Chapter 1
ADAMS/Solver and MSS

Introduction

This “Overview” introduces the computer program Automatic Dynamic Analysis of Mechanical Systems (ADAMS). Engineers use ADAMS/Solver in the field of Mechanical System Simulation (MSS) to simulate both large and small scale motion of mechanical systems. These motions are induced by the action of applied forces or excitations that act upon the system.

The purpose of this Overview is to help you use ADAMS/Solver to design and analyze mechanical systems from the most elementary pendulum, or four-bar mechanism, to the very complex suspension systems and other sophisticated mechanical devices. If you are a more experienced ADAMS/Solver user, you will learn about the most recent innovations in the code.

This Overview includes the following chapters and appendix:

1. ADAMS/Solver and MSS. Describes the role of ADAMS/Solver simulation code in the field of Mechanical Systems Simulation.
3. Modeling Process. Describes a sequence of steps to create an ADAMS/Solver model, run a simulation, and review the results.
4. ADAMS/Solver Statements. Defines the syntax and format of statements used to define input for an ADAMS/Solver simulation.
5. Customizing ADAMS/Solver. Describes customizing procedures to define nonstandard modeling elements or to generate nonstandard output.
6. Executing ADAMS/Solver. Discusses procedures to initiate an ADAMS/Solver simulation and to control the process.
7. Output Files. Summarizes the purpose and contents of various output files.

1.1 Definition of MSS

Mechanical System Simulation (MSS) is the study of the motion of mechanical systems caused by external forces and excitations that act on the system. MSS simulates large displacement motion, i.e., the extent of the relative motion of the components can be on the order of the overall dimensions of the system. The mechanical system consists of rigid and flexible parts interconnected by various kinds of joints and other constraining forces. The net reaction and applied forces drive the relative motion of the parts.

MSS software is of growing, fundamental importance to modern Mechanical Computer-Aided Engineering (MCAE). It allows you to evaluate the design characteristics and performance of a product prior to fabrication of the prototype. With MSS software, you can reduce product development costs, evaluate more design alternatives, and reduce the time it takes to bring a new product to market.
1.2 The Relationship of MSS to Other Technologies

The MSS software complements other programs used for Mechanical Computer-Aided Engineering. MSS can incorporate data from other software products into a complete simulation of system behavior. MSS also produces output that assists in further analysis with other computational tools.

As Figure 1 illustrates, MSS software exchanges data with other applications. A discussion of these applications follows Figure 1.

![Figure 1. MSS Software](image)

1. Computer-Aided Design (CAD) creates mechanical models for MSS and displays MSS simulation results.
2. Finite Element Analysis (FEA) computes mass, stiffness, and damping information to describe flexible bodies in MSS, and uses forces computed by MSS to determine system loads and boundary conditions.
3. Used with software that simulates multidomain actuators, MSS models systems that interact with the forces and power flow from electrical, hydraulic, pneumatic, and mechanical subsystems.
4. MSS uses the output from the modal analysis of an experimental device in a linearized component of an MSS model for a mechanical system.
5. System design packages can apply the techniques of classical and modern control theory to the analysis of system motion simulated with MSS software.
6. Optimization software evaluates system behavior predicted by MSS and adjusts system parameters to improve performance.

In summary, other software programs can use MSS output, provide data for an MSS analysis, or both. MSS plays an important role in integrating MCAE applications.
1.3 ADAMS/Solver as a Computational Tool for MSS

The capabilities of computer software have made advanced Mechanical System Simulation widely available. In particular, the ADAMS/Solver simulation code is an important computational tool in the field of Mechanical Computer-Aided Engineering (MCAE).

During a simulation, ADAMS/Solver describes the locations and orientations of all mechanical system parts in terms of six coordinates, three translational, and three angular. ADAMS/Solver stores time-dependent translational and angular displacements, velocities, and accelerations in the state vector. The state vector also contains current values of the reaction and applied forces on each of the parts at the constrained and inertial locations. Thus, the state vector provides a complete description of the mechanical system for a mechanical system simulation.

You can use the information that an ADAMS/Solver simulation generates to verify the following:

- System performance characteristics
- System response to a set of operating conditions

On the basis of the simulation results, you can adjust the design parameters to improve the performance of the system. A variety of industries use ADAMS/Solver as described below:

- The **automotive** industry uses ADAMS/Solver for the design and evaluation of
  - new and better suspension systems
  - ride and handling characteristics of passenger vehicles
  - anti-lock braking systems

- The **aerospace and aircraft** industries use ADAMS/Solver for the testing of
  virtual prototypes of
  - automatic control systems for maintaining satellite attitude
  - minimization techniques for the effects of the interaction between system controls and the mechanical structure
  - various other mechanisms such as landing gears and latching devices for aircraft doors

- The **general machinery** industry uses ADAMS/Solver to compute and analyze the design characteristics for
  - the power requirements for industrial appliances, such as an orange juice machine
  - the intricate mechanisms in cameras, video recorders, and computer disk drives
  - robotic systems
  - the high-speed switching mechanisms for electrical relays
Figures 2 and 3 are typical of specific applications for the ADAMS/Solver simulation code. Figure 2 illustrates a backhoe, while Figure 3 illustrates a satellite with solar panels.
Chapter 1. ADAMS/Solver and MSS

In Figure 2, ADAMS/Solver simulates part of the work cycle of a backhoe as it digs into the ground with the bucket, scoops up the dirt, and lifts the dirt. In response to articulation instructions from the hydraulic systems, ADAMS/Solver computes all of the reaction forces on the components of the equipment.

The scale for the motion of the backhoe is comparable to its overall dimensions. The excitations (movement of the hydraulic cylinders) and applied forces (interaction between the bucket and the ground) for the backhoe are highly nonlinear and dependent on the instantaneous state of the mechanical system. These are all typical characteristics of problems for mechanical system simulation.

In Figure 3, the gravitational magnetodynamic and aerodynamic forces and moments affect the satellite in orbit about the earth. Because of weight restrictions on a satellite, the solar panels are extremely flexible. ADAMS/Solver computes the effects of the panel flexibility and deformation on the satellite’s overall motion. This information is necessary to design and verify a control system to maintain the attitude of the principal axes of the spacecraft as it moves about the earth.

1.3.1 Input Specifications for the ADAMS/Solver Code

The physical laws of mechanical engineering are the basis for the field of MSS. The MSS simulation tools implement Newton’s laws of motion, or an equivalent formulation of the physical principles. ADAMS/Solver uses the system of Euler-Lagrange equations of motion. This system, composed of algebraic and second order differential equations, requires sophisticated numerical methods for their solution.

ADAMS/Solver defines the following to specify the mechanical model for a simulation:

- The inertial characteristics of the parts
- The relationships between parts
- The driving motions and forces for the system

The model can also include additional differential (first order) and algebraic equations coupled to, or independent of, the mechanical system.
The input data for an ADAMS/Solver simulation includes the following entities:

- The mass and inertia of the rigid bodies or parts
- The definition of the geometric aspects of the system including the centers of mass for the parts, the locations on the parts for the joints that hold the system together, and the points at which the specified motion functions and forces apply
- The connectivity for the system (the mechanisms for connecting the parts) defined in terms of mechanical joints, higher-pair contacts, other constraints and elastic elements
- A description of the external forces and excitations acting on the system
- The graphical attributes that enable animation and visualization of the system behavior

A library of standard joints and higher-pair contacts simplifies the task of defining the connections between parts of the system.

Each ADAMS/Solver simulation begins with an analysis of the input data verifying that the description of the mechanical system is

- Complete
- Correct
- Consistent

The verification process is thorough. A comprehensive set of messages will identify any errors that can occur in the input data. ADAMS/Solver assembles the differential and algebraic equations (DAEs) for the actual simulation of the problem only after the input data passes the above listed standards.
1.3.2 The Analysis Modes

ADAMS/Solver can perform six different types of analyses depending on the characteristics of the problem and the interests that you have. Each of these analysis modes is useful in the appropriate context. As described in the following paragraphs, the various analyses provide different information about the system.

1. ADAMS/Solver performs an initial conditions analysis (the assembly process) prior to the static, quasi-static, and dynamic analyses. After processing the input data and before a simulation begins, ADAMS/Solver requires a consistent set of values for the state of the system. That is, the displacements and velocities have to satisfy the constraint equations that define the system. Mechanical models frequently involve closed-loop systems that are difficult to define as illustrated in Figure 4.

![Figure 4. Four Bar Mechanism](image-url)

In systems such as this, where M represents applied motion, the state variables are not all independent. The constraint equations that define the geometric configuration of the system also define dependencies between the state variables. If the displacement and velocity input values are not sufficiently accurate, they may not satisfy the constraints in the closed-loop. Therefore, ADAMS/Solver may not recognize some of the system velocities and forces.

In complex systems, the initial values that you specify for displacements and velocities often violate the constraint equations. The initial conditions analysis modifies the input data for the displacements and velocities as needed to satisfy the system constraints while minimizing the change in the data. Often, it is convenient to specify only a few initial displacements and velocities exactly, and allow the initial conditions analysis to assemble the rest of the system properly.
2. A **kinematic analysis** simulates the motion of the system. It allows the engineer to determine the range of possible values for the displacement, velocity, and acceleration of any point of interest on a mechanical device. If you specify the mass and inertial properties of a part, ADAMS/Solver also calculates the corresponding applied and reaction forces required to generate the prescribed motions.

The kinematic analysis is algebraic in nature. The implementation in ADAMS/Solver is very fast and accurate. Typical applications include designing a mechanism for the transfer of motion through a linkage between parts and the preliminary examination of a complicated model intended for subsequent dynamic analyses.

3. The **static equilibrium analysis** determines a state for the system to balance all internal and external forces in the absence of any system motions or inertial forces. All system velocities and accelerations are set to zero.

The analysis for static equilibrium is also algebraic. You often find a starting point for a dynamic analysis by using a static analysis to remove unwanted system transients at the start of the simulation. Unbalanced forces in the initial configuration can generate undesirable effects in the dynamic analysis.

4. A **quasi-static analysis** is a sequence of static analyses performed for different configurations of the system, typically, at fixed time intervals throughout a prescribed motion for the system.

5. The most complex and computational demanding mode in ADAMS/Solver is the **dynamic analysis**. Several different integrators are available for finding the solution to the full system of differential and algebraic equations (DAE’s).

The dynamic analysis provides the time-history solution for all of the displacements, velocities, accelerations, and internal reaction forces in a mechanical system driven by a set of external forces and excitations.

6. In the **linear analysis** mode, ADAMS/Solver linearizes the system of nonlinear equations for the model about an operating point. This results in a set of linear time-invariant (LTI) equations in the form

\[ M \ddot{x} + C \dot{x} + Kx = F \]

where \( x \) represents a perturbation of the state from its operating point; \( M, C, \) and \( K \) are the equivalent mass, damping, and stiffness matrices, respectively; and \( F \) is the column matrix of forcing terms.

After ADAMS/Solver computes the various matrices, it performs two different types of linear analyses: eigenanalysis and the generation of state matrices. The eigenanalysis includes the evaluation of both eigenvalues and eigenvectors for the linearized system.
The eigenvalues are the natural frequencies of the system while the eigenvectors represent the modes of the motion associated with each frequency. Generally, both the eigenvalues and the eigenvectors are complex valued.

The state matrices that ADAMS/Solver generates from the linearized system are the coefficients for a representation of the mechanical system as a multiple-input, multiple-output transfer function in the form

\[
\begin{align*}
\dot{x} &= Ax + Bu \\
y &= Cx + Du
\end{align*}
\]

where \( u \) is the set of inputs and \( y \) is the set of outputs. The coefficients \( A, B, C, \) and \( D \) are the state matrices for the mechanical system.

1.3.3 The Output from ADAMS/Solver

At each output time, ADAMS/Solver writes a comprehensive description of the state of the system. Thus, a time history can be recorded in the output files. The output includes any combination of the following items if you specify it before running the simulation:

- Displacements
- Velocities
- Accelerations
- Reaction forces
- Applied forces
- User-defined variables
- User-defined derived results
- States for system modeling elements
- Outputs from system modeling elements (such as linear transfer functions)
- Plant inputs and outputs for the state matrices for a linearized ADAMS/Solver model
- State matrices corresponding to a set of plant inputs and outputs for a linearized ADAMS/Solver model
- Eigenvalues and eigenvectors at specified operating points

ADAMS/Solver writes the information into a number of output files as described in more detail in Chapter 7, “Output Files.”
Chapter 2
Modeling Concepts

Introduction
This chapter is devoted to presenting conceptual material that is unique to MSS and rigid body dynamics. The concepts in this chapter


2. Coordinate Systems. Defines the difference between coordinate systems and reference frames, and presents the different types of coordinate systems available in ADAMS/Solver.

3. Rotation Sequences. Explains why finite rotation sequences are not vector quantities, why infinitesimal rotation sequences can be represented as vector quantities, and representation of orientations, in general.

4. Location and Orientation Methods. Describes two methods available in ADAMS/Solver for locating and orienting coordinate systems.

5. Global versus Local Geometric Data. Highlights the differences between inputting data relative to the global coordinate system and inputting data relative to LPRFs.

6. Degrees of Freedom. Introduces the concept of degrees of freedom in a mechanical system, the redundant constraint handling and overconstraint checking in ADAMS/Solver, the concept of loops in a mechanical system, the Gruebler equation for calculating the degrees of freedom, and the topological data ADAMS/Solver can generate.

7. Force Direction and Application. Discusses how ADAMS/Solver directs and applies forces.
2.1 Reference Frame

A reference frame specifies a reference for the calculation of velocities and accelerations of a system.

2.1.1 Ground Reference Frame

The ground reference frame by definition is the single Newtonian inertial reference that establishes “absolute rest.” By definition, any point belonging to the ground reference frame has zero velocity and acceleration. It is fixed in “absolute space.”

ADAMS/Solver expresses Newton’s Laws in the ground reference frame. For many applications, you can consider the Earth as a Newtonian reference frame, even though it travels in an elliptical path about the sun (and is therefore accelerating), and rotates on its own axis.

2.1.2 Part Reference Frames

Every rigid body has one reference frame called the part reference frame. You can use the terms “rigid body” and “part reference frames” synonymously.
2.2 Coordinate Systems

A coordinate system is essentially a measuring stick used to define kinematic and dynamic quantities. Coordinate systems usually consist of an origin and a set of three mutually orthogonal unit vectors that specify three directions. Associated with each of these directions is a “scale” that permits measurement of quantities in the coordinate system. You can specify all kinematic and kinetic vector quantities as projections of the vector along the three unit vectors $\hat{i}$, $\hat{j}$, and $\hat{k}$, as shown in Figure 5.

Figure 5. A Coordinate System

Figure 5 illustrates a vector $\mathbf{R}$ that is defined as the displacement of point $P$ in the coordinate system $\mathcal{O}XYZ$. $R_x$, $R_y$, and $R_z$ are the measure numbers of $\mathbf{R}$ along the $x$, $y$, and $z$ axes of the coordinate system.

Coordinate systems are useful for specifying a model configuration or its velocities and accelerations. You can fix coordinate systems to a reference frame or move them with respect to a reference frame.

ADAMS/Solver uses three types of coordinate systems. They are

- Ground
Chapter 2. Modeling Concepts

- Local part reference frame
- Marker
2.2.1 Ground Coordinate System

The ground coordinate system rigidly attaches to the ground reference frame or the ground part. In ADAMS/Solver, you specify the locations, orientations, and velocities of all other rigid bodies or “parts” in the ground coordinate system.

2.2.2 Local Part Reference Frame

The Local Part Reference Frame (LPRF) is actually a coordinate system. Every part in ADAMS/Solver has a single LPRF. You can specify the initial location and orientation of a part in ADAMS/Solver by specifying the location and orientation of the LPRF with respect to the ground coordinate system. The LPRF moves with its part. A part’s LPRF defaults to the ground coordinate system whenever you do not specify a LPRF.

2.2.3 Markers

A marker is a coordinate system owned by a part. There are two types of markers in ADAMS/Solver: fixed and floating.

2.2.3.1 Fixed Marker

A fixed marker attaches to a part and moves with that part. To place a marker on a part, you must specify its location and orientation with respect to the LPRF of the part. Markers have many different functions in ADAMS/Solver. They can define the graphical periphery of a part, locate the part center of mass, provide sites for reaction force application and joint attachments, or perform other location and orientation functions. Certain joints, such as ball-and-socket joints, use the origin of a marker to locate their centers of action on the parts. Other joints, such as cylindrical joints, use the z-axes of markers for orienting the translational and/or rotational freedom of motion for the parts.

2.2.3.2 Floating Marker

Some forces and constraints in ADAMS/Solver use floating markers to specify sites for applying forces on parts. You do not specify a fixed position for a floating marker. The force or constraint dictates the position and orientation of the floating marker. This position and orientation can change during the simulation.
2.3 Rotation Sequences

Body-fixed 3-1-3 (Euler angles or z-x-z) rotation sequences are one way ADAMS/Solver allows you to specify part and marker orientations. By default, ADAMS/Solver reports orientations using body-fixed 3-1-3 rotation sequences. Consequently, this section discusses some of the general characteristics of both rotation sequences and orientation methods.

2.3.1 Finite Rotation Sequences are Not Vector Quantities

Kinematics is the study of motions in a system. The displacements, velocities, and accelerations are the variables of interest. Most of these variables are vector quantities. Thus, it is tempting to try to represent rotation sequences with vectors. Certainly, you can assign a direction (the axis of rotation) and a magnitude (the angle of rotation). As the following analysis shows, however, a finite rotation sequence is not a vector.

Suppose $\alpha$ and $\beta$ are two “rotational vectors” associated with two different axes of rotation and with transformation matrices $A$ and $B$. To qualify as elements of a vector space, the rotations $\alpha$ and $\beta$ would have to be commutative. The addition of two consecutive rotations corresponds to the product of the two transformation matrices. $A \cdot B \neq B \cdot A$, and hence $\alpha$ and $\beta$ are not commutative in addition, and therefore are not vectors.

2.3.2 Infinitesimal Rotation Sequences Can Be Represented as Vector Quantities

Let $A$ and $B$ be two transformation matrices corresponding to a sequence of two infinitesimal rotations, that is

\[ A = I + e_1 \]

where $e_1$ is the change in the transformation matrix due to the first infinitesimal rotation.

\[ B = I + e_2 \]

where $e_2$ is the change in the transformation matrix due to the second infinitesimal rotation.

The addition of two rotations corresponds to the multiplication of the corresponding transformation matrices, that is

\[ AB = I + e_1 + e_2 + e_1 e_2. \]

Ignoring higher order infinitesimals, one gets:

\[ AB = I + e_1 + e_2 = I + e_2 + e_1 = BA. \]

Infinitesimal rotations are therefore commutative in addition and can be represented as vectors.
ADAMS/Solver sometimes assumes infinitesimal rotations. For more information, see the discussions of angles under the BEAM, BUSHING, and FIELD statements (Sections 2.5.2, 2.5.3, and 2.5.4, respectively) in the ADAMS/Solver Reference Manual.

2.3.3 Representation of Orientations

A body-fixed 3-1-3 (Euler angle or z-x-z) rotation sequence is one of many ways for representing orientations. In general, the complete representation of an orientation includes the following:

- The axis about which each rotation takes place
- The angle of rotation about each axis
- The sequence of rotations

A body-fixed 3-1-3 rotation sequence defines each of these. Following is a list of other methods you can use to represent three-dimensional rotations and the number of values each requires.

- Direction cosines matrix or transformation matrix (9 values)
- Euler Parameters (4 values)
- Rodrigues Parameters (3 values)

For inputting orientations, ADAMS/Solver permits the use of direction cosines (the x-point and z-point method) and body-fixed 3-1-3 rotation sequences (the Euler angles method). ADAMS/Solver permits numerous methods for outputting orientations. For more information on the numerous methods available for outputting orientation information, see Section 3.4.9, “RCNVRT” in the ADAMS/Solver Subroutines Reference Manual.
2.4 Location and Orientation Methods

For locating and orienting any part with respect to the ground coordinate system or any marker with respect to its LPRF, you can use one of three methods:

- Euler angles method
- X-point-z-point method (also known as the direction cosines method)

Since you use these methods to locate and orient both LPRFs and markers, we refer to the coordinate systems that you locate and orient as “positioned frames” and to the coordinate systems with respect to which you locate and orient them as “base frames.” For example, if you orient a part with respect to the ground coordinate system, the part is the positioned frame and the ground coordinate system is the base frame. Similarly, if you orient a marker with respect to a LPRF, the marker is the positioned frame, and the LPRF is the base frame.

2.4.1 Euler Angles Method

When you use the Euler angles method to locate and orient a positioned frame with respect to a base frame, specify the following geometric data:

- Cartesian coordinates to define the origin of the positioned frame with respect to the base frame
- Body-fixed 3-1-3 Euler angles to define the orientation of the positioned frame with respect to the base frame.

Since Euler angle rotations cumulatively define the orientation of the positioned frame, you may find it difficult to envision orientations that involve rotations of other than 0-, 45-, or 90- degree increments. When orienting a reference frame that requires a complex combination of rotations, you may find it more convenient to use the x-point-z-point orientation method (see Section 2.4.2, “X-Point-Z-Point Method”).
2.4.2 X-Point-Z-Point Method

You can use the x-point-z-point method to specify the location of the positioned frame relative to the base frame and the orientation (in direction cosines) of the positioned frame relative to the base frame. The x point defines the x-axis or the first column of the direction cosines matrix, and the z point defines the z-axis or the third column of the direction cosines matrix. ADAMS/Solver calculates the y-axis or the second column using the orthogonal properties of transformation matrices.

When you use the x-point-z-point method to locate and orient a positioned frame, provide the following geometric data with respect to the base frame:

- Cartesian coordinates of the positioned frame origin (point $Q$)
- Cartesian coordinates of a point on the z-axis of the positioned frame (point $Z$) or of a point on the x-axis of the positioned frame (point $X$). Provide a point on the z-axis when it is important to exactly specify the z-axis of the positioned frame. Provide the Cartesian coordinates of a point on the x-axis when it is important to exactly specify the x-axis of the positioned frame.
- Cartesian coordinates of a point that is in the x-z plane of the positioned frame. If you provided a point on the z-axis of the positioned frame (point $Z$), the point in the x-z plane should be noncollinear with points $Q$ and $Z$ (that is, provide point $X$). If you provided a point on the x-axis of the positioned frame (point $X$), the point in the x-z plane should be noncollinear with points $Q$ and $X$ (that is, provide point $Z$).

For orienting a marker with respect to a LPRF, ADAMS/Solver refers to the three points as $QP$, $ZP$, and $XP$, respectively. For orienting a LPRF with respect to the ground reference frame, ADAMS/Solver refers to the three points as $QG$, $ZG$, and $XG$, respectively. Figure 6 shows the $Q$, $Z$, and $X$ points and their coordinates with respect to the base frame. If vectors $QZ$ and $QX$ are not orthogonal, ADAMS/Solver redirects either $QX$ or $QZ$ so that they are orthogonal. Note that in Figure 6, a point on the z-axis of the positioned frame specifies exactly the z-axis of the frame.
By default, ADAMS/Solver assumes that \( Q \) defines the exact location of the frame origin and that \( Z \) defines an exact axis orientation. Because the third point (\( X \) by default) does not necessarily lie on an axis, ADAMS/Solver determines the vector cross product of vectors \( QZ \) and \( QX \) to form an orthogonal \( y \)-axis. Then, ADAMS/Solver determines the vector cross product of the new \( y \)-axis and the exact axis (the \( z \)-axis by default) to produce the orientation of the remaining axis.
2.5 Global Versus Local Geometric Data

Regardless of the combination of methods that you use to locate and orient markers and LPRFs, the location and orientation of every marker with respect to ground is the summation of both marker and LPRF geometric data. This summation allows you to input points from a series of part drawings in which each drawing has its own LPRF. The geometric data you input to describe the location and the orientation of each LPRF with respect to the ground frame relates the LPRFs to one another.

Although the summation of the geometric data allows you to input data from a series of part drawings with different LPRFs, it can complicate the process of superimposing markers in global space and the process of assigning parallel or perpendicular orientations to their axes. Superposition of markers is sometimes necessary for defining joints. For example, if you define a ball-and-socket joint, you must superimpose a marker origin at the center of the ball with a marker origin at the center of the socket (superimpose one marker with another). Similarly, to allow for the proper assembly of the mechanism, certain joints require that the axis of a marker in one part is parallel or perpendicular to the axis of a marker in the other part.

