IN-39
40344
P.129

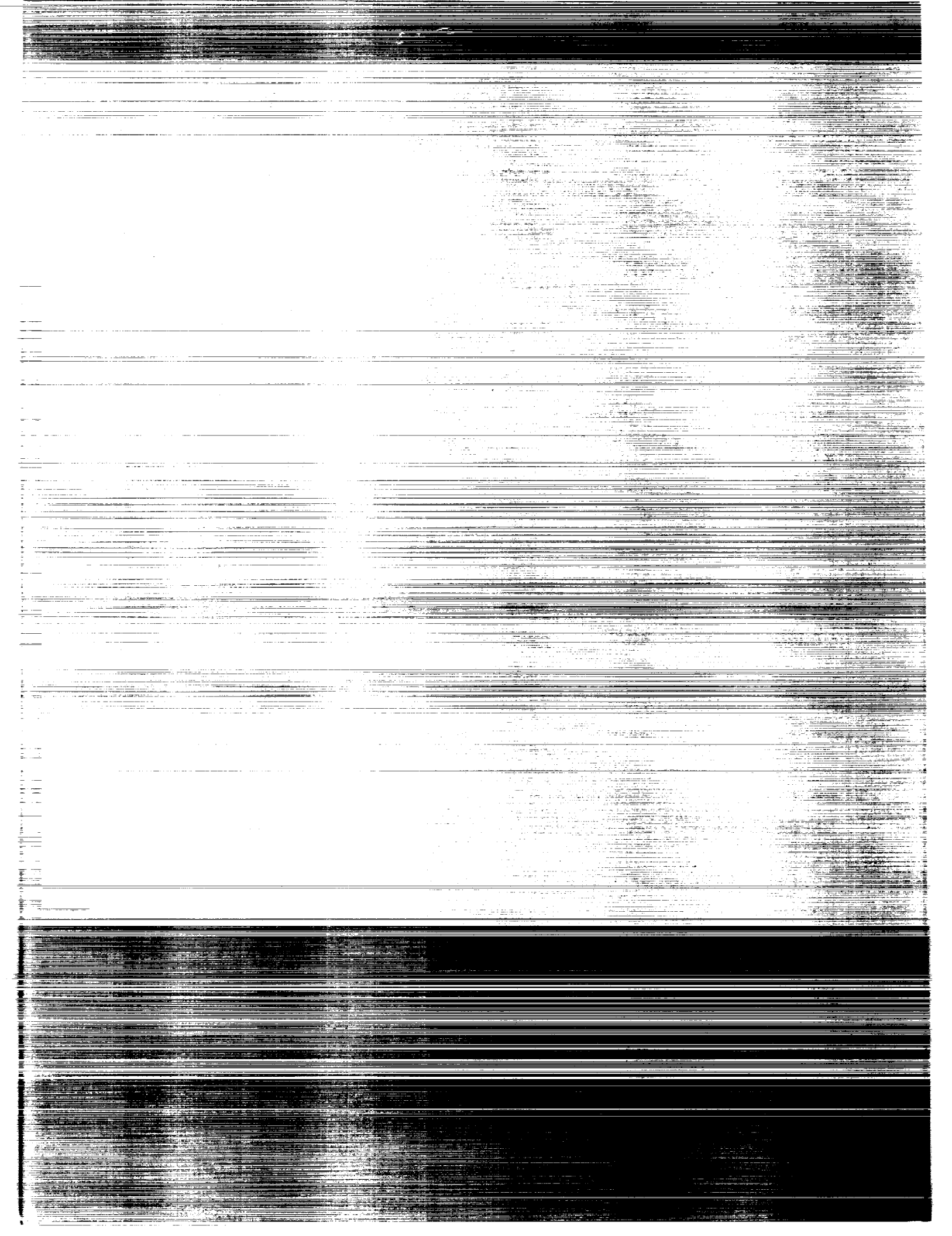
NASA Contractor Report 4401

Large Angle Transient Dynamics (LATDYN) User's Manual

A. Louis Abrahamson, Che-Wei Chang,
Michael G. Powell, Shih-Chin Wu,
Bradford D. Bingel, and Paula M. Theophilos

CONTRACT NAS1-18478
OCTOBER 1991

NASA



NASA Contractor Report 4401

Large Angle Transient Dynamics (LATDYN) User's Manual

A. Louis Abrahamson, Che-Wei Chang,
Michael G. Powell, and Shih-Chin Wu
COMTEK
Grafton, Virginia

Bradford D. Bingel and
Paula M. Theophilos
Computer Sciences Corporation
Hampton, Virginia

Prepared for
Langley Research Center
under Contract NAS1-18478

National Aeronautics and
Space Administration

Office of Management

Scientific and Technical
Information Program

1991

Acknowledgements

LATDYN is the work of many people. Without the creativity, persistance, and cooperation of each one of these individuals over a period of several years, this system of computer codes would not exist. The purpose of this preface is to list the major contributions of the development team members.

The research work of which LATDYN is a byproduct, was sponsored by NASA Langley Research Center and was initiated and monitored by Dr. Jerrold M. Housner. Dr. Housner was also responsible for initial development of the formulation on which LATDYN is based.

Primary responsibility for design, integration, and testing of the LATDYN system rested with COMTEK. Also, COMTEK was responsible for overseeing the project, for carrying out the development of the computational core of LATDYN, for developing the command language, and for writing the User's and Demonstration Problem Manuals.

Vital roles in LATDYN's development were played by four COMTEK employees, Dr. Che-Wei Chang, Dr. Shih-Chin Wu, Mr. Michael G. Powell and Dr. A. Louis Abrahamson. Dr. Chang and Dr. Wu jointly verified Dr. Housner's formulation making some modifications, and worked closely together to bring LATDYN to reality. In this activity, Dr. Chang was primarily responsible for designing and developing the computational core of LATDYN. Dr. Wu was primarily responsible for evaluation and verification of results, and wrote the Demonstration Problem Manual. Mr. Powell assisted with the design of user interfaces, coordinated communication between the different code development teams and implemented the interface between the computational core of LATDYN and Pre- and Postprocessors. Dr. Abrahamson managed the project, developed much of the command language and postprocessor functional user interface, and is the principal author of this manual.

The Preprocessor was developed by Computer Sciences Corporation (CSC) under the direction of Mr. Bradford D. Bingel, who was assisted in testing by Ms. Paula M. Theophilos. Mr. Bingel also played an advisory role in the syntactical design of the LATDYN command language, and wrote the Preprocessor guide.

The Postprocessor was developed and tested singlehandedly by Ms. Maria V. Mitchum of NASA Langley Research Center, using command decoding routines provided by Mr. Bingel. Ms. Mitchum also wrote the Postprocessor guide.

Table of Contents

1. INTRODUCTION

LATDYN is a computer code for modeling the **L**arge **A**ngle **T**ransient **DYN**amics of structures. It also includes special facilities for modeling control systems and for programming changes to the model which may take place during an analysis sequence.

The concept for LATDYN had its origins in research work on computational structural dynamics and control-structure interaction sponsored by NASA Langley Research Center since the mid-1980's. This work had as its goal the achievement of a better understanding of the technology needed for designing the structure and control systems of future spacecraft. From this perspective LATDYN should not be viewed as an end product in itself but rather as a byproduct of research whose goal was technology development.

As a result LATDYN's scope is limited. For example, it does not have the wide library of finite elements that would be found in a commercial code, and 3-D Graphics are not available. Nevertheless, LATDYN does have a measure of the "user friendliness" that has come to be associated with commercial products. There are two reasons for user-friendliness in LATDYN and these are as follows. First, one objective of LATDYN development was to demonstrate ways in which a user interface might be structured to incorporate controls modeling in a transient multi-body dynamics code. A second objective was to make the tool usable by a spectrum of other researchers so that the new techniques incorporated in LATDYN could be widely evaluated.

Other specific objectives in LATDYN development were to,

- ❑ develop and evaluate different formulations in multi-body dynamics,
- ❑ find ways to bring three different disciplines (structural analysis, multi-body dynamics, control system analysis) together in one computational tool,
- ❑ produce a product with some lasting value to research in the field.

The purpose of this manual is to document the status of LATDYN at the end of COMTEK's contract with NASA, and to describe how the code may be used.

BACKGROUND

LATDYN has its roots in three disciplines that have historically been separate:

- ❑ Finite Element Structural Analysis,
- ❑ Multi-Body Dynamics, and
- ❑ Control System Analysis.

At the beginning of the research work which spawned LATDYN, this separation of critical disciplines was percieved as a barrier to the efficient design of structures and control systems for spacecraft. For example, a structure might be designed to be stiffer than otherwise required in order that its flexible modes should not lie within the frequency range of control system response.

The evolution of the three disciplines was briefly as follows. Since the late 1960's almost all structural analysis of aerospace vehicles has been performed using finite element tools, but even up to the mid-1980's finite element analysis tools did not have the capability for modeling the large dynamic motions which spacecraft commonly experienced. Over approximately the same period that the finite element method assumed its dominance in structural analysis, research in the mechanics of interconnected rigid bodies was stimulated by needs in the auto industry and elsewhere, and grew to produce a new discipline encompassing large motions of rigid bodies connected by joints that came to be called "multi-body dynamics". Later, flexible body capability was added by a normal mode approximation. Lagging somewhat behind the other two disciplines but rapidly coming to fruition, a third development was taking place. Computer software embodying numerical solvers and using a symbolic user interface were being combined with modern control formulations to yield a range of new tools which greatly simplified the simulation and design of control systems.

From separate beginnings, which were partly the result of historical accident, came organizational barriers which tended to reinforce boundaries between the disciplines, and to further restrict technical communication between practitioners of these disciplines. This situation provided the impetus and stimulated the research of which LATDYN is a byproduct.

The precurser to the 3-D version of LATDYN, which is described herein, was 2-D LATDYN [1]. This code was developed first in order to gain experience with the wealth of new technology being generated. Here, the name LATDYN refers to the 3-D version only, which although it shares much of the same philosophy of the original 2-D version, is an entirely new code with many new features. The Command Language described in this document is derived from the 2-D Command Language but has been enhanced and changed to reflect the needs of a 3-D modelling system

CAPABILITIES

LATDYN extends and brings together some of the aspects of the three disciplines mentioned above, combining significant portions of their distinct capabilities into a single analysis tool. However, this claim should not be misconstrued.

LATDYN is a research code. It does not have the facilities and range of features commonly associated with commercial codes. It was developed to illustrate how the interdisciplinary barriers discussed above may be overcome. It is informative however, to contrast LATDYN's capabilities and formulation with those of existing commercial codes because these codes embody the disciplinary divisions discussed above.

The formulation for flexible bodies in LATDYN is described in the references [2] [3] [4] [5], and is grounded in conventional finite element methodology . That is, the structure to be mod-

---

[1] Housner, J.M., McGowan, P.E., Abrahamson, A.L., and Powell, M.G., "The LATDYN User's Manual", NASA Technical Memorandum 87635, January 1986.

[2] Housner, J.M., Wu, S-H, Chang, C-W, and Abrahamson, A.L., "A Finite Element Method for Time Varying Geometry in Multi-body Structures", Paper presented at the 29th Structures, Structural Dynamics, and Materials Conference, Williamsburg, Virginia, April 1988.

eled is discretized into a number of finite elements with specific known properties. In finite element programs such as NASTRAN, EAL, ANSYS, and MARC, there are large libraries of element types to enable many different types of structures to be modeled. At present, because it is a new research code, the only continuous flexible elements that 3-D LATDYN contains are beam elements.

The finite element formulation for flexible bodies in LATDYN extends the conventional finite element formulation by using a convected coordinate system for constructing the equations of motion. Conventional finite element formulations do not incorporate the full non-linear inertial effects associated with large displacements and rotations. LATDYN's formulation allows for large displacements and rotations of finite elements subject to the restriction that deformations within each element are small. Large structural deformations may still be modeled by using a denser finite element discretization so that the small deformation approximation holds within each element (see references 2-5).

For rigid bodies and joints LATDYN borrows extensively from methodology used in multi-body dynamics. That is, rigid bodies may be defined and connected together through joints (hinges, ball, universal, sliders, etc.). LATDYN also includes both types of conventional multi-body joint formulations. Joints may be modeled either by constraints (as in ADAMS, DADS, and DISCOS) or by adding joint degrees of freedom (as in TREETOPS).

In multi-body dynamics codes such as ADAMS and DADS it is also possible to combine rigid bodies with flexible ones, flexible bodies being modeled through a selected set of normal modes. Selection of modes is a problematic procedure (even in apparently simple cases as shown in references 2 and 4), due to the difficulty of choosing appropriate boundary conditions for modal determination and in selecting an appropriate subset of modes for the analysis. The selection process is compounded beyond that associated with modal synthesis techniques for small vibration analysis. Furthermore, the use of modes provides no determinable convergent path by which solutions can be checked. That is, adding more modes does not mean that the solution will converge to the correct answer. On the other hand, the finite element approach implemented in LATDYN provides a convergent path for checking solutions simply by increasing mesh density.

Control systems are often modeled using a symbolic language to perform mathematical operations such as matrix manipulations and numerical integration in programs such as MATRIX-x and EASY5, typically using normal modes from a finite element code such as NASTRAN to represent the structure. In some situations of practical current interest, this procedure lacks efficiency and may cause errors. For example, where the structure is changing its configuration throughout the controlled motion, such as in the case of a moving robot arm, or during the deployment sequence of a spacecraft component , the normal modes also vary throughout the motion. Thus, the separation of structural analysis and control analysis in this way becomes cumbersome.

---

[3] Housner, J.M., Wu, S-H, and Chang, C-W, "Controlled Multi-body dynamic Simulation for Large Space Structures", Paper presented at NASA/DOD Controls-Structure Interaction Conference, NASA CP3041, January 1989.

[4] Wu, S-H, Chang, C-W, and Housner, J.M., "Dynamic Analysis of Flexible Mechanical Systems Using LATDYN", Paper presented at 3rd Annual Conference on Aerospace Computational Control, Oxnard, California, August 1989.

[5] Chang, C-W, Wu, S-H, and Housner, J.M., "A Finite Element Approach for the Dynamic Analysis of Joint Dominated Structures", NASA Technical Memorandum to be published.

LATDYN solves this problem by providing symbolic capabilities for modeling control systems which are integrated with the structural dynamic analysis itself. Its command language contains syntactical structures which perform symbolic operations and which are also interfaced directly with the finite element structural model, bypassing the modal approximation. Thus, when the dynamic equations representing the structural model are integrated, the equations representing the control system are integrated along with them as a coupled system. This procedure also has had the side benefit of enabling a dramatic simplification of the user interface for modeling control systems. Rather than dealing with an abstract set of modal parameters, the user is now able to deal directly with the finite element model of the structure in programming a control system. LATDYN's interface includes symbolic operators representing familiar mathematical and laboratory constructs that are the basis of a complex control system. The way in which these programming features have been implemented is relatively open ended because they are written by the user in a form of psuedo-FORTRAN which is converted into standard FORTRAN 77 by the LATDYN Preprocessor.

STATUS

LATDYN* is the first code to use flexible finite elements which can rotate rapidly through large angles while retaining accurate modeling of inertial effects, and also to allow these finite elements to be connected by joints and rigid bodies. The algorithms and code in LATDYN's computational core which embodies this formulation, have been exhaustively checked against other multibody formulations, and we have high confidence in their correctness.

At a higher level, the code has been less well checked. LATDYN has more than seventy commands, each command has several parameters, there are multiple ways to specify many parameters, and there are thousands of combinations of ways in which data may be specified. It has not been possible for us to check them all. We make no apology for this because LATDYN has been developed in a research environment and is not a commercial product. As the code matures through use, we expect confidence in the whole to grow. However, this is not within COMTEK's control and will depend on NASA's maintenance policies for the code.

A further comment on the status of the code is appropriate. One prime objective of the research was to show **how** to accomplish the task of structuring a multi-body finite element code with the intrinsic structures necessary for detailed modeling of control systems. Our work in this regard extended by a significant margin that which was ever actually planned for implementation in the computer code. Nevertheless, many of these advanced ideas are embodied in the command structures described in this manual. Although our work is finished, we have specifically included references to advanced items which were not planned for full implementation. The reasons for leaving these structues in place in the manual are twofold, to accurately represent the work that was done, and to assist any future development.

For example, while our formulation included the capability for hinge (revolute) joints by transformations (adding degrees of freedom) and was never planned to be extended to include universal or ball joints by this method, such a procedure is clearly possible and has

* The code is available through COSMIC and can be obtained by writing to COSMIC, Office of Computing Administration and Planning, Computer Services Annex, The University of Georgia, Athens, GA, 30602.

been included in the command structure and has been built into the Preprocessor. (Note that a LATDYN user can still model universal joints and ball joints by constraints).

Also for example, while rigid body chains linked together by joints and connected to flexible finite elements are clearly possible by recursive implementation of the joint and rigid body transformations included in our formulation, such a recursive structure was never planned for implementation in LATDYN. What might not be clear however, is how such a recursive formulation might be included in a command structure. This problem was solved and is shown in the RMEMBER and RBODY commands as "options in analysis algorithm". These options also include reference to an advanced "auto" capability where the program could itself choose the most efficient algorithm and recursion path to be used.

THE FUTURE

The U.S. Space Program is poised on the brink of exciting new developments. In his space policy speech in mid-1989 President George Bush reaffirmed the Nation's commitment to achieve a permanent manned operational space station in the 1990's. He also set future goals for lunar and planetary exploration as well as missions to planet earth. NASA is currently moving forward to establish programs to meet these goals.

Even at this early planning stage, it is apparent that fulfilment of these broad goals requires the construction, operation and maintainance of large spacecraft. These large spacecraft will include will include the low earth orbit infrastructure necessary to support planetary exploration missions, as well as planetary exploration vehicles themselves and geostationary facilities.

Construction, operations, and maintaince of large spacecraft all involve large angular motions. These motions may consist of pointing the entire spacecraft or articulation of individual components. Analysis of such motions for large structures, is beyond the routine capability of conventional analytical tools without simplifying assumptions. In some instances the motion may be sufficiently slow and the spacecraft (or component) sufficiently rigid to simplify analyses of dynamics and controls by making psuedo-static and/or rigid body assumptions.

In the general case however, each of the three phases of spacecraft performance require analyses that can account simultaneously for flexibility and the inertial effects of the angular motion. The ability to perform such analyses may affect not only the design of the structures themselves and the spacecraft control systems, but can also have a dramatic impact on the speed at which operations could be carried out in space. For example, the remote manipulator arm on the Space Shuttle commonly requires a settling time in excess of ten minutes after each manipulation to allow the vibration to subside.

There is a clear need for new tools to analyse flexible large-angle dynamics and controls, for growth versions of Space Station "Freedom", for the Human Exploration Initiative, and for Mission to Planet Earth. The research work which spawned LATDYN is one step towards the development of such tools, but more needs to be done.

Large angle transient dynamics is inherently computationally intensive. A tool that made the most of existing technology would include a recursive version of the present LATDYN, add a more diverse range of finite elements (such as a beam with bending and torsion but no extension, a plate, and a thin shell), the capability of representing particular flexible components by component modes, bandwidth minimizing routines, advanced integrators, and vectorization / parallelization for supercomputers. In addition, a facility that allowed a user to easily change the level of refinement (more or less elements, more or less modes) and also the way in which components of a structure were modeled (finite elements or component modes) would be very useful.

In addition, methods for designing control systems for flexible articulating structures are not well developed. This appears to be an area where substantial research is needed.

HOW TO USE THIS MANUAL

In this Manual, LATDYN commands are laid out in a sequence which with a few exceptions, follows that which would be used in creating a data case for a transient analysis. That is, tools for model setup are described first, next commands for defining the structural model, then constraints, data tables, applied loads, and initialization of dynamic variables. Instructions for setting up, for controlling, and for programming a transient analysis come last.

For a new user of 3-D LATDYN, there are some important concepts which need to be understood and the ordering of the manual is more useful for reference than for tutorial purposes. For this reason the next section contains a tutorial to guide a new user through some simple examples. Perhaps the best way to gain quick familiarity with LATDYN is to read Section 2, quickly skim the rest of this Users Manual, and then work carefully through the case studies presented in the Demonstration Problem Manual, referring to the Users Manual for detailed explanations of the various commands used there.

For experienced users of other finite element codes there are some particular differences that need to be understood before beginning to set up a model. These topics are also covered in the next section and involve:

- o The different types of points to which structural entities may be connected and the implicit relationships that the point naming conventions imply,
- o Coordinate systems which may be created and used for setup of the model and evaluation of the analysis,
- o User defined Conditions, Variables, Vectors, and Operations.

2. TUTORIAL

The LATDYN System has three parts as shown in Figure 2.1. Each part is a separate computer program which communicates with the others by information stored in files.

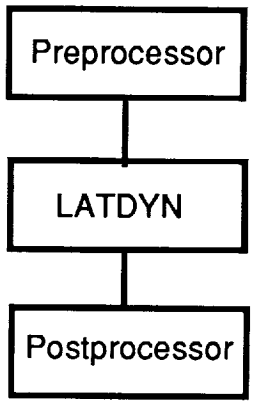


Figure 2.1 The LATDYN System

To show how use LATDYN and to illustrate its structure, a simple example is reviewed here. This example is not intended to show LATDYN's capabilities. The Demonstration Problem Manual contains examples which better illustrate the extent of LATDYN's power.

SIMPLE EXAMPLE

"A rigid projectile with a mass of 100 Kg, is accelerated to a velocity of 10 meters per second by applying a force of 100 Newtons."

LATDYN execution requires four things:

Setting up the input data:
This can be done directly by typing data into the Preprocessor program, but a much better way is to use the editor in your computer system to create a file containing your input data - then you can modify this file later, without having to retype the whole thing.

Running the Preprocessor program:
This program reads and interprets your input data and prepares input files for LATDYN. The Preprocessor is quick and is usually run interactively. It will catch most syntactical and cross-referencing errors that you make.

Running the LATDYN program:
For this example, LATDYN will run quickly and can easily be run interactively. Usually you will want to run it batch, and for large problems you may want to use a supercomputer if available.

Running the Postprocessor program:
This program reads LATDYN output data and plots the results in a variety of ways. It also has facilities to allow you to manage and compare data from several runs, and from other external sources.

Setting up the LATDYN Input Data

To set up the input data for this example you need to:

(1) define the properties of the lumped mass
(2) define a gridpoint
(3) "stick" the mass onto the gridpoint
(4) define the force
(5) specify the transient analysis

So lets's do it!

(1) The command to define the properties of a lumped mass is MASSPROP.

```
$           massname      mass   C.G.    Ix,Iy ,Iz      Ixy,Ixz,Iyz
MASSPROP:   Projectile    100    0,0,0   1,1,1          0,0,0
```

There are several things to notice about this command. Let's take them one at a time. First there is the format. You'll notice the line which starts with a "$" sign, it is a comment. Comments are ignored by the Preprocessor. Here we have used it to label the command parameters, but you can use comments to annotate your input data any way you wish. The rules on comments are found in Chapter 4.

Next there is the command itself. It begins with the key word MASSPROP. This keyword tells the Preprocessor that the following data is to define lumped mass properties. The data itself is divided into parameters. A parameter is separated from other parameters by a comma and/or spaces. The rules and conventions for commands are also in Chapter 4.

We have chosen to give the mass the name "projectile". The name is just for our use and it could be any combination of numbers and letters, but two lumped mass properties cannot have the same name.

The next parameter is the mass itself, which is 100Kg. The command also requires specification of the center of mass. Let's place it at the origin.

The next parameters are rotational mass properties, which are not given since rotational motion is not anticipated. LATDYN has to have non-zero values (otherwise its

mass matrix will be non-positive definite) so let's give arbitrary values of 1 Kg x M^2. Let's also assume that the mass is a sphere so that the products of inertia are all zero.

(2) The command to define a gridpoint is described in Chapter 7. We choose to define one at coordinate location (0,0,0) in the global coordinate system.

$	gridpoint #	x,y,z	Coordinate System
GRIDPT:	#G1	0,0,0	GLOBAL

What is a gridpoint and why do we need one?

A very important concept in finite element methodology is that of gridpoints or nodes. In most finite element programs for structural modeling, a gridpoint is set up as a ficticious entity in space. The finite element model is then constructed by connecting finite elements between gridpoints. For this problem we don't need gridpoints for connecting finite elements, but we do need one to which to attach the mass.

In LATDYN gridpoints are preceeded by the "#" symbol and the letter "G". If a gridpoint is defined as part of a component, then the component ID is also present. The rules and conventions for gridpoint names are given in Chapter 4.

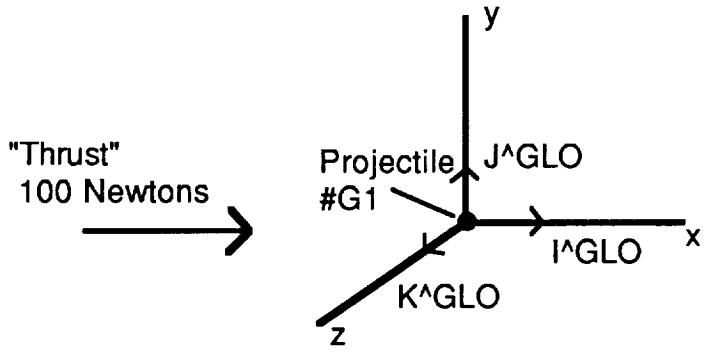


Figure 2.2 Coordinate System and Unit Axis Vectors for Projectile Example

(3) The command to create a mass and "stick" it onto a gridpoint is ADDMASS. It is defined in Chapter 7. We can use it to add a mass to #G1 with the properties which we defined in step (1).

$	gridpoint #	mass prop.name	coordinate system
ADDMASS:	#G1	Projectile	

The ADDMASS command also asks for the name of a coordinate system. We are using the global coordinate system, so we can just leave it blank since that is the default coordinate system.

(4) A force is specified by the APPFORCE command. It is described in Chapter 7. No direction for the motion is specified in this example, so let's assume a direction along the global x-axis.

```
$                force name      magnitude, direction      duration          application point
APPFORCE:   Thrust          MSDV(100, I^GLO)     TIME(0.0, 10.0)      #G1
```

There are many OPTIONs for specifying data with this command. We chose the MSDV option for specifying the magnitude and direction of the force. MSDV stands for **M**agnitude-**S**calar-**D**irection-**V**ector. The magnitude of the force was given as 100 Newtons, and we assumed that the direction of the force was going to be along the global x-axis.

At this point we meet another of LATDYN's special parameters - a unit axis vector. Whenever a coordinate system is defined, its unit axis vectors (**i,j,k**) are also automatically defined. Since the GLOBAL system does not have to be defined, its axis vectors already exist. We can use them as (I^GLO, J^GLO,K^GLO) for (**i,j,k**). (Note that "GLO" is allowed as an abbreviation for "GLOBAL".)

As for the duration of the applied force, we cheated a little and calculated the amount of time (10 seconds) that it would take for the force to accelerate "Projectile" to a velocity of 10 Meters per second. In the next example we'll show how *you can let LATDYN test when the velocity of the projectile has reached the required level and turn off the force automatically*.

