UILU-ENG-94-2017

CIVIL ENGINEERING STUDIES
STRUCTURAL RESEARCH SERIES NO. 596

ISSN: 0069-4274

WARP3D

Dynamic Nonlinear Analysis of Solids Using A Preconditioned Conjugate Gradient Software Architecture

By

Kyle C. Koppenhoefer
Arne S. Gullerud
Claudio Ruggieri
Robert H. Dodds, Jr.
University of Illinois

and

Brian E. Healy
Exxon Production Research Company

A Report on a Research Project
Sponsored by the
U.S. NUCLEAR REGULATORY COMMISSION
OFFICE OF NUCLEAR REGULATORY RESEARCH
DIVISION OF ENGINEERING
WASHINGTON, D.C.

NASA-AMES RESEARCH CENTER
MOFFETT FIELD, CALIFORNIA

DEPARTMENT OF CIVIL ENGINEERING
University of Illinois at Urbana-Champaign
Urbana, Illinois

November 1994

(ph. 3-6933) ← E, TRACI ROGLIN

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. SRS 596	2.	3. Recipient's Accession No.

4. Title and Subtitle
WARP3D: Dynamic Nonlinear Analysis of Solids Using a Pre-conditioned Conjugate Gradient Software Architecture

5. Report Date
November 1994

6.

7. Author(s)
K.C. Koppenhoefer, A.C. Gullerud, C. Ruggieri, R.H. Dodds, B.E. Healy

8. Performing Organization Report No.
UILU-ENG-94-2017

9. Performing Organization Name and Address
University of Illinois at Urbana-Champaign
Department of Civil Engineering
205 N. Mathews Avenue
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.
N61533-92-K-0030
NASA NCC2-5022

12. Sponsoring Organization Name and Address
U.S. Nuclear Regulatory Commission
Office Of Nuclear Regulatory Research
Division Of Engineering
Washington, D.C.

NASA-Ames Research Center,
Moffett Field, California

13. Type of Report & Period Covered
Annual: 10-1-93 to 9-30-94

14.

15. Supplementary Notes

16. Abstract (Limit: 200 words)

This report describes theoretical background material and commands necessary to use the WARP3D finite element code. WARP3D is under continuing development as a research code for the solution of very large-scale, 3-D solid models subjected to static and dynamic loads. Specific features in the code oriented toward the investigation of ductile fracture in metals include a robust finite strain formulation, a general J-integral computation facility (with inertia, face loading), an element extinction facility to model crack growth, nonlinear material models including viscoplastic effects, and the Gurson-Tvergaard dilatant plasticity model for void growth. The nonlinear, dynamic equilibrium equations are solved using an incremental-iterative, implicit formulation with full Newton iterations to eliminate residual nodal forces. Time history integration of the nonlinear equations of motion is accomplished with Newmark's β method. A central feature of WARP3D involves the use of a linear-preconditioned conjugate gradient (LPCG) solver implemented in an element-by-element format to replace a conventional direct linear equation solver. This software architecture dramatically reduces both the memory requirements and CPU time for very large, nonlinear solid models since formation of the assembled (dynamic) stiffness matrix is avoided. Analyses thus exhibit the numerical stability for large time (load) steps provided by the implicit formulation coupled with the low memory requirements characteristic of an explicit code. In addition to the much lower memory requirements of the LPCG solver, the CPU time required for solution of the linear equations during each Newton iteration is generally one-half or less of the CPU time required for a traditional direct solver. All other computational aspects of the code (element stiffnesses, element strains, stress updating, element internal forces) are implemented in the element-by-element, blocked architecture. This greatly improves vectorization of the code on uni-processor hardware and enables straightforward parallel-vector processing of element blocks on multi-processor hardware.

17. Document Analysis a. Descriptors
Finite elements, conjugate gradient, finite-strains, plasticity, Newton, supercomputers

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement
Release Unlimited

19. Security Class (This Report)
UNCLASSIFIED

21. No. of Pages
120

20. Security Class (This Page)
UNCLASSIFIED

22. Price

(See ANSI-Z39.18)

OPTIONAL FORM 272 (4-77)
Department of Commerce

WARP3D

Dynamic Nonlinear Analysis of Solids Using A Preconditioned Conjugate Gradient Software Architecture

By

Kyle C. Koppenhoefer
Arne. S. Gullerud
Claudio Ruggieri
Robert H. Dodds, Jr.
University of Illinois

Brian E. Healy
Exxon Production Research Company

A Report on a Research Project Sponsored by the:

U.S. Nuclear Regulatory Commission
Office of Nuclear Regulatory Research
Division of Engineering
Washington, D.C.
and
NASA–Ames Research Center
Moffett Field, California

University of Illinois
Urbana, Illinois
November 1994

ACKNOWLEDGEMENTS

This investigation was supported by grants principally from the Nuclear Regulatory Commission, the National Aeronautic and Space Administration (Ames Research Facility) and the Heavy Section Steel Technology Program (HSST) at the Oak Ridge National Laboratory.

Early research on the numerical algorithms and software architecture of WARP3D were supported by Grant SCCA 90–82144 from the Illinois Department of Commerce and Grant DE–FG02–85ER25001 from the Department of Energy made to the Center for Supercomputer Research and Development at the University of Illinois.

Computational support for software development and verification was provided by the Department of Civil Engineering HP workstation network made possible, in part, by grants from the Hewlett–Packard Company. Supercomputer (Cray–90) access was made available by the National Aerodynamic Simulation facility operated by the NASA–Ames research center. The support of all these organizations is gratefully acknowledged.

The authors express their appreciation to colleagues Prof. Nick Aravas (University of Pennsylvania), Prof. C. Fong Shih (Brown University), Prof. Alan Needleman (Brown University) and Dr. Sam Key (Sandia National Laboratory) for many helpful suggestions on the implementation of nonlinear constitutive models.

Contents

List of Figures

Chapter 1

Introduction

1.1 What is WARP3D?

This manual describes commands and theoretical background material necessary to use the WARP3D finite element code. WARP3D is under continuing development as a research code for the solution of very large–scale, 3–D solid models subjected to static and dynamic loads. Specific features in the code oriented toward the investigation of ductile fracture in metals include a robust finite strain formulation, a general J–integral computation facility (with inertia, face loading), an element extinction facility to model crack growth, nonlinear material models including viscoplastic effects, and the Gurson–Tvergaard dilatant plasticity model for void growth.

The nonlinear, dynamic equilibrium equations are solved using an incremental–iterative, implicit formulation with full Newton iterations to eliminate residual nodal forces. Time history integration of the nonlinear equations of motion is accomplished with Newmark's β method. A central feature of WARP3D involves the use of a linear–preconditioned conjugate gradient (LPCG) solver implemented in an element–by–element format to replace a conventional direct linear equation solver. This software architecture dramatically reduces both the memory requirements and CPU time for very large, nonlinear solid models since formation of the assembled (dynamic) stiffness matrix is avoided. Analyses thus exhibit the numerical stability for large time (load) steps provided by the implicit formulation coupled with the low memory requirements characteristic of an explicit code. In addition to the much lower memory requirements of the LPCG solver, the CPU time required for solution of the linear equations during each Newton iteration is generally one–half or less of the CPU time required for a traditional direct solver. All other computational aspects of the code (element stiffnesses, element strains, stress updating, element internal forces) are implemented in the element–by–element, blocked architecture. This greatly improves vectorization of the code on uni–processor hardware and enables straightforward parallel–vector processing of element blocks on multi–processor hardware (see Carey and Jiang [11], Flanagan and Taylor [24], Hughes, Ferencz, and Hallquist [41], Healy, Pecknold and Dodds [31] for detailed discussions of blocking strategies).

Research continues to focus on the application of *nonlinear* pre–conditioned conjugate gradient (NLPCG) solvers for solution of large–scale, 3–D finite element models (see for example Biffle [7], [8] [9], Hughes, Ferencz, and Hallquist [41]). The *JAC* codes of Biffle [8] [9] employ NLPCG solvers for the analysis of large, quasi–static solid models. Experience with these codes quickly point out the dominant role played by the relative efficiency of numerical implementations for constitutive models to update stresses. In contrast to the LPCG approach during which stresses are updated *outside* the linear equation solving process, the material state requires updating *inside* each iteration of each load (time) step in the NLPCG approach. The number of NLPCG iterations per step can easily exceed 1000 for even moderate size problems. For simple constitutive models that may be *fully* vectorized, e.g., rate–independent Mises plasticity with a constant hardening, the NLPCG ap-

[†] Numbers in [ ] indicate references listed in Appendix B.

proach has the potential to be very robust and computationally efficient. However, for the increasingly complex nonlinear constitutive models employed in modeling ductile fracture, for example, the stress update routines become very difficult to vectorize and to date are partially vectorized (these models require multi–levels of local Newton solutions to update material state variables). Consequently, the potential benefits offered by NLPCG compared to LPCG are diminished severely. The architecture of WARP3D is designed to accommodate the NLPCG approach in the future should that evolution path for the code become advantageous.

Using WARP3D with the current LPCG strategy, 3–D models containing 30,000–50,000 elements are routinely analyzed on supercomputers (Crays). Models with 8,000 8–node brick elements fit in main memory on 64 MB desktop workstations. They solve with dramatically reduced elapsed times compared to commercial software since no spilling to disk occurs during equation solving coupled with the generally better CPU efficiency of the LPCG solver relative to a conventional direct solver.

WARP3D executes in batch and interactive modes. Traditional batch mode execution is most useful for large analyses on supercomputers which enforce job queuing policies. On Unix workstations, the code is often executed in background (&) mode for long jobs and then interactively during an analysis restart to obtain selected output. Options exist to write information files describing the solution status at completion of each Newton iteration during long analyses executed in batch mode.

WARP3D takes input data from a variety of sources under control of the user. A Patran–to–WARP3D translator program (*patwarp*) is also available to convert a Patran neutral file for the model into a WARP3D input file. Input commands to define the model, loading history, solution parameters, compute and output requests have a format–free, English–like structure. Input files may include extensive user comments and thus are generally self–documenting. Output consists of traditional printed displacements, strains, stresses, etc. in addition to nodal results files in standard Patran format (binary or ascii) written directly by WARP3D. A convenient restart capability provides the facility to segment a long job over multiple runs and to create analysis recovery files in the event of hardware failures or should the solution not converge.

This manual is organized as follows. The remainder of Chapter 1 provides an overview of WARP3D through discussion of an example problem, and background material on the formulation and solution of the governing equations. Chapter 2 describes the commands to define the finite element model, loading history, nonlinear/dynamic solution parameters, compute and output commands. Chapter 3 provides a detailed description of the currently available finite elements and material models. Chapter 4 describes the procedures and commands available to compute *J*–integrals using domain integral techniques. Chapter 5 discusses the procedures and commands to model crack growth. The appendices provide additional details such as the format of nodal results files generated for use in Patran.

A Note About Physical Units

WARP3D does not provide facilities for units conversions. Users are required to specify consistent physical units for all quantities defining the finite element model and loading.

1.2 Illustrative Problem

This section describes the nonlinear analysis of a pre–cracked Charpy–V–Notch (CVN) specimen subjected to impact loading typical of that experienced in a standard, constant velocity test. Figure 1.1 shows the finite element model, dimensions, boundary conditions and loading history. In this example, the 3–D model has one–layer of elements in the thickness direction with plane–strain constraints (w=0) imposed on all nodes. The model has 2008 nodes and 916 elements (8–node bricks with $\bar{\boldsymbol{B}}$ modification). The model was developed and analyzed to support an investigation of crack tip inertia and viscoplastic effects on the near–tip stress fields which drive cleavage fracture in ferritic materials.

The analysis uses the small–strain kinematic formulation with viscoplastic material behavior. Rate–dependent properties characteristic of A533B steel at 100°C are specified. The uniaxial (tensile) inviscid response follows a power–law hardening model (n=10) after yield at σ_0; the viscoplastic response follows a power–law model with an exponent of 35 and a reference strain rate of $1/s$. Displacements imposed at the hammer impact point increase from zero as indicated in the figure to generate a constant velocity loading of 120 in/s after an elapsed time of 5 μs. The analysis covers 200 μs duration in 400 steps with a constant time increment of 0.5 μs. The remainder of this section describes features of the WARP3D input to define the model, loading history, request computations and output, and to compute J–integrals shortly after impact.

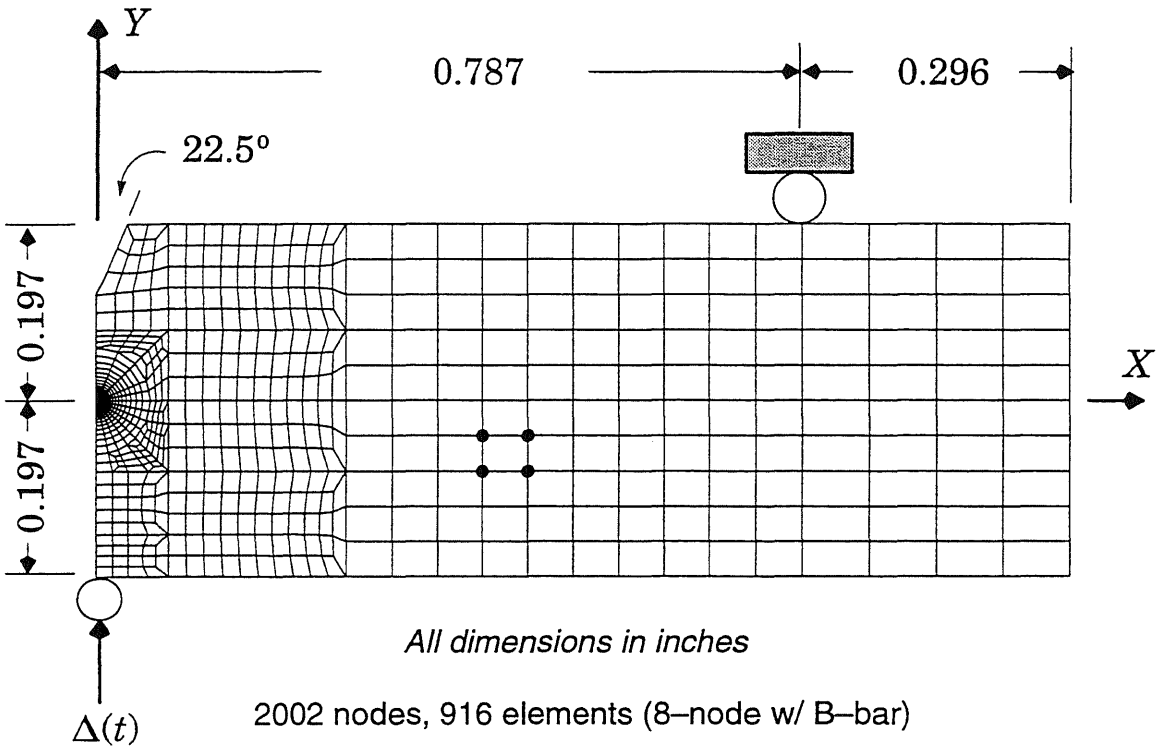
Input for the model begins with a structure command and material definitions.

```
c
c     example cvn analysis with WARP
c
c
structure cvn
c
c
material  a533b
    properties  mises e 30000 nu 0.3 yld_pt 60 n_power 10,
       ref_eps 1.0 m_power 35.0 rho  7.29275e-07
```

WARP3D commands are format free and may begin anywhere on the line. One or more blanks separate data items. A 'c' in column 1 denotes a comment line and is ignored by the input translator. A comma (,) at the end of a line indicates that the input for that command continues on the next line. In the above sequence, we assign a convenient name for the problem (*cvn*) which appears on all printed output and forms the initial part of some output file names. We define a material named *a533b* (any convenient id) and the 'type' of constitutive model as *mises*. Up to 10 materials may be defined as above for subsequent assignment to elements. User assignable properties for the model are specified as shown, with a keyword label followed by a data value. Keywords have easily interpreted names and may be given in any order. Decimal points are optional and may be omitted if not needed to specify the fractional part of a number. Some keywords specify "logical" data values; appearance of the keyword in the input sets the corresponding property value .true. Property *rho* denotes the mass density of the material.

Following the structure id and material definitions, the structure sizes and nodal coordinates are specified as illustrated below:

```
c
number of nodes    2002
number of elements    916
c
```



Material Properties

$E = 30{,}000\ ksi$
$\nu = 0.3$
$\sigma_0 = 60\ ksi$ (inviscid)
$\varrho = 7.29275 \times 10^{-7}\ kip - s^2/in$

$n = 10$ (inviscid power – law hardening)
$m = 35$ (viscoplastic power)
$\dot{\varepsilon}_{ref} = 1\ in/in/s$ (reference strain rate)

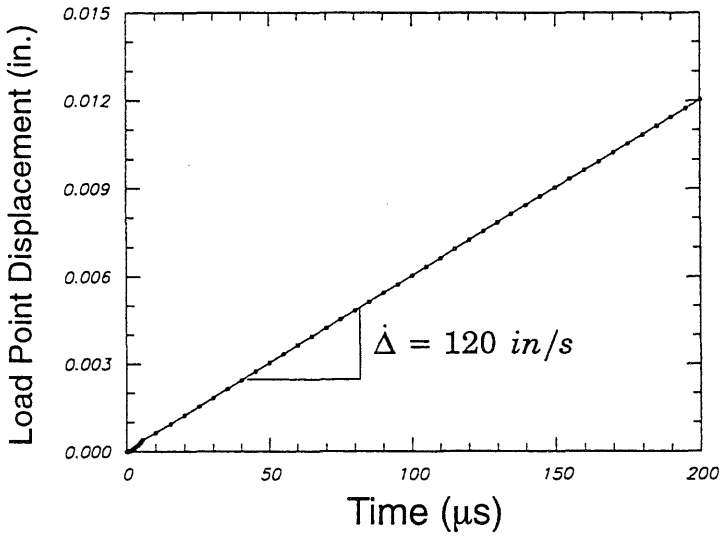


FIG. 1.1—*Pre–cracked Charpy specimen used in illustrative problem.*

```
*echo off
coordinates
     1    .100900006E+01  -.196999982E+00    .000000000E+00
     2    .108300006E+01  -.196999982E+00    .000000000E+00
                          .
                          .
```

The model sizes are required to properly allocate space for internal data arrays. The order of commands to define the sizes is immaterial, and a command of the form *number of nodes 2002 elements 916* applies as well. Nodes and elements must be numbered sequentially and must not have "holes" in the numbering. The **echo off* suppresses data echo of commands as read from the current input file. Various * commands may be specified at any point in the input stream to control the echo, switch to another file for input, etc. Coordinates for nodes are defined in the global *X–Y–Z* system with the origin located at a convenient location. Coordinates for nodes may be specified any number of times; the last specified set of coordinates are retained for analysis. The coordinates here were translated from a Patran neutral file for the model by the *patwarp* program and thus have the *E* format shown.

The 'incidences' define the connectivity of each element node to the corresponding structure node.

```
c
incidences
     1     5     1     4     8     6     2     3     7
     2     8     4    10    12     7     3     9    11
     3    12    10    14    16    11     9    13    15
     4    16    14    18    20    15    13    17    19
     5    20    18    22    24    19    17    21    23
     6    24    22    26    28    23    21    25    27
                          .
                          .
```

Chapter 3 describes the ordering of nodes on the element and the relationship of element nodes to the ordering of Gauss points. Elements may be entered in any order; the last specified set of incidences for the element applies in the analysis. The input translators perform extensive checks on the specified incidences to insure there are no gross errors (e.g., nodes with no elements attached).

The type of each element and the properties for each element are specified next.

```
c
elements
  1-916 type l3disop linear bbar material a533b order 2x2x2
```

In this example, all elements are the 8–node isoparametric (*l3disop*) with a small–strain kinematic formulation (*linear*). The $\overline{B}$ modifications to prevent locking under plastic deformation are requested (*bbar*, a logical property). The previously defined material *a533b* is associated with these elements and the standard 2x2x2 Gauss integration is requested. Other element properties available invoke various output options. All elements have the same material and properties in this example. When this is not the case, any number of similar input lines may be defined to specify the properties. The *integerlist* construction (1–916 above) is convenient and may be used anywhere a list of integers is needed in the input stream. A more general example of an *integerlist* is: 1–400 by 2, 800–600 by –2, 3000–6000 492 496 ...

Each element in the model must be assigned to a "block" for computation. Blocking is required to support optimum vector/parallel operations on supercomputers and is retained

for analyses conducted on Unix workstations. All elements in a block must be the same type (e.g. *l3disop*), have the same material, the same type of kinematic formulation, the same values of some element properties (e.g., integration order, $\overline{\boldsymbol{B}}$) and must not be connected to a common node in the model. This last restriction does not apply for analyses conducted on scalar computers (most Unix workstations) unless the conjugate gradient solver uses the Hughes–Winget preconditioner. The maximum number of elements per block varies with the computer hardware. On Crays, the block size is normally 128 to accommodate vector registers of 128 words in length. On workstations, the cache memory size dictates an optimum block size (usually 32–64).

In this example, the block size is 32; the block number is specified followed by the number of elements in the block and the first element in the block. Elements appearing in a block must be sequentially numbered with no holes. The *patwarp* program which converts a Patran neutral file to a WARP3D input file performs automatic blocking of the elements using a red–black algorithm. The input processors in WARP3D perform exhaustive checks to verify that the rules for blocking assignments are satisfied.

```
c
blocking
      1     32      1
      2     32     33
      3     32     65
      7     32    193
            .
            .
     28     32    865
     29     20    897
```

Nodal constraints in this analysis enforce the plane–strain conditions, the symmetry conditions (u=0) on the crack plane, the v=0 condition at the top, right roller support and the imposed loading to simulate a constant velocity response. A portion of the constraint input is shown below. The specified constraints are the *incremental* displacements imposed over the model during each load (time) step. Constraints may be re–defined as necessary between load steps. When modified, *all* constraints must again be specified. Nodes 499 and 503 in this model are the two nodes at the hammer impact point in the thickness direction. The constraints shown here are applied during load steps 1 and 2. Then a new set of constraints is defined for application in steps 3, 4 with the imposed increments at nodes 499 and 503 doubled in value. Similarly, during steps 5, 6 the v increment at nodes 499, 500 is 3.0E–05; during steps 7, 8 the v increment at nodes 499, 500 is 4.0E–05; and finally during steps 9–400 the v increment at nodes 499, 500 is 6.0E–05. The load point velocity (120 in/s) remains constant over steps 9–400 and is simply the imposed displacement increment / Δt (in this case 6.0E–0/5.5E–06). The slow increase in load point velocity minimizes spurious oscillations in the response.

```
constraints
      1       w      0.0
      2       w      0.0
      3       w      0.0
      4       w      0.0
      5       w      0.0
      6       w      0.0
              .
              .
c
c
```

```
499         v    1.0e-5
503         v    1.0e-5
```

Loads may be applied to the nodes and elements of a model. Element loads, which are dependent on the type of finite element, are converted to equivalent nodal loads by element processing routines. Nodal loads and element loads are grouped together to define *loading patterns.* The loading patterns define the spatial variation and reference amplitudes of loads on a model. Examples of loading patterns include dead load, an internal pressure and simple bending of a component. WARP3D does not currently provide a thermal loading capability.

A *nonlinear* loading condition is declared using previously defined patterns. The term *dynamic* may be used as a synonym for *nonlinear* if desired. A nonlinear/dynamic loading consists of a sequential number of load steps. An incremental–iterative solution is obtained for each load step. For dynamic analyses, a load step is the same as a time step. Each *load step* may consist of loading patterns combined with scalar multipliers. The scaled values of nodal forces (nodal loads and resulting equivalent nodal loads) for the patterns are applied as the new *incremental* load to the model during the step. Loading commands for this example are shown below.

```
c
loading null
  nodal loads
        401 force_y 0
c
 loading disp_ctrl
  dynamic
       step 1-2 null 1.0
       step 3-4 null 2.0
       step 5-6 null 3.0
       step 7-8 null 4.0
       step 9-400 null 6.0
c
```

In this analysis of the CVN specimen, no real "loadings" are needed since the model is loaded by enforced displacements. Nevertheless, a "dummy" loading pattern must be defined to satisfy the syntax requirements for the dynamic loading. Here, the dummy loading is assigned the id "null." The dynamic loading is assigned the id "disp_ctrl." All 400 steps are defined above although this is not required; additional steps may be defined later during the analysis. The scalar multipliers assigned to the pattern (1.0, 2.0, 3.0, 4.0, 6.0) above refer to the relative change in the magnitude of displacement increments. For displacement control loading, these multipliers come into use during extrapolation of displacements from step n to n+1 for accelerating convergence of the Newton iterations.

The user may specify values for a number of nonlinear/dynamic parameters that control the solution procedures In this example, we specify

```
c
 dynamic analysis parameters
  solution technique direct
  maximum iterations 5
  convergence test norm residual tol 0.0005
  nonconvergent solution stop
  time step 0.5e-6
  extrapolate on
```

```
  adaptive solution on
  material messages off
  batch messages on
c
```

A few keywords describing the option are given followed by a required value(s). Some parameters have numerical values while others have *on*, *off* values and others just end with a keyword. Most parameters have suitable default values. A brief explanation of each parameter specified above follows:

- The linear equation solver is specified as *direct*—a skyline Choleski (in–memory) solver. This solver is efficient for small 2–D type models, such as this example. The primary equation solver for large 3–D models uses the linear, preconditioned conjugate gradient algorithm and is requested by the option *lpcg* rather than *direct*.
- The maximum number of Newton iterations to eliminate residual forces in each step is set to 5.
- The Newton convergence test specifies a tolerance of 0.05% on the Euclidean norm of the residual forces relative to the Euclidean norm of the current (total) load vector. Solutions that fail to converge cause termination of the analysis unless the default *stop* value for the *nonconvergent solutions* is changed to *continue*.
- The time step is 5 μs for use in Newmark's β method to integrate the dynamic equilibrium equations.
- *extrapolate on* invokes a nonlinear solution option which imposes the scaled displacement increment computed for step n on the model to start the solution for step n+1. This option greatly accelerates the convergence of Newton iterations for displacement controlled loading.
- The nonlinear *adaptive* strategy is requested; load steps are automatically sub–incremented and re–solved when the specified limit on the number of Newton iterations is reached without convergence. Two levels of adaptivity are available which subdivide, at most, a user specified step into 16 sub–steps. Adaptive solutions that do not converge are terminated and a restart file written.
- Material models (by default) issue messages which notify of first yielding, reversed yielding, and other state changes. These messages are suppressed with *material messages off*.
- This analysis is executed in "batch" mode (&) on a workstation. The *batch messages on* parameter requests that WARP3D write a solution status file following each Newton iteration. The file names are *wm_xxxx_yy* where *xxxx* denotes the step number and *yy* denote the Newton iteration. These files provide information about convergence of the solution.

The model, loading history and solution parameters are now defined. Commands to request an analysis and output of results are given. For the first 10 load steps the commands are:

```
c
  compute displacements for loading disp_ctrl for step 1 2
*echo off
*input from 'forty'
*echo on
c
  compute displacements for loading disp_ctrl for step 3 4
c
 echo off
*input from 'sixty'
*echo on
c
  compute displacements for loading disp_ctrl for step 5 6
c
*echo off
```

```
*input from 'eighty'
*echo on
c
  compute displacements for loading disp_ctrl for step 7 8
c
*echo off
*input from 'one-twenty'
*echo on
c
 compute displacements for loading disp_ctrl for step 9 10
c
 save to file 'cvn_step_10'
c
   output displacements node 798
   output velocity node 798
   output wide eformat strains elements 20-40
   output wide eformat stresses elements 20-40
   output accelerations for elements 100-200 by 2
   output internal_forces 109,110
   output internal_forces 499,503
   output patran binary displ stress strains velocity accelerations
*input from 'domain_define'
stop
```

Here, we request computation of results for load steps 1–2 and then switch the input stream to a file named *forty*. This file contains an entire new set of constraints for the model (the incremental displacements imposed on nodes 499, 503 are increased to 2.0e–5 from 1.0e–5). The first few and last few lines of the file *forty* are

```
constraints
     1       w      0.0
     2       w      0.0
     3       w      0.0
             .
             .
  2002       w      0.0
c
   499       v     2.0e-5
   503       v     2.0e-5
```

The * commands turn off the data echo while the new constraints are being read and then resume the data echo (this is just for convenience and may be omitted). The **input* command specifies the file name for input. We could just as easily have placed the contents of file *forty* in the current input file. The WARP3D input processors sense when the end–of–file condition on *forty* occurs and automatically resume reading from the previous input stream. This sequence of commands is repeated to continue the analysis through load step 10, and in the process ramp the imposed load point velocity to 120 *in/s*.

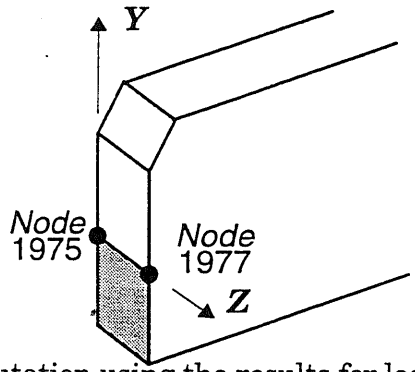
Following completion of the analysis for load step 10, we issue a *save to file ...* command which forces creation of an analysis restart file (sequential, binary) named *cvn_step_10*. The choice of file name resides with the user. This file enables resumption of the analysis at load step 11 in a future program execution (as illustrated subsequently).

Several *output* commands are defined to request printing (to the *current* output device) of nodal and element values (displacements, velocities, accelerations, strains, stresses). These results are displayed in tabular form with appropriate page and column headers. The *internal forces* are reactions at constrained nodal dof. The *output patran ...* command requests creation of *binary* (ascii is optional) files of nodal values written in the required

format for direct post–processing by Patran. These files have the names *pbd#####* , for example, where *pbd* denotes 'patran binary displacements' and ##### indicates the load step number. Appendix A defines the format of Patran results files created by WARP3D.

Finally, an **input* command is specified to read more input from the file *domain_define*. This file contains the input commands

```
c
 domain one
  symmetric
  front nodes  1975 1977  linear
  normal plane nx 1 ny 0 nz 0
  q-values automatic rings  31-35
  print totals
  function type d
 compute domain integral
c
```



We define one "domain" for *J*–integral computation using the results for load step 10. A domain is defined by specifying an "id" (*one* in this example for output headers), the nodes in the domain along the crack front, the *q*–function interpolation order along the front, the orientation of the crack plane relative to the global coordinate system, the number and types of "rings" for *J*–evaluation and output options. The *rings 31–35* option requests that the first *J*–value be computed using elements in the 31^{st} ring of elements enclosing the crack front. *J*–values are then computed over rings 32–35. Values for each ring are printed and statistics shown to assess the path (domain) independence of the values. The *symmetric* parameter causes the code to double *J*–values prior to printing.

In this (effectively) 2–D model, we request computation of a "through–thickness" average *J*–value by specifying *function type d*. In general 3–D models, we specify the sequence *domain ... compute domain integral* at each point on the crack front where *J*–values are required. WARP3D automatically determines that the analysis is dynamic and includes the inertia terms in *J* and crack face loadings if they are present as well.

The input file ends with a *stop* command which terminates program execution. *Restart files must be explicitly requested with the "save" command.*

To restart the analysis at load step 11 in a new execution of the program, the input file for this example begins with the commands

```
c
 retrieve from file 'cvn_step_10'
c
 output displacements 100-200
c
 dynamic analysis parameters
  maximum iterations 4
  convergence test norm residual tol 0.001
  material messages on
  batch messages off
c
 compute displacements for loading disp_ctrl for step 11-20
 save to file 'cvn_step_20'
c
    output displacements node 798
    output velocity node 798
    output wide eformat strains elements ...
       .
```

.
.

The *retrieve* command must be the first non–comment line in the restart file. WARP3D reads this file to restore all internal variables to their values at completion of load step 10. Another *output* command requests more results for step 10 and then several analysis parameters are modified. The analysis for steps 11–20 is requested and the computations are finished, another restart file is created, output commands to print results at step 20 issued, etc.

1.3 Manual Conventions

The input translators for WARP provide a problem oriented language command structure to simply specification of model and solution parameters. This section describes the conventions and notation employed throughout the manual to explain commands.

The appearance within a WARP command of a descriptor of the form

< integer >

implies that the user is to enter an item of data within that position in the statement of the class described by the descriptor (in the above example an integer). The command

number of nodes < integer >

implies that the word *nodes* is to be followed by an integer, such as 1000 or 6870, and that the statement entered by the user as input data should be of the form

```
number of nodes 6870
```

The following are definitions of most of the descriptors used within the language. Those not described below are explained when they first occur in the text.

< integer > a series of digits optionally preceded by a plus or minus sign. Examples are 121, +300, –410.

< real > a series of digits with a decimal point included, or series of digits with a decimal point followed by an exponential indicating a power of 10. Real numbers may be optionally signed. Examples are 1.0, –2.5, 4.3e–01.

< number > is either a < real > or an < integer >. The input translator performs mode conversion as needed for internal storage.

< label > is a series of letters and digits. The sequence must begin with a letter. Input translators also accept the character underbar, _ , as a valid letter. Labels may have the form big_cylinder, for example, to give the appearance of multiple words for readability.

< string > is any textual information enclosed in apostrophes (') or quotes ("). An example is 'this is a string'.

< list > is the notation used to indicate a sequence of positive integer values — usually node and element numbers. Lists generally contain two forms of data that may be intermixed with the same list. The first form of data is a series of integers optionally separated by commas. An example is 1, 3, 6, 10, 12. The second common form of a list implies a consecutive sequence of integers and consists of two integers separated by a hyphen. An example is 1–10, which implies all integers in the sequence 1 through 10. An extension of this form implies a constant increment, e.g., 1–10 by 2 implies 1, 3, 5, 7, 9. A third form, *all*, is sometimes permitted, and implies all physically meaningful integers. The forms of lists are often combined as in ... *nodes 1–100 by 3, 200–300, 500–300 by –3.*

Input to WARP appears as a sequence of English–like commands. Many of the words or phrases in these commands are optional and are permitted for readability or to specify options with a command. In the definition of each command, underlined words are required for proper operation of the input translators. If a portion of a word is underlined, only the underlined portion is required input. Items such as <integer> shown in the command defini-

tions are not underlined but must always be replaced by an item of the specified class. For example, the command phrase defined by

number (of) nodes < integer >

can be shortened to

```
numb of node 10
```

if the user so desires.

In many instances, more than one word is acceptable in a given position within a command. The choices are listed one above the other in the command definition. The command definition

compute { displacements / domain }

indicates that each of the following commands are acceptable

```
compute domain
compute displacements
comp displa
```

Optional words and phrases are enclosed with parentheses, (). In some commands, items may be repeated and/or multiple phrases maybe combined on one data line. This is indicated in the command definition by enclosing the repeatable entries within brackets, [ ]. The command

< integer > [{ X / Y / Z } < number > (,)]

implies that the following sequences are valid:

```
1 x 10 y 10 z 15.3
2 x 15 z 30
30 z -42.5
```

In order to be more descriptive within the command definitions, actual data items (those denoted with <> in the definition) are sometimes described in terms of their physical meaning and followed by the type or class of data item which can be used in the command. For example the command,

structure < name of structure: label >

implies that the data item following the word *structure* is the name of the structure and must a descriptor of type < label >. Examples of acceptable commands are

```
structure cylinder
struct big_block
```

while

```
structure 1a
```

is not acceptable since the name of the structure is not a label (*labels must begin with a letter*).

Continuation Lines

A comma (,) placed at the end of a line causes the subsequent data line to be considered a logical continuation of the current line. There is no limit on the number of continuation lines. Continuation can be invoked at any point in any command.

Comment Lines

Comments may be placed in the input following a Fortran style. The letter 'c' or 'C' appearing in physical column 1 of the data line marks it as a comment line. The line is read and (possibly) echoed by the input translator. The content is ignored and the next data line read.

Line Termination

Line termination is accomplished in one of three ways. First, the last column examined by the input translators is column 72. Secondly, after encountering the first data item on a card, the translators count blanks between data items. If 40 successive blanks are found, the remainder of the line is assumed blank. Finally, a $ indicates an end of line. Space following the $ is ignored by the input translators and is often used for short comments.

1.4 Nonlinear Equations of Motion

The structure occupies the configuration $\boldsymbol{B}_0$ at time $t = 0$ and evolves through time to the deformed configuration $\boldsymbol{B}$ at time t. In the $\boldsymbol{B}_0$ configuration, the structure is undeformed and at rest. In reaching the deformed configuration, the structure may displace in any manner, including simple rigid body translation or rotation in the absence of true deformation. This situation is illustrated in Fig. 1.2. The position vector $\boldsymbol{X}$ identifies a point in the undeformed configuration and $\boldsymbol{x}$ denotes the position vector of the same point in the deformed (current) configuration. The vector $\boldsymbol{d}$ is the displacement vector that takes the point from the initial to the deformed configuration. The coordinates of the structure in the reference configuration represent the geometry interpolated from the parametric coordinates in the isoparametric formulation. The nonlinear implementation of the finite element method in WARP3D employs a continuously updated formulation naturally suited for solids with only translational dof at the nodes. The expression of virtual work defining equilibrium and the equations of motion are defined and solved on the current, $\boldsymbol{B}$, configuration. Throughout the deformation history of the structure, this choice of reference configuration remains in effect.

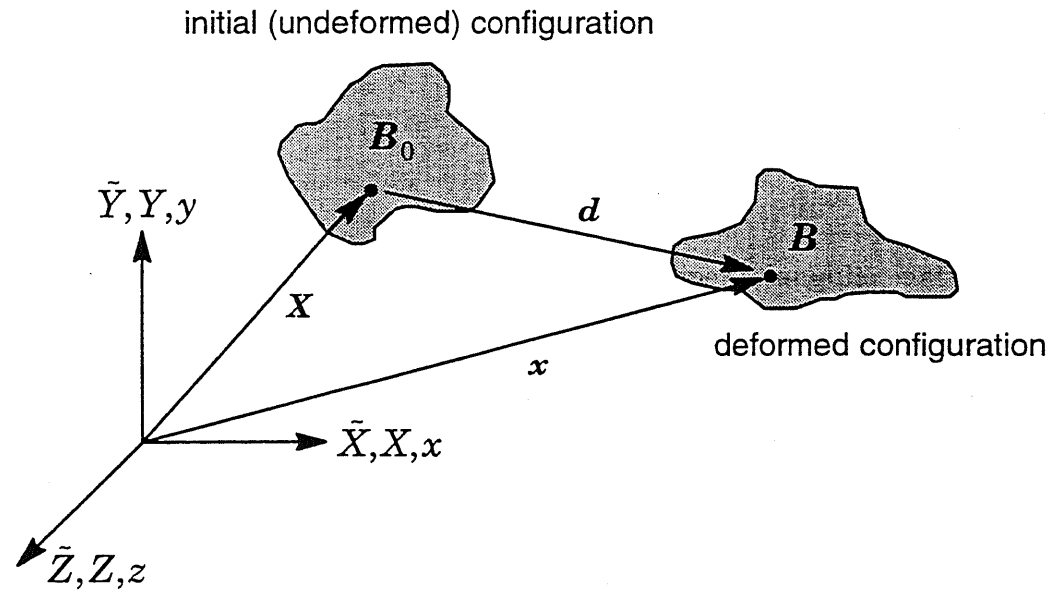


FIG. 1.2—*Definition of initial and current (deformed) configurations. Equations of motion are written on the deformed configuration.*

In the remainder of this section, the equations of motion are derived. Methods for solution of the resulting nonlinear algebraic equations are described in subsequent sections and followed by descriptions of the specific finite element formulations and the adopted formulation to model finite strains and rotations.

The weak formulation of momentum balance equations (virtual work) expressed in the current configuration is given by

$$\int_V \delta\boldsymbol{\varepsilon}^T\boldsymbol{\sigma}\,dV - \int_V \delta\boldsymbol{d}^T\boldsymbol{f}\,dV - \sum_{i=1}^{m} \delta\boldsymbol{d}_i^T\boldsymbol{p}_i = 0 \qquad (1.1)$$

where V denotes the current volume, $\delta\varepsilon$ and σ are the virtual rate of deformation vector and the Cauchy stress vector, $\boldsymbol{f}$ is the body force vector per unit volume in the deformed configuration, and each $\boldsymbol{p}_i$ is a 3×1 vector of concentrated external forces acting at m discrete points (see Malvern [54] , Marsden and Hughes [55]). We use 6×1 vector forms of the symmetric tensors for $\delta\varepsilon$ and σ. The operator δ denotes a small, arbitrary virtual variation. The virtual rate of deformation tensor and the Cauchy stress tensor form a work conjugate pair when defined on the current configuration.