If two markers do not have the correct origin locations or axis orientations, ADAMS/Solver iteratively repositions the LPRFs to bring the two markers to the correct location or orientation before continuing the analysis. For two markers whose locations or orientations are reasonably close to what they should be, these iterations usually do not seriously affect the solution process. For two markers whose locations or orientations are not reasonably close, ADAMS/Solver cannot enforce the proper positioning or alignment during the initial phases of the simulation and will be unable to solve for the behavior of the model. In other cases, ADAMS/Solver may find an initial assembly position different from the intended one, thus leading to a different solution. Consequently, when you are defining some markers, you may have to carefully specify their geometric data and the data for their LPRFs.
Not all models describe the geometry of a system in the foregoing manner. For instance, in the automotive industry, it is common practice to dimension all of the parts in a single and often large drawing of the assembly. Since ADAMS/Solver does not require the LPRFs to lie within the boundaries of the parts to which they belong, it is possible to superimpose all of the LPRFs on the ground coordinate system. When you do this, the initial marker geometric data with respect to the LPRFs is the same as the geometric data with respect to the ground coordinate system. Under these circumstances, the LPRF location and orientation coordinates are all zero. Because they default to zero, you reduce the amount of geometric data that you must provide.

Additionally, even though you input marker geometric data with respect to their different LPRFs, all markers occupying the same location and orientation with respect to the ground coordinate system have identical data. In general, this approach greatly simplifies the summation process for superimposing marker locations or for coordinating marker orientations. When using this approach, keep in mind that ADAMS/Solver superimposes the disparate LPRFs on the ground coordinate system only at time zero. As the mechanism articulates during a subsequent dynamic, kinematic, or quasi-static equilibrium analysis or settles during a subsequent static equilibrium analysis, the LPRFs move with their parts in the stationary ground coordinate system.
2.6 Degrees of Freedom

In mechanical systems, a “degree of freedom” represents a possible motion that is inherent in or characteristic of the system. Therefore, the total number of degrees of freedom of a mechanical system is the number of independent motions that characterize the system.

A freely floating rigid body in 3-D space is said to have six degrees of freedom. This implies that the rigid body can exhibit motion in six independent ways: three translations and three rotations. The degrees of freedom of a system represent the minimum number of displacement coordinates needed to completely specify the system configuration. Once you know these, you can calculate any other information pertinent to the configuration as a function of these “independent” variables.

You can, of course, represent a system with more coordinates than there are degrees of freedom. In such an instance, the coordinates are not all independent. There must be algebraic constraint equations relating some of the coordinates.

For most systems, the number of degrees of freedom is constant. In some systems, the number of degrees of freedom can change as their configurations change.

ADAMS/Solver allows you to specify the position and orientation of each rigid part in the system, regardless of the degrees of freedom in the system. Different types of joints and joint primitives constrain different combinations of motion. Revolute joints, for example, constrain two degrees of rotational freedom and three degrees of translational freedom, whereas cylindrical joints constrain two degrees of rotational freedom and only two degrees of translational freedom. Prescribed motions, defined at joints, constrain one degree of freedom.

The total number of degrees of freedom in a model is therefore equal to the difference between the number of coordinates used to represent the system and the number of active constraints due to joints, motions, and other constraints.
2.6.1 Redundant Constraints and Overconstraint Checking

It is quite possible to construct “legal” and well defined systems where one set of joints can constrain the system in exactly the same way as another set of joints. In mathematical terms, one can state that the equations of constraint due to one set of joints are “redundant” with those of the second set of joints.

An ADAMS/Solver model is a mathematical idealization of the physical system. For this reason, your model can contain redundant constraints if you define your system with the same number and types of joints as the physical system.

An example of a system with redundant constraints is a door supported by two hinges. In a real door, minor violations of the hinge colinearity do not prevent the door from operating because of the deformation of the parts, and what is more important, joints play inherent in the door-hinge connection. In the mathematical model, where parts are rigid and joints do not permit any slop, it is said that one of the two hinges is redundant but “consistent” whenever the axes of the two hinges are collinear. If, however, the axes of the two hinges are not collinear, the door cannot move without “breaking” one of the hinges. In this case, the two hinges are inconsistent and half of their constraints are redundant.

ADAMS/Solver arbitrarily determines which constraints are redundant, “deletes” them from the set of equations, and provides a set of results that characterize the motion and forces in the system. Note that other solutions can also be physically realistic. Systems with redundant constraints do not have a “unique” solution.

An ADAMS/Solver model with fewer than zero degrees of freedom is overconstrained. ADAMS/Solver can simulate an overconstrained model only if the redundant constraints are consistent. Redundant constraints are consistent if a solution satisfying the set of independent constraint equations also satisfies the set of redundant constraint equations.

In the case of the door hinge, ADAMS/Solver ignores all of the equations corresponding to one of the hinges. This means that all the reaction forces are concentrated at one hinge in the ADAMS/Solver solution. ADAMS/Solver arbitrarily sets the reaction forces to zero at the other hinge. Any other distribution of reaction forces that maintains a constancy of the net reaction force and torque is also a correct solution. Therefore, you should not intentionally input redundant constraints in your model. You may also wish to consider replacing redundant constraint elements with approximately equivalent flexible connections.
ADAMS/Solver does not always check joint initial conditions when it does overconstraint checking. If you apply a motion on one joint and initial conditions on another joint, check to ensure that they are not redundant because ADAMS/Solver does not check them for redundancy. As a general rule, do not specify more initial conditions as being exact than the model has degrees of freedom.

For a mechanism with redundant constraints, constraints that are initially consistent can become inconsistent as the mechanism articulates. ADAMS/Solver stops execution as soon as the redundant constraints become inconsistent. For example, consider a planetary gear system with redundant constraints. Slight misalignment errors can accumulate over time, eventually resulting in a failure of the consistency check. If this occurs, manually remove the redundant constraints from the data set, or replace them with flexible connections.

### 2.6.2 Loops

Some, but not all, models include one or more series of parts and constraints that constitute loops. ADAMS/Solver checks all loops and handles loops with redundant constraints similarly to the way it handles models with redundant constraints.

Always check your model for loops before you execute ADAMS/Solver. Although the code can handle loops with redundant constraints, ADAMS/Solver usually runs faster if your model does not contain overconstrained loops.

Loops can be either independent or dependent. Independent loops are loops not entirely created from other loops. Dependent loops are loops entirely created from other loops. The number of independent loops and the number of dependent loops in a particular mechanical system is a system characteristic. In a system with both independent and dependent loops, ADAMS/Solver may classify any individual loop as either independent or dependent, depending upon the classification of the other loops. Figure 7 shows a mechanism with three loops (A, B, and C). Two are independent loops and the remaining loop is a dependent loop.

You can consider any two loops in Figure 7 as independent; the third is automatically a dependent loop.
2.6.3 The Gruebler Equation for Calculating Degrees of Freedom

To determine the degrees of freedom (DOF) for a model or for one of its loops, use the Gruebler equation:

\[
\text{DOF} = [6^*(\text{Number of Parts}-1)] - (\text{Number of Active Constraints})
\]

To evaluate this equation, count the parts and the constraints in the model. Note that the part count includes the ground part. Because the ground part does not contribute any degrees of freedom to the model, the Gruebler equation subtracts one from the total number of parts. Use the chart in Table 1 to count the constraints in your system. This chart lists the system elements that impose constraints and the number of translational, rotational, and total constraints each element imposes.

---

Figure 7. Schematic of a Model with Three Loops

= ground
= a revolute (hinge) joint
= a universal (Hooke’s) joint
= a spherical (ball-and-socket) joint
= a cylindrical joint
10, 11, 12, 13, 14, 15, 16 = joints
101, 102, 103, 104, 105 = parts
Chapter 2. Modeling Concepts

Table 1. Constraints Imposed by System Elements

<table>
<thead>
<tr>
<th>ADAMS Statements</th>
<th>Translational Constraints</th>
<th>Rotational Constraints</th>
<th>Mixed Translational &amp; Rotational Constraints</th>
<th>New Generalized Coordinates</th>
<th>Total Number of DOFs Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUPLER</td>
<td>---</td>
<td>---</td>
<td>1</td>
<td>---</td>
<td>1</td>
</tr>
<tr>
<td>CVCV</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>GEAR</td>
<td>---</td>
<td>---</td>
<td>1</td>
<td>---</td>
<td>1</td>
</tr>
<tr>
<td>JOINT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONVEL</td>
<td>3</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>4</td>
</tr>
<tr>
<td>CYLINDRICAL</td>
<td>2</td>
<td>2</td>
<td>---</td>
<td>---</td>
<td>4</td>
</tr>
<tr>
<td>FIXED</td>
<td>3</td>
<td>3</td>
<td>---</td>
<td>---</td>
<td>6</td>
</tr>
<tr>
<td>HOOKE</td>
<td>3</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>4</td>
</tr>
<tr>
<td>PLANAR</td>
<td>1</td>
<td>2</td>
<td>---</td>
<td>---</td>
<td>3</td>
</tr>
<tr>
<td>RACKPIN</td>
<td>---</td>
<td>---</td>
<td>1</td>
<td>---</td>
<td>1</td>
</tr>
<tr>
<td>REVOLUTE</td>
<td>3</td>
<td>2</td>
<td>---</td>
<td>---</td>
<td>5</td>
</tr>
<tr>
<td>SCREW</td>
<td>---</td>
<td>---</td>
<td>1</td>
<td>---</td>
<td>1</td>
</tr>
<tr>
<td>SPHERICAL</td>
<td>3</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>3</td>
</tr>
<tr>
<td>TRANSLATION-AL</td>
<td>2</td>
<td>3</td>
<td>---</td>
<td>---</td>
<td>5</td>
</tr>
<tr>
<td>UNIVERSAL</td>
<td>3</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>4</td>
</tr>
<tr>
<td>JPRIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATPOINT</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>---</td>
<td>3</td>
</tr>
<tr>
<td>INLINE</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>---</td>
<td>2</td>
</tr>
<tr>
<td>INPLANE</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>---</td>
<td>1</td>
</tr>
<tr>
<td>ORIENTATION</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>---</td>
<td>3</td>
</tr>
<tr>
<td>PARALLEL</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>---</td>
<td>2</td>
</tr>
<tr>
<td>PERPENDICULAR</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>---</td>
<td>1</td>
</tr>
<tr>
<td>MOTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSLATION-AL</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>---</td>
<td>1</td>
</tr>
<tr>
<td>ROTATIONAL</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>---</td>
<td>1</td>
</tr>
<tr>
<td>PTCV</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>UCON</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>---</td>
<td>1</td>
</tr>
</tbody>
</table>

Use the information in Table 1 to write the Gruebler equation for Figure 7.
Chapter 2. Modeling Concepts

The model in Figure 7 has six parts (including ground) and twenty-nine constraints: two revolute joints of five constraints each, two cylindrical joints of four constraints each, two universal joints of four constraints each, and one spherical joint of three constraints. The Gruebler equation for Figure 7 is

\[
\text{DOF} = [6 \times (6 - 1)] - (5 + 5 + 4 + 4 + 4 + 3) \\
= 30 - 29 \\
= 1.
\]

The system has one degree of freedom. Even though ADAMS/Solver can detect and ignore redundant constraints in models, you should calculate the degrees of freedom in each loop. Loop A in Figure 7 has two revolute joints, one cylindrical joint, and one universal joint. The equation for this loop is

\[
\text{DOF} = [6 \times (4 - 1)] - (5 + 5 + 4 + 4) \\
= 18 - 18 \\
= 0.
\]

Loop A has zero degrees of freedom.

Loop B in Figure 7 has one revolute joint, one cylindrical joint, one universal joint, and one spherical joint. The equation for this loop is

\[
\text{DOF} = [6 \times (4 - 1)] - (5 + 4 + 4 + 3) \\
= 18 - 16 \\
= 2.
\]

Loop B has two degrees of freedom.

Loop C in Figure 7 has one revolute joint, two universal joints, two cylindrical joints, and one spherical joint. The equation for this loop is

\[
\text{DOF} = [6 \times (6 - 1)] - (5 + 4 + 4 + 4 + 3) \\
= 30 - 24 \\
= 6.
\]

Loop C has six degrees of freedom. None of the loops in this system has negative degrees of freedom. In other words, the Gruebler count does not indicate overconstrained loops. Note that the Gruebler count for the entire system may not be the sum of the Gruebler counts of the loops.
Try writing the Gruebler equation for the system schematic in Figure 7. Because the schematic has one revolute joint, one translational joint, one universal joint, one spherical joint, and one imparted motion, the equation you write should be

\[
\text{DOF} = \left[ 6 \ast (4 - 1) \right] - (5 + 5 + 4 + 3 + 1) = 18 - 1 = 0.
\]

The schematic in Figure 7 has no loops, so it does not require additional Gruebler equations.

### 2.6.4 Topological Data

You can use the topological data that ADAMS/Solver generates to graph the connections between parts and constraints in the model. This alerts you to the existence of overconstrained loops and possibly to loops that define “structures” in your model. Although an understanding of this topological data helps you find constraint errors in your model, it is not essential to understanding ADAMS/Solver.

The topology output includes the following information:

1. Number of branches — Branches represent kinematic connections (that is, joints, joint primitives, gears, point-to-curve, curve-to-curve, and motions).
2. Number of vertices — Vertices represent parts.
3. Graph connectivity data — Graph connectivity data identifies each branch and the vertices that the branches connect. The branch identifiers are the ADAMS/Solver constraint statement identifiers. The vertex identifiers are the ADAMS/Solver part statement identifiers. From this data, you can construct a complete graph of the loop.
4. System type classification — This classifies the mechanical system that the ADAMS/Solver model defines as one of four types and reports the type. Those types are as follows:
   - Type 0 — These systems contain vertices but no branches, that is, parts but no constraints.
   - Type 1 — These systems contain both vertices and branches, but the branches do not form loops.
   - Type 2 — These systems contain both vertices and branches, and the branches form simple loops. Simple loops contain only branches unique to that loop and do not share branches with other loops.
   - Type 3 — These systems contain both vertices and branches, and the branches form more than one loop, including at least two complex loops. Complex loops share at least one branch with one or more other loops.
5. Number of independent and dependent loops — A simple loop is always an independent loop. A complex loop can be either an independent or a dependent loop. Although both types 2 and 3 systems contain independent loops, only type 3 systems can contain dependent loops. Figure 8 shows a graph of the mechanism in Figure 7.

![Figure 8. Graph of a Model with Three Loops](image)

6. Unusual loop configurations — Two unusual loop configurations can occur in ADAMS/Solver models. These two configurations generally represent unusual mechanical structures. In addition, loops with these configurations are frequently immobile. The topology data includes a warning when your model includes loops with either of these configurations. The two configurations are as follows:
   - Triangular loop configuration
   - Configuration with more than one branch between two vertices (parallel branches)

In certain exceptional cases, mechanical systems can include loops with these configurations and the loops can even be mobile. For example, a gear between two parts produces a triangular loop configuration that is mobile. Two or more joint primitives between two parts produce parallel branches between vertices. A cylindrical joint with a screw joint would also result in parallel branches. Again, the loop containing this construction can be mobile. For the purpose of counting loops, ADAMS/Solver counts parallel branches as a single branch. If your model is not running correctly, always check for unusual loop configurations.
2.7 Force Direction and Application

You can define any force in terms of its magnitude and its direction. Detailed descriptions of how ADAMS/Solver determines the magnitudes of each force type are in Section 2.5 and 3.4 of the ADAMS/Solver Reference Manual. This section discusses how ADAMS/Solver directs and applies single-component forces (SFORCE) and multi-component forces (VFORCE, VTORQUE, and GFORCE).

2.7.1 Translational, Action-Reaction, Single-Component Forces

For translational, single-component, action-reaction forces, you provide ADAMS/Solver with two markers. These markers specify the line along which the instantaneous force acts. If the force is positive, the markers experience a repelling force. If the force is negative, the markers experience an attracting force.

2.7.2 Other Single-Component Forces

For both translational, action-only, single-component and rotational single-component forces, you provide ADAMS/Solver with two markers. One marker declares the point of application for the force. The z-axis of the other marker specifies the direction of the force. ADAMS/Solver evaluates the signed magnitude and applies it to the marker that indicates the point of application. If the force is positive and translational, it acts in the positive direction along the z-axis of the marker that indicates the direction of the force. If the force is positive and rotational (a torque), it acts in the positive direction about the z-axis of the marker that indicates the direction of the force. The right-hand rule defines the positive direction.

2.7.3 Multicomponent Forces

ADAMS/Solver allows you to define translational and rotational force vectors at one marker or between two or more markers (VFORCE, VTORQUE, and GFORCE). These vectors may be dependent on the instantaneous state of the system. Specify the x, y, z measure numbers to define them in a marker coordinate system. The axes of a marker coordinate system define the directions of the vector components. If a vector component is positive and translational, it acts in the positive direction along the corresponding axis of the marker coordinate system. If a vector component is positive and rotational, it acts in the positive direction about the corresponding axis of the marker coordinate system. You must specify all three components of each multicomponent force.
Chapter 3
Modeling Process

Introduction
The process of analyzing a mechanical system with ADAMS/Solver consists of the following seven steps:

1. Creating an idealized model from the physical model
2. Decomposing the model into basic components
3. Drawing a system schematic
4. Selecting a system of units
5. Constructing the system equations
6. Solving the differential and algebraic equations
7. Reviewing the system response

Complete the first four steps before you execute an ADAMS/Solver simulation. After ADAMS/Solver constructs the system equations and solves them, you can examine the results. The sections in this chapter cover these steps in detail.
3.1 Creating an Idealized Model from the Physical Model

The modeling process begins with the idealization of a real mechanism as a system of basic components. For example, under certain circumstances, you might replace an input motion, from a complex system of gears, with a revolute joint and an applied motion. The idealization of the model, that is, the choice of simplifying assumptions, can have a significant effect on the usefulness and validity of the results and on the cost of writing and maintaining the input files for ADAMS/Solver. A high-quality idealization of the physical model, therefore, is very important.

The idealization of a physical model has two steps. First, you define the purpose of the model. Second, you simplify the model to remove unnecessary complexity.

The desired results define the purpose of the model. You should decide on the physical behavior of interest, the simulations you want to perform with the model, the output you need, and the required degree of accuracy. To help define the purpose of the model, there are several key questions you can ask yourself:

- Am I interested in the overall motion of the system, in the static configuration, in the dynamic loads, or in other behavior?
- Will I use the same model for more than one analysis or with different initial conditions or forcing functions?
- What output will I need to analyze the results?
- How will the accuracy of the solution affect the usefulness of my results?

The answers to these questions determine the second part of the idealization process, namely the choice of simplifications for the model.

In many ways, model simplification is a creative process to which you can apply both ingenuity and experience. The simplification process requires a careful examination of the model to determine the significant versus the negligible characteristics, the unnecessary degrees of freedom, and the symmetric and redundant subsystems. The simplification of the model can also require some assumptions or approximations. You may have to rely on sound engineering judgment when portions of the required input data are missing or incomplete.
3.2 Decomposing the Model into Basic Components

After you have defined the purpose of the model and have determined the necessary degree of complexity, you are ready to decompose the model into basic components. Decomposition allows further simplification of the model and encourages definition of the model in a form that can be input to ADAMS/Solver. ADAMS/Solver accepts four basic system components: parts, constraints, forces, and user-defined algebraic and differential equations.

3.2.1 Parts

The term “part,” as defined in ADAMS/Solver, refers to any rigid body or lumped mass. This definition of the term is very different from our day-to-day use of it. That is, “part” generally refers to a wide range of things, including beams, shock absorbers, and bushings. In ADAMS/Solver, a rigid beam is a part, but a flexible beam is a set of lumped masses (that is, parts) connected with forces. Although you might generally think of a shock absorber or a bushing with negligible mass as a part, in ADAMS/Solver you would model such a shock or bushing with only force components.

For each part you define, ADAMS/Solver writes six first-order dynamical equations (relating forces to accelerations) and six first-order kinematical equations (relating positions to velocities).

3.2.2 Constraints

The term “constraint,” as defined in ADAMS/Solver, refers to any of a number of elements that restrict the movement of a part or parts. For each scalar, configuration constraint you define, ADAMS/Solver writes a single algebraic constraint equation. The algebraic equations ADAMS/Solver writes for configuration constraints assure that the configuration is not violated (that is, does not come apart) during the course of the simulation.

ADAMS/Solver offers a wide variety of constraints. These include time-dependent and time-independent constraints, holonomic and non-holonomic constraints, as well as lower-pair and higher-pair contacts. Because ADAMS/Solver includes both joints and joint primitives, you can form any lower-pair contact with ADAMS/Solver. Certain higher-pair contacts such as point-curve constraints and curve-curve constraints are readily definable with corresponding ADAMS/Solver statements. You can implement other types of higher-pair constraints with user-written subroutines. For more information on the constraints available in ADAMS/Solver, see the discussion of constraints in Section 4.4.4, “Constraints.”
3.2.3 Applied Forces

The term “applied force,” as defined in ADAMS/Solver, refers to one of two types of forces: environmental forces or compliance forces. Environmental forces include gravitational forces, aerodynamic forces, electromagnetic forces, and so on. Compliance forces include bushings, tires, flexible booms, and so on. It is not necessary to describe inertial or reaction forces to ADAMS/Solver. ADAMS/Solver uses your parts and constraints to automatically define these. For each applied force component, ADAMS/Solver adds an applied force variable to the dynamical equations of motion and adds one algebraic force definition equation to the equation set.

ADAMS/Solver allows many force characteristics including translational and rotational action, action-reaction and action-only application, moving and fixed application points, single and multiple application points, linear or non-linear dependence, and dependence on any number of a variety of state variables, including displacements, velocities, accelerations, forces, and time. In addition, ADAMS/Solver offers special facilities for describing commonly occurring sets of forces such as bushings, beams, spring-dampers, and tires. For more information on the forces available in ADAMS/Solver, see the discussion of forces in Section 4.4.5, “Forces.”

3.2.4 User-Defined Algebraic and Differential Equations

ADAMS/Solver allows you to add both algebraic and differential equations to the equations for part, constraint, and force components. This powerful feature allows many system attributes with mathematical representations to be included in the idealized ADAMS/Solver model.
3.3 Drawing a System Schematic

To summarize the significant elements of a system, you may find it helpful to draw a system schematic. A list of the information you usually represent in a schematic follows.

- Parts
- Part centers of mass
- Joints, curves, surfaces
- Imparted motions
- Applied forces

Although your mechanism can have other elements, the list indicates those that are typical. This section explains the procedures for creating a system schematic and demonstrates them with the piston-crank mechanism in Figure 9.

![Figure 9. Piston-Crank Mechanism](image)

To begin a schematic of the mechanism in Figure 9, count its parts. The mechanism appears to have six parts: two journal bearings (A and B), the crankshaft, the connecting rod, the piston, and the cylinder. Because the two journal bearings and the cylinder are stationary, they belong to ground and you should model the mechanism in terms of four parts: the crankshaft, the connecting rod, the piston, and ground. In any ADAMS/Solver model, specify only one ground part.
After you determine the parts in your system, draw a stick figure that includes all of them. Figure 10 is a stick figure of the piston-crank mechanism.

In Figure 10, the origin of the global coordinate system is at the intersection of the crankshaft axis and the piston axis. From left to right, the other line segments represent the crankshaft, the connecting rod, and the piston.
Put a part identifier on each part in Figure 10. ADAMS/Solver allows you to have part identifiers with as many as eight digits. In this example, the identifiers have two digits as shown in Figure 11.

You may want to choose identifiers other than those in Figure 11. For instance, parts 03 and 04 could alternatively have the identifiers 300000 and 400000. Whenever possible, you should try to find a numbering scheme for your model that makes it easy to understand the relationships between parts. In some cases, 03 and 04 are not the best identifiers for the throw rod and the piston shown in Figure 11. For example, if the piston-crank mechanism was the first of four such assemblies in an engine, better identifiers for the throw rod and the piston would be 0130 and 0140. In the second assembly, the throw rod and the piston could have the identifiers 0230 and 0240.