(5) The commands to specify the transient analysis are not complicated to implement. You will always need a TITLE,an INTEGRATOR, a TIMESTEP, and a TIMESPAN command. These are required commands and the preprocessor will not let you run LATDYN without them. Let's also add a PRINT and a PLOT command. All of these commands are described in Chapter 10. We will put the four required commands at the beginning of the input file, and the print and plot specifications at the end.

```
$                Title
TITLE:  First LATDYN Tutorial Example
$            Integrator Type
INTEG:     EXPLICIT
$             step size in seconds
TIMESTEP:  .05
$             start time   stop time
TIMESPAN:   0.0        12.0
$                massname          mass     C.G.      Ix,Iy ,z        Ixy,Ixz,Iyz
MASSPROP:  Lump             100     0,0,0      1,1,1          0,0,0
$            gridpoint #      x,y,z          Coordinate System
GRIDPT:     #G1          0,0,0          GLOBAL
```

```
$              gridpoint #      mass prop.name
ADDMASS:    #G1          Lump
$                 force name       magnitude, direction        duration          application point
APPFORCE:    Thrust       MSDV(100, I^GLO)    TIME(0.0, 10.0)      #G1
$              print once every second
PRINT:      TIME( 1 )
$              write plot data at every step
PLOT:       STEP( 1 )
$
END
```

Well that's it! Now you are ready to run the Preprocessor.

Running LATDYN and the Postprocessor

To run LATDYN on a VAX-VMS System, you type

LATDYN filename.

where "filename" is the name of your input data file. The procedure "LATDYN" loads the Preprocessor and starts it executing your input file. The Preprocessor generates two main output files. One contains data to be read into the LATDYN Program, the other contains FORTRAN subroutines generated by the Preprocessor for linking to LATDYN. These subroutines are called PRECALC, BGNSTEP, INLOOP, and ENDSTEP. They are called by LATDYN at specific points in the integration loop (see Chapter 13).

The LATDYN Procedure then does a test compilation of these subroutines to make sure that they are syntactically correct, then it links them to the LATDYN Program object code and submits it as a batch job, since the LATDYN Program usually takes too long to run interactively.

The LATDYN Program writes two main output files, a print file and a plot file. The plot file is specially intended for use with the Postprocessor. The LATDYN Postprocessor is a very user friendly program which allows you to plot and keep track of your data. To start it type

POSTPROC

The plot file which is generated by the LATDYN Program will have the same name as your input file appended with the letters ".PLT". For example, if your input file was PROJ, then the plot file will be called PROJ.PLT. The Postprocessor command to open this file is simply

```
OPEN  PROJ.PLT
```

You could leave the ".PLT" suffix off the filename in the OPEN command because this is the default suffix for a LATDYN plot file. That is, the Postprocessor will assume that your plot file will have a .PLT suffix unless you specify otherwise.

There are many functions which you can now plot as you will see in the Postprocessor Manual. An easy way to find out what commands are available, and what each one does is to type

```
HELP
```

The HELP command brings up a menu on the screen showing a list of all commands. You can get more information about each one by now typing the name of that command. To see a plot of the x-displacement of the projectile versus time, type

```
PLOTX  D  #G1
```

You should now see the requested plot on the screen.

Another very useful feature which is built into the Postprocessor enables you to compare different types of plotted output. To introduce this feature type

```
STORE
```

to store the previous plot in a temporary file called the "storfile". Then type

```
STORON
```

to automatically store all future plots that you create in this file. Now generate a plot of the velocity of the projectile and then one of its acceleration by typing,

```
PLOTX  V  #G1
PLOTX  A  #G1
```

To look at the storfile contents type

```
INDEX
```

and you will see that there are three plots stored there. You can plot all these together on a single plot by typing

```
PS2  1 2 3
```

to compare the displacement, velocity, and acceleration of the projectile versus time.

SIMPLE EXAMPLE VARATION

LATDYN has programming capabilities which could conveniently have been used in the "Projectile" example. We can use these by modifying the APPFORCE command used before by omitting the time duration and making it a conditional command as follows:

```
$                force name      magnitude, direction      application point    condition label
APPFORCE:        Thruster        MSDV(100, I^GLO)          ,,   #G1             ? C1
$
C1:     X(V,#G1) .LT. 10
```

then the force will be subject to condition "C1" and will remain in effect only while that condition is TRUE. Condition "C1" is then defined such that it is true while the x-velocity of gridpoint #G1 is less than to 10 meters per second. When the x-velocity of gridpoint #G1 exceeds this value then the force will turn off automatically.

If you know how to program in FORTRAN then you may recnogize the ".LT." structure as being the same as that which occurs in a FORTRAN "Logical If" to mean "LESS THAN". This is no accident because LATDYN conditions are one of those commands which get translated into FORTRAN by the Preprocessor.

3. TERMINOLOGY

There are many new ideas that have gone into designing LATDYN and it was necessary develop a terminology to describe them. To help with understanding, this section contains a list of LATDYN terminology in alphabetical order with a brief explanation of each term. Some words and concepts may be familiar but the way that they are used may be slightly different from usual. Also included is a list of special characters used in the command syntax with an explanation of their usage.

DEFINITIONS

Attached Reference Entity

An attached reference entity is a reference point, a reference vector, or a coordinate system that is attached to a gridpoint and moves with it during the transient analysis.

Ballpoint

A ballpoint is an element connection point on one side of a ball joint.When a ballpoint is defined at a gridpoint then the gridpoint/ballpoint pair may be regarded as the two sides of a ball joint. Definition of a ballpoint adds three degrees of freedom to the dynamic system. Beam members may be connected to either side of the joint. Possible formats for a ballpoint are #GjBk, #CiGjBk, or #C"name"GjBk where i,j,k are integers.

Beam Member

A flexible beam member is made up, either of a single flexible beam finite element, or of a string of identical flexible beam finite elements. A rigid beam member is a single rigid body with the inertial properties of a rigid beam. The format of a beam member ID is #Mj, #CiMj, or #C"name"Mj where i an j are integers.

Condition

A condition is a user defined logical statement written in a FORTRAN-like syntax that may be used to test for events which may occur during the transient analysis. A condition is referenced by its condition label.

Condition Label

A condition label is the name by which a user defined logical condition is referenced. When appended to certain commands it makes the action of that command subject to the logical condition. This action will depend on whether the command is singular or non-singular. The format of a condition label is Ci or CLi where i is an integer between 1 and 99. When a condition label is appended to a command, then it is preceded by a question mark "?". The question mark may be read as "IF".

Condition Variable

A condition variable is a logical variable which is similar to a FORTRAN logical variable which may appear in conditions. Every user defined condition has a logical condition variable associated with it. A condition variable has the form CLi, where i is an integer between 1 and 99.

Component

A component name is simply an extra identification which can be a number or a name (enclosed in quotes) that can be added to gridpoint and member ID's. When this is done, then that item is regarded as a part of the specified component.

Constraint

A constraint is an equation which removes degrees of freedom from the system. There are many types of constraints in LATDYN. For example, the HINGEJOINT, BALLJOINT, UNIJOINT, CYLJOINT, and TRANSJOINT commands set up joint constraint equations. User defined constraint equations may be created by the SDFC and MDFC commands.

Coordinate Axes

Coordinate systems are defined by the AXES command and may be attached to gridpoints by the ATTACH command. When a new coordinate is defined, it is given a name by the user so that it can be referenced in subsequent commands. When a coordinate system is defined then a set of unit axis vectors are automatically created and can be referenced by the user

Flexible Element

A flexible element is a finite element which is connected between gridpoints and/or jointpoints. A flexible structure is modeled by an assemblage of finite elements representing each part of the structure.

Function

A User Function in LATDYN calculates a single value and is similar in syntax and operation to a FORTRAN function. LATDYN has a wide variety of special user functions to calculate parameters associated with the dynamic model, in addition to all of the standard FORTRAN functions which are automatically available. User Functions may be used in Q-variable definitions and in Conditions.

Gridpoint

Gridpoints are the basic conceptual entities on which a structural model in LATDYN is built. As in most finite element programs for structural modeling, a gridpoint is created by the user as a ficticious entity in space. When it is initially created each gridpoint has six degrees of freedom. A finite element model is constructed by connecting finite elements between gridpoints. In LATDYN jointpoints may also be defined at a gridpoint. Possible formats for a gridpoint are #Gj, #CiGj, or #C"name"Gj where i and j are integers.

Hingepoint

A hingepoint is an element connection point on one side of a hinge (or revolute) joint.When a hingepoint is defined at a gridpoint then the gridpoint/hingepoint pair may be regarded as the two sides of a hinged joint. Definition of a hingepoint

adds one degree of freedom to the dynamic system. Beam members may be connected to either side of the joint. Possible formats for a hingepoint are #GjHk, #CiGjHk, or #C"name"GjHk where i,j,k are integers.

Jointpoint

Jointpoints are the collective name for finite element connection points which are connected to gridpoints by a joint transformation (HINGEPOINT, UNIPOINT, BALLPOINT etc.). That is, when a jointpoint is defined by the user only those degrees of freedom which are necessary to define the joint are added to the system. This is different to creation of a joint by constraints (see Joint Constraint).

Joint Constraint

A joint between two gridpoints may be created by constraints. Initially each gridpoint will posess six degrees of freedom, some of which will be removed by the joint constraints. Examples of commands to set up joint constraint equations include HINGEJOINT, BALLJOINT, UNIJOINT, CYLJOINT, and TRANSJOINT.

LATDYN Program

The FORTRAN computer program which is the computational core of the LATDYN System.

LATDYN

The collective name for the three FORTRAN computer programs which make up the LATDYN System. The three programs are the LATDYN Program, the Preprocessor, and the Postprocessor.

Member

A flexible or rigid structural beam member. A flexible beam member is made up, either of a single flexible beam finite element, or of a string of identical flexible beam finite elements. A rigid beam member is a single rigid body with the inertial properties of a rigid beam. The format for members is #Mj, #CiMj, or #C"name"Mj where i and j are integers.

Operator

An Operator in LATDYN performs multiple actions and/or calculates and assigns multiple values. It is similar in syntax and posesses the full diversity of operations as a FORTRAN subroutine. LATDYN has a wide variety of special user Operators to create, calculate, and manipulate Q-vectors from parameters associated with the dynamic model.

Option

An OPTION in a LATDYN command signifies that there is more than one way to specify the information. An OPTION consists of a keyword followed by one or more parameters enclosed in parentheses.

Postprocessor

The FORTRAN computer program which allows a user to interactively plot and manage a database of LATDYN transient analysis results. It is one of the three programs in the LATDYN System.

Preprocessor

The FORTRAN computer program which translates user commands into a form which can be used by the LATDYN Program. The Preprocessor also performs error checking on user input. It is one of the three programs in the LATDYN System.

Q-Variable

A Q-variable is the generic name given to user defined variables in LATDYN. The name comes from the requirement that all user defined variables begin with the letter "Q". A Q-variable is defined in one of the several types of SET commands, and the format of its definition appears very much like a FORTRAN assignment statement. The right hand side of the assignment may include any standard FORTRAN arithmetic operation, standard FORTRAN functions, and also special LATDYN User Functions. Q-variables may be used as parameters in a number of LATDYN commands, such as applied loads and constraints. They may also appear in other Q-variable assignments. Q-variables are one of the basic tools which enable a user to program control systems. The format for global Q-variables is Qk where k is an integer betwen 1 an 99. The format for local Q-variables is Qname where name is any alphanumeric string not exceeding six characters.

Q-Vector

A Q-vector is the generic name given to user defined vectors in LATDYN. The name comes from the requirement that all user defined vectors have I.D. that begin with the letter "Q". A Q-vector is defined and manipulated in one of the several types of OP commands using standard LATDYN Operators. Q-vectors may also be used as parameters in a number of LATDYN commands, such as applied loads for example. Q-vectors are one of the basic tools which enable a user to program control systems. The format for a Q-vector I.D. is Q^k where k is an integer between 1 and 99.

Reference Point

A Reference Point is used as a reference location in space. It may be used both in setting up a model and as a reference location for measurement during a transient analysis. A reference point may remain stationary, or it may be attached (see ATTACH command) to move with a gridpoint. A reference point adds no degrees of freedom to the system, and finite elements may not be connected to it. Possible formats for reference points are #Rj, #CiRj, or #C"name"Rj where i and j are integers.

Reference Vector

A Reference Vector is a reference quantity that has both direction and magnitude. It may be used both in setting up a model and as a reference for measurement during a transient analysis. A reference vector may remain stationary, or it may be attached (see ATTACH command) to move with a gridpoint. The format for a reference vector I.D. is R^k where k is an integer.

Rigid Body

A rigid body may consist of one or more gridpoints whose relative relationship is fixed. A rigid body has no inherent inertia until masses are attached to one or more of its gridpoints using the ADMASS command.

Singular Command

Some commands are termed singular because only one version of the command can be active at a time. If multiple versions of a singular command are given, then all versions of the command except the first must be conditional (that is they must be qualified by a condition label). The order is important, because LATDYN chooses the last one whose condition is true to be the active command.

Timestep

A timestep is the discrete increment of time for one step in the transient analysis for numeric integration of the differential equations of the users LATDYN model.

T-Variable

Every data table in LATDYN has a T-variable automatically associated with it. The value of the T-variable represents the interpolated value of the dependent variable in the table at the present timestep. A T-variable may be used in most places that a Q-variable may be used. The format for referencing a T-variable is Tk where k is an integer.

T-Vector

Every vector table in LATDYN has a T-vector automatically associated with it. The magnitude and direction of the T-vector represents the interpolated values of the dependent variables in the table at the present timestep. A T-vector may be used in most places that a Q-vector may be used. The format for referencing a T-vector is T^k where k is an integer.

Unipoint

A unipoint is an element connection point on one side of a universal joint. When a unipoint is defined at a gridpoint then the gridpoint/unipoint pair may be regarded as the two sides of a universal joint. Definition of a unipoint adds two degrees of freedom to the dynamic system. Beam members may be connected to either side of the joint. Possible formats for a unipoint are #GjUk, #CiGjUk, or #C"name"GjUk where i,j,k are integers.

Unit Axis Vector

A unit axis vector lies along each of the three coordinate axes of every coordinate system. These are identified by the letters (I,J,K) for the Cartesian coordinates (x,y,z) respectively. A unit axis vector is a reference entity which may be used for a direction reference in a similar way that reference vectors may be used When a coordinate system is defined then a set of unit axis vectors are automatically created and can be referenced by the user as I^axesname, J^axesname, K^axesname. The global system axis vectors I^GLO, J^GLO, K^GLO are predefined.

User Subroutine

A user subroutine is a receptical for the Psuedo-FORTRAN commands specified by a user in SET, OP, and CLj commands.

SUMMARY OF SPECIAL CHARACTERS AND THEIR MEANINGS

Character	Meaning
:	optional terminator to a command name
,	data separator
(space)	data separator
&	continuation line (when used as the last character excluding spaces, on a line)
$	comment flag (everything following it on the line is a comment)
" "	encloses a name
' '	encloses a name
!	indicates a user defined parameter to be replaced as specified in the DEFINE command
?	preceeds a condition label at the end of a command (read IF)
( )	encloses a list of parameters for a command OPTION
^	denotes a vector by being placed between a letter (denoting vector type) and an integer ID (for example, R^4 or T^5 or Q^6)
#	preceeds the ID for gridpoints (#Gi), reference points (#Ri), members (#Mi) with "i" denoting an integer

4. CONCEPTS, SYNTAX, AND CONVENTIONS

CONCEPTS

Element-to-Gridpoint Connections in 3-D LATDYN

LATDYN is designed for modeling the transient dynamics of flexible articulating structures and mechanisms involving joints about which members rotate through large angles. For these applications it has been convenient to extend conventional finite element modeling concepts in two small but significant ways.

One extension involves the concept of jointpoints. The name jointpoint is used to refer to a joint connection which may be a hinge or revolute (single hinge line), a universal (two hinge lines), or a ball (three hinge lines). Whereas a gridpoint introduces six degrees of freedom to the problem, a jointpoint only introduces one degree of freedom for each hinge line. Since a jointpoint by itself may never have as many as six degrees of freedom associated with it, it must always be attached to a gridpoint.

Thus from the users point of view in LATDYN, a finite element may be attached rigidly to a grid point, as in conventional structural finite element codes, or a finite element may be attached to a gridpoint through a jointpoint to which it is connected.

For example, to set up a model of two beams connected together at their ends by a hinge, in LATDYN, one member may be connected to a gridpoint, and the other member may then be connected to the same gridpoint through a hingepoint (that is a jointpoint with one hinge line). Together the gridpoint and the hingepoint form a hinge with the gridpoint on one side and the hingepoint on the other.

These differences for a user, between setting up a conventional finite element model, and setting up a 3-D LATDYN model, can cause some conceptual difficulties depending on how grid points are viewed. In LATDYN it is helpful to view gridpoints and jointpoints as real physical bodies, rather than as ficticious entities. These "point bodies" are always spatially miniscule, but they may have inertial properties associated with them.

When a gridpoint in LATDYN has a jointpoint attached to it, such as a hingepoint for example, it is clearly an approximation to physical reality, because there is no possible spatial detail present to show how the hinge axis and pin supports are offset relative to a mass which may be lumped at the gridpoint. The implicit assumption is that the hinge axis passes through the center of the lumped mass, and that any beam members connected to the gridpoint through the hinge do so at this same infinitesimal point.

Another extension solves this problem. It involves the ability to define a rigid body as comprising a number of gridpoints without using constraints and employing only the degrees of freedom for a single gridpoint. In the formulation, these dependent gridpoints are refered to as "rigid body offset points", but from the users point of view they are identical to any other gridpoints where finite elements may be connected.

These two concepts make it possible to define a rigid body with several grid points spatially offset from each other. In this way the system of adding jointpoints to a gridpoint gains considerably in generality because it is possible to include any spatial detail necessary for accurate modeling of a structure, as different joints may be defined at different offset points. Examples of how this may be done are shown in the Demonstration Problem Manual.

Coordinate Systems

In addition to the global coordinate system LATDYN allows a user to define additional coordinate systems through the AXES command. These coordinate axes may remain fixed relative to the global system for the entire analysis or may move with any specified part of the structure which is being modeled.

Fixed coordinate axes are primarily useful during the setup phase of model development. During a transient dynamic analysis however, which involves large angular rotations, it is often necessary to use moving frames of reference. Moving frames of reference can also be created, as indicated above, by attaching a previously defined set of axes to a gridpoint. The user defined coordinate system then moves with the gridpoint to which it is attached throughout the entire transient analysis.

User Defined Conditions, Variables, Vectors, and Operations

During a transient analysis it is not unusual for a user to wish to change some of the initial specifications of the model. LATDYN enables a user to accomplish this by allowing commands to be qualified as being subject to conditions. When a particular condition is "true" then the commands which are subject to it are active; when the condition is "false" then the commands are inactive.

For example, a constraint which involves the lockup of a deployable truss joint may be initially inactive when the truss is packed. When the truss deploys, then the joint is required to lock and this is accomplished by activating the constraint. This process is programmed by making the constraint which controls the locking of the joint, subject to a condition which includes the angle at which the joint locking mechanism will activate. Initially this condition will be "false" and at some time during the analysis, when the angle condition is satisfied, will become "true" and the constraint will be imposed.

In general the conditions which commands may be subject to, are defined by the user. These conditions may be as simple as a test on the relative angle of two beam members at a hinge, as described above, or may involve a complicated sensing and averaging of multiple filtered accelerations with time delays.

Another way in which a user may write commands which change during an analysis, is by using parameters which are user defined variables (Q-variables) or user defined vectors (Q-vectors). (Note, however, that only a few commands allow this to be done.)

For example, a user may wish to apply a control force to a particular location on a space structure. This control force may be proportional to the weighted average of filtered accelerations from several parts of the structure. In LATDYN it is possible to write an expression which defines a Q-variable to be the magnitude of such a control force.

SYNTAX

Every language needs syntactical rules and the LATDYN Command Language is no exception. In some respects LATDYN contains many more capabilities than conventional structural finite element codes and the command language is more complicated.than in some other codes We have however, tried to make the syntax self-consistent, easily memorized, and containing no unnecessary rules.

With many software packages, especially those developed for personal computers, there is no command language. Instead a user accomplishes all input to the program through a menu structure. Even some modern engineering analysis codes have adopted a menu structure for the user interface. We have not adopted this approach because although it does offer greatly increased simplicity when learning a code, it also has the drawback that it presents significant restrictions on the range of operations which can be accomplished. Menu structures are also renown for enforcing unnecessary tedium on the advanced user.

A command language structure also has other advantages. A user may lay out his model complete with comments and retain it in a self contained form on paper and/or in a computer file which may be easily retained, viewed, copied, edited, updated, or modified in any way using a word processor or text editor.

In contrast to this, files retained from a program operating under a menu interface do not necessarily reflect the detailed sequence or character of the input. They do not usually contain any user comments, and frequently may only be viewed or printed using the program itself.

General Format

A command is a directive to LATDYN which begins with a command name and which may be followed by one or more parameters. For example, the directive

```
TIMESPAN 200 800
```

begins with the TIMESPAN command and is followed by the parameters 200 and 800. Each of the command names have "aliases", abbreviated and/or alternate names. For example, the directives

```
TIMESPAN 200 800
TIMSPAN  200 800
TSPAN    200 800
```

are all equivalent since TIMSPAN and TSPAN are aliases of the TIMESPAN command.

The parameters are separated from the command, and from each other, by a blank or a comma, or a combination of the two. For clarity, the command may be suffixed with a colon. For example, the directives

```
TIMESPAN,200,800
TIMESPAN 200 800
TIMESPAN  200, 800
TIMESPAN:     200   ,   800
TIMESPAN      200       800
```

are all equivalent.

Not all parameters are numeric. The directives

```
COMPONENT Base
LINSPRING  Sp1  #G1 #G2  2.63  0
```

create a new component named "Base" and a lineal spring named "Sp1" respectively. In general, when named entities such as these are being created they do not have to be enclosed in quotes. However, when named entities are referenced or if there are spaces in the name itself, then the name must be enclosed in single or double quotes. For example, the directives

```
COMPONENT 'Section 1'
COMPONENT "Section 1"
```

are equivalent. Notice, however, that the directive

```
COMPONENT Section 1
```

creates a new component named "Section" and "1" is interpreted as a separate parameter.

Some commands have "optional" parameters, which may be omitted if the user so desires. For example, the directive

```
APPFORCE: Forc3, MDV(I^GLO), TIME(0.0, 10.0), #G1, #G2
```

creates a force "Forc3" in the direction of the x-axis in the global system (I^GLO), that is active during the time from zero to ten seconds and is applied to gridpoints 1 and 2. The TIME parameter is optional in this command and may be omitted if the user so desires. For example, if the force is intended to be on for the whole duration of the analysis then the command is simply

```
APPFORCE: Forc3, MDV(I^GLO) ,, #G1, #G2
```

where the two commas together indicate an omitted parameter. Note that the following directives are equivalent:

```
APPFORCE  Forc3  MDV(I^GLO) ,, #G1  #G2
APPFORCE:Forc3,MDV(I^GLO),,#G1,#G2
APPFORCE:Forc3 MDV(I^GLO),, #G1,#G2
APPFORCE:  Forc3    MDV(I^GLO)  ,,    #G1    #G2
```

but that the directive

```
APPFORCE:  Forc3    MDV(I^GLO)        #G1    #G2
```

is illegal because the omitted parameter is not indicated by two adjacent commas. When the omitted parameter is the **last** parameter, then the two commas do not have to be used.

Continuation Lines

Some directives may be too long to fit on a single 80-character line, and will have to be continued onto the next line. The continuation character is the ampersand, "&". For example, the directive

```
AXES  'Region 2'  ORIGIN(1,1,1)  ROTATE(REL,Z,-45,X,120,Y,-15)
```

could also take the form

```
AXES  'Region 2'  ORIGIN(1,1,1)                    &
          ROTATE(REL,Z,-45,X,120,Y,-15)
```

Up to 100 lines (1 beginning line and 99 continuations) are allowed for in a single directive. The "&" character does not have to be located in the last column of the line (that is, it may be followed by spaces).

Comments

Large collections of directives usually benefit from a generous sprinkling of comments. The comment character is the dollar sign, "$", and everything to the right of it is ignored by the preprocessor. There are two types of comments: stand-alone and in-line. Stand-alone comments have a dollar sign at the beginning of the line. For example, the directive

```
$ Control points for one 5-meter Space Station bay.
```

is a stand-alone comment. In-line comments are appended to actual directives. For example, the directive

```
GRIDPT #G12 100 0 200   $ create gridpoint 12
```

contains an in-line comment. When in-line comments and continuation characters are mixed, the continuation character preceeds the in-line comment character. For example, the directive

```
AXES 'Region 2' ORIGIN(1, 0.6, 3.52) & $ define the region 2 axis system
     ROTATE(REL,Z,-45,X,120,Y,-15)
```

demonstrates how the two may be mixed.

Ditto Commands

A "ditto" feature is available when entering many occurences of the same command. For example, the directives

```
GRIDPT #G21 100 400 200
GRIDPT #G22 200 400 200
GRIDPT #G23 300 400 200
GRIDPT #G24 100 500 200
GRIDPT #G25 200 500 200
GRIDPT #G26 300 500 200
GRIDPT #G27 100 600 200
GRIDPT #G28 200 600 200
GRIDPT #G29 300 600 200
```

may also be written as

```
GRIDPT #G21 100 400 200
   "   #G22 200 400 200
   "   #G23 300 400 200
   "   #G24 100 500 200
   "   #G25 200 500 200
   "   #G26 300 500 200
   "   #G27 100 600 200
   "   #G28 200 600 200
   "   #G29 300 600 200
```

CONVENTIONS

Order of Definitions

An important convention embodied in the design of the LATDYN Preprocessor is that **an entity must be defined before it is used.** This convention applies to all gridpoints, reference points, jointpoints, coordinate systems, reference vectors, table variables, table vectors, springs, dampers, members, and all other named entities. There are three major exceptions to this rule, Q-variables, Q-vectors, and Condition Labels.

The Preprocessor checks Q-variable and Condition Label definitions only after it has read all commands. If one has been used but not defined then it will issue an error. Q-vectors are not checked for definition. If a Q-vector is used but not defined, then a null vector will be used.

Conditional Commands

Many of the commands may be made conditional. That is, if at the end of the data for a command the character "?" appears followed by a condition label, then the command is conditional.

This means that the command is only active provided that the condition is TRUE. If at any time the condition becomes FALSE, then the command becomes inactive. For example, the directive

```
C21:   X(D,#G3)  .GT.  1.0
```

defines condition 21 (the x-displacement of gridpoint 3) > 1.0. The directive

```
LINSPRING  Sp1   #G1   #G2    2.63    0      ? C21
```

establishes a conditional lineal spring which is active only when condition 21 is true. The ? C21 is called a condition label.