External force vectors remain constant in magnitude and direction over a load step. The nodal forces $\boldsymbol{p}_i$ may comprise directly applied nodal forces and the (work) equivalent nodal forces due to specified surface tractions applied on element faces and other body forces, e.g., self–weight. Inertial D'Alembert forces arising from accelerations are given by

$$\boldsymbol{f} = -\varrho\ddot{\boldsymbol{d}} \tag{1.2}$$

where ϱ is the mass density in the deformed configuration. By including acceleration forces in $\boldsymbol{f}$ and body forces due to self–weight in $\boldsymbol{p}_i$, Eq. (1.1) becomes

$$\int_V \delta\varepsilon^T\sigma dV + \int_V \delta\boldsymbol{d}^T\varrho\ddot{\boldsymbol{d}}dV - \sum_{i=1}^{m}\delta\boldsymbol{d}_i^T\boldsymbol{p}_i = 0 \ . \tag{1.3}$$

Following standard procedures (Hughes [42], Cook et. al [16]), Eq. (1.3) transforms from a purely continuum form to an (equivalent) finite element form as given below beginning with integrations over each element to define the volume integral over the structure

$$\sum_{j=1}^{\#elem}\int_{V_e^j}\delta\varepsilon^T\sigma dV_e + \sum_{j=1}^{\#elem}\int_{V_e^j}\delta\boldsymbol{d}^T\varrho\ddot{\boldsymbol{d}}dV_e - \sum_{i=1}^{m}\delta\boldsymbol{d}_i^T\boldsymbol{p}_i = 0 \tag{1.4}$$

$$\sum_{j=1}^{\#elem}(\delta\boldsymbol{u}_e^T\boldsymbol{I}_e)_j + \sum_{j=1}^{\#elem}(\delta\boldsymbol{u}_e^T\boldsymbol{M}_e\ddot{\boldsymbol{u}}_e)_j - \sum_{i=1}^{m}\delta\boldsymbol{d}_i^T\boldsymbol{p}_i = 0 \tag{1.5}$$

$$\delta\boldsymbol{u}^T\Big(\sum\boldsymbol{I}_e + \Big(\sum\boldsymbol{M}_e\Big)\ddot{\boldsymbol{u}} - \boldsymbol{P}\Big) = 0 \tag{1.6}$$

where $\boldsymbol{u}$ is the global nodal displacement vector, $\boldsymbol{u}_e$ is an element nodal displacement vector, $\boldsymbol{I}_e$ is an element internal force vector, $\boldsymbol{M}_e$ is an element mass matrix, and $\boldsymbol{P}$ is the global external force vector. Subsequent sections outline procedures to compute the element internal force vector and the element mass matrix as well as the element tangent stiffness matrix. The summations in Eq. (1.6) denote the global assembly process. Since the $\delta\boldsymbol{u}$ are arbitrary in nature,

$$\sum\boldsymbol{I}_e + \Big(\sum\boldsymbol{M}_e\Big)\ddot{\boldsymbol{u}} - \boldsymbol{P} = 0 \ . \tag{1.7}$$

After performing the assembly processes implied by the Σ in Eq. (1.7), the global equation of motions become

$$\boldsymbol{I} + \boldsymbol{M}\ddot{\boldsymbol{u}} = \boldsymbol{P} \ . \tag{1.8}$$

The vectors have size $3\times m$, where m denotes the number of structure nodes. Nonlinearity in $\boldsymbol{I}$ arises from the element internal force vectors (geometric and/or material effects) while $\boldsymbol{P}$ become nonlinear when tractions applied to element faces have constant orientation relative to the deformed face (e.g., pressure loads).

1.5 Dynamic Analysis: Newmark β Method

Numerical integration of the equations of motion in WARP3D is performed using a method attributed to Newmark [64]. This approach employs a two parameter family of equations that define the displacement, velocity, and acceleration at time t_{n+1} in terms of the displacement increment from t_n to t_{n+1} and the kinematic state at time t_n. These equations derive from successive application of the extended mean value theorem of differential calculus. Consider first the velocities at time t_n and t_{n+1}. Use of the extended mean value theorem for the first derivative leads to the equation

$$\dot{\boldsymbol{u}}_{n+1} = \dot{\boldsymbol{u}}_n + \Delta t\ \ddot{\boldsymbol{u}}_\gamma\ ; \qquad \ddot{\boldsymbol{u}}_\gamma \in [\ \ddot{\boldsymbol{u}}_n\ ,\ \ddot{\boldsymbol{u}}_{n+1}\ ]\ . \tag{1.9}$$

Using the relationship

$$\ddot{\boldsymbol{u}}_\gamma = (1-\gamma)\ddot{\boldsymbol{u}}_n + \gamma\ddot{\boldsymbol{u}}_{n+1}\ ; \qquad 0 \le \gamma \le 1 \tag{1.10}$$

Eq. (1.9) can be rewritten as

$$\dot{\boldsymbol{u}}_{n+1} = \dot{\boldsymbol{u}}_n + (1-\gamma)\Delta t\ddot{\boldsymbol{u}}_n + \gamma\Delta t\ddot{\boldsymbol{u}}_{n+1}\ . \tag{1.11}$$

Equation (1.11) provides an exact result for a given time interval if the parameter γ can be chosen correctly. Even so, the constant acceleration $\ddot{\boldsymbol{u}}_\gamma$ upon integration of Eq. (1.9) does not necessarily produce the correct displacement at time t_{n+1} in terms of the displacement and velocity at time t_n. Accordingly, the extended mean value theorem for the second derivative is invoked to yield

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta t\dot{\boldsymbol{u}}_n + \frac{\Delta t^2}{2}\ddot{\boldsymbol{u}}_\beta\ ; \qquad \ddot{\boldsymbol{u}}_\beta \in [\ \ddot{\boldsymbol{u}}_n\ ,\ \ddot{\boldsymbol{u}}_{n+1}\ ]\ . \tag{1.12}$$

Again, a relationship having the form

$$\ddot{\boldsymbol{u}}_\beta = (1-2\beta)\ddot{\boldsymbol{u}}_n + 2\beta\ddot{\boldsymbol{u}}_{n+1}\ ;\ \ 0 \le 2\beta \le 1 \tag{1.13}$$

is employed to recast Eq. (1.12) as

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta t\dot{\boldsymbol{u}}_n + \frac{(1-2\beta)}{2}\Delta t^2\ddot{\boldsymbol{u}}_n + \beta\Delta t^2\ddot{\boldsymbol{u}}_{n+1}\ . \tag{1.14}$$

Equation (1.14) also provides an exact for a given time interval as long as the choice of the parameter β proves to be correct. Of course, in general it is impossible to choose either γ or β correctly without knowing the solution in advance, so that the approximation in the Newmark method lies in the choice of γ and β. Newmark showed that to avoid spurious damping in linear systems, the parameter γ should equal 1/2. The pertinent equations of the Newmark method then become

$$\dot{\boldsymbol{u}}_{n+1} = \dot{\boldsymbol{u}}_n + \frac{\Delta t}{2}(\ddot{\boldsymbol{u}}_n + \ddot{\boldsymbol{u}}_{n+1}) \tag{1.15}$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta t\dot{\boldsymbol{u}}_n + \frac{(1-2\beta)}{2}\Delta t^2\ddot{\boldsymbol{u}}_n + \beta\Delta t^2\ddot{\boldsymbol{u}}_{n+1} \tag{1.16}$$

A wide variety of values for the parameter β are possible. For instance, setting β equal to zero leads to the second central difference method. A choice of $\beta = 1/6$ defines the linear acceleration method, wherein the acceleration is assumed to vary linearly over the time increment. The choice of $\beta = 1/4$ produces the constant average acceleration method. Newmark demonstrated that $\beta = 1/4$ renders the method unconditionally stable for linear problems; other choices must satisfy a time step constraint to maintain stability through-

out the solution. For materially nonlinear problems, Schoeberle and Belytschko [72] established that the use of $\beta = 1/4$ leads to unconditional stability when nonlinear equilibrium iterations (Newton) are performed to satisfy an energy convergence criterion, and for nonlinear elastic problems Hughes [37] found much the same situation. In WARP3D, $\beta = 1/4$ is the default value although users can modify this value.

Use of the Newmark method leads to an implicit dynamic formulation in that the solution of a nontrivial system of equations is required to compute a displacement increment. Assuming that β does not equal zero, Eqs. (1.15, 1.16) are manipulated to the form

$$\Delta \boldsymbol{u}_{n+1} = \boldsymbol{u}_{n+1} - \boldsymbol{u}_n \tag{1.17}$$

$$\dot{\boldsymbol{u}}_{n+1} = \frac{1}{2\beta\Delta t}\Delta \boldsymbol{u}_{n+1} - \frac{(1-2\beta)}{2\beta}\dot{\boldsymbol{u}}_n - \frac{(1-4\beta)}{4\beta}\Delta t\ddot{\boldsymbol{u}}_n \tag{1.18}$$

$$\ddot{\boldsymbol{u}}_{n+1} = \frac{1}{\beta\Delta t^2}\Delta \boldsymbol{u}_{n+1} - \frac{1}{\beta\Delta t}\dot{\boldsymbol{u}}_n - \frac{(1-2\beta)}{2\beta}\ddot{\boldsymbol{u}}_n \tag{1.19}$$

Equations (1.17–1.19) are substituted into the equations of motion and into the chosen iterative nonlinear solution algorithm. The total displacement increment for the current time step is computed, $\Delta \boldsymbol{u}_{n+1}$, and that increment is back substituted into Eqs. (1.17–1.19) to define the velocity and acceleration for the current estimate of the solution at time t_{n+1}.

1.6 Solution of Nonlinear Equations: Newton's Method

Recalling the equation of motion, the residual load vector at any time is expressed as

$$\boldsymbol{R} = \boldsymbol{P} - \boldsymbol{I} - \boldsymbol{M}\ddot{\boldsymbol{u}} \tag{1.20}$$

where $\boldsymbol{P}$ is the external load vector, $\boldsymbol{I}$ is the internal force vector, $\boldsymbol{M}$ is the mass matrix, and $\boldsymbol{u}$ is the nodal displacement vector. The residual defines the out–of–balance force vector that arises from nonlinear effects in $\boldsymbol{I}$ and (possibly) $\boldsymbol{P}$ computed for the current estimate of the nodal displacements, $\boldsymbol{u}$. An iterative solution designed to drive the residual to zero is desired. Newton's method for nonlinear equations, illustrated in Fig. 1.3 for a static analysis, can be derived by assuming that there exists an approximate displacement state, $\bar{\boldsymbol{u}}$, in the neighborhood of the exact solution for which a linear mapping represented by

$$\boldsymbol{R}(\boldsymbol{u}) = \boldsymbol{R}(\bar{\boldsymbol{u}}) + d\boldsymbol{R}(\boldsymbol{u}) = \boldsymbol{R}(\bar{\boldsymbol{u}}) + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{u}} d\boldsymbol{u} \tag{1.21}$$

is a good approximation to the residual load vector. The partial derivative in Eq. (1.21) represents the Jacobian matrix which maps the displacement vector to the residual load vector. Presumably, a better approximation, $\bar{\boldsymbol{u}} + d\boldsymbol{u}$, is obtained by setting Eq. (1.21) to zero. The differential increment of the residual load vector (the mass matrix for a given time step is constant), is given by

$$d\boldsymbol{R} = d\boldsymbol{P} - d\boldsymbol{I} - \boldsymbol{M}d\ddot{\boldsymbol{u}} \ . \tag{1.22}$$

The external loads are assumed to remain constant in direction and magnitude over a load (time) step and thus $d\boldsymbol{P}=\boldsymbol{0}$ (loads can change between steps). By using Eq. (1.19) to define the differential acceleration in terms of Newmark's method and by introducing the structure *tangent* stiffness, we have

$$\boldsymbol{M}d\ddot{\boldsymbol{u}} = \frac{1}{\beta \Delta t^2}\boldsymbol{M}d\boldsymbol{u} \ , \tag{1.23}$$

$$d\boldsymbol{I} = \boldsymbol{K}_T d\boldsymbol{u} \ . \tag{1.24}$$

where $\boldsymbol{K}_T$ denotes the tangent stiffness matrix for the structure. Equation (1.22) can then be written in the form

$$d\boldsymbol{R} = -\boldsymbol{K}_T^d d\boldsymbol{u} \tag{1.25}$$

where

$$\boldsymbol{K}_T^d = \boldsymbol{K}_T + \frac{1}{\beta \Delta t^2}\boldsymbol{M} \tag{1.26}$$

defines the dynamic tangent stiffness. The use of $d\boldsymbol{R}$ from Eq. (1.25) in Eq. (1.21) yields

$$\boldsymbol{R}(\boldsymbol{u}) = \boldsymbol{R}(\bar{\boldsymbol{u}}) - \boldsymbol{K}_T^d d\boldsymbol{u} \tag{1.27}$$

which demonstrates that the dynamic tangent stiffness is the negative of the Jacobian matrix relating the residual load vector to the displacement vector:

$$\boldsymbol{K}_T^d = -\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{u}} \ . \tag{1.28}$$

Setting Eq. (1.27) to zero and rearranging defines

$$\boldsymbol{K}_T^d d\boldsymbol{u} = \boldsymbol{R}(\bar{\boldsymbol{u}}) \tag{1.29}$$

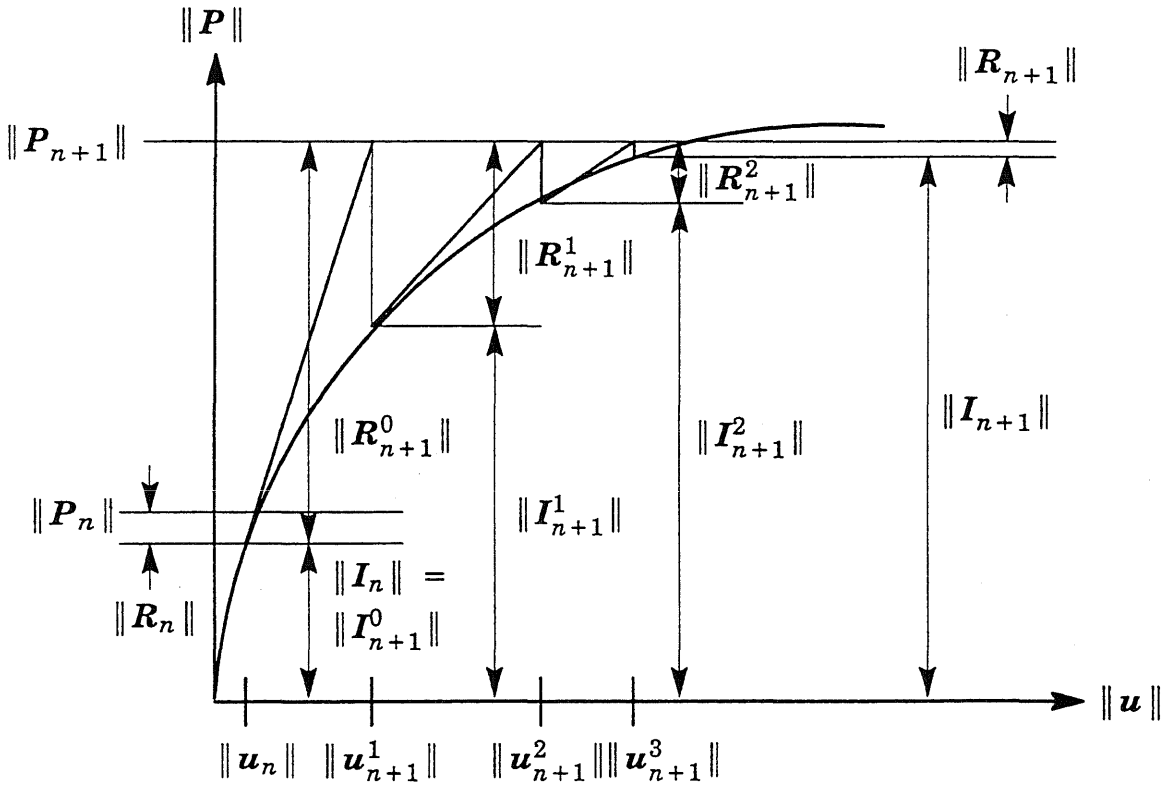


FIG. 1.3—*Illustration of Newton's method for a static analysis*

For finite, rather than differential, increments, the approximate form of Eq. (1.29) becomes

$$\boldsymbol{K}_T^d \delta \boldsymbol{u}_{n+1}^i = \boldsymbol{R}_{n+1}^{i-1} \tag{1.30}$$

where $\delta \boldsymbol{u}_{n+1}^i$ denotes the (corrective) increment of displacement for the current iteration of the time step which advances the solution from n to n+1 and $\boldsymbol{R}_{n+1}^{i-1}$ denotes the residual load after the previous iteration. This residual is defined as

$$\boldsymbol{R}_{n+1}^{i-1} = \boldsymbol{P}_{n+1} - \boldsymbol{I}_{n+1}^{i-1} - \boldsymbol{M}\ddot{\boldsymbol{u}}_{n+1}^{i-1} \tag{1.31}$$

or, after substitution of Eqs. (1.17–1.19), alternatively as

$$\boldsymbol{R}_{n+1}^{i-1} = \boldsymbol{P}_{n+1}^d - \boldsymbol{I}_{n+1}^{i-1} - \frac{1}{\beta \Delta t^2} \boldsymbol{M} \Delta \boldsymbol{u}_{n+1}^{i-1} \tag{1.32}$$

where $\boldsymbol{P}_{n+1}^d$ is the applied load vector at time t_{n+1} modified by terms associated with Eqs. (1.17–1.19):

$$\boldsymbol{P}_{n+1}^d = \boldsymbol{P}_{n+1} + \frac{1}{\beta \Delta t} \boldsymbol{M}\dot{\boldsymbol{u}}_n + \frac{(1-2\beta)}{2\beta} \boldsymbol{M}\ddot{\boldsymbol{u}}_n \ . \tag{1.33}$$

The total change in displacement over the load step, through the current Newton iteration i for the step, is obtained from the summed corrective displacement vectors for the current step, i.e.,

$$\Delta \boldsymbol{u}^i_{n+1} = \sum_{k=1}^{i} \delta \boldsymbol{u}^k_{n+1} \tag{1.34}$$

with the updated estimate for the total displacements at step $n+1$ through iteration i is

$$\boldsymbol{u}^i_{n+1} = \boldsymbol{u}_n + \Delta \boldsymbol{u}^i_{n+1} \ . \tag{1.35}$$

The combination of Eqs. (1.30) and (1.32) defines the basic equation driving the iterative solution associated with the Newton method:

$$\boldsymbol{K}^d_T \delta \boldsymbol{u}^i_{n+1} = \boldsymbol{P}^d_{n+1} - \boldsymbol{I}^{i-1}_{n+1} - \frac{1}{\beta \Delta t^2} \boldsymbol{M} \Delta \boldsymbol{u}^{i-1}_{n+1} \tag{1.36}$$

WARP3D employs a *full* Newton scheme in which the tangent stiffness, $\boldsymbol{K}^d_T$, is updated before the solution of Eq. (1.36) at each iteration. Iterations continue until specified convergence criteria are met or until a specified limit on iterations is reached.

The residual load vector, the dynamic tangent stiffness, and the mass matrix are computed using the element computation algorithms discussed subsequently The solution of the linear simultaneous equations, Eq. (1.36), for the iterative displacement increment is performed by solvers discussed subsequently as well.

Convergence Criteria

Four convergence criteria are provided to support the Newton iterative solution method. They are:

$$1)\ \| \delta \boldsymbol{u}^i_{n+1} \| \ \le \delta_1 \| \delta \boldsymbol{u}^1_{n+1} \| \tag{1.37}$$

$$2)\ \| \boldsymbol{R}^i_{n+1} \| \ \le \delta_2 \| \boldsymbol{P}_{n+1} \| \tag{1.38}$$

$$3)\ \max\left(\ |(\delta u^i_{n+1})_k|, \ k = 1, N_{eq} \right) \ \le \delta_3 \| \delta \boldsymbol{u}^1_{n+1} \| \tag{1.39}$$

$$4)\ \max\left(|(R^i_{n+1})_k|, \ k = 1, N_{eq} \right) \ \le \delta_4 \| \boldsymbol{P}_{n+1} \| \tag{1.40}$$

Tests (2) and (4) include the current reactions for constrained degrees of freedom in the total applied load $\boldsymbol{P}$. This makes possible the use of these two convergence tests for models loaded only by imposed displacements; otherwise $\boldsymbol{P}=\boldsymbol{0}$.

At present there are no mechanisms to control loading in the vicinity of limit points or to otherwise improve performance in such situations, e.g., Riks method.

1.7 Linear Equation Solvers

Solution of the linear set of equations described by Eq. (1.36) is accomplished either by a direct solver or by a linear preconditioned conjugate gradient (LPCG) solver. The direct solver uses a highly optimized, in–memory version of Choleski factorization and back substitution based on profile storage of the upper–triangular portion of the dynamic tangent stiffness matrix for the structure. Because the real memory requirements grow very rapidly for increasingly large 3–D models, the direct solver is recommended only for solution of small 3–D models and for 3–D models which are essentially 2–D, e.g., a one–layer 3–D model to represent a plane–strain or plane–stress problem.

The LPCG solver forms the basis for efficient solution of very large 3–D models in WARP3D. The solution using a LPCG algorithm involves the iterative improvement of an approximate nodal displacement vector, $\boldsymbol{u}$, through a sequence of matrix operations which vectorize naturally and which are amenable to parallel processing. The computational procedure is implemented in an element–by–element architecture which eliminates the need to assemble and store the dynamic tangent stiffness matrix for the structure. Consequently, the memory requirements for solution are dramatically reduced. Moreover, the CPU time required for the LPCG iterative solution frequently is one–half or less of the CPU time required for the direct solver. Both memory and CPU time reductions provided by the LPCG solver are of paramount importance on supercomputers (sometimes making the difference between practical and impractical storage/runtimes). Use of the LPCG solver on Unix workstations often enables the solution of relatively large problems in real memory with CPU time $\approx$ wallclock time. For such problems, the direct solver incurs a severe wallclock time overhead for virtual memory paging to swap the assembled stiffness matrix (often > 200–400 MB) to/from disk storage. Models with 7,500 elements run in–memory on a 64 MB workstation using the LPCG solver with the diagonal preconditioner.

1.7.1 Linear Preconditioned Conjugate Gradient

As stated above, the linear preconditioned conjugate gradient algorithm can be used to solve the linear system of equations in a nonlinear iteration of Newton's method. In the following development, the linear system of equations is denoted by $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{A}$ is understood to be the current estimate of the dynamic tangent stiffness and $\boldsymbol{b}$ the nonlinear residual. The matrix $\boldsymbol{B}$ represents the *preconditioning* matrix. The linear preconditioned conjugate gradient algorithm proceeds as follows:

1) *Initialize:*

$\boldsymbol{x}_0 = \boldsymbol{0}$

for $j = 1,\ N_{eq}$; if j is a constrained dof,
$r_j = 0$
else
$r_j = b_j$
end if

$k = 1$

note: non–zero displacement constraints are placed in the total increment of displacement vector at the beginning of each step and corresponding residual entries are set to zero.

2) *Compute in order:*

$$\boldsymbol{z}_{k-1} = \boldsymbol{B}^{-1}\boldsymbol{r}_{k-1} \tag{1.41}$$

$$\beta_k = \frac{\boldsymbol{z}_{k-1}^T \boldsymbol{r}_{k-1}}{\boldsymbol{z}_{k-2}^T \boldsymbol{r}_{k-2}} \qquad (\beta_1 = 0) \tag{1.42}$$

$$\boldsymbol{p}_k = \boldsymbol{z}_{k-1} + \beta_k \boldsymbol{p}_{k-1} \quad (\boldsymbol{p}_0 = \boldsymbol{0}) \tag{1.43}$$

$$\alpha_k = \frac{\boldsymbol{z}_{k-1}^T \boldsymbol{r}_{k-1}}{\boldsymbol{p}_k^T \boldsymbol{A} \boldsymbol{p}_k} \quad \text{(step length computation)} \tag{1.44}$$

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_k \boldsymbol{p}_k \tag{1.45}$$

$$\boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \alpha_k \boldsymbol{A} \boldsymbol{p}_k \tag{1.46}$$

3) *Check convergence:*

```
if ‖ r_k ‖ ≤ tol ‖ r_0 ‖ then
        LPCG solution converged
else
        if k > iteration limit then
                LPCG solution did not converge
        else
                k = k+1
                return to (2)
        end if
end if
```

The costly operations in the above algorithm are represented by the preconditioning step, Eq. (1.41), and the matrix–vector product required by Eqs. (1.44) and (1.46). Performance of the preconditioning step is discussed below. Because the matrix $\boldsymbol{A}$ is never formed on the global level, the matrix–vector product is computed in blocks of similar, nonconflicting elements.

The key to the performance of the linear preconditioned conjugate algorithm is the choice of a preconditioning matrix, represented by the matrix $\boldsymbol{B}$ in Eq. (1.41). Defining the "A" norm as

$$\| \boldsymbol{x} \|_A = \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \tag{1.47}$$

the rate of convergence in this norm is given by

$$\| \boldsymbol{x} - \boldsymbol{x}_k \|_A = \| \boldsymbol{x} - \boldsymbol{x}_0 \|_A \left[\frac{1 - \sqrt{\varkappa}}{1 + \sqrt{\varkappa}} \right]^{2k} \tag{1.48}$$

where $\varkappa$ is the condition number

$$\varkappa = \lambda_{\max}(\boldsymbol{B}^{-1}\boldsymbol{A}) / \lambda_{\min}(\boldsymbol{B}^{-1}\boldsymbol{A}) \tag{1.49}$$

and $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum eigenvalues of $\boldsymbol{B}^{-1}\boldsymbol{A}$ (see Concus, et al. [15], Golub [25], Hughes et al. [41]). The preconditioning matrix should resemble the inverse of $\boldsymbol{A}$ so that $\varkappa$ approaches unity and convergence is enhanced, and it should also be a relatively trivial matter to invert the preconditioning matrix. Two preconditioners are available in WARP3D as outlined below.

Diagonal Preconditioner

The first and simplest preconditioning matrix is the diagonal preconditioner

$$\boldsymbol{B} = diag(\boldsymbol{A}) \tag{1.50}$$

which represents diagonal scaling or an acceleration of the Jacobi iterative method. Instead of using the current estimate of the dynamic tangent stiffness $\boldsymbol{A}$, it is also possible to employ the diagonal elements of the current estimate of the tangent stiffness or the mass matrix as the preconditioner, although no real advantage results since $\boldsymbol{A}$ must be available in some form (in WARP3D, upper triangular storage by element) to calculate the step length and the linear residual in Eqs. (1.44) and (1.46). The evaluation of Eq. (1.41) using the diagonal preconditioner is accomplished on the global level, as it consists of a simple vector multiply.

Hughes–Winget Preconditioner

The second preconditioner available is the Crout element–by–element preconditioner described by Hughes, et. al. [40], [41]. This preconditioner is an attractive one because it conforms to the element storage of data inherent in the finite element method and it provides an easily vectorizable algorithm for block and parallel processing. The preconditioner consists of the product decomposition

$$\boldsymbol{B} = \boldsymbol{W}^{1/2} \times \prod_{e=1}^{N_{el}} \boldsymbol{L}_p^e \times \prod_{e=1}^{N_{el}} \boldsymbol{D}_p^e \times \prod_{e=N_{el}}^{1} \boldsymbol{U}_p^e \times \boldsymbol{W}^{1/2} \tag{1.51}$$

where

$$\boldsymbol{W} = diag(\boldsymbol{A}) \tag{1.52}$$

and $\boldsymbol{L}_p^e$, $\boldsymbol{D}_p^e$, $\boldsymbol{U}_p^e$ are the lower triangular, diagonal, and upper triangular matrices of the Crout factorization of the corresponding Winget regularized element matrix defined by

$$\overline{\boldsymbol{A}}^e = \boldsymbol{I} + \boldsymbol{W}^{-1/2}(\,\boldsymbol{A}^e - \boldsymbol{W}^e\,)\boldsymbol{W}^{-1/2};\ \boldsymbol{W}^e = diag(\boldsymbol{A}^e) \tag{1.53}$$

The reverse element ordering in the upper triangular product of Eq. (1.51) insures that $\boldsymbol{B}$ is symmetric, and the Winget regularization dictates that the regularized element matrix be positive–definite. Since the regularized element matrix is also symmetric, $\boldsymbol{L}_p^e$ is the transpose of $\boldsymbol{U}_p^e$ and need not be computed or require additional storage. The upper triangular and diagonal matrix factors for a given element are computed by Eqs. (1.54)–(1.57), for each matrix column k as k varies from one to the number of element degrees of freedom.

$$U_{ik}^{e^*} = \overline{A}_{ik}^e\ ; \qquad i = 1,\ k-1 \tag{1.54}$$

$$U_{ik}^{e^*} = \overline{A}_{ik}^e - \sum_{j=1}^{i-1} U_{ji}^e U_{jk}^{e^*}\ ; \qquad i = 2,\ k-1 \tag{1.55}$$

$$U_{ik}^e = \frac{U_{ik}^{e^*}}{D_{ii}^e}\ ; \qquad i = 1,\ k-1 \tag{1.56}$$

$$D_{kk}^e = \overline{A}_{kk}^e - \sum_{j=1}^{k-1} U_{jk}^{e^*} U_{jk}^e\ ; \qquad i = 1,\ k-1 \tag{1.57}$$

The factorization is performed for all elements each time the matrix $\boldsymbol{A}$ is recomputed in the course of the nonlinear iterative solution. In practice, all element matrices are stored and

manipulated in a compact upper triangular vector form. Performance of the element regularizations and factorizations is accomplished in blocks of similar, nonconflicting elements using the element computation algorithms.

The steps required to solve Eq. (1.41) given the preconditioning matrix of Eq. (1.51) are listed as follows:

1) *Global diagonal scaling:*

$$\boldsymbol{z}_0^* = \boldsymbol{W}^{-1/2}\boldsymbol{r}_{k-1} \tag{1.58}$$

2) *Element forward reduction:*

$$\boldsymbol{z}_i^* = (\boldsymbol{L}_p^i)^{-1}\boldsymbol{z}_{i-1}^* \; ; \qquad i = 1,\ N_{el} \tag{1.59}$$

3) *Element diagonal scaling:*

$$\hat{\boldsymbol{z}}_0 = \boldsymbol{z}_{N_{el}}^* \rightarrow \; \hat{\boldsymbol{z}}_i = (\boldsymbol{D}_p^i)^{-1}\hat{\boldsymbol{z}}_{i-1} \; ; \quad i = 1,\ N_{el} \tag{1.60}$$

4) *Element back substitution:*

$$\tilde{\boldsymbol{z}}_{N_{el}+1} = \hat{\boldsymbol{z}}_{N_{el}} \rightarrow \tilde{\boldsymbol{z}}_i = (\boldsymbol{U}_p^i)^{-1}\tilde{\boldsymbol{z}}_{i+1} \; ; \; i = N_{el},\ 1 \tag{1.61}$$

5) *Global diagonal scaling:*

$$\boldsymbol{z}_{k-1} = \boldsymbol{W}^{-1/2}\tilde{\boldsymbol{z}}_1 \tag{1.62}$$

The element operations implied by Eqs. (1.59)–(1.61) are again executed in blocks of similar, nonconflicting elements. Element diagonal scaling is achieved at the global level through the equation

$$\hat{\boldsymbol{z}}_{N_{el}} = \hat{\boldsymbol{W}}^{-1}\boldsymbol{z}_{N_{el}}^* \tag{1.63}$$

where

$$\hat{\boldsymbol{W}}^{-1} = \prod_{e=1}^{N_{el}} \boldsymbol{D}_p^{e\,-1} \tag{1.64}$$

is premultiplied during the regularization and factorization procedure.

1.7.2 Direct Solver

The direct solver dynamically allocates sufficient real memory to store the upper–triangular, profile format of the dynamic tangent stiffness matrix for the structure. Virtual memory (paging) facilities automatically provided by the operating system permit solutions even when the memory required for data storage exceeds the real memory available. The "wallclock" time increases dramatically for solutions that incur significant paging overhead.

The direct solver uses a Choleski procedure to perform forward reduction of the load vector simultaneously with factorization of the dynamic tangent stiffness (see Zienkiewicz and Taylor [83]). Inner loops of the factorization, forward pass and the back pass steps are performed with calls to assembly language routines provided by the computer manufacturer to obtain maximum performance on each platform.

When the direct solver is entered for the first time during program execution, various statistics about the solution are printed, including the actual number of equations to be

solved (constrained dof do not appear in the assembled equations), the amount of real memory required for storage in profile format, the maximum (profile) column height and the average column height.

As for all "node" based direct solvers, the ordering of structure nodes plays the critical role in determining the computational effort required for solution. The factorization time increases approximately as the square of the average column height and linear in the number of equations. WARP3D provides no facilities for node renumbering to reduce column heights.

Pre-processing programs should be used to re-number model nodes to minimize the profile before using the direct solver.

1.8 Element Formulations

Development of the finite element formulation for three dimensional isoparametric elements begins with interpolation of the element displacements and coordinates. The description that follows refers to the kinematic nonlinear formulation; simplifications to obtain the conventional linear kinematic formulation are straightforward.

All quantities are described relative to a fixed set of Cartesian axes, $\bar{\boldsymbol{X}}$, defined at t=0. Let $\boldsymbol{X}$ denote the Cartesian position vectors for material points at $t = 0$ (see Fig. 1.2). Position vectors for material points at time t are denoted $\boldsymbol{x}$. The displacements of material points are thus given by $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{X}$ and the material point velocities by $\dot{\boldsymbol{u}}$ (later we also use $\boldsymbol{v}$ to denote material point velocities). Components of $\boldsymbol{X}, \boldsymbol{x}, \boldsymbol{u}$ and $\dot{\boldsymbol{u}}$ are all defined using the basis vectors for axes $\bar{\boldsymbol{X}}$. In static analyses we associate the time–like parameter t with a specified level of loading imposed on the model. Stress and deformation rates are thus defined with respect to the applied loading rather than with time.

1.8.1 Interpolating Functions

The velocity of a material point at t is interpolated from the nodal velocities using a conventional element interpolating ("shape") function matrix in the form

$$\dot{\boldsymbol{d}} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix}_{3\times 1} = \hat{\boldsymbol{N}} \begin{bmatrix} (\dot{\boldsymbol{u}}_e^x)_{n\times 1} \\ (\dot{\boldsymbol{u}}_e^y)_{n\times 1} \\ (\dot{\boldsymbol{u}}_e^z)_{n\times 1} \end{bmatrix}_{3n\times 1} = \hat{\boldsymbol{N}}\dot{\boldsymbol{u}}_e \tag{1.65}$$

where n here denotes the number of element nodes. Note the non–conventional ordering of nodal displacements in $\dot{\boldsymbol{u}}_e$ which facilitates vectorization of numerical computations. The coordinates of a material point in the configuration at time t are interpolated from the nodal coordinates at t using the same shape functions, resulting in the similar equation

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{3\times 1} = \hat{\boldsymbol{N}} \begin{bmatrix} (\boldsymbol{c}_e^x)_{n\times 1} \\ (\boldsymbol{c}_e^y)_{n\times 1} \\ (\boldsymbol{c}_e^z)_{n\times 1} \end{bmatrix}_{3n\times 1} = \hat{\boldsymbol{N}}\boldsymbol{c}_e \tag{1.66}$$

where $\boldsymbol{c}_e = \boldsymbol{c}_{e,t=0} + \boldsymbol{u}_e$. The element shape functions, one for each element node, are functions of the parametric variables ξ, η, and ζ. For convenience, they are grouped in the row vector

$$\boldsymbol{N} = \langle\, N_1\ N_2\ \ldots\ N_n \,\rangle_{1\times n} \tag{1.67}$$

The shape function derivatives with respect to the parametric variables are represented by the row vectors

$$\boldsymbol{N}_{,\xi} = \langle\, N_{1,\xi}\ N_{2,\xi}\ \ldots\ N_{n,\xi} \,\rangle_{1\times n} \tag{1.68}$$

$$\boldsymbol{N}_{,\eta} = \langle\, N_{1,\eta}\ N_{2,\eta}\ \ldots\ N_{n,\eta} \,\rangle_{1\times n} \tag{1.69}$$

$$\boldsymbol{N}_{,\zeta} = \langle\, N_{1,\zeta}\ N_{2,\zeta}\ \ldots\ N_{n,\zeta} \,\rangle_{1\times n} \tag{1.70}$$

The element shape functions are collected in the element shape function matrix defined by

$$\hat{\boldsymbol{N}} = \begin{bmatrix} \boldsymbol{N} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{N} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{N} \end{bmatrix}_{3\times 3n} \tag{1.71}$$

1.8.2 Cartesian Derivatives

The Jacobian matrix relating differentials in parametric and Cartesian ($\boldsymbol{x}$) coordinates is given by

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}_{3\times 3} \tag{1.72}$$

with the inverse of the Jacobian matrix denoted by

$$\boldsymbol{\Gamma} = \boldsymbol{J}^{-1} \tag{1.73}$$

The gradients of velocity with respect to the $\boldsymbol{x}$ configuration are contained in the vector defined by

$$\dot{\boldsymbol{\Theta}} = \begin{bmatrix} \dot{\boldsymbol{d}}_{,x} \\ \dot{\boldsymbol{d}}_{,y} \\ \dot{\boldsymbol{d}}_{,z} \end{bmatrix} \equiv \begin{bmatrix} \dot{\boldsymbol{\Theta}}_x \\ \dot{\boldsymbol{\Theta}}_y \\ \dot{\boldsymbol{\Theta}}_z \end{bmatrix} = \begin{bmatrix} \dot{u}_{,x} \\ \dot{v}_{,x} \\ \dot{w}_{,x} \\ \dot{u}_{,y} \\ \dot{v}_{,y} \\ \dot{w}_{,y} \\ \dot{u}_{,z} \\ \dot{v}_{,z} \\ \dot{w}_{,z} \end{bmatrix}_{9\times 1} \tag{1.74}$$

The velocity gradients in parametric space constitute the vector

$$\dot{\boldsymbol{\Phi}} = \begin{bmatrix} \dot{\boldsymbol{d}}_{,\xi} \\ \dot{\boldsymbol{d}}_{,\eta} \\ \dot{\boldsymbol{d}}_{,\zeta} \end{bmatrix} = \begin{bmatrix} \dot{u}_{,\xi} \\ \dot{v}_{,\xi} \\ \dot{w}_{,\xi} \\ \dot{u}_{,\eta} \\ \dot{v}_{,\eta} \\ \dot{w}_{,\eta} \\ \dot{u}_{,\zeta} \\ \dot{v}_{,\zeta} \\ \dot{w}_{,\zeta} \end{bmatrix}_{9\times 1} \tag{1.75}$$

The two velocity gradient vectors are related by the equation

$$\dot{\boldsymbol{\Theta}} = \hat{\boldsymbol{\Gamma}}\dot{\boldsymbol{\Phi}} \tag{1.76}$$

where

$$\hat{\boldsymbol{\Gamma}} = \begin{bmatrix} \Gamma_{11}\boldsymbol{I}_3 & \Gamma_{12}\boldsymbol{I}_3 & \Gamma_{13}\boldsymbol{I}_3 \\ \Gamma_{21}\boldsymbol{I}_3 & \Gamma_{22}\boldsymbol{I}_3 & \Gamma_{23}\boldsymbol{I}_3 \\ \Gamma_{31}\boldsymbol{I}_3 & \Gamma_{32}\boldsymbol{I}_3 & \Gamma_{33}\boldsymbol{I}_3 \end{bmatrix}_{9\times 9} \tag{1.77}$$

where $\boldsymbol{I}_3$ denotes a 3×3 identity matrix. The velocity gradients in parametric space are expressed in terms of the nodal velocities by

$$\dot{\boldsymbol{\Phi}} = \boldsymbol{G}\dot{\boldsymbol{u}}_e \tag{1.78}$$

where

$$\boldsymbol{G} = \begin{bmatrix} \hat{N}_{,\xi} \\ \hat{N}_{,\eta} \\ \hat{N}_{,\zeta} \end{bmatrix} = \begin{bmatrix} N_{,\xi} & 0 & 0 \\ 0 & N_{,\xi} & 0 \\ 0 & 0 & N_{,\xi} \\ N_{,\eta} & 0 & 0 \\ 0 & N_{,\eta} & 0 \\ 0 & 0 & N_{,\eta} \\ N_{,\zeta} & 0 & 0 \\ 0 & N_{,\zeta} & 0 \\ 0 & 0 & N_{,\zeta} \end{bmatrix}_{9\times 3n} \tag{1.79}$$

1.8.3 $\boldsymbol{B}$ Matrix

At time t, we impose a compatible virtual displacement field on the the current (deformed) configuration. The corresponding virtual deformation is defined using the 6×1 vector form of the symmetric deformation tensor

$$\delta\varepsilon = \begin{Bmatrix} \delta\varepsilon_x \\ \delta\varepsilon_y \\ \delta\varepsilon_z \\ \delta\gamma_{xy} \\ \delta\gamma_{yz} \\ \delta\gamma_{xz} \end{Bmatrix} = \begin{bmatrix} \delta u_{,x} \\ \delta v_{,y} \\ \delta w_{,z} \\ \delta u_{,y} + \delta v_{,x} \\ \delta v_{,z} + \delta w_{,y} \\ \delta w_{,x} + \delta u_{,z} \end{bmatrix}_{6\times 1} \tag{1.80}$$

where it is understood that, for example, that $\delta u_{,x} = \partial(\delta u)/\partial x$. In terms of the virtual nodal displacements, we write in conventional form

$$\delta\varepsilon_{(6\times 1)} = \boldsymbol{B}_{(6\times 3n)}\delta\boldsymbol{u}_{e(3n\times 1)} \tag{1.81}$$

where the strain–displacement $\boldsymbol{B}$ matrix is constructed as follows. Define the Boolean matrix $\tilde{\boldsymbol{B}}$ by

$$\tilde{\boldsymbol{B}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}_{6\times 9} \tag{1.82}$$

which permits expression of the strain–displacement matrix $\boldsymbol{B}$ by

$$\boldsymbol{B}_{(6\times 3n)} = \tilde{\boldsymbol{B}}_{(6\times 9)}\hat{\boldsymbol{\Gamma}}_{(9\times 9)}\boldsymbol{G}_{(9\times 3n)} \tag{1.83}$$

The vectors and matrices presented in this section form the building blocks of the key element quantities determined below.