When you finish identifying the parts in your stick figure, add symbols to the figure to represent the ground and the other system elements. Table 2 contains a set of popular symbols for common elements.
For some models, you may need to create additional symbols to represent system elements that are not as common. Table 2 illustrates all the symbols you need to define the piston-crank mechanism. Add the appropriate symbols from Table 2 to Figure 11. The results are shown in Figure 12.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>System Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Symbol]</td>
<td>Ground</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Center of mass</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Orthogonal axis triad</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Translational (prismatic) joint</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Revolute (hinge) joint</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Cylindrical joint</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Universal (Hooke’s) joint</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Spherical (ball-and-socket) joint</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Planar joint</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Rack-and-pinion joint</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Screw joint</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Imparted motion</td>
</tr>
<tr>
<td>![Symbol]</td>
<td>Applied force</td>
</tr>
</tbody>
</table>
As Figure 12 shows, put one ground symbol on the ground part and another on the cylinder that is attached to ground. Put a center-of-mass symbol at the center of mass of each part (except the ground part), and add an orthogonal axis triad to represent the reference frame for each part (except the ground part). Indicate a revolute joint on one end of the crankshaft, a spherical joint between the crankshaft and the connecting rod, a universal joint between the connecting rod and the piston, and a translational joint between the piston and ground at the cylinder.

Because one revolute joint is sufficient to describe the movement of the crankshaft with respect to ground, use only one revolute joint symbol. (See Section 2.6.1, “Redundant Constraints and Overconstraint Checking.”) You can put it anywhere along the rotational axis of the crankshaft, but a convenient place is the flywheel end, as shown in Figure 12. To indicate that the crankshaft is to be turned, put an imparted motion symbol on one end of it. To indicate a force due to pressure, put an applied force symbol on the piston. Together the part identifiers and the symbols on the stick figure complete the system schematic.
3.4 Selecting a System of Units

ADAMS/Solver characterizes mechanisms and their behavior with four fundamental quantities or dimensions: mass, length, time, and force. By specifying the proper units information to ADAMS/Solver, you may use any system of units to build your model. Once you have chosen a set of units, you must use them consistently throughout your model. For example, if you choose length, units of millimeters, and time units of seconds, you must enter velocities in millimeters per second.

ADAMS/Solver allows you to directly select most common units for mass, length, time and force. You may also provide a units consistency factor if you want to use a set of units not covered by the common choices.

3.5 Constructing the System Equations

As discussed in Section 3.2, “Decomposing the Model into Basic Components” the system of equations governing the behavior of any mechanism in ADAMS/Solver consists of:

- Six first-order dynamical equations for each part (relating forces to accelerations)
- Six first-order kinematical equations for each part (relating positions to velocities)
- A single algebraic constraint for each motion constraint
- A single algebraic equation for each scalar force component
- Any number of user-defined algebraic or first-order differential equations

ADAMS/Solver formulates all of the equations on the above list and assembles them into a single system to be solved simultaneously for the time-dependent values of the entries in the state vector.
The column matrix of the unknowns for the system of equations is known as the state vector. It is made up of:

- The translational velocities for each part
- The angular velocities for each part
- The three Cartesian coordinates for the center of mass of each part
- The three orientation angles that define the attitude of each part
- The applied forces
- The constraint forces (or Lagrange multipliers)

You can write the equations in the system in general form as the following three blocks of equations:

\[
\begin{align*}
\mathbf{M}(\mathbf{q}, \dot{\mathbf{u}}, \ddot{\mathbf{u}}, f, t) &= 0 \\
\dot{\mathbf{u}} - \dot{\mathbf{q}} &= 0 \\
\Phi(\mathbf{q}, f, t) &= 0
\end{align*}
\]  \tag{1}

where the underscore indicates a matrix and the dot (.) denotes the time derivative. The column matrix \(\mathbf{q}\), called the generalized coordinates, contains the translational and rotational variables. The column matrix \(\dot{\mathbf{u}}\) contains the time derivatives of the components of \(\mathbf{q}\) as indicated by the second block of equations which are the kinematic differential equations. The constraint forces and applied forces comprise the column matrix \(f\) and \(t\) is time.

The column matrix \(\mathbf{M}\) represents all of the differential equations governing the dynamics of the problem plus any user-defined differential equations that you may have added to the system. The column matrix \(\Phi\) includes all of the algebraic equations that make up the constraints for the model. A typical dynamic differential equation is

\[m \ddot{z} - mg - \lambda = 0\]

while an example of a constraint equation is

\[z \cdot L \cos \Theta = 0\]

where \(z\) is the vertical coordinate of the center of a part with mass \(m\) and orientation \(\Theta\).

The variable \(\lambda\) represents a constraint force on the part. For this example, \(L\) is some constant in the model and \(g\) is the acceleration due to gravity.
3.6 Solving the Differential and Algebraic Equations

The system (Eq (1)) is made up only of equations like the two examples shown above. Thus, an ADAMS/Solver model is governed by a very sparse, coupled collection of mixed nonlinear differential and algebraic equations (DAE’s).

ADAMS/Solver can solve various combinations of the DAE’s in different analysis modes to perform kinematic, static, or dynamic simulations. The following paragraphs contain a brief outline of the methods employed in ADAMS/Solver.

Let y be the state vector or column matrix composed of all of the variables in \( q, \dot{q}, \text{ and } f \). Then the system (Eq (1)) can be rewritten as the matrix equation

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

which is a single set of \( N \) equations in \( 2N \) unknowns. In order to reduce the number of unknowns in the system (Eq (2)) from \( 2N \) to \( N \), the time derivative of each component is approximated by the Backward Differentiation Formula (BDF). For a dynamic analysis, the ADAMS/Solver code solves the full matrix of equations \( G \) at discrete points in time \( t \).

Given the solution at time \( t_n \), an interpolatory polynomial through several previous values of each component of \( y \) is used to predict its value and the value of its derivative at time \( t_{n+1} \). This polynomial is known as the predictor and the degree of the polynomial is the order of the method, or integration order.

The following paragraphs contain a brief outline of the methods employed in ADAMS/Solver.

3.6.1 Performing a Dynamics Analysis

Two principal methods are available for performing a dynamic analysis:

1. Integrators using DAE’s
2. Integrators using a reduced set of differential equations (ODE’s)

To start the dynamics problem at the first time step, ADAMS/Solver performs an initial value analysis that provides a consistent set of values for the state vector and its first derivative at time zero. The initial value and the derivative are used for a linear prediction to time \( t_1 \) and the first order formula is used in the system (Eq (2)).
3.6.2 Integrators Using DAE’s

In ADAMS, the GSTIFF integrator, the DSTIFF integrator, and the BDF integrator with variable coefficients predict, then correct a full set of DAE’s using the Backward Differentiation Formula (BDF) until the corrector convergence criteria is met, or until the corrector reaches the maximum number of iterations, which is under your control. (Note that the BDF integrator with variable coefficients is accessed with the DYNAMICS or DSOLVER statement/command.) If the corrector fails, the step size is adjusted and the predictor-corrector algorithm is tried again.

The integration step concludes with a test for the error tolerance. If the difference between the predicted values of \( y \) and the converged, corrected values of \( y \) agree to within the error tolerance that you specified, then the solution is accepted and ADAMS/Solver continues with the next step. Otherwise, the time step is cut by a factor related to the degree by which the error tolerance was violated and the predictor-corrector algorithm is attempted again.

For the three BDF integrators, the predictor-corrector integration scheme is summarized as follows:

**Predict**

1. Predict values for \( y \) and \( \dot{y} \) at \( t_{n+1} \) that satisfy the predictor interpolatory polynomial.

**Correct**

2. Evaluate \( G \). If \( G(y, \dot{y}, t_{n+1}) = 0 \), then the solution is given by the current values in the column matrix \( y \), or else continue.

3. Solve the Newton-Raphson linear equation for an update \( \Delta y \) to \( y \) to make the \( G \) vector closer to 0.

\[
J \Delta y = G(y, \dot{y}, t_{n+1})
\]

where \( J = \) Jacobian matrix for the DAE’s.

4. Update \( y \) at the \( (k+1) \)th Newton-Raphson iteration as

\[
y^{k+1} = y^k + \Delta y^k
\]

5. Repeat steps 2 through 4 until the iterations satisfy the convergence criteria.
Error control
6. Estimate the integration error and compare it to the error tolerance. If the error
is too large, reject the step.
7. Compute the optimum step size and order for the next step.
8. If the current time has reached the ending time, stop the simulation.
   Otherwise, increment the time by the time step and go to step 1.

The predictor-corrector, described in these eight steps, is implemented in
ADAMS/Solver with several qualifications and improvements to enhance
efficiency and robustness of this algorithm.

3.6.3 Integrators Using a Reduced Set of ODE’s

The ABAM (Adams-Bashforth and Adams-Moulton) integrator in ADAMS relies on a
coordinate partitioning scheme to reduce DAE’s to ODE’s. Using the algebraic
constraint equations, the set of system coordinates is partitioned into an independent
and a dependent set. Based on this partitioning, a reduced set of ODE’s representing
the system are obtained. This reduced set of ODE’s are integrated using the ABAM
integrator.

The steps involved in determining and integrating a reduced set of ODE’s can be
summarized as follows.

Coordinate Partitioning
1. Using the algebraic constraint equations, partition the total set of coordinates
   \( \{ q \} \) for the system into a set of independent and a set of dependent
   coordinates.

   \[
   \{ q \} = \begin{cases} 
   \{ q^i \} \\
   \{ q^d \}
   \end{cases}
   \]

   where \( \{ q \} = \) total set of system coordinates, \( \{ q^i \} = \) partitioned set of
   independent coordinates, and \( \{ q^d \} = \) partitioned set of dependent coordinates

Predict
2. Predict value of independent coordinates at time \( t_{n+1} \) using the explicit
   Adams-Bashforth explicit formulae. This gives us

   \[
   \{ q^i \}^p
   \]

   where superscript \( p \) indicates predicted values. This prediction is based on the
   value of \( \{ q^i \} \) in the preceding time steps.
3. Correct the predicted value of the independent coordinates using the Adams-Moulton formulae to give

\[ \{ q^i \}^c \]

where superscript \( c \) indicates corrected values.

4. Having obtained the corrected independent coordinates, compute all the dependent coordinates and other system variables.

**Error Control**

Error control is similar to that stated above in “Error Control” for integrators using DAE’s.

### 3.6.4 Static and Kinematic Analyses

For static analysis, the coupled set of DAE’s is considered. The dynamical effects are reduced to zero by setting all velocities to zero and then solving the DAE’s. Mass effects are included while computing the static equilibrium.

For a kinematic analysis, the iterative algorithm is used to solve only the block of algebraic constraint equations in the system (Eq (1)).
3.7 Reviewing the System Response

Section 3.1, “Creating an Idealized Model from the Physical Model” explains that when you idealize your model it is important to consider the results you want. Whether you write your ADAMS/Solver data set, or use ADAMS/View to generate it, you should carefully indicate what data you want output. In general, it is best to output any information you think is useful for model verification or system analysis. When ADAMS/Solver performs a dynamic, kinematic, or quasi-static simulation, ADAMS/Solver outputs data at each time step. When ADAMS/Solver performs a static simulation, ADAMS/Solver outputs data once.

You can output data for displacements, velocities, accelerations, reaction forces, applied forces, user-defined variables, generic system elements, and system modeling element inputs and outputs at each output step to the Request and/or Results File. ADAMS/View can read either file for plotting purposes. Plots are often useful for studying the specific behavior of a model. For example, suppose you need to know the maximum translational loading at a joint. You could plot the translational reaction forces at the joint over the course of the entire simulation and quickly determine the maximum load. Figure 13 shows a plot of the translational reaction forces in the translational joint between the base and the upper link of a robot arm.

![Figure 13. Plot of Translational Joint Reaction Force](image-url)
ADAMS/Solver also can output graphical data at each output step to the Graphics File. Graphical output is often useful for viewing the overall behavior of the model and for pinpointing specific problems, such as:

- improper connectivity
- misapplied motions or forces
- unwanted oscillatory behavior
- clearance problems

Figure 14 shows superimposed frames from the animation of a robot arm. Both single-frame and animated displays are available with either solid fill or shaded rendering. For more information about the wide scope of tools available in ADAMS/View for reviewing the response of your system, see “Animation,” “Numeric Results,” “Plots,” and “Printing,” in the ADAMS/View User’s Reference Manual.

Figure 14. Superimposed Frames from the Animation of a Robot Arm
Chapter 3. Modeling Process
Chapter 4
ADAMS/Solver Statements

Introduction

This section focuses on helping you to create an ADAMS/Solver model of a system using ADAMS/Solver statements. To do this, we have presented general schemes to show you how to organize the data in the input file, and the syntax and format rules that govern ADAMS/Solver statements. Lastly, we briefly summarize the available ADAMS/Solver statements and categorize them in easily understandable groups.

The principal method of specifying a model definition to ADAMS/Solver is with a model definition input data set. A model definition input data set is composed of several statements. Each statement defines an instance of a model definition element or entity. These include the specification of mass and inertia properties, geometry, and connectivity, as well as analysis control and output specification.

Each statement consists of an identifier and a set of keywords that qualify the statement. You may specify most system properties and data in the model definition input data set. You may also use the data set to refer to special purpose FORTRAN-77 subroutines that you may have written to describe “non-standard” system characteristics.
4.1 Organization in the Data Set

Model definition input data files always begin with a TITLE statement and end with the END statement. The statements and comments between the TITLE and END statements constitute the body of the data set. Statements may be defined in any order without affecting the results of an analysis. However, a careful arrangement of them can greatly increase data set readability and simplify debugging, mode sharing, and modification. This section suggests two methods for organizing data sets and discusses the implications of each.

The following sample structure shows a data set for the piston-crank mechanism discussed in Section 3.3, “Drawing a System Schematic.” The geometric information in the data set describes the mechanism in the top dead-center position.

PISTON COMPRESSOR SAMPLE DATA SET

!*!!! PARTS AND THEIR MARKERS !!!* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PART/01, GROUND                      GROUND—PART 01
MARKER/0103                          GLOBAL REFERENCE FOR REQUESTS
MARKER/0102, QP=0,0,-2              REVOLUTE JOINT TO CRANK
MARKER/0108, QP=12,0,0,            TRANSLATIONAL JOINT,
, REULER=90D,90D,0                   TO PISTON
MARKER/0109, QP=12.286,0,0         GAS FORCE APPLICATION POINT
PART/02, MASS=2, CM=0201             CRANK—PART 02
, IP=5,6,10
MARKER/0201, QP=0,0,-3              CENTER OF MASS
MARKER/0202, QP=0,0,-2              REVOLUTE JOINT TO GROUND
MARKER/0203                          GRAPHICS MARKER
MARKER/0204, QP=2,0,0               SPHERICAL JOINT TO ROD
PART/03, MASS=1, CM=0305             ROD—PART 03
, IP=1.5,1.6,3, QG=2,0,0             LPRF IN GLOBAL POSITION
MARKER/0305, QP=4,0,0               CENTER OF MASS
MARKER/0304, QP=0,0,0               SPHERICAL JOINT TO CRANK
MARKER/0306, QP=8,0,0               UNIVERSAL JOINT TO PISTON
PART/04, MASS=.5, CM=0407            PISTON—PART 04
, QG=10,0,0                         LPRF IN GLOBAL POSITION
MARKER/0407, QP=1,0,0               CENTER OF MASS
, REULER=90D,90D,0                   UNIVERSAL JOINT TO ROD
MARKER/0406, ZP=0,1,0               TRANSLATIONAL JOINT TO
, XP=0,0,1
MARKER/0408, QP=2,0,0               VISCOS FORCE
, REULER=90D,90D,0                   GROUND AND GAS FORCE
MARKER/0409, QP=2,0,0               VISCOS FORCE
, REULER=90D,90D,0                   GROUND AND GAS FORCE

!*!!! CONSTRAINTS !!!* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
JOINT/0201, REVOLUTE                 CRANK TO GROUND
, I=0202, J=0102
JOINT/0302, SPHERICAL               ROD TO CRANK
, I=0304, J=0204
JOINT/0403, UNIVERSAL               PISTON TO ROD
, I=0406, J=0306
JOINT/9401, TRANSLATIONAL           PISTON TO GROUND
, I=0408, J=0108
MOTION/0201, JOINT=0201             SPECIFIED CRANK MOTION
, FUNCTION=POLY(TIME, 0, 0, 360D)
Chapter 4. ADAMS/Solver Statements

!****** FORCES*******************************************************
SFORCE/0401, I=0408, J=0109       PISTON VISCOUS DAMPING
   , TRANSLATION
   , FUNCTION=POLY(VR(408, 109), 0, 0, -0.2)
SFORCE/0104, I=0409, J=0109       ISENTROPIC GAS FORCE
   , TRANSLATION
   , FUNCTION=1422.0 * DM(0409, 0109) ** (-1.4) - 185.4

!****** REQUEST OUTPUT************************************************
REQUEST/01, DISPLACEMENT, I=0201, J=0103
   , COMMENT=CRANK CENTER OF MASS DISPLACEMENT
REQUEST/02, ACCELERATION, I=0201, J=0103
   , COMMENT=CRANK CENTER OF MASS ACCELERATION
REQUEST/03, DISPLACEMENT, I=0305, J=0103
   , COMMENT=ROD CENTER OF MASS DISPLACEMENT
REQUEST/04, ACCELERATION, I=0305, J=0103
   , COMMENT=ROD CENTER OF MASS ACCELERATION
REQUEST/05, DISPLACEMENT, I=0407, J=0103
   , COMMENT=PISTON CENTER OF MASS DISPLACEMENT
REQUEST/06, ACCELERATION, I=0407, J=0103
   , COMMENT=PISTON CENTER OF MASS ACCELERATION
REQUEST/07, FORCE, I=0202, J=0102
   , COMMENT=CRANK/GROUND REACTION
REQUEST/08, FORCE, I=0304, J=0204
   , COMMENT=ROD/CRANK REACTION
REQUEST/09, FORCE, I=0406, J=0306
   , COMMENT=PISTON/ROD REACTION
REQUEST/10, FORCE, I=0408, J=0108
   , COMMENT=PISTON/GROUND REACTION
REQUEST/11, FORCE, I=0408, J=0109
   , COMMENT=PISTON VISCOUS DAMPING
REQUEST/12, FORCE, I=0409, J=0109
   , COMMENT=GAS PRESSURE FORCE

!****** GRAPHICS OUTPUT***********************************************
GRAPHICS/0201                     CRANK OUTLINE
   , OUTLINE=0201,0203,0204
GRAPHICS/0202, CYLINDER           FLYWHEEL
   , CM=0201, RADIUS=2
   , LENGTH=-2, SIDES=20, SEG=20
GRAPHICS/0301                     ROD OUTLINE
   , OUTLINE=0304,0306
GRAPHICS/0401, CYLINDER           PISTON RIGHT HALF
   , CM=0407, RADIUS=.75
   , LENGTH=1, SIDES=20, SEG=20
GRAPHICS/0402, CYLINDER           PISTON LEFT HALF
   , CM=0407, RADIUS=.75
   , LENGTH=-1, SIDES=20, SEG=20

!****** SOLUTION AND OUTPUT PARAMETERS*******************************
ACCGRAV/ JGRAV =-386.088          GRAVITY BODY FORCE
   , GC=386.088                     AND SYSTEM OF UNITS
OUTPUT/ GSSAVE, REQSAVE           TURN ON GRAPHICS
   , AND REQUEST FILES
END

The TITLE statement for this example is PISTON COMPRESSOR SAMPLE DATA SET. Notice that the statements are in blocks or sections separated by blank lines. In some cases, the comments occupy entire lines; and, in other cases, the comments occupy parts of lines following executable statements. ADAMS/Solver requires neither sections nor comments, but both increase data set readability.
The data set can be organized into six major sections: parts and their markers; constraints; forces; request output; graphics output; and solution, execution, and output parameters. When you have part drawings that provide marker coordinates with respect to separate LPRFs, this organization is particularly convenient. The major disadvantage of this organization is that it separates statements for joints, forces, and requests from the MARKER statements to which they refer. For example, the two MARKER statements that define a joint are beneath the PART statements and not with the JOINT statement. In short, this organization makes it necessary to search the data set to find all the information you need concerning a particular system element, but it is “hierarchically” organized.

An alternate organization scheme for the piston-crank mechanism follows. This scheme emphasizes a “relational” organization of data.

PISTON COMPRESSOR SAMPLE DATA SET

| PART MODULES | PART/01,GROUND | PART/02, MASS=2, CM=02, IP=5, 6, 10 | PART/03, MASS=1, CM=06, IP=1.5, 1.6, 3 |
| PRIMARY MARKERS | MARKER/01 | MARKER/02 | MARKER/03 | MARKER/04 | MARKER/05 | MARKER/06 | MARKER/07 | MARKER/08 | MARKER/09 |
| REQUESTS | REQUEST/01, DISPLACEMENT, I=02, J=01, COMMENT=CRANK CG DISPLACEMENT | REQUEST/02, ACCELERATION, I=02, J=01, COMMENT=CRANK CG ACCELERATION | REQUEST/03, DISPLACEMENT, I=06, J=01, COMMENT=ROD CENTER OF MASS DISPLACEMENT | REQUEST/04, ACCELERATION, I=06, J=01, COMMENT=ROD CENTER OF MASS ACCELERATION |
| GRAPHICS | GRAPHICS/01, OUTLINE=03,04,05 CRANK OUTLINE | GRAPHICS/02, CYLINDER, CM=03 FLYWHEEL | GRAPHICS/03, OUTLINE=07,08 ROD OUTLINE | GRAPHICS/04, OUTLINE=09,10,11 ROD OUTLINE |

| PRIMARY MARKERS | MARKER/01 | MARKER/02 | MARKER/03 | MARKER/04 | MARKER/05 | MARKER/06 | MARKER/07 | MARKER/08 | MARKER/09 |
| REQUESTS | REQUEST/01, DISPLACEMENT, I=02, J=01, COMMENT=CRANK CG DISPLACEMENT | REQUEST/02, ACCELERATION, I=02, J=01, COMMENT=CRANK CG ACCELERATION | REQUEST/03, DISPLACEMENT, I=06, J=01, COMMENT=ROD CENTER OF MASS DISPLACEMENT | REQUEST/04, ACCELERATION, I=06, J=01, COMMENT=ROD CENTER OF MASS ACCELERATION |
| GRAPHICS | GRAPHICS/01, OUTLINE=03,04,05 CRANK OUTLINE | GRAPHICS/02, CYLINDER, CM=03 FLYWHEEL | GRAPHICS/03, OUTLINE=07,08 ROD OUTLINE | GRAPHICS/04, OUTLINE=09,10,11 ROD OUTLINE |

| PRIMARY MARKERS | MARKER/01 | MARKER/02 | MARKER/03 | MARKER/04 | MARKER/05 | MARKER/06 | MARKER/07 | MARKER/08 | MARKER/09 |
| REQUESTS | REQUEST/01, DISPLACEMENT, I=02, J=01, COMMENT=CRANK CG DISPLACEMENT | REQUEST/02, ACCELERATION, I=02, J=01, COMMENT=CRANK CG ACCELERATION | REQUEST/03, DISPLACEMENT, I=06, J=01, COMMENT=ROD CENTER OF MASS DISPLACEMENT | REQUEST/04, ACCELERATION, I=06, J=01, COMMENT=ROD CENTER OF MASS ACCELERATION |
| GRAPHICS | GRAPHICS/01, OUTLINE=03,04,05 CRANK OUTLINE | GRAPHICS/02, CYLINDER, CM=03 FLYWHEEL | GRAPHICS/03, OUTLINE=07,08 ROD OUTLINE | GRAPHICS/04, OUTLINE=09,10,11 ROD OUTLINE |
Chapter 4. ADAMS/Solver Statements

Graphics

GRAPHICS/04, CYLINDER, CM=10 PISTON RIGHT HALF
  RADIUS=.75, LENGTH=1, SIDES=20, SEG=20
GRAPHICS/05, CYLINDER, CM=10 PISTON LEFT HALF
  RADIUS=.75, LENGTH=-1, SIDES=20, SEG=20
MARKER/10, QP=11,0,0 GRAPHICS MARKER
  REULER=90D,90D,0

*******CONSTRAINT MODULES*****************************

CONSTRAINT - (REVOLUTE) CRANK ON GROUND
JOINT/01, REVOLUTE, I=11, J=12
MARKER/11, PART=02, QP=0,0,-2 CRANK MARKER
MARKER/12, PART=01, QP=0,0,-2 GROUND MARKER
REQUEST/07, FORCE, I=11, J=12
  COMMENT=CRANK/GROUND REACTION

CONSTRAINT - (SPHERICAL) ROD ON CRANK
JOINT/02, SPHERICAL, I=13, J=14
MARKER/13, PART=03, QP=2,0,0 ROD MARKER
MARKER/14, PART=02, QP=2,0,0 CRANK MARKER
REQUEST/08, FORCE, I=13, J=14
  COMMENT=ROD/CRANK REACTION

CONSTRAINT - (UNIVERSAL) PISTON ON ROD
JOINT/03, UNIVERSAL, I=15, J=16
MARKER/15, PART=04, QP=10,0,0 PISTON MARKER
  ZP=10,1,0, XP=10,0,1
MARKER/16, PART=03, QP=10,0,0 ROD MARKER
REQUEST/09, FORCE, I=15, J=16
  COMMENT=PISTON/ROD REACTION

CONSTRAINT - (TRANSLATIONAL) PISTON ON GROUND
JOINT/04, TRANSLATIONAL, I=17, J=18
MARKER/17, PART=04, QP=12,0,0 PISTON MARKER
  REULER=90D,90D,0
MARKER/18, PART=01, QP=12,0,0 GROUND MARKER
  REULER=90D,90D,0
REQUEST/10, FORCE, I=17, J=18
  COMMENT=PISTON/GROUND REACTION

CONSTRAINT - (MOTION) GROUND MOTION ON CRANK
MOTION/01, JOINT=01, FUNCTION=POLY(TIME, 0, 0, 360D)

******* FORCE MODULES*****************************

FORCE - (SFO) PISTON VISCOUS DAMPING
SFORCE/01, I=19, J=20, TRANSLATION
  FUNCTION=POLY(VR(19, 20), 0, 0, -0.2)
MARKER/19, PART=04, QP=12,0,0 PISTON MARKER
MARKER/20, PART=01, QP=12.286,0,0 GROUND MARKER
REQUEST/11, FORCE, I=19, J=20
  COMMENT=PISTON VISCOUS DAMPING

FORCE - (SFO) ISENTROPIC GAS PRESSURE FORCE
SFORCE/02, I=21, J=22, TRANSLATION
  FUNCTION=1422.0 * DM(21, 22) ** (-1.4) - 185.4
MARKER/21, PART=04, QP=12,0,0 PISTON MARKER
MARKER/22, PART=01, QP=12.286,0,0 GROUND MARKER
REQUEST/12, FORCE, I=22, J=21
  COMMENT=ISENTROPIC GAS PRESSURE FORCE

******* SOLUTION AND OUTPUT PARAMETERS****************

ACCGRAV/ JGRAV=-386.088 GRAVITY BODY FORCE
  GC=386.088 AND UNITS SYSTEM
OUTPUT/ GRSAVE, REQSAVE ENABLE GRAPHICS AND REQUEST FILES

END
Although the second sample data set defines the same model as the first sample data set, the two differ in their organizations. Notice that in the second sample data set, the statements and comments are in four sections rather than in six. The four sections are for parts; constraints; forces; and solution, execution, and output parameters. Note that at time zero, the LPRF of each part has the same location and orientation as the ground coordinate system.