Condition labels have a variety of appearances. For example, the directives

```
LINSPRING  Sp1  #G1 #G2  2.63  0  ?21
LINSPRING  Sp1  #G1 #G2  2.63  0  ? 21
LINSPRING  Sp1  #G1 #G2  2.63  0  ?C21
LINSPRING  Sp1  #G1 #G2  2.63  0  ?    C21
LINSPRING  Sp1  #G1 #G2  2.63  0  ?CL21
LINSPRING  Sp1  #G1 #G2  2.63  0  ? CL21
```

are all equivalent. If a condition label is allowed, it is always the directive's last parameter.

When a condition label is defined it takes the form

```
CLj:  (logical condition)          or,
Cj:   (logical condition)
```

A condition label may also be used as a *logical variable* inside the logical condition itself. For example,

```
CL6:  (Q2 .GT. 1) .AND. CL2
```

That is, condition 6 will be true provided that two conditions are satisfied, namely Q2 must be greater than 1, and, condition 2 must be true.

When a condition label is used in this way inside a condition it is called a "condition variable" and must be referenced in the form CLj. The alternate form Cj is not allowed for condition variables.

Multiple Versions of Singular Commands

Some commands which use condition labels are "singular", meaning that the command may be specified many times, but all occurences after the first must have a condition label. For example, the directives

```
TIMESTEP: .01
TIMESTEP: .005  ? C3
TIMESTEP: .001  ? C7
```

establish a baseline (default) time step increment at .01 seconds. When condition 3 is true, the timestep increment becomes .005 seconds. If condition 7 becomes true, then the timestep increment becomes .001 seconds. If both conditions 3 and 7 are true, then the timestep is still .001 seconds since it is set by the most recent command in the sequence whose condition is true. Thus, if the directives were organized as

```
TIMESTEP: .01
TIMESTEP: .001  ? C7
TIMESTEP: .005  ? C3
```

and both conditions 3 and 7 are true, then the timestep increment becomes .005 since it is now the most recent command in the sequence.

The reason for requiring that certain commands be singular is simple. For example, in the case of the TIMESTEP command it is not possible for the program to perform time integration using more than one timestep simultaneously. There is often a need however, to change from one timestep to another during the integration.

Options in Command Parameters

Some LATDYN commands allow different options for portions of the sequenced data. That is, they allow data to be specified in different ways. For example, the orientation of a beam member about its axis may be specified by a reference point (POINT option):

```
FMEMBER: BeamMember#, SINGLE(startgrid#, endgrid#),          &
         POINT(ref.point#), beampropname
```

or through specifying a direction by a vector:

```
FMEMBER: BeamMember#, SINGLE(startgrid#, endgrid#),          &
         VECTOR(ref.vect.ID), beampropname
```

Note that in each of the above cases, the option is selected by use of a key word (POINT or VECTOR), which is followed by a number of parameters enclosed between parentheses.

Another illustration of the general form for options in parameter specification is also contained in the above examples. The keyword "SINGLE" denotes that a single element beam member is being defined, that is, the entire beam is to be represented by a single finite element. To create a multi-element beam member, the option keyword "MULTI" is used, for example

```
FMEMBER: BeamMember#, MULTI(String3),        &
         VECTOR(ref.vect.ID), beampropname
```

where "String3" is the name of a string of gridpoints as defined in a GRIDSTRING command. This FMEMBER command creates a string of finite elements along the string of gridpoints, to form the multi-element member.

Gridpoint (and Jointpoint) Notation

In most finite element codes gridpoints (also called "nodes") are referenced in commands and functions by an ID which consists simply of an integer.

In LATDYN, this integer ID must be preceeded by the characters "#G". If the gridpoint is defined as being part of a component, then it must be referenced by its component ID and its gridpoint ID, in the form "#CiGj". For example, gridpoint 3 may be referenced in a function X as

```
X(D,#G3)
```

or if gridpoint 3 is part of component 2, then it is referenced as

```
X(D,#C2G3)
```

or if it is part of a named component "Base", then it is referenced as

```
X(D,#C"Base"G3)
```

Note that, there should be no comma or other delimeter between the "C" part of the ID and the "G" part. Also, Gridpoint ID numbers may be repeated on different components. That is, #C1G1 is not the same as #C2G1, and both are different from #G1.

To reference a jointpoint which is attached to a gridpoint, the format is "#GjJk", where J= "H" for a hinged jointpoint, J= "U" for a universal jointpoint, J= "B" for a ball jointpoint. If the gridpoint is part of a component then the ID takes the form "#CiGjJk".

In general, a jointpoint reference may be given in most places that a gridpoint reference is required, and for most purposes a user may think of jointpoints as gridpoints. Exceptions to this general rule are noted in the manual.

Reference Point Notation

Reference points are defined in a REFPT command and may be referenced in several other commands. The form that the reference takes is the letter "R" preceeded by the "#" sign and an integer: #Rj. For example (#R3, #R26).

If a reference point is defined as belonging to a numbered component then it is referenced in the form #CiRj, or if it is defined as belonging to a named component then it is referenced as #C"name"Gj.

Beam Member and Element Notation

Beam members consist of one or more beam finite elements The integer ID for a beam member must be preceeded by the characters "#M" and the integer ID for a beam finite element must be preceeded by the letter "E". A beam element may only be referenced by first specifying its member ID.

If the member is also part of a component, then the component ID must also be given. For example,

#M1 - to reference member 1
#M3E2 - to reference element 2 of member 3
#C2M5E4 - to reference element 4 of member 5 in component 2
#C"Flange"M5E4 - to reference element 4 of member 5 in component "Flange"

Note: 1) There should be no comma or other delimeter between the "M" part and the "E" part of the ID, or between the "C" part and the "M" part.

2) Element ID numbers usually start at "one" on each member and run continuously from one to N (N=number of elements on a member).

3) Member ID numbers do not have to start with "one" on each component. There may also be gaps in the sequence of member numbers of a component and member numbers may be repeated on other components (#C1M1 is not the same as #C2M1).

Notation for Q-Variables

A Q-variable is the generic name given to user defined variables. In LATDYN, all user defined variables are required to begin with the letter "Q", hence the name Q-variables. A Q-variable is defined in one of the several types of SET commands, and the format of its definition appears very much like a FORTRAN assignment statement. The right hand side of the assignment may include any standard FORTRAN arithmetic operation, standard FORTRAN functions, and also special LATDYN User Functions.

Q-variables may be used as parameters in a number of LATDYN commands, such as applied loads and constraints. They may also appear in other Q-variable assignments. Q-variables are one of the basic tools which enable a user to program a control system.

The format for global Q-variables is Qk where k is an integer betwen 1 an 99. The format for local Q-variables is Qname where name is any alphanumeric string not exceeding six characters. For example, Q3, Q56, Q88 are global Q-variables whereas, QINC, QSPIN, QTEMP are local Q-variables.

Notation for Table Variables

Table variables refer to user defined tables which are interpolated at each timestep to give the current value of the table variable. The format of a table variable is simply defined by the letter "T" followed by an integer in the range 1 to 99. For example, T7, T12.

Vector Notation

There are three types of vectors in LATDYN:

R^k - a reference vector (k is any integer ID).
T^k - a table vector (k is an integer betweeen 1 and 99).
Q^k - a general purpose Q-vector (k is an integer betweeen 1 and 99).

In the notation shown above, all three types of vectors contain the carat symbol (^) between the letter designator (R, T, or Q) and the ID number k.

In addition to the three types of general vectors, there are also unit vector triads which are associated with each coordinate system:

I^GLOBAL, J^GLOBAL, K^GLOBAL - for the global coordinate system (GLO and GLOB are permitted abbreviations), and
I^axesname, J^axesname, K^axesname - for a user defined coordinate system

The global unit vector triad does not require definition. Other user defined coordinate system unit vector triads are automatically defined when the coordinate system is defined using the AXES command.

Functions and Operators

LATDYN contains many functions and operators. Functions may be employed in user defined conditions and Q-variables. Operators perform tasks such as filtering,setting up, or multiplying vectors. These parts of the LATDYN Command Language are restructured by the Preprocessor into FORTRAN code which is then linked with the LATDYN object code. Because of this, there is great user freedom in how to use these capabilities.

However, the penalty paid for this latitude is that it is difficult to devise procedures to fully check the syntax and parameters in these commands. One checking facility which has been included in LATDYN is a user option which is invoked by placing the "@" character in front of a function or operator.

When the LATDYN Preprocessor detects a function or operator preceeded by a @, then it checks the number, type, and where possible the range, of the arguments for that function or operator.

String Replacements

The DEFINE command of 3-D LATDYN enables the replacement of any alpha-numeric strings which begins with the special character "!" and which ends with a comma or a space, wherever it occurs throughout the command sequence, to be replaced with the appropriate string specified in the DEFINE command.

5. PREPROCESSOR COMMANDS

Most of the commands listed in this manual may be considered to be LATDYN System commands. That is, they are read and translated by the Preprocessor for the purpose of setting up data for input to the LATDYN program. Other commands fall into different categories. For example, LATDYN has a category of commands whose sole purpose is to direct the operation of the Preprocessor. Commands of this type are called Preprocessor commands.

Reading a User's Input File

The command to instruct the Preprocessor to read a user's input file takes the form:

READ, READFILE, READF

READ: input file name

input file name the name of the file containing LATDYN command directives

EXAMPLES

READ: ROBOT.DAT

Terminating Preprocessor Operation

The command to instruct the Preprocessor to terminate takes the form:

END, EXIT, HALT, QUIT, STOP

END

EXAMPLES

END

Directing Preprocessor Echo Messages

When the Preprocessor reads a user directive it echos that directive either to the user's interactive display screen or to a file. The form of the command to specify this action is:

ECHO, ECH

ECHO: echo message destination

echo message destination the name of the file to which echo messages are to be sent , or the word "ME" to direct echo messages to the screen of the users inteactive terminal. The default is the file ECHO.

EXAMPLES

ECHO: ME
ECHO: SCRATCH

Directing Preprocessor Fatal Error Messages

When the Preprocessor encounters an error in a user directive it categorizes that error as a "fatal" error or as a "warning" error. Fatal errors cause a mandatory termination of the job, and no output files for the LATDYN program are produced. A user has the option of directing fatal error messages to an interactive display screen or to a file. The form of the command to specify this action is:

FATALS, FATAL, FAT

FATALS: fatal message destination

fatal message destination the name of the file to which fatal messages are to be sent , or the word "ME" to direct fatal messages to the screen of the users inteactive terminal. The default is ME.

EXAMPLES

FATALS: ME
FATALS: SCRATCH

Directing Preprocessor Warning Error Messages

When the Preprocessor encounters an error in a user directive it categorizes that error as a "fatal" error or as a "warning" error. Warning errors do not cause a mandatory termination of the job. A user has the option of directing warning error messages to an interactive display screen or to a file. The form of the command to specify this action is:

WARNINGS, WARNING, WARN

WARNINGS: warning message destination

warning message destination the name of the file to which warning messages are to be sent , or the word "ME" to direct warning messages to the screen of the users inteactive terminal. The default is WARNINGS

EXAMPLES

```
WARNINGS: ME
WARNINGS: SCRATCH
```

6. TOOLS FOR MODEL SETUP

This section includes information on commands to assist the user in setting up a LATDYN model. These include coordinate systems which may be defined in a variety of ways, special Euler angle systems, reference points and reference vectors and components, specified default items, and parameter substitution.

Some features which appear simple can perform quite complicated operations and it may not always how things have been done. This is particularly true for model setup because, as shown in the tutorial section of this manual the LATDYN system is divided into three programs, and it is usually semi-transparent to the user which operations are performed by the Preprocessor program and which are performed by the LATDYN program. To help to avoid confusion, this section explains which operations that are associated with each command are performed by each program.

COORDINATE AXES AND EULER ANGLES

It is often useful to define additional sets of coordinate axes. This may be accomplished with the AXES command. Moving coordinate systems may be created by using the ATTACH command to attach a previously defined coordinate system, to a part of the structure being modeled.

The AXES command functions primarily within the Preprocessor which computes the appropriate transformation matrices for transforming subsequent entities which are defined in this coordinate system.and performs the appropriate transformations. The ATTACH command on the other hand functions primarily within the LATDYN program which receives axis definition information from the Preprocessor and uses it to set up moving coordinate systems, where required.

Creating a New Coordinate System

There is a unit vector triad which is associated with each coordinate system: I^GLO, J^GLO, K^GLO - for the global coordinate system, I^axesname, J^axesname, K^axesname - for a user defined coordinate system. The global unit vector triad does not require definition. User defined coordinate system unit vector triads are automatically defined when the coordinate system is defined using the AXES command. The form of the AXES command is:

AXES, AXIS, AX

AXES: new axesname, OPTION for origin spec., OPTION for orientation spec., OPTION for coordinate syst. with respect to which data is specified

new axesname — any alphanumeric character string identifying the new coordinate system. ("GLOBAL, GLOB, GLO" are reserved names and cannot be used here.)

OPTION for origin specification - ORIGIN or omitted (default = global origin)

ORIGIN(x, y, z)

x , y , z — origin of the coordinate system in Global coordinates or in coordinates relative to a specified coordinate system (see WRT OPTION).

OPTION for orientation specfication - ANGLE, AXPTS, ROTATE, or USANGLE

ANGLE(Yaw, Pitch, Roll) - Specification of the orientation of the coordinate system will be by the LATDYN default Euler angle system (yaw, pitch, roll). A definition of these is given in the next section entitled - The Default Euler Angle System in LATDYN.

yaw, Pitch, Roll - Euler angles giving the orientation of the new set of coordinate axes.

Of the many possible conventions for definition of these rotation angles, the one chosen for use as a default in LATDYN is based on a right-handed Cartesian coordinate system with the angles defined as follows:

Yaw - The first rotation is about the y-axis, *Pitch* - The second rotation is about the new position of the z-axis after the "yaw" rotation has been performed, *Roll* - The third rotation is about the final resulting position of the x-axis, after "yaw" and "pitch" rotations have been performed.

AXPTS(axlab1, x1,y1,z1, axlab2,x2,y2,z2) - Orientation of the coordinate system will be specified by two points.

axlab1 — label (X, Y, or Z) of the first axis to be defined.

x1,y1,z1 — coordinates of a point . The first axis is defined by the line from the origin to this point.

axlab2 — label (X, Y, or Z) of the second axis to be defined. (Note: the second axis must not be parallel to the first axis.)

x2,y2,z2 — coordinates of a point. The second axis is defined by the **plane** containing the first axis and the new point. Thus the second axis does not necessarily pass through the point (x2,y2,z2), if the line from this point to the origin is not normal to the first axis.

Note that only two coordinate axes need to be defined (X&Y, X&Z, or Y&Z) since the third orthogonal axis will be computed according to the right-hand rule.

Note also that if the ORIGIN option is specified first in this command, then that origin will be used in conjunction with the points given in this commmand for defining the new coordinate axes. For example if the new origin is at (1,1,1) and a point is given for the new "X" axis at (2,1,1) then this new X-axis will have direction vector (1,0,0) parallel to the global X-axis, since it runs from (1,1,1) to (2,1,1) in the global system.

On the other hand, if the AXPTS option is specified before the ORIGIN option, then the axis vectors for the new system will be initially defined to run from the origin of the global system. When the ORIGIN option is encountered, these vectors will be translated to this new origin. For example, if a point is given for the new X-axis at (2,1,1), then the X-axis direction vector is (2,1,1).

ROTATE(type, ax1, rot1, ax2, rot2, ax3, rot3) - Orientation of the coordinate system will be by specification of (up to) three rotations in a specified *order*.

type REL or ABS. Defines whether the rotations given in this command are to be performed about the new (relative) positions of the coordinate axes, or about the original (absolute) positions of the axes.

ax1, ax2, ax3 axis labels (X, Y, or Z) of the axes about which rotations are to be performed.

rot1, rot2, rot3 - rotation angles about the specified axes. Positive rotations are clockwise as viewed from the origin of the coordinate system along the axis of rotation.

USANGLE(rot1, rot2, rot3) - Specification of the orientation of the coordinate system will be by the users default Euler angle system as defined in the USEULER command.

rot1, rot2, rot3 - Euler angles giving the orientation of the new set of coordinate axes. The convention used for these angles is as given in the USEULER command.

Note: The USANGLE option allows from one to twenty rotations to be specified.

OPTION for Coord. System relative to which data is specified - WRT or omitted

WRT(axesname) All data given in this command is specified with respect to the named coordinate system. (If this option is omitted then the GLOBAL system is assumed.)

axesname Name of the coordinate system with respect to which the new system is defined. This system must already have been defined.

EXAMPLES

```
AXES: Sys1 ORIGIN(1 1 1) ROTATE(ABS X 90.0)
AXES: Tip_System  ORIGIN(16.2 9.3 0.0)                          &
            AXPTS(X 20 9.3 0.0   Y 16.2  12  0.0)   WRT(Sys1)
AXES: Sys2  ANGLE(90.0 20.0 0.0)
  "    Sys3  ORIGIN(30  20  20)
```

The Default Euler Angle System in LATDYN

Of the many possible conventions for definition of these rotation angles, the one chosen for use as a default in LATDYN is based on a right-handed Cartesian coordinate system with the angles defined as follows:

Yaw - The first rotation is about the y-axis,

Pitch - The second rotation is about the new position of the z-axis after the "yaw" rotation has been performed,

Roll - The third rotation is about the final resulting position of the x-axis, after "yaw" and "pitch" rotations have been performed.

This definition of rotations may be remembered by relating it to the maneuvers of an aircraft with the x-axis along the fuselage centerline, the y-axis vertical and the z-axis along the right wing. As the aircraft turns from the taxiway to point down the runway in preparation for takeoff it performs a "yaw" rotation. As the aircraft takes off, its nose rises performing a "pitch" rotation. Subsequently in preparation for executing a turn, the aircraft's right wing drops and the left one rises as the aircraft "rolls" about the centerline of the fuselage.

Creating a User Defined Euler Angle System

Because there are many definitions of "Euler Angles" there are also many strong opinions on which definition represents the best, or the easiest, or the most intuitive way, of defining them. LATDYN allows a user to create his or her own favorite Euler Angle Definition by use of the USEULER command. The effect of this command is to set up this system for a user to reference in all future AXES commands by the OPTION keyword, USANGLE.

USEULER, EULER, USEUL

USEULER: type, ax1, ax2, ax3	
type	REL or ABS. Defines whether the rotations given in this command are to be performed about the new (relative) positions of the coordinate axes, or about the original (absolute) positions of the axes.
ax1, ax2, ax3	axis labels (X, Y, or Z) of the successive axes about which rotations are to be performed. From one to twenty rotations can be specified.

EXAMPLES

```
USEULER: REL, Y, Z, Y    $ Goldstein's favorite system
USEULER: REL, Y, Z, X    $ LATDYN's default system
```

REFERENCE POINTS

Reference points are created by a user to provide a spatial reference either during model setup (or, for the measurement of relative motion during a transient analysis - see ATTACH command). Defining a reference point adds no degrees of freedom to the system.

Define a Reference Point

REFPOINT, REFPT, RPOINT, RPT, REF

REFPT: ref#, x, y, z, axesname	
ref #	reference point identifier (#Rj or #CiRj)
x, y, z	cartesian coordinates
axesname	refers to a coordinate system defined by the AXES command. The cartesian coordinates (x,y,z) are specified in this coordinate system. If left blank, the default system will be assumed. The default system is the "GLOBAL" system unless another default system is specified in the DEFAULT command.

EXAMPLES:

```
REFPT:  #R1   1.1  2.2  3.3    $ define #R1 in Global System
   "    #R2   1.1  2.2  0
   "    #C3R1  0.0  0.0  0.0  C3Sys   $ #C3R1 is at origin of C3Sys
```

REFERENCE VECTORS

There are two types of vectors in LATDYN which can be used for reference. The following type is defined explicitly:

R^k - a reference vector defined by a REFVECT command

where k is an integer ID number. In the notation shown above, a reference vector contains the carat symbol (^) between the letter designator "R" and the ID number "k".

There is also a type of reference vector that is defined implicitly. Whenever a new coordinate system is defined using the AXES command , then unit vector triads which are associated with each coordinate axis of the system are defined automatically. They may then be refered to as

I^axesname, J^axesname, K^axesname - for the x,y,z axes of a user defined coordinate system

The global unit vector triad does not require definition. It is always available and may be refered to as

I^GLOBAL, J^GLOBAL, K^GLOBAL - for the x,y,z axes of the global coordinate system. Permitted abbreviations are GLO and GLOB.

Define a Reference Vector

Reference vectors are defined using the REFVECT command:

REFVECT, REFVECTOR, RVECTOR, RVECT, REFV

REFVECT: refvect ID, OPTION for vector specification

Ref. Vector I.D. Reference vector identifier R^k where k is an integer.

OPTION for vector spec. POINTS, RCOORD, or SCOORD (Three options are available for defining the magnitude and orientation of a reference vector.)

POINTS(grid a# or ref a# or coords, grid b# or ref b# or coords) - base and tip of vector will be defined by two points or their coordinates.

grid a# or ref a# or coords - grid or reference point identifier at base of vector (#Gj, R#j, or #CiGj), or cartesian coordinates of a point in a specified coordinate system (x,y,z,axesname).

grid b# or ref b# or coords - grid or reference point identifier at tip of vector (#Gj, R#j, or #CiGj), or cartesian coordinates of a point in a specified coordinate system (x,y,z,axesname).

RCOORD(x,y,z, axesname for definition) - Cartesian (Rectangular) Coordinates Option

x,y,z Cartesian (rectangular) coordinates of tip of vector. The base of the vector is at the origin of the specified coordinate system.

axesname for def. - refers to a coordinate system defined in the AXES command. Base of vector is at origin. The default is the global system.

SCOORD (Mag, yaw,pitch, axesname for def.) - Spherical Coordinates Option

Mag Vector magnitude

yaw, pitch yaw and pitch angles

axesname for def - Refers to a coordinate system defined in the AXES command. *yaw & pitch* are defined in this system. The default is the global system.

NOTE: A reference vector may be fixed in a moving frame of reference. That is, it may maintain a constant orientation with respect to any physical part of the model - see ATTACH command.

EXAMPLES

```
REFVECT: R^1 POINTS(#G3, 0,1,1,GLO) $ Vect. runs from #G3 to global point
$                                     (0,1,1).
REFVECT: R^6 POINTS(#G3, #G4) $ Vect. runs from #G3 to #G4.
REFVECT: R^6 POINTS(#R8, #G4) $ Vect. runs from #R8 to #G4
REFVECT: RCOORD(4,5,6, Sys2) $ Vect runs from origin of Sys2,
```

```
$                                        to Sys2 point (4,5,6)
REFVECT: SCOORD(10, 45,90, GLO)  $ Vect of magnitude 10, runs from global
$                                        origin at yaw and pitch angles of 45
$                                        and 90 degrees respectively.
```

USE OF COMPONENTS

Grid Points and members may be defined as parts of a component simply by assigning them a component I.D. when they are created. This facility is useful in itself simply as an aid in keeping track of grid point, reference point, and member number since items that are part of a component have an additional unique identifier. For example, #C1G1, #C2G1, and #G1 are all different unique gridpoints.

Before this capability may be used however, it is first necessary to declare a component by the COMPONENT command. For declaring the existence of a new component, this command takes the form,

COMPONENT, COMP

COMPONENT: new cmpnt. name	
new cmpnt. name	The name of the new component to be declared. This name may take one of two forms: a) Integer I.D. - #Ci (The component is to be referenced by an integer I.D. which takes the form #Ci where "i" is the integer identification, e.g. #C1, #C6, #C23). b) Name I.D. - #C"name" (The component is to be referenced by an alphanumeric I.D. contained in quotes, e.g. #C"Base", #C"ARM"). Note: when component has no spaces in its name then the quotes(" ") do not have to be used in this command.

EXAMPLES:

```
COMPONENT:      #C1
     "          #C"Longeron1"
     "          #C23
COMP:      C10
```

PARAMETER SUBSTITUTION AND SETTING DEFAULTS

Most of the commands listed in this manual may be considered to be LATDYN System commands. That is, they are read and translated by the Preprocessor for the purpose of set-

ting up data for input to the LATDYN program. Some other commands fall into a slightly different category.

For example, the Preprocessor contains several advanced features which are especially useful for building large data cases, or for building multiple data cases which differ only by a few command directives or parameters. These features include the capability for parameter substitution and for resetting several system defaults from directives contained in the user input.

Parameter Substitution

The purpose of parameter substitution is to allow a user to easily change frequently occuring items in the input data. It may also be used to highlight particular parameters in the input data for the purposes of conducting a parameter trade study. When parameters for substitution appear in the body of the input file they must be preceeded by the "!" character. When the Preprocessor detects this special character, it recnogizes that a parameter to be substituted will follow and performs the substitution. The command to define a parameter has the form:

DEFINE, DEF

DEFINE: parameter name, value to be substituted for parameter

parameter name alphanumeric ID for parameter

value to be substituted for parameter - alphanumeric string to be substituted for all subsequent occurrences of the parameter

EXAMPLES

```
$          parameter name    value
DEFINE:    LocationX         7.15697E4

$          grid#     x             y      z
GRIDPT:    #G5       !LocationX    5.1    9.4
GRIDPT:    #G6       !LocationX    8.3    9.4
GRIDPT:    #G7       !LocationX    14.96  9.4
GRIDPT:    #G8       !LocationX    17.77  9.4
```

Set Default Component, Axes, Gridpoint, Jointpoint, Member or Element

A set of useful features associated with components, axes, gridpoints, jointpoints, reference points, members, and elements is the ability to define and redefine defaults. In subsequent commands, if the ID for these entities are omitted, then the Preprocessor will assume the default setting for that entity as specified in.the DEFAULT command This capability is especially useful for shorthand notation and in creating complex structures. The form of the command is,

DEFAULT, DEFAUL, DEFAU, DEFA

DEFAULT: item for which default is given, default ID

item for which default is given COMPONENT, COMP, or CO for specifying a component default,
AXES, AXIS, or AX for specifying a coordinate system default
GRID for specifying a gridpoint default,
HINGE or HIN for specifying a hingepoint default
BALL for specifying a ballpoint default
UNI for specifying a unipoint default
MEMBER or MEMB for specifying a member default
ELEMENT or ELEM for specifying an element default

default ID the ID to be used as a default for item specified.

If the item to be defaulted is COMPONENT then this ID may take one of two forms:
a) Integer I.D. - i (All generic component references are to be renamed by the integer "i". Thus, the following generic references #CG1, #CG4, #CG7, become #CiG1, #CiG4, #CiG7.)
b) Name I.D. - name (All generic component references are to be renamed by an alphanumeric I.D. contained in quotes, e.g. #C"Base", #C"ARM". Thus the following generic references #CG1 and #CG4, become #C"name"G1 and #C"name"G4).

If the item to be defaulted is AXES then this ID is an axesname. This allows a user to name a default coordinate system, which has already been created by the AXES command, for use with all following GRIDPOINT, GRIDSTR, GRIDSTRE OR REFPT commands. That is, if no coordinate system is named in these commands, then the system specified here will be used instead of the GLOBAL system.

If the item to be defaulted is GRID then this ID is an integer "j" giving.the ID of the default gridpoint #Gj. All generic gridpoint references are to be renamed by the integer "j". Thus, the following generic references #C1G, #G, #C2GH1, become #C1Gj, #Gj, #C2Gj.