1.8.4 Internal Force Vector

The element internal force vector is derived from the internal virtual work term in Eq. (1.4) given by

$$\sum_{j=1}^{\#elem} \int_{V_e^j} \delta\varepsilon^T \sigma dV_e + \sum_{j=1}^{\#elem} \int_{V_e^j} \delta\boldsymbol{d}^T \varrho \ddot{\boldsymbol{d}} dV_e - \sum_{i=1}^{m} \delta\boldsymbol{d}_i^T \boldsymbol{p}_i = 0 \tag{1.84}$$

Using the virtual deformation expressed in terms of the element $\boldsymbol{B}$ matrix, we have (for a single element)

$$\int_{V_e} \delta\varepsilon^T \sigma dV_e = \delta \boldsymbol{u}_e{}^T \int_{V_e} \boldsymbol{B}^T \sigma dV_e \tag{1.85}$$

where again V_e denotes the element configuration at t, σ denotes the symmetric Cauchy stresses expressed in 6×1 vector format at t, and the $\boldsymbol{B}$ matrix is evaluated using coordinates of element nodes at t, $\boldsymbol{c}_e = \boldsymbol{c}_{e,t=0} + \boldsymbol{u}_e$. Using Eq. (1.6) we see that the element internal force vector is given by

$$\boldsymbol{I}_{e(3n\times 1)} = \int_{V_e} \boldsymbol{B}^T \sigma dV_e = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \boldsymbol{B}^T \sigma \, |\boldsymbol{J}| \, d\xi d\eta d\zeta \tag{1.86}$$

The global internal force vector is obtained through global assembly of the element internal force vectors.

1.8.5 Strain Increment for Stress Updating

Newton's method advances the global solution from time step n to n+1 through a series of iterative improvements to the solution at n+1. Let i denote the current Newton iteration for the solution at n+1, $\boldsymbol{u}^{(i)}_{n+1}$ the i^{th} estimate for the element nodal displacements at n+1 and $\boldsymbol{u}_n$ the converged solution for element nodal displacements at n. Using the mid–increment configuration, the i^{th} estimate for the strain increment over the step is given by

$$\Delta\varepsilon^{(i)} = \boldsymbol{B}_{n+\frac{1}{2}} \left(\boldsymbol{u}^{(i)}_{n+1} - \boldsymbol{u}_n \right) \tag{1.87}$$

where the $\boldsymbol{B}$ matrix is evaluated using nodal coordinates $\boldsymbol{c}_e = \boldsymbol{x}_{n+1/2}$. The strain increment $\Delta\varepsilon^{(i)}$ is passed to the stress updating (constitutive) models, after rotation effects are neutralized as described in Section 1.9.4, to obtain the new estimate for the Cauchy stresses at n+1, $\sigma^{(i)}_{n+1}$.

Key and Krieg [47] and Nagtegaal and Veldpaus [62] have demonstrated that Eq. (1.87) defines a constant rate of logarithmic strain over the step. In a one–dimensional setting, integration of the strain rate to define a total strain measure using the mid–point rule above remains surprisingly accurate for very large increments. In multi–dimensional problems, the interpretation of logarithmic strain holds if the principal directions of strain rotate to match the rigid body motion. This rarely happens and thus accumulated increments of converged $\Delta\varepsilon$ values do not represent a valid total strain measure.

1.8.6 Tangent Stiffness Matrix

The element tangent stiffness matrix is defined in terms of the rate of the element internal force vector by

$$\dot{\boldsymbol{I}}_e = \left[\boldsymbol{K}_T\right]_e \dot{\boldsymbol{u}}_e \tag{1.88}$$

From Eq. (1.86) the rate of the element internal force vector is

$$\dot{\boldsymbol{I}}_e = \int_{V_e} \dot{\boldsymbol{B}}^T \sigma \, dV_e + \int_{V_e} \boldsymbol{B}^T \dot{\sigma} \, dV_e \tag{1.89}$$

The first term in Eq. (1.89) can be manipulated into the form (see Zienkiewicz and Taylor [84])

$$\int_{V_e} \dot{\boldsymbol{B}}^T \boldsymbol{\sigma}\, dV_e = \left[\int_{V_e} \boldsymbol{G}^T \hat{\boldsymbol{I}}^T \boldsymbol{M}_\sigma \hat{\boldsymbol{I}} \boldsymbol{G}\, dV_e \right] \dot{\boldsymbol{u}}_e \tag{1.90}$$

where

$$\boldsymbol{M}_\sigma = \begin{bmatrix} \sigma_1 \boldsymbol{I}_3 & \sigma_4 \boldsymbol{I}_3 & \sigma_6 \boldsymbol{I}_3 \\ \sigma_4 \boldsymbol{I}_3 & \sigma_2 \boldsymbol{I}_3 & \sigma_5 \boldsymbol{I}_3 \\ \sigma_6 \boldsymbol{I}_3 & \sigma_5 \boldsymbol{I}_3 & \sigma_3 \boldsymbol{I}_3 \end{bmatrix}_{9\times 9} \tag{1.91}$$

Eq. (1.90) defines the so–called "initial–stress" or geometric stiffness matrix

$$\left[\boldsymbol{K}_T^g\right]_e = \int_{V_e} \boldsymbol{G}^T \hat{\boldsymbol{I}}^T \boldsymbol{M}_\sigma \hat{\boldsymbol{I}} \boldsymbol{G}\, dV_e \tag{1.92}$$

The second term in Eq. (1.89) resolves to

$$\int_{V_e} \boldsymbol{B}^T \dot{\boldsymbol{\sigma}}\, dV_e = \left[\int_{V_e} \boldsymbol{B}^T \boldsymbol{E} \boldsymbol{B}\, dV_e \right] \dot{\boldsymbol{u}} \tag{1.93}$$

where $\boldsymbol{E}$ (6×6) denotes the constitutive matrix relating the (spatial) rate of the deformation to the spatial rate of Cauchy stress, as in

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{E}\dot{\boldsymbol{\varepsilon}} = \boldsymbol{E}\boldsymbol{B}\dot{\boldsymbol{u}}_e \tag{1.94}$$

Since $\dot{\boldsymbol{\sigma}}$ does not vanish under motion corresponding to a rigid rotation (see Johnson and Bammann [45], Rubinstein and Atluri [71]), a rotation neutralized stress rate must be employed in development of the constitutive matrix, $\boldsymbol{E}$. In WARP3D, the Green–Naghdi [27] stress rate is used to formulate $\boldsymbol{E}$ (see Section 1.9.4 for the stress updating strategy).

Upon combining Eqs. (1.92) and (1.93), the element tangent stiffness matrix may be written as

$$\left[\boldsymbol{K}_T\right]_e = \int_{V_e} \left[\, \boldsymbol{G}^T \hat{\boldsymbol{I}}^T \boldsymbol{M}_\sigma \hat{\boldsymbol{I}} \boldsymbol{G} + \boldsymbol{B}^T \boldsymbol{E} \boldsymbol{B} \,\right] dV_e \tag{1.95}$$

$$= \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \left[\, \boldsymbol{G}^T \hat{\boldsymbol{I}}^T \boldsymbol{M}_\sigma \hat{\boldsymbol{I}} \boldsymbol{G} + \boldsymbol{B}^T \boldsymbol{E} \boldsymbol{B} \,\right] |\boldsymbol{J}|\, d\xi d\eta d\zeta \tag{1.96}$$

When required for the direct solver, the tangent stiffness matrix for the structure (in global coordinates) is obtained through the usual assembly of element matrices.

All deformation dependent quantities appearing in Eq. (1.96) refer to values for the i^{th} iteration of step n+1, i.e, $\boldsymbol{B}$ is evaluated using the nodal coordinates $\boldsymbol{x}_{n+1}^{(i)}$, the Cauchy stresses appearing in $\boldsymbol{M}_\sigma$ are $\sigma_{n+1}^{(i)}$ and $\boldsymbol{E}$ is the tangent modulus which advances the spatial rate of Cauchy stress from n to n+1 (i^{th} iteration) *consistent* with the stress updating procedure for the strain increment $\Delta\varepsilon^{(i)}$.

The stiffness formulations employed in WARP3D do not correspond to either of the traditional procedures, Total Lagrangian (T.L.) or Updated Lagrangian (U.L.), (see Bathe

[6], Zienkiewicz and Taylor [84]). In T.L., the tangent stiffness is expressed using all deformation quantities relative to the configuration at t=0. In U.L., the converged solution at n provides the reference configuration for all quantities needed in $[K_T]$. Both of these approaches require the inclusion of additional (nonlinear) terms in $\boldsymbol{B}$ and the use of 2*nd* Piola–Kirchoff stresses rather than the Cauchy stress.

The present formulation, with minor differences, follows closely that used in the NIKE codes (Hallquist [29], [30]).

1.8.7 Mass Matrix

The element consistent mass matrix is derived from the inertial virtual work term in Eq. (1.4) given by

$$\int_{V_e} \delta \boldsymbol{d}^T \varrho \ddot{\boldsymbol{d}} dV_e \tag{1.97}$$

where integration is over the (current) deformed volume and ϱ denotes the mass density per unit of deformed volume. Upon substitution of Eq. (1.65) and its second time derivative, noting that the shape functions are independent of time, Eq. (1.97) becomes

$$\int_{V_e} \delta \boldsymbol{d}^T \varrho \ddot{\boldsymbol{d}} dV_e = \delta \boldsymbol{u}_e{}^T \left[\int_{V_e} \varrho \hat{\boldsymbol{N}}^T \hat{\boldsymbol{N}} dV_e \right] \ddot{\boldsymbol{u}}_e \tag{1.98}$$

A comparison with Eq. (1.6) reveals that the element consistent mass matrix has the form

$$\boldsymbol{M}_e = \int_{V_e} \varrho \hat{\boldsymbol{N}}^T \hat{\boldsymbol{N}} dV_e = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \varrho \hat{\boldsymbol{N}}^T \hat{\boldsymbol{N}} |\boldsymbol{J}| d\xi d\eta d\zeta \tag{1.99}$$

where $|\boldsymbol{J}|$ is evaluated using nodal coordinates at t. Considering the block diagonal structure of Eq. (1.71), the element consistent mass matrix is also block diagonal, and it is only necessary to compute the block diagonal mass matrix corresponding to one of the three continuum degrees of freedom and to assign this matrix to the other two nodal freedoms.

The mass density ϱ appearing in Eq. (1.99) corresponds to the current configuration, as the inertial body force acts there. It may be expressed in terms of the mass density in the undeformed (t=0) configuration by

$$\varrho_0 = \varrho |\boldsymbol{F}| \tag{1.100}$$

where $|\boldsymbol{F}|$ denotes the determinant of the deformation gradient, $\boldsymbol{F} = \partial \boldsymbol{x}/\partial \boldsymbol{X}$. Using the relation $dV_e = |\boldsymbol{F}| dV_0$, and Eq. (1.100), the element consistent mass matrix may be expressed using quantities referenced to the t=0 configuration

$$\boldsymbol{M}_e = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \varrho_0 \hat{\boldsymbol{N}}^T \hat{\boldsymbol{N}} |\boldsymbol{J}_0| d\xi d\eta d\zeta \tag{1.101}$$

where $|\boldsymbol{J}_0|$ is the determinant of the coordinate Jacobian at t=0. The element consistent mass matrix defined by Eq. (1.101) is independent of time; consequently, the element tangent and secant consistent mass matrices are equal.

It is also possible to define a diagonal element lumped mass matrix. This is accomplished in the following manner (Hinton, et al. [35]):

1) Compute the diagonal terms of the block diagonal consistent mass matrix corresponding to one of the continuum degrees of freedom.
2) Accumulate the mass of these diagonal terms. Scale the diagonal terms by the ratio of the total element mass related to the continuum degree of freedom to the accumulated mass so that the total mass of the diagonal terms is correct. Assign the diagonal terms to the other two continuum degrees of freedom. This is the element lumped mass matrix.

Once again, either the global consistent or lumped mass matrix is found through assembly of the element matrices.

1.9 Finite Strain Plasticity

The theoretical basis and numerical implementation of a constitutive architecture suitable for finite strains and rotations are described in this section. The constitutive equations governing finite deformation are formulated using strains–stresses and their rates defined on an *unrotated* frame of reference. Unlike models based on the classical Jaumann [44] (or corotational) stress rate, the present model predicts physically acceptable responses for homogeneous deformations of exceedingly large magnitude. The associated numerical algorithms accommodate the large strain increments which may arise routinely in the implicit solution of the global equilibrium equations employed in WARP3D. The resulting computational framework divorces the finite rotation effects on strain–stress rates from integration of the rates to update the material response over a load (time) step. Consequently, all of the numerical refinements developed previously for small–strain plasticity (radial return, kinematic hardening, consistent tangent operators, dilatant plasticity models for continuum descriptions of void growth) are utilized without modification.

Two fundamental assumptions (and points of criticism, see Simo and Hughes [77]) underlie the present implementation of this framework in WARP3D: (1) additive decomposition of elastic and plastic strain rates expressed on the current configuration remains a valid description of the deformation, and (2) material elasticity maybe adequately represented by an isotropic, hypoelastic model. These assumptions require that plastic strains (and rates) greatly exceed elastic strains (and rates). Such conditions are easily realized in the study of ductile fracture in metals which possess large E/σ_0 ratios. For other materials, such as polymers, the *ad hoc* treatment of elasticity adopted here becomes unsuitable— at best. A multiplicative decomposition of the deformation gradient into elastic and plastic components, when coupled with a proper hyperelastic treatment of material elasticity, is clearly more appropriate (Moran, Ortiz and Shih [58], Simo and Ortiz [76]). Nevertheless, the essential features of the present finite–strain plasticity formulation provide the core technology adopted in large–scale finite element codes, including NIKE ([29] [30]), DYNA ([26]), PRONTO ([78] [79]), ABAQUS–Standard [33] and ABAQUS–Explicit ([34]).

The following sections describe the basis for the constitutive framework and the detailed, step–by–step implementation in WARP3D. Once the kinematic transformations have eliminated rotation effects on rates of tensorial quantities, the stress updating procedures for each constitutive model are those for the conventional small–strain formulation. Details of the usual small–strain computations are described in Chapter 5 for each of the material models currently available.

The reader interested in an extensive description, the numerical implementation details and the criticism of this finite–strain plasticity framework is referred to the monograph of Simo and Hughes [77], specifically Chapters 6 and 7.

1.9.1 Kinematics, Strain–Stress Measures

Development of the finite strain plasticity model begins with consideration of the deformation gradient

$$\boldsymbol{F} = \partial \boldsymbol{x}/\partial \boldsymbol{X}, \qquad \det(\boldsymbol{F}) = J > 0 \tag{1.102}$$

where $\boldsymbol{X}$ denotes the Cartesian position vectors for material points defined on the configuration at t=0. Position vectors for material points at time t are denoted $\boldsymbol{x}$ (configuration $\boldsymbol{B}$ in Fig. 1.4, after Flanagan and Taylor [23]). The displacements of material points are thus given by $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{X}$. The polar decomposition of $\boldsymbol{F}$ yields

$$\boldsymbol{F} = \boldsymbol{VR} = \boldsymbol{RU} \tag{1.103}$$

where $\boldsymbol{V}$ and $\boldsymbol{U}$ are the left– and right–symmetric, positive definite stretch tensors, respectively; $\boldsymbol{R}$ is a orthogonal rotation tensor. The principal values of $\boldsymbol{V}$ and $\boldsymbol{U}$ are the stretch ratios, λ_i, of the deformation. These two methods for decomposing the motion of a material point are illustrated in Fig. 1.4. In the initial configuration, $\boldsymbol{B}_0$, we define an orthogonal reference frame at each material point such that the motion relative to these axes is only deformation throughout the loading history. With the $\boldsymbol{RU}$ decomposition, for example, these axes are "spatial" during the motion from $\boldsymbol{B}_0$ to $\boldsymbol{B}_u$; they are not altered by deformation of the material. However, during the motion from $\boldsymbol{B}_u$ to $\boldsymbol{B}$ these axes are "material"; they rotate with the body in a *local average sense* at each material point. Strain–stress tensors and their rates referred to these axes are said to be defined in the *unrotated* configuration (Johnson and Bammann [45] and Atluri [4]).

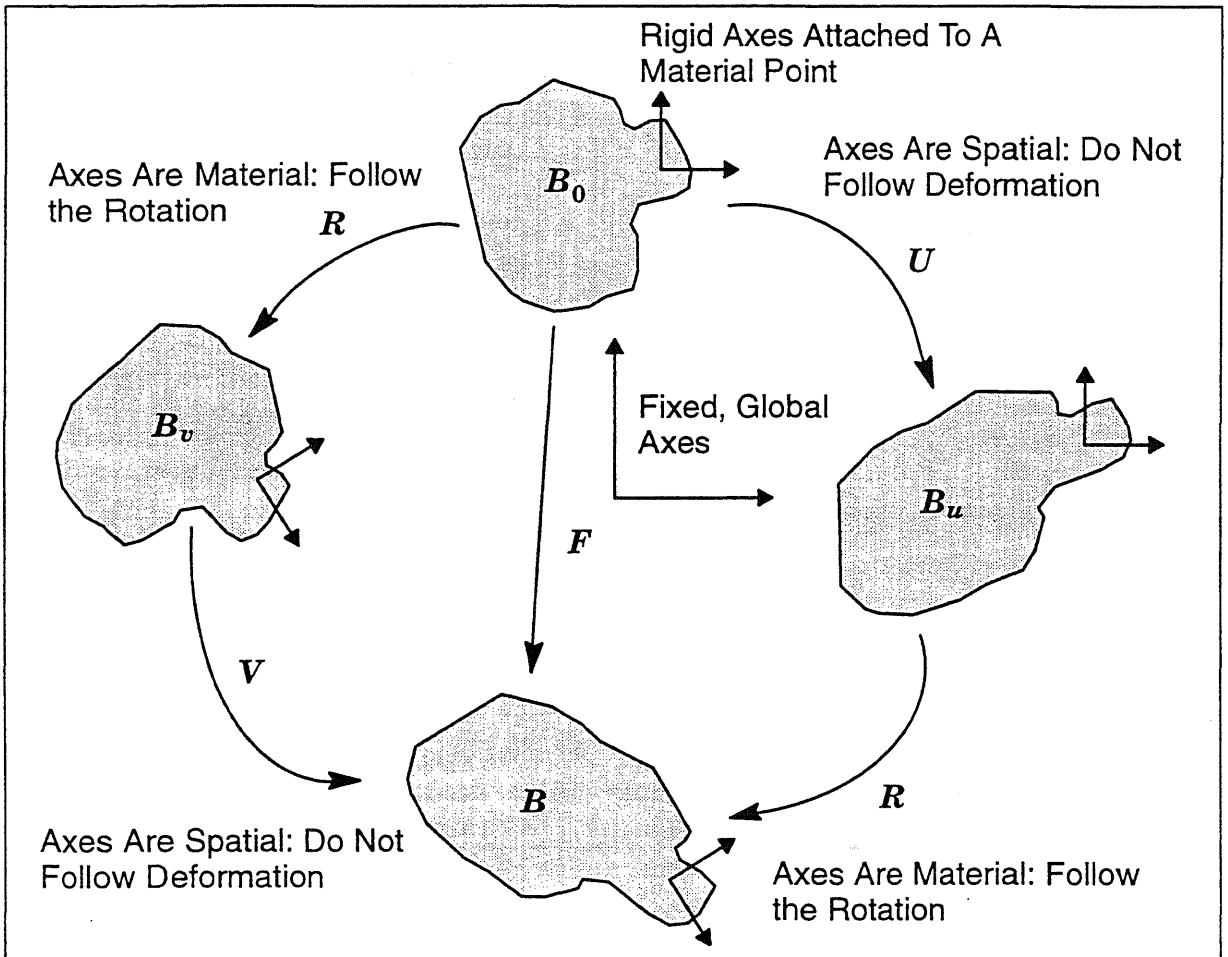


FIG. 1.4—*Motion of Model Using Polar Decomposition ([23])*

The material derivative of displacement with respect to an applied loading parameter is written as $\boldsymbol{v} = \dot{\boldsymbol{x}}$ (i.e., the material point velocity in dynamic analyses). The spatial gradient of this material derivative with respect to the current configuration is given by

$$\boldsymbol{L} = \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{X}} \frac{\partial \boldsymbol{X}}{\partial \boldsymbol{x}} = \dot{\boldsymbol{F}} \boldsymbol{F}^{-1} \ . \tag{1.104}$$

The symmetric part of $\boldsymbol{L}$ is the spatial rate of the deformation tensor, denoted $\boldsymbol{D}$; the skewsymmetric part, denoted $\boldsymbol{W}$, is the spin rate or the vorticity tensor. Thus,

$$\boldsymbol{L} = \boldsymbol{D} + \boldsymbol{W} \tag{1.105}$$

where

$$\boldsymbol{D} = \tfrac{1}{2}(\boldsymbol{L} + \boldsymbol{L}^T); \quad \boldsymbol{W} = \tfrac{1}{2}(\boldsymbol{L} - \boldsymbol{L}^T) \ . \tag{1.106}$$

$\boldsymbol{W}$ represents the rate of rotation of the principal axes of the spatial rate of deformation $\boldsymbol{D}$. When integrated over the loading history, the principal values of $\boldsymbol{D}$ are recognized as the logarithmic (true) strains of infinitesimal fibers oriented in the principal directions if the principal directions do not rotate. It is important to note that $\boldsymbol{D}$ and $\boldsymbol{W}$ have no sense of the deformation history; they are instantaneous rates.

Using the $\boldsymbol{RU}$ decomposition of $\boldsymbol{F}$, the spatial gradient $\boldsymbol{L}$ may be also written in the form

$$\boldsymbol{L} = \dot{\boldsymbol{R}}\boldsymbol{R}^T + \boldsymbol{R}\dot{\boldsymbol{U}}\boldsymbol{U}^{-1}\boldsymbol{R}^T \tag{1.107}$$

in which the following relations are used

$$\dot{\boldsymbol{F}} = \boldsymbol{R}\dot{\boldsymbol{U}} + \dot{\boldsymbol{R}}\boldsymbol{U} \tag{1.108}$$

and

$$\boldsymbol{F}^{-1} = (\boldsymbol{RU})^{-1} = \boldsymbol{U}^{-1}\boldsymbol{R}^{-1} = \boldsymbol{U}^{-1}\boldsymbol{R}^T \ . \tag{1.109}$$

The first term in Eq. (1.107) is the rate of rigid–body rotation at a material point and is denoted $\boldsymbol{\Omega}$ (see Dienes [20]). The spin rate $\boldsymbol{W}$ and $\boldsymbol{\Omega}$ are identical when the principal axes of $\boldsymbol{D}$ coincide with the principal axes of the current stretch $\boldsymbol{V}$ (this observation plays an essential role later in development of a linearized tangent operator). Simple extension and pure rotation satisfy this condition. The symmetric part of the second term in Eq. (1.107) is called the *unrotated* deformation rate tensor (sometimes the *rotation neutralized* deformation rate) and is denoted $\boldsymbol{d}$

$$\boldsymbol{d} = \tfrac{1}{2}(\dot{\boldsymbol{U}}\boldsymbol{U}^{-1} + \boldsymbol{U}^{-1}\dot{\boldsymbol{U}}) \ . \tag{1.110}$$

The unrotated rate of deformation defines a material strain rate relative to the orthogonal reference frame indicated on configuration $\boldsymbol{B}$ in Fig. 1.4.

Using the orthogonality property of $\boldsymbol{R}$ that $d(\boldsymbol{R}^T\boldsymbol{R})/dt$=0

$$\boldsymbol{R}^T\dot{\boldsymbol{R}} + \dot{\boldsymbol{R}}^T\boldsymbol{R} \equiv \boldsymbol{0} \tag{1.111}$$

the unrotated deformation rate may be expressed in the simpler form as

$$\boldsymbol{d} = \boldsymbol{R}^T\boldsymbol{D}\boldsymbol{R} \ . \tag{1.112}$$

The principle of virtual displacements (Section 1.4) demonstrates that the spatial rate of deformation, $\boldsymbol{D}$, and the symmetric Cauchy (true) stress, σ, are work conjugate in the sense that work per unit volume in the current configuration is given by $\sigma_{ij}D_{ij}$. Since components of both $\boldsymbol{D}$ and σ are defined relative to the fixed, global axes, the work conjugate stress measure for $\boldsymbol{d}$ on the unrotated configuration is given simply by

$$\boldsymbol{t} = \boldsymbol{R}^T\sigma\boldsymbol{R} \tag{1.113}$$

where $\boldsymbol{t}$ is termed the *unrotated* Cauchy stress, i.e., σ is the tensor $\boldsymbol{t}$ expressed on the fixed global axes.

1.9.2 Selection of Strain and Stress Rates

The simplest form of a hypo–elastic constitutive relation is adopted to couple a materially objective stress rate with a work conjugate deformation rate. The Jaumann and Green–Naghdi objective rates of Cauchy stress are

$$\tilde{\sigma} = \dot{\sigma} - \boldsymbol{W}\sigma + \sigma\boldsymbol{W} = \boldsymbol{E} : \boldsymbol{D} \quad \text{(Jaumann)} \tag{1.114}$$

$$\hat{\sigma} = \dot{\sigma} - \boldsymbol{\Omega}\sigma + \sigma\boldsymbol{\Omega} = \boldsymbol{E} : \boldsymbol{D} \quad \text{(Green–Naghdi)} \tag{1.115}$$

where the modulus tensor $\boldsymbol{E}$ may depend linearly on the current stress tensor and on history dependent state variables ($\boldsymbol{E} : \boldsymbol{D}$ denotes $E_{ijkl}D_{kl}$). Once the objective stress rate is evaluated using $\boldsymbol{E} : \boldsymbol{D}$, the spatial rate of Cauchy stress, $\dot{\sigma}$, is found by computing $\boldsymbol{W}$ or $\boldsymbol{\Omega}$ and transposing the above equations. In a finite–element setting, these rate expressions are numerically integrated to provide incremental values of the Cauchy stress corresponding to load (time) steps.

When $\boldsymbol{D}$ vanishes both the Jaumann and Green–Naghdi rates predicted by the constitutive models also vanish; however, the two stress rates lead to different spatial rates of Cauchy stress since $\boldsymbol{W}$ and $\boldsymbol{\Omega}$ are generally not identical. Use of the spin tensor $\boldsymbol{W}$ in Eq. (1.114) causes the physically unreasonable (oscillatory) response predicted for the finite shear problem; the Green–Naghdi rate leads to a realistic response. However, the debate ofver physically meaningful stress rates continues.

The Jaumann rate is adopted extensively in finite element codes—the quantity $\boldsymbol{W}$ is readily available as a by–product of computing $\boldsymbol{D}$ whereas computation of $\boldsymbol{\Omega}$ requires polar decompositions of $\boldsymbol{F}$. Hughes and Winget [39] recognized that a constant spin rate $\boldsymbol{W}$ (and rotation rate $\boldsymbol{\Omega}$) limits the acceptable step sizes for implicit codes. They developed a numerical integration scheme for Eq. (1.114) that retains objectivity for rotation increments exceeding 30°. Such refinements, however, do not remove the fundamental cause ($\boldsymbol{W}$) of the oscillatory response in simple shear. Roy, Fossum and Dexter [70] recently implemented a 2–D, implicit finite–element code based on the Green–Naghdi rate as expressed in Eq. (1.115). They employed the Hughes–Winget procedure to integrate $\hat{\sigma}$ using $\boldsymbol{\Omega}$ computed from polar decompositions of $\boldsymbol{F}$ at the start and end of each load increment.

The Green–Naghdi rate may be written alternatively as the rate of *unrotated* Cauchy stress, $\dot{\boldsymbol{t}}$, expressed on the fixed, Cartesian axes

$$\hat{\sigma} = \boldsymbol{R}\dot{\boldsymbol{t}}\boldsymbol{R}^T = \boldsymbol{E} : \boldsymbol{D} \ . \tag{1.116}$$

Transformation of the spatial deformation rate $\boldsymbol{D}$ in this expression to the unrotated deformation rate $\boldsymbol{d}$ yields

$$\dot{\boldsymbol{t}} = \boldsymbol{E} : \left(\boldsymbol{R}^T\boldsymbol{D}\boldsymbol{R}\right) = \boldsymbol{E} : \boldsymbol{d} \ . \tag{1.117}$$

Constitutive computations, equivalent to the Green–Naghdi rate in Eq. (1.115), therefore can be performed using stress–strain rates defined on the unrotated configuration. Updated values of $\boldsymbol{t}$ are rotated via $\boldsymbol{R}$ to obtain the updated Cauchy stress at the end of a load increment. The numerical problems of integrating the rotation rates in Eqs. (1.114) and (1.115) are thus avoided. Moreover, tensorial state variables of the plasticity model, e.g., the back–stress for kinematic hardening, are also defined and maintained on the unrotated configuration and thus never require correction for finite rotation effects. Hallquist [29], [30] was apparently the first to recognize the simplicity derived from this constitutive framework and used it in the NIKE and DYNA codes. Later, this framework was adopted by Flanagan and Taylor for the PRONTO–2D [78] and PRONTO–3D [79] codes, by Biffle

and Blandford for the JAC–2D [7] and JAC–3D [8] codes, and most recently in the commerical ABAQUS–EXPLICIT [34] code. The potential disadvantage of this constitutive framework is the numerical effort to compute $\boldsymbol{R}$ from the polar decomposition $\boldsymbol{F}=\boldsymbol{R}\boldsymbol{U}$ at thousands of material points for each of many load steps. For explicit codes in which time steps are necessarily very small to maintain stability, an efficient (forward) integration scheme developed by Flanagan and Taylor [23] may be used to update $\boldsymbol{R}$ without the polar decomposition. The polar decomposition issue is discussed in the section on numerical procedures.

1.9.3 Elastic–Plastic Decomposition

Further developments require kinematic decomposition of the total strain rate $\boldsymbol{d}$ into elastic and plastic components. The multiplicative decomposition of the deformation gradient

$$\boldsymbol{F} = \boldsymbol{F}^e\boldsymbol{F}^p \tag{1.118}$$

appears most compatible with the physical basis of elastic–plastic deformation in crystalline metals (see, for example, Lee [50] and Asaro [3]). $\boldsymbol{F}^p$ represents plastic flow (dislocations) while $\boldsymbol{F}^e$ represents lattice distortion; rigid rotation of the material structure may be considered in either term. Substitution of this decomposition into the spatial rate of the displacement gradient Eq. (1.104) yields

$$\boldsymbol{L} = \dot{\boldsymbol{F}}^e\boldsymbol{F}^{-e} + \boldsymbol{F}^e\dot{\boldsymbol{F}}^p\boldsymbol{F}^{-p}\boldsymbol{F}^{-e} = \boldsymbol{L}^e + \boldsymbol{F}^e\boldsymbol{L}^p\boldsymbol{F}^{-e} \ . \tag{1.119}$$

We now impose the restriction that elastic strains remain vanishingly small compared to the unrecoverable plastic strains; a behavior closely followed by ductile metals having an elastic modulus orders of magnitude greater than the flow stress. Consequently, $\boldsymbol{F}^p$ and $\boldsymbol{F}^e$ are uniquely determined by unloading from a plastic state. This considerably simplifies the above expression and permits separate treatment of material elasticity and plasticity. Using the left polar decomposition and writing the stretch as the product of elastic and plastic parts yields

$$\boldsymbol{F} = \boldsymbol{F}^e\boldsymbol{F}^p = \boldsymbol{V}^e\boldsymbol{V}^p\boldsymbol{R} \tag{1.120}$$

Identifying the elastic deformation as

$$\boldsymbol{F}^e = \boldsymbol{V}^e \tag{1.121}$$

and using the small elastic strain assumption, we have

$$\boldsymbol{F}^e = \boldsymbol{I} + \boldsymbol{e}^e \approx \boldsymbol{I} \ . \tag{1.122}$$

Consequently, the expression for $\boldsymbol{L}$ is approximated by

$$\boldsymbol{L} \approx \boldsymbol{L}^e + \boldsymbol{L}^p \ . \tag{1.123}$$

As in Eq. (1.106), the symmetric part of this approximation for $\boldsymbol{L}$ is taken as $\boldsymbol{D}$ with the result that

$$\boldsymbol{D} \approx \boldsymbol{D}^e + \boldsymbol{D}^p \ . \tag{1.124}$$

Given the restriction of vanishingly small elastic strains, the multiplicative decomposition of the deformation gradient in Eq. (1.118) leads to the familiar additive decomposition of the spatial deformation rate $\boldsymbol{D}$ into elastic and plastic components. The transformation of $\boldsymbol{D}$ to the unrotated configuration using Eq. (1.112) provides the decomposition scheme needed for $\boldsymbol{d}$ as

$$\boldsymbol{d} = \boldsymbol{R}^T(\boldsymbol{D}^e + \boldsymbol{D}^p)\boldsymbol{R} = \boldsymbol{d}^e + \boldsymbol{d}^p \ . \tag{1.125}$$

Once the above transformation of elastic and plastic strain rates onto the unrotated configuration is accomplished, the remaining steps in development of the finite–strain plasticity theory are identical to those for classical small–strain theory.

If the elastic strains are not vanishingly small, the incrementally linear form of this hypo–elastic material model predicts hysteretic dissipation and residual stresses for some closed loading paths, for example, the path defined by finite extension→finite shear→tension unloading→shear unloading (Kojic and Bathe [49]). Uncoupled loading–unloading for extension and shear produces no residual stresses. For finite–strain plasticity of ductile metals having large modulus–to–yield stress ratios this situation is not a serious concern since plastic strains are commonly 50–100 times greater than the elastic strains.

1.9.4 Numerical Procedures

The global solution is advanced from time (load) t_n to t_{n+1} using an incremental–iterative Newton method. Iterations at t_{n+1} to remove unbalanced nodal forces are conducted under fixed external loading and no change in the prescribed displacements for displacement controlled loading. Each such iteration, denoted i, provides a revised estimate for the total displacements at t_{n+1}, denoted $\boldsymbol{u}^{(i)}_{n+1}$. Fully converged displacements at t_n are denoted $\boldsymbol{u}_n$. Following Pinsky, Ortiz and Pister [67] a mid–increment scheme is adopted in which deformation rates are evaluated on the intermediate configuration at $(1-\gamma)\boldsymbol{u}_n + \gamma\boldsymbol{u}^{(i)}_{n+1}$. The choice of $\gamma = 1/2$ represents a specific form of the generalized trapezoidal rule that is unconditionally stable and second–order accurate. Key and Krieg [47] have demonstrated the optimality of the mid–point configuration for integrating the rate of deformation and the resulting correspondence with logarithmic strain (for uniaxial conditions).

The following sections describe the computational processes performed at each material (Gauss) point to: 1) update stresses and to 2) provide a consistent tangent matrix for updating the global stiffness matrix. A brief discussion of the procedure to compute the polar decomposition of the deformation gradient is also provided.

Stress Updating Procedure

The computational steps are:

Step 1. Compute the deformation gradients at $n + 1/2$ and $n + 1$

$$\boldsymbol{F}^{(i)}_{n+1} = \frac{\partial\left(\boldsymbol{X} + \boldsymbol{u}^{(i)}_{n+1}\right)}{\partial \boldsymbol{X}}; \tag{1.126}$$

$$\boldsymbol{F}^{(i)}_{n+1/2} = \frac{\partial\left(\boldsymbol{X} + \boldsymbol{u}^{(i)}_{n+1/2}\right)}{\partial \boldsymbol{X}} \tag{1.127}$$

Step 2. Compute polar decompositions at $n + 1/2$ and $n + 1$

$$\boldsymbol{F}^{(i)}_{n+1} = \boldsymbol{R}^{(i)}_{n+1} \cdot \boldsymbol{U}^{(i)}_{n+1} \tag{1.128}$$

$$\boldsymbol{F}^{(i)}_{n+1/2} = \boldsymbol{R}^{(i)}_{n+1/2} \cdot \boldsymbol{U}^{(i)}_{n+1/2} \tag{1.129}$$

Step 3. Compute the i th estimate for the spatial deformation increment over the step from the $\boldsymbol{B}$ matrix for the element, see Eq. (1.87) and Section 1.8.5.

$$\Delta\boldsymbol{\varepsilon}^{(i)} = \boldsymbol{B}^{(i)}_{n+1/2}\left(\boldsymbol{u}^{(i)}_{n+1} - \boldsymbol{u}_n\right) \tag{1.130}$$

$$\Delta \boldsymbol{D}^{(i)} \leftarrow \Delta \varepsilon^{(i)} \quad \text{(convert } 6 \times 1 \text{ vector to symmetric tensor)} \tag{1.131}$$

This procedure, as compared to the more conventional scheme using Eqs. (1.104) and (1.106), provides a straightforward method to utilize the $\overline{\boldsymbol{B}}$ formulation (to replace $\boldsymbol{B}$) for finite strains thereby reducing volumetric locking in the element.

Step 4. Rotate the increment of spatial deformation to the unrotated configuration

$$\Delta \boldsymbol{d}^{(i)} = \boldsymbol{R}^{(i)T}_{n+1/2} \cdot \Delta \boldsymbol{D}^{(i)} \cdot \boldsymbol{R}^{(i)}_{n+1/2} \tag{1.132}$$

Step 5. The terms of the symmetric tensor $\Delta \boldsymbol{d}^{(i)}$ define the strain increments for use in a conventional small–strain model. Invoke the small–strain model to provide the i th estimate for the *unrotated* Cauchy stress at $n + 1$

$$\boldsymbol{t}^{(i)}_{n+1} \leftarrow \mathcal{C}\left(\boldsymbol{t}_n, H^j_n, \boldsymbol{q}_n, \Delta \boldsymbol{d}^{(i)}\right) \tag{1.133}$$

where $\mathcal{C}$ denotes the small–strain integration process (typically, an elastic–predictor, return mapping algorithm). The integration process requires the material state at n: the unrotated Cauchy stress ($\boldsymbol{t}_n$), a set of scalar state variables denoted by H^j_n, and a set of tensorial state variables denoted by $\boldsymbol{q}_n$ which are maintained on the unrotated configuration in the model history data.

Step 6. The unrotated Cauchy stress at $n + 1$ is transformed to the Cauchy stress at $n + 1$ required for subsequent computation of element internal forces

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{R}_{n+1} \boldsymbol{t}_{n+1} \boldsymbol{R}^T_{n+1} \tag{1.134}$$

Key advantages of the above steps are the absence of half–angle rotations applied to stresses (and tensorial state variables) found in co–rotational rate formulations, Eqs. (1.114) and (1.115), and most importantly, the ability to use an existing small–strain constitutive model for *Step* 5 without modification since all quantities are referred to the unrotated configuration. The disadvantage is the need to perform two polar decompositions for the stress update at each material (Gauss) point.

Consistent Tangent Operators

Tangent operators, denoted here by $\boldsymbol{E}$, are needed to form new element stiffness matrices for the i th Newton iteration during solution for step n+1 as expressed in Eqs. (1.93) and (1.96). The operators couple increments of the spatial deformation tensor expressed on the current configuration with increments of the spatial Cauchy stress required by the fully updated formulation adopted in WARP3D. Because the incremental–iterative Newton solution at the global level uses *finite* increments of quantities to advance the solution from n to n+1, rather than simple rates $\times$ dt, the tangent operators should provide incremental, *secant* relationships.

For plasticity models implemented in a small–strain setting, Simo and Taylor [75] presented the first formalized procedures to develop the (secant) relationships and coined the phrase *consistent tangent operator*. For small–strains, *consistency* implies that the finite stress increment predicted by the tangent operator, $\boldsymbol{E}^c$, acting on a finite strain increment matches (to first order), the stress increment determined by the procedures used to integrate the plasticity rate equations over the step, i.e.,

$$\boldsymbol{\tau}^{(i)}_{n+1} = \boldsymbol{\tau}_n + \boldsymbol{E}^c : \left(\boldsymbol{\varepsilon}^{(i)}_{n+1} - \boldsymbol{\varepsilon}_n\right) = \boldsymbol{\tau}_n + \int_{t_n}^{t_{n+1}} \dot{\boldsymbol{\tau}} dt \tag{1.135}$$

where τ denotes the stress measure in the small–strain setting.