The major advantage of this data set organization is that it groups MARKER statements with the statements for parts, constraints, forces, and requests to which they refer. In addition, the position and orientation of each LPRF is the same; so to superimpose two markers for a joint, you simply input the same geometric data for them. To describe any point in more than one part, constraint, force, or request, this organization requires that you use multiple MARKER statements so that each MARKER statement is used only once. This data set organization decreases the time you spend searching for data and makes the data set easier to modify.

In general, when you are defining parts with respect to LPRFs, an organization like the one in the first sample data set is easier to use. However, if you are superimposing all the LPRFs on the ground coordinate system, an organization like the one in the second sample data set is easier to use. The primary objective, of course, is to find an organization that works best for you and for the types of models you are defining. In some cases, you may find it useful to employ a data set organization other than these two.
4.2 Statement Format

The general format for all ADAMS/Solver statements is in Figure 15.

```
NAME/[id,] ARG1 = {v1,...,vn}
c(v1,...,vn)
[ ] Optionally select the item
```

Every statement except the TITLE statement includes a name that indicates the statement type. Most statements require a slash (/) delimiter, an identifier (id), and one or more arguments (ARG1,...,ARGn). Statement arguments may indicate a condition, values (...,vn), a character string (c) with values, or an expression (e). You can classify arguments into one of the three following categories:

- **Conditional** -- These arguments are of the form \( \text{ARGUMENT} \)
- **Assignment** -- These arguments are of one of two forms
  - \( \text{ARGUMENT} = \{v1[,...,vn]\} \)
  - \( \text{ARGUMENT} = \{c1[,...,cn]\} \)
- **Function Expression** -- These arguments are of the form:
  - \( \text{ARG} = \text{e} \)
  - \( \text{ARG} = \text{user}(r1[,...,r30]) \)

A different set of arguments characterizes each ADAMS/Solver statement. If you do not include mandatory arguments, ADAMS/Solver outputs an error message and stops execution after the input phase. Some of the arguments in a statement may be optional, therefore, the number of arguments you use for a statement can vary. Argument order can also vary without changing the overall meaning of a statement. For example, the argument orders in the statements below differ, but they provide ADAMS/Solver with the same information.

```
REQUEST/04, FORCE, I=0304, J=0204
REQUEST/04, I=0304, FORCE, J=0204
```

Although it is unnecessary to order arguments in a consistent fashion, such a practice facilitates locating argument information. The following sections describe each argument category and its distinctive attributes.
4.2.1 Conditional Arguments
Arguments that specify a quality are termed conditional arguments. Conditional arguments do not explicitly contain any argument values. They consist solely of the argument key word. For example, consider the SFORCE statement from the first sample data set in Section 4.1, “Organization in the Data Set.”

```
SFORCE/0104, I=0408, J=0109
    TRANSLATION
    FUNCTION=1422.0 * DM(0408, 0109) * * (-1.4) - 185.4
```

The conditional argument in this statement is TRANSLATION. TRANSLATION specifies that the force is translational (as opposed to being rotational).

4.2.2 Assignment Arguments
Arguments that assign one or more values to a statement attribute are termed assignment arguments. Each of these arguments consists of the argument key word, an equal sign (=), and a set of values you want to assign to the attribute. In exemplary formats, lower-case terms indicate the types of values that are to follow the argument key word. The types of values may be identifiers (id1,...,idn), Cartesian coordinates (x, y, z), or angular values (a, b, c). Types of values may be integers (i1,...,in), real numbers (r1,...,rn), or character strings (c1:...:cn). In some cases, arguments accept more than one kind of value; their formats specify varying type (v1,...,vn). Regardless of the kinds of values you enter for an assignment argument, you must precede all values except the first with either a comma or a colon. A comma must separate numerical values and a colon must separate character strings. You may break assignment arguments for statement before any comma (,) or colon (:). If you break the argument before a comma, do not retain the separating comma.

The following PART statement from the first sample data set includes the assignment arguments MASS, CM, IP, and QG:

```
PART/03, MASS=1, CM=0305
    IP=1.5,1.6,3, QG=2,0,0
```

Here, the value of 1 following the MASS keyword indicates the value of the mass of the part, the value following CM indicates the identifier of the part center-of-mass marker, the value following IP indicates the three principal mass moments of inertia of the part about the part center-of-mass marker, and the values following QG indicate the initial coordinates of the LPRF of the part with respect to the ground coordinate system.
4.2.3 Function Arguments

Arguments that allow you to define their values with a FORTRAN-like expression in the data set or with an external user-written subroutine are termed function expression arguments. Each of these arguments consists of the argument, an equal sign (=), and an expression that ADAMS/Solver can evaluate. There are two types of function expression arguments: those that define an expression (e) and those that pass constants to a user-written evaluation subroutine. The following are examples of the two types of function expression arguments:

- This MOTION statement from the first sample data set includes a function expression argument that uses an expression to compute a MOTION.
  
  \[ \text{MOTION/0201, JOINT=0201, FUNCTION=POLY(TIME,0,0,360D)} \]

- The following statement includes a function expression argument that defines constants ADAMS/Solver is to pass to the user-written subroutine MOTSUB:

  \[ \text{MOTION/0201, JOINT=0201, FUNCTION=USER(0, 0, 6.28)} \]

In the first example, the expression specifies the four values ADAMS/Solver is to pass to the POLY function to determine the MOTION. Chapter 5, “Customizing ADAMS/Solver,” explains how to write a function expression. Chapter 4, “Function Expressions,” in the ADAMS/Solver Reference Manual discusses the creation of function expressions in more detail. In most cases, defining an expression is easier than writing and then passing values to a user-written subroutine as in the second example. Chapter 2, “User-Written Subroutines,” and Chapter 3, “Utility Subroutines,” in the ADAMS/Solver Subroutines Reference Manual discuss the user-written subroutines and other ADAMS/Solver utilities you may access from user-written subroutines.

You may break a function expression for continuation anywhere except in the middle of a number, in the middle of a function name, or between a function name and the left parenthesis that follows it. If you break the expression before a comma (,), you must retain the comma (in addition to the continuation comma).

To terminate a function expression, you must put the function expression argument last in your statement or put a backslash (\) at the end of the expression to explicitly specify the end of the expression.

Some users outside the United States who use ADAMS/Solver on IBM systems must use the double quotation marks (“”) instead of the backslash and other alternate characters. (Ask your systems support personnel which you should use.)
4.3 Statement Syntax

Like all languages, the statements specifying a model definition must follow certain syntactical rules. ADAMS/Solver relies on these rules to decode a statement and understand it. This section explains the syntactical rules that govern statement definitions and their interpretation.

4.3.1 Angles

- ADAMS/Solver assumes all angular data is in radians.
- If you want to indicate that angular values are in degrees, follow the values with a D, as in these examples.
  
  \[45^\circ\] 
  \[45D\] 
  \[1.E-3D\]

D does not specify double precision in ADAMS/Solver as it does in FORTRAN. In ADAMS/Solver, a D after a number indicates that the number is to be scaled by the factor \(\frac{\pi}{180}\).

- All output angular data defaults to radians, except for the rotational displacement data in the tabular output file. Use the DSF argument on the command/statement (see the OUTPUT statement and command, Sections 2.10.4 and 3.7.1, in the ADAMS/Solver Reference Manual) if you want to scale angular displacement output from degrees to radians.

4.3.2 Arguments

- Each argument indicates a condition, assigns a value, or evaluates an expression. ADAMS/Solver has three basic types of arguments. They are described in “Format.”
- An equal sign (=) separates the argument keyword from its values. Argument values can be in the form of numbers (with the ‘E’ added to indicate scientific notation and/or the ‘D’ added to indicate degrees), alphabetic characters, a combination of numbers and alphabetic characters, a string of words, and function expressions.
- Five or more consecutive blank spaces in an argument (except a function expression argument) cause ADAMS/Solver to ignore the rest of the line (see “Blanks”).
- ADAMS/Solver accepts both upper- and lower-case letters for arguments. You may arrange arguments in any order within the statement to which they
Chapter 4. ADAMS/Solver Statements

- You may break function expression arguments before any comma (,) or colon (:). If you break the argument before a comma, retain the comma (in addition to the continuation comma). If you break the argument before a colon, retain the colon. (See Section 4.3.5, “Continuations,” in this chapter).

- You may break assignment arguments before any comma (,) or colon (:). If you break the argument before a comma, do not retain the comma (in addition to the continuation comma). If you break the argument before a colon, retain the colon. (See Section 4.3.5, “Continuations,” in this chapter.)

- If you do not provide an optional argument or value, ADAMS/Solver uses the default.

- Do not assign a real number value to an integer argument unless there is nothing except a zero after the decimal point. If you assign a real number with a fractional part after the decimal point to an integer argument, ADAMS/Solver issues an error message.

- You may use scientific notation for both real and integer values.

- The following errors in the use of arguments cause ADAMS/Solver to issue an error message and stop execution at the end of the input phase:
  - You assign argument values that are not numbers of the permissible set of values for that argument.
  - You input an invalid number of values for an argument.
  - You input an ambiguous argument abbreviation.
  - You input an invalid value type for an argument (for example, you input a character value, and the argument requires a numeric value; or you input a real number with a nonzero fractional part, and the argument requires an integer).
  - You input two or more arguments that are mutually exclusive.
  - You do not provide a required argument or a required value for an argument.

4.3.3 Blanks and Tabs

- ADAMS/Solver ignores all data following five or more consecutive blank spaces on any line. There are two exceptions to this rule: Function expressions and the TITLE statement.

- Do not put blanks in numerical values.

- Blank spaces provide spacing within statements, and blank lines provide spacing between groups of statements.

- You can have as many blank lines as you want in an ADAMS/Solver data set.

- Do not use tabs in ADAMS/Solver statements. You may use tabs in
4.3.4 Comments

- Use comments to describe statements and groups of statements. ADAMS/Solver ignores comments.

- An exclamation point (!) indicates to ADAMS/Solver that the line is a comment. Some users outside the United States who run ADAMS/Solver on IBM systems must use the percent sign (%) instead of the exclamation point. (See your systems support personnel to find out which you should use.)

- At least five consecutive blank spaces will also indicate a comment everywhere except within a function expression argument. Within a function expression argument, five blank spaces do not cause ADAMS/Solver to stop reading the line.

- An ADAMS/Solver data set can have any number of consecutive or nonconsecutive comments.

- A comment can be on any line between the TITLE statement and the END statement. (ADAMS/Solver does not read beyond the END statement.) A comment can be in the middle of a statement or even in the middle of an argument, but if a comment is an entire continuation line, it must start with either a continuation character (see Section 4.3.5, “Continuations,” in this section) or an exclamation mark.

4.3.5 Continuations

- Commas in column one are used to indicate continuation lines. To continue a statement on the next line, break the line before any statement comma (,) or colon (:) and put a continuation comma in column one of the following line before the rest of the statement. If you break the statement at a comma directly preceding an argument, you do not need to retain the comma preceding the argument. If you break the statement before a comma within a function expression argument, use both the comma separating the values within the argument and the continuation comma. If you break the statement before a comma within an assignment argument, use only one comma. If you break the statement before a colon in either type of argument, retain the colon.

- Alternately, ampersands (&) can indicate continuation lines. To continue a statement on the next line, break the statement before any statement comma (,) or colon (:) and put an ampersand (&) after the last statement character on the line to indicate that the following input line is a continuation of the line with the ampersand. Because ADAMS/Solver ignores all comments, putting an ampersand after the comment will not indicate to ADAMS/Solver that the following input line is a continuation line. You cannot use an ampersand to continue a function expression on the next line. If you must continue a function expression on the next line, use a comma in the first column of the
new line as described above.

4.3.6 Delimiters

- Delimiters allow ADAMS/Solver to interpret input, and they make data sets easier to read. ADAMS/Solver commonly requires or allows six delimiters: the slash (/), the equal sign (=), the comma (,), the colon (:), the backslash (\), and the semicolon (;).

- The slash (/) separates a statement name from the remainder of the statement.

- The equal sign (=) separates an argument key word from a value or a set of values.

- The comma (,) separates arguments within a command and separates multiple integer values \((i_1,\ldots,i_n)\) and multiple real values \((r_1,\ldots,r_n)\) within multiple valued arguments.

- The backslash (\) (or, for some users, the double quotation marks) instead of the comma separates a function expression argument from any arguments that follow it.

- The semicolon (;) separates two statements on the same line.

- The following errors in the use of delimiters cause ADAMS/Solver to issue an error message and stop execution at the end of the input phase:
  - You do not use a delimiter when it is required (for example, you omit the backslash between a function expression argument and the following argument).
  - You use a delimiter when it is not allowed.
  - You use two or more delimiters when only one is allowed (for example, you use two commas between two arguments on the same line.)
4.3.7 Identifiers

- Identifiers distinguish statements of the same type.
- In a data set, you should not have two or more statements with the same name and the same identifier (for example, two GRAPHICS statements with the same identifier). If your data set does include two or more statements with the same name and the same identifier, ADAMS/Solver issues one or more warning messages and stores the last instance only. Not all statement types require an identifier.
- If a statement type requires an identifier, the identifier string must follow a slash (/).
- The identifier you assign to a statement can be any positive integer with eight or fewer digits.
- The following errors in the use of identifiers cause ADAMS/Solver to issue an error message and stop execution at the end of the input phase:
  - You omit a statement identifier that ADAMS/Solver requires.
  - You use a statement identifier that is negative, is a real number, or is a positive integer with more than eight digits.
- The following error in the use of identifiers causes ADAMS/Solver to issue a warning message and to ignore the identifier:
  - You include a statement identifier with a statement that does not require an identifier.

4.3.8 Numbers

- A real number is a number with a decimal point and/or a fractional part behind the decimal point.
- A single plus (+) or minus (-) sign can precede any number or exponent. Use of the plus sign is optional.
- Numbers must not contain blanks or commas.
- If a number contains a decimal point and/or an exponent, ADAMS/Solver interprets it as a real number.
- ADAMS/Solver stores real numbers in a format that retains approximately fifteen significant digits in 32-bit machines.
- If a number does not contain either a decimal point or an exponent, ADAMS/Solver interprets it as an integer. You may assign an integer value when ADAMS/Solver expects a real argument. However, if you do this, ADAMS/Solver converts the integer to a real number before storing it.
• Integer values input to ADAMS/Solver must be greater than $-2^{31} - 1$ and less than $+2^{31} - 1$.

• You may assign a real number value when ADAMS/Solver expects an integer if the fractional part of the real number is zero. If you do this, ADAMS/Solver converts the real number to an integer before storing it.

• You may input exponents for real numbers and for integers you assign to real arguments. The exponent indicates an integer power of ten by which ADAMS/Solver is to multiply the mantissa. To indicate the exponent, immediately follow the number with an $E$ (this is consistent with the $E$ in FORTRAN). Then follow the $E$ with the positive or negative integer exponent. The allowable range for exponents depends on the particular computer on which you are running ADAMS/Solver. On the VAX/VMS, exponents must be integers greater than -38 and less than +38. Examples of numbers with valid exponents are below.

$$0.314159E1$$
$$314.159E01$$
$$314.159E-02$$
$$314159E+2$$

• Internally, ADAMS/Solver uses radians for angular units. ADAMS/Solver includes a degrees-to-radians conversion factor to allow for easy conversion of degree quantities to radian quantities. Put a $D$ immediately following a number that you want ADAMS/Solver to scale by that conversion factor. ADAMS/Solver multiplies the number by $\frac{\pi}{180}$ before storing it. You may use this conversion factor for any number in the data set, including numbers in function expressions. But this conversion factor is especially useful when you are assigning values to arguments that expect angular quantities. For example, ADAMS/Solver converts the argument

$$\text{REULER} = 90D, 180.0D, 2.70E+2D$$

to

$$\text{REULER} = 1.5708, 3.1416, 4.714$$

before storing the information. $D$ does not specify double precision in ADAMS/Solver as it does in FORTRAN.

• The following errors in the use of numbers cause ADAMS/Solver to issue an error message and stop execution at the end of the input phase:
  - You include a fractional part other than zero in a real number you input for an integer argument value.
  - You include a blank or a comma in a number.
4.3.9 Statements

- A statement must begin in one of the first five columns of the line. If it does not, ADAMS/Solver treats it as a comment and does not read it.
- Statements can be defined only between column 1 and column 80.
- To continue a statement that is longer than eighty columns, use an ampersand (&) after the last statement character in the line to be continued or use a comma (,) as the first character in all continuation lines (see Section 4.3.5, “Continuations”). If you use the ampersand, the last statement character can be in the seventy-ninth column at most because the ampersand would then have to be in the eightieth column.
- ADAMS/Solver can read two or more statements on the same line if they are separated by semicolons (;).
- ADAMS/Solver accepts both upper- and lower-case letters for statement names.
- Because the TITLE statement is read but not executed, it can include any combination of printable characters and blank spaces.
- The following errors in the use of statements cause ADAMS/Solver to issue an error message and stop execution at the end of the input phase:
  - You do not include all required arguments in a statement.
  - You put two or more statements on the same line without semicolons separating them.
  - You continue a statement on the following line without either an ampersand at the end of the first line or a comma at the beginning of the following line.
Chapter 4. ADAMS/Solver Statements

4.4 Statement Groups

This section groups all ADAMS/Solver statements according to their functions and explains the use of each statement.

The names of the statement groups are as follows:

- Data Set Delimiter
- Inertial Data
- Geometry
- Constraints
- Forces
- Generic Systems Modeling
- Aggregate Element
- Reference Data
- Analysis Parameters
- Output

4.4.1 Data Set Delimiter

The data set delimiter statements control data entry.

`END` Indicates the end of your data set.

`TITLE` Begins and describes your data set.

4.4.2 Inertial Data

The inertial data statement defines inertia properties for bodies in a system. At this time, the only inertial data statement is the `PART` statement which defines a rigid body.

It is also possible to model inertia effects with other statements, such as `DIFF`, `SFORCE`, `GFORCE`, `GSE`, and `LSE`.

`PART` Defines the inertia properties of a rigid body and its initial position, orientation, and velocity.
4.4.3 Geometry

The geometry statements are used for graphic display and for specifying markers.

**GRAPHICS** Creates a three-dimensional graphic for display on a graphics terminal.

**MARKER** Identifies a fixed location and orientation with respect to LPRF or identifies a floating location and orientation on a part that is the point of application for certain forces and constraints. You may use markers to define any point that is significant to the analysis or to the display of your model.

4.4.4 Constraints

The constraint statements define joints and connects between parts as well as motion at joints. Constraints limit one or more degrees of freedom between parts.

Table 3 lists all the constraints available in ADAMS/Solver. Some of the constrains are time dependent and others are time independent.

<table>
<thead>
<tr>
<th>Time-Dependent Constraints</th>
<th>Time-Independent Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recognizable Combinations</td>
</tr>
<tr>
<td>MOTION</td>
<td>CVCV</td>
</tr>
<tr>
<td>UCON</td>
<td>GEAR</td>
</tr>
<tr>
<td></td>
<td>JOINT</td>
</tr>
<tr>
<td></td>
<td>PTCV</td>
</tr>
</tbody>
</table>

Time-dependent constraints can be implemented with **MOTION** and **UCON** statements. ADAMS/Solver has two groups of time-independent constraints: recognizable combinations and primitive combinations. In general, recognizable combinations have physical counterparts, and primitive combinations do not. Recognizable combinations can be implemented with the **CVCV**, **GEAR**, **JOINT**, and **PTCV** statements, and primitive combinations can be implemented with the **COUPLER**, **JPRIM**, and **UCON** statements.
A **UCON** statement defines a user-written constraint. It is the only constraint that can be either time dependent or time independent. The **UCON** statement can model holonomic as well as nonholonomic constraints.

The constraints available in ADAMS/Solver are summarized as follows:

- **COUPLER**: Relates the translational and/or the rotational motion of two or three joints.
- **CVCV**: Defines a curve-to-curve constraint which restricts a planar curve on one part to be in contact with, and tangent to, a planar curve on a second part.
- **GEAR**: Defines a gear pair. Examples include spur, helical, planetary, bevel, and rack-and-pinion gear pairs.
- **JOINT**: Describes a physically recognizable combination of constraints such as constant and velocity, cylindrical, fixed, Hooke, planar, rack-and-pinion, revolute, screw, spherical, translational, and universal joints.
- **JPRIM**: Describes a joint primitive, which constrains one, two, or three degrees of either translational or rotational freedom.
- **MOTION**: Determines a translational or a rotational motion in a joint as a function of time. In ADAMS/Solver you can apply motion to a translational, a revolute, or a cylindrical joint.
- **PTCV**: Defines a point-to-curve constraint which restricts a fixed point defined on one part to lie on a curve defined on a second part.
- **UCON**: Allows you to define a system constraint as a function of time and as many as thirty of the part variables.
4.4.5 Forces

You can use force statements to specify flexibility in a system, represent environmental fields or disturbances, as well as to define non-idealized joints. Forces in ADAMS/Solver must act at markers. Any number of forces and torques (translational and rotational forces) can act at a specific marker. ADAMS/Solver will vectorially summarize all of the forces and torques acting on the marker.