If the item to be defaulted is HINGE then this ID is an integer "k" giving.the ID of the default hingepoint #Hk. All generic hingepoint references are to be renamed by the integer "k". Thus, the following generic references #C1G2H, #G1H, #C"arm"G5H, become #C1G2Hk, #G1Hk, #C"arm"G5Hk.

If the item to be defaulted is BALL then this ID is an integer "k" giving.the ID of the default ballpoint #Bk. All generic ballpoint references are to be renamed by the integer "k". Thus, the following generic references #C1G2B, #G1B, #C"arm"G5B, become #C1G2Bk, #G1Bk, #C"arm"G5Bk.

If the item to be defaulted is UNI then this ID is an integer "k" giving.the ID of the default unipoint #Uk. All generic unipoint references are to be renamed by the integer "k". Thus, the following generic references #C1G2U, #G1U, #C"arm"G5U, become #C1G2Uk, #G1Uk, #C"arm"G5Uk.

If the item to be defaulted is MEMBER then this ID is an integer "j" giving.the ID of the default member #Mj. All generic member references are to be renamed by the integer "j". Thus, the following generic references #C1M, #M become #C1Mj, #Mj.

If the item to be defaulted is ELEMENT then this ID is an integer "k" giving.the ID of the default element #MjEk. All generic member references

are to be renamed by the integer "k". Thus, the following generic references #C1MjE, #MjE become #C1MjEk, #MjEk.

EXAMPLES

One example of the utility of the DEFAULT command is achieved by combining it with the READ command to generate multiple copies of a component located in different physical locations. For example, the component command may be used to set up a default for component ID and coordinate system as follows:

```
DEFAULT:  COMPONENT   1
    "     AXES        SYS1
READ: xfile
DEFAULT:  COMPONENT   2
    "     AXES        SYS2
READ: xfile
DEFAULT:  COMPONENT   3
    "     AXES        SYS3
READ: xfile
```

where the file "xfile" may contain the following,

```
$       grid#  x    y    z
GRIDPT: #CG1,  0,   0,   0
$
GRIDPT: #CG2,  .1,  0,   0
$
GRIDPT: #CG3, 3.25,  6.8,  9.3
$
GRIDPT: #CG4, 5.93,  24.71, 8.56
$
GRIDPT: #CG5, 7.40,  45.77, 0.0
$
GRIDPT: #CG6, 44.1,  33.33, 15.5
$
GRIDPT: #CG7, 12.7,  15.9,  17.556
$
```

Note that relative references are also permitted. For example, the following sequence of commands defines gridpoints #G51 through #G57:

```
DEFAULT:      GRID     50
$
$       grid#  x    y    z
$
GRIDPT: #G+1,  0,   0,   0
$
GRIDPT: #G+2,  .1,  0,   0
$
GRIDPT: #G+3, 3.25,  6.8,  9.3
$
GRIDPT: #G+4, 5.93,  24.71, 8.56
$
GRIDPT: #G+5, 7.40,  45.77, 0.0
$
GRIDPT: #G+6, 44.1,  33.33, 15.5
$
```

```
GRIDPT: #G+7, 12.7,   15.9,   17.556
$
```

The following sequence of commands defines gridpoints #C2G8, #C2G9 and #C2G10:

```
DEFAULT:      GRID    9
$
$    grid#   x    y    z
GRIDPT: #C2G-1,  0,    0,   0
$
GRIDPT: #C2G,  .1,    0,   0
$
GRIDPT: #C2G+1, 3.25,   6.8,   9.3
$
```

Clear Default Component, Axes, Gridpoint, Jointpoint, Member or Element

The NODEFAULT command is the converse of the DEFAULT command. Its action is to erase defaults established in previous DEFAULT commands. The form of the command is,

NODEFAULT

NODEFAULT: item for which default is to be cleared

item for which default is to be cleared - COMPONENT, COMP, or CO for clearing a component default,
AXES, AXIS, or AX for clearing a coordinate system default
GRID for clearing a gridpoint default,
HINGE or HIN for clearing a hingepoint default
BALL for clearing a ballpoint default
UNI for clearing a unipoint default
MEMBER or MEMB for clearing a member default
ELEMENT or ELEM for clearing an element default

EXAMPLES

```
NODEFAULT:  COMP
NODEFAULT:  AXES
```

Show Default Component, Axes, Gridpoint, Jointpoint, Member or Element

The SHOWDEFAULT command is intended for interactive operation with the preprocessor. Its action is to show defaults established in previous DEFAULT commands. The form of the command is,

SHOWDEFAULT

EXAMPLES

SHOWDEFAULT

7. DEFINING A STRUCTURAL MODEL

PROPERTIES

Commands in this section set up properties which are then available for use by subsequent commands. For example, the MATPROP command may be used to set up material properties for aluminum which may then be used in a BEAMPROP command to define the cross-sectional mass and stiffness properties of a beam. These beam properties may then be used by subsequent FMEMBER and RMEMBER commands to create a structure consisting of beam members with these cross-sectional properties.

Define a Material Property

The MATPROP command enables the definition of an isotropic set of material properties for use in subsequent commands to create structural entities such as beam members. The form of the command is,

MATPROP, MATPRO, MAT

MATPROP; Matname, E, G, rho

Matname	any alphanumeric character string. The same material name cannot be used in two MATPROP commands.
E	Young's modulus
G	Shear Modulus
rho	mass density

EXAMPLES

```
MATPROP:  Alum     1.E7  .4E7   .101
MATPROP:  StSteel  2.8E7 1.2E7  .29
```

Define the Properties of a Beam Cross-Section

The properties of a beam cross-section are defined in a BEAMPROP command. These properties include a reference to a material property (which should be defined before this command is used), and calculated values of cross-sectional area and moments of area. The command may be made conditional to allow beamproperties to change during a transient analysis (see "Multiple Versions of Singular Commands" in Chapter 4). The form of the command is,

BEAMPROP, BPROP, BPRO

BEAMPROP; beampropname, matname, A, Iy, Iz, J ? Cj

beampropname	any alphanumeric character string. The same beampropname cannot be used in two BEAMPROP commmands
Matname	refers to a material name defined by a MATPROP command.
A	beam X-sectional area
Iy	second moment of area about y-axis of beam
Iz	second moment of area about z-axis of beam
J	polar moment of area (Iy + Iz)
Cj	optional condition label

EXAMPLES

```
BEAMPROP:  TrussA  Alum    5.6   350   350   700
BEAMPROP:  TrussB  StSteel 2.5    70   150   220
```

Define The Properties of a Lumped Mass

The MASSPROP command allows the properties of a lumped mass to be specified. A lumped mass does not actually become part of the structure, until it is created and "stuck" onto a gridpoint or jointpoint by an ADMASS command. Each ADMASS command refers to a set of lumped mass properties specified by a MASSPROP command. When a lumped mass is created by an ADMASS command, these properties are transformed to the required position and orientation. Thus, a lumped mass may be specified in a MASSPROP command using the most convenient coordinate system for the user since it may be oriented and positioned later

If several identical lumped masses are used in a structure, it is only necessary to define one set of mass properties in a MASSPROP command. Later ADMASS commmands may refer to this same set of properties, rotate and translate them in a variety of ways, and "stick" them onto multiple gridpoints or jointpoints.

A complex rigid mass which is composed of several simple geometric constituents may be defined by multiple MASSPROP commands, one for each of its simpler geometric constituents. These may later be reoriented, positioned, and attached to the same gridpoint or jointpoint by ADMASS commands to form the composite complex mass.

MASSPROP, MSPROP, MASPRO, MASS

MASSPROP: massname, mass, x, y, z, Ix, Iy, Iz, Ixy, Ixz, Iyz

massname	any alphanumeric character string. The same massname cannot be used in two MASSPROP commands.
mass	magnitude of the mass
x, y, z	coordinates of center of mass
Ix, Iy, Iz	moments of inertia about x,y,z axes

Ixy, Ixz, Iyz	products of inertia

EXAMPLES

MASSPROP:	Cylinder	3.267	5.6, 7.3, 9.9	572.2, 572.2, 1987.4	860, 920, 920
MASSPROP:	Sphere	3.267	5.6, 7.3, 0.0	450, 450, 450	0, 0, 0

GRIDPOINTS

Gridpoints are the basic conceptual entities on which a structural model in LATDYN is built. As in most finite element programs for structural modeling, a gridpoint is set up as a ficticious entity in space. The finite element model is then constructed by connecting finite elements (such as beam members) between gridpoints.

When a gridpoint is defined in LATDYN it is located at a specified point in 3-D space. It has zero mass, zero moments and products of inertia, and occupies zero volume. However, LATDYN anticipates that this gridpoint is going to be used as a structural "building block" by the user, so LATDYN automatically assigns six degrees of freedom to it when it is created. If the user defines a gridpoint and never attaches any finite elements or masses to it, then LATDYN's computations will blow-up, because a mass matrix with zeroes on the diagonal will have been generated. Physically this means that any force however small (even one due to numerical rounding error in the computer), will cause an infinite acceleration on a massless gridpoint.

Define a Single Grid Point

LATDYN has several commands to create gridpoints. The most basic of these commands creates a single gridpoint and has the form,

GRIDPT, GRID, GRPT, GR

GRIDPT: grid #, x, y, z, axesname

grid #	grid point identifier (#Gj or #CiGj, where i = component number, j = gridpoint number)
x, y, z	cartesian coordinates
axesname	the cartesian coordinate system in which the coordinates (x,y,z) are specified (must already have been created by an AXES command).
	If left blank, the default system will be assumed. The default system is the "GLOBAL" system unless another default system is specified in the DEFAULT command.

EXAMPLES

```
GRIDPT:  #G1   2.0 6.0 5.3
   "     #G3   2.0 6.0 10.3
```

```
"     #C1G1  0   0    0     Sys1   $ define #C1G1 at origin of Sys1
"     #G6   1.3E2  2.54E2  0   Ax2
```

Define a String of Equally Spaced Gridpoints Between Two Coordinate Locations

There are several possible reasons for wanting to create a straight line of equally spaced gridpoints, but the usual reason is for multi-element beam member. Once the lineal string of gridpoints is created, a multi-element beam member is constructed by attaching identical beam finite elements between each successive pair of gridpoints.

Sometimes it is useful to first run a model with one finite element in a beam member. After studying the results of this crude model it may be desireable to modify the model to have two, five or ten finite elements per member. LATDYN provides an easy way to do this by using "gridstrings".

A string of gridpoints may be defined and given a name by one of the following gridstring commands. This name may be referenced in the FMEMBER or RMEMBER commands. To change the number of finite elements in a beam member it is only necessary to change the number of gridpoints in the gridstring.

The form of the command to create a string of equally spaced gridpoints between two coordinate locations is,

GRIDSTR, GRIDSTRING, GRSTR

GRIDSTR; gridstring name, grid #, ng, x1, y1, z1, x2, y2, z2, axesname

gridstring name	alphanumeric name for gridpoint string to be created (two gridstrings with the same name may not exist).
grid #	gridpoint identifier (#Gj, #CiGj, where i = component number, j = gridpoint number). Defines the ID of the first new gridpoint to be created (at location x1,y1,z1).
	New gridpoints are sequentially numbered incrementing by "one" from this ID. If a component number is part of the ID then all gridpoints in the string will belong to the same component.
ng	number of grid points to be defined. This is equal to the total number of gridpoints in the string (must be > or = 2).
x1,y1,z1	cartesian coordinates of first grid point in string
x2,y2,z2	cartesian coordinates of last grid point in string
"axesname"	the cartesian coordinate system in which the coordinates (x,y,z) are specified (must already have been created by an AXES command).
	If left blank, the default system will be assumed. The default system is the "GLOBAL" system unless another default system is specified in the DEFAULT command.

EXAMPLES

```
GRIDSTR:  String1    3     0,0,0      1,1,1    $ define three gridpoints between
$                                                (0,0,0) and (1,1,1) in the global system.
$
GRIDSTR:  String2    10    2,3,5      28.5,40.1,16.3      System3    $ define
$              three gridpoints between (2,3,5) and (28.5,40.1,16.3) in the System3
```

Fill in a String of Equally Spaced Gridpoints Between Two Points Which Have Already Been Defined

This command operates in a very similar way to the GRIDSTR command. In that command it was assumed that neither of the endpoints of the gridstring already existed. In the present command it is assumed that both endpoints already exist, and a gridstring is defined between them.

One possible disadvantage with the earlier command is that the final gridpoint number changes when the number of points in the gridstring are changed. In the present command this is not the case.

The command to fill in a string of equally spaced gridpoints between two existing gridpoints or jointpoints has the form,

GRIDSTRF, GRIDSTRINGF, GRSTRF

GRIDSTRF: gridstringname, first new grid#, ng, start point#, end point#

gridstringname	alphanumeric name for gridpoint string. No two gridstrings may have the same name.
first new grid #	identifier for first new grid point to be created, (#Gj or #CiGj where i = component number, j = gridpoint number). Subsequent gridpoints to be created in the string will be identified by gridpoint numbers incrementing by "one". If a component number is given, then they will belong to the same component specified.
ng	total number of grid points in the string, including the first and last. That is, the number of new gridpoints = (ng - 2).
startpoint#	identifier for starting point (#Gj, #CiGj, #GjJk, or #CiGjJk where i = component number, j = gridpoint number, k= jointpoint number).
endpoint#	identifier for end point (#Gj, #CiGj, #GjJk, or #CiGjJk where i = component number, j = gridpoint number, k= jointpoint number).

EXAMPLES

```
GRIDSTRF:  String1  #G101   4    #G1     #G2  $ define two new gridpoints
$          between #G1 and #G2 and number them #G101, #G102, #G103, #G104.
$
```

```
GRIDSTRF:  String2  #G10   3    #G2H1   #G3H1 $ define one new gridpoint
$                  between #G2H1 and #G3H1 and number them #G10, #G11, #G12.
```

Extend a String of Equally Spaced Grid Points Out from a Point Which Has Already Been Defined

This command operates in a very similar way to the GRIDSTR and GRIDSTRF commands. In the first command it was assumed that neither of the endpoints of the gridstring already existed. In the second command it is assumed that both endpoints already existed. In the present command it is assumed that only one endpoint already exists.

The command to extend a string of gridpoints out from an existing gridpoint or jointpoint has the form,

GRIDSTRE, GRIDSTRINGE, GRSTRE

GRIDSTRE: gridstringname, first new grid#, ng, start point#, x, y, z, axesname

gridstringname	alphanumeric name for grid point string
first new grid #	identifier for first new gridpoint to be created, (#Gj or #CiGj where i = component number, j = gridpoint number). Subsequent gridpoints to be created in the string will be identified by gridpoint numbers incrementing by "one". If a component number is given, then they will belong to the same component specified.
ng	total number of grid points in the string, including the first. That is, the number of new gridpoints = (ng - 1).
startpoint#	identifier for starting point (#Gj, #CiGj, #GjJk, or #CiGjJk where i = component number, j = gridpoint number, k= jointpoint number).
x, y, z	coordinates of ending gridpoint.
axesname	the cartesian coordinate system in which the coordinates (x,y,z) are specified (must already have been created by an AXES command). If left blank, the default system will be assumed. The default system is the "GLOBAL" system unless another default system is specified in the DEFAULT command.

EXAMPLES

```
GRIDSTRE:  String1  #G101   4    #G1   1,1,1   $ define three new gridpoints
$          between #G1 and (1,1,1) and number them #G101, #G102, #G103, #G104.
$
GRIDSTRE:  String2  #G10   3    #G2H1   4,2.5,9 $ define two new gridpoints
$                  between  #G2H1 and (4,2.5,9) and number them #G10, #G11, #G12.
```

JOINTPOINTS

There are two possible general ways for a user to set up joints. The most computationally efficient way is through the use of a special type of gridpoint called a jointpoint. Jointpoints may be defined at any gridpoint, once the gridpoint itself has been defined. The other way that joints may be created is through the use of constraints which will be discussed later.

When a jointpoint is defined, only the degrees of freedom needed for that specific type of joint are added to the problem. This is not the case when joints are created through the use of constraints. Also constraints may need to be stabilized, a process which requires additional iterations at each timestep.

LATDYN contains three standard jointpoints - hinge (or revolute), ball, and universal. A jointpoint always occupies the same physical location in space as the gridpoint to which it is attached. **Only hingepoints are currently implemented**.

Defining a Hingepoint at a Grid Point

The simplest way to think of a gridpoint/hingepoint pair is as the two sides of a hinged joint (see Figure 5.1). A beam member or a lumped mass may be connected to either side of the joint. For example, if a gridpoint has a hingepoint attached to it then one beam member may be connected to the gridpoint itself and a second beam member may be connected to the hingepoint.

Just as when a gridpoint is defined LATDYN adds degrees of freedom to the model, so also are degrees of freedom added when a hingepoint is defined. The difference is that six degrees of freedom get added for each gridpoint that is defined, but only one degree of freedom gets added for a hingepoint. Thus a gridpoint by itself has six degrees of freedom while a gridpoint/hingepoint pair has seven degrees of freedom.Since this hingepoint cannot exist without the gridpoint to which it is attached, it is necessary that the gridpoint be defined first.

The HINGEPT command defines two related entities. The first entity is the hingepoint itself. The second entity is the Hinge Axis Vector. There are several options for a user to specify the initial direction of the hinge axis vector. After it has been defined, the hinge axis vector remains fixed to the gridpoint moving and rotating with it. The orientation of the hingepoint is obtained by a simple rotation from the gridpoint about the hinge axis. The magnitude of this rotation is the angular displacement of the hinge. At the start of a transient analysis, the angular displacement of the hinge is assumed to be zero, unless otherwise specified.

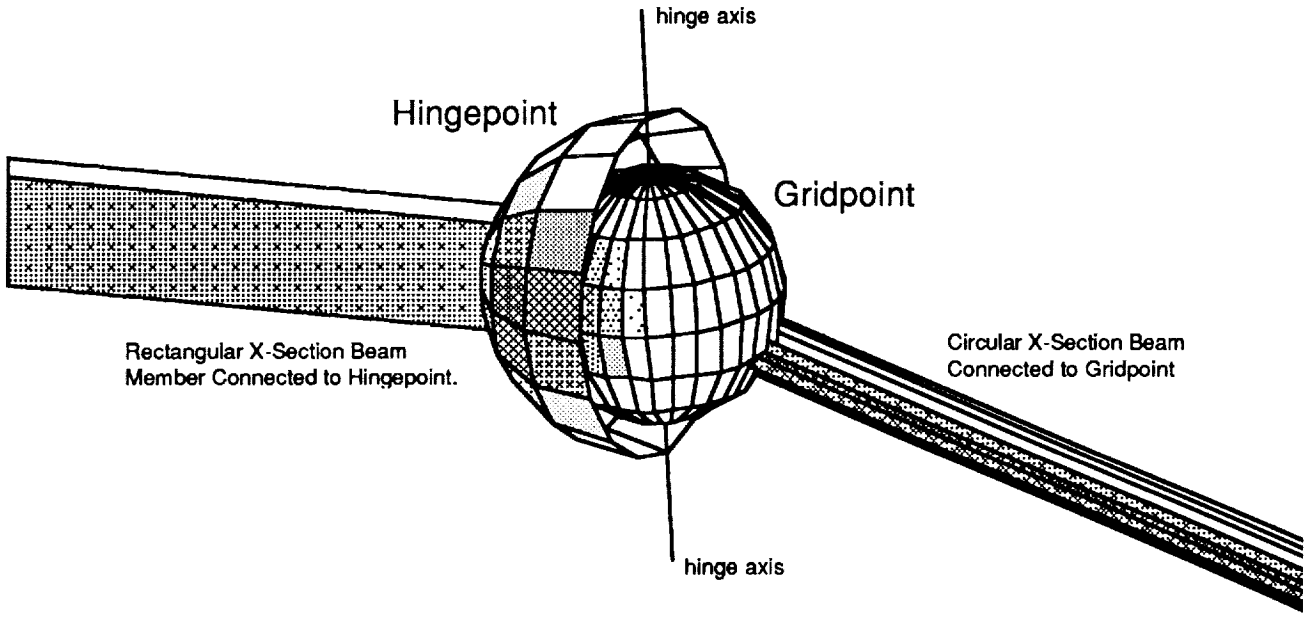


Figure 7.1 Conceptual Sketch of Two Beams Connected to Gridpoint/Hingepoint Pair

The HINGEPT command has the following form,

HINGEPT, HINGEPOINT, HPOINT, HPT

HINGEPT: hingepoint#, initial angle, OPTION for axis orientation spec.

hingepoint# point identifier (#GiHk or #CiGjHk where i = component number, j= gridpoint number, k= hinge number)

initial angle the initial rotation of the hinge (degrees)

OPTION for axis orientation specification - POINT, or VECTOR

POINT(axis point or coords) - Axis orientation is given by the specification of a point. The hinge axis vector being defined extends from the gridpoint to the axis point.

axis point or coords - gives the identifier of a reference point (#Rj or #CiRj) or a grid point (#Gj or #CiGj). An alternate form of the point specification is to give the coordinates (x,y,z,axesname). The initial position of the hinge axis is defined as the line joining the grid point to which the hinge-point is attached, with this reference point.

VECTOR (vect.ID for axis) - Axis orientation is given by a vector.

Vect. I.D. for Axis - may be a reference vector or a vector from a unit triad (R^i, I^axesname, J^axesname, K^axesname) Vector gives the initial orientation of the hinge axis.

EXAMPLES

```
HINGEPT:   #G3H1   90.0   VECTOR(R^1)
HINGEPT:   #G5H1   0.0    POINT(#R2)
HINGEPT:   #C3G4H1   0.0    POINT(12.2,10.6,13.3,GLO)
```

Defining a Universal Jointpoint at a Grid Point

This command is not yet implemented. The following brief explanation is intended to illustrate what directions we envisage LATDYN development to take.

The simplest way to think of a gridpoint/unipoint pair is as the two sides of a universal joint. A beam member or a lumped mass may be connected to either side of the joint,.just as was the case for gridpoint/hingepoint pair For example, if a gridpoint has a unipoint attached to it then one beam member may be connected to the gridpoint itself and a second beam member may be connected to the unipoint.

When a gridpoint is defined LATDYN adds six degrees of freedom to the model, when a unipoint is defined LATDYN adds another two degrees of freedom to the model. Thus a gridpoint by itself has six degrees of freedom while a gridpoint/unipoint pair has eight degrees of freedom.

Defining a Ball Jointpoint at a Grid Point

This command is not yet implemented. The following brief explanation is intended to illustrate what directions we envisage LATDYN development to take.

The simplest way to think of a gridpoint/ballpoint pair is as the two sides of a balljoint. A beam member or a lumped mass may be connected to either side of the joint,.just as was the case for gridpoint/hingepoint pair and a gridpoint/unipoint pair. For example, if a gridpoint has a ballpoint attached to it then one beam member may be connected to the gridpoint itself and a second beam member may be connected to the ballpoint.

When a gridpoint is defined LATDYN adds six degrees of freedom to the model, when a ballpoint is defined LATDYN adds another three degrees of freedom to the model. Thus a gridpoint by itself has six degrees of freedom while a gridpoint/ballpoint pair has nine degrees of freedom.

BEAM MEMBERS

Beams are a basic structural building block for many structures. Flexible beam elements are currently the only true finite element in the LATDYN library. Rigid beam members should not be regarded as finite elements in the usual sense since they are implemented either through constraints or through a transformation as a "rigid body offset".

Define a Flexible Beam Member Consisting of one Beam Element or a String of Beam Elements

The flexible beam finite element currently implemented in LATDYN uses a cubic functional for bending and linear functionals for torsion and stretching. The terminology used is that one beam member consists of one or more beam elements.

The selection of beam elements is important. If elements are too large, then they will not accurately represent large deformation of a beam or detailed stress distributions. If they are too small, then high natural frequencies, which may be of no practical interest in the dynamic motion, will be built into the system. If an explicit integrator is used these high frequencies will force a very small timestep for numerical stability, and the transient analysis will be slow.

The form of the command to define a flexible beam member is,

FMEMBER, FMEMB, FMEM

FMEMBER: Beam member #, OPTIONS for connection points, OPTIONS for orientation spec., beampropname

beam member # identifier for beam member being created by this command (#Mj or #CiMj)

OPTIONS for specifying grid points - (SINGLE or MULTI)

SINGLE (start grid#, end grid#) - for single element beam member

start grid # identifier of grid point or joint point connected at end "a" of member. (#Gj or #CiGj or #GjJk or #CiGjJk, where i = component number, j = gridpoint number, k= jointpoint number, and J= (H,B or U) is the joint identifying letter.)

end grid # identifier of grid point connected at end "b" of member (#Gj or #CiGj or #GjJk or #CiGjJk, where i = component number, j = gridpoint number, k= jointpoint number, and J= (H,B, or U) is the joint identifying letter.)

The coordinate locations of the start grid# and end grid# may not be coincident.

MULTI (gridstringname) - for multi-element beam members

gridstringname - this parameter should reference a gridstring name as specified in a GRIDSTR or GRIDSTRF command.

OPTIONS for orientation specification - (POINT or VECTOR)

The X-axis of a beam member is defined to lie along its length. The y- and z-axes are the principal axes of the beam. The following orientation options are used to orient the (principal) y-axis of the beam member. The (principal) z-axis is then

POINT(ref point #, grid#, or x,y,z,axesname) - Specification of the principal y-axis orientation by a point

ref point #, grid#, or x,y,z,axesname) - The y-axis of the beam lies in the plane which contains the x-axis of the beam and this point. The form of the argument for a reference point is (R#j or C#iR#j). Alternatively another grid point may be used to define this plane (#Gj or #CiGj), or the coordinates of a point in a specified axes system may be given (x,y,z,axesname).

VECTOR (ref. or unit triad vector I.D.) - Specification of principal y-axis orientation by a reference vector

ref. or unit triad vector I.D. - R^k, I^axesname, J^axesname, or K^axesname. The principal y-axis of the beam lies in the plane containing this vector and the start grid#.

beampropname references a beampropname defined in a BEAMPROP command

EXAMPLES

```
FMEMBER:  #M1, SINGLE(#G1,#G2), POINT(#R1), post_arm
FMEMBER:  #M3, MULTI(Truss_1), VECTOR(R^3), Truss
FMEMBER:  #M4, MULTI(Truss_2), VECTOR(R^3), Truss
FMEMBER:  #C'Top'M1, SINGLE(#G5,#G6), POINT(0.0 12.3 18.9, TopAxes), &
          TopRib
```

Define a Rigid Beam Member

Although rigid members are generated from the users point of view, very much like flexible members, they are quite different in the program. Rigid members are not finite elements in the conventional sense. Eventually LATDYN will contain three different ways of generating rigid members, but at the moment there is only one methodology that has been implemented, that is "rigid body offsets".

Rigid body offsets are implemented by a transformation from a gridpoint. Inside the computational core of the program, only the degrees of freedom for the gridpoint at one end of the member are present. No extra degrees of freedom for other gridpoints exist, although this fact is transparent to the user.