In the finite–strain framework adopted for WARP3D, the notion of a consistent tangent operator for the stress–update procedure on the unrotated configuration follows directly as (in matrix–vector form)

$$\{\boldsymbol{t}\}^{(i)}_{n+1} = \{\boldsymbol{t}\}_n + \{\Delta\boldsymbol{t}\}^{(i)} = \{\boldsymbol{t}\}_n + [\boldsymbol{E}^*]^{(i)}_{n+1}\left\{\Delta\boldsymbol{d}^{(i)}\right\} \tag{1.136}$$

where the * denotes the 6×6 consistent tangent operator defined on the unrotated configuration and the vector form of the symmetric, unrotated deformation tensor, $\Delta\boldsymbol{d}^{(i)}$, is used.

The needed form of the above relation for the fully updated solution strategy, expressed by Eq. (1.93), is

$$\sigma^{(i)}_{n+1} = \sigma_n + \boldsymbol{E}^c : \left(\varepsilon^{(i)}_{n+1} - \varepsilon_n\right) = \sigma_n + \int_{t_n}^{t_{n+1}} \dot{\sigma}\, dt \ . \tag{1.137}$$

where the spatial rate of Cauchy stress is integrated over $n\rightarrow n+1$. Using the Green–Naghdi rate of Cauchy stress from Eq. (1.115), the above expression becomes

$$\sigma^{(i)}_{n+1} = \sigma_n + \boldsymbol{E}^c : \left(\varepsilon^{(i)}_{n+1} - \varepsilon_n\right) = \sigma_n + \int_{t_n}^{t_{n+1}} \left(\hat{\sigma} + \boldsymbol{\Omega}\sigma - \sigma\boldsymbol{\Omega}\right) dt \ . \tag{1.138}$$

Simo and Hughes [77] and Cuitino and Ortiz [18] discuss the difficulty of constructing the consistent tangent operator implied above by $\boldsymbol{E}^c$ which includes potentially large–rotation effects over the step coupled with material stress increments caused by the deformation increment.

In the following we use a variation of the approximate linearization to define the transformation $[\boldsymbol{E}^*] \rightarrow [\boldsymbol{E}]$ employed in the NIKE codes and in ABAQUS. Computational experience indicates the procedure is quite robust and maintains good rates of convergence in the Newton iterations. We drop the iteration indicator (i) for simplicity and we use the vector form, $\Delta\varepsilon$, of the symmetric, spatial deformation tensor, $\Delta\boldsymbol{D}$. A mix of tensor and matrix–vector operations provides the most straightforward presentation.

The relationship between the tensor forms of the spatial deformation rate and the unrotated deformation rate, Eq. (1.112), is re–written in matrix–vector form as (using standard conversion of the rotation operation from tensor to matrix format)

$$\{\dot{\varepsilon}\} = [\boldsymbol{T}]\{\boldsymbol{d}\} \tag{1.139}$$

where the 6×6 matrix $[\boldsymbol{T}]$ is defined using $\boldsymbol{R}_{n+1}$. The terms of $[\boldsymbol{T}]$ are given by

$$[\boldsymbol{T}] = \begin{bmatrix} R_{11}^2 & R_{12}^2 & R_{13}^2 & 2R_{11}R_{12} & 2R_{13}R_{12} & 2R_{11}R_{13} \\ R_{21}^2 & R_{22}^2 & R_{23}^2 & 2R_{21}R_{22} & 2R_{23}R_{22} & 2R_{21}R_{23} \\ R_{31}^2 & R_{32}^2 & R_{33}^2 & 2R_{31}R_{32} & 2R_{33}R_{32} & 2R_{31}R_{33} \\ R_{11}R_{21} & R_{12}R_{22} & R_{13}R_{23} & (R_{11}R_{22}+R_{21}R_{12}) & (R_{12}R_{23}+R_{13}R_{22}) & (R_{11}R_{23}+R_{13}R_{21}) \\ R_{21}R_{31} & R_{32}R_{22} & R_{23}R_{33} & (R_{21}R_{32}+R_{22}R_{31}) & (R_{22}R_{33}+R_{32}R_{23}) & (R_{21}R_{33}+R_{23}R_{31}) \\ R_{11}R_{31} & R_{12}R_{32} & R_{13}R_{33} & (R_{11}R_{32}+R_{12}R_{31}) & (R_{12}R_{33}+R_{13}R_{32}) & (R_{11}R_{33}+R_{31}R_{13}) \end{bmatrix} \tag{1.140}$$

The rate of unrotated Cauchy stress, Eq. (1.117), may then be written in matrix form as

$$\{\dot{t}\} = [E^*]\{d\} = [E^*][T]^T\{\dot{\varepsilon}\} \quad . \tag{1.141}$$

where orthogonality of the rotation matrix $[T]$ is used. Note that $[E^*]$ actually used in computations is the consistent tangent operator defined by Eq. (1.136). Now the Green–Naghdi stress rate in Eq. (1.116) becomes

$$\{\hat{\sigma}\} = [T]\{\dot{t}\} = [T][E^*][T]^T\{\dot{\varepsilon}\} \tag{1.142}$$

and existing symmetries of $[E^*]$ are preserved through the $[T]$ transformation.

We invoke the relationship between the Green–Naghdi stress rate and the spatial rate of Cauchy stress rate given by Eq. (1.115). The left side of Eq. (1.115) is simply the symmetric tensor form of $\{\hat{\sigma}\}$ given above. To arrive at a tractable form for the $-\Omega\sigma + \sigma\Omega$ terms, the approximation $W \doteq \Omega = \dot{R}R^T$ is adopted. Nagtegaal and Veldpaus [62] demonstrated the validity of this approximation when the rate of logarithmic strain remains constant over the step, which is consistent with the present stress updating procedure. Moreover, they showed that the $-W\sigma + \sigma W$ terms could be re–cast in matrix form (using the $W = L - D$ decomposition with L given by $\partial v/\partial x$ in Eq. (1.104)) as

$$-W\sigma + \sigma W \rightarrow [Q]\{\dot{\varepsilon}\} \tag{1.143}$$

where the assumption of incompressibility becomes necessary to arrive at a symmetric form of $[Q]$. The terms of $[Q]$ are

$$[Q] = \begin{bmatrix} 2\sigma_{11} & 0 & 0 & \sigma_{12} & 0 & \sigma_{13} \\ 0 & 2\sigma_{22} & 0 & \sigma_{12} & \sigma_{23} & 0 \\ 0 & 0 & 2\sigma_{33} & 0 & \sigma_{23} & \sigma_{13} \\ \sigma_{12} & \sigma_{12} & 0 & \frac{1}{2}(\sigma_{11}+\sigma_{22}) & \frac{1}{2}\sigma_{13} & \frac{1}{2}\sigma_{23} \\ 0 & \sigma_{23} & \sigma_{23} & \frac{1}{2}\sigma_{13} & \frac{1}{2}(\sigma_{22}+\sigma_{33}) & \frac{1}{2}\sigma_{12} \\ \sigma_{13} & 0 & \sigma_{13} & \frac{1}{2}\sigma_{23} & \frac{1}{2}\sigma_{23} & \frac{1}{2}(\sigma_{11}+\sigma_{33}) \end{bmatrix} \tag{1.144}$$

By expressing each term of Eq. (1.115) in matrix–vector form, the spatial rate of Cauchy stress is given by

$$\{\dot{\sigma}\} = \left[[T][E^*][T]^T - [Q]\right]\{\dot{\varepsilon}\} = [E]\{\dot{\varepsilon}\} \quad . \tag{1.145}$$

This expression defines the finite strain–rotation form of the tangent operator for use in construction in the element tangent stiffness in Eq. (1.96). This form is not a true consistent operator as the kinematic transformation uses the *rate* expressions at n+1 rather than the secant relationship from n to n+1. Use of the constitutive consistent $[E^*]$ seems to be far more important for convergence.

The tangent operator defined in Eq. (1.145) appears in the NIKE–2D and NIKE–3D (implicit) codes which also adopt a Green–Naghdi stress rate and stress updating procedure followed here. However, the $[Q]$ term is omitted in forming the element tangent stiffness such that $[E] \doteq [E^*]$. Our numerical experiments indicate that inclusion of $[Q]$ is essential to maintain quadratic rates of convergence in the global Newton iterations when large portions of the model undergo nearly homogeneous deformation. In other instances, $[Q]$ may be omitted as in the NIKE codes without a detrimental effect on convergence rates.

Polar Decomposition

The polar decomposition F=RU is a key step in the stress–updating algorithm and must be performed twice for each Gauss point for each stress update, i.e., at $n + 1/2$ and $n + 1$.

The computational effort required for the polar decomposition should be insignificant relative to the element stiffness computation and the equation solving effort. For their explicit code, Flanagan and Taylor [23] developed an algorithm for the integration of $\dot{\boldsymbol{R}} = \boldsymbol{\Omega R}$ that maintains orthogonality of $\boldsymbol{R}$ for the very small displacement increments characteristic of explicit solutions. Numerical tests readily show their procedure fails for large displacement increments experienced with implicit global solutions. The following algorithm removes such approximations by providing an exact construction of $\boldsymbol{R}$ and $\boldsymbol{U}$ for arbitrary size load steps and yet remains computationally very efficient with the framework of an implicit solution.

Step 1. Compute the right Cauchy–Green tensor

$$\boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F} \tag{1.146}$$

and its square

$$\boldsymbol{C}^2 = \boldsymbol{C}^T\boldsymbol{C} \tag{1.147}$$

where only the upper–triangular form of the symmetric products (6 terms) are actually computed and stored.

Step 2. Compute the eigenvalues λ_1^2, λ_2^2 and λ_3^2 of $\boldsymbol{C}$. A Jacobi transformation procedure specifically designed for 3×3 matrices is used to extract the eigenvalues. For scalar computers, the do–loops are eliminated by explicitly coding each off–diagonal rotation form. Two or, at most, three sweeps are needed to obtained eigenvalues converged to a 10^{-6} tolerance. The procedure vectorizes easily since there are no transcendental functions to evaluate; the number of iterations is fixed at two or three for all material points in a contiguous block of elements.

Step 3. Compute invariants of $\boldsymbol{U}$ and the det($\boldsymbol{F}$)

$$I_U = \lambda_1 + \lambda_2 + \lambda_3 \tag{1.148}$$

$$II_U = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_1\lambda_3 \tag{1.149}$$

$$III_U = \lambda_1\lambda_2\lambda_3 = J = det(\boldsymbol{F}) \tag{1.150}$$

Step 4. Form the upper triangle of the symmetric, right stretch, $\boldsymbol{U}$, and it's symmetric inverse, $\boldsymbol{U}^{-1}$ (see Hoger and Carlson [36])

$$\boldsymbol{U} = \beta_1(\beta_2\boldsymbol{I} + \beta_3\boldsymbol{C} - \boldsymbol{C}^2) \tag{1.151}$$

where $\mathbf{I}$ denotes a unit tensor with the β coefficients defined by

$$\beta_1 = 1/(I_U II_U - III_U), \quad \beta_2 = I_U III_U, \quad \beta_3 = I_U^2 - II_U \tag{1.152}$$

Similarly, the inverse of $\mathbf{U}$ may be formed directly as

$$\boldsymbol{U}^{-1} = \gamma_1(\gamma_2\boldsymbol{I} + \gamma_3\boldsymbol{C} + \gamma_4\boldsymbol{C}^2) \tag{1.153}$$

where the γ coefficients defined by

$$\gamma_1 = 1/III_u(I_U II_U - III_U), \quad \gamma_2 = I_U II_U^2 - III_U(I_U^2 + II_U), \tag{1.154}$$

$$\gamma_3 = -III_U - I_U(I_U^2 - 2II_U), \quad \gamma_4 = I_U \tag{1.155}$$

Step 5. Form $\boldsymbol{R}$ as the product

$$\boldsymbol{R} = \boldsymbol{F}\boldsymbol{U}^{-1} \tag{1.156}$$

Chapter 2

Model Definition

This chapter describes the commands to define a finite element model, to define a nonlinear/dynamic solution algorithm, to request an analysis for a number of load steps and to request output. Commands in this chapter are described in the recommended order of input:

- structure name and sizes (number of nodes and elements)
- definition of "materials" for association with elements in the model. Materials provide linear elastic properties, material density, nonlinear properties and a "type" of constitutive algorithm, e.g., rate–dependent Mises plasticity with isotropic hardening.
- the type of each finite element in the model, the kinematic formulation for the element (large or small displacements) and the values of any properties for the element, e.g., the order of numerical integration
- the *X–Y–Z* coordinates for all model nodes in the model global coordinate system
- the incidences for all elements in the model. Incidences define the connectivity of element nodes to model nodes
- the assignment of contiguous lists of elements to "blocks" for analysis. Blocking is required to support vector/parallel operations on supercomputers and is retained for analyses conducted on Unix workstations. All elements in a block must be the same type, have the same material, the same type of kinematic formulation, the same values of element properties and must not be connected to a common node.
- displacement constraints imposed on nodes of the model, either zero or non–zero.
- loading patterns for the model. Loading patterns consist of nodal forces and element loads which are converted to equivalent nodal forces.
- a nonlinear/dynamic loading which defines the increment of load to be applied during each load/time step. Loading increments for a step are defined using the loading patterns.
- parameters to control the nonlinear/dynamic solution process, e.g., the time increment for dynamic analysis, the type of equation solver (direct, conjugate gradient), the maximum number of Newton iterations.
- a request to compute displacements for a list of load steps
- a request to output computed nodal and element results. Results for use by humans are directed to the current output device with appropriate pagination, headers, labels, etc.
- a request to output computed nodal and element results in the format defined by the PATran modeling software. These results files are readable by Patran without further conversion.
- a request to compute and output values for the *J*–integral in fracture mechanics models
- a "save" computed to write all current model data and results to a sequential binary file for later use to re–start an analysis.
- a "stop" command to terminate program execution.

In typical analyses, multiple compute, output, *J*–integral and save commands appear in the input. Parameters to control the nonlinear/dynamic solution algorithm, e.g., the time step, may be modified between analyses for sets of load steps. Constraints can be modified between analyses for load steps to effect incremental changes in the boundary conditions.

Some model descriptors cannot be modified once defined. For example, the number of nodes and elements, the element types and properties, the coordinates, the incidences and the blocking cannot be altered.

2.1 Model Name and Sizes

The definition of a new finite element model begins with specification of an alphanumeric identifier. The identifier appears on all pages of output. The command has the form

structure < name: label >

The first eight characters of model names are recognized. Longer names are accepted on the command but truncated to the eight character limit.

The number of nodes and number of elements in the model must be specified prior to any other command related to nodal or elemental quantities. WARP uses the specified sizes to support checking of the input data as it is entered and to support exhaustive consistency checking of the structural model for errors prior to the first compute request. An example of such an error is a node with no elements attached. The model sizes are defined with a command having the form

number (of) [{ nodes / elements } < size: integer (,) >]

Examples of the above commands are:

```
structure bend_strip
  number of nodes 3450 elements 4230
```

and

```
structure bend_strip
  number of nodes 3450
  number of elements 4230
```

All node and element identifiers are positive integers beginning with the value 1. Nodes and elements must each be numbered sequentially.

Once specified, the number of nodes and elements cannot be modified through user commands.

Limits on Number of Nodes and Elements

The maximum number of nodes and elements permitted in a model varies with the version of WARP being executed and the computer executing the program. Typical limits are 10,000 nodes and 10,000 elements for a Unix workstation version and 30,000 elements and 30,000 nodes for a Cray version. These limits are easily changed through one line in the source code followed by a re–compilation on the hardware platform.

2.2 Material Definitions

Finite elements in a model are associated with "materials" from which they derive elastic properties, mass density and nonlinear characteristics, if necessary. Through the *material* command, the user specifies a convenient name for the material, the type of constitutive model (e.g., rate–dependent Mises) and the values of any properties required by the material model. Material definitions must precede the specification of element properties during input.

Some models provide an option to specify nonlinear response in the form of a piecewise–linear description, i.e., a tensile stress–strain curve. The *stress–strain curve* command is used to describe points on the piecewise–linear curve for use by the material model.

This section describes the *material* and *stress–strain curve* commands. When a *material* command references a *stress–strain curve*, there is no requirement that the referenced curve be defined previously. Consistency checks are performed prior to any computations.

2.2.1 Material Command

A *material* command on a separate line initiates the material definition sequence. Any number lines may follow to define the properties required for the material model. The definition of an element requires the following information:

The command syntax is

material < material id: label >

properties < model type: label > [< matl. prop: label > (< value >)]

The logical input line for the properties may be continued over multiple physical input lines with commas at any point. Subsequent sections in Chapter 3 define the "type" of material models currently available and the properties required for each model type.

An example of material specification is

```
material al2024t
  properties mises e 10350 nu 0.3 yld_pt 50.0 n_power 10,
                  rho 0.1254e-07
```

In this example, the material is named "al2024t" and the computational model for the material is "mises" (one of the models described in Chapter 3). Keywords "e", "nu", "n_power" are properties of the *mises* model assignable by the user.

The following example refers to *stress–strain curve 3* for a piecewise–linear description of the uniaxial, tensile stress–strain curve

```
material a36
  properties mises e 30000 nu 0.3 curve 3 rho 0.1254e-07
```

Once defined, the specification for a material cannot be modified at any further point in the analysis.

2.2.2 Stress–Strain Curve Command

The uniaxial, tensile stress–strain response of certain materials requires a general segmental curve description for a realistic representation. Materials that exhibit a sharp yield point, a Luder's band and then strain hardening are classic examples not amenable to mod-

eling with the power–law type curves. Figure 2.1 provides an example of a stress–strain curve described with a piecewise–linear model.

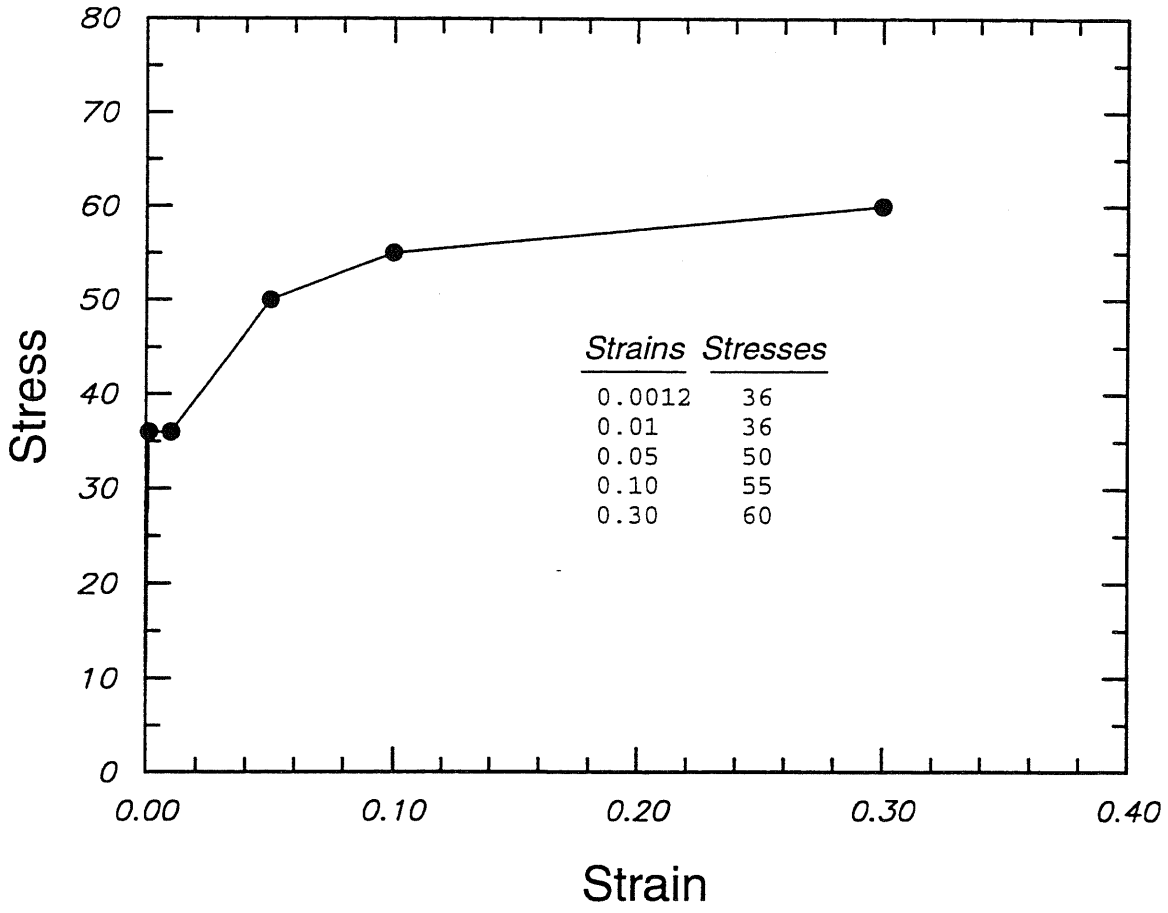


FIG. 2.1—*Example of piecewise–linear stress–strain curve.*

Points on such curves are specified with a simple command sequence *stress–strain curve* where each such curve required in the analysis is assigned an integer number for identification. The curve may then be referenced in a *material* command as described above. The command syntax is

stress(–strain) (curve) < curve number: integer >

[< strain value: numr > < stress value: numr > (,)]

Curve points are input as strain–stress pairs; use as many lines as needed to specify the points. Multiple pairs may be specified on a line. All strain–stress values must be positive. *Do not specify the (0,0) point on the curve.* The first point defines the yield strain and yield stress. Young's modulus specified in the *material* command *must* match the value implied by the yield strain–yield stress pair. After the last specified point, the response is assumed perfectly plastic.

The strain values input here are the *total* strains (not the plastic strains!). For large–deformation analyses, the values should correspond to the logarithmic strain–Cauchy stress; for small strain–analyses the values should be engineering strain–engineering stress.

A maximum of 5 curves may be specified for use in an analysis. Each curve may have up to 10 strain–stress pairs defined.

The above curve is described with the command sequence

```
stress-strain curve 3
  0.0012 36,  0.01 36,  0.05 50,
  0.10 55, 0.30 60
```

2.3 Element Types and Properties

The types of finite elements and their properties are specified prior to any compute requests. An *elements* command on a separate line initiates the element definition sequence. Any number lines may follow to define the types and properties of all elements in the model. The definition of an element requires the following information:

- the "type" of element (only *l3disop* is currently available)
- the kinematic formulation (small or large displacements)
- reference to a previously defined "material" that defines elastic properties, mass denisty and the nonlinear properties (if required)
- a list of element property identifiers and associated values, e.g., the order of numerical integration.

The command syntax is

elements

< element nos.: list > type < element type: label > { linear | nonlinear } (,)

material < matl. id: label > [< elem. prop: label > < value >]

The logical input line may be continued over multiple physical input lines with commas at any point. Subsequent sections in Chapter 3 define the "type" of elements currently available and the properties available for each element type. Element properties typically have a property keyword followed by a value. Some element properties are "logical" values which take on "true" values by the presence of the keyword.

The keyword *linear* requests a conventional small displacement, small strain element formulation. This is the default formulation and is adopted if no specification is given. The keyword *nonlinear* requests a geometric nonlinear formulation that models large rotations and finite strains.

Every element must have an associated *material*. Materials must be specified prior to their use in element specification.

An example of elements specification is

```
elements
  1-40 type l3disop linear material a36 center_output bbar,
                        order 2x2x2
  500-1000, 1200-200 by -2 l3disop nonlinear material al_2024t,
                        order 2x2x2 long
```

Once defined, the specification for an element cannot be modified at any further point in the analysis.

2.4 Nodal Coordinates

The coordinates of nodes are specified relative to the global Cartesian reference axes. During model definition, the command *coordinates* initiates the translation of nodal coordinate data. Any number of *coordinates* commands may be given prior to a *compute* request. The existing coordinates for nodes are simply overwritten by any newly specified values. The input syntax is

coordinates

< node number: integer > [{ x, y, z } < coord. value: number > (,)]

< node number: integer > [< coord. value: number > (,)]

where the second form applies the default ordering of entries *X–Y–Z*. When using the second form, coordinates not specified take on the last previously defined values. For example, the sequence

```
coordinates
  4 3.2 5.2 6.4
  10 4.1
```

defines the *Y* coordinate of node 10 as 5.2 and the *Z* coordinate of node 10 as 6.4.

The default *X–Y–Z* ordering for the second input form may be modified by the *default* command

coordinates

default [{ x, y, z }]

< node number: integer > [< coord. value: number > (,)]

where any number of *default* commands may be given.

Some examples illustrating various options to define nodal coordinates are given below.

```
coordinates
  4 x 2.5 y 3.0 z 4.1
  10 z -20 y 40 x 20
  11 -5.23 6.23
  default z y x
    3  15.3  14.2   10.5
```

```
default x y z
  10 -13.5 10.5 -20.4
```

At any point during input of the coordinates, the *dump* command is available to request a listing of current coordinates for all nodes of the model.

```
coordinates
  4 x 2.5 y 3.0 z 4.1
  10 z -20 y 40 x 20
  11 -5.23 6.23
  dump
  default z y x
    3  15.3  14.2   10.5
  default x y z
    10 -13.5 10.5 -20.4
  dump
```

2.5 Element Incidences

Each node of an element in the model must be "mapped" onto the corresponding global node. Element *incidences* establish this correspondence. During model definition, the command *incidences* initiates the translation of element incidence data. Any number of *incidences* commands may be given prior to a *compute* request. The existing incidences for elements are simply overwritten by any newly specified values. The input syntax is

incidences

< element number: integer > [< global node i: integer list > (,)]

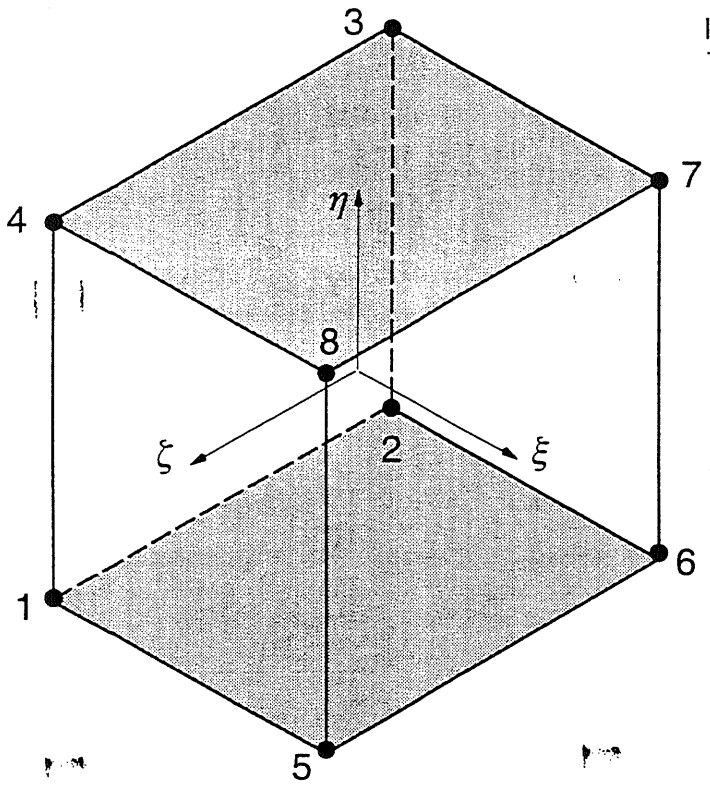
where <global node i> denotes the number of the global node to which the element node is attached. Note that the list of global node numbers may be specified as an integer list.

An example of the incidences command is

```
incidences
  1    13-20
  2    5 40 65 83 92 120 44 98
  3    140-144 178 162 183
```

The number of entries in the integer list must equal the number of nodes on the element (8 for *l3disop*). Error messages are issued by the input processor if the number of nodes is less than required, if a node number exceeds the number of structure nodes, etc. A warning message is issued if the same node appears more than once in the integer list.

The ordering of nodes on the *l3disop* element is shown in Fig. 2.2. Isoparametric coordinates for the nodes and Gauss points are given for reference.



Isoparametric Coordinates of Nodes

Node	ξ	η	ζ
1	–1	–1	1
2	–1	–1	–1
3	–1	1	–1
4	–1	1	1
5	1	–1	1
6	1	–1	–1
7	1	1	–1
8	1	1	1

Coordinates of Gauss Pts.

Point	ξ	η	ζ
1	–a	–a	–a
2	–a	a	–a
3	a	–a	–a
4	a	a	–a
5	–a	–a	a
6	–a	a	a
7	a	–a	a
8	a	a	a

a=0.57735

FIG. 2.2—*Local node ordering for the 8–node isoparametric element "l3disop." Isoparametric coordinates for the element nodes and Gauss points are listed.*

2.6 Element Blocking

All element level computations in WARP proceed on a block–by–block basis to facilitate vectorization and parallel processing of element blocks. Each element must be assigned to a block. The maximum number of elements in a block is set by a compile–time variable in the WARP source code and is selected to optimize performance on specific types of computers. On a CRAY–90, for example, the block size is 128 since the vector processor units have registers each of length 128 words. On a Unix workstation, the block size impacts the efficient use of cache memory; large blocks cause severe thrashing in the cache. A typical block size for a workstation is 32. Blocking dramatically improves computational performance of the code even on Unix workstations without vector hardware. Very efficient subroutines to perform common vector–matrix operations available on workstations provide the improved performance during element level operations.

The assignment of elements to blocks is most conveniently handled by the pre–processor software employed to create the finite element model. The *patwarp* program, for example, converts a Patran neutral file into a WARP input file and performs the element–to–block assignments. The block assignment commands have the form

blocking

< block : integer > < block size: integer > < first element in block: integer >

The following example input describes the blocking for a model with 520 elements.

```
blocking
     1   120     1
     2   112   121
     3   109   233
     4   100   342
     5    42   442
     6    24   484
     7    11   508
     8     2    519
```

The following rules define the proper assignment of elements to blocks. All elements in a block

- must be sequentially numbered
- must be the same type; e.g., *l3disop*
- must have the same kinematic formulation (*linear* or *nonlinear*)
- must have the same associated material
- must have the same integration order (e.g. $2 \times 2 \times 2$)
- must not share a common node if:
  - execution is on a vector/parallel computer (Cray, Convex)

  or

  - the Hughes–Winget pre–conditioner is selected for the conjugate gradient solver

This last requirement nearly always necessitates a re–numbering of the elements in the model to eliminate node conflicts within blocks. The *patwarp* pre–processor, for example, employs a simple "red–black" strategy to re–number elements before constructing the WARP input file.

The input translators perform checks to insure that blocking assignments follow these rules.

2.7 Nodal Constraints

WARP currently supports constraints applied to nodes in the global, Cartesian system. The sequence is initiated with the *constraints* command. When the *constraints* command is encountered by the input translators, all previously defined constraints are destroyed. Thus to modify constraints between load (time) steps, all the constraints must be specified — not just the constraints that have changed. The input syntax is

constraints

< node list: list > [{ u v w } (=) < constraint value: number > (,)]

Examples of constraints input are:

```
constraints
   1-100 by 3 w 4.3 v 0 u 0
   24 u = -1.3 w 0.0
```

In a nonlinear analysis, the currently defined constraints are interpreted as the *incremental displacement change* enforced over the next load (time) step. A non–zero constraint is enforced during the first iterative cycle for the load step. In subsequent iterations, no displacement change is permitted on the constrained displacements to maintain the value of the specified increment.

The *dump* command may be specified to request a display (listing) of the current constraints information taken from internal tabels.

2.8 Loads

Loads may be applied to the nodes and elements of a model. Element loads, which are dependent on the type of finite element, are converted to equivalent nodal loads by element processing routines. Nodal loads and element loads are grouped together to define *loading patterns*. The loading patterns define the spatial variation and reference amplitudes of loads on a model. Examples of loading patterns include dead load, an internal pressure and simple bending of a component. WARP does not currently provide a thermal loading capability.

Once loading patterns are defined, a *nonlinear* loading condition is defined. The term *dynamic* may be used as a synonym for *nonlinear* if desired. A nonlinear/dynamic loading consists of a sequential number of load steps. An incremental–iterative solution is obtained for each load step. For dynamic analyses, a load step is the same as a time step. Each *load step* may consist of loading patterns combined with scalar multipliers. The scaled values of nodal forces (nodal loads and resulting equivalent nodal loads) for the patterns are applied as the new *incremental* load to the model during the step.

A static linear analysis must be performed as the first step of a static nonlinear analysis. A static nonlinear analysis is solved as a dynamic analysis with: (1) a very large time increment or (2) zero mass for the model. The user selects one of the two procedures by setting the time increment and the model mass.

The first sections describe the commands to define nodal forces and element loads that construct a loading pattern. Commands are then defined to specify load steps in a nonlinear/dynamic analysis (or step 1 of a static, linear analysis).

2.8.1 Loading Patterns

A new loading pattern is defined through a command of the form

loading < loading identifier: label >

where the loading identifier is used in subsequent commands to identify the loading, for example, in compute and output requests. Only the first eight characters of the identifier are processed; all loading patterns must have unique identifiers.

When an existing loading pattern is referenced in this command, newly specified node and element loads are *added* to the previously specified loads for that loading pattern. If the command references an existing nonlinear loading condition, the previously defined information for all steps is destroyed and replaced by the newly specified input.

2.8.2 Nodal Loads

A sequence of nodal load definitions has the form

nodal (loads)

$$\text{< node list: list >} \left[\begin{Bmatrix} \text{force\_x} \\ \text{force\_y} \\ \text{force\_z} \end{Bmatrix} \text{(=) < force value: number > (,)} \right]$$

Nodal loads are additive; if the same node and direction appear in two different loading commands the sum of two loads is applied to the model. An example sequence to define a loading condition and a set of nodal forces is

```
loading unit_pull
  nodal loads
     1-40 60-90 force_z -2.3 force_x 14
     3240 3671 4510-5000 force_z -3.12
     35 force_x 2
```

In the above example, node 35 has a total force in the *X*–direction of 16 (14 from the first line + 2 from the last line).

2.8.3 Element Loads (not yet implemented)

A sequence of element load definitions has the form

element (loads)
< elements: list > < type of element loading >
< elements: list > < type of element loading >
•
•

where the <type of element loading> is either a body force or a face traction. The types of element loads and commands to define them are dependent on the type of element. Refer to Chapter 4 for this information.

When the analysis includes geometric nonlinear effects (large displacements), equivalent loads for the incrementally applied surface tractions are re–computed at the beginning of each load step using the current (deformed) geometry of the elements.

Nodal loads are always applied in the global coordinate system and are thus unaffected by the deformed geometry.

2.8.4 Step Loads

The loading type designated *dynamic* or *nonlinear* defines the combinations of pattern loads for each time step in a dynamic analysis or each load step in a static nonlinear analysis. These commands have the form

loading < loading identifier: label >
nonlinear
steps < steps: list > [< pattern id: label > < multiplier: number > (,)]

where the keyword *dynamic* may be substituted as a synonym for *nonlinear*. Nodal and element loads cannot be specified within a nonlinear/dynamic loading definition above. The multiplier value must follow each pattern id— a multiplier value is required input. As indicated, multiple pattern loads may be combined with different multipliers to define a load increment for a time step in a dynamic analysis or a load step in a static nonlinear analysis. An example of this command sequence is

```
loading crush
  nonlinear
     steps 1-10 unit_pressure 2.3 unit_tens -1.2
     steps 11-200 pull 0.2
```

where the loading patterns *unit_pressure*, *unit_tens* and *pull* have been defined previously. Although the steps are defined in ascending sequence in the above example, the steps may be defined in any order; the final set of steps must comprise a sequential list.

Modifying Step Definitions

During the course of a nonlinear or dynamic analysis, it is often necessary to define additional steps or to modify the definition of steps yet to be analyzed. For example, previously defined, but unsolved, load steps may need to have a reduced multiplier based on current convergence properties.

Two approaches are available to perform this task. In the first approach, a new nonlinear/dynamic loading condition may be defined with the desired definition for the new/modified load steps. Subsequent compute requests then refer to this new loading. In the second approach, the existing nonlinear/dynamic loading condition is redefined. The input translators require that all load steps 1, 2, 3, ... be re–defined with this approach. A warning message is issued to the user about this feature when an existing nonlinear/dynamic loading condition is redefined.

2.8.5 Displacement Control Loading

A nonlinear/dynamic loading condition with appropriate step definitions must be always be specified for a model. This becomes a slight inconvenience when the model is loaded only by imposed non–zero displacements at selected nodes. The recommended procedure for displacement control loading is:

- define a loading pattern "dummy" with a meaningless, zero nodal force (put a force of 0.0 on one node)
- define the nonlinear/dynamic loading condition. All steps refer to the loading pattern "dummy" with a multiplier of 1.0.

This procedure forces the processing routines to create the necessary internal data structures required for an analysis. An example of these commands is

```
loading dummy
  nodal loads
      1 force_x 0.0

loading crush
  nonlinear
      steps 1-100 dummy 1.0
```

Effects of Step Multipliers

The pattern multiplier (1.0 above) plays no role in the solution of displacement control loadings unless the *extrapolate* option of the nonlinear solution algorithm is invoked (extrapolate is *on* by default). When the extrapolate option is in effect, the incremental displacements computed from the solution over step n–1 to n are scaled and applied to the model to start the iterative (Newton) solution from n to n+1. The displacement scaling factor is computed from the specified step multipliers for steps n (say f_n) and n+1 (f_{n+1}) as f_{n+1}/f_n. Thus only the ratios of the multipliers are significant for displacement control with *extrapolate on*. When the non–zero constraints are modified during a displacement control analysis, the loading step multipliers must be modified accordingly by the user; otherwise the extrapolation ratio (f_{n+1}/f_n) is computed incorrectly.

To illustrate, consider the following example. Non–zero constraints are specified to load the model. The dummy loading pattern and nonlinear loading are defined as above with

step multipliers of 1.0 for load steps 1–10. After step 10, the user modifies the constraints to reduced the imposed increment (uniformly) by one–half, possibly to reduce the number of Newton iterations for convergence in subsequent steps. Load steps 11, 12, 13, ... must have a multiplier of 0.5 for correct extrapolation. In step 11, the extrapolation multiplier is 0.5/1.0=0.5 while in steps 12, 13, ... the multiplier again becomes 1.0.

2.9 Solution Parameters

The nonlinear (and dynamic) computational procedure in WARP follows an incremental–iterative strategy with full Newton iterations to eliminate residual nodal forces caused by nonlinear behavior. The user has full control over the solution procedures through a wide range of parameters. Each of the parameters has a built–in default value which may be re–defined by the user. The values of these parameters are declared by the user before computation begins for the first load step; those values remain in effect unless modified by the user as the solution progresses through the load steps. New values for these parameters may be defined whenever the input translators accept new input lines. The most current values of the parameters then control subsequent computations over load steps.

The specification of solution parameters begins with a command of the form

$$\left\{\begin{array}{l}\underline{\text{solution}}\\ \underline{\text{nonlinear}}\\ \underline{\text{dynamic}}\end{array}\right\}\ (\underline{\text{analysis}})\ (\underline{\text{para}}\text{meters})$$

and terminates whenever a command is given that does not define a parameter controlling the analysis. The following sections describe each of the parameters assignable by the user and the command syntax. An example defining values for selected solution parameters is shown below for reference.

```
dynamic analysis parameters
   solution technique lnpcg
   preconditioner ebe
   lnr_pcg conv test res tol 0.01
   maximum linear iterations 2000
   maximum iterations 10
   convergence test norm res tol .5
   time step 0.05
   trace solution on
   linear stiffness iteration one off
c
c
compute displacements for loading dead_live step 1-5
```

2.9.1 Linear Equation Solver

The linearized set of equilibrium equations for the model is solved by one of two computational procedures. The first is a "direct" solver which assembles the upper–triangular stiffness matrix for the model (in skyline format) and executes a conventional Choleski factorization, forward load pass and backward load pass. The second is an iterative, element–by–element, linear preconditioned conjugate gradient solver (LPCG). This solver does not assemble the structural stiffness matrix and thereby greatly reduces memory requirements. A choice of two preconditioners is available: (1) a diagonal preconditioner which employs the diagonal terms of the dynamic stiffness for the model, and (2) Crout factorization of the "regularized" dynamic tangent stiffness (implemented on an element–by–element basis as outlined by Hughes–Winget).

Direct Solver

The direct solver provides an "exact" solution for the linearized equations within round–off features of the computer hardware. The direct solver is recommended for all problems smaller than a few hundred nodes and for all problems in which 3–D elements model a

plane–strain or plane–stress configuration. Such models have very large in–plane dimensions and only one element in the thickness direction. Memory requirements for the assembled stiffness are relatively small for such models (usually less than 30 MB) and they have a small profile/bandwidth. The vectorized implementation of the direct solver maintains very high efficiency under these conditions.

