Almost always either a modeling error, or a non-smooth input function is responsible for this.

8.5. Debugging Support in ADAMS/Solver

The user can get a wealth of information about the states and internal doings of the Solver as it advances in time the configuration of the model. There are three main statements/commands that expose this information to the user:

a) DEBUG/EPRINT
b) DEBUG/RHSDUMP
c) DEBUG/JMDUMP

8.5.1. DEBUG/EPRINT

This statement/command is used to obtain a synopsis of the processes that take place during a solution sequence. A solution sequence should be regarded as any action or algorithm invoked by the Solver to advance the simulation or to execute a user issued command. For example, issuing a "simulate/statics" command will cause the Solver to a) set up the set of equations that needs to be solved to find an equilibrium configuration; and b) employ and algorithm to find the solution of the assembled equation set. DEBUG/EPRINT will send out informative messages and warnings during both stages a)

and b). Acting on the warnings sent out by the solver is going to make for better models and more robust/fast simulations. As an example of a warning message, you might get

```
---- START: WARNING ----
 An Out-of-range X value has been used in the spline calculation.
 Extrapolation is required for curve 1 of SPLINE/123.
 The X value (-100.000000) should be between -50.000000 and 55.000000
---- END: WARNING ----
```

In this case, the user defined the spline curve anticipating that the free parameter will always assume values anywhere between –50 and 55. Nevertheless, the model configuration is such that the spline is being evaluated for a value of –100. This is an indication that either the spline range initial assumption was wrong and the bounds should be altered, or that somewhere else there could possibly be problem in the model that led to unreasonable values for the spline parameter.

There are some differences in the amount and type of information that is put out by the ADAMS/SolverC++ and ADAMS/SolverF77. Either one though sends out information summarizing the progress towards finding a solution. The algorithms employed are specific to each analysis type (position IC, statics, kinematics, dynamics, etc.), but the common denominator is that the solution is always computed as a set of successive iterations that each corrects the previous system configuration to bring it closer to the actual solution. This iterative process can and should be monitored for each model that is problematic. Below is shown an actual portion of the output that you get by having DEBUG/EPRINT and running a dynamics analysis with the GSTIFF integrator.

```
Integration step No. 7,         Order = 2           Cumulative iterations of
 From  5.000000000E-04, Step =  8.33333E-05            the corrector: 315
Iteration   Residual (or equation error)    Change in the Variable     Jacobian
count    Maximum  Element/ID  Equation    Maximum  Element/ID  Variable   new?
  ___   ________ _____________________  ________ _____________________  ___
   1    -4.7E+05  Part/10 Phi Torque      6.2E+05  RevJnt/13 LAM          yes
   2     2.5E+00  Part/11 Psi Torque      5.0E+02  RtnMot/2 LAM           no
   3    -2.0E-03  Part/11 Psi Torque      2.6E-01  RevJnt/15 LAM          no
   4     1.3E-04  Part/10 Theta Torque    1.4E-04  RevJnt/13 LAM          no


Integration step No. 8,          Order = 2           Cumulative iterations of
 From  5.833333333E-04, Step =  8.33333E-05            the corrector: 319
Iteration   Residual (or equation error)    Change in the Variable     Jacobian
count    Maximum  Element/ID  Equation    Maximum  Element/ID  Variable   new?
  ___   ________ _____________________  ________ _____________________  ___
   1    -4.1E+05  Part/10 Theta Torque    6.7E+05  RtnMot/2 LAM           yes
   2    -9.0E-01  Part/11 Psi Torque      5.5E+01  RtnMot/2 LAM           no
   3    -9.2E-05  Part/11 Psi Torque      2.1E-02  RevJnt/15 LAM          no
   4    -1.6E-05  Part/11 Psi Torque     -2.7E-05  RtnMot/2 LAM           no


Integration step No. 9,          Order = 2           Cumulative iterations of
 From  6.666666667E-04, Step =  8.33333E-05            the corrector: 323
Iteration   Residual (or equation error)    Change in the Variable     Jacobian
count    Maximum  Element/ID  Equation    Maximum  Element/ID  Variable   new?
  ___   ________ _____________________  ________ _____________________  ___
   1     2.6E+05  Part/10 Theta Torque     3.3E+05  RevJnt/13 LAM          yes
   2     1.4E-01  Part/11 Psi Torque     -5.9E+00  RevJnt/13 LAM          no
   3     7.7E-06  Part/11 Psi Torque      5.5E-02  TrnJnt/2 LAM           no
```

```
4    -8.6E-06  Part/11 Psi Torque      2.0E-03  TrnJnt/2 LAM          no
```