Because this methodology is very efficient and accurate, since there are no constraints to stabilize, this is the preferable way to implement rigid members in LATDYN.

In the way that they are presently implemented, rigid body offsets are quite restrictive. For example, it is not possible to connect two rigid bodies together through a joint, using the rigid body offset option. We plan ultimately to remedy this with the implementation of a general "recursive" formulation, and this option does currently appear in the command. At present however, only the rigid body offset is implemented and this will be assumed by the program in all cases.

RMEMBER, RMEMB, RMEM

RMEMBER: Beam member #, OPTIONS for connection points, OPTIONS for orientation spec., beampropname, OPTIONS for analysis algorithm

beam member # — identifier for beam member being created by this command (#Mj or #CiMj)

OPTIONS for specifying grid points - SINGLE or MULTI

SINGLE (start grid#, end grid#) - for single element beam member.

start grid # — identifier of gridpoint or jointpoint connected at end "a" of member (#Gj or #CiGj or #GjJk or #CiGjJk, where i = component number, j = gridpoint number, k= jointpoint number, and J= (H,B or U) is the joint identifying letter.)

end grid # — identifier of grid point connected at end "b" of member. (#Gj or #CiGj or #GjJk or #CiGjJk, where i = component number, j = gridpoint number, k= jointpoint number, and J= (H,B,U, or T) is the joint identifying letter.)

The coordinate locations of the start grid# and end grid# may not be co-incident.

MULTI(gridstringname) - for multi-element beam members

gridstringname - This parameter should reference a gridstring name as specified in a GRIDSTR or GRIDSTRF command.

OPTIONS for orientation specification - POINT or VECTOR

The X-axis of a beam member is defined to lie along its length. The y- and z-axes are the principal axes of the beam. The following orientation options are used to orient the (principal) y-axis of the beam member. The (principal) z-axis is then automatically oriented by the program.

POINT(ref point #, grid#, or x,y,z,axesname) - Specification of y-axis orientation by a point

ref point #, grid#, or x,y,z,axesname - The y-axis of the beam is given by the line joining the starting point of the beam member with the reference point (R#j or C#iR#j). Alternatively another grid point may be used to define this plane (#Gj or #CiGj), or the coordinates of a point in a specified axes system may be given (x,y,z,axesname).

VECTOR(ref. or unit triad vector I.D.) - Specification of orientation by a reference vector

ref. or unit triad vector I.D. - R^k, I^axesname, J^axesname, or K^axesname

beampropname references a beampropname defined in a BEAMPROP command. (For a rigid member, the values specified for stiffeness are irrelevant.)

OPTIONS for analysis algorithm - CNSTRNT, OFFSET, RECUR, AUTO
NOTE: Only the OFFSET option is presently implemented.

CNSTRNT The rigid beam member is to be implemented through constraints.

OFFSET The rigid beam is to be implemented as a rigid body offset. That is, the second grid point specified in the command (or, each gridpoint after the first in the case of a gridstring) will be treated in the program as a simple rigid body offset from the first gridpoint.

For this option to be used, the following must be true:
o Both gridpoints mentioned in the RMEMBER command must be simple gridpoints. Jointpoints are not allowed.
o Only the first gridpoint may appear in another RMEMBER or RBODY command. Neither the second gridpoint itself, or any jointpoints attached to it may be part of any other rigid body or rigid member.
o If the first gridpoint appears in other RMEMBER or RBODY commands which also specify the "OFFSET" option, then this gridpoint must be the first gridpoint in these commands.

RECUR(Treename) This rigid beam member is to be treated as part of an open loop rigid tree mechanism.

Treename This parameter refers to a recursion path set up in a RBTREE command. The rigid beam member connectivity must follow the rules for an open loop rigid body tree.

AUTO This is the default option. Algorithms in the LATDYN program will choose which option to implement.

EXAMPLES

```
RMEMBER:  #M1, SINGLE(#G1,#G2), POINT(#R1), post_arm, OFFSET
RMEMBER:  #M3, MULTI(Truss_1), VECTOR(R^3), Truss, AUTO
RMEMBER:  #M4, MULTI(Truss_2), VECTOR(R^3), Truss, CNSTRNT
RMEMBER:  #C'Top'M1, SINGLE(#G5,#G6), POINT(0.0 12.3 18.9, TopAxes), &
          TopRib
```

RIGID BODIES

Rigid bodies are generated in exactly the same way as rigid members. The only difference is that the RMEMBER command automatically adds that mass which is appropriate for the beam member being specified, whereas the RBODY command adds no mass. It is left to the user to add the appropriate rigid lumped mass by the ADMASS command.

Eventually LATDYN will contain three different ways of generating rigid members and rigid bodies, but at the moment there is only one methodology that has been implemented, that is "rigid body offsets".

Rigid body offsets are implemented by a transformation from a gridpoint. Inside the computational core of the program, only the degrees of freedom for the gridpoint at one end of the member, or one gridpoint in the body, are present.

Define a Rigid Body

RBODY, RIGIDBODY, RIG

RBODY: bodyname, grid#, grid#, ..., OPTION for analysis algorithm

bodyname	alphanumeric identifier. (The same name may not be used for two rigid bodies.)
grid#, grid#, ...	list of gridpoints located on the rigid body which is being defined by this command
OPTIONS for analysis algorithm - CNSTRNT, OFFSET, RECUR, AUTO	**NOTE: Only the OFFSET option is presently implemented.**
CNSTRNT	The rigid body is to be implemented through constraints.
OFFSET	The rigid body is to be implemented as a rigid offsets from the first gridpoint specified. That is, subsequent grid points specified in the command will be treated in the program as a simple rigid body offsets from the first gridpoint.
	For this option to be used, the following must be true: o All gridpoints mentioned in the RBODY command must be simple gridpoints. Jointpoints are not allowed. o Only the first gridpoint may appear in another RMEMBER or RBODY command. Neither the second gridpoint itself, or any jointpoints attached to it may be part of any other rigid body or rigid member. o If the first gridpoint appears in other RMEMBER or RBODY commands which also specify the "OFFSET" option, then this gridpoint must be the first gridpoint in these commands.
RECUR(Treename)	This rigid body is to be treated as part of an open loop rigid tree mechanism.
Treename	This parameter refers to a recursion path set up in a RBTREE command. The rigid body connectivity must follow the rules for an open loop rigid body tree.
AUTO	This is the default option. Algorithms in the LATDYN program will choose which option to implement.

EXAMPLES

```
RBODY: Pivot   #G3,#G4,#G5,#G7     AUTO
RBODY: Root    #C2G1 #C2G2    OFFSET
RBODY: Small_Arm     #G9,#G10,#G11    CNSTRNT
```

Define a Recursion Path for a Jointed Tree of Rigid Bodies and/or Rigid Beam Members

This command is not yet implemented. The following brief explanation is intended to illustrate what directions we envisage LATDYN development to take.

Rigid bodies may be created in LATDYN using several different commands (RMEMBER for rigid beam members, RBODY for generalized rigid bodies). The way in which these commands are implemented in the LATDYN computational core may be controlled by the user, or may be left to the program itself to decide by using the AUTO option.

If a user elects to direct the program to set up an open loop jointed tree of rigid bodies by recursive (rigid body) transformations then it is necessary that he also specify the paths back to the "root" that the recursion must take. The way in which a user may specify a recursion path is in the RBTREE command.

LUMPED MASSES

Lumped masses are created by the ADMASS command which is described below. First however, it is necessary to specify the properties of the lumped mass which is to be created. This is done using the MASSPROP command.

The MASSPROP command allows the properties of a lumped mass to be specified. A lumped mass does not actually become part of the structure, until it is created and "stuck" onto a gridpoint or jointpoint by an ADMASS command. Each ADMASS command refers to a set of lumped mass properties specified by a MASSPROP command. When a lumped mass is created by an ADMASS command, these properties are transformed to the required position and orientation. Thus, a lumped mass may be specified in a MASSPROP command using the most convenient coordinate system for the user since it may be oriented and positioned later.

Add a Mass to a Grid Point or Jointpoint

The ADMASS command takes a lumped mass which was defined in a MASSPROP command and attaches it to a gridpoint or jointpoint. The command has the form:

ADMASS, ADDMASS, ADMAS, ADD

ADMASS: grid #, mass name, axesname	
grid #	grid point identifier (#Gj or #CiGj or #GjJk or #CiGjJk, where i = component number, j = gridpoint number, k= joint point number).
massname	references a mass name defined in a MASSPROP command

axesname refers to a coordinate system defined by an AXES command. The moments and products of inertia specified in the MASSPROP command referenced above are then taken to be given with respect to this coordinate system. If this parameter is left blank, then the GLOBAL system is used.

EXAMPLES

```
ADMASS: #G6  Pivot  Pivot_System
ADMASS: #G4  Link_Arm   Link_Arm_System
```

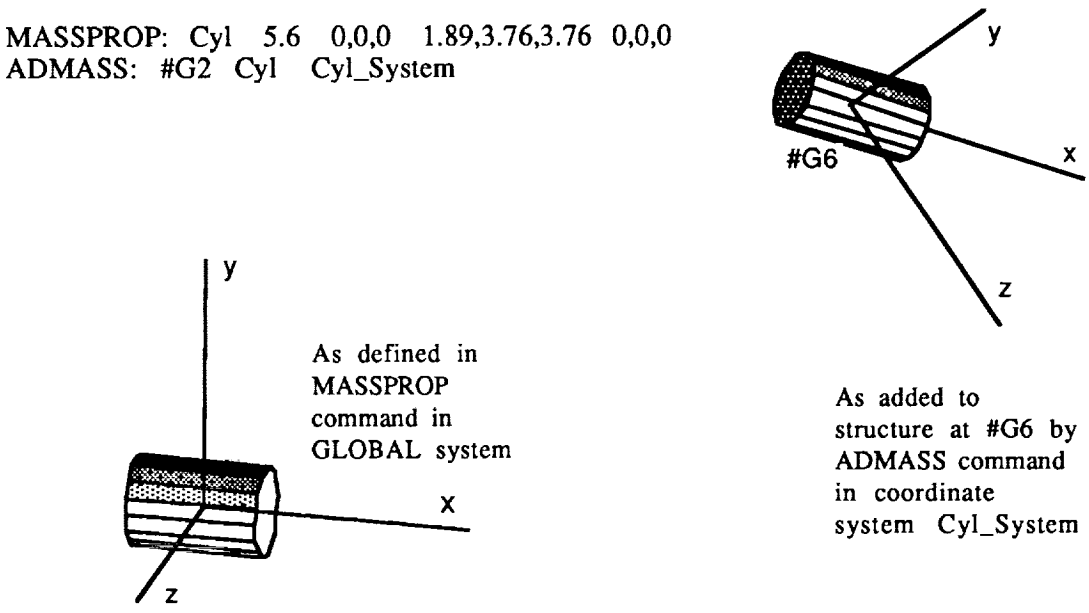


Figure 7.2 Example of the Use of the MASSPROP and ADMASS Commands

ROTATIONAL ELEMENTS

Rotational elements in LATDYN consist of rotational springs, rotational dampers, and rotational actuators. Rotational springs and dampers are passsive linear elements. Rotational actuators may be non-linear active or passive devices and a substantial amount of user control may be applied to creating them. Rotational actuators are described under applied loads in Chapter 10. A specific type of rotational non-linear joint is also included in the command set, but this has not yet been implemented. This specific type of joint may in any case be created by the user as a rotational actuator.

Rotational springs and dampers always act about hinge lines, so that commands which create these el;ements require reference to a hinge line. These hinges may have been created by transformations (adding degrees of freedom) or imposed by constraints (removing degrees of freedom). In the case of hinges which have been created by transformations, references will be to hingepoints. In the case of hinges which have been created by constraints, references will be to a hinge joint constraint name.

Define a Rotational Spring

ROTSPRING, ROTSPR, RSPRING, RSPR

ROTSPRING: spring name, OPTION for connecting ends of spring, k, p ? Cj

spring name — alphanumeric name for rotational spring (Note: no two rotational springs may have the same name.)

OPTION for connecting ends of spring - UX, CX, BX, or GX

UX (hingepoint #) — uniaxial connection for connection between a grid point and a hingepoint attached to it.

CX (hingepoint a#, hingepoint b#) - co-axial connection for connection between two coaxial hingepoints attached to the same grid point.

BX (hingepoint a#, hingepoint b#) - bi-axial connection for connection between any two hingepoints attached to the same gridpoint, or between two different gridpoints on the same rigid body or rigid member.

GX (hinge constraint name) - general connection for a rotational spring about a hinge axis between two gridpoints, where the hinge is defined by constraints. The hinge constraint name is defined by a HINGEJOINT command. **NOTE: The GX option is not presently implemented.**

k — spring constant (real value)

p — pre-rotation. Parameter allows definition of spring pretension by allowing a pre-rotation of the second connection point. A positive pre-rotation is clockwise looking along the axis vector of the second hingepoint.

Cj — optional condition label.

When a ROTSPRING command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
ROTSPRING:  Pivot_Spring   UX(#G1H1)   3.625   0
ROTSPRING:  Sp2   UX(#G16H1)   Q3   0.0   ? C3
ROTSPRING:  Spring6   CX(#G2H1,#G2H2)   17.56   -90
```

Define a Rotational Damper

ROTDAMPER, ROTDAMP, RDAMPER, RDAMP

ROTDAMPER: dampername, OPTION for connection of ends, c ? Cj

damper name — alpha numeric name for rotational damper. (Note: no two rotational dampers may have the same name.)

OPTION for connection of ends - UX, CX, BX, or GX

UX(hingepoint #) — uniaxial connection for connection between a grid point and a hingepoint attached to it.

CX(hingepoint a#, hingepoint b#) - co-axial connection for connection between two coaxial hinge degrees of freedom attached to the same grid point.

BX(hingepoint a#, hingepoint b#) - bi-axial connection for connection between any two hinge degrees of freedom attached to the same grid-point, or between two different gridpoints on the same rigid body.

GX(hinge constraint name) - general connection for a rotational damper about a hinge axis between two gridpoints, where the hinge is defined by constraints. The hinge constraint name is defined by a HINGEJOINT command. **NOTE: The GX option is not presently implemented.**

c — damping constant (real value)

Cj — optional condition label (Cj or CLj, j is an integer 1 to 99)

When a HINGEJOINT command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
ROTDAMPER:  Pivot_Damper   UX(#G1H1)   8.78
ROTDAMPER:  D2   UX(#G16H1)   Q3   ? C3
ROTDAMPER:  Damper7   CX(#G2H1,#G2H2)   17.56
```

Define a Rotational Non-Linear Joint

This command postulates a specific type of rotational non-linear joint where the joint contains both spring and damping aspects. That is, joint forces are proportional to a two dimensional function of displacement and velocity which may be input in tabular form. Since this command has not yet been implemented, this specific type of joint may be created by using a rotational actuator command.

LINEAL ELEMENTS

Lineal elements act in a line between two gridpoints. The force exerted by a lineal element on each of the endpoints is equal in magnitude and opposite in direction. In the case of a lineal spring, the force is proportional to the displacement of the two points along the line of action of the spring. In the case of a lineal damper, the force is proportional to the relative velocity of the two points along the line of action of the damper.

Complicated lineal springs and dampers with non-linear spring and damping constants may be created easily, by using the lineal actuator command which is documented under the section on applied loads in Chapter 10.

Flexible impacts may be analysed by creating a lineal spring and damper between the two impacting parts of the structure, at the instant of impact, through the use of conditions.

Define an Lineal Spring Acting in a Line Between Two Grid Points

LINSPRING, LINSPR, LSPRING, LSPR

LINSPRING: spring name, grid a#, grid b#, k; prestretch ?Cj

Spring name	alphanumeric spring name. (Note: two lineal springs may not have the same name.)
grid a #	grid point identifier for one end of spring (#Gj or #CiGj)
grid b#	grid point identifier for other end of spring (C#j or #CiGj)
k	spring constant (real value)
prestretch	spring prestretch (negative for compression, positive for extension)
Cj	optional condition label (Cj or CLj, j is an integer 1 to 99)
	When a LINSPRING command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
LINSPRING:  Spring1  #G1   #G2    3.56E4   -3.0
LINSPRING:  Spring2  #G43  #G9    Q16  0   ?C1
LINSPRING:  Puller    #C2G4  #C5G2   9.5   0.0
```

Define an Extensional (Lineal) Damper Acting in a Line Between Two GridPoints

LINDAMPER: damper name, grid a#, grid b#, c ? Cj

Damper name	alphanumeric damper name. (Note: two lineal dampers may not have the same name.)
grid a #	grid point identifier for one end of damper (#Gj or #CiGj)
grid b#	grid point identifier for other end of damper (C#j or #CiGj)
c	damper constant (real value)
Cj	optional condition label (Cj or CLj, j is an integer 1 to 99)
	When a LINDAMPER command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
LINDAMPER:  Damper1   #G1   #G2    3.56E4
LINDAMPER:  Damper2   #G43  #G9    Q16        ?C1
LINDAMPER:  Puller    #C2G4 #C5G2  9.5
```

8. CONSTRAINTS

Constraints in some areas of mechanics are also called boundary conditions. Mathematically they consist of an additional set of equations which "constrain" the model to behave in a certain way. For example, two gridpoints may be clamped together so that they may not separate or rotate relative to each other. The equations constraining this motion consist of six linear equations setting the displacements and rotations of the two gridpoints equal to each other in each of their six degrees of freedom.

However, LATDYN essentially solves Newton's second law at every timestep, namely,

$$M \ddot{x} = f$$

This approach requires constraint equations to be expressed in terms of acceleration.

In the case of the six linear equations to clamp one gridpoint to another, described above, it is easy to differentiate each equation twice (with respect to time) and apply the constraints. But this is almost a trivial case. There are many other standard constraints which are frequently needed that are not nearly so simple (such as hinge joints, universal joints, cylindrical joints, etc.), which are complicated functions of angle and involve derivatives of rotational transformations.

For many standard constraints, LATDYN takes care of this complexity for the user, and it is only necessary to specify the constraint in a command to have it automatically applied. The list of constraints which are built into LATDYN include hinge joints, universal joints, ball joints, cylindrical joints, translational joints, constant distance links, fixing and clamping, and geared drive connections.

In LATDYN all constraint commands may be qualified by an optional condition label. This means that the constraint command is active when the condition is true and inactive when it is false. But, constraints are an integral part of the structural model and although LATDYN allows you to add or delete them at will, it does not guarantee that you will get the correct answers if you do this.

For example, if two gridpoints are moving towards each other during a transient analysis and you build in a test to sense when they touch, and then use this test to turn on a CLAMP constraint - you will not get the right answer. This is because the two gridpoints have impacted. To get the correct answer the principal of conservation of momentum must be applied. LATDYN does not do this automatically. (One way to resolve the problem is to create an elastic impact by adding a spring and damper between the two gridpoints and waiting until the relative velocities become zero before applying the constraint.)

In general, it is okay to use a condition to turn off a constraint which has been present since the start of the analysis, but the user must be cautious when turning on a constraint during an analysis.

JOINTS IMPLEMENTED BY CONSTRAINTS

There are two possible general ways for a user to create joints between flexible or rigid bodies. The most computationally efficient way is through the use of a special type of gridpoint called a jointpoint as was explained in Chapter 7. When a jointpoint is defined, only the degrees of freedom needed for that specific type of joint are added to the problem. For example, suppose a hinge is to be created by making a hingepoint at a gridpoint. The original gridpoint had six degrees of freedom, the hingepoint adds one for a total of seven.

The other way that joints may be created is through the use of constraints between two gridpoints. For example, suppose a hinge joint is to be created between two gridpoints by constraints. Each gridpoint has six degrees of freedom and there are also five constraint equations for a total of seventeen equations to be solved at each timestep. Of course, there are still only seven real (independent) degrees of freedom, because the five constraint equations are used to eliminate five degrees of freedom from the twelve of the two gridpoints, but there is a lot of extra computational work involved.

Define a Hinge (Revolute) Joint via Constraints

Two gridpoints may be constrained together to operate as a hinge joint. A HINGEJOINT command imposes five constraints which removes five degrees of freedom from the twelve to give a net total of seven independent degrees of freedom at the joint.

The five constraints defined by the command are given a collective name by the user so that they may be referenced in other commands (such as ROTSPRING).

HINGEJOINT, HNGEJNT, HJOINT, HJNT, HINGE

HINGEJOINT: constraint name, grida#, gridb#, OPTION for axis orientation spec. ? Cj

constraint name — alphanumeric name for the constraints defined by the hingejoint command. (Note: two hingejoints may not have the same name.)

grida#, gridb# — identifiers of gridpoints to be joined by a hinge (#Gj or #CiGj). Note: jointpoints cannot be used in this command. Also note: the two gridpoints must be in the same physical location.

OPTION for axis orientation specification - POINT or VECTOR

POINT(axis point or coords) - Axis orientation is given by the specification of a point. The hinge axis vector being defined extends from the gridpoint to the axis point.

axis point or coords - gives the identifier of a reference point (#Rj or #CiRj) or a grid point (#Gj or #CiGj). An alternate form of

the point specification is to give the coordinates (x,y,z,axesname). The initial position of the hinge axis is defined as the line joining the grid point to which the hinge-point is attached, with this reference point. Initial hinge rotation is zero.

VECTOR (vectID for axis) - Axis orientation is given by a vector.

Vect. I.D. for Axis - may be a reference vector or a vector from a unit triad (R^i, I^axesname, J^axesname, K^axesname). Vector gives the initial orientation of the hinge axis. Initial rotation angle of the hinge is zero.

Cj — optional condition label (Cj or CLj, j is an integer from 1 to 99).

When a HINGEJOINT command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
HINGEJOINT:  RevJnt1   #G1  #G2   POINT(#R1)
HINGEJOINT:  HJ16      #G23 #G24  POINT(1.3,4.2,0.0,GLO)
HINGEJOINT:  Hinge_Joint4  #C1G2 #C1G3  VECTOR(I^GLO)  ? C1
```

Define a Universal Joint via Constraints

Two gridpoints may be constrained together to act as a universal joint. Each gridpoint by itself has six degrees of freedom for a total of twelve. A UNIJOINT command imposes four constraints which remove four degrees of freedom from the twelve to give a net total of eight independent degrees of freedom at the joint.

The four constraints defined by the command are given a collective name so that they may be referenced in other commands.

UNIJOINT, UNIJNT, UJOINT, UJNT, UNIV, UNI

UNIJOINT: constraint name, grida#, gridb#, OPTION for axis orient. spec. ?Cj

constraint name — alphanumeric name for the constraints defined by the unijoint command. (Note: two unijoints may not have the same name.)

grida#, gridb# — identifiers of gridpoints to be joined by a universal joint (#Gj or #CiGj). Note: jointpoints cannot be used in this command. Also note: the two gridpoints must be in the same physical location.

OPTION for axis orientation specification - POINTS or VECTORS.

Note: For a conventional universal joint, the two axes specified should be normal to each other.

Note: The first axis specified is attached to grida#, the second to gridb#.

POINTS(axis point or coords, axis point or coords) - Orientation of the two axes is given by the specification of two points. The axis vectors being defined extend from the gridpoint to the axis points.

axis point or coords - gives the identifier of a reference point (#Rj or #CiRj) or a grid point (#Gj or #CiGj). An alternate form of the point specification is to give the coordinates (x,y,z,axesname). The initial position of the hinge axis is defined as the line joining the grid point to which the joint is attached, with this point.

VECTORS(axis vect.ID, axis vect.ID) - Orientation of the two axes of the universal joint is given by two vectors.

Vect. I.D. for Axis - gives hinge axis on gridpoint may be a reference vector or a vector from a unit triad (R^i, I^axesname, J^axesname, K^axesname). Vector gives the initial orientation of the hinge axis.

Cj optional condition label (Cj or CLj, j is an integer from 1 to 99).

When a UNIJOINT command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
UNIJOINT:  Uj3    #G1 #G2   POINTS(16.3,14.2,0.0,GLO,#G12)
UNIJOINT:  Uni_Joint  #G56 #G57  VECTORS(K^JSys, I^JSys)
UNIJOINT:  U1     #C16 #C17  POINTS(#G1,#G2)
```

Define a Balljoint via Constraints

Two gridpoints may be constrained together to act as a ball joint. Each gridpoint by itself has six degrees of freedom for a total of twelve. A BALLJOINT command imposes three constraints which remove three degrees of freedom from the twelve to give a net total of nine independent degrees of freedom at the joint.

The three constraints defined by the command are given a collective name so that they may be referenced in other commands.

BALLJOINT: constraint name, grid a#, grid b# ? Cj

constraint name alphanumeric name for the constraints defined by the BALLJOINT command.(Note: two balljoints may not have the same name.)

grid a#, grid b# identifiers of the two grid points to be balljointed together.

Note: jointpoints cannot be used in this command. Also note: the two gridpoints must be in the same physical location.

Cj optional condition label (Cj or CLj, j is an integer from 1 to 99).