The direct solver becomes inefficient very quickly as the 3–D nature and size of the model increases— in terms of both required memory for the assembled stiffness and the factorization time. Memory space required by the direct solver is dynamically allocated at execution time and can easily exceed 300–500 MB for large 3–D models. Even if the computer hardware has sufficient real memory to eliminate virtual memory paging for problems of this size, the CPU time required for factorization often exceeds greatly the CPU time require for solution with LPCG solver.

No user assignable parameters are available for the direct solver. The direct solver is the default computational procedure in WARP and is explicitly specified with the command

solution (technique) direct

Conjugate Gradient Solver

The linear preconditioned conjugate gradient (LPCG) solver iteratively improves an initial estimate for the solution of the linearized equilibrium equations. The iterations continue until further changes in the displacement increments yield no significant improvement in the solution.

The element–by–element implementation of the LPCG solver eliminates construction of the assembled stiffness matrix. Memory requirements for the LPCG solver are thus many times smaller than for the direct solver. A 7,000 element/node model runs without (virtual memory) paging on a 64 MB Unix workstation.

The number of LPCG iterations required to converge on the correct displacement increment varies with the characteristics of the model. The very best convergence rate derives from a model of uniformly (cube) sized elements arranged in a cube. In this case, the diagonal preconditioner (DPC) provides a solution in a number of LPCG iterations less than the square root of the number of active nodal degrees of freedom (dof). The DPC performs exceptionally well in dynamic analyses with small time increments. Some models that exhibit very poor LPCG convergence with DPC in static loading converge very rapidly in dynamic loading. For non–uniform element sizes, large time increments in dynamic analyses, decreased "three–dimensionality" of the model and increased nonlinearity, the number of LPCG iterations may exceed 3–5 $\times$ (active no. of dof)$^{1/2}$.

Fracture mechanics models with focused meshes and orders of magnitude variations in element sizes define a very difficult configuration for the LPCG solver. For models at the extremes of these conditions, a solution may not be possible with the DPC. The Hughes–Winget preconditioner (HWPC) is available for LPCG solution in these models. The HWPC increases the computational cost per LPCG iteration by a factor of $\approx$ 2.1–2.3 $\times$ the cost per DPC iteration. The HWPC produces a converged solution in nearly all cases which fail with the DPC. Moreover, when both DPC and HWPC produce solutions, the number of LPCG iterations with HWPC is often 0.3–0.4$\times$ the number of DPC iterations which yields a net reduction in total solution times. Numerical experiments with large models provide guidance on the optimum choice of a preconditioner.

The LPCG solver employs the following convergence test to assess the solution quality. Let r_0 be the residual vector for solution of the linear equations evaluated for the initial (estimated) displacement increment:

$$K_D \cdot \Delta u_0 - \Delta P = r_0$$

where K_D denotes the dynamic stiffness. The initial displacement vector Δu_0 is set to zero except those for non–zero terms of the user specified (current) constraints. The vector ΔP denotes the incremental load. During Newton iteration 1, ΔP contains the applied load over the step (including inertia effects); during subsequent iterations, ΔP contains the residual load. LPCG iterations continue until at the kth iteration with Δu_k available

$$\| r_k \| \leq (user\ tol/100) \times \| r_0 \|$$

where $\| \ \|$ denote the Euclidean norm. Tolerance values are specified in (%); thus, a user tolerance of 0.01 (%) is reasonably strict and often used. Tolerance values of 0.001–1.0 have been used successfully in various models. The user specified tolerance exerts a dramatic impact on the required number of LPCG iterations and the total CPU time. Excessively tight tolerances do not provide real improvements in solutions. If the model has a linear elastic material and a kinematically linear formulation, the convergence tests performed after the first Newton iteration of a load step provide a very good indicator of the linear solution quality. An excessively large residual indicates that a smaller LPCG tolerance value is needed.

The approximate nature of LPCG solution provides an opportunity to balance accuracy and CPU time for linear equation solving with the number of Newton iterations required to eliminate residual forces arising from nonlinear behavior. During the first few Newton iterations of a load step, excessive accuracy during solution of the linear equations is often un–warranted as the force imbalances due to nonlinearity far exceed those due to remaining errors in displacement increments from the LPCG solver. The Newton iterations correct, simultaneously, the incremental displacement vector for the step due to nonlinear effects and due to small residuals in the LPCG solver. Experimentation with LPCG and Newton tolerance values in nonlinear analyses often yields substantial decreases in total solution times.

The program terminates execution if the specified number of LPCG iterations is exceeded (the default limit is 10 iterations).

The commands to specify a LPCG solver have the form

solution (technique) lnpcg

preconditioner (type) { diagonal | hughes–winget }

lnpcg (convergence) (tests) residual (tolerance) < number >

maximum linear iterations < integer >

trace lnpcg_solution { on | off }

An example is

```
nonlinear analysis parameters
   solution technique lnpcg
```

```
preconditioner hughes-winget
lnpcg conv test res tol 0.01
maximum linear iterations 2000
   •
   •
```

During solution of a large nonlinear model, the LPCG solver with the DPC may converge very rapidly during early load steps when nonlinear effects remain small. Once the DPC requires an excessive number of LPCG iterations in later load steps, the preconditioner can be switched to *ebe*.

2.9.2 Dynamic Analysis Parameters

The time increment over each load step and the β factor for the Newmark time integration scheme are defined by the commands

time step < number >

newmark beta < number >

The default time step size is 1000000 and the default value of the Newmark β factor is 1/4.

Static analyses in WARP are achieved by using a very large time step or by setting the model mass to zero. The time step must be a positive number. By setting a realistic time step for the analysis with a zero mass, analyses for viscoplastic effects may be performed without inertia effects.

2.9.3 Newton Iteration Parameters

The nonlinear solution in each load/time step is accomplished with a full Newton iterative procedure by default. The dynamic tangent stiffness is updated prior to each equilibrium iteration and at the beginning of the step. Newton iterations are numbered 1, 2, 3, ... where the increment of applied forces and imposed displacements comprise the load vector for iteration 1. During subsequent iterations, the load vector consists of the current (total) residual forces. Users may request use of the linear–elastic stiffness for the solution of iteration 1 with a command of the form

linear stiffness (for) iteration one { on / off }

This option enhances convergence when the incremental load during the step causes inelastic unloading. The default value is *off*.

Iteration Limit

The limit on Newton iterations is defined by

maximum iterations < integer >

The default limit is 10.

Nonconvergent Solutions

By default, the program terminates execution if Newton iteration limit is reached without convergence. A restart file named <structure id>_noncnv_db is created. Users can request that program execution continue to the next load step with the command

Convergence Tests

Four types of tests are available to assess convergence of the Newton iterations. Define the following quantities:

nonconvergent solutions { stop / continue }

$\| R_k \|$	Euclidean norm of the residual force vector for the model following solution of iteration k of the step
$max[absR_k^{(i)}]$	maximum (absolute) entry in the residual force vector for the model following solution for iteration k of the step (only active dof are considered)
$\| P \|$	Euclidean norm of the total force vector applied to the model (includes reactions at constrained dof and inertia effects)
$\| \Delta u_1 \|$	Euclidean norm of the incremental displacement vector for the model computed during iteration 1 of the load step
$\| \Delta u_k \|$	Euclidean norm of the incremental displacement vector for the model computed during iteration k of the load step
$max[abs\Delta u_k^{(i)}]$	maximum (absolute) entry in the displacement vector for the model following solution for iteration k of the step

Using these quantities, the four convergence tests are defined as follows:

Test 1: $\| \Delta u_k \| \leq (user\ tol/100) \times \| \Delta u_1 \|$

Test 2: $\| R_k \| \leq (user\ tol/100) \times \| P \|$

Test 3: $max[abs\Delta u_k^{(i)}] \leq (user\ tol/100) \times \| \Delta u_1 \|$

Test 4: $max[absR_k^{(i)}] \leq (user\ tol/100) \times \| P \|$

where $\| \ \|$ denote the Euclidean norm. Multiple convergence tests may be defined; convergence requires satisfaction of all tests. Tolerance values are specified in (%); thus, a user tolerance of 0.01 (%) is reasonably strict and often used. Tolerance values of 0.001–0.05 have been used successfully in various models. The user specified tolerance exerts a dramatic impact on the required number of Newton iterations and the total CPU time. Excessively tight tolerances do not provide real improvements in solutions.

Commands to define the convergence tests are

convergence (tests) [< test type>]

where the test types parallel the four tests defined above. The command to define Tests 1 and 2 is:

norm { displacement / residual (load) } tolerance < tolerance: number >

Similarly, command to define Tests 3 and 4 is:

maximum { displacement / residual (load) } tolerance < tolerance: number >

An example of convergence test commands is:

```
nonlinear analysis parameters
  maximum iterations 10
  convergence test norm res tol 0.01 maximum displ tol 0.01
  nonconvergent solutions continue
```

2.9.4 Adaptive Step Size Control

In a nonlinear analysis (static or dynamic), it is often difficult to estimate *a priori* the appropriate load step sizes which provide rapid convergence of the Newton iterations. WARP

provides a simple facility to reduce automatically load step (and time step) sizes when the solution appears to be diverging or converging slowly. By default, the adaptive step size feature is not used.

The adaptive algorithm is very simple. When the user specified limit on Newton iterations is reached and the solution has not converged, the load step (and time step) is subdivided into four (4) equal increments and the solution for the load step restarted. Steps are not renumbered during this process so that output messages indicate four solutions of the load step. The output messages indicate which *fraction* of the user specified load step is being analyzed, e.g., 0.25 to 0.5.

Material models may also request an immediate load step reduction when the adaptive solution strategy is enabled. State variable updating may experience convergence difficulties requiring a reduction in load step size.

If the solution does not converge in any one of the 4 subincrements, that subincrement is further subdivided into four more increments and the solution restarted. Only two such levels of step reduction are permitted; nonconverged solutions at the second level cause program termination. In many cases, the first level of step reduction is sufficient. In other cases, one or more of the 0.25 fractions must be subdivided to obtain convergence. The adaptive algorithm performs level two reduction only for the level one fractions that do not converge.

The command to control adaptive load step sizes is

$$\underline{\text{adaptive}}\ (\underline{\text{solution}}) \left\{ \begin{array}{c} \underline{\text{on}} \\ \underline{\text{off}} \end{array} \right\}$$

When the adaptive procedure restarts the analysis for a load step or subincrement, it forces the first iteration to be resolved using the linear stiffness for the model. This is required since the current estimate for the solution at n+1 is not valid for use to recompute element matrices. The full Newton process resumes at the next iteration. WARP manager routines handle these processes automatically.

Adaptive load step control is strongly recommended for users attempting the nonlinear solution of new classes of problems until experience with the convergence characteristics are known. For parametric studies of problems with well known convergence characteristics, adaptive load step control should not be used as it often dramatically increases analysis run times (the code repeatedly learns what size steps converge!). Analyses run much faster when the user specifies load step sizes known to exhibit good convergence characteristics.

Non–Zero Constraints

When a load step is subdivided, the non–zero constraints (e.g., $\Delta u_{10} = 0.1$) imposed by the user are reduced by the same adaptive factors as the step load. The actual constraint values specified by the user and stored in program data structure are not modified. Rather, scaled values are imposed during the equation solving process.

2.9.5 Batch Status Messages

During solution of a large nonlinear problem in batch mode (e.g. on a Cray), it proves convenient to have occasional information about the progress of the solution (load step/iteration number, convergence rate, etc.) WARP provides an option to produce status messages independent of the normal (standard) output file for the job. A status file is written after each equilibrium if each step; files are named wm_####_@@ where #### denotes the load step number and @@ denotes the iteration number. The contents for an example file are:

```
newton convergence tests step: 1 iteration:   1 @ cpu:   43.6
-------------------------------------------------------------
    completed fraction over step: 1.00000
    maximum residual force:   0.179549E+00 @ node:   1356
    test 2: norm of residual load vector:          0.13631E+01
            norm of total load vector:             0.23237E+01
            ratio*100:        58.66192
```

If the message file exists from a previous analysis, the new information overwrites the old file. By default, no batch message files are written. The command to control batch messages is

batch (messages) { on / off }

2.9.6 CPU Time Limit

On some systems, batch jobs are executed with a user specified limit set on the CPU time for the job. If the WARP execution exceeds the CPU time limit, the program is aborted by the operating system and all results after the last written restart file are lost. Estimating the required CPU time for highly nonlinear problems may be very difficult, especially when similar problems have not been executed previously.

To help users with this problem, WARP provides its own cpu time limit feature. The user informs WARP of the allowable CPU time (in secs) for the job. At the beginning of the solution for load step n+1, WARP assumes that the solution time for the step is the same as the time required the solution of load step n. The total CPU time estimated to advance the solution through load step n+1 is computed using this procedure and compared to the user specified limit. If the estimated time exceeds 90% of the user limit, WARP writes a restart file named xxxxx_overtime_db for load step n and terminates the job (xxxxx denotes the structure name).

The command to control this option is

cpu (time) (limit) { on < limit: secs > / off }

By default the cpu time limit feature is *off*.

2.9.7 Displacement Extrapolation

In nonlinear analyses, the use of an extrapolated displacement vector frequently enhances the convergence rate of the Newton iterations— especially for "smooth" responses in plasticity. The incremental displacements computed from the solution over step n–1 to n are scaled and applied to the model to start the iterative (Newton) solution from n to n+1. The displacement scaling factor is computed from the specified step multipliers for steps n (say f_n) and n+1 (f_{n+1}) as f_{n+1}/f_n. Alternatively, users may specify directly the multiplier value. Only one loading pattern is permitted in the step definition when the extrapolate option is in effect.

The extrapolated displacement vector is employed at the beginning of load step n+1 to compute a set of incremental nodal forces for application to the model during iteration 1. The strains/stresses/internal forces are updated for the extrapolated displacement vector

but the material states are not retained for the next iteration. The direct solver is executed during iteration 1 with this incremental load vector. For the LPCG solver, the solution during iteration 1 is bypassed altogether with the extrapolated displacement vector passed directly to the strain/stress updating routines. Current experience indicates these are optimum solution procedures.

When the non–zero constraints are modified during a displacement control analysis, the loading step multipliers must be modified accordingly by the user; otherwise the extrapolation ratio (f_{n+1}/f_n) is computed incorrectly. To illustrate, consider the following example. Non–zero constraints are specified to load the model. The dummy loading pattern and nonlinear loading are defined as above with step multipliers of 1.0 for load steps 1–10. After step 10, the user modifies the constraints to reduced the imposed increment (uniformly) by one–half, possibly to reduce the number of Newton iterations for convergence in subsequent steps. Load steps 11, 12, 13, ... must have a multiplier of 0.5 for correct extrapolation. In step 11, the extrapolation multiplier is 0.5/1.0=0.5 while in steps 12, 13, ... the multiplier again becomes 1.0.

The command to control displacement extrapolation is

extrapolate { on ((multiply) (by) < scale factor: number >) / off }

When the *multiply by* option is given, the user specified scale factor supercedes the computed scale factor.

Numerical experiments reveal significant improvements in the Newton convergence rate, especially for displacement controlled loading. For this reason, *extrapolate on* is the system default. The *extrapolate* option is correctly processed when used with adaptive load step control.

2.9.8 Material Model Messages

The material models have built–in features to print status messages during stress update. An option is provided to suppress all such informative messages generated by material models. Messages about severe conditions in the material models are not suppressed with this option. For example, the material model may request an immediate load step reduction when adaptive load control is enabled. In such cases, the material model prints a message to this effect with the reason it requests a load step reduction. The command to control printing of informative material messages is

material (messages) { on / off }

Material messages are *on* by default.

2.9.9 Residual Loads Printing

Residual forces at nodes may be printed during Newton's iterations to facilitate debugging of problems which exhibit unusual convergence. To request printing of residual loads, use the command

Residual loads printing is *off* by default.

print residual (loads) (for) (iterations) < integer list >

2.10 Compute Requests

Solution For Load Steps

The nonlinear (and dynamic) solution for a series of one or more load steps is requested with the command

compute displacements (for) loading < nonlinear load id: label >
(for) steps < integer list >

A comma may be used anywhere in the line for continuation. WARP compares the last step number solved against the list of steps provided. A list of steps for computation is generated from this process and computations initiated. For example, let steps 1–10 be analyzed in the first *compute* command. The second *compute* command requests computation for steps 20–25. WARP automatically inserts steps 11–19 into the list of steps for computation.

WARP verifies the data provided in this command for correctness, e.g., the nonlinear load must exist and the steps requested must be defined in that load step. When errors are encountered, the command is ignored and a new input line read.

Once this command is accepted and computations begin, the user cannot intervene in the solution process until the analysis for all steps in the list is completed.

Examples of *compute* commands are:

```
compute displa load test steps 1-20
compute displacements for loading crush for steps 15-30
```

Domain Integral (J)

Once the solution for a load step is available, a domain integral evaluation to compute the J–integral may be requested. The domain(s) for computation must be defined immediately prior to the *compute* request. Chapter 4 describes commands to define domains for J computation. The *compute* command has the form

compute domain (integral)

2.11 Output Requests

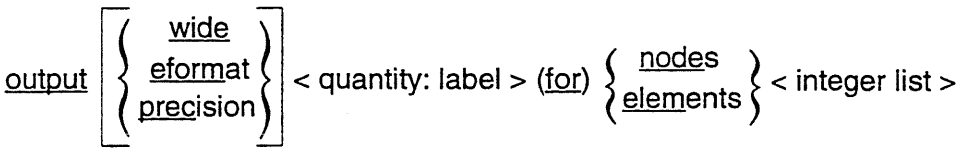
The output command provides computational results in two forms:

- printed output with page and column headers
- Patran nodal result files in either binary or ASCII formats.

Output commands must be given immediately after completion of the solution for a load step. Once the solution for load step n converges, WARP immediately updates all internal variables to prepare for solution of step n+1; only results for load step n are available for output.

2.11.1 Printed Output

The command to request printed output has the form



where

< quantity > is *one* of the following *displacements, velocities, accelerations, strains, stresses, internal_forces.*

The destination for printed output is the *current* output device specified by the user. The output device is either a disk file or the workstation display. The output file is declared using the standard output (i.e., the < file name) convention of Unix on the program invocation command or through the **output to <file>* command available in WARP (refer to Section 2.13 for a description of * commands).

By default, output routines which generate printed results format values to fit on an 8.5 in. x 11 in page oriented in portrait mode. The *wide* option permits extension of output up to 132 columns for eventual printing in the landscape orientation.

Numerical results are printed with an F12.6 format. An E12.5 format is requested with the *eformat* option. These *precision* option increases these fields to F26.16 and E26.16.

When a list of elements is specified for output of displacements, velocities, accelerations or internal forces, results are printed for the nodes of each element in the list (not the merged set of nodes for all elements in the list). Only lists of elements are permitted for output of strains and stresses. When the < integer list > of elements/nodes is omitted, the results are printed at *all* elements/nodes of the model.

The *internal forces* are reactions at constrained nodal dof; at unconstrained nodal dof, they are the remaining force imbalance due to nonlinear response and/or linear equation solving. Separate algebraic sums of the *X, Y, Z* components of these forces are printed following the nodal results to assist in the checking of reactions.

All strain/stress quantities refer to the global Cartesian coordinate system for the model. The number of strain/stress items printed for each element and the number/location of the points with printed results are specified with element properties. For example, the element logical property *long* requests an extended set of strain/stress results at the output points. The additional quantities include principal values, maximum shear values, state variables supplied from the material models, etc. The *short* output option is the default. The location/number of strain points is specified with the element logical properties: *gausspts, nodpts* or *center_output*. Node point values are extrapolated from the Gauss point values.

(S) $\varepsilon_x,\ \varepsilon_y,\ \varepsilon_z,\ \gamma_{xy},\ \gamma_{yz},\ \gamma_{xz}$

(S) $\varepsilon_{eff} = \frac{\sqrt{2}}{3}\sqrt{(\varepsilon_x - \varepsilon_y)^2 + (\varepsilon_y - \varepsilon_z)^2 + (\varepsilon_x - \varepsilon_z)^2 + 1.5(\gamma_{xy}^2 + \gamma_{yz}^2 + \gamma_{xz}^2)}$

$I_1 = \varepsilon_x + \varepsilon_y + \varepsilon_z$

$I_2 = \varepsilon_{xy}^2 + \varepsilon_{yz}^2 + \varepsilon_{xz}^2 - \varepsilon_x\varepsilon_y - \varepsilon_y\varepsilon_z - \varepsilon_x\varepsilon_z$

$I_3 = \varepsilon_x(\varepsilon_y\varepsilon_z - \varepsilon_{yz}^2) - \varepsilon_{xy}(\varepsilon_{xy}\varepsilon_z - \varepsilon_{yz}\varepsilon_{xz}) + \varepsilon_{xz}(\varepsilon_{xy}\varepsilon_{yz} - \varepsilon_y\varepsilon_{xz})$

$\varepsilon_1 \le \varepsilon_2 \le \varepsilon_3$ (Principal strain values)

l_1, m_1, n_1 (Cosines for direction 1)

l_2, m_2, n_2 (Cosines for direction 2)

l_3, m_3, n_3 (Cosines for direction 3)

(S)– value included with *short* output option. All values included with *long* option.

FIG. 2.3—*Strain values for output*

The center point values are numerical averages of Gauss point values. The default output location is *gausspts*. Figure 2.3 summarizes the element strain output quantities; Fig. 2.4 summarizes the element stress output quantities.

Several examples of output commands are

```
output wide eformat precision displacements for nodes 1-300 by 2

output stresses elements 900-1500 by 2 300-500

output accelerations for elements 20-40 100-300 by 3
```

2.11.2 Patran Compatible Result Files

The command to request Patran output files has the form

$$\underline{\text{output}}\quad \underline{\text{patran}} \left\{ \begin{matrix} \underline{\text{binary}} \\ \underline{\text{formatted}} \end{matrix} \right\} \text{< quantity: label >}$$

where

< quantity > is *one* of the following *displacements, velocities, accelerations, strains, stresses, internal forces.*

Both *binary* and *formatted* nodal results are sequential files created with the Fortran *open* statement and written with ordinary Fortran *write* statements.

Binary files have names with the prefixes *pbd, pbv, pba, pbi, pbe* and *pbs* followed by the 5 digit load step number. The '*b*' indicates a binary file format. Similarly, '*d*' – displacements, '*v*' – velocities, '*a*' – accelerations, '*i*' – internal forces, '*e*' – strains and '*s*' – stresses.

(*S*) $\sigma_x,\ \sigma_y,\ \sigma_z,\ \sigma_{xy},\ \sigma_{yz},\ \sigma_{xz}$

(*S*) $U_o = \int_0^{\varepsilon} \sigma d\varepsilon$ (Work density)

(*S*) $\sigma_{vm} = \frac{1}{\sqrt{2}}\sqrt{(\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_x - \sigma_z)^2 + 6(\sigma_{xy}^2 + \sigma_{yz}^2 + \sigma_{xz}^2)}$

c_1, c_2, c_3 (State variables from material model)

$I_1 = \sigma_x + \sigma_y + \sigma_z$

$I_2 = \sigma_{xy}^2 + \sigma_{yz}^2 + \sigma_{xz}^2 - \sigma_x\sigma_y - \sigma_y\sigma_z - \sigma_x\sigma_z$

$I_3 = \sigma_x(\sigma_y\sigma_z - \sigma_{yz}^2) - \sigma_{xy}(\sigma_{xy}\sigma_z - \sigma_{yz}\sigma_{xz}) + \sigma_{xz}(\sigma_{xy}\sigma_{yz} - \sigma_y\sigma_{xz})$

$\sigma_1 \le \sigma_2 \le \sigma_3$ (Principal stress values)

l_1, m_1, n_1 (Cosines for direction 1)

l_2, m_2, n_2 (Cosines for direction 2)

l_3, m_3, n_3 (Cosines for direction 3)

(*S*)– value included with *short* output option. All values included with *long* option.

FIG. 2.4—*Stress values for output*

Formatted files have with the prefixes *pfd, pfv, pfa, pfi, pfe* and *pfs* followed by the 5 digit load step number. The '*f*' indicates a formatted file format. Similarly, '*d*' – displacements, '*v*' – velocities, '*a*' – accelerations, '*i*' – internal forces, '*e*' – strains and '*s*' – stresses.

Figure 2.5 summarizes the data column assignments for Patran strain/stress results files. The first six strain/stress values that appear at each model node in the Patran files are the numerical average for the contribution of each element at the node. Effective strain, Mises value, strain invariants, principal values and directions are computed from the averaged nodal values. The energy density and the three material model state variables are the averaged nodal values. The stress invariants, principal values and directions are computed from the averaged nodal values.

PDA Engineering, Inc. (the developers of Patran) publishes specifications for the formatted and binary structures of these results files. Appendix A provides skeleton Fortran programs to read the binary and formatted forms of the results files.

Please note the following:

- these files contain result values *only* at model nodes. Strains and stresses are nodes are obtained using a two step process: (1) extrapolation of Gauss point values to element nodes and then (2) numerical averaging of all nodal values.
- invariants, principal values and direction cosines are computed using the averaged nodal results for the strain and stress components
- the effective strain (e_{eff}), *mises* effective stress (σ_{vm}) are computed using the averaged nodal results for the strain and stress components
- the work density (U_0) and material model state variables (c_1, c_2, c_3) are the extrapolated and then averaged nodal values

- it is not possible, at present, to specify a list of nodes to appear in the Patran results files. Results are written for all nodes in the model.

Data Column	Strain Value	Data Column	Stress Value
1	ε_x	1	σ_x
2	ε_y	2	σ_y
3	ε_z	3	σ_z
4	γ_{xy}	4	σ_{xy}
5	γ_{yz}	5	σ_{yz},
6	γ_{xz}	6	σ_{xz}
7	ε_{eff}	7	U_o
8	I_1	8	σ_{vm}
9	I_2	9	c_1
10	I_3	10	c_2
11	ε_1	11	c_3
12	ε_2	12	I_1
13	ε_3	13	I_2
14	l_1	14	I_3
15	m_1	15	σ_1
16	n_1	16	σ_2
17	l_2	17	σ_3
18	m_2	18	l_1
19	n_2	19	m_1
20	l_3	20	n_1
21	m_3	21	l_2
22	n_3	22	m_2
		23	n_2
		24	l_3
		25	m_3
		26	n_3

FIG. 2.5—*Column numbers for strain–stress results in Patran data files*

2.12 Analysis Restart

Create A Restart File

To maintain the highest possible performance, WARP allocates all data structures in memory during execution and does not use databases on magnetic disk to temporarily hold (*swap*) data arrays. At completion of load step *n*, the user may request creation of a binary (sequential) file of data arrays required to restart execution at that point in the solution. The default form of the *save* command is

save (structure) (< structure id: label >)

where the structure id is optional. If omitted, the last specified structure name is used. The data file created with this command has the name @_db where @ denotes the first 8 characters of the structure id.

An explicit name for the restart file may be specified with the command

save (to) file < file name: label *or* string >

where a <string> must be used if the file name starts with/or contains special characters. The optional phrase *structure* may also be included in this command to maintain readability.

Examples of the *save* command are

```
save
save structure cylindrical_bar
save to file bar_step_450
save to file '452_model_bar'
save structure bar to file '325_bar'
```

Restart file sizes increase with the model size and solution characteristics. For example, a 7200 node, 5700 element model using a large displacement formulation and the rate–dependent Mises model requires 52 MB of space for each restart file on a Cray.

In a typical analysis, the solution is advanced 10 to 50 load steps then a new restart file is requested. WARP can be executed later to output results for the load step in a restart file. The explicit naming feature enables creation of a series of unique restart files at various points in the analysis.

Access A Restart File

To restart execution of WARP, the first (non–comment) command must be

retrieve (structure) (< structure id: label >)

or using an explicit name for the restart file

retrieve (from) file < file name: label *or* string >

where a <string> must be used if the file name starts with/or contains special characters. The optional phrase *structure* may also be included in this command to maintain readability.

Once the restart file is opened and read into memory, WARP displays a message indicating the load step number *n* for the restart file and the time completed in the analysis (useful for dynamic analyses). Commands to request output, to analyze additional load steps, etc. may then be given as usual.

2.13 Utility (*) Commands

Several utility commands are provided to manipulate input–output files, to control command echo, etc. Each of these commands begins with an * and these commands may be given at any time during input.

* *Echo Command*

The * *echo* command controls the "echoing" of input commands to the current output device. By default, all commands are echoed. The * *echo* command has the form

```
* echo { on  }
       { off }
```

* *Input Command*

The * *input* command controls the location from which input commands are read for processing. By default, the input stream is the user's interactive display or the Unix *stdin* device. The input stream can be switched to a disk file or switched back to the interactive display

```
* input (from) { display                                  }
               { (file) < file name: label or string >    }
```

where the <string> form is required with file names not meeting the definition of a <label>. * *input from file ...* commands may be contained within referenced input files to create an input "stack" up to 10 levels deep. When an end–of–file condition is reached on the current file, the stack is popped to again read from the previous file. When reading of the last file completes, the input stream returns to the user's display. In a batch job, the program is terminated by the WARP command processor if an end–of–file condition occurs at the highest level.

* *Output Command*

The * *output* command controls the location (stream) to which usual WARP output is directed. By default, the output stream is the user's interactive display or the Unix *stdout* device. The output stream can be switched to a disk file or switched back to the interactive display

```
* output (to) { display                                  }
              { (file) < file name: label or string >    }
```

where the <string> form is required with file names not meeting the definition of a <label>.

* *Time Command*

The * *time* command outputs the elapsed CPU time ins seconds for the current job.

```
* time
```

* *Reset Command*

When the WARP command processor interprets the command stream, errors of various types may be detected. When errors are encountered, the command processors set an inter-

nal flag .true. to prevent a *compute* command from attempting a solution. This internal flag can be set to the "no error" condition with the * *reset* command, which has the form

* reset

Chapter 3

Elements and Material Models

This chapter describes the elements and material models currently available. The formulations and computational procedures unique to the elements/models are outlined in detail sufficient for their proper use.

3.1 Element Type: *l3disop*

This eight–node, isoparametric element provides the fundamental modeling capability in WARP. The element formulation employs a conventional tri–linear displacement field. With the $\overline{B}$ modifications of Hughes [38], the element exhibits minimal locking under fully incompressible material response and exhibits slightly improved bending response. The element performs well under finite deformations encountered, for example, near severe discontinuities and near crack fronts.

The element formulation supports geometrically nonlinear analysis (large displacements, rotations, finite strains), materially nonlinear analysis and combined geometric/material nonlinear analysis.

For dynamic analyses, the diagonal (lumped) mass matrix derives from the scaled terms of the consistent mass matrix.

All element computations take place in the global coordinate system for the model. Strains and stresses output by the model reference the global coordinate axes.

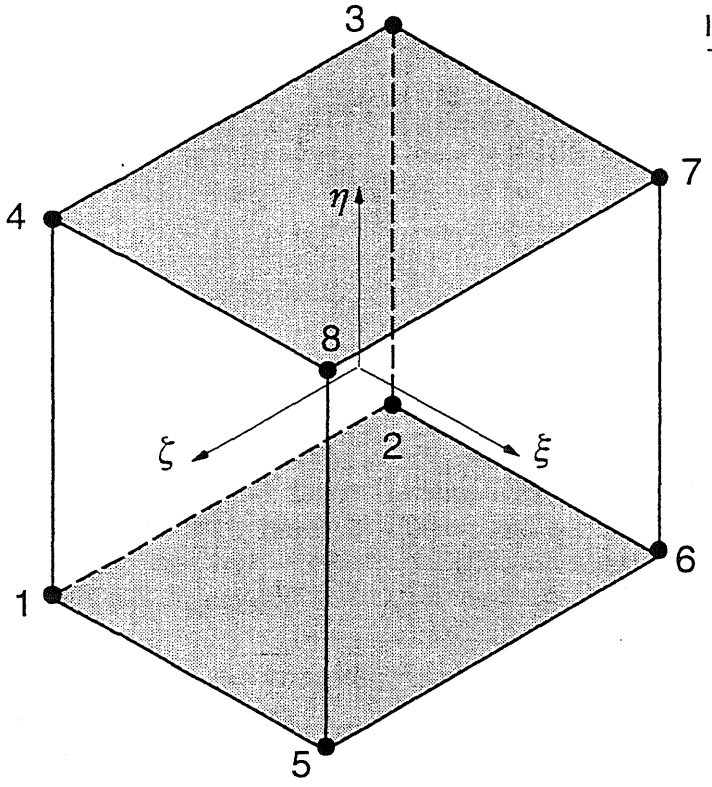
For modeling initially sharp crack fronts, this element is frequently degenerated or collapsed into a wedge shape. While this modeling technique causes no problems for a small–strain analysis, difficulties in Newton convergence of the global solution can be experienced when the collapsed elements have the geometric nonlinear formulation. The remedy is to model the crack front as a very small tube (i.e. a *keyhole*) or to model the crack tip as an initially blunt notch with a root radius very small compared to the crack length or remaining ligament length.

3.1.1 Node and Gauss Point Ordering

Figure 3.1 shows the ordering of nodes for the element, the orientation of parametric axes (ξ, η, ζ), and the location of Gauss points for the standard $2 \times 2 \times 2$ order.

Element results are frequently output at the "center point" which corresponds to parametric location (0,0,0).

Isoparametric elements provide a powerful capability to model the geometry of irregularly shaped bodies. The parent element in parametric coordinates is mapped into the global Cartesian space using (current) coordinates of the nodes and the linear interpolation functions. The element behavior remains adequate unless the mapped shape becomes unreasonable (either the initial, undeformed shape or the current shape if geometric nonlinear analysis). Corner angles on each face must be $>0°$ and $<180°$. The best element response is obtained for angles within the range $90° \pm 30°$. Large aspect ratios should be avoided if possible. The best element behavior derives from a cubical shape; however, rect-



Isoparametric Coordinates of Nodes

Node	ξ	η	ζ
1	–1	–1	1
2	–1	–1	–1
3	–1	1	–1
4	–1	1	1
5	1	–1	1
6	1	–1	–1
7	1	1	–1
8	1	1	1

Coordinates of Gauss Pts.

Point	ξ	η	ζ
1	–a	–a	–a
2	–a	a	–a
3	a	–a	–a
4	a	a	–a
5	–a	–a	a
6	–a	a	a
7	a	–a	a
8	a	a	a

a=0.57735

FIG. 3.1—*Local node ordering for the 8–node isoparametric element "l3disop." Isoparametric coordinates for the element nodes and Gauss points are listed.*

angular prism shapes with aspect ratios of 10–20 are commonly used without undue loss of accuracy, especially if the strain field varies gently in the "long" direction.

Element routines check for badly distorted elements by examining the determinant of the coordinate jacobian at the integration points (using the current nodal coordinates for geometric nonlinear analysis). Zero or negative values indicate a severely distorted element. Messages identifying these problems are printed with information about the element.

3.1.2 Element Properties

Table 3.1 summarizes the user–assignable values that control element behavior. Element properties are defined by the name of the property, a < label >, followed by a value. Logical properties are set .true. simply by the appearance of the property name. The default behavior for the *l3disop* element is this: small–strain formulation, 2x2x2 Gauss integration, $\overline{B}$ formulation, and output of a short list of strains–stresses at the Gauss points.

Element Property	Keyword	Mode	Default Value
Geometrically *linear* formulation	*linear*	Logical	True
Geometrically *nonlinear* formulation	*nonlinear*	Logical	False
Material associated with element	*material*	Label	none
Order of Gauss integration	*order*	String	2x2x2
Use $\overline{B}$ formulation	*bbar*	Logical	True
Do not use $\overline{B}$ formulation	*no_bbar*	Logical	False
Output strains–stresses at Gauss points	*gausspts*	Logical	True
Output strains–stresses at element nodes	*nodpts*	Logical	False
Output strains–stresses at (0,0,0) in element	*center_output*	Logical	False
Output minimal set of strain–stress values	*short*	Logical	True
Output full set of strain–stress values	*long*	Logical	False

Table 3.1 Properties for *l3disop* element

3.1.3 Output Options

Printed strain–stress results may be obtained at the Gauss points (default), the element nodes or at the parametric centerpoint of the element (0,0,0). Figures 2.3 and 2.4 define each of the strain–stress values output by the element.

Nodal values of σ_{ij}, ε_{ij} are computed by extrapolation of Gauss point values using linear, Lagrangian polynomials. Values of effective strain, Mises stress, invariants, principal values, etc. are computed from these extrapolated values. The state variables provided by the material model in the stress data are simply extrapolated to element nodes (the element output routine is unaware of the contents of these variables).

The centerpoint values of σ_{ij}, ε_{ij} are the simple numerical average of Gauss point values. Values of effective strain, Mises stress, invariants, principal values, etc. are computed from these averaged values. The state variables provided by the material model in the stress data are simply averaged (the element output routine is unaware of the contents of these variables). The material model state variables contained in the stress data are simply averaged (the element output routine is unaware of the contents of these variables).

The *short* option requests printing of a reduced set of output values. The invariants, principal values and direction cosines are omitted. This is the default output option.

3.1.4 Mass Formulation

The element (diagonal) mass matrix is evaluated once at the start of computations for the first load step. Entries of the lumped mass are proportional to the diagonal entries of the element consistent mass. The proportionality factor is defined to preserve the total mass of the element, e.g., the sum of the diagonal terms for the $\ddot{v}_i$ accelerations equals the element mass. This procedure always generates positive values for the lumped mass and leads to optimal conver gence rates with mesh refinement.

The element mass matrix for analysis is thus given by

$$m^e_{pq} = \begin{cases} a\delta_{ij}\displaystyle\int_{V_e} \varrho N_a N_b \, dV_e & a = b \\ 0 & a \neq b \end{cases} \tag{3.1}$$

where ϱ denotes the mass density of the undeformed material. N_a denotes the usual linear interpolating functions for the element node a. The scaling factor a is given by

$$a = \int_{V_e} \varrho \, dV_e \Bigg/ \left[\sum_{a=1}^{8} \int_{V_e} \varrho N_a^2 \, dV_e\right] \tag{3.2}$$

total element mass — *sum of diagonal entries of consistent mass*

3.1.5 Element Loads (not yet implemented)

Loads available for the *l3disop* element include body forces and face tractions. A sequence of element load definitions has the form

element (loads)
< elements: list > < type of element loading >
< elements: list > < type of element loading >
•
•

where the <type of element loading> is either a body force or a face traction.

Body Forces

Body forces are specified by the intensity (units of F/L^3) and the direction along one of the coordinate axes. The body force intensity is constant over the element. The body force loads are defined by the command

body (forces) [{ bx / by / bz } (=) < force intensity: number > (,)]

Face Tractions

Tractions applied to the faces of elements may have a direction along one of the global coordinate axes or a direction normal to the specified face. Figure 3.2 defines the face num-

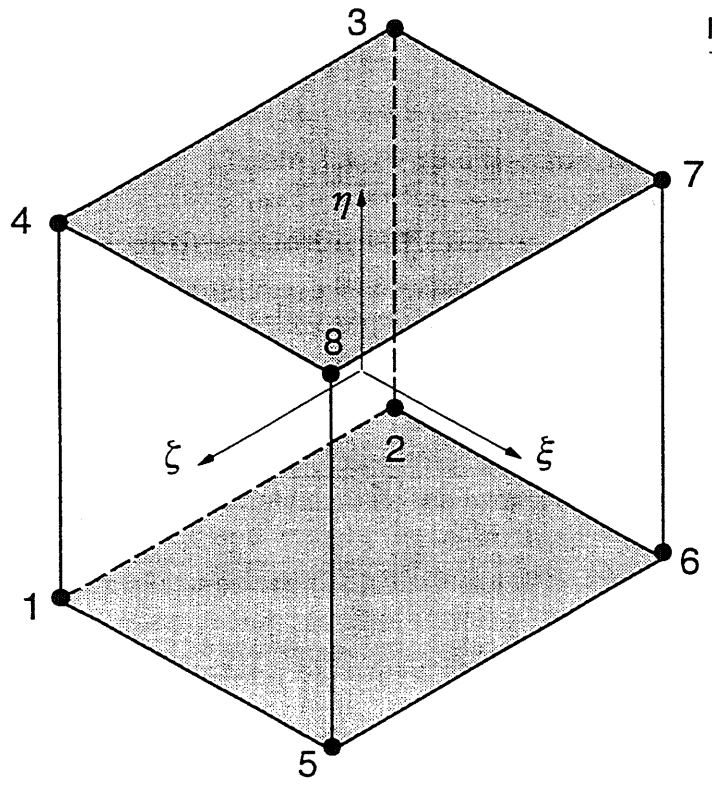
bers. The commands define the loaded face of the element, the loading intensity (units of F/L^2), and the loading direction. When the traction is aligned with one of the coordinate axes, the command has the form

face < face number: integer> (tractions) [{ tx / ty / tz } (=) < intensity: number > (,)]

For a normal (pressure) loading, use a command of the form

face < face number: integer> pressure (=) < intensity: number >

where a positive value for the intensity denotes a load directed into the face, i.e., a positive pressure loads the face in compression.