The force definition statements available in ADAMS/Solver are summarized as follows:

- **ACCGRAV**: Specifies the magnitude and direction of the acceleration field due to gravity. In conjunction with the part masses, the gravitational field defines a body force at the center of gravity of each part.
- **BEAM**: Defines a massless beam with a uniform cross section.
- **BUSHING**: Defines a massless bushing with linear stiffness and damping properties.
- **FIELD**: Defines a translational force and a rotational force between two markers.
- **GFORCE**: Defines a generic force element consisting of three orthogonal translational force components and three orthogonal torque components.
- **NFORCE**: Creates a multi-point force element which establishes linear force-displacement (stiffness) and/or force-velocity (damping) relationships between up to 351 markers in the model.
- **SFORCE**: Applies a rotational or a translational force between two markers whose magnitude depends on any combination of system variables, user-defined variables, and time.
- **SPRINGDAMPER**: Defines a massless spring and/or damper with linear stiffness and damping properties.
- **VFORCE**: Creates a generic translational vector force element that is specified using three orthogonal components in a user-specified coordinate system.
- **VTORQUE**: Defines a generic rotational force element (i.e., a torque) that is specified using three orthogonal components in a user-specified coordinate system.
### 4.4.6 Generic Systems Modeling

There are several statements in ADAMS/Solver that provide you with capability of defining a system or subsystem in terms of a set of equations that you have derived yourself or obtained elsewhere.

ADAMS/Solver allows you to simply append these equations to the set of equations defining the mechanical system. Complete coupling of the user-defined equations to the mechanical system is allowed, i.e., the user-defined equations may depend on the instantaneous state of the mechanical system and vice-versa. ADAMS/Solver will solve all equations simultaneously.

User-defined equation sets may be defined using the **DIFF**, **GSE**, **LSE**, **TFSISO**, and **VARIABLE** statements.

These generic systems modeling statements create one or more general differential and/or algebraic equations. They allow you to define your own variables through differential/algebraic equations. Other ADAMS/Solver statements and user-written subroutines can access these variables. These elements are extremely useful in modeling subsystems, like control systems mathematically.

- **DIFF**: Defines a differential equation that describes a user-defined variable in terms of its time derivative.
- **GSE**: Defines a system of explicit differential and (optionally) algebraic equations in state-space form. **ARRAY** statements are used to specify inputs, outputs, and statements.
- **LSE**: Defines a system of constant coefficient, explicit, differential, and algebraic equations in the classic state-space format when used with associated **ARRAY** and **MATRIX** statements.
- **TFSISO**: Defines a single-input, single-output transfer function as a ratio of two polynomials in the Laplace domain when used with associated **ARRAY** statements.
- **VARIABLE**: Defines a scalar algebraic equation for independent use or as part of the **PINPUT**, **POUTPUT**, or **ARRAY** statements.
4.4.7 Aggregate Element

Aggregate element statements create an element composed of several other ADAMS/Solver statements. At this time, the only aggregate element statement is the TIRE statement which creates a collection of statements to represent a tire, including a PART, JOINT, MARKER, GFORCE, and GRAPHICS.

TIRE  Defines a tire element in ADAMS/Solver. A TIRE can model both vehicle-terrain force interactions as well as tire rotational effects. The tire is an aggregate element, i.e., it represents a complexity that is composed of a number of more basic ADAMS/Solver statements: PART, MARKER, JOINT, GRAPHICS, and GFORCE. These entities are automatically created by ADAMS/Solver whenever a TIRE statement is defined.

4.4.8 Reference Data

The reference data statements provide access to user-defined or computed data that is obtained from a simulation.

ARRAY  Enters an array of real numbers, or defines an input, output, state, or initial conditions array for an LSE, GSE, or TFSISO.

CURVE  Defines a three-dimensional parametric curve that may be referenced by PTCV, CVCV, or GRAPHICS statements and by function expressions.

MATRIX  Provides the means for inputting a matrix of numerical values. It is used primarily to support other statements such as CURVE, LSE, and NFORCE.

PINPUT  Defines a list of VARIABLE statements that ADAMS/Solver recognizes as system input during a LINEAR/STATEMAT analysis.

POUTPUT  Defines a list of VARIABLE statements that ADAMS/Solver recognizes as system output during a LINEAR/STATEMAT analysis.

SPLINE  Defines discrete data that can be interpolated by the AKISPL and CUBSPL functions and the AKISPL and CUBSPL data access subroutines.

STRING  Defines a character string that may be referred to later in the execution of ADAMS/Solver.
Chapter 4. ADAMS/Solver Statements

4.4.9 Analysis Parameters

The analysis parameter statements control parameters used during analysis.

- **BDFPAR** Controls the corrector algorithm for the BDF (Backwards-Differentiation-Formula) integrator accessible from the DSOLVER and DYNAMICS statements.
- **CPAPAR** Controls the coordinate partitioning algorithm for the ABAM (Adams-Bashforth and Adams-Moulton) integrator.
- **DEBUG** Allows you to output data for debugging your data set.
- **DSOLVER** Selects either the BDF integrator or the ABAM integrator.
- **DSTIFF** Specifies the integration error tolerance and other parameters for a dynamic analysis using the DASSL integration algorithm.
- **DYNAMICS** Provides an automatic way to select an integrator and an easy way to select a consistent set of error tolerances.
- **EQUILIBRIUM** Controls the error tolerance and other parameters for static equilibrium and quasi-static equilibrium analyses.
- **GSTIFF** Specifies error tolerances and other parameters for dynamic analyses using the Gear stiff integration algorithm.
- **IC** Specifies error tolerances and other parameters for the analysis of the initial conditions and reconciling DSTIFF integrator output.
- **INTPAR** Controls parameters for the ABAM integrator.
- **KINEMATICS** Controls the error tolerance and other parameters for a kinematic analysis.
- **SENSOR** Senses a predefined event and effects a set of simulation controls when the event occurs.
- **UNITS** Sets the appropriate units for an ADAMS/Solver analysis. This ensures that force computations, both on input and on output, will correctly conform to the user's choice of length, mass, and time units.
- **WSTIFF** Specifies error tolerances and other parameters for dynamic analyses using Wielenga’s stiff integration algorithm. This algorithm is now unsupported and has been replaced with the new BDF algorithm, which is accessible from the DSOLVER and DYNAMICS statements.
Please discontinue use of the \texttt{WSTIFF} statement.
4.4.10 Output

The output statements control the format and content of the ADAMS/Solver output files.

**FEMDATA**
Produces a special data file containing all reaction forces and all applied forces (except gravity) acting on each part.

**LIST/NOLIST**
Allows or suppresses output of your input data. You can use combinations of LIST and NOLIST to output any part of your input data set.

**MREQUEST**
Indicates multiple sets of simulation results that you want ADAMS/Solver to output.

**OUTPUT**
Controls the generation of Request and Graphics files. In addition, this statement controls the form, format, coordinates, filtering, and scaling of request data in the Tabular Output File.

**REQUEST**
Indicates a set of simulation results that you want ADAMS/Solver to output.

**RESULTS**
Creates the Results file, which may include all the simulation output from ADAMS/Solver.
4.5 A Comparison of Integrators in ADAMS/Solver

The integration algorithms invoked by the DSTIFF and GSTIFF commands and statements are all modified implementations of the Backward Differentiation Formula (BDF) method for the solution of systems of DAEs. The Backward Differentiation Formula provides both a stable numerical scheme for stiff DAEs and an efficient technique for general purpose applications.

The DSOLVER statement and command invokes either a BDF integrator with variable leading coefficients for integrating a set of DAE’s or the ABAM integrator for integrating a reduced set of ODEs obtained by coordinate partitioning. The BDF integrator is better for numerically stiff problems (i.e., models with a mix of high and low frequencies.) The ABAM integrator may be better for simulation of systems undergoing sudden changes or for simulations of systems with high active frequencies.

Table 4, along with the corresponding footnotes, defines the default values for all of the arguments on the DSTIFF and GSTIFF integrator commands and statements.

<table>
<thead>
<tr>
<th>Options</th>
<th>DSTIFF</th>
<th>GSTIFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>$10^3$</td>
<td>$10^3$</td>
</tr>
<tr>
<td>HINIT$^{(1)}$</td>
<td>$1/20 \times$ the output step</td>
<td>$1/2 \times$ the output step</td>
</tr>
<tr>
<td>HMIN</td>
<td>None$^{(2)}$</td>
<td>$10^6 \times$ HMAX</td>
</tr>
<tr>
<td>HMAX</td>
<td>Infinite$^{(3)}$</td>
<td>The output step</td>
</tr>
<tr>
<td>KMAX$^{(4)}$</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>MAXIT</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>PATTERN$^{(5)}$</td>
<td>first-fifth-ninth</td>
<td>first-fifth-ninth</td>
</tr>
<tr>
<td>RECONCILE</td>
<td>On or off</td>
<td>n/a$^{(6)}$</td>
</tr>
<tr>
<td>WATCH</td>
<td>Nine options</td>
<td>n/a$^{(6)}$</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Restore default values</td>
<td>Restore default values</td>
</tr>
<tr>
<td>LIST</td>
<td>List current values</td>
<td>List current values</td>
</tr>
</tbody>
</table>

$^{(1)}$ The output step, which determines the default value for HINIT is defined by the value for the END or DURATION argument and by the value for the STEPS or DTOUT argument on the SIMULATE command. See Section 3.8.10 in the ADAMS/Solver Reference Manual.

$^{(2)}$ No maximum step size is imposed. The values of the state vector at the output points are interpolated from the computed solution.

$^{(3)}$ The default value shown for KMAX is the maximum acceptable value for the maximum order of the integrator.

$^{(4)}$ Setting the first value of PATTERN to FALSE has no effect on the GSTIFF integrator. The Jacobian matrix will always be recomputed for the first iteration of the corrector. However, the DASSL
The integrator will attempt to use the Jacobian matrix inherited from the previous time step. Note that, in addition to the specified evaluations, the Jacobian matrix will be recomputed whenever the integrator finds that a new one is required. This option is not available for the given integrator.

The DSOLVER statement and command allow for selection of a BDF or ABAM integrator. The monitoring and control options are set by the INTPAR statements and commands. Default values of these arguments are given in Table 5.

Table 5. Default Values for the Monitoring and Command Options for the ABAM Integrator

<table>
<thead>
<tr>
<th>Options</th>
<th>INTPAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTINUE</td>
<td>OFF</td>
</tr>
<tr>
<td>HMAX</td>
<td>Infinite</td>
</tr>
<tr>
<td>INTERPOLATE</td>
<td>ON</td>
</tr>
<tr>
<td>KMAX</td>
<td>12</td>
</tr>
<tr>
<td>NTABS_ERROR</td>
<td>0.0005</td>
</tr>
<tr>
<td>NTREL_ERROR</td>
<td>0.0005</td>
</tr>
<tr>
<td>RECONCILE</td>
<td>ALL</td>
</tr>
<tr>
<td>WATCH</td>
<td>Six options</td>
</tr>
<tr>
<td>STEP_MAXIMUM</td>
<td>1E6</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Restore default values</td>
</tr>
<tr>
<td>LIST</td>
<td>List current values</td>
</tr>
</tbody>
</table>
Chapter 5
Customizing ADAMS/Solver

Introduction
ADAMS/Solver is unique amongst software products in the sense that it allows you to customize it to solve special purpose problems involving non-standard phenomena. You may enhance the simulation and modeling capabilities of ADAMS/Solver in three distinct ways:

• Using function expressions to define non-standard physical phenomena in your ADAMS/Solver model definition input data set

• Writing your own subroutines to describe physical phenomena that are not formally supported in ADAMS/Solver. You can compile and link these modules to the standard ADAMS/Solver libraries and create your own customized version of ADAMS/Solver

• Using Callable ADAMS/Solver allows you to embed it into other general purpose software packages, e.g., customized pre- and postprocessors and optimization tools.
5.1 Function Expressions

ADAMS/Solver contains a rich library of F-77 intrinsic, kinematic, and kinetic functions. These may be accessed directly in the model definition input file by setting the instantaneous value(s) of selected arguments of statements to be equal to function expressions.

5.1.1 What is A Function Expression

A function expression is a single FORTRAN-like expression that evaluates a single value. This value may be a function of system variables such as displacements between markers, velocities, accelerations, and current simulation time. You can write function expressions such as arguments in ADAMS/Solver modeling elements. This allows you to specify certain aspects of model elements such as force-displacement relationships in a spring-damper in a very easy but general manner. The function expression capability enables you to model a wide variety of phenomena, without writing user-written subroutines.

5.1.2 The Advantages of Function Expressions

Function expressions allow for great flexibility in model definition. You have the power to define dynamic relationships amongst model components without using user-written subroutines. The standard ADAMS/Solver executable is able to handle a diverse set of models in this manner. You are not required to create a specialized executable of ADAMS/Solver which contains special purpose subroutines that you had to write. The entire model is described in one data set, making the model more understandable. If you can develop a symbolic expression and are familiar with FORTRAN syntax, you will have the knowledge to write a correct function expression.
5.1.3 The Uses of Function Expressions

The following statements use the function-expression capability as part of the element definition:

**MOTION**
In this statement, use the function expression to define the explicit dependence of a joint displacement, velocity, or acceleration, on time. *The function expression is therefore limited and dependent on system constants and simulation time only.*

**SFORCE, VFORCE, VTORQUE, GFORCE**
In these statements, use the function expression to define the mathematical dependence of the element outputs, (i.e., forces and torques) on the current state of the system. Therefore, for these statements, there is no restriction on the functions on which the force/torque value is dependent. Such a general capability allows for easy definition of nonlinear force-deformation relationships, including friction, energy loss due to damping, and control feedback forces.

**DIFF**
In this statement, use the function expression to define an implicit or explicit first-order differential equation or an implicit algebraic equation. This capability allows you to define your own equations for a subsystem for which you already have a mathematical model.

**VARIABLE**
In this statement, use the function expression to define an algebraic equation. This capability allows you to optimize the efficiency of computations by storing the intermediate calculations as state variables. They can then be referred to by other ADAMS/Solver elements or function expressions.

**SENSOR**
In this statement, use the function expression to define the instantaneous value of an event that is being sensed.

**REQUEST**
In this statement, use the function expression to define specific user-defined simulation output that is to be written to ADAMS/Solver output files.
5.1.4 Syntax and Notation

A function expression may be composed of any valid combination of simple constants, operators, parameters, and available standard ADAMS/Solver functions.

5.1.4.1 Simple Constants

Function expressions may include integers and real numbers only. Complex numbers are not supported at this time. Any legal number is accepted by the function expression. The definition of a legal integer and a legal real number is quite machine dependent, and may therefore change from system to system. Typically in ADAMS/Solver, an integer is limited to having an absolute value less than $2^{31}-1$. A real number in ADAMS/Solver, is typically limited to having an absolute value less than $10^{26}$.

5.1.4.2 Operators

ADAMS/Solver allows the standard FORTRAN-77 set of operators. The operators and their sequence of evaluation are shown in Table 6.

Table 6. Procedure Rules for Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>Exponentiation</td>
<td>1</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>4</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>5</td>
</tr>
</tbody>
</table>

ADAMS/Solver executes exponentiation (**), before all other operators and executes multiplication (*) and division (/) before addition (+) and subtraction (-). When a statement has operators of the same priority, ADAMS/Solver executes them from left to right unless it is in an explicit violation of ANSI standards. You can use parentheses to alter the precedence of operators. For example, in the equation

\[
\text{FUNCTION} = (1 - \text{TIME}) * 30 / \Pi
\]

ADAMS/Solver subtracts “TIME” from “1” before it performs multiplication of the result with “30” and division by “\Pi” (3.1415...). The precedence rules in ADAMS/Solver are consistent with the precedence rules in FORTRAN. For more information, refer to an ANSI FORTRAN reference manual.
5.1.4.3 Functions

A function expression may contain built-in ADAMS/Solver constants and “standard”
functions such as \texttt{TIME} and \texttt{STEP(...)}. The standard ADAMS/Solver functions are
described in subsequent sections.

Some functions require input parameters. You may specify these parameters as
function sub-expressions themselves or as constants. Some functions have optional
parameters, i.e., these parameters need not be defined for complete specification of a
function. If an optional parameter occurs in the middle of a parameter sequence, you
must use zero as a placeholder for that particular optional parameter. If the optional
parameter occurs at the end of a parameter sequence, you may ignore it entirely.
ADAMS/Solver will give unspecified optional parameters default values prior to
evaluating the function. These default values will be explained in the context of the
individual functions.

5.1.4.4 Continuation Lines

Function expressions may span several lines. To indicate that a line is a continuation of
a function expression, start the new line with a comma (\texttt{,}). You can break the
expression anywhere except in the middle of a number, in the middle of a name, or
between a function and its left bracket. If you break the expression at a comma that is
part of the expression, you must use both the expression comma and the continuation
comma. There is no limitation to the number of lines that you may use to define a
complex function expression.

5.1.4.5 Blanks

A function expression can contain any number of blank spaces. They are often used to
improve readability of the function expression. Note that unlike the regular statement
syntax, five consecutive blank spaces in an expression do not terminate input of the
expression. The following restrictions exist on the usage of blanks:

\begin{itemize}
  \item You cannot put a blank space in the middle of a number.
  \item ADAMS/Solver does not accept a blank space between a function and its left bracket.
\end{itemize}
5.1.5 Limitations

ADAMS/Solver imposes some limitations in the definition of functions.

5.1.5.1 Access of Floating Markers

Function expressions belonging to REQUEST and SENSOR statements may reference floating markers. The use of floating markers in all other function expressions is not allowed. ADAMS/Solver will flag references to floating markers as errors in these cases.

5.1.5.2 Nesting of Functions

Functions, sub-expressions or operations may be nested only ten levels deep. For example: \( \text{FUNCTION} = f_1(f_2(f_3(f_4))) \), where \( f_1, f_2, f_3, f_4 \) are function sub-expressions, is a syntactically legal function definition. However

\[ \text{FUNCTION} = f_1(f_2(\ldots (f_{11})\ldots) \) is not a legal definition, since sub-expressions have been nested eleven levels deep.

5.1.5.3 Limitations on Number of Symbols a Single Function Expression Can Have

Function expressions must not use more that one thousand symbols. This includes operators, brackets, functions, numbers etc.

5.1.5.4 Limitations on Number of Standard ADAMS/Solver Elements to Which a Function Can Refer

A function is limited to being dependent on a maximum of 25 elements of each type. Therefore, each expression may be a function of a maximum of twenty-five PARTS, twenty-five FORCES, twenty-five DIFFs, twenty-five VFORCES, etc.
### 5.1.6 Overview of Function Expressions Available in ADAMS/Solver

Table 7 explains the functions available in ADAMS/Solver.

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Brief Explanation</th>
<th>Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORTRAN-77 Intrinsic Functions</strong></td>
<td>Emulate FORTRAN math. run-time libraries</td>
<td>AINT, ANINT, NINT, ABS, MOD, SIGN, DIM, MAX, MIN, SQRT, EXP, LOG, LOG10, SIN, COS, TAN, ASIN, ACOS, ATAN, ATAN2, SINH, COSH, TANH</td>
</tr>
<tr>
<td><strong>Pre-defined Parameters</strong></td>
<td>Commonly used constants or variables</td>
<td>TIME, PI, DTOR, RTOD, MODE</td>
</tr>
<tr>
<td><strong>Kinematic Functions</strong></td>
<td>Return current kinematic state of the model</td>
<td>DM, DX, DY, DZ, AX, AY, AZ, PSI, THETA, PHI, YAW, PITCH, ROLL, VM, VX, VZ, VR, WM, WX, WY, WZ, ACCX, ACCY, ACCZ, WDTM, WDX, WDY, WDTZ</td>
</tr>
<tr>
<td><strong>Generic Force Functions</strong></td>
<td>Return the net force acting at a marker</td>
<td>FM, FX, FY, FZ, TM, TX, TY, TZ</td>
</tr>
<tr>
<td><strong>Element Specific Force Functions</strong></td>
<td>Return force at an attachment point of the element</td>
<td>BEAM, BUSH, FIELD, SPDP, SFORCE, GFORCE, VFORCE, VTORQ, PTCV, CVCV, JOINT, JPRIM, MOTION</td>
</tr>
<tr>
<td><strong>System Element Functions</strong></td>
<td>Return current state of system modeling element</td>
<td>PINVAL, POUVAL, VARVAL, ARYVAL, DIF, DIF1</td>
</tr>
<tr>
<td><strong>Orthogonal Functions</strong></td>
<td>Allow approximating functions</td>
<td>POLY, CHEBY, FORCOS, FORSIN</td>
</tr>
<tr>
<td><strong>Piecewise Interpolation</strong></td>
<td>Interpolates data points</td>
<td>AKIMA, CUBSPL, CURVE</td>
</tr>
<tr>
<td><strong>Special ADAMS Functions</strong></td>
<td>Commonly used &quot;non-standard&quot; functions</td>
<td>BISTOP, HAVSIN, IMPACT, SHF, STEP</td>
</tr>
<tr>
<td><strong>Arithmetic IF Function</strong></td>
<td>Allows branching IF in expression</td>
<td>IF</td>
</tr>
</tbody>
</table>
5.2 User-Written Subroutines

As an alternate to writing function expressions in your model definition input files, you may write your own subroutines and link these with ADAMS/Solver. User-written subroutines, while a little more difficult to use, provide a degree of generality and flexibility that function expressions do not.

- You have the full power of FORTRAN at your disposal to describe the mathematical equations defining the non-standard phenomena.
- You do not lose any efficiency or decrease the speed of simulation by linking in user-written subroutines.
- In our experience, incorrectly coded user-written subroutines are the most difficult of all models to debug. You should therefore be a cautious user of user-written subroutines.

5.2.1 Guidelines for Writing Subroutines

Use these general guidelines when you write user-written subroutines.

5.2.1.1 Using Subroutine Arguments

The subroutine input and output arguments are in lower-case letters in Chapter 2, “User-Written Subroutines,” in the ADAMS/Solver Subroutines Reference Manual to indicate that you can choose any legal FORTRAN variable for the argument name. The input arguments are first followed by the output arguments.

Input arguments provide you with values ADAMS/Solver assigns from information in the corresponding command or statement, or from calculations indicating the current system state. Do not change the values of these input arguments. Any attempt to change them will have unpredictable results.

Output arguments provide ADAMS/Solver with information you have derived. Unless explicitly stated otherwise, you should always supply a value for an output argument.

Because FORTRAN uses argument order to distinguish arguments from each other, you must not change the order of arguments when passing them to or when receiving them from subroutines. Also, remember to use a variable of the proper type and of the proper dimension for each argument.
5.2.1.2 Declaring Floating-Point Variables

You need to declare your floating-point variables the same way ADAMS/Solver does. This ensures that the values are properly transferred between your subroutines and ADAMS/Solver subroutines.

The length of floating-point variables varies with the machine, but ADAMS/Solver uses variables that are about sixty-four bits (eight bytes) long.

On most machines, such as VAX, Sun, and Hewlett-Packard, ADAMS/Solver uses DOUBLE PRECISION, floating-point variables. Since an untyped floating-point variable usually defaults to single-precision (i.e., REAL), you must explicitly declare all your floating-point variables to be DOUBLE PRECISION. We have done this in all our example subroutines, for instance.

On a few machines, such as Cray, ADAMS/Solver uses REAL variables because single-precision variables are already about sixty-four bits long. On these machines, declare your floating-point variables to be REAL.

5.2.1.3 Using Parameters to Make Subroutines More General

One input argument ADAMS/Solver always supplies to user-written subroutines is the parameter values from the FUNCTION=USER() argument on the corresponding statement or command. If you use FUNCTION=USER (4.0, 5.0, 6.0), for instance, ADAMS/Solver will pass a PAR array containing the values 4.0, 5.0, and 6.0, and will pass NPAR with a value of 3.

You can use this feature to make your subroutines more flexible. If your calculations depend on the distance between markers 101 and 201, for instance, you could write these identifiers directly in your subroutine. But if you wanted to change the marker identifiers, or use the subroutine with a different model, you would have to edit the FORTRAN file, recompile, and relink with ADAMS/Solver.

You can avoid these time-consuming changes by putting the marker identifiers in the FUNCTION=USER() argument and then using them as variables in your subroutine. If you wish to change them, you only need to change the FUNCTION=USER() argument in your statement or command, then re-run ADAMS/Solver with the same executable. Many of the example subroutines illustrate this technique.
5.2.1.4 Avoid Discontinuities

When you use a subroutine to define motions, sensors, or forces in a model, you must ensure that these functions are continuous. This implies that if the defined function were to be plotted using pen and paper, the entire function could be drawn without lifting the pen from the paper.