When a BALLJOINT command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
BALLJOINT:  BallJoint1  #G1   #G2
BALLJOINT:  Bj3     #G5,#G6
BALLJOINT:  B7      #C'Rod'G1    #C'Mount'G2
```

Define a Cylindrical Joint Linking Two Gridpoints via Constraints

Two gridpoints may be constrained together to operate as a cylindrical joint in which one gridpoint is free to translate along and rotate about a specified axis which is fixed relative to another gridpoint.

CYLJOINT, CYLJNT, CJOINT, CJNT

CYLJOINT: constraint name, grida#, gridb#, OPTION for axis orient. spec. ?Cj

constraint name — alphanumeric name for the constraints defined by the CYLJOINT command. Note: the same constraint name cannot be used for two CYLJOINTs.

grida#, gridb# — identifiers of gridpoints to be joined by a cylindrical joint (#Gj or #CiGj). Note: that jointpoints cannot be used in this command.

NOTE: The axis of the cylindrical joint is defined by the line joining the two points. If the points are coincident (that is, they occupy the same physical position in space) then it is necessary to use the following OPTION for specifying the initial axis orientation. If the two gridpoints are at separate locations then the use of this option will result in an error.

OPTION for initial axis orientation specification - POINT or VECTOR

POINT(axis point or coords>)- Axis orientation is given by the specification of a point. The axis vector being defined extends from either gridpoint (since they must be coincident for this option to be used) to the axis point.

axis point or coords - gives the identifier of a reference point(#Rj or #CiRj) or a grid point (#Gj or#CiGj), or the coordinates of a point (x,y,z,axesname). The initial position of theaxis is defined as the line joining grida# or gridb# with this point. Initial relative rotation angle is zero.

VECTOR (vectID for axis) - Initial axis orientation is given by a vector.

Vect. I.D. for Axis - may be a reference vector or a vector from a unit triad (R^i, I^axesname, J^axesname, K^axesname). Vector gives the initial orientation of the hinge axis. Initial rotation angle of the joint is zero.

Cj — optional condition label (Cj or CLj, j is an integer from 1 to 99).

When a CYLJOINT command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

CYLJOINT: Cyl_Joint1 #G1 #G2
CYLJOINT: CJ3 #G5 #G6 VECTOR(R^6) $ #G5 and #G6 are initially coincident
CYLJOINT: Cyljoint #C5G1, #C5G2 POINT(3.5,6.7,9.0,GLO) $ so are #C5G1 and 2

Define a Translational Joint Linking Two Gridpoints via Constraints

Two gridpoints may be constrained together to act as a translational joint in which one gridpoint is free to translate along a specified axis which is fixed relative to another gridpoint. Both gridpoints are fixed in their relative orientations and may not rotate relative to each other.

TRANSJOINT, TRANSJNT, TJOINT, TJNT

TRANSJOINT: constrnt. name, grida#, gridb#, OPTION for axis orient. spec. ?Cj

constraint name	alphanumeric name for the constraints defined by the TRANSJOINT command. Note: the same name may not be used for two TRANSJOINTs.
grida#, gridb#	identifiers of gridpoints to be joined by a translational joint (#Gj or #CiGj). Note: that jointpoints cannot be used in this command.

NOTE: The axis of the translational joint is defined by the line joining the two points. If the points are coincident (that is, they occupy the same physical position in space) then it is necessary to use the following OPTION for specifying the initial axis orientation. If the two points are separated in space then use of this option will result in an error.

OPTION for initial axis orientation specification - POINT or VECTOR

POINT(axis point or coords) - Axis orientation is given by the specification of a point. The axis vector being defined extends from either gridpoint (since they must be coincident for this option to be used) to the axis point.

axis point or coords - gives the identifier of a reference point(#Rj or #CiRj) or a grid point (#Gj or#CiGj), or the coordinates of a point (x,y,z,axesname). The initial position of the axis is defined as the line joining grida# or gridb# with this point. Initial relative rotation angle is zero.

VECTOR(vect.ID for axis) - Initial axis orientation is given by a vector.

Vect. I.D. for Axis - may be a reference vector or a vector from a unit triad (R^i, I^axesname, J^axesname, K^axesname). Vector gives the initial orientation of the hinge axis. Initial rotation angle of the joint is zero.

Cj	optional condition label (Cj or CLj, j is an integer from 1 to 99).
	When a TRANSJOINT command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
TRANSJOINT:  Trans_Joint1  #G1  #G2
TRANSJOINT:  CJ3    #G5  #G6   VECTOR(R^6)
$                                          #G5 and #G6 are initially coincident
TRANSJOINT:  Transjoint    #C5G1, #C5G2   POINT(3.5,6.7,9.0,GLO)
$                                          so are #C5G1 and 2
```

MISCELLANEOUS SPECIAL CONSTRAINTS

Other special constraints include the ability to fix a gridpoint at a particular location in space, or to clamp two gridpoints together. Also included is a constant distance link constraint.

Fix a Grid Point

The FIX command fixes a gridpoint at its present location and orientation in the global system. The command has the form

FIX

FIX: constraint name, grid # ? Cj

constraint name	alphanumeric name for the constraints defined by the FIX command. (Note: two FIX commands may not have the same name.)
grid #	identifier of the grid point to be fixed (#Gj or #CiGj)
Cj	optional condition label (Cj or CLj, j is an integer from 1 to 99).
	When a FIX command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
FIX:  Fix_G1  #G1
FIX:  F7    #G7  ?C3
```

Clamp Two Grid Points Together Rigidly

Usually there is no need to clamp two gridpoints together, since many members or other elements may be connected to a single gridpoint. One use for this command, however, is to let two gridpoints come together or separate during a transient analysis, since the command may be turned on or off by a condition.

CLAMP

CLAMP: constraint name, grid a #, grid b# ? Cj

constraint name	alphanumeric name for the constraints defined the CLAMP command. (Note: two CLAMP commands may not have the same name.)
grid a#, grid b#	identifiers of the two grid points to be clamped together.
Cj	optional condition label (Cj or CLj, j is an integer from 1 to 99).
	When a CLAMP command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

CLAMP: Clamp_1&2 #G1 #G2
CLAMP: clp8 #G8 #G9

Define a Constant Distance Link Between Two Gridpoints

A massless link may be defined between two gridpoints such that the distance between them is constrained to be constant. The relative orientation of the two points is NOT constrained.

DISTLINK, DISLINK, DLINK, NOMASSLINK

DISTLINK: constraint name, grida#, gridb# ?Cj

constraint name	alphanumeric name for the constraints defined by the DISTLINK command. Note: the same constraint name cannot be used for two DISTLINKs.
grida#, gridb#	identifiers of gridpoints to be joined by a massless link (#Gj or #CiGj). Note: that jointpoints cannot be used in this command.
Cj	optional condition label
	When a DISTLINK command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES
DISTLINK: Link1 #G67 #G68
DISTLINK: Link2 #G5 #G6

GENERALIZED CONSTRAINTS

All the constraint comands described in the previous section have been programmed into LATDYN and may simply be invoked by the user. Additional constraint capabilities also exist to allow the advanced user to input and apply customized constraint equations.

Define a Single Degree of Freedom Constraint

This command allows the user to constrain a single degree of freedom of a specified gridpoint (or the hinge motion of a hingepoint).

SDFC, SPC

SDFC: constraint name, grid #, d.o.f. code, rhs ? Cj

constraint name	alphanumeric name for the constraint defined by the command. (Note: two SDFC commands may not have the same name.)
grid #	grid point identifier (#Gj or #CiGj or #GjJk or #CiGjJk, where i = component number, j = gridpoint number, k= joint number).
d.o.f. code	identifies the degree of freedom to be constrained X, Y, or Z - gridpoint translation WX, WY, or WZ - gridpoint rotation H - hinge rotation (grid# must be a hingepoint)
rhs	right hand side. (May be a real number, a Q-variableor a T-variable). Gives the acceleration of the specified degree of freedom.
Cj	optional condition label (Cj or CLj, j is an integer from 1 to 99). When a SDFC command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false. Note: Constraints in LATDYN are applied to acceleration. For example, a single degree of freedom constraint, $x = 0$ should be differentiated twice by the user before being applied. In this particular case nothing changes since the equation is simply applied as, $\ddot{x} = 0$ but the case of a non zero rhs gives the acceleration of the degree of

freedom. For example, $\ddot{x} = Q1$
sets the acceleration of the specified degree of freedom "x".

EXAMPLES

```
SDFC: X_limit  #G4  X  0.0
SDFC: No_Rot_WX  #G7  WX  0
SDFC: No_Rot_WY  #G7  WY  0
SDFC: No_Rot_WZ  #G7  WZ  0
```

Define a Multi-Degree of Freedom Constraint

This command allows the user to input any constraint equation which he or she has derived. By allowing Q-variables as coefficients, it also includes the capability for any non-linear constraint which is an arbitrary function of the model state vector.

It is important for the user to understand that constraints in LATDYN are applied to acceleration. Thus all coefficients must be derived on this basis (see notes in SDFC command).

MDFC,MPC

```
MDFC: constraint name, grid #, d.o.f.code, coefficient,  &
          grid #, d.o.f. code, coefficient,  &
          ... grid #, d.o.f. code, coefficient,      rhs ? Cj
```

Constraint name	alphanumeric name for the constraint defined by this command.(Note: no two MDFC commands may have the same name.)
grid #, d.o.f. code, coefficient	- defines a term in the constraint relation
grid #	grid point identifier (#Gj or #CiGj or #GjJk or #CiGjJk, where i = component number, j = gridpoint number, k= offset point number).
d.o.f. code	identifies the degree of freedom to be constrained X, Y, or Z - gridpoint translation WX, WY, or WZ - gridpoint rotation H - hinge rotation (grid# must be a hingepoint)
Coefficient	may be a real number, a Q-variable or a T-variable)
rhs	right hand side (may be a real number, a Q-variable, or a T-variable)
Cj	optional condition label (Cj or CLj, j is an integer from 1 to 99). When a MDFC command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
MDFC: Special_1  #G5,X,1.0 #G5,Y,1.0  #G5,Z,1.0  #G6,X,Q1          &
                                                  #G6,Y,Q2 #G6,Z,Q3 0.0
```

9. DATA TABLES

Data tables in LATDYN serve multiple functions. For example, they may be used for setting up a non-linear spring, or to define a motion trajectory of a gridpoint, or to specify an applied load. The way in which data tables have been made multi-purpose is through their links with user defined Q- and T-variables (and in higher dimensional tables with Q- and T-vectors).

A T-variable is a special type of LATDYN variable which is automatically created when a table is defined and which represents the value of the dependent variable at the present timestep. A Q-variable is a general purpose user variable defined in a SET command.

Before using data tables it is helpful to know how to create Q-variables, and to understand in what order various operations are performed to integrate the dynamic equations of motion. This sequence is shown in Chapter 13, Figures 13.1 and 13.2 .

TABLE INTERPOLATION

When defining a one-dimensional data table, the user gives a list of (x,y) pairs of values. If we term "x" the independent variable and "y" the dependent variable, then the x-values do not have to be equally spaced but they do have to be monotonically increasing (or decreasing).

In using tables it is often necessary to interpolate between the data points. That is, it is desireable for a user to be able to treat the data in the table as though it were a continuous function instead of a set of discrete values. LATDYN has two **general** ways to do this. A Q-variable (or TIME) may be used to represent the independent variable, and either,

- o a T-variable may be used to represent the dependent variable, or,
- o another Q-variable may be used to represent the dependent variable.

The first way is the simplest, but it has some limitations. When a table is created by the user (in a TABLE1 or TABLE2 command), LATDYN it automatically creates a "T-variable" with the same ID number as was given to the table. When this T-variable is used as a parameter in other commands (such as SDFC, APPFORCE, APPTORQUE, ROTACTUATOR, or LINACTUATOR), then LATDYN automatically interpolates the table each time it executes the command, using the interpolated value of the table for the T-variable. Thus, the T-variable may be viewed as a continuous function which is derived from discrete data points given in the table.

The T-variable corresponds to the dependent variable (y-values) given in the table. LATDYN knows how to interpolate the table because the user also specifies a spline interpolation function and a parameter to associate with the independent variable (the x-values), to interpolate the table. This parameter may be TIME or a Q-variable.

For example, suppose that data for a non-linear spring force is given in a TABLE1 command (say ID=T3). The force (y-values) depends on the distance between two gridpoints (x-values). A Q-variable (say Q6) is defined in a SET command as the distance between the two gridpoints and is also given as the independent variable in the TABLE1 command. The non-linear spring data can then be used as though it were a continuous function represented by the table variable "T3". For example, a non-linear spring may be defined in a LINACTUATOR command with the force represented by "T3".

There is one complicating factor to this flexible scheme of using tables. If calculations are not done in the right order you won't get the right answer! To illustrate this point let's look again at the example given above. At each timestep in this example it is necessary first that LATDYN computes the gridpoint positions corresponding to the current timestep, then we want to use these positions to compute "Q6" (the distance between the required gridpoints), then to use this value to interpolate in the table to get a current value of the spring force "T3", and finally to use this value of the spring force to solve Newton's Law for this timestep so as to get the acceleration state vector for the model.

If the explicit integrator were being used, then with reference to figure 13.1, it can be seen that "Q6" should be computed in user subroutines PRECALC and BGNSTEP. If the implicit integrator were being used, then with reference to figure 13.2, it can be seen that "Q6" should be computed in user subroutines PRECALC and INLOOP.

In general T-variables are the easy way to use the interpolated value from a data table in the LINACTUATOR, ROTACTUATOR, SDFC, MDFC, APPFORCE, and APPTORQUE commands, because they will automatically be evaluated each time LATDYN executes that command. Thus, the only thing that the user has to worry about is whether the correct value of the independent variable is current when the interpolation is performed. If the independent variable is TIME itself then it is always correct. When the independent variable is a Q-variable which is computed in subroutine BGNSTEP (for an explicit integrator) or in subroutine INLOOP (for an implicit integrator), then in general it will be correct.

For other purposes, T-variables may also be computed at a specific sequence point in both integrators as shown in figures 13.1 and 13.2, for those tables where the AUTOCALC(ON) option is specified. The reason for this option is the great variety of ways in which T-variables and Q-variables may be used together in calculations. For example, a T-variable may be used in the calculation of a Q-variable which may be used as the independent variable in the calculation of another T-variable. In cases like this, the order in which calculations are performed become very important.

The fully controllable way to ensure that calculations are performed in the correct order is to use Q-variables to represent both dependent and independent variables. Special "operators" are provided in LATDYN to enable a user to perform table interpolations. Since operators are called in the user's given sequence and may be mixed with Q-variable calculations, this makes it easy for a user to control the precise point at which interpolations are performed and it is even possible to perform iterative interpolations at a timestep. Operators to perform interpolations include TCALC1, TCALC2, TVCALC1, and TVCALC2.

Define a Table (which is a Function of a Single Independent Variable)

Data tables in LATDYN come with special facilities for interpolation, and for using interpolated values as T-variables or Q-variables.

When creating a data table it is necessary to specify the parameter to be used to give the value of the independent degree of freedom for carrying out these interpolations. The interpolated value is obtained by fitting a spline under tension to the curve specified as data points given by the user in the TABLE1 command.

Results of the interpolation are placed in a T-variable. A T-variable is a special type of LATDYN variable which is automatically created when a table is defined. The I.D. of the T-variable is the same as that of the table itself, that is, T1 for the table with an I.D. of "1".

TABLE1, TAB1, T-VAR1, TVAR1

```
TABLE1: Table I.D., ind.var., Spline Tension, OPTION for automatic evaluation,
        filename or set of data values (as follows)
        x1, y1     &
        x2, y2     &
        x3, y3     &
        x4, y4     &
        ___etc___
```

Table I.D. — Tk where k is an integer. Note: two tables may not have the same I.D. number

ind.var. — TIME or Q-variable or another T-variable. Defines how the data points in the following table are to be interpreted by the program.
TIME - All values of the independent variable in the following table are relative to the starting time of the analysis run as specified in the TIMESPAN command.
Q-var - All values of the independent variable in the following table are taken to be relative to the value of the Q-variable specified. When the interpolation is performed, the value of the specified Q-variable at that time is the value to be used for the interpolation.
T-var - All values of the independent variable in the following table are taken to be relative to the value of the T-variable specified. When the interpolation is performed, the value of the specified T-variable at that time is the value to be used for the interpolation.

Spline Tension — A spline under tension is used for interpolations. This parameter is a real number indicating the curviness desired. If the value is nearly zero (e.g. .001) the resulting curve is approximately a cubic spline. If the value is large (e.g. 50) the resulting curve is piecewise linear. If the value is exactly

zero, an exact cubic spline is produced. A standard value for spline tension is approximately 1.

OPTION for automatic evaluation - AUTOCALC, or omitted

AUTOCALC(ON) (default option) table will be interpolated at every timestep immediately after subroutine BGNSTEP (explicit integration) or after subroutine INLOOP (implicit integration) (see figures 13.1 and 13.2).

The results of the interpolation will be placed in the T-variable corresponding to this table.
Note: this process may be inefficient because multiple evaluations at a timestep may be performed because when a T-variable is used in one of the LINACTUATOR, ROTACTUATOR, SDFC, MDFC, APPFORCE, and APPTORQUE commands, then the table is always automatically interpolated when it is needed.
Note: the order of evaluation of table variables is in the numeric sequence of their I.D. numbers.
A T-variable is a special type of LATDYN variable which represents the interpolated value of the dependent variable at the present timestep. The interpolated value is obtained by fitting a spline under tension, to the curve specified by the data points given in this command.

AUTOCALC(OFF) automatic interpolations will be performed.only when a T-variable is used in a LINACTUATOR, ROTACTUATOR, SDFC, MDFC, APPFORCE, or APPTORQUE command.

The way in which a user can control the interpolation of table variables is described more fully in the section on user programming using the TCALC1 operator described below.

filename or set of data values data may be given on a separate file **or** as part of this command.

If data is given on a separate file then the filename is given here. Data on the file must contain an initial record giving the number of data sets.

Data sets in the commmand or on a file may be separated by a comma or spaces. Data within the command must contain the "&" continuation character at the end of each line. Data within a file needs no special characters.

x1, x2, x3, ... data for independent variable (real numbers)

y1, y2, y3, ... data for dependent variable (real numbers)

EXAMPLES

```
$          ID    ind.var.  tension   auto. eval.          data file
TABLE1:    T4    TIME      .8        AUTOCALC(OFF)        TEST7
TABLE1:    T6    Q10       1.0       AUTOCALC(ON)         React.DAT

$          ID    ind.var.  tension   auto. eval.          data
TABLE1:    T2    Q5        1.2       AUTOCALC(ON)         0.0   3.6   &
                                                          0.5   4.5   &
                                                          1.0   4.9   &
                                                          1.3   5.1   &
                                                          1.5   5.05
```

Define a Data Table (which is a Function of Two Independent Variables)

The only difference between 1-D and 2-D tables is that two independent variables are required to define the dependent variable. The command has the form,

TABLE2, TAB2, T-VAR2, TVAR2

```
TABLE2: Table I.D., ind.var1, ind.var2, Spline Tension, OPTION for automatic
        evaluation, filename or set of data values (as follows)
        x1, y1, z1      &
        x2, y2, z2      &
        x3, y3, z3      &
        x4, y4, z4      &
        ___etc___
```

NOTE: The TABLE2 Command is not presently implemented in the preprocessor but is implemented in the computational core.

Table Parameter I.D. — Tk where k is an integer. Note: two tables may not have the same I.D. number

ind.var1, ind.var2 — TIME or Q-variable or another T-variable. Defines how the data points in the following table are to be interpreted by the program.
TIME - All values of the independent variable in the following table are relative to the starting time of the analysis run as specified in the TIMESPAN command.
Q-var - All values of the independent variable in the following table are taken to be relative to the value of the Q-variable specified. When the interpolation is performed, the value of the specified Q-variable at that time is the value to be used for the interpolation.
T-var - All values of the independent variable in the following table are taken to be relative to the value of the T-variable specified. When the interpolation is performed, the value of the specified T-variable at that time is the value to be used for the interpolation.

Spline Tension — A spline under tension is used for interpolations. This parameter is a real number indicating the curviness desired. If the value is nearly zero (e.g. .001) the resulting curve is approximately a cubic spline. If the value is large (e.g. 50) the resulting curve is piecewise linear. If the value is exactly zero, an exact cubic spline is produced. A standard value for spline tension is approximately 1.A spline under tension is used for interpolations.

OPTION for automatic evaluation - AUTOCALC, or omitted

AUTOCALC(ON) — (default option) table will be interpolated at every timestep immediately after subroutine BGNSTEP (explicit integra-

tion) or after subroutine INLOOP (implicit integration) (see figure).

The results of the interpolation will be placed in the T-variable corresponding to this table.
Note: this process may be inefficient because multiple evaluations at a timestep may be performed because when a T-variable is used in one of the LINACTUATOR, ROTACTUATOR, SDFC, MDFC, APPFORCE, or APPTORQUE commands, then the table is always automatically interpolated when it is needed.
Note: the order of evaluation of table variables is in the numeric sequence of their I.D. numbers.
A T-variable is a special type of LATDYN variable which represents the interpolated value of the dependent variable at the present timestep. The interpolated value is obtained by fitting a spline under tension, to the curve specified by the data points given in this command.

AUTOCALC(OFF) automatic interpolations will be performed.only when a T-variable is used in a LINACTUATOR, ROTACTUATOR, SDFC, MDFC, APPFORCE, or APPTORQUE command.

The way in which a user can control the interpolation of table variables is described more fully in the section on user programming using the TCALC2 operator described below.

filename or set of data values data may be given on a separate file **or** as part of this command.

If data is given on a separate file then the filename is given here. Data on the file must contain an initial record giving the number of data sets.

Data sets in the commmand or on a file may be separated by a comma or spaces. Data within the command must contain the "&" continuation character at the end of each line. Data within a file needs no special characters.

x1, x2, x3, ... data for first independent variable (real numbers)

y1, y2, y3, ... data for second independent variable (real numbers)

z1, z2, z3, ... data for dependent variable (real numbers)

EXAMPLES

```
$          ID     ind.var.  tension   auto. eval.          data file
TABLE2:    T4     TIME      .8        AUTOCALC(OFF)        TEST7
TABLE2:    T6     Q10       1.0       AUTOCALC(ON)         React.DAT

$          ID     ind.var.  tension   auto. eval.                data
TABLE2:    T2     Q5        1.2       AUTOCALC(ON)         0.0   3.6   10.3  &
                                                            0.5   4.5   15.9  &
                                                            1.0   4.9   25.8  &
                                                            1.3   5.1   30.4  &
                                                            1.5   5.05  35.5
```

Interpolate and Differentiate a Data Table

The fully controllable way to use the interpolated value from a data table in one of the following commands by using Q-variables.instead of T-variables (Q-variables are described in Chapter 13).

Whereas T-variables are the easy way to use the interpolated value from a data table in the LINSPRING, LINDAMPER, ROTSPRING, ROTDAMPER, SDFC, MDFC, APPFORCE, and APPTORQUE commands, the more flexible way is to use Q-variables in conjunction with a TCALC operator.

When a TCALC operator is inserted in a user subroutine by an OP command and executed during a LATDYN run, it forces immediate interpolation of the specified table, and puts the interpolated value of the dependent variable into a Q-variable. A side benefit of using the TCALC operators is that it is also possible to extract derivatives, which are also placed in Q-variables. All these variables are then easily available for use both in the above commands, and in subsequent user program statements (SET and OP commands).

The TCALC operator for a Table Variable which is a Function of One Independent Variable is as follows,

TCALC1(itable, Qi, Qj, Qk)	OPERATOR to Interpolate and Differentiate a Data Table which is a Function of One Independent Variable.

itable	Table I.D. for which user has specified data in a TABLE1 command (integer value)
Qi	interpolated value of the table
Qj, Qk	computed values of first and second differentials respectively

TCALC2(itable, Qi, Qj, Qk, Ql, Qm, Qn)	OPERATOR toInterpolate and Differentiate a Data Table which is a Function of Two Independent Variables.

itable	Table Variable I.D. for which user has specified data in a TABLEVAR1 command (integer value)
Qi	interpolated value of the table
Qj, Qk	computed values of first differentials with respect to the two independent variables
Ql, Qm	computed values of second differentials with respect to the two independent variables
Qn	computed value of the cross differential with respect to the two independent variables

TABLE VECTORS

T-vectors are related to vector tables in just the same way that T-variables are related to ordinary data tables. Vector tables are defined as shown below.

NOTE: Table Vectors are not presently implemented in the preprocessor but are implemented in the computational core.

Define a Vector Table (which is a function of a Single Independent Variable)

TABLEVEC1, TABVEC1, TVECTOR, TVEC1

TABLEVEC1: vector table I.D., ind.var., Spline Tension, OPTION for automatic calculation, OPTION for table spec.

vector table I.D. — Table vector identifier k, where k is an integer

ind.var. — TIME or Q-variable or another T-variable. Defines how the data points in the following table are to be interpreted by the program.
TIME - All values of the independent variable in the following table are relative to the starting time of the analysis run as specified in the TIMESPAN command.
Q-var - All values of the independent variable in the following table are taken to be relative to the value of the Q-variable specified. When the interpolation is performed, the value of the specified Q-variable at that time is the value to be used for the interpolation.
T-var - All values of the independent variable in the following table are taken to be relative to the value of the T-variable specified. When the interpolation is performed, the value of the specified T-variable at that time is the value to be used for the interpolation.

Spline Tension — A spline under tension is used for interpolations. This parameter is a real number indicating the curviness desired. If the value is nearly zero (e.g. .001) the resulting curve is approximately a cubic spline. If the value is large (e.g. 50) the resulting curve is piecewise linear. If the value is exactly zero, an exact cubic spline is produced. A standard value for spline tension is approximately 1.A spline under tension is used for interpolations.

OPTION for automatic evaluation - AUTOCALC, or omitted

AUTOCALC(ON) — (default option) table will be interpolated at every timestep immediately after subroutine BGNSTEP (explicit integra-

tion) or after subroutine INLOOP (implicit integration) (see figure).

The results of the interpolation will be placed in the T-vector corresponding to this table.
Note: this process may be inefficient because multiple evaluations at a timestep may be performed because when a T-vector is used in one of the APPFORCE, or APPTORQUE commands, then the table is always automatically interpolated when it is needed.
Note: the order of evaluation of T-vectors is in the numeric sequence of their I.D. numbers.
A T-vector is a special type of LATDYN variable which represents the interpolated value of the dependent variables at the present timestep. The interpolated value is obtained by fitting a spline under tension, to the curve specified by the data points given in this command.

AUTOCALC(OFF) automatic interpolations will be performed.only when a T-vector is used in an APPFORCE, or APPTORQUE command.

The way in which a user can control the interpolation of table vectors is described more fully in the section on user programming using the TVCALC1 operator described below.

OPTION for table specification - RCOORD or SCOORD

RCOORD(t1, x1, y1, z1, t2, x2, y2, Z2, t3, x3, y3, z3, etc ----- Axesname for def)
- coordinates of vector will be given in rectangular coordinates in a specified coordinate system.

t1, t2, t3 ... values of independent variable

xi,yi,zi ... coordinates of dependent vector

Axesname for def. - refers to coordinate system defined in AXES command

SCOORD (t1, Mag1, Az1, El, t2, Mag2, Az2, El2, t3, Mag3, Az3, El3 etc----- Axesname for def) - coordinates of vector will be given in spherical coordinates in a specified coordinate system.

t1, t2, t3 ... values of independent variable

Magi,Azi,Eli - coordinates of dependent vector

Axesname for def. - refers to coordinate system defined in AXES command

---

Interpolate and Differentiate a Vector Table

The TVCALC operators for vector tables described in this section are analogous to the TCALC operators described earlier for ordinary tables. The TCALC operator for a Table Vector which is a Function of One Independent Variable is as follows,

TVCALC1(itable, Q^i, Q^j, Q^k)

OPERATOR toInterpolate and Differentiate a Vector Table which is a Function of One Independent Variable.

itable vector table I.D. for which user has specified data in a TABLEVECT1 command

Q^i, Q^j, Q^k computed values of the table and the first and second differentials respectively

10. APPLIED LOADS AND CONTROLS

One of LATDYN's strengths is its facility for allowing the user to program a wide variety of loads. These loads may range from a simple constant force or torque to a complete control system for a spacecraft. This capability has been provided through a rich variety of functions and operators which allow the user access to any of the dynamic states of the model. By using Q-variables and Q-vectors, calculations may be performed on "measurements" of dynamic states and fed-back as control forces. That is (in controls terminology) Q-vectors and Q-variables act as sensors and state estimators.

Applied loads fall into three general groups: (1) Externally applied forces or torques, where the reaction of the applied force or torque is grounded to an entity outside the system, (2) Actuators, where the action and reaction are self contained within the system, and (3) Force fields, such as gravity.

APPLIED FORCES AND TORQUES

Externally applied forces in LATDYN have magnitude and direction. Forces may be constant or may vary with time in a prescribed way, or may be the result of feedback from a control system.

Externally applied torques have magnitude and are applied about a torque axis vector. Both the magnitude and direction of the torque may be constant or may vary with time in a prescribed way, or may be the result of feedback from a control system.