Isoparametric Coordinates of Nodes

Node	ξ	η	ζ
1	–1	–1	1
2	–1	–1	–1
3	–1	1	–1
4	–1	1	1
5	1	–1	1
6	1	–1	–1
7	1	1	–1
8	1	1	1

Face No. Nodes

1	1–2–3–4
2	5–8–7–6
3	1–5–6–2
4	4–8–7–3
5	2–6–7–3
6	1–5–8–4

FIG. 3.2—*Face numbers for applying tractions to the 8–node isoparametric element "l3disop."*

Element loads are additive; if the same element and direction appear in two different loading commands the sum of two loads is applied to the model. An example sequence to define a loading condition and a set of element loads is

```
loading one
  element loads
     1-40 620-800 by 2 face 6 pressure -2.1
     140 face 3 tractions tx -0.5 ty 14.34 tz 42.6
     3256-4000 body forces bz 12.3 bx -32.4
     20 body force bx -3
```

In the above example, element 20 has both a normal face pressure on face 6 and body forces in the x and y directions.

Large Displacement Effects on Loads

When the analysis includes geometric nonlinear effects (large displacements), equivalent loads for the incrementally applied surface tractions are re–computed at the beginning of each load step using the current (deformed) geometry of the elements.

3.1.6 Strains–Stresses for Geometric Nonlinear Formulation

The *nonlinear* property requests a geometrically nonlinear element formulation. Stress values output by the element are then the *Cauchy* (true) stresses. The Cauchy stress defines tractions over internal surfaces in the *deformed* configuration. The Cauchy stress components, σ_{ij}, are aligned with the global coordinate axes for the model. The symmetric Cauchy stress satisfies the equilibrium conditions

$$\int_S \boldsymbol{n} \cdot \boldsymbol{\sigma}\, dS = \int_V \left(\frac{\partial}{\partial \boldsymbol{x}}\right) \cdot \boldsymbol{\sigma}\, dV \tag{3.3}$$

where $\boldsymbol{n}$ defines a unit outward normal to the deformed surface S, and V denotes the deformed volume of the body.

The increment of strain that advances the solution from load step n to n+1 is given by

$$\Delta\varepsilon = \overline{\boldsymbol{B}}(\boldsymbol{x}_{n+1/2})\Delta\boldsymbol{u} \tag{3.4}$$

where the $\overline{\boldsymbol{B}}$ form the linear–strain displacement matrix is evaluated at the mid–step deformed configuration. This corresponds to a finite increment of the rate of deformation tensor, $\boldsymbol{D}_{n+1/2} \cdot \Delta t$, over the step. Converged increments of $\Delta\varepsilon$ are summed over k load steps to define a measure of strain for output as

$$\varepsilon = \sum_{n=1}^{n=k} \Delta\varepsilon \tag{3.5}$$

where the shear strains follow the usual engineering definition, i.e., $\Delta\gamma_{xy} = 2 \times \Delta\varepsilon_{xy}$.

The increment of strain $\Delta\varepsilon$ is identified as the rate of logarithmic strain with respect to the current (deformed) configuration.

3.1.7 The $\overline{\boldsymbol{B}}$ Formulation

Many methods of alleviating the locking which occurs in fully integrated elements have been proposed in the literature. The so–called $\overline{\boldsymbol{B}}$ method (Hughes [38]) implemented for the *l3disop* element in WARP3D is outlined below.

The strains are divided into deviatoric and dilatational parts in the following manner.

$$\varepsilon_{ij} = \varepsilon_{ij}^{dev} + \varepsilon_{ij}^{dil} \qquad \varepsilon_{ij}^{dil} = \frac{1}{3}\delta_{ij}\sum_{k=1}^{3}\varepsilon_{kk} \qquad \varepsilon_{ij}^{dev} = \varepsilon_{ij} - \varepsilon_{ij}^{dil} \tag{3.6}$$

The strain–displacement matrix, $\boldsymbol{B}$, is divided divided into a dilatational and deviatoric parts in the same manner as

$$\boldsymbol{B} = \left[\boldsymbol{B}_1\ \boldsymbol{B}_2\ \ldots\ \boldsymbol{B}_n\right] \tag{3.7}$$

where

$$\boldsymbol{B}_i = \begin{bmatrix} B_1 & 0 & 0 \\ 0 & B_2 & 0 \\ 0 & 0 & B_3 \\ B_2 & B_1 & 0 \\ 0 & B_3 & B_2 \\ B_3 & 0 & B_1 \end{bmatrix} \qquad \boldsymbol{B}_i^{dil} = \frac{1}{3}\begin{bmatrix} B_1 & B_2 & B_3 \\ B_1 & B_2 & B_3 \\ B_1 & B_2 & B_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{B}_i^{dev} = \boldsymbol{B}_i - \boldsymbol{B}_i^{dil} \tag{3.8}$$

with the subscript i is omitted for clarity on each term inside the [] and

$$B_{i1} = \frac{\partial N_i}{\partial x} \qquad B_{i2} = \frac{\partial N_i}{\partial y} \qquad B_{i3} = \frac{\partial N_i}{\partial z} \tag{3.9}$$

The dilatational contribution to the stiffness causes locking for near incompressible conditions and is replaced the dilatational part of the strain–displacement matrix with a modified dilatational part, $\overline{\boldsymbol{B}}^{dil}$. The strain–displacement matrix is replaced by $\overline{\boldsymbol{B}}$ defined as:

$$\overline{\boldsymbol{B}}_i = \boldsymbol{B}_i^{dev} + \overline{\boldsymbol{B}}_i^{dil} \qquad \text{where } \overline{\boldsymbol{B}}_i^{dil} = \frac{1}{3}\begin{bmatrix} \overline{B}_1 & \overline{B}_2 & \overline{B}_3 \\ \overline{B}_1 & \overline{B}_2 & \overline{B}_3 \\ \overline{B}_1 & \overline{B}_2 & \overline{B}_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.10}$$

where again the subscript i on each term in the [] has been omitted. The $\overline{\boldsymbol{B}}$ matrix can then be written out explicitly in the following form (with subscript i omitted inside [])

$$\overline{\boldsymbol{B}}_i = \frac{1}{3}\begin{bmatrix} 2B_1 + \overline{B}_1 & \overline{B}_2 - B_2 & \overline{B}_3 - B_3 \\ \overline{B}_1 - B_1 & 2B_2 + \overline{B}_2 & \overline{B}_3 - B_3 \\ \overline{B}_1 - B_1 & \overline{B}_2 - B_2 & 2B_3 + \overline{B}_3 \\ 3B_2 & 3B_1 & 0 \\ 0 & 3B_3 & 3B_2 \\ 3B_3 & 0 & 3B_1 \end{bmatrix} \tag{3.11}$$

Several options for defining $\overline{\boldsymbol{B}}_i^{dil}$ have been proposed in the literature. Here we use the "mean dilatation" approach suggested by Nagtegaal, et al. [59]. A volume averaged (mean) $\tilde{\boldsymbol{B}}_i$ matrix is computed over the element as

$$\tilde{\boldsymbol{B}}_i = \frac{1}{V_e}\int_{V_e} \boldsymbol{B}_i dV_e \tag{3.12}$$

with $\overline{\boldsymbol{B}}_i^{dil}$ at each Gauss point taken as the dilatational component of $\tilde{\boldsymbol{B}}_i$ as in Eq. (3.10). To save computations, only the three terms needed from $\tilde{\boldsymbol{B}}_i$ to compute $\overline{\boldsymbol{B}}_i^{dil}$ are actually evaluated

$$\overline{B}_{i1} = \frac{\partial \overline{N}_i}{\partial x} = \frac{1}{V_e}\int_{V_e} \frac{\partial N_i}{\partial x}\, dV_e \tag{3.13}$$

$$\overline{B}_{i2} = \frac{\partial \overline{N}_i}{\partial y} = \frac{1}{V_e}\int_{V_e} \frac{\partial N_i}{\partial y}\, dV_e \tag{3.14}$$

$$\overline{B}_{i3} = \frac{\partial \overline{N}_i}{\partial z} = \frac{1}{V_e}\int_{V_e} \frac{\partial N_i}{\partial z}\, dV_e \tag{3.15}$$

using the standard $2\times2\times2$ Gauss quadrature.

This formulation provides an element with the same dilatational strain and mean stress at each of the $2\times2\times2$ Gauss points. When plane strain constraints are imposed on the $\overline{\boldsymbol{B}}$ element, the ε_z is not restricted to 0 at each Gauss point, but is only restricted to 0 over the element as a whole, i.e., for *center_output* the ε_z value is zero.

When large displacement effects are present, the current coordinates of the element nodes are adopted to form $\overline{\boldsymbol{B}}$ to compute virtual strains for internal force computation as in

$$\boldsymbol{IF}_e = \int_{V_e} \overline{\boldsymbol{B}}^T(\boldsymbol{x}_{n+1})\sigma_{n+1}\, dV_e \tag{3.16}$$

where σ denotes the Cauchy stresses and V_e the current (deformed) element volume at n+1.

3.1.8 Example

The following example illustrates the specification of *l3disop* elements in a model.

```
structure cct
c
material a533b
  properties ....
c
number of nodes 25642 22092
c
elements
  14000-22092 type l3disop  nonlinear material a533b order 2x2x2,
                      long bbar center_output
c
```

3.2 Material Model Type: *deformation*

The flow or incremental theory of plasticity with a Mises yield surface has been extensively employed in elastic–plastic analyses. Alternatively, plasticity can be described by a deformation theory which assumes that the *strain path* at each material point remains linear (proportional) over the full range of loading. Deformation plasticity is essentially a nonlinear elasticity theory. For a proportional strain path, deformation and incremental plasticity theories provide identical results. Deformation plasticity *does not* correctly model the path–dependent behavior of materials for radical departures from proportional loading.

Deformation plasticity offers significant savings of computational effort compared to flow theory plasticity. Much larger load steps may be imposed on the model and only a few Newton iterations are needed for convergence at each load step. The number of computations performed in the material model is greatly reduced compared to a general incremental theory model since there is no explicit yield surface to complicate matters. Solutions tend to be very stable compared to those for flow theory especially when a region of the model contains large strain gradients, e.g., at a crack.

This model employs a representation of the uniaxial (tensile) stress–strain curve consisting of three parts: an initial, linear response followed by a small circular transition to a pure, power–law model.

The model supports only a small–strain formulation and rate–independent reponse. The assumptions of purely proportional loading in the model are questionable at best when finite strains and large rotations of material elements occur.

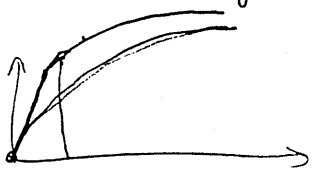
3.2.1 Formulation and Computational Procedures

The uniaxial stress–strain curve for the material is represented by the following relations (refer to Fig. 3.3):

$$\frac{\varepsilon}{\varepsilon_0} = \frac{\sigma}{\sigma_0} \qquad \text{for} \quad \frac{\sigma}{\sigma_0} \le K_1 \tag{3.17}$$

$$\frac{\varepsilon}{\varepsilon_0} = \varepsilon_{Nc} - \sqrt{r_{Nc}^2 - \left(\frac{\sigma}{\sigma_0} - \sigma_{Nc}\right)^2} \qquad \text{for} \quad K_1 \le \frac{\sigma}{\sigma_0} \le K_2 \tag{3.18}$$

$$\frac{\varepsilon}{\varepsilon_0} = \left(\frac{\sigma}{\sigma_0}\right)^n \qquad \text{for} \quad \frac{\sigma}{\sigma_0} > K_2 \tag{3.19}$$



where,

ε_0	reference stress (yield stress)
σ_0	reference stress (yield stress)
n	hardening exponent for power–law region
K_1, K_2	lower, upper stress limits for transition
$\varepsilon_{Nc}, \sigma_{Nc}$	center of circular transition arc
r_{Nc}	radius of transition arc

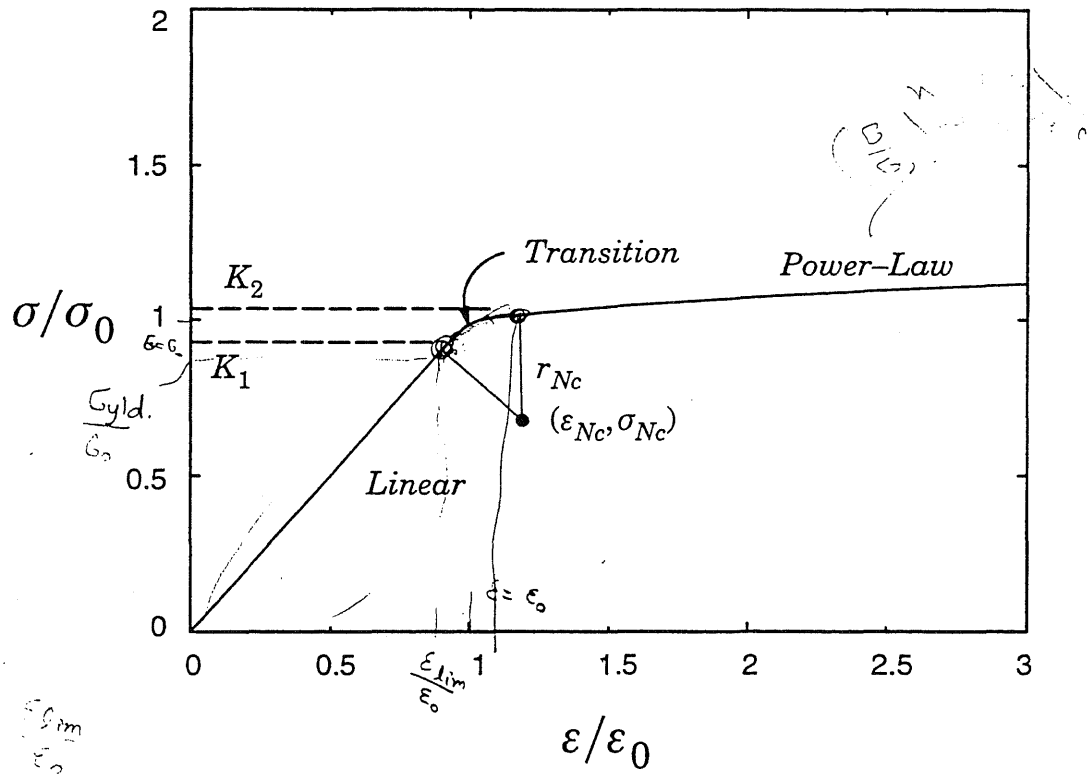


FIG. 3.3—*Uniaxial (tensile) stress–strain curve for the "deformation" plasticity material model.*

Given the linear limit, K_1, the model is able to compute the upper limit for the transition, K_2, based on the hardening exponent as well as the center of the small transitional arc and the corresponding radius. K_1 has the value 0.95.

Using an effective stress defined from the von Mises yield function and an effective strain defined from the Prandlt–Reuss relations, the total stress components in terms of the total strain components are given by:

$$\frac{\sigma_{ij}}{\sigma_0} = \frac{1}{3(1-2\nu)}\frac{\varepsilon_{kk}}{\varepsilon_0}\delta_{ij} + \frac{2}{3}\frac{\sigma_e/\sigma_0}{e_e/\varepsilon_0}\frac{e_{ij}}{\varepsilon_0} \qquad (3.20)$$

where the effective stress and strain are defined by:

$$\sigma_e^2 = \tfrac{1}{2}\left[(\sigma_{11}-\sigma_{22})^2 + (\sigma_{22}-\sigma_{33})^2 + (\sigma_{33}-\sigma_{11})^2 + 6(\sigma_{12}^2+\sigma_{23}^2+\sigma_{13}^2)\right] \qquad (3.21)$$

$$e_e^2 = \tfrac{2}{9}\left[(\varepsilon_{11}-\varepsilon_{22})^2 + (\varepsilon_{22}-\varepsilon_{33})^2 + (\varepsilon_{33}-\varepsilon_{11})^2 + \tfrac{3}{2}(\gamma_{12}^2+\gamma_{23}^2+\gamma_{31}^2)\right] \qquad (3.22)$$

Full details of the formulation may be found in the Appendix of the thesis by Wang [82].

3.2.2 Model Properties

The properties defined for material model *deformation* are listed in Table 3.2.

Model Property	Keyword	Mode	Default Value
Young's modulus	*e*	Number	0.0
Poisson's ratio	*nu*	Real	0.0
Mass density	*rho*	Real	0.0
Reference yield stress (σ_0)	*yld_pt*	Number	0.0
Power law exponent (n)	*n_power*	Number	0.0

Table 3.2 Properties for *deformation* Material Model

$$\text{resid.} = \left(\frac{\sigma}{\sigma_0}\right)^n + \frac{2}{3}\left(\nu - \frac{1}{2}\right)\frac{\sigma}{\sigma_0} - \frac{\varepsilon}{\varepsilon_0} = 0$$

3.2.3 Model Output

By default, the material model prints no messages during computations unless the numerical algorithms fail to converge. If requested, the material model prints the element number and strain point number whenever the effective stress *first* exceeds the specified yield stress. This option is requested with the nonlinear solution parameter *material messages on* (refer to Section 2.9.8)

The model makes available the exactly integrated strain energy density, U_0, to the element routines for subsequent output.

3.2.4 Computational Efficiency

The computational routines for this model process elements in blocks of a size matched to the vector length of the computer (i.e., Crays) or to the cache size of the workstation. All model computations are written in vectorized code except for the local Newton loop to update the scalar stress σ_e using the uniaxial stress–strain curve in Fig. 3.3. Compared to the general rate–dependent Mises model discussed later, this model is much faster. It is, however, slower than the fully vectorized Mises model with constant hardening described in the following section.

3.2.5 Example

The following example defines the properties for an A533B material frequently used in fracture models and assigns the material to some elements.

```
structure cct
c
material a533b
  properties deformation e 30000 nu 0.3 n_power 10 yld_pt 60.0,
      rho  7.3e-07
c
number of nodes 25642 22092
c
elements
  14000-22092 type l3disop  linear material a533b order 2x2x2,
```

```
                                long bbar
      c
```

3.3 Material Model Type: *bilinear* (mises)

This material model extends the small–strain Mises plasticity theory to include the effects of finite strains and rotations. Rate–independent, incremental theory of plasticity with (constant) isotropic and kinematic hardening is employed with the von Mises yield surface expressed in terms of the Cauchy (true) stress. This model is formulated for the analysis of ductile metals which undergo large plastic strains but small elastic strains. By this we mean that the unloaded configuration obtained after significant plastic deformation is negligibly different from the deformed configuration. This assumption simplifies considerably the treatment of material elasticity and permits additive decomposition into elastic and plastic components of strain increments defined with respect to the deformed configuration.

The constitutive framework for WARP3D outlined in Chapter 1 neutralizes finite rotation effects during stress–update and computation of the consistent tangent moduli. The small–strain, stress–updating procedures follow a single–step, elastic–predictor radial–return algorithm. The algorithm is unconditionally stable for large strain increments and provides superb accuracy in the updated stresses (for a single step method). Inelastic unloading–reloading events are processed without difficulty. The last section provides an overview of the radial–return procedures implemented for this model.

This model employs a representation of the uniaxial (tensile) stress–strain curve consisting of an initial, linear response followed by linear hardening. Purely isotropic, purely kinematic and mixed isotropic–kinematic hardening are offered as options.

The *bilinear* model provides a very computationally efficient alternative to the general Mises model described in the next section when the rate–independent, constant hardening assumptions apply in the analysis. All computational steps of stress–updating and consistent tangent generation are vectorized. This model has the best computational performance.

The following sections describe needed parameters to utilize this material model. Full details of the numerical implementation are provided in the final section.

3.3.1 Stress–Strain Curve and Hardening Options

The uniaxial stress–strain curve for the material is represented by the linear hardening model shown in Fig. 3.4.

For a small–strain analysis (*linear* kinematic formulation), specify *engineering* values for the strain(ε_E) and stress (σ_E). For a finite–strain analysis (*nonlinear* kinematic formulation), specify the uniaxial stress–strain curve using the logarithmic strain, ε, and the true (Cauchy) stress, σ. *For a finite–strain analysis, the user should convert conventional engineering strain, ε_E, and engineering (nominal) stress, σ_E, values for input using the relations*:

$$\sigma = \sigma_E(1 + \varepsilon_E) \tag{3.23}$$

$$\varepsilon = \ln(1 + \varepsilon_E) \tag{3.24}$$

The above conversions assume incompressible, homogeneous deformation. The true stress–true strain curve discussed here assumes homogeneous, uniaxial deformation of the material, i.e., prior to necking. Once necking occurs, the above expressions are no longer applicable. More elaborate corrections, for example those developed by Bridgeman, are required.

Once yielding begins, three strain hardening options are available. The strain hardening option is selected with the β (*beta*) model property. The rate of strain hardening is con-

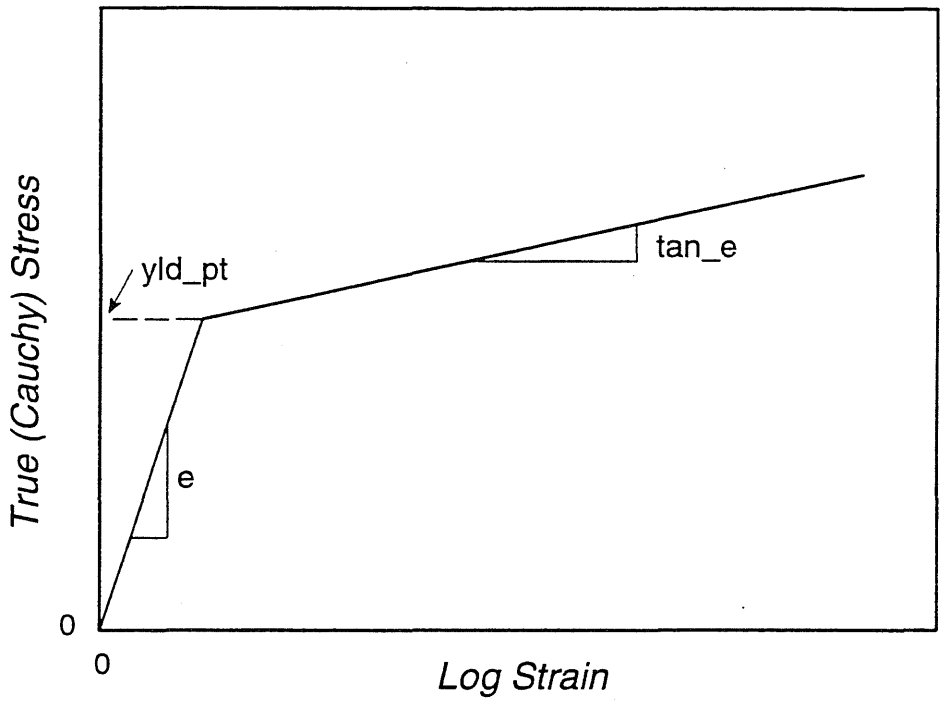


FIG. 3.4—*Uniaxial (tensile) stress–strain curve for the "bilinear" plasticity material model. For finite–strain analysis, input the Cauchy stress and log strain description.*

trolled by the user–specified tangent modulus, E_T, and the value of β. The strain hardening options are:

1. *Isotropic hardening* ($\beta = 1.0$). The radius of the yield surface increases in proportion to the plastic modulus, $H' = EE_T/(E - E_T)$. *This is the default hardening option.*

2. *Kinematic hardening* ($\beta = 0$). The radius of the yield surface remains constant at the initial yield value. The yield surface translates in the direction normal to the surface at the current stress contact point. The rate of translation is governed by the plastic modulus, $H' = EE_T/(E - E_T)$, of the uniaxial stress–strain curve.

3. *Mixed hardening* ($0 < \beta < 1.0$). Part of the strain hardening is isotropic and part is kinematic. The value of β controls the proportion assigned to each hardening model, e.g., $\beta = 0.25$ requests that 25% of the hardening be processed as kinematic and 75% as isotropic.

3.3.2 Model Properties

The properties defined for material model *bilinear* are listed in Table 3.3.

3.3.3 Model Output

By default, the material model prints no messages during computations. If requested, the material model prints the element number and strain point number whenever the effective stress *first* exceeds the specified yield stress. This option is requested with the nonlinear solution parameter *material messages on* (refer to Section 2.9.8)

The model makes available the strain energy density, U_0, to the element routines for subsequent output. U_0 at step n+1 is evaluated using the trapezoidal rule

$$U_0^{n+1} = U_0^n + \frac{1}{2}\left(\boldsymbol{t}^{n+1} + \boldsymbol{t}^n\right) : \Delta \boldsymbol{d} \tag{3.25}$$

where the unrotated Cauchy stresses and unrotated strain increments are adopted for the finite–strain formulation.

Model Property	Keyword	Mode	Default Value
Young's modulus	*e*	Number	0.0
Poisson's ratio	*nu*	Real	0.0
Mass density	*rho*	Real	0.0
Yield stress	*yld_pt*	Number	0.0
Hardening modulus (E_T)	*tan_e*	Number	0.0
Hardening mixity (β)	*beta*	Number	1.0

Table 3.3 Properties for *bilinear* Material Model

3.3.4 Computational Efficiency

The computational routines for this model process elements in blocks of a size matched to the vector length of the computer (i.e., Crays) or to the cache size of the workstation. All model computations are written in vectorized code. Compared to the general rate–dependent Mises model discussed later, this model is significantly faster.

3.3.5 Example

The following example defines the properties for a mild steel material frequently used in fracture models and assigns the material to some elements.

```
structure cct
c
material a516
  properties bilinear e 30000 nu 0.3 yld_pt 60.0 tan_e 300.0,
      rho  7.3e-07
c
number of nodes 25642 22092
c
elements
  14000-22092 type l3disop  linear material a516 order 2x2x2,
                      long bbar
c
```

3.3.6 Plasticity Algorithms

During a time step from state n to state n+1, global equilibrium iterations, designated by i, are performed at a constant external load level to reduce the residual sufficiently close

to zero. Each iteration allows a new estimate of the strain rate to be determined at the state $n+1$ which is associated with the iteration. With this estimate, the stress at the ith update of state $n+1$ is computed. This process is termed the stress recovery and is the principal focus of a material model. For stress updating, the ith estimate of the strain increment over the step is used, $\Delta\varepsilon = \varepsilon^{i}_{n+1} - \varepsilon_n$, which defines the so–called 'path independent' strategy. The current implementation of this model does not use subincrementation schemes which subdivide the strain increment.

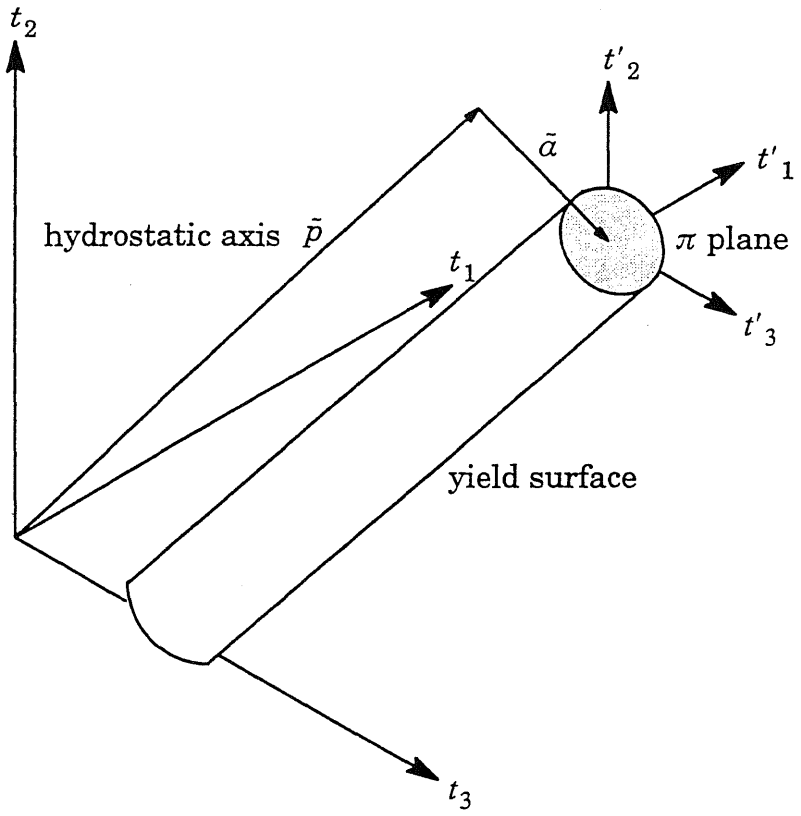


FIG. 3.5—*Mises yield surface in principal stress space*

Also necessary at each global iteration is a constitutive tangent operator that relates stress rate to strain rate, or changes in stress to changes in strain, so that increments of displacement from $n+1$ at $i-1$ to $n+1$ at i may be computed and strain rates estimated. This task is also the responsibility of the material model.

The small–strain plasticity model is based on rate independent, isotropic J_2 flow theory considering both isotropic and kinematic hardening and utilizing a bilinear uniaxial material response. The stress recovery during plastic flow is performed using an elastic predictor–radial return numerical integration scheme (see Key [47], Kreig and Key [52], Dodds [21], Keppel and Dodds [46] for additional details). A consistent rather than a continuum tangent operator is computed for use in the calculation of the element tangent stiffness matrix in order to maintain quadratic convergence in the global nonlinear solution (see Simo and Taylor [75]). The complete algorithm for the stress recovery and the evaluation of the consistent tangent operator at a given material point is developed and outlined in the following discussion.

Stress Recovery

Let t_{ij}, d_{ij}, and a_{ij} be the stress, strain rate, and back stress respectively. Deviator values and norms associated with these tensors are defined by

$$( )'_{ij} = ( )_{ij} - \frac{( )_{kk}}{3}\delta_{ij} \; ; \quad \| ( )_{ij} \| = \sqrt{( )_{ij}( )_{ij}} \tag{3.26}$$

Because the vector corresponding to a_{ij} in principal stress space lies in the π plane of the yield surface, a_{kk} is zero and the deviator of the back stress is the back stress. Accordingly, the deviator relative stress is given as

$$\xi'_{ij} = t'_{ij} - a_{ij} \tag{3.27}$$

allowing the Mises yield surface (Fig. 3.5) to be described by the equation

$$\frac{\xi'_{ij}\xi'_{ij}}{2} - k^2 = 0 \tag{3.28}$$

where k is proportional to the radius of the yield surface in the π plane.

The strain rate is decomposed into elastic and plastic components by the equation

$$d_{ij} = d^e_{ij} + d^p_{ij} \tag{3.29}$$

The unit normal tensor is defined as

$$n_{ij} = \frac{\xi'_{ij}}{\| \xi'_{ij} \|} \tag{3.30}$$

so that plastic strain rate can be described by the equation

$$d^p_{ij} = \lambda n_{ij} \tag{3.31}$$

Increments of the plastic strain will thus be normal to the yield surface in stress space. As a consequence of J_2 flow theory, d^p_{kk}, the change in plastic volume with time, is zero; the deviator plastic strain rate is therefore equal to the plastic strain rate.

The effective plastic strain rate and the effective stress are defined as

$$\dot{\bar{e}}^p = \sqrt{\frac{2}{3} d^p_{ij} d^p_{ij}} \tag{3.32}$$

and

$$q = \sqrt{3J'_2} \; ; \qquad J'_2 = \tfrac{1}{2} t'_{ij} t'_{ij} \tag{3.33}$$

The derivative of the effective stress with respect to the effective strain is the plastic modulus H'. For a Mises yield surface with a bilinear uniaxial stress –strain diagram, H' is given as

$$H' = \frac{E E_T}{E - E_T} \tag{3.34}$$

where E and E_T are Young's modulus and the tangent modulus, respectively. Note that for a bilinear material, E_T and H' are constants.

Along with Eq. (3.31), the evolution equations for the material are given by

$$\dot{\alpha}_{ij} = \frac{2}{3}(1-\beta)H' d^p_{ij} = \frac{2}{3}(1-\beta)H'\lambda n_{ij} \tag{3.35}$$

$$\dot{k} = \frac{\sqrt{2}}{3}\beta H' \parallel d^p_{ij} \parallel = \frac{\sqrt{2}}{3}\beta H'\lambda \tag{3.36}$$

$$\dot{t}'_{ij} = 2G(d'_{ij} - d'^{(p)}_{ij}) \tag{3.37}$$

$$\dot{p} = \frac{\dot{t}_{kk}}{3} = K d_{kk} \tag{3.38}$$

The parameter β controls the type of hardening used in the analysis. It measures the proportion of the hardening which is isotropic, ranging in value between zero and one. Values of β = 0.0, 1.0, and 0.25 indicate pure kinematic hardening , pure isotropic hardening, and 25% isotropic hardening – 75% kinematic hardening. The parameters K and G are the bulk and shear moduli of the material.

The material point is assumed to be strained at a constant rate during the time step. Rate tensors are evaluated at state n+1/2 when integrated to produce an increment over the step. Consequently, the hydrostatic stress p of Eq. (3.38) and the elastic predictor trial deviator stress at state n+1 are computed as

$$^{n+1}p = {}^{n}p + K\Delta t\,{}^{n+1/2}d'_{kk} \tag{3.39}$$

$$^{n+1}t'^{t}_{ij} = {}^{n}t'_{ij} + 2G\Delta t\,{}^{n+1/2}d'_{ij} \tag{3.40}$$

The trial deviator relative stress is defined in terms of the trial deviator stress and the back stress at state n:

$$^{n+1}\xi'^{t}_{ij} = {}^{n+1}t'^{t}_{ij} - {}^{n}\alpha_{ij} \tag{3.41}$$

At this stage, if the material point is elastic, the stress recovery is essentially complete. It remains only to re–combine the hydrostatic stress and the trial deviator stress. If the material point is in the state of plastic flow, using Eq. (3.37) the trial deviator stress is modified by a stress increment corresponding to a radial return to the yield surface in order to calculate the updated deviator stress at state n+1:

$$^{n+1}t'_{ij} = {}^{n+1}t'^{t}_{ij} - 2G\Delta t\,{}^{n+1/2}d'^{(p)}_{ij} = {}^{n+1}t'^{t}_{ij} - 2G\lambda\Delta t n_{ij} \tag{3.42}$$

For simplicity of notation, in Eq. (3.42) and later λ is taken as evaluated at state n+1/2 and n_{ij} at state n+1. The updated back stress at state n+1 follows from Eq. (3.35):

$$^{n+1}\alpha_{ij} = {}^{n}\alpha_{ij} + \frac{2}{3}(1-\beta)H'\lambda\Delta t n_{ij} \tag{3.43}$$

Combining Eqs. (3.42) and (3.43), the deviator relative stress at state n+1 is expressed as

$$^{n+1}\xi'_{ij} = {}^{n+1}\xi'^{t}_{ij} - \lambda\Delta t\left[\, 2G + \frac{2}{3}(1-\beta)H' \,\right] n_{ij} \tag{3.44}$$

Specifying the unit normal tensor at state n+1 to be

$$n_{ij} = \frac{{}^{n+1}\xi'^{t}_{ij}}{\parallel {}^{n+1}\xi'^{t}_{ij} \parallel} \tag{3.45}$$

and substituting into Eq. (3.44) leads to the relationship

$$\| {}^{n+1}\xi'_{ij} \| = \| {}^{n+1}\xi'^{t}_{ij} \| - \lambda\Delta t\left[2G + \tfrac{2}{3}(1-\beta)H' \right] \tag{3.46}$$

Recasting Eq. (3.28) as

$$\| {}^{n+1}\xi'_{ij} \| - \sqrt{2}\ {}^{n+1}k = 0 \tag{3.47}$$

and noting from Eq. (3.36) that

$${}^{n+1}k = {}^{n}k + \frac{\sqrt{2}}{3}\beta H'\lambda\Delta t \tag{3.48}$$

allows Eq. (3.46) to be manipulated, yielding

$$\lambda\Delta t = \frac{\| {}^{n+1}\xi'^{t}_{ij} \| - \sqrt{2}\ {}^{n}k}{2G\left(1 + \frac{H'}{3G}\right)} \tag{3.49}$$

$\lambda\Delta t$ is backsubstituted into the preceding equations to resolve all stresses and state variables. It is possible to directly compute $\lambda\Delta t$ because H' is a constant signifying that the effective stress is a linear function of the effective plastic strain. If this function is not linear, then it would be necessary to iterate to determine $\lambda\Delta t$.

A flow chart illustrating the steps required for the recovery of stresses is displayed in Fig. 3.6. The algorithm above is implemented in a vector form. The six components of stress tensors are arrayed in the order { 11 22 33 12 23 13 }. Strain tensors correspond to vectors with identical ordering but with diagonal terms doubled to form engineering strains.

Consistent Tangent Operator

The tangent operator required for the calculation of element tangent stiffness matrices satisfies the following relationship between stress rate and the total strain rate:

$$\dot{t}_{ij} = C_{ijkl}\ d_{kl} \tag{3.50}$$

For a material point in the elastic state, the isotropic tangent operator is given by

$$C^{E}_{ijkl} = K\delta_{ij}\delta_{kl} + 2G\left[\tfrac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \tfrac{1}{3}\delta_{ij}\delta_{kl} \right] \tag{3.51}$$

Once the material point experiences plastic flow, the tangent operator is given by

$$C^{EP}_{ijkl} = K\delta_{ij}\delta_{kl} + 2G\left[\tfrac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \tfrac{1}{3}\delta_{ij}\delta_{kl} \right] - 2G\gamma n_{ij}n_{kl} \tag{3.52}$$

$$\gamma = \frac{1}{\left[1 + \frac{H'}{3G}\right]} \tag{3.53}$$

The operator of Eq. (3.52) is termed the continuum tangent operator. Its use is compatible with an exact integration of the evolution equations, which are continuum in nature. However, the elastic predictor–radial return procedure for stress updating does not represent an exact integration; it is in essence a secant approach. Not surprisingly, use of the continuum tangent operator leads to a degradation in the quadratic convergence characteristic of the global Newton iterations. Simo and Taylor [75] established a tangent operator compatible with the elastic predictor–radial return algorithm which preserves the qua-

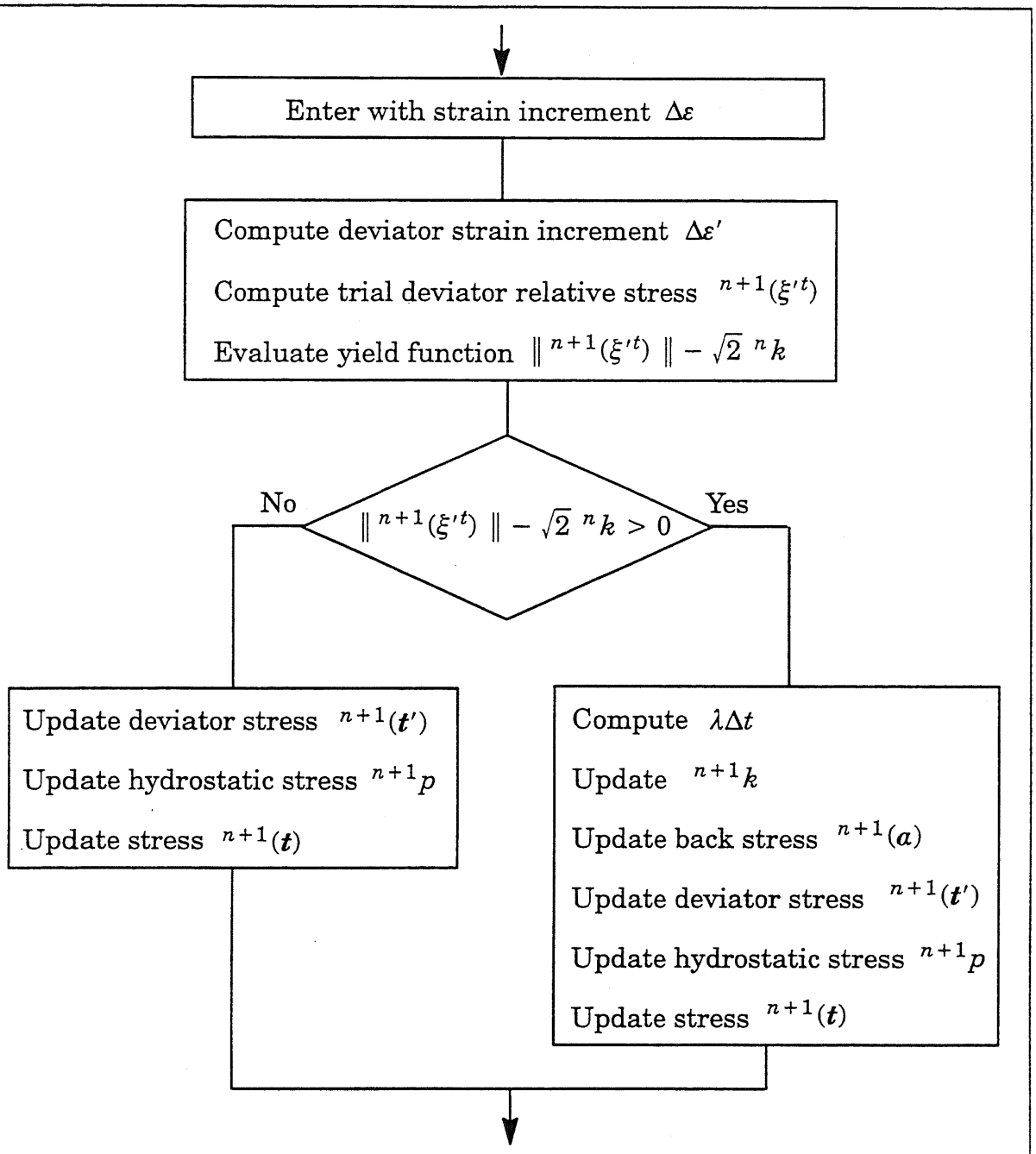


FIG. 3.6—*Stress recovery procedure for bilinear (Mises) material model*

dratic convergence. It is often termed the *consistent tangent* operator, and it is the tangent operator employed in WARP. The consistent tangent operator is given by the following equations:

$$C^{EP}_{ijkl} = K\delta_{ij}\delta_{kl} + 2GB\left[\ \tfrac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \tfrac{1}{3}\delta_{ij}\delta_{kl}\ \right] - 2G\bar{\gamma}n_{ij}n_{kl} \tag{3.54}$$

$$B = \frac{\left[\sqrt{2}\ {}^{n+1}k + \frac{2}{3}(1-\beta)H'\lambda\Delta t\right]}{\| {}^{n+1}\xi'^{t}_{ij} \|} \; ; \quad \bar{\gamma} = \frac{1}{\left[1 + \frac{H'}{3G}\right]} - (1 - B) \tag{3.55}$$

In the code, the above tangent operator is applied in a 6x6 matrix form that relates a stress vector to an engineering strain vector.