The information reported in this example is as follows:

a) The current integration step (7, 8, 9, etc). Note that this number increases provided the time step was successful. Although it is not the case in this example, suppose step 8 would fail. Then the integration step number will not be incremented to 9 unless the integrator can advance the simulation from time T=5.833333333E-04, where the integrator finds itself at the beginning of step 8
b) The integration order stays 2 over these three integration steps as indicated in the output. Recall that for GSTIFF, the integration order is anywhere from 1 to 6; a higher integration order indirectly indicates a smooth simulation.
c) The current simulation time is indicated after the word "From"; it says that for instance the integration step 9 finds the simulation at time 6.666666667E-04, ready to take an integration step of "Step = 8.33333E-05"
d) To advance the simulation to time 6.666666667E-04 (see integration step 9), the integrator reports that it has taken a total number of 323 corrector iterations.
e) Recall that in GSTIFF, a non-linear system is solved to retrieve the configuration of the system during implicit integration. As such, an iterative Newton-like method (see section 1.1) is employed to find out the solution. These iterations are indicated by the "Iteration count", which is reset at the end of each integration step. In this example notice that each of the three integration steps took the same number, that is 4, of iterations in the corrector stage in order to converge to a solution.
f) During each iteration, the solver reports the largest violation in satisfying the equations that it's solving for the given analysis mode (dynamics in this case). As it can be seen, initially the largest violation is quite large (2.6E+05 for integration step 9). The Solver also reports which equation registered this large imbalance. In our example, for integration step 9, this imbalance occurs in the torque-balance equation for part 10.
g) The last piece of information provided is the value of the largest correction that the Newton-like iterative solver computed. Recall from section 1.1 that these corrections gradually bring the system configuration towards the correct one (towards the solution). In our example, notice that the corrections become gradually smaller. This is the expected behavior, and convergence failure occurs otherwise.
h) Look for trends in f) and g) as a *starting* point for investigating your model. Keep in mind that the equations or correction reported most may have nothing to do directly with a modeling error and is only a symptom.

Note that the output shown before is specific to GSTIFF, and other algorithms/solver modes produce different but qualitatively similar output.

Although not appearing in the above eprint output, another piece of information that the Solver supplies to the user is the occurrence of a singular Jacobian matrix. This is in conjunction for instance with the correction phase in GSTIFF, or when trying to find an equilibrium configuration and the Newton-Raphson method runs into a singular Jacobian.

```
VMNPO:SING_MTX
   ------------------------------------------------------------------------
   The system matrix has become numerically singular with this pivot order!
   ------------------------------------------------------------------------
   Equation for the singular row ...... MOTION 270_1032.MOT10320301
                                ...... (row=MOTION/10320301 CONSTR.)
   Variable for the singular column ... PART 270_1032.PAR10323
                                ...... (col=PART/10323 Theta)
   Pivot magnitude ................... 4.05476E-17
   ------------------------------------------------------------------------


SOLVE:REGEN
   Symbolic refactorization attempted at time = 1.7274.
```

When solving for the Newton-Raphson corrections (see section 1.1) the Jacobian needs to be factored to determine its LU factorization. The pivots in the LU factorization; the diagonal entries in U, are chosen at the beginning of the simulation and they are re-used time and again upon a new call for the solution of this system. Especially for long simulations after a while the original pivot sequence results in a singular matrix. This requires a refactorization to compute a new pivot sequence in the LU factorization, and possibly a decrease of the integration step-size. The user is flagged when the solver runs into such a scenario, as indicated in the ADAMS/SolverF77 message above.

DEBUG/EPRINT is an extremely useful tool in diagnosing simulation problems. By inspecting eprint output for problematic models, you can find the root cause of your problem and thus be able to fix the problem yourself (see the discussion in sections 8.1 through 8.3), or call the ADAMS hotline and provide the support engineer with additional information that can help him/her diagnose the issue. To conclude, *always* start the process of tinkering with a model simulation by having the DEBUG/EPRINT on. You are encouraged to have eprint on and experiment with different settings of the INTEGRATOR, EQUILIBRIUM, etc., statements/commands. This is the way to gain understanding of what parameters influence what aspect of the simulation, and eventually become a power ADAMS/Solver user.