Define a Force and Apply it to a Grid Point or Set of Grid Points

APPFORCE, AFORCE

APPFORCE: Forcename, OPTION for force spec., OPTION for duration, list of grid points where force is to be applied ? Cj

Forcename — alphanumeric identifier for the force which is being defined by this command. (Note: two APPFORCE commands may not have the same name.)

OPTION for force specification - MDV, MSDV, or MVDV

MDV(Vector I.D.) - magnitude & direction given by a simple vector.

Vector I.D. — R^k, T^k, Q^k, I^axesname, J^axesname, or K^axesname

MSDV(Scalar Mag., Vector Dir.) - magnitude is given by a scalar. direction is given by a vector.

Scaler Mag. — real number, Q-Variable or T-Variable

Vector Dir. R^k, T^k, Q^k, I^axesname, J^axesname, or K^axesname

MVDV (Vector Mag., Vector Dir.) - magnitude is the magnitude of the first vector, direction is the direction of the second vector.

OPTION for force duration - TIME or omitted

TIME(t1, t2) - Start time and stop time for force application

t1,t2 start time, stop time

gridpoints list of grid points where force is to be applied.

Cj optional condition label (Cj or CLj, j is an integer from 1 to 99)

When an APPFORCE command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
APPFORCE: Axial_Thruster MDV(R^1)      ,,      #G6
APPFORCE: F1   MSDV(5.67,R^16)  TIME(0.0, 2.5)  #G1,#G2,#G3,#G4  ?C1
APPFORCE: F2   MVDV(Q^4,R^4)     ,,     #C5G2
```

Define a Torque and Apply it to a Grid Point or Set of Grid Points

APPTORQUE, APPTORQ, ATORQUE, ATORQ

APPTORQUE: Torque Name, OPTION for Torque Spec, OPTION for torque duration, list of grid points where torque is to be applied ? Cj

Torquename alphanumeric identifier for the torque which is being defined by this command. (Note: two APPTORQUE commands may not have the same name.)

OPTION for Torque specification - MDV, MSDV, or MVDV

MDV(Vector I.D.) magnitude and direction given by a single vector. Direction of vector is the torque axis. Magnitude is the magnitude of the torque.

Vector I.D. R^k, T^k, Q^k, I^axesname, J^axesname, or K^axesname

MSDV(Scalar Mag., Vector Dir.) - magnitude is given by a scalar, direction is given by a vector.

Scalar Mag. Real number, Q-Variable or T-Variable

Vector Dir. R^k, T^k, Q^k, I^axesname, J^axesname, or K^axesname

MVDV(Vector Mag, Vector Dir) - magnitude is the magnitude of the first vector, direction is the direction of the second vector.

OPTION for torque duration - TIME or omitted

TIME (t1, t2) Start time and stop time for torque application

t1,t2 start time, stop time

gridpoints	list of grid points where force is to be applied.
Cj	optional condition label
	When an APPTORQUE command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
APPTORQUE:  Axial_Torque  MDV(R^1)     ,,     #G6
APPTORQUE:  T1   MSDV(5.67,R^16)  TIME(0.0, 2.5)  #G1,#G2,#G3,#G4  ?C1
APPTORQUE:  T2   MVDV(Q^4,R^4)     ,,     #C5G2
```

ACTUATORS

Actuators in LATDYN always act between two points on the structural model. A lineal actuator acts in a straight line between two gridpoints. A rotational actuator acts about a hinge axis, between a gridpoint and a hingepoint, between two gridpoints, or between two hingepoints.

Define a Lineal Actuator

LINACTUATOR, LACTUATOR, LINACT, LACT

```
LINACTUATOR: Actuator name, grid a#, grid b#, force magnitude, OPTION for
        duration  ? Cj
```

actuator name	alphanumeric actuator name. (Note: two LINACTUATOR commands may not have the same name.)
grid a#	grid point identifier for one end of actuator (#Gj or #CiGj or #GjJk or #CiGjJk, where i = component number, j = gridpoint number, k= jointpoint number).
grid b#	grid point identifier for other end of actuator (#Gj or #CiGj or #GjJk or #CiGjJk, where i = component number, j = gridpoint number, k= offset point number).
force magnitude	may be a real number, a Q-variable, or a T-variable
	Note: A positive value of actuator force means that the grid points are being pushed apart.
OPTION for force duration	TIME or omitted
TIME(t1, t2)	Start time and stop time for force application
t1,t2	start time, stop time
Cj	optional condition label

When a LINACTUATOR command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
LINACTUATOR:  Piston_1   #G1 #G2   ,,    Q4
LINACTUATOR:  Actuator2   #G6 #G7   T1   TIME(0.0, 1.5)   ? C3
```

Define a Rotational Actuator

ROTACTUATOR, RACTUATOR, ROTACT, RACT

ROTACTUATOR: actuator name, OPTION for connection of ends, Mag, OPTION for duration ? Cj

actuator name — alphanumeric name for rotational actuator. (Note: two ROTACTUATOR commands may not have the same name.)

OPTION for connection of ends - UX, CX, BX, or GX

UX(hingepoint #) — uniaxial connection for connection between a grid point and a hingepoint attached to it.

CX(hingepoint a#, hingepoint b#) - co-axial connection for connection between two coaxial hinge degrees of freedom attached to the same grid point.

BX(hingepoint a#, hingepoint b#) - bi-axial connection for connection between any two hinge degrees of freedom attached to the same gridpoint, or between two different gridpoints on the same rigid body.

GX(hinge constraint name) - general connection for a rotational damper about a hinge axis between two gridpoints, where the hinge is defined by constraints. The hinge constraint name is defined by a HINGEJOINT command.

Magnitude — magnitude of the torque applied by the actuator. May be a real number or a T-variable or a Q-variable.

Note: A positive value of the actuator torque means that the torque on the second connection point is positive (clockwise) about the hinge axis. (Obviously the torque on the first connection point is then negative).

OPTION for torque duration - TIME or omitted

TIME(t1, t2) - start time and stop time for torque application

t1,t2 — start time, stop time

Cj — optional condition label (Cj or CLj, j is an integer from 1 to 99)

When an ROTACTUATOR command is qualified by a condition label, it means that the command is active when the condition is true and inactive when it is false.

EXAMPLES

```
ROTACTUATOR:  Torq_Wheel_1   UX(#G1H1)   „    Q4
ROTACTUATOR:  Torq_Wheel_2   CX(#G1H1,#G1H2)   „    Q4
ROTACTUATOR:  Rot_Act2   GX(Hinge_Joint3)   T1   TIME(0.0, 1.5)   ? C3
```

Gravity Force Field

Two versions of a gravity force field have been built into LATDYN. One version gives a constant gravity field and is intended for use in modeling terrestrial structures. The other gives a gravity field which varies with distance from the center of the earth, and is intended for use in modeling structures in orbit.

GRAVITY, GRAV

GRAVITY: OPTION for specifying gravity field

OPTION for specifying gravity field - CONSTANT or GRADIENT

CONSTANT(g, vector for dirn. of gravity field) - to specify a constant field

g	the acceleration due to gravity
vector for dirn. of gravity field	- reference vector I.D. (R^i, I^axesname, J^axesname, K^axesname)

GRADIENT(g, x, y, z)	to specify a gravity gradient field
g	the acceleration due to gravity at the origin of the global coordinate system
x, y, z	the location of the center of the earth in global coordinates

EXAMPLES

```
GRAVITY:   CONSTANT(32.2, Z^GLO)
GRAVITY:   GRADIENT(32.2, 0.0,0.0,-1.0E5)
```

11. INITIALIZATION OF DYNAMIC VARIABLES

INITIAL VELOCITIES

When a structure is not initially at rest, it is necessary to define these initial velocities. This is often not a simple task because it is necessary for the dynamic states throughout the structure to be consistant with each other. Often the easiest way to achieve consistant initial dynamic states is to run LATDYN starting the structure from rest, and then to use the dynamic states derived at a certain point in the motion as the starting point for a new analysis.

To Give a Single Gridpoint an Initial Velocity

The default value for initial velocity is zero. The form of the command to give a gridpoint a non-zero initial velocity is:

VELOCITY, VELOC, VEL

VEL: grid#, vx, vy, vz, wx, wy, wz, axesname

grid#	identifies the gridpoint to which the velocity will be applied (#Gj or #CiGj) - jointpoints are not allowed.
vx, vy, vz	(x, y, z) components of gridpoint translational velocity in the named axes system.
wx, wy, wz	(x, y, z) components of gridpoint rotational velocity in the named axes system.
axesname	refers to a coordinate system defined by the axes command. The translational and rotational velocity components are given relative to this coordinate system. If this parameter is omitted or left blank, then the GLOBAL system is assumed.
	Note - for the purpose of this command the coordinate system is always assumed to be stationary.

EXAMPLES

VEL: #G2, 3.6,0,0 0,6.7,0 GLO
VEL: #C6G4 0,0,0 12.2, 12.2,0 Sys1

To Give a Hingepoint an Initial Rotational Velocity about its Hinge Axis Relative to the Gridpoint.

VELHINGE, VELHNG, VHINGE, VHNG, VELH

VELHNG: hingepoint#, wh

hingepoint#	identifies the hingepoint to which the velocity will be applied (#GjHk or #CiGjHk).
wh	rotational velocity of hingepoint about the hinge axis relative to the gridpoint.

EXAMPLES

VELHNG: #G2H1, 3.6
VELHNG: #C6G4H2 3.14159

12. SETTING UP A TRANSIENT ANALYSIS

To perform a transient analysis it is necessary to provide a title for the data case and to select a number of options to control the analysis itself. These options include an explicit or implicit integrator, integration timestep and timespan, inclusion of gyroscopic terms in the equations of motion, and mass matrix updating options and constraint stabilization methods.

It is also possible to create moving frames of reference, and to specify periodic calculation of instantaneous linearized frequencies and mode shapes.

LATDYN has two major forms of output, a print file and a plot file. The print file is intended to be used primarily for debugging and verification of a model data case. The plot file is intended for use with the LATDYN Postprocessor, an interactive program to plot data from a LATDYN run, and to provide facilities for managing and comparing the output data of several comparison runs. The Postprocessor also contains facilities for exporting LATDYN output data to other programs, and for importing the output from other programs for comparison with LATDYN output.

This section of the manual contains a description of commands to set up a LATDYN transient analysis. **The four required commands are all in this section. They are TITLE, INTEGRATOR, TIMESTEP, and TIMESPAN.**

TITLES

There are two forms of title in LATDYN, a short form and a long form. The short form is called TITLE and the long form is called NOTES. The short form may not exceed 80 characters in length and must be present on all data cases.

Both TITLE and NOTES data are different from comments which are written into the user command file. A title defined with the TITLE command is passed from the command file to LATDYN by the Preprocessor and subsequently to the Postprocessor by LATDYN. Thus it is always present on all intermediate data files and may be used to identify that data or to title plots. NOTES data is treated similarily except that it may not be used to title plots.

Defining a Title for a Data Case

The TITLE command is a required command and each data case may contain only one title. The form of the command is:

TITLE

TITLE: data case title

data case title one line of up to 80 characters

EXAMPLES
TITLE: Space Crane Transient Analysis (Implicit Integrator)

Writing Notes on the Data Case

NOTES, NOTE, SUBTITLE, STITLE

NOTES: data case notes

data case notes	list of any number of lines of up to 80 characters per line

EXAMPLES
NOTES: This is a preliminary run to compare motion over a short timespan
" with data generated from a rigid analysis. No pickup of a body is included.

ANALYSIS CONTROL COMMANDS

Integrator Type Selection

This command allows a user to control the type of algorithm which is to be used for time integration:

INTEGRATION, INTEGR, INTEG

INTEG: OPTION for integrator type, OPTION for automatic step adjustment

OPTION for integrator type - EXPLICIT or IMPLICIT

EXPLICIT(alpha)	to select explicit time integration
alpha	alpha value for updating velocity (0 <= Alpha <= 1, default =.5)

IMPLICIT(alpha, beta, conv.criteria, epsrel, epsabs) - to select implicit integration

alpha	alpha value for updating velocity (0 <= Alpha <= 1, default =.5)
beta	beta value for updating the displacement (only for translational degrees of freedom (0 <= Beta <= 5, default =.25)
conv. criteria	- E, F, or A (E for energy tolerence, F for force tolerence, A for acceleration tolerence) (default = A)

epsrel	relative convergence criterion (real value >= 0, default = .001). Note: if epsrel=0 then the relative convergence is not used.
epsabs	absolute convergence criterion (real value >= 0, default=.001). Note: if epsabs=0 then the absolute convergence is not used.

Note: Both epsrel and epsabs may not be equal to zero.

Note: If solution changes a great del in magnitude during the integration and you wish to see this change then a relative epsilon should be used. If the solution does not change a great deal, or if you do not wish to see this change then an absolute epsilon should be used. In general a mixed criteria is probably the best and safest choice. For solutions large in magnitude it is essentially relative error, and for solutions small in magnitude it is essentially absolute error.

OPTION for automatic step adjustment - AUTOSTEP

AUTOSTEP(OFF)	automatic step adjustment off (default)
AUTOSTEP(ON)	automatic step adjustment on

Notes on Time Integration:

o Explicit integration is only stable if the user specified timestep is small enough. This stability limit is governed by the highest natural frequency that the user has built into the model. That is, the maximum timestep for which the integration will be stable is one-half of the period of the highest natural frequency in the model. When the timestep exceeds this value a very rapid divergence occurs and within a few timesteps the program aborts.
Because explicit integration usually requires a very small timestep for stability, the results of the integration may be viewed with a high level of confidence.

oImplicit integration is stable over much larger timesteps than explicit integration, but the computer time taken for each timestep is longer. Selection of the most computationally efficient algorithm will depend on the details of the problem being solved.
Also, implicit integration tends to smooth rapid variations in the dynamic solution if too large a timestep is specified. This question about solution accuracy with implicit integration may require convergence studies to give the required level of confidence in a solution to a particular problem.

Integration Time Step

This is a required command which supplies the time integration step. It is a singular command of which multiple versions are allowed. A singular command in LATDYN means that although multiple versions of the command may be specified in the user command file, only one version of the command may be active at any time in the transient analysis. That is, all but one version of the command must be qualified by a condition. Here the order of the different versions of the commands in the input file is important. LATDYN chooses the last one whose condition is .TRUE. to be the active version af the command.

TIMESTEP, TIMSTEP, TSTEP

TIMESTEP: delt ? Cj

delt	time step increment
Cj	refers to the user defined condition with label "j"

EXAMPLES

In the following example, all three statements appear together to specif the timestep for one run. If neither C1 nor C2 are true then the timestep is .01. If C2 is true then the timestep is .0001. If C2 is false and C1 is true then the timestep is .001.

```
TIMESTEP:  .01
    "      .001   ? C1
    "      .0001  ? C2
```

Solution Time Span Specification

This is a required command which supplies the starting and terminating times of the transient response computations. Data takes the form,

TIMESPAN, TIMSPAN, TSPAN

TIMESPAN: t1, t2

t1	starting time for the transient response analysis
t2	end time for transient response analysis

EXAMPLES

```
TIMESPAN:  0.0   3.0
TIMESPAN:  0.0   .2E-3
```

Mass Matrix Update Interval

This is an optional command used to designate updating of the system mass matrix. It is a singular command of which multiple versions are allowed. A singular command in LATDYN means that although multiple versions of the command may be specified in the user command file, only one version of the command may be active at any time in the transient analysis. That is, all but one version of the command must be qualified by a condition. Here the order of the different versions of the commands in the input file is important. LATDYN chooses the last one whose condition is .TRUE. to be the active version af the command.

If no INCMASS command is given, then the mass matrix is updated at every timestep. For the implicit case the mass matrix from the previous timestep is used, and no updates are performed inside the iteration loop.

INCMASS, INCMAS, INC

INCMASS: inc ? Cj

inc	this parameter has two possible meanings depending on whether the integration is explicit or implicit, Explicit Integration, inc = number of timesteps between mass matrix updating Implicit Integration,

inc = number of first iterations within a timestep for which mass matrix will be updated. (The mass matrix is always updated at every timestep.)

Cj refers to the user defined condition with label "j"

EXAMPLES

In the following example, all three statements appear together to specify the mass matrix update interval for one run. If neither C1 nor C2 are true then the interval is 1. If C2 is true then the interval is 100. If C2 is false and C1 is true then the interval is 10.

```
INCMASS:  1
    "     10   ? C1
    "     100  ? C2
```

Inclusion of Gyroscopic Terms in the Equations of Motion

This command enables the user to specify whether to include quadratic velocity dependent gyroscopic terms in the equations of motion. Gyroscopic terms are especially important when high rotational velocities are involved, but their effect is negligable for low rotational velocities. The default is ON.

CHKGYR, CHECKGYRO, CHKGYRO

CHKGYR: ON or OFF

ON gyroscopic terms will be included
OFF gyroscopic terms will be omitted

EXAMPLES

```
CHKGYR:  ON
```

To Control the Application of Baumgarte's Constraint Stabilization Equation

Constraint stabilization is often useful to prevent the likelyhood of constraint violations from taking place after integrating for a large number of timesteps. If the ABSTB command is omitted then Baumgarte constraint stabilization will be **off**.

ABSTB

ABSTB: OPTION to turn constraint stabilization on or off

OPTION to turn constraint stabilization on or off - (alpha & beta) or OFF

(alpha & beta) turns constraint stabilization on (default is off)
alpha - value of alpha in Baumgarte's constraint equation (alpha > 0, default value = 5)
beta - value of beta in Baumgarte's constraint equation (beta must not be = 0, default value = 5)

OFF turns constraint stabilization off. No parameters.

EXAMPLES
ABSTB: .3 .35
ABSTB: OFF

SPECIAL FACILITIES TO BE USED DURING THE ANALYSIS

Creating a Moving Reference - Point, Vector, or Coordinate Axes System

Fixed reference points, vectors, and coordinates axes are primarily useful during the setup phase of model development. During a transient dynamic analysis involving large angular rotations however, it is often necessary to use moving reference entities or frames of reference. Moving references can be set up by attaching a previously defined reference entity to an actual physical part of the structure being modeled.

ATTACH

ATTACH: OPTION for reference entity to be attached, gridpoint# ? Cj

OPTION for reference entity to be attached - AXES, REFPT, REFVECT

AXES(list of axesnames) - the coordinate systems named in the list will be fixed to the specified gridpoint.

list of axesnames - identifiers of coordinate systems defined in the AXES command

REFPT(list of reference points) - the reference points named in the list will be fixed to the specified gridpoint.

list of reference points - identifiers of reference points (#Ri)

REFVECT(list of reference vectors) - the reference vectors named in the list will be fixed to the specified gridpoint.

list of reference vectors - identifiers of reference vectors (R^i)

gridpoint# gridpoint or jointpoint identifier(#Gj, #CiGj, #GjJk, or #CiGjJk)

Cj optional condition label

NOTE: A moving reference entity - that is, a reference point, a reference vector, or a coordinate system - may be attached to only one gridpoint or jointpoint at any time during the analysis. However, because it may be desireable to **change** the point to which reference entities are attached, during the analysis, this command may be qualified by a condition (Cj) and multiple versions may be given by the user. Since only one version of the command (with a particular reference entity list) may be active at a particular time it is in this sense a "singular" command for each set of specified reference entities, but the first version of the command may have a condition label attached to it since it is also possible that if no version of the command is active then the specified reference entities will simply not move. That is, they will be fixed in the global system.

EXAMPLES

If a particular reference entity appears in a list of similar entities in one ATTACH command, then that entity **may not** appear in any other ATTACH command unless it appears in an identical list of reference entities. For example, the following commands are legal:

```
ATTACH: REFPT(#R1,#R2,#R3,#R4), #C"Sprocket"G1
     "   REFPT(#R1,#R2,#R3,#R4), #C"Slider"G4   ? C5
```

Here the reference points (#R1 through #R4) are attached to gridpoint 1 on the component "Sprocket". They will retain their same relative positions relative to this point throughout the analysis until condition 5 becomes true, then their point of attachment will be switched to gridpoint 4 on component "Slider", and will henceforth retain their same relative positions to this point. However the following commands,

```
ATTACH:  REFPT(#R1,#R2,#R3,#R4), #C"Sprocket"G1
     "   REFPT(#R1), #C"Slider"G4   ? C5
```

are not legal because reference point #R1 appears in different lists in two ATTACH commands. To be legal these commands should be written in the form:

```
ATTACH:   REFPT(#R2,#R3,#R4), #C"Sprocket"G1
$
ATTACH:   REFPT(#R1), #C"Sprocket"G1
     "    REFPT(#R1), #C"Slider"G4   ? C5
```

To Specify the Calculation of Instantaneous Linearized Frequencies and Modes

The FREQ command is used to specify the calculation of linearized modes and frequencies of the system at particular times in the transient dynamic analysis. This command may be qualified by a condition (Cj) and multiple versions may also be given by the user. However only one version of the command may be active at a particular time and in this sense it is a "singular" command, but the first version of the command may have a condition label attached to it since it is also possible that no FREQ command may be active.

FREQUENCY, FREQ

FREQ: OPTION for specifying calculation intervals, H or L, m, n ? Cj

OPTION for specifying calculation intervals - TIME or STEP

TIME(delta t)	calculation specified at time intervals
delta t	time interval
STEP(delta k)	calculation specified as a number of integration timesteps
delta k	step interval
H or L	specifies whether the highest (H) or the lowest (L) frequencies and modes are to be calculated.
m	specifies how many frequencies are to be calculated
n	specifies how many modeshapes are to be calculated (must be <= m)
Cj	optional conditional label

EXAMPLES
FREQ: TIME(.5) L 10 10
FREQ: STEP(1000) H 5 5

OUTPUT CONTROL COMMANDS

To Specify Periodic Printing of Results from the Transient Dynamic Analysis

The print file is intended to be used primarily for debugging and checking details of a model data case. The PRINT command which generates it is a "singular" command of which multiple versions are allowed. That is, only one PRINT command may be active at any time. Here the order of the different versions of the commands in the input file is important. When several conditional PRINT commands are given, then the one chosen is the last one whose condition is TRUE. This logical structure enables a user to adjust printing based on events which occur during the dynamic analysis.

PRINT,PRI

PRINT: OPTION for specifying print intervals ? C

OPTION for interval specification - STEP or TIME

STEP(k1, axesname, k2, LG, k3, k4, k5) - Specification for print period is given in terms of a number of timesteps.

k1	an integer specifying the printing of dynamic states (displacements, velocities, and accelerations of gridpoints and jointpoints) every k1 timesteps. If k1 is zero or omitted then no dynamic states are printed.
axesname	identifies the user defined coordinate system with respect to which components are to be printed. Note that the dynamic states are **NOT** expressed relative to the motion of this coordinate system.
k2	an integer specifying the printing of the internal forces at each end of a flexible beam member every k2 timesteps. If k2 is zero or omitted then no internal forces are printed.
LG	is a letter (either L for local or G for global) specifying that printing of internal member forces are to be in the local or global systems. (The local system rotates with each member.)

k3 is an integer specifying the printing of the massmatrix every k3 timesteps. If k3 is zero or omitted then the mass matrix is not printed.

k4 is an integer specifying printing of the stiffness matrix every k4 timesteps. If k4 is zero or omitted then the stiffness matrix is not printed.

k5 is an integer specifying printing of the gyroscopic matrix and gyroscopic vector every k5 timesteps. If k5 is zero or omitted then the gyroscopic matrix and vector are not printed.

TIME(t1, axesname, t2, LG, t3, t4, t5) - Specification for print period is given in terms of a time interval.

t1 an integer specifying the printing of dynamic states (displacements, velocities, and accelerations of gridpoints and jointpoints) every t1 seconds. If t1 is zero or omitted then no dynamic states are printed.

axesname identifies the user defined coordinate system with respect to which components are to be printed. Note that the dynamic states are **NOT** expressed relative to the motion of this coordinate system.

t2 an integer specifying the printing of the internal forces at each end of a flexible beam member every t2 seconds. If t2 is zero or omitted then no internal forces are printed.

LG is a letter (either L for local or G for global) specifying that printing of internal member forces are to be in the local or global systems. (The local system rotates with each member.)

t3 is an integer specifying the printing of the massmatrix every t3 seconds. If t3 is zero or omitted then the mass matrix is not printed.

t4 is an integer specifying printing of the stiffness matrix every t4 seconds. If t4 is zero or omitted then the stiffness matrix is not printed.

t5 is an integer specifying printing of the gyroscopic vector every t5 seconds. If t5 is zero or omitted then the gyroscopic vector is not printed.

Cj is an optional reference to the j'th user defined condition.

When this parameter is present, then the command is made conditional. That is, the command is active only when the condition is TRUE. Since this is a singular command of which multiple versions are allowed, the one chosen when multiple versions of the command are present is the last one whose condition is TRUE. This logical structure enables a user to adjust printing based on events which occur during the dynamic analysis.

EXAMPLES

```
PRINT:   STEP(1000   GLO   1000   L)
PRINT:   TIME( .1    GLO   .1   G   1    1    1)
```

To Specify Periodic Output of Results from the Transient Dynamic Analysis to a Postprocessor Plot File

The plot file is intended for use with the LATDYN Postprocessor, a interactive program to plot data from a LATDYN run. The Postprocessor also provides facilities for managing and comparing the output data of several LATDYN runs, and it contains facilities for exporting LATDYN output data to other programs and for importing the output from other programs to compare with LATDYN output.

The PLOT Command is a "singular" command of which multiple versions are allowed. That is, only one PLOT command may be active at any time. When several conditional PLOT commands are given, then the one chosen is the last one whose condition is TRUE. This logical structure enables a user to adjust plotting based on events which occur during the dynamic analysis.

PLOT

PLOT: OPTION for specifying plot intervals ? Cj

OPTION for specifying plot intervals - STEP or TIME

STEP(k1)	Specification of plot period is given in terms of a number of timesteps.
k1	an integer specifying the plot file output of dynamic states (displacements, velocities, and accelerations of gridpoints and jointpoints) every k1 timesteps.
TIME(t1)	Specification of plot period is given in terms of a time interval
t1	an integer specifying the plot file output of dynamic states (displacements, velocities, and accelerations of gridpoints and jointpoints) every t1 seconds.
Cj	is an optional reference to the j'th user defined condition.

When this parameter is present, then the command is made conditional. That is, the command is active only when the condition is TRUE. Since this is a singular command of which multiple versions are allowed, the command chosen when multiple versions are present is the last one whose condition is TRUE. This logical structure enables a user to adjust plotting based on events which occur during the dynamic analysis.

EXAMPLES