3.4 Material Model Type: *mises*

This material model extends the capabilities offered by the *bilinear* (mises) model to include power–law hardening and viscoplastic effects.

The *mises* model provides two options for the inviscid uniaxial (tensile) stress–strain curve: (1) linear hardening after yield and (2) pure power–law hardening after an initially linear response prior to yield. This generalized form of Mises plasticity supports only isotropic hardening.

To introduce a rate dependence into the model, we adopt a power–law viscoplastic relationship suitable for ductile metals undergoing large amounts of plastic straining. The viscoplastic strain rate is given by

$$\dot{\varepsilon}^{vp} = D\left[\left(\frac{q}{\sigma_e}\right)^m - 1\right] \tag{3.56}$$

where D and m are user–specified material constants, q denotes the rate–dependent (uniaxial) tensile stress and σ_e the inviscid (uniaxial) tensile stress. The viscosity term is often written in the form $D=1/\eta$. For a moderately rate–sensitive material, such as an A533B pressure vessel steel at 100° C, typical values of D and m are 1.0 (*in./in./sec*) and 35, respectively. In the simplest case, σ_e is specified to remain constant at the yield stress, σ_0 (the linear hardening model with $E_T = 0$). More generally, σ_e is a linear or power–law function of ε^{vp}.

The following sections describe needed parameters to utilize the *mises* material model. Additional details for rate–dependent features of the model are then provided. All other aspects of the formulation follow those of the *bilinear* model described in the previous section.

3.4.1 Stress–Strain Curves and Hardening

The inviscid uniaxial stress–strain curve for the material is represented by the linear hardening model shown in Fig. 3.4, by a linear, power–law model shown in Fig. 3.7 or by a general piecewise linear curve of the type shown in Fig. 2.1. To maintain continuous values of E_T between the linear and power–law regions, a small (cubic) transition curve is inserted automatically in the description of the stress–strain curve. For the piecewise linear curve, E_T is taken (automatically) to vary linearly over a small distance on each side of a curve point at which the tangent modulus exhibits a jump discontinuity.

For a small–strain analysis (*linear* kinematic formulation), specify *engineering* values for the strain(ε_E) and stress (σ_E). For a finite–strain analysis (*nonlinear* kinematic formulation), specify the uniaxial stress–strain curve using the logarithmic strain, ε, and the true (Cauchy) stress, σ. *For a finite–strain analysis, the user should convert conventional engineering strain, ε_E, and engineering (nominal) stress, σ_E, values for input using the relations:*

$$\sigma = \sigma_E(1 + \varepsilon_E) \tag{3.57}$$

$$\varepsilon = \ln(1 + \varepsilon_E) \tag{3.58}$$

The above conversions assume incompressible, homogeneous deformation. The true stress–true strain curve discussed here assumes homogeneous, uniaxial deformation of the material, i.e., prior to necking. Once necking occurs, the above expressions are no longer applicable. More elaborate corrections, for example those developed by Bridgeman, are required.

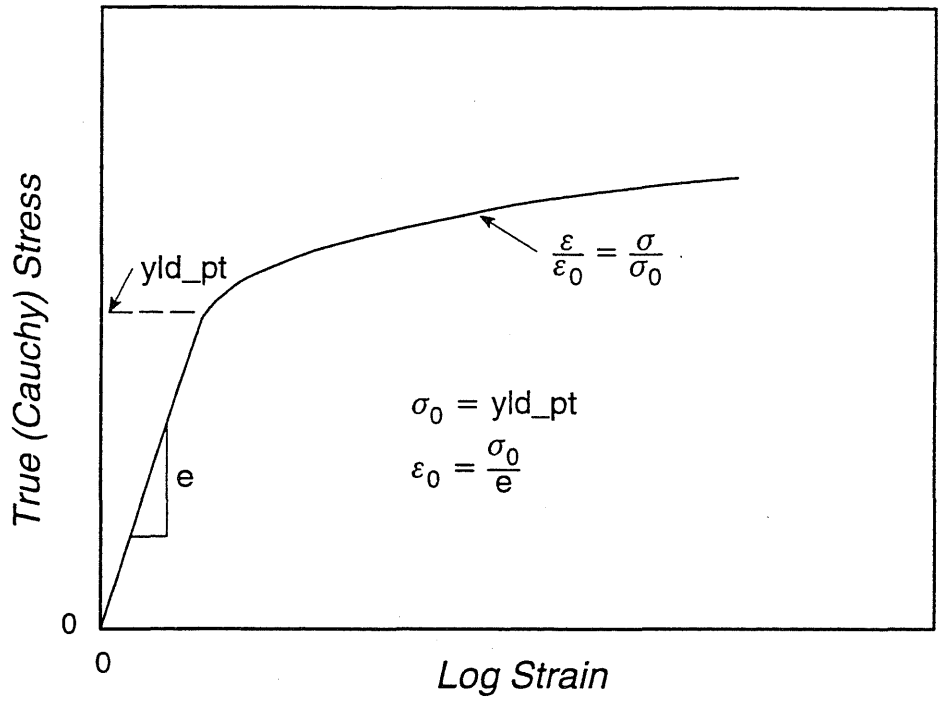


FIG. 3.7—*Power–law form of the inviscid uniaxial (tensile) stress–strain curve for the "mises" plasticity material model. For finite–strain analysis, input the Cauchy stress and log strain description.*

Once yielding begins, the invisicid hardening follows the isotropic model.

3.4.2 Model Properties

The properties defined for material model *mises* are listed in Table 3.4. When the *curve* option is invoked to indicate a separately defined piecewise–linear stress–strain curve, Young's modulus must still be specified.

3.4.3 Model Output

By default, the material model prints no messages during computations. If requested, the material model prints the element number and strain point number whenever the effective stress *first* exceeds the specified yield stress. This option is requested with the nonlinear solution parameter *material messages on* (refer to Section 2.9.8)

The model makes available the strain energy density, U_0, to the element routines for subsequent output. U_0 at step n+1 is evaluated using the trapezoidal rule

$$U_0^{n+1} = U_0^n + \tfrac{1}{2}\left(\boldsymbol{t}^{n+1} + \boldsymbol{t}^n\right) : \Delta \boldsymbol{d} \tag{3.59}$$

where the unrotated Cauchy stresses and unrotated strain increments are adopted for the finite–strain formulation.

3.4.4 Computational Efficiency

The computational routines for this model process elements in blocks of a size matched to the vector length of the computer (i.e., Crays) or to the cache size of the workstation. The

majority of model computations are written in vectorized code. The local Newton loop to solve the scalar consistency equation executes in scalar mode (efforts will be made soon to vectorize this operation since the impact on performance is noticeable). The use of a piecewise–linear stress–strain curve may adversely impact computational efficiency to an even greater extent due to the logic of searching the points defining the curve.

Our testing indicates the piecewise–linear model combined with the viscoplastic option can reduce the convergence rate of global Newton iterations. No such degradation is experienced with purely linear hardening or power–law hardening combined with viscoplasticity.

This model is computationally much less efficient than the simple *bilinear* model of the previous sections.

Model Property	Keyword	Mode	Default Value
Young's modulus	*e*	Number	0.0
Poisson's ratio	*nu*	Real	0.0
Mass density	*rho*	Number	0.0
Yield stress	*yld_pt*	Number	0.0
Hardening modulus (E_T)	*tan_e*	Number	0.0
Power law exponent (n)	*n_power*	Number	0.0
Reference strain rate (D)	*ref_eps*	Number	0.0
Viscous exponent (m)	*m_power*	Number	0.0
Stress–strain curve	*curve*	Number	0

Table 3.4 Properties for *mises* Material Model

3.4.5 Example

The following example defines the properties for two mild steels material frequently used in fracture models and assigns the material to some elements.

```
structure cct
c
stress-strain curve 3
   36 0.0012,  36 0.01,  50 0.05,
   55 0.10, 60 0.30c
c
material a533b
  properties mises e 30000 nu 0.3 yld_pt 60.0 n_power 10,
      rho  7.3e-07 ref_eps 40 m_power 20
```

```
c
material a36
  properties mises e 30000 nu 0.3 curve 3 rho  7.3e-07
c
number of nodes 25642 22092
c
elements
  14000-22092 type l3disop linear material a533b order 2x2x2,
                      long bbar
c
```

3.4.6 Plasticity Algorithms

The formulation and implementation of the general, rate–dependent *mises* model differs from the *bilinear* model in the complexity of computing the term $\lambda\Delta t$. Eqs. (3.44) and (3.45) define the deviatoric terms of the updated stress state as a return to the new yield surface along the direction of trial elastic deviator (which for Mises is normal to the updated yield surface). Eq. (3.46) then represents the scalar product of each side of Eq. (3.44) and defines the so–called *scalar consistency equation* for determination $\lambda\Delta t$. Using the relationships of Eqs. (3.32) and (3.33), the consistency equation may be written in the simpler form

$$\sigma^T - 3G\Delta\varepsilon^p - \sigma_e(\varepsilon^p_{n+1}) \equiv 0 \tag{3.60}$$

where σ^T is the equivalent uniaxial stress computed from the elastic trial stress at the step $(n+1)$ [see Eq. (3.40)], G is the elastic shear modulus, $\Delta\varepsilon^p$ is the unknown increment of plastic strain over the step, $\Delta\varepsilon^p = \varepsilon^p_{n+1} - \varepsilon^p_n$, and $\sigma_e(\varepsilon^p_{n+1})$ is the equivalent (Mises) stress corresponding to the plastic strain at the end of the step. The analyst provides the functional relationship, $\sigma_e(\varepsilon^p)$ through the stress–strain curves described previously [to make this simpler, the analyst actually specifies $\sigma_e(\bar{\varepsilon})$ rather than $\sigma_e(\varepsilon^p)$].

The following sections describe the techniques used to solve Eq. (3.60) first for the inviscid case with power–law hardening and then for the general viscoplastic case. Once $\Delta\varepsilon^p$ and $\sigma_e(\varepsilon^p_{n+1})$ are determined from these calculations, the correction of the trial deviatoric stress to the yield surface and the inclusion of hydrostatic stress terms proceeds exactly as for the bilinear model. Also discussed here is the proper definition of H' for use in the consistent tangent operator to model the power–law hardening and viscoplastic cases.

Rate–Independent Consistency Equation

When the uniaxial tensile response has other than linear (constant) hardening, Eq. (3.60) is nonlinear in the plastic strain increment, $\Delta\varepsilon^p$, and requires an iterative solution. For the power–law model of the uniaxial response

$$\frac{\varepsilon_e}{\varepsilon_0} = \frac{\sigma_e}{\sigma_0}, \qquad \varepsilon_e \le \varepsilon_0 \tag{3.61a}$$

$$\frac{\varepsilon_e}{\varepsilon_0} = \left(\frac{\sigma_e}{\sigma_0}\right)^N, \quad \varepsilon_e > \varepsilon_0 \tag{3.61b}$$

where σ_0 and ε_0 are reference yield stress and strain levels that also define E. To evaluate Eq. (3.61b) given an estimate of $\Delta\varepsilon^p$, substitute

$$\varepsilon_{e,n+1} = \frac{\sigma_{e,n+1}}{E} + \varepsilon^p_n + \Delta\varepsilon^p \tag{3.62}$$

into Eq. (3.61b) which provides the needed expression for $\sigma_e(\varepsilon^p_{n+1})$ as

$$\frac{\sigma_{e,n+1}}{\sigma_0} = \left(\frac{\sigma_{e,n+1}}{\sigma_0}\right)^N - \frac{\varepsilon^p_{n+1}}{\varepsilon_0} \ . \tag{3.63}$$

This is a simple, nonlinear equation solved readily for $\sigma_{e,n+1}$ using a local Newton scheme. Define the residual, R, of Eq. (3.63) by

$$R = \left(\frac{\sigma_{e,n+1}}{\sigma_0}\right)^N - \frac{\sigma_{e,n+1}}{\sigma_0} - \frac{\varepsilon^p_{n+1}}{\varepsilon_0} \ . \tag{3.64}$$

For the (i) estimate of $\sigma_{e,n+1}$, find the change in R such that $R + dR$=0 where

$$dR = \frac{\partial R}{\partial \sigma_{e,n+1}} d\sigma_{e,n+1} \ . \tag{3.65}$$

The required derivative is found to be

$$\frac{\partial R}{\partial \sigma_{e,n+1}} = \frac{1}{E}\left[N\left(\frac{\sigma_{e,n+1}}{\sigma_0}\right)^{N-1} - 1\right] . \tag{3.66}$$

Successive improvements to the value of $\sigma_{e,n+1}$ are thus

$$\sigma^{(i+1)}_{e,n+1} = \sigma^{(i)}_{e,n+1} + d\sigma^{(i)}_{e,n+1} = \sigma^{(i)}_{e,n+1} - \frac{R^{(i)}}{\left.\frac{\partial R}{\partial \sigma_{e,n+1}}\right|_{(i)}} \tag{3.67}$$

Iterations continue until convergence on $\sigma_{e,n+1}$ is achieved. A suitable convergence test is

$$\left|\sigma^{(i+1)}_{e,n+1} - \sigma^{(i)}_{e,n+1}\right| \le tol\ \sigma^{(i+1)}_{e,n+1} \tag{3.68}$$

where we specify 10^{-6} for *tol*. A robust starting estimate $\sigma^{(1)}_{e,n+1}$ is given by

$$\sigma^{(1)}_{e,n+1} = \sigma_0\left(\frac{\varepsilon^p_{n+1} + \varepsilon_0}{\varepsilon_0}\right)^{1/N} \ . \tag{3.69}$$

We find convergence is achieved in at most 3 iterations over Eq. (3.64) – (3.68). The instantaneous plastic modulus, needed for the consistent tangent, is given by

$$H'_{n+1} = \frac{EE_{T,n+1}}{E - E_{T,n+1}} \tag{3.70}$$

where

$$E_{T,n+1} = \frac{E}{N}\left(\frac{\sigma_{e,n+1}}{\sigma_0}\right)^{(1-N)} \ . \tag{3.71}$$

With a converged value for $\Delta\varepsilon^p$ given by the solution of Eq. (3.60), the updated stress state is computed by the usual radial return to the updated yield surface (isotropic hardening)

$$\{\sigma\}_{n+1} = \{\sigma^T\}_{n+1} - \frac{3G\Delta\varepsilon^p}{\sigma^T}\{S^T\}_{n+1} \ , \quad \{\ \} \textit{ implies a } 6x1 \textit{ vector} \tag{3.72}$$

where $\{S^T\}_{n+1}$ is the deviatoric portion of the trial elastic stress state $\{\sigma^T\}_{n+1}$.

Rate–Dependent Consistency Equation

To introduce a rate dependence into the model, re–write the consistency equation of Eq. (3.60) in the form

$$\sigma^T - 3G\Delta\varepsilon^{vp} - q(\varepsilon^{vp}_{n+1}, \sigma_{e,n+1}, \Delta t) \equiv 0 \tag{3.73}$$

where q denotes the rate–dependent equivalent stress, $\sigma_{e,n+1}$ becomes the inviscid equivalent stress (which may be a nonlinear function of ε^{vp}) and $\Delta t = t_{n+1} - t_n$. We adopt a power–law viscoplastic relationship suitable for ductile metals undergoing large amounts of plastic straining. The viscoplastic strain rate is given by

$$\dot{\varepsilon}^{vp} = \frac{1}{\eta}\left[\left(\frac{q}{\sigma_e}\right)^m - 1\right] \tag{3.74}$$

where η and m are material constants. The viscosity term is often written in the form $D=1/\eta$. In the simplest case, σ_e is specified to remain constant at the yield stress, σ_0. More generally, σ_e is a nonlinear function of ε^{vp}.

The integration of Eq. (3.74) with a backward Euler procedure yields

$$\Delta\varepsilon^{vp} = \frac{\Delta t}{\eta}\left[\left(\frac{q_{n+1}}{\sigma_{e,n+1}}\right)^m - 1\right] \tag{3.75}$$

which is solved for q_{n+1}

$$q_{n+1} = \sigma_{e,n+1}\left[\left(\frac{\eta\Delta\varepsilon^{vp}}{\Delta t}\right) + 1\right]^{1/m} . \tag{3.76}$$

We observe in Eq. (3.76) that as $\eta/\Delta t \rightarrow 0$ the inviscid solution is recovered. By using a local Newton solver, the rate–dependent consistency equation, Eq. (3.73), is solved for $\Delta\varepsilon^{vp}$ with q_{n+1} defined as in Eq. (3.76). Convergence is again achieved in a few iterations. With $\Delta\varepsilon^{vp}$ known, the updated stress state at $n+1$ is given by the usual radial–return to the yield surface

$$\{\sigma\}_{n+1} = \{\sigma^T\}_{n+1} - \frac{3G\Delta\varepsilon^{vp}}{\sigma^T}\{\boldsymbol{S}^T\}_{n+1} . \tag{3.77}$$

To form the consistent tangent, the instantaneous plastic modulus for the rate–dependant equivalent stress is required

$$H'_{q,n+1} = \left.\frac{dq}{d\varepsilon^{vp}}\right|_{n+1} \tag{3.78}$$

We must obtain $H_{q,n+1}$ by differentiating the *algorithm* that defines the evolution of q. From Eq. (3.75) we obtain

$$d\dot{\varepsilon}^{vp}_{n+1} = d(\Delta\varepsilon^{vp}) = \frac{m\Delta t}{\eta}\left(\frac{q_{n+1}}{\sigma_{e,n+1}}\right)^{m-1}\left(\frac{\sigma_{e,n+1}dq_{n+1} - q_{n+1}d\sigma_{e,n+1}}{\sigma^2_{e,n+1}}\right) . \tag{3.79}$$

By substituting for $d\sigma_{e,n+1}$ in terms of the plastic modulus for the inviscid response Eq. (3.70)

$$d\sigma_{e,n+1} = H'_{n+1} d\varepsilon^{vp}_{n+1} \tag{3.80}$$

Eq. (3.79) is solved for $H'_{q,n+1}$ as

$$H'_{q,n+1} = \left.\frac{dq}{d\varepsilon^{vp}_{n+1}}\right|_{n+1} = \frac{\eta\sigma_{e,n+1}}{m\Delta t}\left(\frac{q_{n+1}}{\sigma_{e,n+1}}\right)^{1-m} + \left(\frac{q_{n+1}}{\sigma_{e,n+1}}\right)H'_{n+1} \tag{3.81}$$

The plastic modulus $H'_{q,n+1}$ provides the value of H' that appears in the consistent tangent operator, Eq. (3.54). Note that as $\eta/\Delta t \rightarrow 0$ in Eq. (3.81), $q_{n+1}/\sigma_{e,n+1}$ must also $\rightarrow 1$ and the inviscid H'_{n+1} is recovered.

3.5 Material Model Type: *gurson*

This material model implements the Gurson–Tvergaard (GT) plastic potential to predict the response of an elastic–plastic solid containing voids (Gurson [28], see Tvergaard [81] for a comprehensive review). The basis for the model derives from analyses of a single cell containing a centered spherical void of initial volume fraction f_0. The void volume fraction, f, increases under loading and eventually leads to a gradual loss of stress carrying capacity for a macroscopic material element. With this model, a material element effectively contains a void of volume fraction f and (solid) matrix material of volume fraction $(1–f)$. The matrix response follows the material's uniaxial (tensile) stress–strain properties which can be represented in one of several ways and can also include viscoplastic effects.

The GT yield condition is given by

$$g(\sigma_e, \sigma_m, \bar{\sigma}, f) = \left(\frac{\sigma_e}{\bar{\sigma}}\right)^2 + 2q_1 f \cosh\left(\frac{3q_2\sigma_m}{2\bar{\sigma}}\right) - \left(1 + q_3 f^2\right) = 0 \tag{3.82}$$

where σ_e denotes the (Mises) equivalent (macroscopic) stress, σ_m is the mean (macroscopic) stress, $\bar{\sigma}$ is the (Mises) equivalent stress of the matrix and f is the current void fraction. Factors q_1, q_2, and q_3 introduced by Tvergaard improve the model predictions for periodic arrays of cylindrical and spherical voids. When $f = 0$ the yield condition reduces to conventional J_2 plasticity. The computations for this model should be carried out with the finite strain formulation (*nonlinear* element property) so that Cauchy stresses are used in the evaluation of Eq. (3.82).

The current implementation employs a backward Euler technique developed by Aravas [2] to integrate the plasticity rate equations. This procedure is unconditionally stable thereby permitting the use of larger load increments than is possible with traditional forward Euler and semi–explicit procedures. However, the use of exceptionally large load increments can lead to non–convergence of Newton loops within the model to resolve updated state variables.

This model offers two forms for the uniaxial (tensile) response of the matrix material, an option to include viscoplasticity in the response and a strain–controlled nucleation model to initiate new voids at severe levels of plastic deformation.

3.5.1 Stress–Strain Curves

The inviscid uniaxial stress–strain curve for the material is represented by the linear hardening model shown in Fig. 3.4, by a linear, power–law model shown in Fig. 3.7 or by a general piecewise linear curve of the type shown in Fig. 2.1. To maintain continuous values of E_T between the linear and power–law regions, a small (cubic) transition curve is inserted automatically in the description of the stress–strain curve. For the piecewise linear curve, E_T is taken (automatically) to vary linearly over a small distance on each side of a curve point at which the tangent modulus exhibits a jump discontinuity.

For a small–strain analysis (*linear* kinematic formulation), specify *engineering* values for the strain(ε_E) and stress (σ_E). For a finite–strain analysis (*nonlinear* kinematic formulation), specify the uniaxial stress–strain curve using the logarithmic strain, ε, and the true (Cauchy) stress, σ. *For a finite–strain analysis, the user should convert conventional engineering strain, ε_E, and engineering (nominal) stress, σ_E, values for input using the relations:*

$$\sigma = \sigma_E(1 + \varepsilon_E) \tag{3.83}$$

$$\varepsilon = \ln(1 + \varepsilon_E) \tag{3.84}$$

The above conversions assume incompressible, homogeneous deformation. The true stress–true strain curve discussed here assumes homogeneous, uniaxial deformation of the material, i.e., prior to necking. Once necking occurs, the above expressions are no longer applicable. More elaborate corrections, for example those developed by Bridgeman, are required.

3.5.2 Viscoplasticity

To introduce a rate dependence into the matrix response, we adopt a power–law viscoplastic relationship suitable for ductile metals undergoing large amounts of plastic straining. The viscoplastic strain rate is given by

$$\dot{\varepsilon}^{vp} = D\left[\left(\frac{q}{\sigma_e}\right)^m - 1\right] \tag{3.85}$$

where D and m are user–specified material constants, q denotes the rate–dependent (uniaxial) tensile stress and σ_e the inviscid (uniaxial) tensile stress. The viscosity term is often written in the form D=1/η. For a moderately rate–sensitive material, such as an A533B pressure vessel steel at 100° C, typical values of D and m are 1.0 (*in./in./sec*) and 35, respectively. In the simplest case, σ_e is specified to remain constant at the yield stress, σ_0 (the linear hardening model with $E_T = 0$). More generally, σ_e is a linear or power–law function of ε^{vp}.

We recommend that the piecewise–linear description of the tensile stress–strain curve not be used with the viscoplastic option at this time. Convergence of the global Newton iterations is sometimes reduced; no such problems occur for the linear or power–law hardening options.

3.5.3 Nucleation Model

The volume fraction of voids increases over an increment of load due to continued growth of existing voids and due to the formation of new voids caused by interfacial decohesion of inclusions or second phase particles. Thus,

$$df = df_{growth} + df_{nucleation} \ . \tag{3.86}$$

The growth component is defined by the current volume fraction of voids and the macroscopic change in void fraction is (the matrix material satisfies plastic incompressibility)

$$df_{growth} = (1 - f)d\boldsymbol{\varepsilon}^p : \mathbf{I} = (1 - f)d\varepsilon_p \ . \tag{3.87}$$

We adopt an evolution model for nucleation based on current plastic strain in the matrix

$$df_{nucleation} = \mathcal{A}(\bar{\varepsilon}^p)d\bar{\varepsilon}^p \ . \tag{3.88}$$

Chu and Needleman suggest a form for $\mathcal{A}$ as

$$\mathcal{A} = \frac{f_N}{s_N\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{\bar{\varepsilon}^p - \varepsilon_N}{s_N}\right)^2\right] \tag{3.89}$$

where the nucleation strain follows a normal distribution with a mean value ε_N and a standard deviation s_N with the volume fraction of void nucleating particles given by f_N. A simpler form of Gurson's model which neglects nucleation is derived by setting $\mathcal{A} \equiv 0$ (three fewer material parameters are then required).

Model Property	Keyword	Mode	Default Value
Young's modulus	*e*	Number	0.0
Poisson's ratio	*nu*	Real	0.0
Mass density	*rho*	Number	0.0
Yield stress	*yld_pt*	Number	0.0
Hardening modulus (E_T)	*tan_e*	Number	0.0
Power law exponent (n)	*n_power*	Number	0.0
Reference strain rate (D)	*ref_eps*	Number	0.0
Viscous exponent (m)	*m_power*	Number	0.0
Initial porosity (f_0)	*f_0*	Number	0.0
Yield function parameter q_1	*q1*	Number	1.5
Yield function parameter q_2	*q2*	Number	1.0
Yield function parameter q_3	*q3*	Number	2.25
Include nucleation of new voids	*nucleation*	Logical	.False.
Nucleation parameter f_N	*f_n*	Number	0.04
Nucleation parameter s_N	*s_n*	Number	0.10
Nucleation parameter ε_N	*e_n*	Number	0.30
Put element in *killable* list	*killable*	Logical	.False.
Suppress step size cutbacks	*no_cutback*	Logical	.False.
Stress–strain curve	*curve*	Number	0

Table 3.5 Properties for *gurson* Material Model

3.5.4 Element Extinction

Under increasing deformation, the void volume fraction reaches a level at which the element capacity to resist stress decreases precipitously. This final stage of deformation just prior to material separation is not realistically predicted with the GT model (even though the numerical computations remain stable to very high levels of f, approaching 0.5).

Chapter 5 describes an extinction procedure which removes elements from the model and slowly reduces the remaining tractions to zero. This occurs when the void fraction f reaches a user–specified level, denoted f_E. The crack growth procedures in Chapter 5 apply only to elements which have an associated *gurson* material with the material logical property *killable* specified.

When the *killable* property is not specified, the stress updating process continues with f increasing. Eventually, the model routines may request load step reductions to stabilize the state update process.

3.5.5 Adaptive Step Sizes

By default, the *gurson* model routines request a global load step *cutback* when the state update process fails to converge. If the nonlinear solution parameter *adaptive on* is in effect (see Section 2.9.4), the global load step reduction occurs and subsequent *gurson* computations nearly always converge. If the nonlinear solution parameters have *adaptive off*, the *gurson* routines print an message describing the convergence problem and terminate the analysis.

Users may disable the automatic cutback requests in the material model through the model property *no_cutback*. If the state update process fails to converge, the model immediately terminates execution of the program.

3.5.6 Model Properties

The properties defined for material model *gurson* are listed in Table 3.5. When the *curve* option is invoked to indicate a separately defined piecewise–linear stress–strain curve, Young's modulus must still be specified.

3.5.7 Model Output

By default, the material model prints no messages during computations. If requested, the material model prints the element number and strain point number whenever the effective stress *first* exceeds the specified yield stress. This option is requested with the nonlinear solution parameter *material messages on* (refer to Section 2.9.8). Messages about requests for global load step reductions are always printed.

The model makes available the strain energy density, U_0, to the element routines for subsequent output. U_0 at step n+1 is evaluated using the trapezoidal rule

$$U_0^{n+1} = U_0^n + \frac{1}{2}\left(\boldsymbol{t}^{n+1} + \boldsymbol{t}^n\right) : \Delta \boldsymbol{d} \tag{3.90}$$

where the unrotated Cauchy stresses and unrotated strain increments are adopted for the finite–strain formulation.

The element stress output contains up to three values for the material model "state" variables. These values for the *gurson* material are:

mat_val1: matrix plastic strain, $\bar{\varepsilon}^p$

mat_val2: matrix equivalent stress, $\bar{\sigma}$
mat_val3: current void fraction, f

3.5.8 Computational Efficiency

The computational routines for this model process elements in blocks of a size matched to the vector length of the computer (i.e., Crays) or to the cache size of the workstation. The majority of model computations are written in vectorized code. The local Newton loops to solve the scalar consistency equations execute in scalar mode. The use of a piecewise–linear stress–strain curve may adversely impact computational efficiency to an even greater extent due to the logic of searching the points defining the curve.

Our testing indicates the piecewise–linear model combined with the viscoplastic option can reduce the convergence rate of global Newton iterations. No such degradation is experienced with purely linear hardening or power–law hardening combined with viscoplasticity.

This model is computationally less efficient than the mises model of the previous section.

3.5.9 Example

The following example defines the properties for a mild steel material frequently used in fracture models and assigns the material to some elements.

```
structure cct
c
material a533b
  properties gurson e 30000 nu 0.3 yld_pt 60.0 n_power 10,
      rho  7.3e-07 ref_eps 40 m_power 20 f_0 0.005 killable
c
number of nodes 25642 22092
c
elements
  14000-22092 type l3disop  linear material a533b order 2x2x2,
                      long bbar
c
```

3.5.10 Plasticity Algorithms

Material Elasticity and Yield Criterion

The material is elastically isotropic and for a specified increment of total (macroscopic) strain,

$$\Delta\varepsilon = \varepsilon_{n+1} - \varepsilon_n \tag{3.91}$$

the trial (T) elastic stress state is defined by

$$\sigma^T_{n+1} = \sigma_n + \boldsymbol{D}^e : \Delta\boldsymbol{\varepsilon} \ . \tag{3.92}$$

We use bold italics to denote a second–order tensor, bold roman indicates a symmetric fourth–order tensor, and : denotes the operator consistent for the order of tensors involved, e.g., $(\mathbf{C} : \boldsymbol{B})_{ij} = C_{ijkl}B_{kl}$. *Italic* symbols denote scalar variables. All tensor components are given with respect to a fixed, Cartesian system.

We define $\boldsymbol{S}^T_{n+1}$ as the deviatoric component of $\boldsymbol{\sigma}^T_{n+1}$ from which the equivalent (macroscopic) stress is given by

$$q^T_{n+1} = \left(\frac{3}{2}S^T_{ij}S^T_{ij}\right)^{1/2}_{n+1} \ . \tag{3.93}$$

Similarly, the trial hydrostatic stress is given by

$$p^T_{n+1} = -\frac{1}{3}\sigma^T_{n+1} : \boldsymbol{I} = -\frac{1}{3}(\sigma_{11} + \sigma_{22} + \sigma_{33})^T_{n+1} \ . \tag{3.94}$$

Gurson's yield function is given by

$$g = \left(\frac{q}{\bar{\sigma}}\right)^2 + 2q_1 f \cosh\left(-\frac{3}{2}\frac{q_2 p}{\bar{\sigma}}\right) - (1 + q_3 f^2) = 0 \tag{3.95}$$

where q_1, q_2, q_3 are material constants, f is the current void fraction and $\bar{\sigma}$ is the current (Mises) equivalent stress of the matrix. Most often, $q_1 = 1.5$, $q_2 = 1$ and $q_3 = q_1^2$ to match the response of discrete hole growth models under pure shear and pure hydrostatic loading. We evaluate the yield criterion for the trial elastic state using current values of the (scalar) state variables

$$g^T_{n+1} = g(q^T_{n+1}, p^T_{n+1}, f_n, \bar{\sigma}_n) \ . \tag{3.96}$$

The material loading is defined by

$$\begin{aligned} g^T_{n+1} &< 0 \quad \textit{linear - elastic} \\ g^T_{n+1} &\geq 0 \quad \textit{plastic loading} \end{aligned} \ . \tag{3.97}$$

Unloading from a previously plastic state is treated inelastically such that

$$\sigma_{n+1} = \sigma^T_{n+1} \tag{3.98}$$

with the internal state variables retaining their values at n.

Plasticity Rate Equations

When the material is loading plastically as indicated by Eq. (3.97), the macroscopic continuum flow rule is expressed as

$$d\varepsilon^p = d\Lambda \frac{\partial g}{\partial \sigma} \tag{3.99}$$

where $d\Lambda$ is the (positive) plastic multiplier. Integration of the plastic strain rate over the step using the backward Euler procedure yields

$$\Delta\varepsilon^p = \Delta\Lambda \frac{\partial g}{\partial \sigma}\Big|_{n+1} \ . \tag{3.100}$$

The derivative of the yield function in Eq. (3.100) is written in the terms of the hydrostatic and deviatoric contributions to provide

$$\Delta\varepsilon^p = \Delta\Lambda\left(-\frac{1}{3}\frac{\partial g}{\partial p}\boldsymbol{I} + \frac{\partial g}{\partial q}\boldsymbol{n}\right)\Big|_{n+1} \tag{3.101}$$

where the unit normal $\boldsymbol{n}$ is defined by

$$\boldsymbol{n}_{n+1} = \frac{3}{2q_{n+1}}\boldsymbol{S}_{n+1} \ . \tag{3.102}$$

To simplify subsequent expressions, we introduce definitions for the (scalar) volumetric and deviatoric plastic strain as

$$\Delta\varepsilon_p = -\Delta\Lambda\left(\frac{\partial g}{\partial p}\right)\Big|_{n+1} \tag{3.103}$$

$$\Delta\varepsilon_q = \Delta\Lambda\left(\frac{\partial g}{\partial q}\right)\Big|_{n+1} \tag{3.104}$$

and substitute into Eq. (3.101) to give

$$\Delta\boldsymbol{\varepsilon}^p = \tfrac{1}{3}\Delta\varepsilon_p\boldsymbol{I} + \Delta\varepsilon_q\boldsymbol{n}_{n+1} \ . \tag{3.105}$$

If the updated stress state at $n+1$ is written in terms of the usual volumetric and deviatoric components

$$\boldsymbol{\sigma}_{n+1} = -p_{n+1}\boldsymbol{I} + \boldsymbol{S}_{n+1} \tag{3.106}$$

then with the notation defined by Eq. (3.102), the updated stress state also may be written in the form

$$\boldsymbol{\sigma}_{n+1} = -p_{n+1}\boldsymbol{I} + \boldsymbol{S}_{n+1} = -p_{n+1}\boldsymbol{I} + \tfrac{2}{3}q_{n+1}\boldsymbol{n}_{n+1} \tag{3.107}$$

In terms of the trial elastic stress state, the updated stress state may be constructed as follows:

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}_n + \mathbf{D}^e : (\Delta\boldsymbol{\varepsilon} - \Delta\boldsymbol{\varepsilon}^p) \tag{3.108}$$

where the first two terms on the right side combine to define the trial elastic state such that

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^T_{n+1} - \mathbf{D}^e : \Delta\boldsymbol{\varepsilon}^p \ . \tag{3.109}$$

The negative term on the right side defines a plastic stress correction for the trial elastic stress. Using Eq. (3.105), this term may be expressed in the form

$$\mathbf{D}^e : \Delta\boldsymbol{\varepsilon}^p = K\Delta\varepsilon_p\boldsymbol{I} + 2G\Delta\varepsilon_q\boldsymbol{n}_{n+1} \tag{3.110}$$

where K and G are the elastic bulk and shear modulus, respectively. Substitution of Eq. (3.110) into Eq. (3.109) provides a convenient form of the updated stress as

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^T_{n+1} - K\Delta\varepsilon_p\boldsymbol{I} - 2G\Delta\varepsilon_q\boldsymbol{n}_{n+1} \ . \tag{3.111}$$

In the above equation, the trial elastic stress state is corrected (i.e. *returned*) to the updated yield surface. In deviatoric space, the return direction is along the normal defined by $\boldsymbol{n}_{n+1}$. Using the material elasticity, we also have the following relations for the updated hydrostatic and equivalent stress

$$p_{n+1} = p^T_{n+1} + K\Delta\varepsilon_p \tag{3.112}$$

$$q_{n+1} = q^T_{n+1} - 3G\Delta\varepsilon_q \tag{3.113}$$

which prove very useful in the numerical processes described below.

The key step in the backward Euler scheme defines the return normal direction as the deviatoric direction of the trial elastic state as, using Eq. (3.93) and Eq. (3.102),

$$\boldsymbol{n}_{n+1} = \frac{3}{2q^T_{n+1}}\boldsymbol{S}^T_{n+1} \tag{3.114}$$

which yields finally

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}^T_{n+1} - K\Delta\varepsilon_p\boldsymbol{I} - \frac{3G\Delta\varepsilon_q}{q^T_{n+1}}\boldsymbol{S}^T_{n+1} \ . \tag{3.115}$$

This choice for the return direction simplifies greatly numerical solution for the updated stress state; in 3–D the number of unknowns is reduced by 6, the number of unique terms in $\boldsymbol{n}_{n+1}$. A more detailed discussion of similar return mapping algorithms is given by Simo.

From Eq. (3.111), a knowledge of $\Delta\varepsilon_p$, $\Delta\varepsilon_q$ fully defines the updated stress state. The numerical solution must determine values for these scalar parameters so that $\Delta\boldsymbol{\varepsilon}^p$ satisfies the flow rule over the step and the updated stresses satisfy the yield criterion. In the process of computing $\Delta\varepsilon_p$, $\Delta\varepsilon_q$, the internal state variables are updated as well.

Internal State Variables

Gurson's model includes a set of state variables which partition the macroscopic stress–strain into the matrix material and the "smeared" voids. These state variables define the microscopic plastic strain in the matrix and the current volume fraction of voids.

Plastic Strain in the Matrix

Plastic work in the matrix is taken to be a relative fraction, 1–f, of macroscopic plastic work such that

$$(1-f)\bar{\sigma}d\bar{\varepsilon}^p = \boldsymbol{\sigma} : d\boldsymbol{\varepsilon}^p \tag{3.116}$$

where $\bar{\varepsilon}^p$ denotes the matrix plastic strain. This rate equation is integrated over the step using backward Euler and solved for the increment of plastic strain in the matrix

$$\Delta\bar{\varepsilon}^p = \frac{\boldsymbol{\sigma}_{n+1} : \Delta\boldsymbol{\varepsilon}^p}{(1-f_{n+1})\bar{\sigma}_{n+1}} \tag{3.117}$$

where the numerator simplifies considerably to provide

$$\Delta\bar{\varepsilon}^p = \frac{-p_{n+1}\Delta\varepsilon_p + q_{n+1}\Delta\varepsilon_q}{(1-f_{n+1})\bar{\sigma}_{n+1}} \ . \tag{3.118}$$

A variety of models for the evolution of $\bar{\sigma}$, the equivalent matrix stress, with increasing plastic strain in the matrix may be defined. Both inviscid and power–law viscoplastic models are discussed in a subsequent section.