Discontinuous functions are difficult to handle because most numerical integration schemes require, and therefore assume, that system equations are inherently continuous. If you violate this rule, ADAMS/Solver may produce unexpected results such as integration failure at the discontinuity. Discontinuous functions are created most often when using IF statements in FORTRAN to piece together separate functions. You must be especially careful when using this type of logic to ensure that the complete function is indeed continuous.

5.2.1.5 FORTRAN Input/Output Units

If you need to read or write files from your user subroutines, use FORTRAN units 90 through 99. ADAMS/Solver reserves these units for your use. ADAMS/Solver will not use these units, even in a future release.

5.2.2 User-Written Subroutine Classifications

User-written subroutines are of one of the following three types:

- Driver
- Evaluation
- Restart

The driver user-written subroutine allows you to control ADAMS/Solver from a subroutine, rather than by interactive commands or command files.

The evaluation user-written subroutines allow you to use FORTRAN-77 features to define functions not otherwise available with ADAMS/Solver and to tailor ADAMS/Solver to individual needs.

The restart user-written subroutines ensure that SAVE and RELOAD commands also restart your other user-written subroutines.
The user-written subroutines you compile and link act as interfaces between your data set and the utility subroutines you want to use. For example, from a driver user-written subroutine, you can call the following three types of utility subroutines: the control subroutines, the access subroutines, and the function subroutines. From the evaluation user-written subroutines, you can call the access subroutines, the function subroutines, and (in some cases) the setup subroutines. In other words, you can call the access subroutines and the function subroutines from either type of user-written subroutine, but you can call the control subroutines only from the driver subroutine and you can call the setup subroutines only from certain evaluation subroutines (see Table 8).

5.2.2.1 Driver

CONSUB and DRVINT are the only driver user-written subroutines. (For information about DRVINT, refer to Section 4.1.1 in the ADAMS/Solver Subroutines Reference Manual.) The term driver means that CONSUB and DRVINT can initiate and regulate an ADAMS/Solver simulation. ADAMS/Solver invokes CONSUB when you issue a CONTROL command. By calling the control utility subroutines, CONSUB can automatically modify an ADAMS/Solver model, request one or more simulations and process output at one or more points in time. CONSUB gives you great flexibility in automation of repetitive data set changes and ADAMS/Solver analyses.

The purpose of a driver user-written subroutine is to further automate the analysis and modification of ADAMS/Solver models. This is a very powerful tool; however, most ADAMS/Solver users will not need to use a driver user-written subroutine.

5.2.2.2 Evaluation

The evaluation subroutines are user-written FORTRAN subroutines that allow you to do complex mathematical operations to model nonstandard phenomena with ADAMS/Solver. A separate defining statement in your ADAMS/Solver data set is necessary to define constants passed to each evaluation subroutine you are using. Table 8 shows the names of the statements that invoke the evaluation subroutines as well as their corresponding subroutine names and functions.
<table>
<thead>
<tr>
<th>Statement Names</th>
<th>Subroutine Names</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURVE</td>
<td>CURSUB</td>
<td>Allows you to define your own curve representation.</td>
</tr>
<tr>
<td>DIFF</td>
<td>DIFSUB</td>
<td>Allows you to define differential equations for variables either explicitly or implicitly.</td>
</tr>
<tr>
<td>FIELD</td>
<td>FIESUB</td>
<td>Allows you to calculate field components that are nonlinear functions of the relative displacements and the relative velocities of the field I and J markers.</td>
</tr>
<tr>
<td>GFORCE</td>
<td>GFOSUB</td>
<td>Allows you to calculate force and torque components when the expressions are lengthy and awkward or when the definition of the displacement requires additional features of FORTRAN-77.</td>
</tr>
<tr>
<td>GSE</td>
<td>GSESUB, GSEX, GSEXU, GSEYX, GSEYU</td>
<td>Allows you to define systems of differential and algebraic equations. GSEX, GSEXY, GSEYX, and GSE allow you to define partial derivatives of the equations.</td>
</tr>
<tr>
<td>MOTION</td>
<td>MOTSUB</td>
<td>Allows you to calculate time-dependent displacements when the expression is lengthy and awkward or when the definition of the displacement requires additional features of FORTRAN-77.</td>
</tr>
<tr>
<td>REQUEST</td>
<td>REQSUB</td>
<td>Allows you to request output when the expression is lengthy and awkward or when the definition of the output requires additional features of FORTRAN-77.</td>
</tr>
<tr>
<td>SENSOR</td>
<td>SENSUB</td>
<td>Allows you to sense functions when the expression is lengthy and awkward or when the definition of your function requires additional features of FORTRAN-77.</td>
</tr>
<tr>
<td>SFORCE</td>
<td>SFOSUB</td>
<td>Allows you to define single-component forces when the expression is lengthy and awkward or when the definition of your force requires additional features of FORTRAN-77.</td>
</tr>
<tr>
<td>TIRE</td>
<td>TIRSUB</td>
<td>Allows you to define your own tire representation.</td>
</tr>
<tr>
<td>UCON</td>
<td>UCOSUB</td>
<td>Allows you to define constraint combinations not available through the standard constraints.</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>VARSUB</td>
<td>Allows you to calculate algebraic variable when the expression is lengthy and awkward or when the definition of your variable requires additional features of FORTRAN-77.</td>
</tr>
<tr>
<td>VFORCE</td>
<td>VFOSUB</td>
<td>Allows you to calculate force components when the expression is lengthy and awkward or when the definition of your force requires additional features of FORTRAN-77.</td>
</tr>
</tbody>
</table>
5.2.2.3 Restart

The restart user-written subroutines allow you to save and later restart ADAMS/Solver simulations. The restart subroutines, SAVSUB and RELSUB, allow you to save and reload required data to restart your other user-written subroutines. While executing a SAVE command, ADAMS/Solver calls the SAVSUB. While executing a RELOAD command, ADAMS/Solver calls the RELSUB.

You need only to write a SAVSUB and RELSUB when

- you have other user-written subroutines,
- those subroutines store data internally from call to call, and
- you intend to use the SAVE and RELOAD commands.

Suppose, for example, you have written an SFOSUB that initially reads in some coefficients from a file and saves them in an internal array. In order to SAVE and RELOAD a simulation, you must do the following: write a SAVSUB to store the coefficients in the ADAMS/Solver Save File and write a RELSUB to restore them to the array. This way, when you reload a simulation, ADAMS/Solver restores your SFOSUB data as well. Table 9 shows the names of the command that invoke the restart subroutines as well as their corresponding subroutine names and functions.

Table 9. Commands that Invoke the Restart Subroutines, and Their Corresponding Subroutine Names and Functions

<table>
<thead>
<tr>
<th>Command Names</th>
<th>Subroutine Names</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELOAD</td>
<td>RELSUB</td>
<td>Reloads necessary information for restarting user-written subroutines.</td>
</tr>
<tr>
<td>SAVE</td>
<td>SAVSUB</td>
<td>Stores necessary information for restarting user-written subroutines.</td>
</tr>
</tbody>
</table>
5.3 Utility Subroutines

ADAMS/Solver provides you with a library of utility subroutines that perform several important functions for you. You will need to use these subroutines to perform some of the following operations:

- Access of data defined in the model definition input data set
- Access of current state information from ADAMS/Solver
- Interpolate through raw data using standard curve fitting techniques
- Evaluate “standard” functions
- Set up functional dependencies

In general, follow the same guidelines for utility subroutines as for user-written subroutines. See Section 5.2.1, “Guidelines for Writing Subroutines” in this chapter and Chapter 3, “Utility Subroutines” in the ADAMS/Solver Subroutines Reference Manual for more information.

5.3.1 Classification

Four types of utility subroutines are available in ADAMS/Solver:

- Execution control subroutines that direct and control the progress of one or more simulations;
- Data access subroutines that transmit system information from ADAMS/Solver for subsequent computations and allow you to get information about system behavior;
- Setup subroutines that provide ADAMS/Solver with information necessary to set up the equations for an analysis;
- General utility subroutines that help you write user-written subroutines more quickly and efficiently.
5.3.1.1 Execution Control

The control subroutines, ANALYS, DATOUT, GTCMAT, and MODIFY, are utility subroutines that, when called by the driver subroutine CONSUB, regulate an ADAMS/Solver simulation. ANALYS and MODIFY contain the FORTRAN equivalents of interactive commands to initiate analysis and to modify the ADAMS/Solver data set, respectively.

DATOUT produces output from the simulation or simulations. These subroutines are of most value if you want to run many simulations on a single data set or on variations of that data set. Table 10 lists the control subroutines and their functions.

Table 10. Control Subroutines and Their Functions

<table>
<thead>
<tr>
<th>Control Subroutines</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYS</td>
<td>Passes control to the ADAMS/Solver simulation routines.</td>
</tr>
<tr>
<td>DATOUT</td>
<td>Produces output for a simulation controlled by the CONSUB driver subroutine.</td>
</tr>
<tr>
<td>GTCMAT</td>
<td>Computes the compliance matrix for a set of markers in an ADAMS/Solver model.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>Modifies ADAMS/Solver data set automatically according to your specifications.</td>
</tr>
</tbody>
</table>
5.3.1.2 Data Access

The access subroutines allow user-written subroutines to transmit information to and from ADAMS/Solver. Table 11 lists the access subroutines and their functions.

Table 11. Access Subroutines and Their Functions

<table>
<thead>
<tr>
<th>Data Access Subroutines</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKISPL</td>
<td>Accesses the data in a SPLINE statement and fits Akima cubic curves (splines) to the data.</td>
</tr>
<tr>
<td>CUBSPL</td>
<td>Accesses the data in a SPLINE statement and fits traditional cubic curves (splines) to the data.</td>
</tr>
<tr>
<td>ERRMES</td>
<td>Outputs messages to document error conditions that occur in the subroutines you write.</td>
</tr>
<tr>
<td>GETCPU</td>
<td>Retrieves the current amount of CPU time used.</td>
</tr>
<tr>
<td>GETINM</td>
<td>Retrieves information as to whether the current ADAMS command input mode is interactive or indirect (Command File Mode).</td>
</tr>
<tr>
<td>GETINT</td>
<td>Retrieves the type of integrator used for the current simulation.</td>
</tr>
<tr>
<td>GETMOD</td>
<td>Accesses an integer variable that specifies type current analysis mode.</td>
</tr>
<tr>
<td>GETSLV</td>
<td>Retrieves the name of the active linear systems solver, either HARWELL or CALAHAN.</td>
</tr>
<tr>
<td>GETSTM</td>
<td>Retrieves the current simulation time.</td>
</tr>
<tr>
<td>GETVER</td>
<td>Retrieves the current ADAMS version number as it appears in the ADAMS banner.</td>
</tr>
<tr>
<td>GTARAY</td>
<td>Accesses the double-precision numbers that an ARRAY statement stores.</td>
</tr>
<tr>
<td>GTSTRG</td>
<td>Accesses the character string that a STRING statement stores.</td>
</tr>
<tr>
<td>SYSARY</td>
<td>Accesses a set of system state values, such as all displacement components of a marker.</td>
</tr>
<tr>
<td>SYSFNC</td>
<td>Accesses a single system state value, such as the x displacement of a marker.</td>
</tr>
<tr>
<td>TIMGET</td>
<td>Returns the simulation time corresponding to the last successful simulation step.</td>
</tr>
<tr>
<td>TIRARY</td>
<td>Provides tire state values for use in REQSUBs and SENSUBs.</td>
</tr>
<tr>
<td>USRMES</td>
<td>Allows you to output messages for information or for documenting error conditions that occur in the subroutines that you write.</td>
</tr>
</tbody>
</table>
5.3.1.3 Setup
The UCOVAR setup subroutine used with UCOSUB informs ADAMS/Solver of the part displacements and velocities that you are using in your user-defined constraint (UCON).

5.3.1.4 General
ADAMS/Solver has function subroutines that you can call from your user-written subroutines to compute and return a value to your user-written subroutines. Most of these subroutines can be accessed from a function expression also.

See Table 7, “Overview of Functions Available in ADAMS/Solver,” for more information.

Table 12 lists the function subroutines and their functions.

<table>
<thead>
<tr>
<th>Function Subroutines</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>BISTOP</td>
<td>Evaluates a force restricting displacement of a part in two opposite directions.</td>
</tr>
<tr>
<td>CHEBY</td>
<td>Evaluates a Chebyshev polynomial.</td>
</tr>
<tr>
<td>FORCOS</td>
<td>Evaluates a Fourier cosine series.</td>
</tr>
<tr>
<td>FORSIN</td>
<td>Evaluates a Fourier sine series.</td>
</tr>
<tr>
<td>HAVSIN</td>
<td>Evaluates a haversine function.</td>
</tr>
<tr>
<td>IMPACT</td>
<td>Evaluates a force restricting displacement of a part in one direction.</td>
</tr>
<tr>
<td>ISTRNG</td>
<td>Converts an integer variable to a character string.</td>
</tr>
<tr>
<td>POLY</td>
<td>Evaluates a polynomial.</td>
</tr>
<tr>
<td>RCNVRT</td>
<td>Converts rotational coordinates from one coordinate system to another.</td>
</tr>
<tr>
<td>RSTRNG</td>
<td>Converts a double-precision variable to a character string.</td>
</tr>
<tr>
<td>SHF</td>
<td>Evaluates a simple harmonic function</td>
</tr>
<tr>
<td>STEP</td>
<td>Approximates a step function with a cubic polynomial</td>
</tr>
<tr>
<td>TCNVRT</td>
<td>Converts translational coordinates from one coordinate system to another.</td>
</tr>
</tbody>
</table>
5.4 Callable ADAMS/Solver Subroutines and Their Functions

Callable ADAMS/Solver allows the ADAMS/Solver analysis software to be embedded into other general purpose software packages, such as customized pre-and postprocessors and optimization tools.

There are three types of Callable ADAMS/Solver subroutines:

- Driver — that sets up and controls the ADAMS/Solver analysis.
- Execution Control — that directs and controls the progress of one or more simulations.
- Monitor — that evaluates progress of the simulation.

Table 13 lists the callable subroutines and their functions.

<table>
<thead>
<tr>
<th>Callable ADAMS/Solver</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3BGN</td>
<td>Sets preliminary initializations for ADAMS/Solver.</td>
</tr>
<tr>
<td>A3CMD</td>
<td>Processes ADAMS/Solver commands.</td>
</tr>
<tr>
<td>A3INPT</td>
<td>Defines input and output unit numbers, sets various defaults, and reads in and verifies the ADAMS/Solver data set.</td>
</tr>
<tr>
<td>A3MNTR</td>
<td>Evaluates progress of the simulation.</td>
</tr>
<tr>
<td>A3MSG</td>
<td>Provides a verbal description of status codes.</td>
</tr>
<tr>
<td>A3TERM</td>
<td>Terminates the session.</td>
</tr>
<tr>
<td>DREVINT</td>
<td>Drives ADAMS/Solver.</td>
</tr>
</tbody>
</table>
Chapter 6
Executing ADAMS/Solver

Introduction
ADAMS analysis can be executed in standard ADAMS/Solver or in customized applications created using Callable ADAMS/Solver. The description that follows applies to execution of standard ADAMS/Solver. For details on Callable ADAMS, refer to Chapter 4 of the ADAMS/Solver Subroutines Reference Manual.

The fundamental distinction that you have to make for an application of the ADAMS/Solver code is between the various modes of analysis. That is, once the statements in the data set define the problem, you must choose to simulate the mechanical system in one of the following analysis modes:

- **Dynamics** Fully dynamic simulation of the multibody system
- **Kinematics** Kinematic simulation of the motion
- **Linear** An analysis of a linearization of the system
- **Statics** Static or quasi-static equilibrium analysis
- **Transient** A kinematic or dynamic simulation

Regardless of the analysis mode you choose, you must complete the following steps to execute ADAMS/Solver:

- Compile user-written subroutines (if any) and link the resulting object codes to the ADAMS/Solver program to create an executable image. This step is not required if there are no user-written subroutines.
- Run the ADAMS/Solver executable module.
- Issue the necessary ADAMS/Solver commands.

This section explains how to execute ADAMS/Solver as follows:

- “Using the MDI Procedure File” briefly discusses using the MDI procedure file to prepare and invoke the ADAMS/Solver executable.
- “Using the ADAMS/Solver Execution Modes” discusses the procedures for executing ADAMS/Solver.
- “Understanding the Execution Phases” discusses the phases of ADAMS/Solver execution.
- “Understanding the Command Syntax” indicates the syntactical rules governing ADAMS/Solver commands.
- “Understanding the Command Groups” organizes and defines
ADAMS/Solver commands according to their functions.
6.1 Using the MDI Procedure File

A procedure file is part of your ADAMS/Solver package. It is a set of instructions to the operating system that will invoke an ADAMS/Solver simulation for you. In Unix, you enter the command ‘mdi’ to run the procedure. In VMS, you enter ‘@mdi’ or a command such as ‘@mdi’ to run the procedure. The details of using the procedure vary depending on the computer system you are using. See “Guide to Running the ADAMS Product Line on Your System” (available on-line) for more information on using the procedure on your specific system.

The procedure allows you to select from several options when running ADAMS/Solver. On most systems, you may choose between running ADAMS/Solver interactively or in batch. On some systems, you may run ADAMS/Solver in debug mode which allows you to use the system’s source level debugger to monitor and debug your user-written subroutines.

You may also use the procedure to link your user-written subroutines with ADAMS/Solver to create a user executable. If you have written subroutines, you must first compile them using the system’s FORTRAN77 compiler and then use the ‘mdi’ procedure file to link them with ADAMS/Solver. The procedure will also allow you to link with optional libraries such as those supplied with ADAMS/Tire and ADAMS/Android. You can then use the procedure to run your user executable.

Once you have made your selections and entered the requested information, the procedure will load the standard or user ADAMS/Solver executable and commence execution.

6.2 Using The ADAMS Product Menu

The ADAMS Product Menu provides all the functionality of the MDI Procedure File, but with a Graphical User Interface. You can execute ADAMS/Solver by double-clicking on its button. You can also click-and-drag the mouse on the ADAMS/Solver button to access the linking and processing options.
6.3 Using the ADAMS/Solver Execution Modes

There are two ways you can execute ADAMS/Solver. The first, interactive execution, requires that you monitor the execution interactively and input commands as necessary (although you can anticipate all prompts and use a Command File to automatically input a sequence of commands). The second, batch execution, requires a command file containing the necessary ADAMS/Solver commands. Both require a file containing an ADAMS/Solver data set with complete information about the system you are analyzing.

6.3.1 Interactive Mode

During interactive execution, ADAMS/Solver sends prompts and messages to the screen for the information it needs. This does not occur during batch execution. The first prompt that appears (in quotes) and possible response are as follows.

“Enter ADAMS/Solver data set name or carriage return? (or STOP):”

Choose one of the following:

1. To stop execution here or at any point during interactive execution, enter STOP (see Section 3.8.11 “STOP” command in the ADAMS/Solver Reference Manual).
2. To go directly to the Enter Command prompt, enter a carriage return. You must eventually enter a FILE command to load a data set and specify the output file name.
3. If you know the name of the file containing the data set, enter the file name. If you do not know the file name, enter a question mark (?). If you enter a question mark, ADAMS/Solver
   a. Lists on the screen all the ADAMS/Solver data sets in the current directory that have been previously executed, and
   b. Prompts you as follows:
      “Select a file by number or enter a nondisplayed file:”
   c. Choose one of the following:
      i. If you want ADAMS/Solver to use one of the listed files, type in the sequence number of the file.
      ii. If you want to use a different file name, specify that file name.

After you enter a data set name, ADAMS/Solver prompts you for the output file name:

Enter ADAMS output file name or?
(default name is same as input):”

The various types of output files ADAMS/Solver produces after analyzing a data set all have the same prefix with different file qualifiers to indicate the file type.
On the VAX/VMS system, the file qualifiers are listed in Table 14.

<table>
<thead>
<tr>
<th>File</th>
<th>File Qualifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAMS/Solver input (data set)</td>
<td>.ADM</td>
</tr>
<tr>
<td>Tabular Output</td>
<td>.OUT</td>
</tr>
<tr>
<td>Diagnostics</td>
<td>.MSG</td>
</tr>
<tr>
<td>Graphics</td>
<td>.GRA</td>
</tr>
<tr>
<td>Request</td>
<td>.REQ</td>
</tr>
<tr>
<td>Results</td>
<td>.RES</td>
</tr>
<tr>
<td>Femdata</td>
<td>.FEM</td>
</tr>
<tr>
<td>Message Data Base</td>
<td>.MDB</td>
</tr>
</tbody>
</table>

These are the only inputs required during the input phase of the simulation. ADAMS/Solver will read through the data set, check it for consistency, correctness, and completeness prior to performing an analysis.

**6.3.2 Batch Execution Mode**

In batch execution mode, you provide a Command File rather than enter commands interactively. The Command File contains the same responses and commands you would enter directly. For example:

Line 1 Input file name.

Line 2 Output file name (or optionally a blank line if you want the output file to have the same name as the input file).

Lines 3 and following.

The `SIMULATE` command and any number of additional ADAMS/Solver interactive commands (except `BACKUP`, `CENTER`, and `ZOOM`).
6.4 Understanding the Execution Phases

For simplicity, an ADAMS/Solver execution may be thought of as occurring in four distinct and sequential phases. These are:

- Input
- Input check
- Analysis
- Output

6.4.1 Input

The main function of the input phase is to store the model described in the data set in program memory. While doing this, ADAMS/Solver will perform several useful checks on the provided information. These include:

- Checking that statement syntax is not violated.
- Checking that arguments have the right type and number of values.
- Checking that mandatory arguments for each statement are specified and that exclusive sets of arguments are not specified.

The main purpose of the checks performed at this stage is to ensure that each statement that is stored has been defined completely and correctly. Once a data set has been completely read and accepted, ADAMS/Solver has essentially verified that each statement individually is syntactically correct and complete.

ADAMS/Solver now has to verify that the group of statements specified are consistent in their definition and do indeed define a logical system.
6.4.2 Input Check

Input check is the phase where ADAMS/Solver verifies that you have defined a proper system in the model definition input file.

ADAMS/Solver will perform the following checks at this stage:

- Ensures that elements being cross referenced indeed do exist. For instance, if a `PART` statement refers to a particular `CM` marker, ADAMS/Solver will verify that the `CM` marker exists and that it belongs to the part referring to it.
- Ensures that element definitions are realistic and do not violate physical principles. For instance, ADAMS/Solver will calculate the angle between the z-axes of the two markers of a revolute joint to see if they are approximately parallel as required, and not perpendicular.
- Computes the system degrees of freedom and report it to you.
- Tests the system for the existence of redundant constraints.
- Sets up an analysis representation of the input data that you have specified. The analysis representation is designed to optimally satisfy the data needs of equation set up and equation solution procedures.

Once the analysis representation of the model has been set up, ADAMS/Solver is ready to perform an analysis.

6.4.3 Analysis

The output from a simulation is generated in the analysis phase. Use the `SIMULATE` command to specify a non-linear analysis of the system and use the `LINEAR` command to specify a linear analysis of the system.

Linear analysis can be performed only on systems with one or more degrees of freedom. It is not applicable to systems with zero degrees of freedom. All of the other types of analyses may be performed on systems with zero or more degrees of freedom. If you issue a `SIMULATE/DYNAMIC` command for a system with zero degrees of freedom, ADAMS/Solver will proceed to use numerical integration to solve for the system response even though a kinematic analysis may be more appropriate.

After completing a simulation, you may want to run another analysis of the same system or continue the simulation. ADAMS/Solver will permit both.

6.4.4 Output

The output phase of the simulation is typically the last phase. It essentially consists of collecting, formatting, and writing the various output files that ADAMS/Solver generates.

See the `OUTPUT` statement and command, Sections 2.10.4 and 3.7.1, in the `ADAMS/Solver Reference Manual` for various output options that are available to you.
Chapter 6. Executing ADAMS/Solver

by statement and command.
6.5 Understanding the Command Syntax

ADAMS/Solver commands generally follow the syntactical rules governing ADAMS/Solver statements (see Section 4.3, “Statement Syntax”). Exceptions and special cases appear in this section under the following headings:

- Angles
- Arguments
- Continuations

6.5.1 Angles

ADAMS/Solver generally assumes all angular data in an ADAMS/Solver command is in radians. This manual, however, notes exceptions. Therefore, if you want to use degrees as the input value for your angular data instead of the default, use a D after the value to indicate degrees.