```
PLOT: 10
  "    1  ? C2
```

13. A SYMBOLIC LANGUAGE FOR TRANSIENT ANALYSIS

There are many situations in control/structure dynamic analysis where some form of symbolic language is needed to program the model. For example,using a symbolic language to define mathematical operations is very convenient in modeling control systems for moving robot arm, or where the structure is changing its configuration throughout the controlled motion such as during the deployment sequence of a spacecraft component.

The alternative to a symbolic language is requiring the user to write subroutines. This process can be difficult and error prone in the extreme. In LATDYN, a different approach has been adopted. The Preprocessor writes the subroutines for you!

LATDYN providing symbolic capabilities for modeling control systems which are integrated with the structural dynamic analysis itself. Its command language contains syntactical structures which perform symbolic operations and which are also interfaced directly with the finite element structural model. Thus, when the dynamic equations representing the structural model are integrated, user computations, perhaps representing the control system, are integrated along with them as a coupled system.

SYMBOLIC PROGRAMMING CONCEPTS

LATDYN contains a special class of commands which allow a user to program conditions, calculations, function evaluations, or more complex operations which are to be performed during the transient analysis.These commands are SET, OP, and CLj.

Conditions are logical expressions written by the user in FORTRAN-like syntax. At any time step, a condition may be true or false. In LATDYN, each user-defined condition must be given a label by the user when it is defined using the "CLj" command. Many commands in LATDYN may be made conditionally dependent on a specific user-defined condition by attaching a question mark "?", followed by a condition label Cj to the end of the command. If the logical statement associated with the condition label is true then the command becomes active. If the logical statement associated with the condition label is false, then the command becomes inactive.

The basic commands to program calculations to be performed at each timestep are SET and OP. The SET Command enables a user to program arithmetic calculations and function evaluations which will be stored in user defined variables called **Q-Variables**. The OP command enables a user to program more complex operations by calling LATDYN "operators". For example, a "vector operator" may perform computations which result in a vector. The resultant vector will be stored in a special user defined vector called a **Q-Vector.**

Since both Q-Variables and Q-Vectors may be used as parameters in a variety of commands, this capability allows a user to *program* these commands with variables which depend on the results of the transient analysis.

Creating User Subroutines

Because SET, OP, and CLj commands are a special class of commands which allow a user freedom to program a transient analysis, it is necessary that a user be able to control the order of the calculations specified in these commands in the context of LATDYN's time integrators.

The default order of these calculations is set so that they will be performed at the beginning of a transient analysis, and at the end of each timestep thereafter. In addition, if a user is performing the analysis using an implicit integrator, then the calculations will also be performed within the implicit iteration loop.

The PROGRAM command tells the LATDYN Preprocessor in which subroutines to place the user's programmed calculations. In this way an advanced user may override the defaults described above. All SET, OP, and CLj commands following this command will be placed in the named user subroutines in the same order in which they appear in the user's command file. A subsequent PROGRAM command cancels all settings from a previous PROGRAM command.

PROGRAM, PROG, PRO

PROGRAM: subroutine name, subroutine name, ...

subroutine name	PRECALC, BGNSTEP, ENDSTEP, INLOOP or ALL, or NONE
	Each subroutine name has a specific meaning in a LATDYN transient analysis sequence as shown in figures 13.1 and 13.2.
PRECALC	subroutine to initialize variables. All user program statements following this command will be evaluated before the start of time integration.
BGNSTEP	subroutine to perform user operations, and evaluate user Q-variables and conditions at the start of a timestep.
ENDSTEP	subroutine to perform user operations, and evaluate user Q-variables and conditions at the end of a timestep.
INLOOP	subroutine to perform user operations, and evaluate user Q-variables and conditions at the end of an implicit iteration loop within the timestep integration.
ALL	all of the above
NONE	none of the above

EXAMPLES

```
PROGRAM: PRECALC
PROGRAM: PRECALC BGNSTEP INLOOP
PROGRAM: ENDSTEP
```

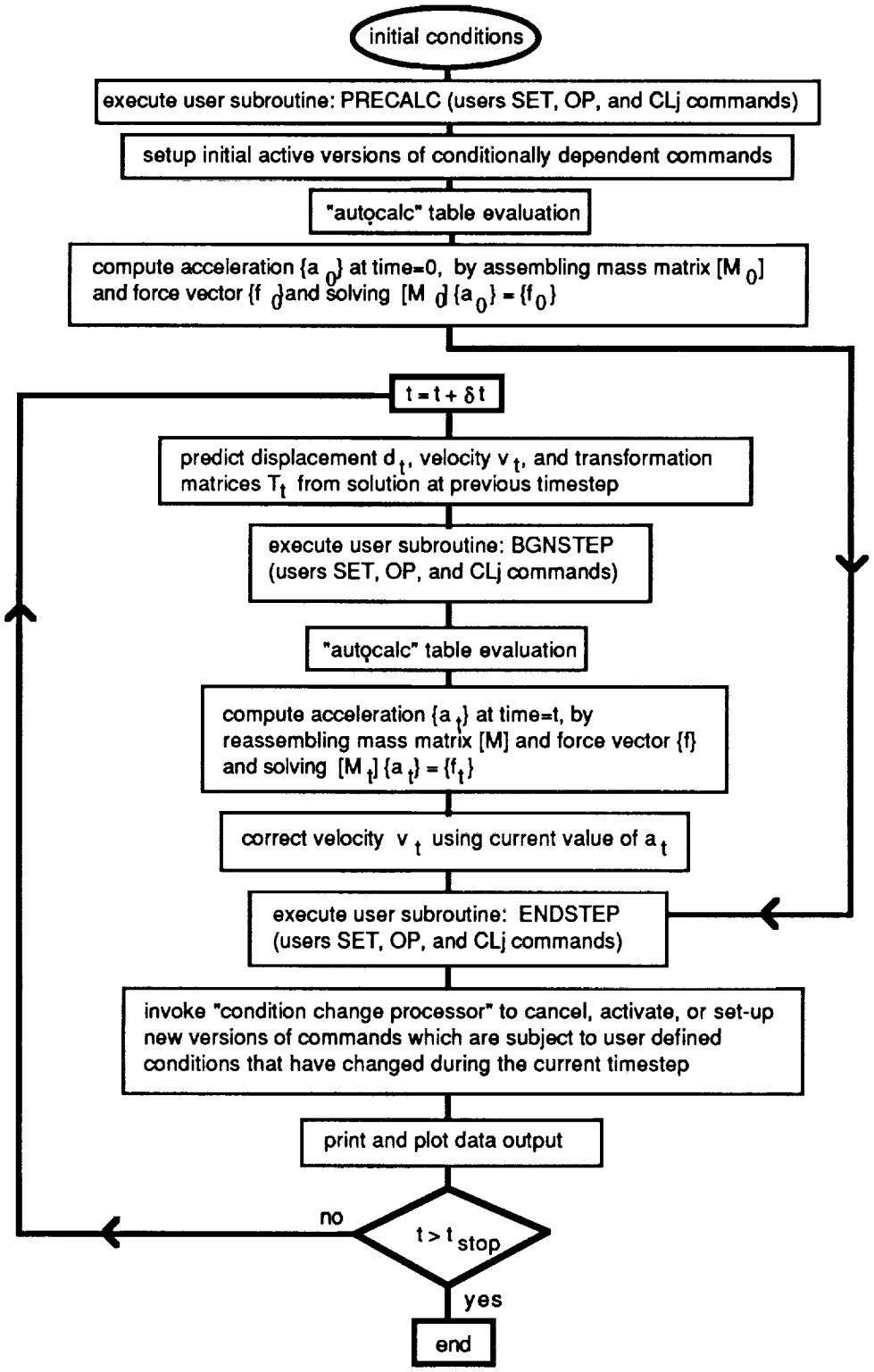


Figure 13.1 Flow Chart for Explicit Integrator

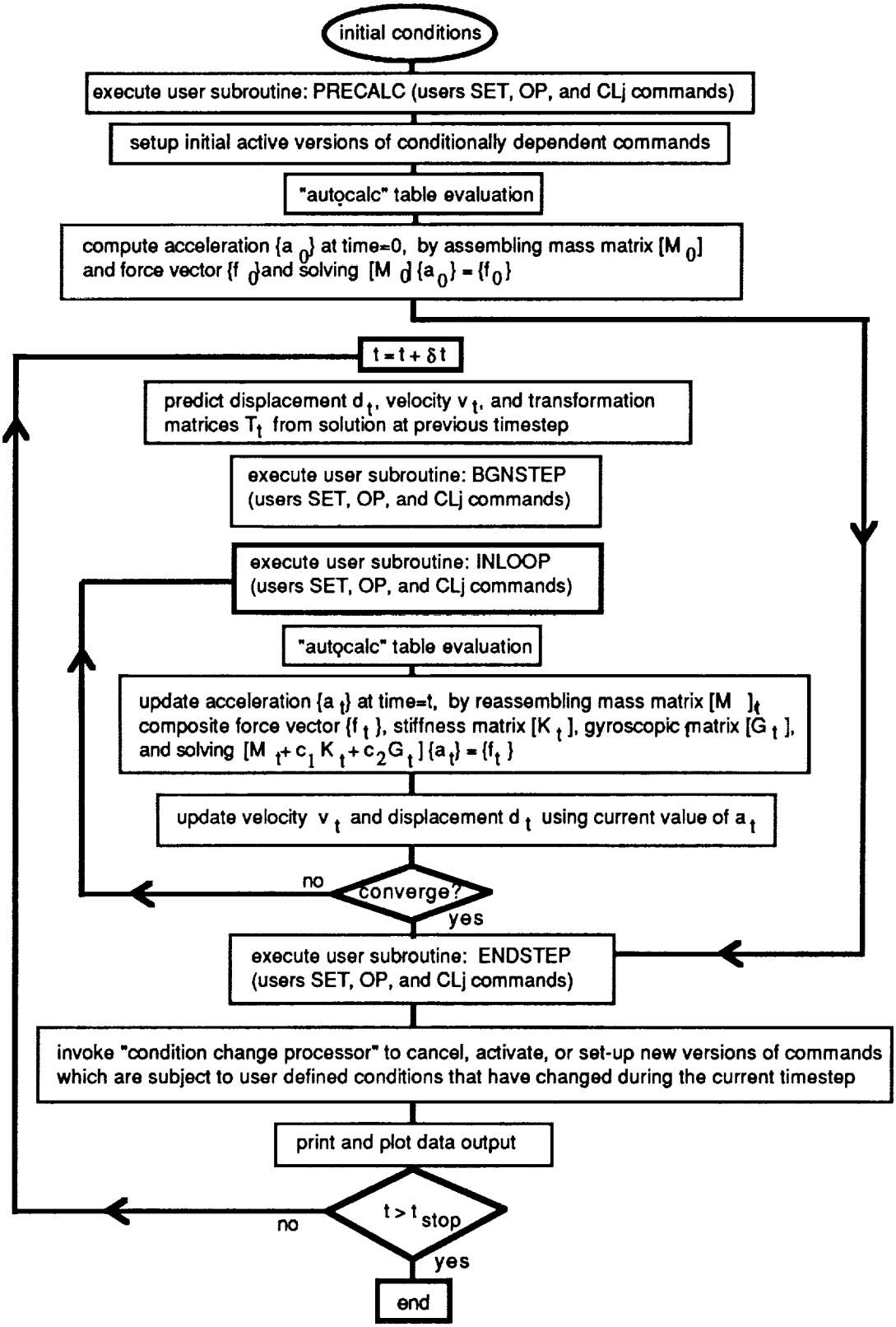


Figure 13.2 Flow Chart for Implicit Integrator

Defining a Condition

CLj, Cj

CLj: FORTRAN Logical Expression

FORTRAN Logical Expression - any valid FORTRAN logical expression which may include references to LATDYN internal functions and/or other condition labels.

EXAMPLES

```
CL1:  X(D,3)  .GT.  1.6
CL2:  ABS(X(V,3))  .LT.  1.E-3      .AND.  CL1
C4:  (Q14+Q16)  .LT.  Q20
CL5:   .NOT.CL1    .OR.    Q5.GT.0.0
```

Creating Q-Variables

The SET command enables a user to define a Q-variable. Q-variables are true program variables which are computed during a LATDYN run in special user subroutines which are called from the LATDYN computational core. The PROGRAM command determines which subroutines in which to insert the specified computations.

SET

SET: Qn= FORTRAN arithmetic expression ? Cj

n — the Q-Variable ID, that is, an integer between 1 and 99

FORTRAN arithmetic expression - any valid FORTRAN arithmetic expression which may include references to LATDYN internal functions

Cj — j'th optional condition label. If this parameter is present then the Q-Variable is SET whenever the command is executed provided that Cj is TRUE.

EXAMPLES

```
SET:  Q2 = MAG(V,#G6) + MAG(V,#G8)
SET:  Q27 = YAW(V,#G4)
SET:  Q5 = H(A,#G7H1)  ? C11
SET:  Q9 = SIN(PI * SPIN(D,#G3)/180)
```

Using Operators in LATDYN

LATDYN includes a large number of symbolic operators for performing a variety of operations. For example, symbolic operators may be used to make Q-vectors from gridpoint

motion quantities or to perform vector algebra. The basic form of the command to invoke any LATDYN operator is as follows:

OP, OPERATOR, OPER

OP: symbolic name (list of parameters) ? Cj

symbolic name	operator name from LATDYN operator library (must conform to the rules for FORTRAN subroutine names for user written operators)
(list of parameters)	parameters as specified for this operator (must be enclosed in parentheses and parameters must be separated by commas, as to conform with the rules for an argument list in a FORTRAN subroutine CALL statement).
Cj	j'th optional condition label. If this parameter is present then the operation is performed whenever the command is executed provided that Cj is TRUE.

EXAMPLES

```
OP:  MKQV1(V, #G3, Q^1)    $ make a Q-vector from the velocity of #G3
OP:  VXV(R^6, Q^5, Q^8)  ? C3  $ take the cross product of R^6 with Q^5 to give Q^8
OP:  MKQV3(V, #C'spinner'G2H1, Q^44)   $ make Q^44 from the rot,vel. of hinge 1,
                                          at #C'spinner'G2
```

LIST OF CONSTANTS FOR USE IN Q-VARIABLES AND CONDITIONS

The following parameters may be used in Q-variables and conditions,

PI	3.141592654
PI2	2*PI
PIH	PI/2
T	integration time
DT	current integration step size

LIST OF FUNCTIONS FOR USE IN Q-VARIABLES AND CONDITIONS

ABVD(vectora ID, vectorb ID)
ABVR(vectora ID, vectorb ID)

FUNCTIONS to calculate the angle between two vectors in degrees (ABVD) or radians (ABVR).

vector a ID - R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname
vector b ID - R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname

ACCM (member#, y, z, h_y, h_z, x_n)

FUNCTION to calculate the acceleration at a point on a flexible beam member. Returns value that would be measured by an accelerometer attached to the member.

member#	member or element identifier (#Mj, #MjEk, #CiMj, or #CiMjEk)
y, z	components giving the orientation of the accelerometer relative to the principal y- and z-axes.
h_y, h_z	distances from the neutral axis in the principal x-y and x-z planes of the member.
x_n	normalized distance along the member or element (=0 at start grid#, =1 at end grid#)

ANGLEX (p, grid#)
ANGLEY (p, grid#)
ANGLEZ (p, grid#)

FUNCTIONS to calculate the angles which the motion vector makes with the global X, Y, Z coordinate axes.

p	motion parameter (D - displacement, V - velocity, A - acceleration)
grid#	gridpoint identifier (#Gj or#CiGj)

ANGLEWX (p', grid#)
ANGLEWY (p', grid#)
ANGLEWZ (p', grid#)

FUNCTIONS to calculate the angles which the gridpoint rotational velocity and acceleration vectors make with the global X, Y, Z coordinate axes.

p'	motion parameter (V - velocity, A - acceleration)
grid#	gridpoint identifier (#Gj or#CiGj)

DIST (grida#, gridb#)

FUNCTION to calculate the distance between two gridpoints.

grid#a, gridb#	gridpoint identifier (#Gj or#CiGj or #Ri)

DOT (Vector a ID, Vector b ID)

FUNCTION to calculate the Dot (scalar) product of two vectors

Vector a ID	R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname
Vector b ID	R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname

H (p, hingepoint#)

FUNCTION to calculate the rotational displacement, velocity, and acceleration of a hingepoint relative to its gridpoint.

p	motion parameter (D - displacement, V - velocity, A - acceleration)
hingepoint#	himgepoint identifier (#GjHk or#CiGjHk or#Ri)

MAG (p, grid#)

FUNCTION to calculate the magnitude of gridpoint motion vector in global system

p	motion parameter (D - displacement, V - velocity, A - acceleration)
grid#	gridpoint identifier (#Gj or#CiGj or#Ri)

MAGV (Vector ID)

FUNCTION for Finding the Length of a Vector

Vector ID	Q^k, R^k, or T^k

MAGVL (Vector ID, p, AXES, "axesname")

FUNCTION for Finding the Length of a Vector With Respect to the Translational Motion of a User Defined Coordinate System

Vector ID	Identifier of vector whose length is to be found (Q^k, R^k, or T^k)
p	Motion Quantity of the coordinate system with respect to which the length of the vector is being calculated. D-displacement, V-velocity, A-acceleration
AXES	the word AXES
"axesname"	identifier of user defined coord. system (must be enclosed in quotes)

SEP (p, grida#, gridb#)

FUNCTION to calculate the separation motion between two gridpoints. Returns the component of relative motion along the line joining the gridpoints.

p	motion parameter (D - relative displacement, V - relative velocity, A - relative acceleration)
grid#a, gridb#	gridpoint identifier (#Gj or#CiGj or #Ri)

STRAIN (member#, h_y, h_z, x_n)

FUNCTION to calculate the strain in a flexible beam member. Returns value that would be measured by a strain gauge placed parallel to x-axis of member.

member#	member or element identifier (#Mj, #MjEk, #CiMj, or #CiMjEk)
h_y, h_z	distances from the neutral axis in the principal x-y and x-z planes of the member.
x_n	normalized distance along the member or element (=0 at start grid#, =1 at end grid#)

YAW (p, grid#)
PITCH (p, grid#)
ROLL (p, grid#)

FUNCTIONS to calculate gridpoint Euler angles and their velocities and accelerations.

p	motion parameter (D - displacement, V - velocity, A - acceleration)
grid#	gridpoint identifier (#Gj or#CiGj or#Ri)

WX (p', grid#)
WY (p', grid#)
WZ (p', grid#)

FUNCTIONS to calculate components of gridpoint rotational velocity and acceleration vectors.

p'	motion parameter (V - velocity, A - acceleration)
grid#	gridpoint identifier (#Gj or#CiGj or #Ri)

WMAG (p', grid#)

FUNCTION to calculate the magnitude of gridpoint rotational velocity and acceleration vectors in global system

p'	motion parameter (V - velocity, A - acceleration)
grid#	gridpoint identifier (#Gj or#CiGj or#Ri)

X (p, grid#)
Y (p, grid#)
Z (p, grid#)

FUNCTIONS to calculate the components of grid point motion in global system

p	motion parameter (D - displacement, V - velocity, A - acceleration)
grid#	gridpoint identifier (#Gj or#CiGj or #Ri)

XLOC (grid#)
YLOC (grid#)
ZLOC (grid#)

FUNCTIONS to calculate the location of a gridpoint in global system

grid#	gridpoint identifier (#Gj or#CiGj or#Ri)

XV(vector ID) YV(vector ID) ZV(vector ID)	FUNCTIONS to compute the X,Y, or Z components of a vector in the global system.

vector ID identifier of the vector whose components are to be found (Q^k, R^k, or T^k)

XVL(vector ID, AXES, "axesname") YVL(vector ID, AXES, "axesname") ZVL(vector ID, AXES, "axesname")	FUNCTIONS to compute the X, Y, or Z components of a Vector in a User Defined Local Coordinate System

NOTE: Function is not presently implemented.

vector ID identifier of the vector whose components are to be found (Q^k, R^k, or T^k)

AXES the word AXES

"axesname" identifier of user defined coordinate system with respect to which the components are to be evaluated (must be enclosed in quotes). Note: for evaluating components, the coordinate system is assumed to be instantaneously stationary.

XVWL(vector ID, p, AXES, "axesname") YVWL(vector ID, p, AXES, "axesname") ZVWL(vector ID, p, AXES, "axesname")	FUNCTIONS to compute the X, Y, or Z components of a Vector in a User Defined Local Coordinate System with Respect to the Motion of that System.

NOTE: Function is not presently implemented.

vector ID identifier of the vector whose components are to be found (Q^k, R^k, or T^k)

p Motion Quantity of the coordinate system with respect to which the length of the vector is being calculated. D-displacement, V-velocity, A-acceleration

AXES the word AXES

"axesname" identifier of user defined coordinate system with respect to which the components are to be evaluated (must be enclosed in quotes).

LIST OF OPERATORS FOR MAKING Q-VECTORS

MKQV1 (p, grid #, Q^k)	OPERATOR for making a Q-vector from a gridpoint translation quantity

p motion parameter (D - displacement, V - velocity, A - acceleration)

grid# gridpoint identifier (#Gj or#CiGj or#Ri)
Q^k identifier of Q-vector to be made

MKQV2 (p, grid #, Q^k)

OPERATOR for making a Q-Vector from a Grid Point Rotation Quantity

p rotational motion parameter (D - displacement, V - velocity, A - acceleration)
grid# gridpoint identifier (#Gj or#CiGj or #Ri)
Q^k identifier of Q-vector to be made

MKQV3 (p, hingepoint#, Q^k)

OPERATOR for making a Q-Vector from Motion of a Hinge

p rotational motion parameter (D - displacement, V - velocity, A - acceleration)
grid# gridpoint identifier (#Gj or#CiGj or#Ri)
Q^k identifier of Q-vector to be made

Note: The vector Q^k which is created by the MKQV3 operator always points in the direction of the hinge axis. The length of the vector is equal to the magnitude of the motion quantity about that axis relative to the grid point to which it is attached.

MKQV4 (p, grid #, 'axesname', Q^k)

OPERATOR for making a Q-Vector from the Relative Translation of a Grid Point in a User Defined Coordinate System

p motion parameter (D - displacement, V - velocity, A - acceleration)
grid# gridpoint identifier (#Gj or#CiGj or#Ri)
Q^k identifier of Q-vector to be made

MKQV5 (p, grid #, axesname, Q^k)

OPERATOR for making a Q-Vector from the Relative Rotation of a Grid Point in a User Defined Coordinate System

p rotational motion parameter (D - displacement, V - velocity, A - acceleration)
grid# gridpoint identifier (#Gj or#CiGj or#Ri)
Q^k identifier of Q-vector to be made

MKQV6 (Vector I.D. for Direction, Magnitude, Q^k)

OPERATOR for making a Q-Vector With the Direction of Another Vector but with a Different Magnitude

Vector I.D. for Direction - gives the direction of the Q-Vector to be made.(R^i, T^i, Q^i, I^axesname, J^axesname, K^axesname)
Magnitude Gives the magnitude of the Q-Vector to be made. May be a real number or a Q-variable.

Q^k gives the I.D. of the Q-Vector to be made.

MKQV7 (Cx, Cy, Cz, Q^k)

OPERATOR for making a Q-Vector by Specifying its Components in the Global System

Cx, Cy, Cz Components of the Q-Vector to be made, expressed in the global system (may be real numbers or Q-Variables)
Q^k identifier of the Q-vector to be made.

MKQV8 (Vector I.D., Q^k)

OPERATOR for Setting a Q-Vector Equal to Another Vector.

Vector I.D. identifies the vector that the Q-vector is to be set equal to (R^i, T^i, Q^i, I^axesname, J^axesname, K^axesname).
Q^k identifier of the Q-vector to be made.

MKQV9 (p, AXES, "axesname", Q^k)

OPERATOR for making a Q-Vector to give the Translational Motion of the Origin of a User Defined Coordinate System

p motion parameter (D - displacement, V - velocity, A - acceleration)
AXES the word AXES
"axesname" refers to the name of the coordinate system (must be enclosed in quotes)
Q^k the vector to be made giving the translational motion of the origin.

MKQV10 (p, AXES, "axesname", Q^k)

OPERATOR for making a Q-Vector to give the Rotational Motion of the Origin of a User Defined Coordinate System

p rotational motion parameter (D - displacement, V - velocity, A - acceleration)
AXES the word AXES
"axesname" refers to the name of the coordinate system (must be enclosed in quotes)
Q^k the vector to be made giving the rotational motion of the origin.

LIST OF OPERATORS FOR VECTOR ALGEBRA

VPV (Vectora ID, Vectorb ID, Q^k)

OPERATOR: Vector Plus Vector

Vectora ID R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname
Vectorb ID R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname
Q^k Vector a + Vector b

VMV (Vectora ID, Vectorb ID, Q^k) — OPERATOR: Vector Minus Vector

Vector a ID	R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname
Vector b ID	R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname
Q^k	Vector a - Vector b

VxV (Vectora ID, Vectorb ID, Q^k) — OPERATOR: Vector Times Vector (cross product)

Vector a ID	R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname
Vector b ID	R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname
Q^k	Vector a x Vector b

CXV (constant, vectora ID, Q^k) — OPERATOR: Constant Times a Vector

Constant	real number or Q-Variable
Vector a ID	R^i or T^i or Q^i or I^axesname or J^axesname or K^axesname
Q^k	the product of C times vectora

ALPABETICAL LISTING OF COMMANDS

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-4401		

4. Title and Subtitle

Large Angle Transient Dynamics (LATDYN) User's Manual

5. Report Date	6. Performing Organization Code
October 1991	

7. Author(s)

A. Louis Abrahamson, Che-Wei Chang, Michael G. Powell, Shih-Chin Wu, Bradford D. Bingel, and Paula M. Theophilos

8. Performing Organization Report No.	10. Work Unit No.
LaRC/LUM 1.0	505-63-53

9. Performing Organization Name and Address

COMTEK
702 East Woodland Road
Grafton, Virginia 23692

11. Contract or Grant No.	13. Type of Report and Period Covered
NAS1-18478	Contractor Report (Final)

12. Sponsoring Agency Name and Address

NASA Langley Research Center
Hampton, Virginia 23665

14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Jerrold M. Housner
A. Louis Abrahamson, Che-Wei Chang, Michael G. Powell, and Shih-Chin Wu: COMTEK, Grafton, Virginia.
Bradford D. Bingel and Paula M. Theophilos: Computer Sciences Corporation, Hampton, Virginia.

16. Abstract

"LATDYN" is a computer code for modeling the Large Angle Transient DYNamics of structures. The objective in developing the code was to investigate new techniques for analysing flexible deformation and control/structure interaction problems associated with large angular motions of spacecraft. Such motions may consist of pointing the entire spacecraft or articulation of individual components, events which occur frequently during construction, operation, and maintaince of large spacecraft .

This type of analysis is beyond the routine capability of conventional analytical tools without simplifying assumptions. In some instances the motion may be sufficiently slow and the spacecraft (or component) sufficiently rigid to simplify analyses of dynamics and controls by making psuedo-static and/or rigid body assumptions.

LATDYN introduces a new approach to the problem by combining finite element structural analysis, multi-body dynamics, and control system analysis,.in a single tool. It includes a new type of finite element that can deform and rotate through large angles at the same time, and which can be connected to other finite elements either rigidly or through mechanical joints. LATDYN also provides symbolic capabilities for modeling control systems which are interfaced directly with the finite element structural model.Thus, the non-linear equations representing the structural model are integrated along with the equations representing sensors, processing and controls as a coupled system.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Large Angle Transient Dynamics Control - Structure Interaction Flexible Multi-Body Dynamics	Unclassified - Unlimited Subject Category 39

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	136	A07

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171

NASA-Langley, 1991