8.5.2. DEBUG/RHSDUMP

The DEBUG/RHSDUMP statement/command is intended for the power user who has a basic understanding about the concept of ADAMS state, residual, Newton-Raphson correction. This is the type of information that DEBUG/RHSDUMP will output and it will help the user to quickly

a) Inspect all the entries in the array of force-balance residuals, constraint violation, etc. to check that the values are as expected. For instance, a modeling error in an obscure element might be easily noticed in a large value in a very large initial residual in satisfying the force-balance equation. Note that each equation that is formulated in ADAMS is given a name, and this name along with the violation (residual) associated with that equation is sent out to the user

b) Inspect all the values of the state to find any state that has an abnormally large (or small value). ADAMS/Solver has names associated with every variable, such that the user can monitor name-values state information.
c) Inspect the values of the corrections computed at the end of *each* Newton iteration. This information is of limited use, but if an extremely large value is noticed, or the opposite – when a zero value is obtained where a non-zero correction is expected is an indication that either (*i*) some residual is not correctly computed (see a) above), or (*ii*) some Jacobian entries are not computed correctly.

Note that the amount of information that is sent out to the user in DEBUG/RHSDUMP mode is large. Keep this in mind when running long simulations with large models. Only turn this feature on at the beginning of the simulation for a short while, or right before the model runs into a problem that you try to diagnose.

8.5.3. DEBUG/JMDUMP

The information obtained with the DEBUG/JMDUMP command/statement is primarily used for debugging of the code by ADAMS developers. It is basically a dump of the Jacobian matrix assembled by the Solver for the purpose of carrying out Newton-Raphson iterations. The information sent out indicates each partial derivative that is computed in the process of assembling the Jacobian; that is, a value is provided, along with information regarding what equation this partial derivative is of, and with respect to what variable the partial is taken.

The information output with DEBUG/JMDUMP is typically useful for small models, and even then analyzing the output obtained with DEBUG/EPRINT and DEBUG/RHSDUMP is more useful and should be considered first. Note that for large models the amount of information generated can become very quickly overwhelming.

Appendix A. Integration Jacobian

This appendix contains the integration Jacobian as generated during dynamic analysis of a mechanical system. The integrator used was backward Euler, the DAE solution is based on an index 3 approach. $\mathbf{I}$ is the identity matrix of appropriate dimension.

$$\Psi_y = \begin{bmatrix}
\frac{1}{h}\mathbf{M} & \mathbf{0} & \mathbf{0} & \left[\Phi_p^T\lambda - \left(\Pi^P\right)^T \mathbf{f}\right]_p & \left[\Phi_p^T\lambda - \left(\Pi^P\right)^T \mathbf{f}\right]_\varepsilon & \Phi_p^T & -\left(\Pi^P\right)^T & \mathbf{0} \\
\mathbf{0} & \mathbf{I} & -\mathbf{B}^T\bar{\mathbf{J}}\mathbf{B} & \mathbf{0} & -\left[\left(K_\zeta\right)^T\right]_\varepsilon & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \frac{1}{h}\mathbf{I} & -\left[\left(K_\varepsilon\right)^T\right]_\zeta & \left[\Phi_\varepsilon^T\lambda - \left(\Pi^R\right)^T \bar{\mathbf{n}}\right]_p & \left[\Phi_\varepsilon^T\lambda - \left(\Pi^R\right)^T \bar{\mathbf{n}} - \left(K_\varepsilon\right)^T\right]_\varepsilon & \Phi_\varepsilon^T & \mathbf{0} & -\left(\Pi^R\right)^T \\
-\mathbf{I} & \mathbf{0} & \mathbf{0} & \frac{1}{h}\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & -\mathbf{I} & \mathbf{0} & \frac{1}{h}\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \Phi_p & \Phi_\varepsilon & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
-\mathbf{F}_u & \mathbf{0} & -\mathbf{F}_\zeta & -\mathbf{F}_p & -\mathbf{F}_\varepsilon & -\mathbf{F}_\lambda & \mathbf{I}-\mathbf{F}_f & -\mathbf{F}_{\bar{n}} \\
-\mathbf{T}_u & \mathbf{0} & -\mathbf{T}_\zeta & -\mathbf{T}_p & -\mathbf{T}_\varepsilon & -\mathbf{T}_\lambda & -\mathbf{T}_f & \mathbf{I}-\mathbf{T}_{\bar{n}}
\end{bmatrix}$$