6.5.2 Arguments

If ADAMS/Solver finds one of the following syntactical errors in the arguments of a command, then the code issues an error message and ignores the entire command:

- An invalid number of values for an argument
- An ambiguous argument abbreviation
- An invalid value type for an argument (for example, you input a character value and the argument requires a numeric value or you input a real number with a nonzero fractional part and the argument requires an integer)
- Omission of a required argument or a required value for an argument

If you assign an argument value that is not a member of the set of permissible values for that argument, ADAMS/Solver issues an error message and ignores the argument, but executes the rest of the command.

6.5.3 Continuations

ADAMS/Solver allows a command line to be at most eighty characters long. To input a command that is more than one line long, put an ampersand (&) as the last character of the input line. ADAMS/Solver will respond by asking you to type in the next character. You may enter as many lines of input as you like.
6.6 Command Groups

This section groups all ADAMS/Solver interactive commands according to their functions and explains the use of each command. All ADAMS/Solver commands are available in the graphic interactive mode and in the non-graphic interactive mode (see Section 3.8.4, “EXECUTION,” in the ADAMS/Solver Reference Manual). The names of the command groups are as follows.

- Inertial Data
- Geometry
- Constraints
- Forces
- Geometric System Modeling
- Reference Data
- Analysis Parameters
- Output
- Simulation
- Information
- Request and Graphics

6.6.1 Inertial Data

PART Redefines and/or lists the data you input for a PART statement.

6.6.2 Geometry

MARKER Redefines and/or lists all the data for a MARKER statement.
6.6.3 Constraints

- **JOINT**: Lists all the data for a JOINT statement.
- **MOTION**: Redefines and/or lists the data you input for a MOTION statement.
- **UCON**: Redefines and/or lists the data you input for a UCON statement.

6.6.4 Forces

- **ACCGRAV**: Redefines and/or lists the data you input for a ACCGRAV statement.
- **BEAM**: Redefines and/or lists the data you input for a BEAM statement.
- **BUSHING**: Redefines and/or lists the data you input for a BUSHING statement.
- **FIELD**: Redefines and/or lists the data you input for a FIELD statement.
- **GFORCE**: Redefines and/or lists the data you input for a GFORCE statement.
- **SFORCE**: Redefines and/or lists the data you input for an SFORCE statement.
- **SPRINGDAMPER**: Redefines and/or lists the data you input for a SPRINGDAMPER statement.
- **VFORCE**: Redefines and/or lists the data you input for a VFORCE statement.
- **VTORQUE**: Redefines and/or lists the data you input for a VTORQUE statement.

6.6.5 Geometric Systems Modeling

- **VARIABLE**: Redefines and/or lists the data you input for a VARIABLE statement.

6.6.6 Reference Data

- **ARRAY**: Redefines and/or lists the data you input for an ARRAY statement.
- **STRING**: Redefines and/or lists the data you input for a STRING statement.
Chapter 6. Executing ADAMS/Solver

6.6.7 Analysis Parameters

- **BDFPAR**: Controls the corrector algorithm for the BDF integrator accessible from the DSOLVER and DYNAMICS statements.
- **CPAPAR**: Controls parameters for the coordinate partitioning algorithm.
- **DEBUG**: Redefines and/or lists the data you input for a DEBUG statement.
- **DSOLVER**: Defines the two fundamental components of the mathematical methods for a dynamic solution: the form of the equations and the integration algorithm.
- **DSTIFF**: Specifies the integration error tolerance and other parameters for a dynamic analysis using the DASSL integration algorithm.
- **DYNAMICS**: Provides an automatic way to select an integrator and an easy way to select a consistent set of error tolerances.
- **EQUILIBRIUM**: Redefines and/or lists the data you input for an EQUILIBRIUM statement.
- **GSTIFF**: Redefines and/or lists the data you input for a GSTIFF statement.
- **IC**: Redefines and/or lists the data you input for an IC statement.
- **INTPAR**: Controls parameters common to multiple integrators.
- **KINEMATICS**: Redefines and/or lists the data you input for a KINEMATICS statement.
- **LSOLVER**: Selects a linear equation solver.
- **SENSOR**: Redefines and/or lists the data you input for a SENSOR statement.
- **WSTIFF**: Redefines and/or lists the data you input for a WSTIFF statement.

6.6.8 Output

- **OUTPUT**: Redefines and/or lists the data you input for an OUTPUT statement.
- **REQUEST**: Redefines and/or lists the data you input for a REQUEST statement.
6.6.9 Simulation

Simulation commands are those commands that initiate and control ADAMS/Solver simulations. The simulation commands include a command to pass control to a driver subroutine, a command to toggle between graphics and alphanumeric modes and to obtain new initial conditions, a command to request a linear analysis, a command to request a dynamic, a kinematic, a static equilibrium, or a quasi-static equilibrium analysis, as well as the \textit{STOP} command.

\textbf{ACTIVATE} \hspace{2em} Activates a statement that was previously turned off by the \textit{DEACTIVATE} command.

\textbf{CONTROL} \hspace{2em} Branches program control to the driver user-written subroutine \textit{CONSUB} (see Section 2.1.1, “CONSUB,” in the \textit{ADAMS/Solver Subroutines Reference Manual}.)

\textbf{DEACTIVATE} \hspace{2em} Turns off a statement that was previously turned on by the \textit{ACTIVATE} command, or that was active by default.

\textbf{EXECUTION} \hspace{2em} Toggles between the graphics and the alphanumeric modes and initiates restart.

\textbf{FILE} \hspace{2em} Loads a new mile from a data set file, or executes commands from a command file.

\textbf{LINEAR} \hspace{2em} Computes the eigenvalues and mode shapes for the ADAMS/Solver model and computes the state matrices representation of the ADAMS/Solver model. (See the \textit{ADAMS/Linear Option}.)

\textbf{PREFERENCES} \hspace{2em} Controls general options for ADAMS/Solver execution.

\textbf{RELOAD} \hspace{2em} Restarts a simulation from a previously saved model or simulation state.

\textbf{REQUEST} \hspace{2em} Redefines and/or nets the data you input for a \textit{REQUEST} statement.

\textbf{SAVE} \hspace{2em} Stores the current ADAMS/Solver model and simulation conditions or only the current simulation states.

\textbf{SIMULATE} \hspace{2em} Specifies the type of analysis and (if applicable) the end time and the number of output steps.

\textbf{STOP} \hspace{2em} Terminates execution of ADAMS/Solver in the interactive mode and returns control to the operating system.
Chapter 6. Executing ADAMS/Solver

6.6.10 Information
The information commands invoke help or display system information.

- **HELP**  
  Explains the commands available in ADAMS/Solver.

- **HOTLINE**  
  Displays information about the hotline services at Mechanical Dynamics.

- **INFO**  
  Lists the vector components for displacement, velocity, acceleration, or force of one marker with respect to another.

- **MENU**  
  Lists all of the ADAMS/Solver interactive commands.

- **SHOW**  
  Lists statistics for any of these topics: CPU time, graphics options, simulation status, and time.

- **TIME**  
  Shows the simulation time, the integration step size (\( \text{STEP} \)), the number of function evaluations (\( \text{IFCT} \)), and the number of integration steps (\( \text{ISTP} \)).

6.6.11 Request and Graphics
The request and graphics commands allow you to display and to manipulate REQUEST and GRAPHICS statement output.

- **ACENTER**  
  Lists the global x, y, and z values of the current display center.

- **AINIT**  
  Resets the display to its original orientation about the global x-axis, the global y-axis, and the global z-axis.

- **AUTOSCALE**  
  Recenters and rescales the data for a graphics display.

- **BACKUP/ZOOM**  
  Changes the scale of the current display. BACKUP shrinks the display to a rectangular area you define, whereas ZOOM enlarges a rectangular area you define.

- **CENTER**  
  Brings up cross hairs that allow you to define a new center for your display.

- **CLEAR**  
  Clears the graphics screen.

- **DISPLAY**  
  Generates a graphics display for the current simulation time.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAW</td>
<td>Displays parts, markers, and/or joints with their identifiers.</td>
</tr>
<tr>
<td>DT</td>
<td>Displays and modifies the simulation time between successive display frames.</td>
</tr>
<tr>
<td>ERASE/RESTORE</td>
<td>Disables or enables the graphics output from as many as three GRAPHICS statements.</td>
</tr>
<tr>
<td>PLOT</td>
<td>Plots the REQUEST statement output from the current data set execution.</td>
</tr>
<tr>
<td>RETURN</td>
<td>Indicates you do not want to enter more graphics commands at this time, but want execution to continue.</td>
</tr>
<tr>
<td>SET</td>
<td>Allows you to set the options for graphics display.</td>
</tr>
</tbody>
</table>
Chapter 6. Executing ADAMS/Solver
Chapter 7
Output Files

Introduction
ADAMS/Solver accepts input from three separate files:

- Model definition (.ADM)
- Command (.ACF)
- Save (no qualifier exists for this file)

As output, ADAMS/Solver may create the following files:

- Tabular output (.OUT)
- Message file (.MSG)
- Graphics file (.GRA)
- Request file (.REQ)
- Save (no qualifier exists for this file)
- Results (.RES)
- Femdata (.FEM)
- Message data base (.MDB)
- State matrices (no qualifier exists for this file)

See Figure 16 for an illustration of how this works in ADAMS/Solver.
This section explains the purpose and usage of each of the output files that ADAMS/Solver generates. Some of these files are always generated, whereas the rest are optionally written out.

Table 15 summarizes ADAMS/Solver statements and commands that can affect the output being generated for a particular simulation.
Table 15. An Overview of the Statements and Commands that Affect the Contents of ADAMS/Solver Output Files

<table>
<thead>
<tr>
<th>Statements</th>
<th>File Name</th>
<th>Abbreviation and File Name Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEMDATA</td>
<td>FEM</td>
<td>.FEM</td>
</tr>
<tr>
<td>LIST/NOLIST</td>
<td>Tabular Output</td>
<td>.OUT</td>
</tr>
<tr>
<td>MREQUEST</td>
<td>Tabular Output Request</td>
<td>.OUT .REQ</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Graphics Request</td>
<td>.GRA .REQ</td>
</tr>
<tr>
<td></td>
<td>Tabular Output</td>
<td>.OUT</td>
</tr>
<tr>
<td>REQUEST</td>
<td>Tabular Output Request</td>
<td>.OUT .REQ</td>
</tr>
<tr>
<td>RESULTS</td>
<td>Results</td>
<td>.RES</td>
</tr>
</tbody>
</table>

**Commands**

<table>
<thead>
<tr>
<th>Commands</th>
<th>File Name</th>
<th>Abbreviation and File Name Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINEAR</td>
<td>Tabular Output Results</td>
<td>.OUT .RES</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Graphics Request</td>
<td>.GRA .REQ</td>
</tr>
<tr>
<td></td>
<td>Tabular Output</td>
<td>.OUT</td>
</tr>
<tr>
<td>SAVE</td>
<td>Save</td>
<td>None</td>
</tr>
<tr>
<td>RESULTS</td>
<td>Results</td>
<td>.RES</td>
</tr>
<tr>
<td>REQUEST</td>
<td>Tabular Output Request</td>
<td>.OUT .REQ</td>
</tr>
</tbody>
</table>
7.1 Tabular Output File

7.1.1 Description

The Tabular Output File is a sequential access, formatted file that can be read and interpreted by you. ADAMS/Solver will always create this file. The data contained in the tabular output file is dependent on the output and debug options that you specify. Briefly, the tabular output file may contain the following data:

- The program header specifying time and date of creation of the file, the ADAMS/Solver version number, and other license related information
- The title of the data set being analyzed
- A map of the equations and variables in your system
- A table of part initial displacement values
- A table of part initial velocity values
- A table of part principal inertias and the orientation of the part principal axes with respect to the LPRF
- A time history of the request output as specified by REQUEST and MREQUEST output
- DEBUG data
- Mode frequencies, energies, and shapes computed by the LINEAR command

In some cases, you have the option to create a concatenated Tabular Output File containing more than one set of output. For more information, see Section 7.10, “Concatenated Output Files.”
7.1.2 Usage

Use the Tabular Output File to generate tabular values of REQUESTs from an ADAMS/Solver simulation. Remember that ADAMS/View can perform the same tasks as a postprocessing function.

When debugging ADAMS/Solver models, you will find it useful to use the debugging options DUMP and REQDUMP in the DEBUG statement (see Section 2.9.3 in the ADAMS/Solver Reference Manual). The output form these options will be contained in the tabular output file.

Several factors influence the volume of data that ADAMS/Solver outputs in request tables.

- The NOPRINT argument on the OUTPUT command/statement suppresses the printing of request data in the Tabular Output File (see the OUTPUT statement and command, Sections 2.10.4 and 3.7.1, in the ADAMS/Solver Reference Manual).

- Simulations that a CONSUB driver subroutine initiate and an ANALYS control subroutine control and do not produce output unless the DATOUT control subroutine is called (see Section 3.1.2, “DATOUT,” in the ADAMS/Solver Subroutines Reference Manual).

- The type of analysis you specify with the SIMULATE command affects the volume of data. For example, a static equilibrium analysis outputs request data only at static equilibrium, but dynamic, kinematic, and quasi-static equilibrium analyses output request data at each simulation step.

- The STEPS argument on the SIMULATE command specifies the number of output steps that affects the volume of output.

- The DT command, or the DT argument on a SENSOR, may change the simulation time between output steps and, as a result, alter the number of output steps (see Section 3.10.9, “DT,” and Section 2.9.12, “SENSOR,” in the ADAMS/Solver Reference Manual.)

Similarly, arguments on the OUTPUT command/statement control the format of numerical output, the coordinates of angular output, and the line length of output (see the OUTPUT statement and command, Sections 2.10.4 and 3.7.1, in the ADAMS/Solver Reference Manual).
7.2 Message File

7.2.1 Description
The Message File is a sequential access, formatted file that contains all the messages that are generated in an ADAMS/Solver simulation. Messages are categorized into four distinct groups:

- Informative
- Warning
- Error
- Program fault

Each message begins with a message key that identifies the message and the subroutine producing it. Table 16 summarizes the different types of messages. Messages are stored in the sequence in which they are created.

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informative</td>
<td>Conveys simulation information to you.</td>
</tr>
<tr>
<td>Warning</td>
<td>Signals potential problems and notes inconsistencies in your data set or in the solution process. Warnings are nonfatal and do not terminate simulation.</td>
</tr>
<tr>
<td>Error</td>
<td>Flags data set or user input errors that terminate execution.</td>
</tr>
<tr>
<td>Program Fault</td>
<td>Indicates programming errors that warrant contacting the Hotline at Mechanical Dynamics, Inc.</td>
</tr>
</tbody>
</table>
Chapter 7. Output Files

7.3 Results File

7.3.1 Description

The Results File contains all of the output from an ADAMS/Solver simulation. Results files may be formatted and sequential access, or binary and direct access.

The Results File together with the model description input file comprise a “complete” description of a system and its response. Each results file can store the results from one simulation. For more information on results files, see Appendix C, “Results File Structure,” in the ADAMS/Solver Reference Manual.

In some cases, you have the option of creating a concatenated Results File containing more than one set of output. For more information, see Section 7.10, “Concatenated Output Files.”

7.3.2 Usage

Use the RESULTS statement (see Section 2.10.6, “RESULTS,” in the ADAMS/Solver Reference Manual) to generate a Results File. To control the amount of data output to this file, use options in the RESULTS statements.

In general, data that ADAMS/Solver saves in the Results File contains the least amount of information needed to completely specify the system response. You may use the system geometry information to compute other information that you may need. This is the preferred way to analyze the results of a simulation.

Note

The Results File is in a form that can be read by the ADAMS/View postprocessor and not the user.
Chapter 7. Output Files

7.4 Graphics File

7.4.1 Description

The Graphics File is a sequential access formatted file containing information that enables a system to be drawn pictorially. This data is typically read in by a graphical postprocessor. The graphics file consists of two major sections:

- A section where system geometry is stored.
  This data is invariant.
- A time dependent section.
  This section of data is output in blocks, one block for each point in time. The position and orientation of each part is written out in each block.

In some cases, you have the option of creating a concatenated Graphics File containing more than one set of output. For more information, see Section 7.10, “Concatenated Output Files.”

7.4.2 Usage

Generate a Graphics File by using the OUTPUT/GRSAVE option (see Section 2.10.4, “OUTPUT” in the ADAMS/Solver Reference Manual). You may use any ADAMS/Solver postprocessor, such as ADAMS/View, to display the system and animate it.

Note

Use ADAMS/View with the Results File and not the Graphics File for postprocessing.
7.5 Request File

7.5.1 Description

The Request File is a sequential access, formatted file that is used for creating x-y plots of simulation results. The Request File is formatted and designed for being easily read in by postprocessor.

Like the Graphics File, the Request File also consists of two major sections:

- The first section contains a description of the various requests being output.
- The second section contains a time history of all the request values. This data is output in blocks, one block for each point in time containing all of the request values.

In some cases, you have the option of creating a concatenated Request File containing more than one set of output. For more information, see Section 7.10, “Concatenated Output Files.”

7.5.2 Usage

If you want to generate a Request File, use the OUTPUT/REQSAVE option (see Section 2.10.4, “OUTPUT” in the ADAMS/Solver Reference Manual). This will write all REQUEST and MREQUEST output to the Request File.

Subsequently, you may read the Request File using a postprocessor like ADAMS/View and plot the various signals or manipulate them in other ways.
Chapter 7. Output Files

7.6  Save File

7.6.1  Description
The Save File contains information that allows you to restart a subsequent analysis from the saved state, rather than from time zero. It contains values for all the simulation states, such as displacements, velocities, forces, and user-defined variables.

You may also choose to save the entire model along with the states. In this case, the Save File will also contain a complete description of all the parts, joints, etc., in the model.

7.6.2  Usage
To generate a Save File, issue the SAVE command. Choose SYSTEM to save your current model and simulation states, or STATES to save just the states.

Use the RELOAD command to read the information from a Save File back into ADAMS/Solver. If you reload the entire system, ADAMS/Solver will replace the current model and simulation states with those from the file. If you reload states, ADAMS/Solver will maintain the current model, but replace the simulation states with those from the file.

7.7  FEM File

7.7.1  Description
The FEM File contains information describing the loads applied to each part during the simulation. Structural analysis techniques, such as finite-element analysis, can use these loads to predict detailed stresses within the parts.

ADAMS/FEA Interface converts the information in the FEM File into formats that can be directly used with MSC/NASTRAN and ANSYS. See the ADAMS/FEA Interface User's Guide for more information about interfacing to these and other FEA programs.

7.7.2  Usage
To generate a file of displacement, velocity, acceleration, and force data for input to ADAMS/FEA Interface, include the FEMDATA statement in your ADAMS/Solver data set (see Section 2.10.1, “FEMDATA” statement in the ADAMS/Solver Reference Manual).
7.8 Message Data Base File

7.8.1 Description

In some installations ADAMS/Solver creates a Message Data Base File. This file allows ADAMS/Solver and ADAMS/Kinematics to be more tightly coupled to ADAMS/View and CAD/CAM/CAE interfaces. The message database stores information that is also used to generate the text in the Message File and to also track how much of the simulation has been complete. The information is stored in a format that simplifies its retrieval by other software.

7.8.2 Usage

The message data base utility is needed if you want to stay within the same interface environment during preprocessing, analysis, and post-processing, as explained in the following procedure:

- An ADAMS/Solver analysis is initiated from within ADAMS/View or CAD/CAM/CAE interface to ADAMS/Solver.
- The ADAMS/Solver run completes.
- The ADAMS/Solver interface software checks the Message Data Base File to determine the success of the simulation.
- If the simulation is successful, the interface gives you the option of reviewing any ADAMS/Solver informational messages or you may proceed directly to postprocessing.
- If the simulation failed, the interface allows you to review the ADAMS/Solver error messages and warnings and informs you of the time of simulation failure. You can then return to preprocessing to correct the error(s).
Chapter 7. Output Files

7.9 State Matrices File(s)

7.9.1 Description

The State Matrices File(s) contains information that can be taken to matrix manipulation packages for performing analysis on linearized ADAMS/Solver models. ADAMS/Solver produces these files when the LINEAR command requests state matrices output (see the LINEAR command in the ADAMS/Solver Reference Manual).

7.9.2 Usage

You can request the output in the MATRIXx or MATLAB format. If you request the MATRIXx format, you will generate a single file that contains the state matrices (A, B, C, and D), state variables, plant input and output identifiers. This file is in the MATRIXx “MATSAVE” format. If you want to perform further analysis, you can load this file into MATRIXx. If you request the state matrices in the MATLAB format, you will be generating up to seven files at one time. These files are written in the ASCII flat file format. Each of the four state matrices, namely A, B, C, and D, is written to a separate file. In addition, state variable identifiers, plant input and output identifiers are each written to separate files. Exclusion of plant inputs and/or plant outputs from the model description will results in less than seven files being output.
7.10 Concatenated Output Files

In some cases, you have the option to produce concatenated Tabular Output Files, Request Files, Results Files, and Graphics Files. These files contain two or more complete sets of output written one after another and separated by a line starting with '<++>'. The Graphics File, for example, might contain three complete sets of graphics output, the Results File three sets of results output, the Tabular Output File three sets of tables, and the Request File three sets of request output. Each set of output might come from a different simulation, or a different portion of a simulation.

Each set of output in a concatenated file is identical to a standard, single set of output. For example, you could use a text editor to extract a single set of output and use it as a standard file.

ADAMS generates concatenated files in two cases:

- When you use the RELOAD command to restart a simulation
- When you use an interactive command such as ACTIVATE or DEACTIVATE to change the topology of the system

7.10.1 Using the RELOAD Command to Restart a Simulation

If you issue a RELOAD command, but do not supply the OUTPUT_PREFIX argument, ADAMS will write the separator and new headers to the currently open Results, Graphics, Tabular Output, and Request Files, and then append subsequent output to them. ADAMS will also insert a separator in the Message File. If you add the OUTPUT_PREFIX argument, ADAMS will create a new set of output files.

If you use the RELOAD command to run a series of related simulations (a parameter study, for instance), concatenating the output files may help organize your output. All your output will be held in one Graphics, one Results, and one Request File, rather than individual files for each simulation.
7.10.2 Using an Interactive Command to Change the Topology of the System

If you use interactive commands to change the topology of the system, ADAMS will concatenate output. You can change the system topology by activating or deactivating a force or constraint, changing a force or constraint type, or changing where a force or constraint is applied. When you change the topology in the middle of a simulation, ADAMS will, by default, write the separator and new headers to the currently-open files and then append subsequent output as a new set of data. In this case, ADAMS will not insert a separator in the Message File.

This ensures that the topology change is correctly reflected in the output, but it does create two sets of output that cannot be plotted or animated continuously. The NOSEPARATOR argument on the OUTPUT statement and the command prevents ADAMS from breaking up the output. Instead, ADAMS creates a continuous set of output that you can plot and animate from beginning to end. Although this may be convenient, note that the output could be misleading. A single Graphics File will not show the effects of any geometry changes you make during the simulation. While the part motions will be correct, the force, joint, and marker graphics will remain in their original positions, even if you moved them during the simulation.
Appendix A

The Ten Easy Steps to Happy Simulations

by Michael F. Steigerwald

(Reprinted from “Dynamic Dimensions” News and Views from Mechanical Dynamics, Inc.)

It is difficult for ADAMS/Solver to pinpoint the source of the problem, but ADAMS/Solver will provide a detailed list of possibilities. It is often even more difficult for the user to pinpoint the source of the problem. I, however, have two advantages. First of all, I have the ADAMS/Solver source code at my fingertips. Secondly, I’ve been at this awhile.

Working on models that have frustrated users has helped me develop some “debugging” procedures that I follow each time such a problem is brought to my attention. Over the next few issues, I’ll be explaining these ideas to you, so that you can use them yourself, thereby saving us all a great deal of time and effort. I call them “The Ten Easy Steps to Happy Simulations.”

**STEP 1** Make sure that the data set contains **DEBUG/EPRINT** and at least one **GRAPHIC** for each **PART** and force.

**DEBUG/EPRINT**

Without a doubt, **GRAPHICS** are the most powerful debugging tool, but the **EPRINT** keyword on the **DEBUG** statement/command is #2. **EPRINT** provides a synopsis of each iteration of each simulation step. Even if the simulation is successful, **EPRINT** can tell you where the simulation slows down and what elements are the most troublesome.

It is almost essential to use **EPRINT** during the early stages of the modeling process. Once you’ve debugged the model, you may wish to turn **EPRINT** off to decrease the amount of output and speed things up, a little. I, for one, almost always use **EPRINT**. The only time I’ve been known to turn it off is for simulations that take so many steps that I can’t store all of the diagnostics on disk. But anyway...

**PART GRAPHICs**

No matter how insignificant you may think a **PART** is, a picture is worth a thousand words. I can’t tell you how many times I’ve been told, “**EPRINT** says I’m having trouble with this **PART**, but that’s impossible. It’s a dummy **PART** that’s completely locked by **JOINTs** and **MOTIONs**.” Fifteen minutes later, after I’ve put some real klunky **GRAPHICS** on that **PART**, sure enough, we find that the **PART** is actually moving. Turns out that he’d used a cylindrical **JOINT** instead of a translational **JOINT**,
or some such thing.

PART GRAPHICS help identify incorrect MOTIONS. A PART that should be moving, doesn’t. A PART that shouldn’t be moving, does. A PART should be moving, but not like that. This PART should be rotating at 20 rpm, but it’s rotating at 2000 rpm. Get the idea?

Don’t spend a whole lot of time making real exotic GRAPHICS at first. Save that until the end of the project, when you have to show the boss. Right now, the only important thing is to get the model correct. After all, that’s what we get paid for, isn’t it? Besides, the GRAPHICS have a way of evolving with the model anyway.

Make sure that the GRAPHICS are three-dimensional. You may think you have a planar model, but it may take nearly forever to find out that you have erroneous out-of-plane MOTIONs if all you use are circles and arcs. Detail isn’t important, yet, but the picture should approximate the size of the PART. Fortunately, ADAMS/Solver has several 3D GRAPHIC elements that can be used “quick and dirty.” In no particular order, they are: BOX, CYLINDER, and OUTLINE. We can play Michelangelo later. For now, these are plenty dangerous. By the way, you may find it helpful to use the same ID numbers for the PART and its GRAPHIC.

The BOX is the champion of quick and dirty GRAPHICS. It only needs four arguments: a corner and three dimensions. Put a marker down in the front, lower, left-hand corner of the PART and approximate the X, Y, Z dimensions. Two minutes later, you’ve got a working picture of the car body, aircraft, robot arm, or whatever.

CYLINDERS are great for PARTs that are connected using revolute, cylindrical, or translational Joints. It usually helps to attach the CYLINDER to the I and J markers of the JOINT. This way, the CYLINDER will always have the same axis as the JOINT and will represent the PART MOTION quite nicely. For translational JOINTs, a four sided cylinder looks like a BEAM with a square cross-section. The LENGTH of the cylinder should be the length of the PART. CYLINDERS can be used to quickly model axles, steering racks, robot arms, and BEAMS, to name a few.

OUTLINES can be quickly implemented as simple GRAPHICS for more complicated PARTs. The term “stick figure” applies quite nicely. Just use one OUTLINE for the PART, but make sure that the OUTLINE includes all of the JOINT and force markers attached to that PART. The result will probably be quite ugly, but you’d be amazed at how well it will show the MOTION of the PART.
Appendix A. Ten Easy Steps to Happy Simulations

Force GRAPHICS

Force GRAPHICS help detect incorrect forcing FUNCTIONS. You should use a force GRAPHIC for each constraint (JOINTs and JPRIMs) and applied forces (SPRING, SFORCE, BUSHING, FIELD, and BEAM) in the model. The force GRAPHIC will draw a vector for the force and torque applied to the _marker. These vectors are scaled so that the largest vector fits into the GRAPHIC window at the beginning of the simulation. If one of these vectors mysteriously explodes just before the simulation fails, it could indicate a problem with one of your FUNCTIONS or subroutines.

STEP 2 Execute ADAMS/Solver.

Now you’re ready to give your simulation a try. Actually, the chances are pretty good that it will be successful, and you may not have needed to put in all these GRAPHICS. If so, you’ve got a head start on your presentation GRAPHICS. If not, however, you have everything in place to systematically locate the problem. We’ll consider how to locate the problem further in the next issue. In the meantime, I hope you’ll find that you can glean a lot of information from what we’ve covered so far.

STEP 3 Identify the last successful step.

Now we need to glean as much information as we can from the diagnostics. The failure diagnostics tell us when the simulation failed, why it failed, and what to try next.

The three most important pieces of information to retrieve are, in order:

1) The last successful time,
2) The most problematic equation, and,
3) The most problematic variable.

Fortunately, all of this information is laid out and repeated several times in the failure diagnostics. You will find that the diagnostics in the message file are more verbose than the failure diagnostics that are printed on the screen.

Knowing when the simulation last succeeded, you may be able to use your knowledge about what’s going on around that time to determine the cause of the difficulty.
Appendix A. Ten Easy Steps to Happy Simulations

Ask yourself questions like these:

1) What is my simulation supposed to be doing now?
2) Is it doing it? You might want to use the postprocessor to graphically review the successful portion of the simulation.
3) Do I have any time-dependent functions that grow or change in nature near this time?
4) Has my model encountered an event, such as a pothole or barrier, that changes the system significantly?
...and so on.

STEP 4 Identify the problematic equations and variables.

The failure and EPRINT diagnostics will tell you two things:

1) The equation that has the greatest error, i.e., has been violated most, and,
2) The variable that is changing the most.

Of the two, it is usually more important to find the source of difficulty for the problematic equation. The failure diagnostics will identify the problematic equation and variable at the time of the failure, and the EPRINT diagnostics will identify the problematic equation and variable at each solution step.

Compare the problematic equation and variable identified by the failure diagnostics to those identified at recent steps by the EPRINT diagnostics. If a certain variable or equation has a “history” of trouble, you should take some time to analyze it. If you haven’t already done so, you should insert REQUESTs that will allow you to plot this information.

Ask yourself these questions about problematic equations (s):

1) Does this equation involve a function that may be discontinuous?
2) Does this equation involve a function that asymptotically tends to infinity (or zero)?
3) Does this equation involve a “switching” function; one that switches rapidly from positive to negative values?
4) Does this function have a very high frequency?
Appendix A. Ten Easy Steps to Happy Simulations

Ask yourself these questions about problematic variable(s):

1) Is this a force that may be discontinuous?
2) Is this variable switching from zero to a non-zero value on alternating iteration/steps?
3) Is this an angle (or torque) that may be spinning rapidly, like a tire spinning in mud?
4) Does the magnitude of this variable (from EPRINT) seem right?

If nothing obvious leaps out at you, don’t panic: you’ll find the problem eventually. In the meantime, just remember the problematic equations and variables, so you can keep an eye on them later.

STEP 5 Execute ADAMS/Solver again.

Armed with this information, exit ADAMS/Solver and start from the beginning. For long simulations that fail late in the game, you might save some time by creating a restart file using the SAVE command. You can use the RELOAD command to restart the simulation closer to the point of interest, as if a solution point from the previous simulation were the initial conditions for this simulation.

STEP 6 Simulate up to the last successful step.

Change the arguments on your SIMULATE command so that the END time is the last successful step from the previous simulation. You really only need the one output step to get there, and you don’t need EPRINT for this portion, so feel free to use this information to speed things up.

What if the last simulation never took a successful step? Proceed to step 7.

STEP 7 Turn the interactive GRAPHICS on.

As trite as it may sound, a picture is worth a thousand words. Used in conjunction with DEBUG/EPRINT, the GRAPHICS keyword on the EXECUTION command enables you to monitor the solution process visually. After each solution step (or iteration in some cases), ADAMS/Solver prompts you to:

HIT RETURN TO DISPLAY GRAPHICS

The same options are available interactively as in the postprocessor: zoom, change view, rotate, etc.
Appendix A. Ten Easy Steps to Happy Simulations

**STEP 8**  
Watch the next step, paying careful attention to GRAPHICS of the problematic equations and variables.

By now, all of the following commands should have been issued:

```
SIMULATE/whatever analysis you need
STEPS = 1, END = successful time
DEBUG/EPRINT
EXECUTION/GRAPHICS
SIMULATE/whatever analysis you need,
STEPS = 1, END = unsuccessful time
```

What to look for? Mostly, that comes with experience. In general, though, the FORCE GRAPHICS tell the story.

Keep a sharp eye out for force vectors that at seem to “explode” on a particular iteration or step. If that happens, you’re halfway home. Another possibility is a force vector that disappears occasionally. This indicates a force that is switching on and off, which leads to solution difficulties. Use the ZOOM, DRAW, ERASE/RESTORE, and SET commands to identify the force. Then, trace it back to your data set.

Typically, you’ll find that the problem is in a FUNCTION argument or one of your subroutines. Be particularly aware of IF constructs inside of the FUNCTION or in your own FORTRAN. If you have a discontinuity in a FUNCTION, the odds are very slim that ADAMS/Solver will be able to find the solution. Discontinuous effects can be approximated very efficiently using the STEP, IMPACT, BISTOP, AKISPL, and CUBSPL functions. Take my advice, avoid IFs like the plague.

Another thing to look for, especially during statics, is a PART that is mysteriously moving freely or rapidly. For static equilibrium, each degree of freedom must have forces that can statically balance that freedom. For example, if you have a tire attached to a vehicle with a revolute JOINT, but there are no torques about that revolute JOINT, it is possible that you will never achieve static equilibrium. You are the best judge of proper MOTION of the PARTs in your system. If you see something peculiar, home in on it.

99.44% of the time, you will have spotted the problem by following these steps.
STEP 9 Trace the problem back to the data set.

If you’re still stuck, you may be getting a little concerned along about now. You’ve done just about everything you can think of, but the problem persists. You’re wondering, “Where have I gone wrong? What did I overlook?”

If the world was perfect, you’d be able to go directly to the line in the input file that is causing the problem and change it. Unfortunately, it’s not going to be that easy, but we’ll get there eventually. But it’s going to take a little more time and digging.

The most important thing you’ve done so far is to identify the problematic equation(s). If you’ve gotten this far without any better ideas, we must now dissect the equation (hopefully there will be only one) to find the source of the problem. If you have more than one, this discussion should help you identify the most likely one. In the worst case, you’ll have to apply these procedures to all of your equations.

Return to your message file, and look at the first failure after the last successful step. Specifically, let’s examine the equation error for the first iteration. This is the left-most real number in your EPRINT diagnostics and it identifies the largest error committed by the predictor for any equation. The corrector will converge in just a few iterations if the predictor is sufficiently close to the solution. In other words, if everything is behaving as it should, the number should be less than 1,000, and the corrector should converge in four iterations or less.

If this number is large, the corrector probably will not converge. ADAMS/Solver will predict again with a smaller step size. This assumes that the previous prediction carried the approximation too far away from the solution. For smooth functions, a smaller step will keep the approximation closer to the solution, so that the corrector will easily converge.

If this first EPRINT error stays large even for very small steps, the mechanism has probably encountered an “event,” such as a pothole or other barrier, for which the equations are not smooth. If this is the case, your only recourse is to identify the non-smooth equations and try to smooth them out without changing their nature very much.

In my experience, non-smooth behavior such as this is the root of all evil!

That said, let’s spend some time talking about ways to find and fix these equations.
Identifying Non-smooth Equations

It is quite likely that the non-smooth equation is the one identified by EPRINT. It is even more likely that the troublesome equation is one for which a FUNCTION has been written, since this is the only way to include non-smooth effects.

If the equation identified by EPRINT is a MOTION, DIFF, SFORCE magnitude, or user-written FIELD, your best bet is to analyze the behavior of this FUNCTION in the neighborhood of the failure. This is most easily accomplished using REQUESTs and the REQDUMP argument on the DEBUG command. Here’s how:

1) Create a REQUEST for the force and any variables that the force depends on.
2) Simulate up to one step before the failure.
3) Issue the DEBUG/REQDUMP command.
4) Continue the simulation.

DEBUG/REQDUMP will calculate the REQUESTs and print to the output file on every corrector iteration. Look at the force magnitude and its independent variables for each iteration. Hopefully, you will notice that the force value changes suddenly, before the dependent variables do. If not, and this is unlikely, you will have to continue to trace the mechanism topologically.

The more you can learn about this FUNCTION in this neighborhood, the better. Depending on your resources, you may be able to sweep out carpet plots of the FUNCTION to visually identify the problem.

ADAMS/Solver uses a modified Newton-Raphson method to solve the equations of motion. Assuming that the solution exists, and that the predictor keeps the approximation sufficiently close to the solution (an excellent assumption, by the way), there are only three kinds of FUNCTIONs that Newton-Raphson can’t solve:

- FUNCTIONs that are convex on one side of the solution and concave on the other, such as ARCTAN.
- FUNCTIONs that are concave and symmetric in a small neighborhood of the solution, but flat outside of this neighborhood.
- FUNCTIONs that are discontinuous in this neighborhood. Of the three, discontinuities are the most likely to occur.
These $\text{FUNCTION}$s can lead to discontinuities:

- The dreaded $\text{IF FUNCTION}$. I claim that the $\text{IF FUNCTION}$ should only be used in conjunction with the $\text{MODE FUNCTION}$ to create $\text{FUNCTION}$s that behave differently according to analysis type (dynamic, static, etc.). This is the only safe use for the $\text{IF FUNCTION}$.

- Magnitude $\text{FUNCTION}$s: $\text{DM}$, $\text{VM}$, $\text{ADDM}$, etc. These $\text{FUNCTION}$s are discontinuous in the neighborhood of zero, like absolute values.

- Angular $\text{FUNCTION}$s: $\text{PSI}$, $\text{THETA}$, $\text{PHI}$, and so forth. These $\text{FUNCTION}$s can be discontinuous at $\pi/2$.

- $\text{USER written FUNCTION}$. Watch your logic! Just as the $\text{IF FUNCTION}$ is dangerous, branching in FORTRAN can be more complicated. Those of us who can wander around in a source-level debugger have an advantage here.

If the equation identified by $\text{EPRINT}$ is a $\text{PART}$, $\text{BEAM}$, $\text{SPRING}$, $\text{FIELD}$, or $\text{BUSHING}$ equation, you will probably need to topologically trace the problem to an element with a function. Look at all of the elements attached to the problematic element. Do any of them have $\text{FUNCTION}$s?

Is it possible that the discontinuous force is being transmitted to this element? For example, suppose a stiff $\text{BEAM}$ is attached to a $\text{PART}$ that is excited by a non-smooth $\text{SFORCE}$. The source of the problem is the $\text{SFORCE}$, but the problem shows up in the $\text{BEAM}$.

Continue to trace the “shock” until you find the offending $\text{FUNCTION}$.

You may also try the brute force method. Insert $\text{MREQ/ id, FORCE, APPFOR=ALL}$ in your data set, then turn on $\text{DEBUG/REQDUMP}$ in the neighborhood of the failure. This will produce requests for all applied forces in the data set, which may help you find the problematic one.

Another possibility is to change, or “turn off,” candidate $\text{FUNCTION}$s, one by one. If the simulation succeeds after a minor modification, this could indicate that the modified function was the culprit.
Appendix A. Ten Easy Steps to Happy Simulations

This whole discussion has centered around a large equation error. What if the equation error is small, say less than 1,000? Well, it may be that Newton-Raphson isn’t trying hard enough. The default for the maximum number of iterations for an integration step and the default for the pattern of Jacobian re-evaluations are empirically chosen. To sort of throw the “kitchen sink” at the problem, set the maximum number of iterations to 25 and set the pattern of Jacobian re-evaluations to all “trues” to see if that improves performance any.

If all else fails, it may just be that your problem cannot be solved within the specified error tolerance. Try doubling the value of the error tolerance in the neighborhood of the failure until the simulation fails. Even with a ridiculous error tolerance, there must be something non-smooth going on. If it succeeds with a looser tolerance, try to reduce tolerance again as soon as possible after the “event.”

STEP 10 Fix the problem in the data set.

We now assume that the problem has been identified. But how do we fix it? I have a few simple rules here that I think you’ll find helpful.

Every problem I’ve ever encountered with ADAMS/Solver models involved one of these four things:

- Discontinuous functions
- Insufficient damping
- Poorly scaled models, or
- Poorly chosen error tolerances

Banish Discontinuities

Perhaps by now, you’ve grown tired of hearing it, but I can’t overemphasize the need to strip the model of all discontinuities.

A good place to start is to replace all IF FUNCTIONs with equivalent STEP s, IMPACT s, or BISTOP s, whichever is most appropriate. How, you may wonder, can a STEP replace an IF? Well, when ADAMS/Solver uses brute force to wade through the discontinuity of an IF, it does so by approximating it with a polynomial (more or less). In fact, that’s what the STEP-like FUNCTIONs were designed for. Their advantage is continuous first and second derivatives, which are essential to ADAMS/Solver. I contend that every IF function has an equivalent STEP function that can be used to replace it.
This is best illustrated by examples.

1. A simple on/off switch would be most intuitively modeled like this:
   \[
   \ldots, \text{FUNCTION} = \text{IF} (\text{TIME}-1.0, 0, 1)
   \]
   As far as ADAMS/Solver is concerned, this is numerically equivalent to:
   \[
   \ldots, \text{FUNCTION} = \text{STEP} (\text{TIME}-1, 0, 0, e1)
   \]
   The value of \(e\) should be determined by experimentation: make it as small as possible without causing a failure. Easy!

2. One cycle of a square wave is the difference between two \text{STEP}s:
   \[
   \ldots, \text{FUNCTION} = \text{STEP} (\text{TIME}-1, 0, 0, e, 1) - \text{STEP} (\text{TIME}2, 0, 0, e1)
   \]

3. One cycle of a saw-tooth wave is the difference between two \text{STEP}s, multiplied by a linear function:
   \[
   \ldots, \text{FUNCTION} = \text{TIME} \times (\text{STEP} (\text{TIME}-1, 0, 0, e, 1) - \text{STEP} (\text{TIME}2, 0, 0, e1))
   \]

4. Hysteresis can be modeled using a \text{SPLINE}:
   \[
   \text{SPLINE/id} \\
   , \quad x = 0, 1, 2, 3 \\
   , \quad y = 1, 0, 1, 2, 3 \\
   , \quad y = 1, 0, -1, -2, -3
   \]
   For additional examples, the best source I can think of is the textbook I used for the only Controls class I managed to pass. For a rule of thumb, understand that the following are ADAMS/Solver equivalents, with the exception that the \text{IF} is discontinuous:
   \[
   \text{IF} (\text{expr1}:\text{expr2}, \text{expr3}, \text{expr4})
   \]
   \[
   \text{STEP} (\text{expr1}, -e, \text{expr2}, (\text{STEP} (\text{expr1}, 0, \text{expr3}, e, \text{expr4})))
   \]
   Most (if not all) magnitude \text{FUNCTION}s (DM, VM, ACCM, WM, WDTM, FM, TQM) are not necessary. Seldom are they appropriate, unless they are squared, e.g.,
   \[
   \text{KE} = 1/2 \times \text{VM(I,J)} \times \text{VM(I,J)}
   \]
   Magnitude \text{FUNCTION}s have discontinuous derivatives near zero. It’s a common mistake to assume that your \text{FUNCTION} will never go through zero. Physically, that may be the case. Numerically, though, Newton-Raphson may take iterations that will produce temporarily unexpected values. If you don’t account for this, you may be in for some trouble.
The **DM FUNCTION** is usually employed to model something like a spring. First of all, if you're modeling a spring, you'll be money ahead to use the **SPRINGDAMPER** statement. Secondly, most forces can be modeled in such a way that $DZ(I, J, J)$ is a more appropriate value.

Kinetic energy is about the only thing I can think of that correctly uses the **VM FUNCTION**. **VM** should always be replaced by **VR**.

I have personally never had to do anything complicated enough to require using **WM**, **ACCM**, or **WDTM**. So my suggestion is to avoid them when possible, and watch out for them when it's not.

Angular **FUNCTION**s are even more difficult to predict, primarily since they depend so much on each other. For **AX**, **AY**, **AZ**, **PSI**, **THETA**, **PHI**, **YAW**, **PITCH**, and **ROLL**, there are two sources of discontinuity: the dreaded Euler singularity (a.k.a., gimbal lock) and certain limits on angular output ($\hat{\psi}$, $\hat{\theta}$, etc.)

There is no easy way to avoid angular discontinuities. Typically, a continuous function of angles must depend on trigonometric functions of all three angles (psi, theta, and phi, etc.). Euler angles and Rodriguez parameters are continuous. If you can write your function in terms of these parameters, and are willing to go the **RCNVRT** route, it might be worth the trouble.

The ideal, of course, is to control the angles yourself, but this will involve rather complicated subroutines. You will need to store past values of the angles to determine what their values should be after the discontinuity. Carefully constructed, this code could turn out to be a rather useful utility.

**Insufficient Damping**

High frequencies will only be a problem if they result in abnormally small step sizes. What do I mean by abnormally small? ADAMS/Solver probably won’t run into trouble for step sizes $\leq 1E-8$, which correspond to frequencies greater than 1 MHz.

You can generally determine the highest active frequency of the model by looking at the integration step size.

As a rule of thumb, assume that the integration step size is on the order of one tenth of the period of the highest active frequency.

Of course, a high frequency component shouldn’t be a problem if it belongs there. But if you really need to model something that requires such small integration step sizes, you’ll be money ahead if you change the time scale of your simulation. For example, you may want to use milliseconds as a unit of time, instead of seconds.
Appendix A. Ten Easy Steps to Happy Simulations

The real problem is when you have a high frequency that’s in there by mistake. This will most easily be found graphically. That’s why it’s so important to include graphics for each part. One of them is bound to be vibrating at a high frequency, and you’ll see it.

If you prefer a more analytical approach, insert $MREQ/id, DISP, PART = ALL$ in your data set. This will produce a displacement request for every part in the model. Then you can use the FFT (Fast Fourier Transform) capability of the postprocessor to transform these signals from the time domain to frequency domain. This should provide a general idea of the vibration modes and frequencies.

How can you remove the high frequency? Well, generally, an erroneously high frequency arises from forgotten damping values. Are you using undamped springs, fields, bushings, or beams in the model? What about things modeled with sforsces? You will need to determine if you have the proper damping characteristics.

Another possibility, albeit less likely, is insufficient inertia. Perhaps you “guessed” at some masses or inertias that have turned out too low. It may also be possible that you need to use constraints to remove some degrees of freedom altogether.

**Poor Scaling**

The scale of the model may be what’s wrong when there are no “tough” functions in the data set, but ADAMS/Solver still can’t converge. If there is a wide disparity in the magnitude of the dimensions of the model, ADAMS/Solver may not be able to maintain sufficient accuracy to converge within tolerance.

It used to be that the most likely cause of poor scaling was small time steps. This problem was removed in ADAMS 5.2.1. Just in case, you can monitor the condition number of the Jacobian matrix using the $WSTIFF$ statement/command. The smaller the condition number, the worse the scaling. Anything less than $1E-20$ is really bad.

While the time scale of the simulation occasionally needs to be changed, more often the problem is that some other dimension is out of scale. The pertinent dimensions are displacement, mass, force, and time. Scan the data set for the largest and smallest magnitudes (ignore zeros) of these values. Treat stiffness coefficients as forces. The ratio of the largest to smallest magnitudes gives some idea of the scale of the model. A scale of “1” is perfect (and also useless). As the ratio increases, so will your woes.
You may wish to consider changing the units used. For example, substitute meters for millimeters, etc. I realize that this is usually a pain in the neck, especially for velocities. But it is also generally computationally more efficient. The time spent to re-scale a model is usually recovered through a reduction in debugging and overall computing time.

While you’re rescaling the model, don’t forget about the GC argument on the ACCGRAV statement. This can occasionally be a quick and dirty scale fixer.

**Bad ERROR Tolerances**

I would be astonished if you got to this point without solving the problem, but it is possible, in theory at least, that the error tolerance you are using is too high or too low. A high tolerance may allow such an error accumulation that the system has lost all physical meaning. A low tolerance may not be achievable by numerical means.

Begin by increasing (or decreasing) your error tolerance by a factor of ten. Repeat this procedure until you attain similar approximations for two consecutive error tolerances. These results should be quite accurate.

**Conclusion**

That’s my bag of tricks in a nutshell. I have enjoyed sharing them with you. The bag is constantly growing, from techniques learned within MDI and from those our customers share with us. I’m eager to learn about your debugging techniques, as well. On the other side of the coin, of course, are you poor folks who obey all these commandments to the letter, but still cannot climb the mountain. When that happens, we on the hotline staff are here for you. If you have a problem that defies solution, call us - we’ve never run across a simulation that couldn’t be fixed.