Evolution of Void Fraction

The volume fraction of voids increases over an increment due to continued growth of existing voids and due to the formation of new voids caused by interfacial decohesion of inclusions or second phase particles. Thus,

$$df = df_{growth} + df_{nucleation} \ . \tag{3.119}$$

The growth component is defined by the current volume fraction of voids and the macroscopic change in void fraction is (the matrix material satisfies plastic incompressibility)

$$df_{growth} = (1-f)d\boldsymbol{\varepsilon}^p : \underline{\boldsymbol{I}} = (1-f)d\varepsilon_p \ . \tag{3.120}$$

We adopt an evolution model for nucleation based on current plastic strain in the matrix

$$df_{nucleation} = \mathcal{A}(\bar{\varepsilon}^p)d\bar{\varepsilon}^p \ . \tag{3.121}$$

Chu and Needleman suggest a form for $\mathcal{A}$ as

$$\mathcal{A} = \frac{f_N}{s_N\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{\bar{\varepsilon}^p - \varepsilon_N}{s_N}\right)^2\right] \tag{3.122}$$

where the nucleation strain follows a normal distribution with a mean value ε_N and a standard deviation s_N with the volume fraction of void nucleating particles given by f_N. A simpler form of Gurson's model which neglects nucleation is derived by setting $\mathcal{A} \equiv 0$ (three fewer material parameters are then required).

Equation (3.120) and its component terms are integrated using backward Euler to obtain

$$\Delta f = (1 - f_{n+1})\Delta\varepsilon_p + \mathcal{A}(\bar{\varepsilon}^p_{n+1})\Delta\bar{\varepsilon}^p \quad . \tag{3.123}$$

Equations (3.118) and (3.123) comprise a pair of coupled, nonlinear algebraic equations to update the microscopic state variables f, $\bar{\varepsilon}^p$ for specified values of the macroscopic plastic strains $\Delta\varepsilon_p$, $\Delta\varepsilon_q$.

Response of the Matrix Material

A variety of models for the evolution of $\bar{\sigma}$, the equivalent matrix stress, may be defined. Here we consider two inviscid models, the first of which is

$$\bar{\sigma} = \sigma_0 + H'\bar{\varepsilon}^p \tag{3.124}$$

where H' is the specified (constant) plastic hardening modulus (H' may be zero) and σ_0 is the specified uniaxial yield stress. The second inviscid model is a simple power–law with initially linear response

$$\frac{\bar{\varepsilon}}{\varepsilon_0} = \frac{\bar{\sigma}}{\sigma_0}, \qquad \bar{\varepsilon} \le \varepsilon_0 \tag{3.125}$$

$$\frac{\bar{\varepsilon}}{\varepsilon_0} = \left(\frac{\bar{\sigma}}{\sigma_0}\right)^N, \quad \bar{\varepsilon} > \varepsilon_0 \tag{3.126}$$

where the total equivalent strain in the matrix, $\bar{\varepsilon}$, is simply $\bar{\varepsilon} = \bar{\sigma}/E + \bar{\varepsilon}^p$ and $E = \sigma_0/\varepsilon_0$. Eq. (3.126) is solved iteratively for $\bar{\sigma}$ with a local Newton loop for a given value of plastic strain in the matrix, $\bar{\varepsilon}^p$. The plastic modulus, H', is then found by

$$H' = \frac{EE_T}{E - E_T} \tag{3.127}$$

where the tangent modulus is defined from Eq. (3.126) by

$$E_T = \frac{E}{N}\left(\frac{\bar{\sigma}}{\sigma_0}\right)^{(1-N)} \quad . \tag{3.128}$$

To model a viscoplastic matrix material, we adopt a power–law model of the form

$$\dot{\bar{\varepsilon}}^p = \frac{1}{\eta}\left[\left(\frac{\bar{\sigma}}{\sigma_e}\right)^m - 1\right] \tag{3.129}$$

where η and m are material constants and σ_e is the inviscid equivalent stress for the matrix. The viscosity term is often written in the form D=1/η. In the simplest case, σ_e is specified to remain constant at the yield stress, σ_0. More generally, σ_e is a nonlinear function of $\bar{\varepsilon}^p$ along the lines of Eq. (3.126).

The integration of Eq. (3.129) with a backward Euler procedure yields

$$\Delta\bar{\varepsilon}^p = \frac{\Delta t}{\eta}\left[\left(\frac{\bar{\sigma}_{n+1}}{\sigma_{i,n+1}}\right)^m - 1\right] \tag{3.130}$$

where subscript i denotes the inviscid response at the same plastic strain in the matrix. This expression is solved directly for $\bar{\sigma}_{n+1}$

$$\bar{\sigma}_{n+1} = \sigma_{i,n+1}\left[\left(\frac{\eta\Delta\bar{\varepsilon}^p}{\Delta t}\right) + 1\right]^{1/m} . \tag{3.131}$$

We observe in Eq. (3.131) that as $\eta/\Delta t \to 0$ the inviscid solution is recovered. Each of the above models for $\bar{\sigma}_{n+1}$ are functions of the plastic strain in the matrix and can thus be resolved during the solution for $\Delta\varepsilon_p$, $\Delta\varepsilon_q$. The plastic modulus is given by

$$H' = \frac{d\bar{\sigma}}{d\bar{\varepsilon}^p}\Big|_{n+1} = \frac{\eta\sigma_i}{m\Delta t}\left(\frac{\bar{\sigma}}{\sigma_i}\right)^{1-m} + \left(\frac{\bar{\sigma}}{\sigma_i}\right)H_i' \tag{3.132}$$

where all terms on the RHS of (3.132) are evaluated at n+1.

Summary of Updating Process

The stress updating process requires computation of a set of stresses defined by Eq. (3.115) for which the flow conditions given in Eq. (3.103) and Eq. (3.104) are satisfied consistent with updated values of the internal state variables. The proportionality factor $\Delta\Lambda$ is eliminated by dividing Eq. (3.103) by Eq. (3.104) to define the relationship between the increments of volumetric and deviatoric plastic strain as

$$\Delta\varepsilon_p\left(\frac{\partial g}{\partial q}\right) + \Delta\varepsilon_q\left(\frac{\partial g}{\partial p}\right) = 0 . \tag{3.133}$$

This relationship together with satisfaction of the yield criterion at n+1 using stresses of Eq. (3.115)

$$g_{n+1} = g(q_{n+1}, \bar{\sigma}_{n+1}, p_{n+1}, f_{n+1}) = 0 \tag{3.134}$$

defines a pair of nonlinear algebraic equations for numerical solution. The primary unknown variables in these two equations are the macroscopic plastic strains $\Delta\varepsilon_p$, $\Delta\varepsilon_q$.

These equations are solved iteratively using Newton's method. Given estimates for $\Delta\varepsilon_p$ and $\Delta\varepsilon_q$, the updated stress state, p_{n+1} and q_{n+1}, are given by Eq. (3.112) and Eq. (3.113). The internal state variables, $\bar{\varepsilon}^p$, $\bar{\sigma}$ and f, are updated to n+1 by solving these three equations simultaneously, Eqs. (3.118) and (3.123) are repeated for clarity)

$$\Delta\bar{\varepsilon}^p = \bar{\varepsilon}^p_{n+1} - \bar{\varepsilon}^p_n = \frac{-p_{n+1}\Delta\varepsilon_p + q_{n+1}\Delta\varepsilon_q}{(1-f_{n+1})\bar{\sigma}_{n+1}} . \tag{3.135}$$

$$\Delta f = f_{n+1} - f_n = (1-f_{n+1})\Delta\varepsilon_p + \mathcal{A}(\bar{\varepsilon}^p_{n+1})\Delta\bar{\varepsilon}^p . \tag{3.136}$$

$$\bar{\sigma}_{n+1} = \bar{\sigma}(\bar{\varepsilon}^p_{n+1}) \tag{3.137}$$

The numerical complexity in updating $\bar{\varepsilon}^p$ and f depends on the form adopted for $\bar{\sigma}(\bar{\varepsilon}^p)$ and whether or not the nucleation component of f is included. If $\bar{\sigma}(\bar{\varepsilon}^p)$ of the form defined by Eq. (3.124) is adopted and $\mathcal{A} \equiv 0$, the above three equation reduce to a single linear equation for $\Delta\bar{\varepsilon}^p$ after which Δf is found directly as well. In other cases, another level of Newton's iterations is required to resolve $\Delta\bar{\varepsilon}^p$ and Δf consistent with $\bar{\sigma}$.

Chapter 4

Domain Integrals

4.1 Introduction

Finite element methods are especially powerful for computing linear and nonlinear fracture mechanics parameters. For linear analyses, the stress–intensity factors, K_I, are readily determined from the energy release–rate interpretation of the J–integral (Rice [69], Knowles and Sternberg [51], Budiansky and Rice [10]). For nonlinear analyses, the intensity of deformation along the crack front is generally characterized by the Crack Tip Opening Displacement (CTOD) and/or a pointwise value of the J–integral. In two–dimensions, the J–integral sets the amplitude of the singular field near a sharp crack tip, as given by the HRR solutions (Rice and Rosengren [68], Hutchinson [43]), under certain limiting conditions involving material constitutive behavior and the extent of plastic deformation relative to the uncracked ligament size. In three–dimensions, the situation is not nearly so clear; the nature of near–tip fields in 3–D remains a focus of current research. Remote from traction free surfaces, the crack front fields may closely resemble those of plane–strain; near free surfaces the fields exhibit strong 3–D effects. However, purely mechanical arguments concerning the energy flux show that the J–integral provides a local energy release rate independent of the exact singular form of the near tip fields. Under these conditions, J characterizes the *crack driving force.*

This chapter describes the Domain Integral (DI) capabilities implemented in WARP to compute J–integral values in 3–D (Mode I) following solution for a load step (Li, et al. [53], Moran and Shih [56] [57], Shih, et al. [74]). The DI procedures are more general and simpler for the analyst to specify than the earlier Virtual Crack Extension (VCE) technique (Parks [66], Helen [32]). The analyst defines nodal values of a *weight function* which may be interpreted as the motion of material near the crack front due to a *virtual* crack extension. The numerical computations then require evaluation a volume integral over elements in 3–D which includes the energy density, the stress field, the displacement, velocity, acceleration fields and the weight functions. Weight functions over elements are constructed from the specified nodal values using conventional isoparametric procedures. This quickly becomes an onerous task; however, capabilities are included for automatic generation of the weight function values which greatly simplify J computations in 3–D crack configurations. An option for the user to specify directly the weight function values on a node–by–node basis remains available.

The procedures described in this chapter may be invoked following a linear or nonlinear solution for a load step (static/dynamic). The user provides input commands to define a "domain" for evaluation of J followed by a *compute domain integral* command. The specification of a single "automatic" domain by the user typically causes J evaluations over many separate domains of increasing distance from the crack front. The computed J–value for each domain and the variations J–values between the domains are printed (minimum J, maximum J and average J for assessment of path independence).

The DI procedures currently implemented have these features/limitations:

- the material response is considered nonlinear elastic when the material model employs an incremental plasticity theory (this is a very common assumption and avoids unnecessary complications that arise from the explicit partial derivative of the stress work density)

- the kinetic energy and accelerations of crack region material in dynamic loading are included in J
- the effects of finite strains and finite rotations at material points are included in J
- the effects of *rapid* crack growth are not included in J ("slow" crack growth under dynamic loading is supported)
- the effects of loads applied to the crack faces are included in J using an approximate technique (these terms maintain path independence for domains remote from the front)
- thermal loading and other initial strain/stress effects are not included in J

The next section of this chapter provides a summary of the theoretical basis for the DI method. Other sections describe the numerical algorithms to evaluate the volume integrals and input commands. Sample output from a computation illustrates the various information available.

4.2 Background

4.2.1 Local Energy Release Rates

A *local* value of the mechanical energy release rate, denoted $J(s)$, at each point s on a planar, non–growing crack front under general dynamic loading is given by

$$J(s) = \lim_{\Gamma \to 0} \int_{\Gamma} \left[(W + T) n_1 - P_{ji} \frac{\partial u_i}{\partial X_1} n_j \right] d\Gamma \tag{4.1}$$

where W and T are the stress–work density and the kinetic energy density per unit volume at t=0; Γ is a vanishingly small contour which lies in the principal normal plane at s, and n is the unit vector normal to Γ (see Fig. 4.1). P_{ji} denotes the non–symmetric 1st Piola–Kirchhoff (1st PK) stress tensor which is work conjugate to the displacement gradient expressed on the t=0 configuration, $\partial u_i / \partial X_j$, i.e., the stress–work rate is simply $P_{ij} \partial u_i / \partial X_j$ per unit volume at t=0. All field quantities are expressed in the local orthogonal coordinate system, X_1—X_2—X_3, at location s on the crack front.

This important result was first derived by Eshelby [22] and independently by Cherepanov [13], and later by others considering only mechanical energy balance for a local translation of the crack front in the X_1 direction (Mode I). Any form of loading (including crack face tractions) and arbitrary material behavior is permitted when $\Gamma \to 0$. All proposed forms of path independent integrals (contour, area, volume) for application in fracture mechanics derive from Eq. (4.1) by specialization of the loading and material behavior (see for example, Amestoy et al. [1], Bakker [5], Carpenter et al. [12], de Lorenzi [19] and Kishimoto et al. [48]).

Moran and Shih [56][57] have proven the *local* path independence of J on the actual shape of Γ in the limit as $\Gamma \to 0^+$. To have both path independence and a non–vanishing, finite value, the integrand of Eq. (4.1) must have order $1/r$. The quantity J defined by Eq. (4.1) has no direct relationship to the form of the near–tip strain–stress fields, except for very limited circumstances. For plane–stress and plane–strain conditions, with nonlinear elastic material response and small–strain theory, J of Eq. (4.1) simplifies to the well–known J–integral due to Rice [69] that exhibits *global* path independence. Under the additional limitation of small–scale yielding (SSY), J sets the amplitude of the HRR singular fields. The role of J as a single parameter which characterizes the near tip strain–stress fields for arbitrary loading (static, thermal, dynamic) and 3–D configurations is a topic of much current research.

The stress–work density (W) per unit initial volume may be defined in terms of the mechanical strains as

$$W = |\boldsymbol{F}| \int_0^t \left(\boldsymbol{t} : \boldsymbol{d}^m \right) dt \tag{4.2}$$

where $|\boldsymbol{F}|$ denotes the determinant of the deformation gradient $\boldsymbol{F} = \partial \boldsymbol{x} / \partial \boldsymbol{X}$, $\boldsymbol{t}$ denotes the unrotated Cauchy stress and $\boldsymbol{d}$ is the *mechanical* fraction of the unrotated rate of deformation tensor. The kinetic energy density is given directly by

$$T = \tfrac{1}{2} \varrho \left(\frac{\partial u_i}{\partial t} \right)^2 \tag{4.3}$$

where ϱ is the material mass density (sum on i) in the initial configuration at t=0.

The direct evaluation of Eq. (4.1) is cumbersome in a finite element model due to the geometric difficulties encountered in defining a contour that passes through the integration points. Such a contour is desired since the most accurate stress and strain quantities are available at the integration points. Moreover, the limiting definition of the contour requires extensive mesh refinement near the crack tip to obtain meaningful numerical results. The next section develops the Domain Integral equivalent of Eq. (4.1) which is naturally suited for finite element models.

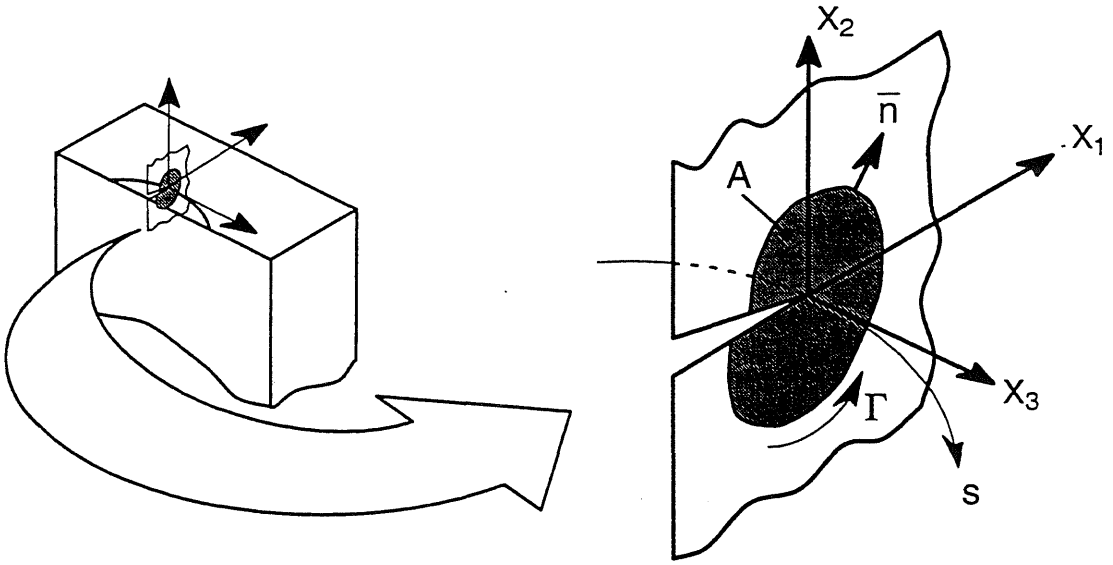


FIG. 4.1—*Local J-integral in 3-D.*

4.2.2 Domain Integral Formulation

By using a weight function, which may be interpreted as a virtual displacement field, the contour integral of Eq. (4.1) is converted into an area integral for two dimensions and into a volume integral for three dimensions (Li, et al. [53], Nikishkov and Atluri [65]). The resulting expressions are (see Fig. 4.2):

$$\overline{J}_{a-c} = \int_{s_a}^{s_c} [\, J(s)\; q_t(s) \,]\; ds = \overline{J}_1 + \overline{J}_2 + \overline{J}_3 \tag{4.4}$$

where the each integral is defined by

$$\overline{J}_1 = \int_{V_0} \left(P_{ji} \frac{\partial u_i}{\partial X_k} \frac{\partial q_k}{\partial X_j} - W \frac{\partial q_k}{\partial X_k} \right) dV_0 \tag{4.5}$$

$$\overline{J}_2 = - \int_{V_0} \left(\frac{\partial W}{\partial X_k} - P_{ji} \frac{\partial^2 u_i}{\partial X_j \partial X_k} \right) q_k \; dV_0 \tag{4.6}$$

$$\overline{J}_3 = - \int_{V_0} \left(T \frac{\partial q_k}{\partial X_k} - \varrho \frac{\partial^2 u_i}{\partial t^2} \frac{\partial u_i}{\partial X_k} q_k + \varrho \frac{\partial u_i}{\partial t} \frac{\partial^2 u_i}{\partial t \partial X_k} q_k \right) dV_0 \tag{4.7}$$

q_k denotes a component of the vector weight function in the k coordinate direction, $q_t(s)$ represents the resultant value of the weight function at point s on the crack front, V_0 represents the volume of the domain surrounding the crack tip in the (undeformed) configuration at t=0, and s denotes positions along the crack front segment.

The vector function q is directed parallel to the direction of crack extension. When all field quantities of the finite element solution are transformed to the local crack front coordinate system at point s, and Mode I extension is considered, only the q_1 term of the weight function is non–zero.

Body forces (other than inertial loading) are assumed to be zero for simplicity. The treatment of crack face tractions involves an additional integral discussed subsequently. $J(s)$ is the local energy release rate that corresponds to the perturbation at s, $q_t(s)$. Figure 4.2 shows a typical domain volume defined for an internal segment along a three–dimensional surface crack.

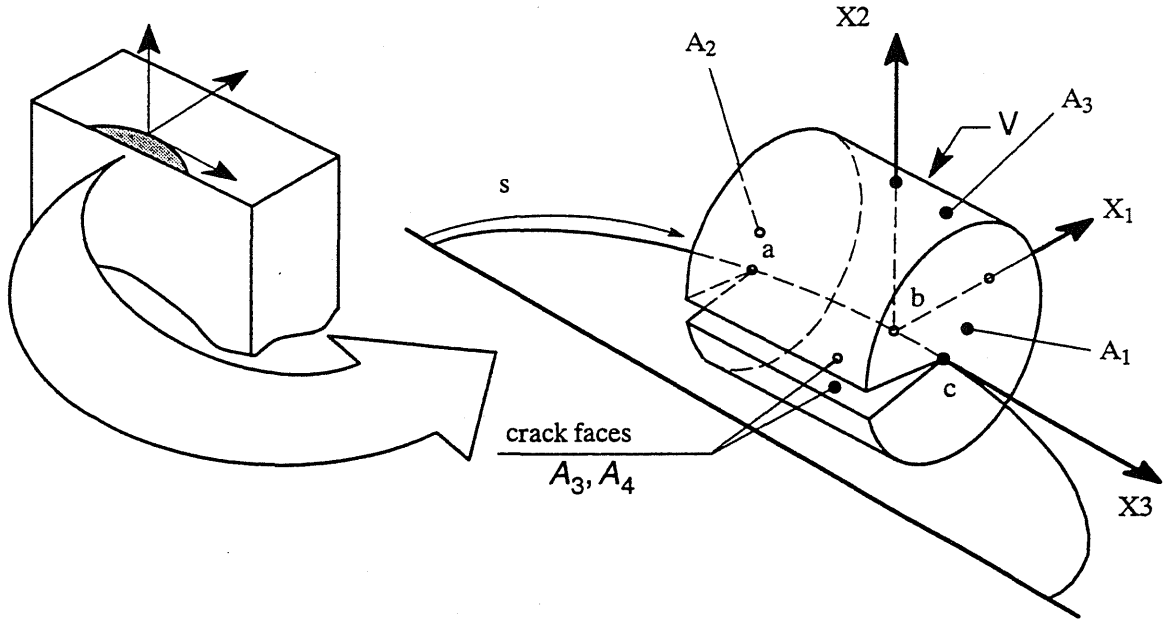


FIG. 4.2—*Finite volume for use in Domain Integral formulation*

The q–function must vanish on the surfaces A_1,A_2 and A_3 in Fig. 4.2 for the development of Eqs. (4.5) through (4.7) from (4.4). This requirement makes area integrals (line integrals in two dimensions) defined on these surfaces vanish. Fig. 4.3 shows the variation in amplitude of a valid q–function for the domain shown in Fig. 4.2. All material over which the q–function and its first derivative are non–zero must be included in the volume integrals. The value of q at each point in the volume, V_0, is readily interpreted as the virtual displacement of a material point due to the virtual extension of the crack front, $q_t(s)$.

An approximate value of $J(s_b)$ is obtained by applying the mean–value theorem over the interval $s_a < s < s_c$. The pointwise value of the J–integral at s_b is given by (see Fig. 4.3):

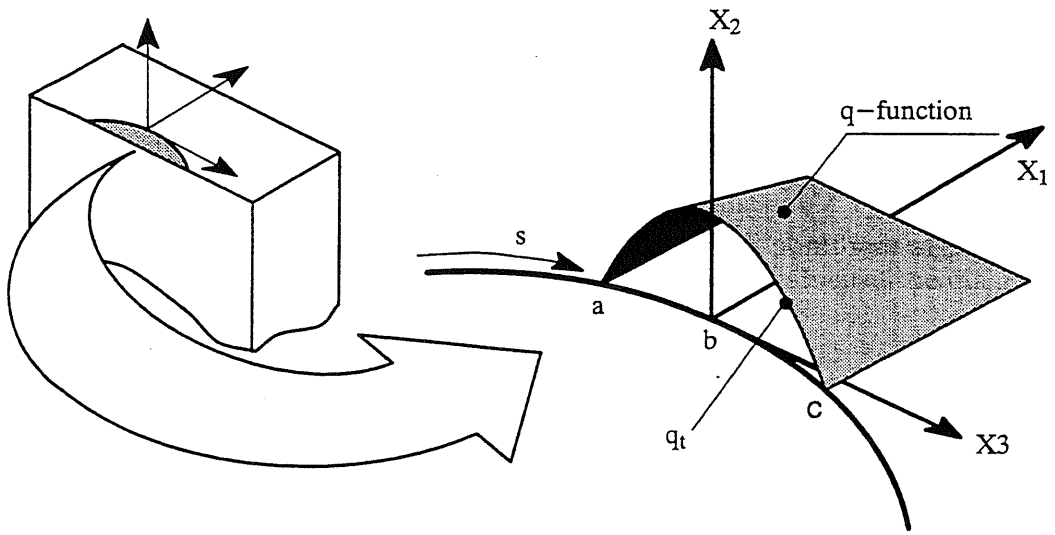


FIG. 4.3—*Variation of weight function, q, over volume at crack front*

$$J(s = b) \approx \frac{\int_{s_a}^{s_c} J(s)\, q_t(s)\, ds}{\int_{s_a}^{s_c} q_t(s)\, ds} = \frac{\overline{J}}{A_q} \tag{4.8}$$

where $\overline{J}$ is the energy released due to the crack–tip perturbation, $q_t(s)$. The increase in crack–area corresponding to this perturbation, A_q, is simply the integral of $q_t(s)$ along the crack front from s_a to s_c.

For common through crack test specimens, e.g. SE(B), C(T), the crack front is generally straight or only slightly curved. For such crack geometries, the average J for the entire crack front value is obtained by the application of a uniform $q_t(s)$ across the full crack front.

The above volume integrals are evaluated by Gauss quadrature. Derivatives of the q–function over each finite element in V are computed by standard isoparametric techniques from specified values of q at element nodes. The higher order derivatives are computed by either: 1) extrapolating Gauss point values to the element nodes and applying standard isoparametric techniques or, 2) interpolating the Gauss point values to a lower order integration within the element.

4.2.3 Domain Form of the *J*–Integral: Discussion

In a small displacement gradient formulation, the $\overline{J}_2$ integral vanishes for an elastic material (linear or nonlinear) in the absence of thermal strains as shown in the following manner. We replace the 1st PK stresses by the conventional (symmetric) stress tensor applicable when strains and displacement gradients are small. Then

$$-P_{ji}\frac{\partial^2 u_i}{\partial X_j \partial X_1} \approx -\sigma_{ij}\frac{\partial^2 u_i}{\partial X_j \partial X_1} \tag{4.9}$$

By exchanging the order of differentiation, inserting the (symmetric) small–strain tensor and using symmetry of σ_{ij}, the second term in Eq. (4.6) is rewritten as:

$$-\sigma_{ij}\frac{\partial^2 u_i}{\partial X_j \partial X_1} = -\sigma_{ij}\frac{\partial}{\partial X_1}\left(\frac{\partial u_i}{\partial X_j}\right) = -\sigma_{ij}\frac{\partial \varepsilon_{ij}}{\partial X_1} \tag{4.10}$$

The chain rule is now evoked to expand the first term in Eq. (4.6), again assuming small–displacement gradients. The resulting derivative of strain energy density with respect to strain is the stress tensor for elastic materials. The result is:

$$\frac{\partial W}{\partial X_1} = \frac{\partial W}{\partial \varepsilon_{ij}}\frac{\partial \varepsilon_{ij}}{\partial X_1} = \sigma_{ij}\frac{\partial \varepsilon_{ij}}{\partial X_1} \tag{4.11}$$

The two terms defining the integrand of $\overline{J}_2$ thus sum to zero for elastic materials.

Dynamic loading effects appear in the $\overline{J}_3$ term of the domain integral representation of the J–integral. The first term in $\overline{J}_3$ provides the flux of the kinetic energy in the direction of the crack propagation. The second and third terms arise from the explicit partial derivative, $(\partial/\partial X_1)$, of the kinetic energy density. The second term contains material accelerations and the third term is identified with the spatial gradient of the velocities. The second term, containing the material accelerations, has been found to make significant contributions to the total $\overline{J}$–integral for non–propagating cracks. This term is similar in form to domain integrals that accommodate ordinary body forces.

For an elastic structure under static loading (without any thermal strains), $\overline{J}_2$ and $\overline{J}_3$ are identically zero. For incremental (load path dependent) plasticity, the deviation of $\overline{J}_2$ from zero indicates the degree of non–proportional loading experienced over the domain of integration.

For many practical cases, the loading produces nearly proportional material histories within the domain of integration; in such cases the very small contribution of $\overline{J}_2$ is neglected. Shih, Moran and Nakamura [74] neglected $\overline{J}_2$ for J–integral calculations. Vargas and Dodds show that up to 15% of the J–integral in a 2–dimensional static case can be due to $\overline{J}_2$ for incremental plasticity models when the plastic strains and the elastic strains within the domain are similar in magnitude. For larger plastic strains, however, this difference diminishes to less than 0.1%, which justifies the use of $\overline{J}_2 + \overline{J}_3$ as an approximation to Eq. (4.4) for large amounts of plastic deformation. The contribution of $\overline{J}_2$ in the presence of thermal strain gradients within the integration domain is significant.

The derivation of Eqs. (4.4) through (4.7) is mathematically rigorous. Provided sufficient resolution of the crack–tip stress–strain fields exists for accurate numerical integration, the calculated $\overline{J}$–integral equals the weighted $J(s)$, where $J(s)$ is the contour definition in the limit as the contour shrinks onto the crack tip. For a given $q_t(s)$, i.e., the crack front variation of the weighting function, many combinations of domain volume and distribution of the q–function are possible. Thus, similar to *path independence* arguments for the contour J–integral, *domain independence* arguments apply for the domain J–integral. In practice, several domains defined concentrically about the crack tip are evaluated to insure domain independence of the computed J–integral. In the general case of thermal loading and inelastic material response all three components of the J–integral are required for the calculated value to be domain independent.

Summary

The $\overline{J}_2$ term in WARP3D domain integral processors is assumed ≡ 0 for all cases. Numerical evaluation of the $\overline{J}_1$ integral requires only straightforward application of isoparametric element techniques once the computed field quantities are transformed from the global X—Y—Z coordinate system to the $X_1 - X_2 - X_3$ system at point s on the front. In this simplified form, Eq. (4.5) becomes

$$\overline{J}_1 = \int_{V_0} \left(P_{ji} \frac{\partial u_i}{\partial X_1} \frac{\partial q}{\partial X_j} - W \frac{\partial q}{\partial X_1} \right) dV_0 \ . \tag{4.12}$$

Similarly, the kinetic energy and inertial loading terms from Eq. (4.7) become

$$\overline{J}_3 = - \int_{V_0} \left(T \frac{\partial q}{\partial X_1} - \varrho \frac{\partial^2 u_i}{\partial t^2} \frac{\partial u_i}{\partial X_1} q + \varrho \frac{\partial u_i}{\partial t} \frac{\partial^2 u_i}{\partial t \partial X_1} q \right) dV_0 \tag{4.13}$$

WARP domain integral processors evaluate only the first two terms of this integral. The third term (velocity) is vanishing small unless high speed crack propagation takes place.

When crack face tractions are present, an additional contribution to the J–integral is computed using

$$\overline{J}_4 = - \int_{A_3+A_4} t_i \frac{\partial u_i}{\partial X_1} q \ dA_0 \ . \tag{4.14}$$

where t_i denotes the face traction expressed in the front system and $A_3 + A_4$ denotes the upper and lower portion of the loaded faces (refer to Fig. 4.2).

Eqs. (4.12) through (4.14) are implemented to support finite–strains and finite–rotations as indicated under the assumption that $\overline{J}_2$ vanishes. Thus the present formulation applies most realistically to models in which displacements impose large (rigid) rotations on the domain but in which finite strains are confined to the usual blunting zone ahead of the crack tip. An example is a pin loaded, single–edge notch tension specimen, SE(T), containing a deep notch, i.e., $a/W > 0.5$. Under increased loading, the specimen may undergo relatively large rotations as the line of action of the axial load re–aligns with the center point of the remaining ligament. Finite strains are confined to the near tip region. The present formulation includes the effects of such large (rigid) rotations of the specimen on J–values.

and,

$$\frac{\partial u_i}{\partial X_1} = \sum_{I}^{n} \sum_{m}^{3} \frac{\partial N_I}{\partial \eta_m} \frac{\partial \eta_m}{\partial X_1} u_{iI} \tag{4.20}$$

where N is the number of element nodes. Similar procedures are followed for evaluation of the first two terms of Eq. (4.13); the third term in this equation is neglected.

4.3.3 Crack Face Traction Integral

The crack face traction integral, Eq. (4.14), is evaluated using the equivalent nodal loads corresponding to the applied crack face tractions. The crack face integral is thus evaluated numerically using the expression

$$\int_{A_3+A_4} t_i \frac{\partial u_i}{\partial X_1} \, q \, dA_0 = \sum_{k} \sum_{l} q_l \left\{ \partial u_i / \partial X_1 \right\}_l^T \left\{ P_i \right\}_l \tag{4.21}$$

where k is taken over elements with non–zero crack face tractions; l is taken over all element nodes on the loaded face; $\{P\}$ is the vector of equivalent nodal loads at element node l due to the surface traction. Displacement derivatives at the element nodes needed in Eq. (4.21) are obtained by extrapolating derivatives computed at Gauss point locations. Lagrangian polynomials are again adopted for the extrapolation. Not only is this technique more accurate than evaluating derivatives directly at the element nodes, the difficulty in computing derivatives at nodes on the crack front due to the singularity is avoided (extrapolated derivatives are not singular). Numerical tests demonstrate that the approximate expression given in Eq. (4.21) works very well.

The computational routines determine which element faces are loaded by examining the equivalent nodal loads for the complete element. If an element load vector indicates that more than one face is loaded, the lowest numbered element face is processed and a warning message is issued to the user. Because this procedure was adopted (thereby eliminating the need to respecify crack face loads during J computation), crack face loads and thermal loads should not be specified in the same loading condition — the computational routines will mistake the equivalent nodal loads due to the thermal loading for crack face loading.

If all nodes of an element have non–zero equivalent loads, a body force load is assumed to exist and no DI contributions are computed.

Users must include in the list of elements to process all elements with crack face loading if any node on the face has a non–zero q value.

4.3.4 Coincident Crack Front Nodes

The use of degenerated brick–type elements generally leads to meshes with multiple, coincident nodes along the crack front. To simplify specification of the q–function over the domain volume, the q–value for only *one* of the coincident nodes at such crack front positions is required. The remaining coincident nodes at corresponding crack front positions are located and assigned the same value for q. The procedure followed to locate coincident nodes is outlined below.

For each user specified node along the crack front, the numerical procedure constructs coordinates for a cubical prism centered at the node, then locates all other nodes of the model that lie within the prism. Such nodes are treated as coincident with the specified node

4.3 Numerical Procedures

This section describes the numerical procedures implemented to evaluate the Domain Integrals described previously. An understanding of these procedures is necessary for the correct use of the commands described subsequently.

4.3.1 Definition of the q–Function

Consistent with the isoparametric formulation, the q–function within an element has the form

$$q(\xi,\eta,\zeta) = \sum_{i=1} N_i(\xi,\eta,\zeta) q_i \tag{4.15}$$

where q_i are the specified values of the q–function at the element nodes. The user defines: (1) a list of nodes along the crack front included in the computations to evaluate A_q, (2) elements over which integrations are to be performed, (3) q_i at nodes over the volume, V, and (4) orientation of the crack front coordinate axes at the point s under consideration.

When collapsed elements are defined along the crack front producing coincident nodes, only one of the coincident nodes at each location is specified; the computational routines locate the remaining coincident nodes and assign them the same value of q. To define the orientation of the crack front axes relative to the global axes, users specify the components of a unit vector normal to the crack plane.

The specification of nodal q–values becomes exceedingly tedious for 3–D analyses. An "automatic" procedure is available as an option for generation of q–values. This procedure requires that the user specify: front nodes along the crack front, the number of domains required for checking path independence and components of the unit vector normal to the crack plane. The domain processors create domains of increasing distance from the crack tip using the mesh topology.

4.3.2 Volume Integrals

The volume integrals are numerically evaluated using the same Gaussian quadrature procedures adopted for element stiffness generation. The integral in Eq. (4.12) presents no difficulties as both W and the stresses are available at the Gauss point locations and the q–function derivative is readily computed from specified nodal values and Eq. (4.15). Gauss quadrature applied to Eq. (4.12) yields the expression for numerical computations as

$$\overline{J}_1 = -\sum_p \left[W \frac{\partial q}{\partial X_1} - P_{ji} \frac{\partial u_i}{\partial X_1} \frac{\partial q}{\partial X_j} \right]_p \det \left[\frac{\partial X_m}{\partial \eta_m} \right]_p w_p \tag{4.16}$$

where the summation extends over all Gauss quadrature points (p) and w_p denotes the Gauss weight values. The 1st PK stresses are computed from the unrotated Cauchy stresses using the two step transformation

$$\boldsymbol{\sigma} = \boldsymbol{R} \cdot \boldsymbol{t} \cdot \boldsymbol{R}^T \tag{4.17}$$

$$\boldsymbol{P} = |\boldsymbol{F}| \boldsymbol{\sigma} \cdot \boldsymbol{F}^{-T} \tag{4.18}$$

Cartesian derivatives of q and the displacements are obtained in the usual manner using the chain rule

$$\frac{\partial q}{\partial X_1} = \sum_I^n \sum_m^3 \frac{\partial N_I}{\partial \eta_m} \frac{\partial \eta_m}{\partial X_1} q_I \tag{4.19}$$

and are assigned the same q–value. Dimensions for the cubical prism are defined as follows: for 2 or more nodes specified along the crack front (3–D models), the prism extends $\pm\ R \times tol$ about the node, where R is the distance between the first two listed nodes on the crack front.

The value 0.001 is currently specified for *tol*. While this value has proven adequate for most crack front meshes, models with exceptionally large element lengths along the front may require a smaller value for *tol* (at present this requires a change in the source code).

4.3.5 Computation of A_q

The area under the q–function along the crack front, denoted A_q, is required to normalize J for arbitrary magnitudes of the specified q–function in Eq. (4.8), see also Fig.4.3. Thus, A_q may be interpreted as area of crack extension represented by a virtual crack extension q. The value of A_q is defined by

$$A_q = \int_{s=a}^{s=c} q(s)\ ds \tag{4.22}$$

which is numerically evaluated using Gauss quadrature as

$$A_q = \sum_p \sum_I N_I(s_p)\ q_I \left[\sqrt{dX_1^2 + dX_2^2}\ \right]_p w_p \tag{4.23}$$

where the functional form of q over the segment of crack front under consideration, $a \le s \le c$, is specified by the user to vary in a piecewise linear, parabolic or cubic manner. Lagrangian interpolating functions, $N_I(s)$, are used to construct the piecewise functions for q along the crack front. The length of crack front over $a \le s \le c$ is computed with the expression

$$L = \sum_p \left[\sqrt{dX_1^2 + dX_2^2}\ \right]_p w_p \tag{4.24}$$

and is displayed for checking purposes.

4.3.6 Output From Computations

The printed output displayed during Domain Integral computations is organized in a hierarchial manner at the load step for the user specified domains. By default, only the results for each complete domain are printed; an option to print contributions for each element is available. The values printed for each domain (or element in a domain) are labeled *DM1* through *DM5* and correspond to the terms in Eqs. (4.12) through (4.14) as follows

$$DM_1 = -\int_{V_{e(0)}} W \frac{\partial q}{\partial X_1}\ dV_0 \tag{4.25}$$

$$DM_2 = \int_{V_{e(0)}} P_{ji} \frac{\partial u_i}{\partial X_1} \frac{\partial q}{\partial X_j}\ dV_0 \tag{4.26}$$

$$DM_3 = -\int_{V_{e(0)}} T \frac{\partial q}{\partial X_1}\ dV_0 \tag{4.27}$$

$$DM_4 = \int_{V_{e(0)}} \varrho \frac{\partial^2 u_i}{\partial t^2} \frac{\partial u_i}{\partial X_1} q \; dV_0 \tag{4.28}$$

$$DM_5 = -\int_{A_3+A_4} t_i \frac{\partial u_i}{\partial X_1} q \; dA_0 \tag{4.29}$$

The sum of these integrals over all elements of the domain is displayed followed by A_q, the area under the q–function along the crack front. The J–integral value is printed as the sum of the integrals divided by A_q. The units of J are $F - L/L^2$.

The average, maximum, and minimum J values are summarized in tabular form. Separate sums are also printed for static and dynamic contributions.

4.4 Commands for Domain Integrals

4.4.1 Outline of Process

Once the analysis completes for the list of load steps appearing in the current *compute displacements* command, WARP command processors read the next data line. This can be an *output* command, another *compute* command or a *domain* command (as well as a number of other valid commands).

The *domain* command initiates the input sequence to specify information about a domain for computation of the J–integral. Following specification of a valid domain, the input command *compute domain integral* invokes the domain integral processors to perform the computations using analysis results for the most recent (current) load step analyzed.

To evaluate J over different domains using results for the current load step, simply repeat the *domain ... compute domain integral* sequence as often as desired. WARP stores only the definition of the most recently defined domain. When J is evaluated using the same domain definitions at many load steps, the **input from file* command proves convenient to eliminate repetition. The *domain* definitions and *compute domain integral* commands are defined in a separate input file and simply referenced with the **input from file* feature of WARP.

At completion of domain integral computations, other commands may be given to compute displacements for additional steps, request other output, alter solution parameters, etc.

4.4.2 Input Error Correction

The processor of domain integral commands recovers easily from most syntax errors. Messages indicating the error are displayed and a new input line read; simply re–enter the corrected form of the command. The new information overwrites previous values.

The input processor performs immediate checks for obvious errors in the specified data. More extensive consistency checking of the domain definition occurs during the actual numerical computations.

4.4.3 Components of a Domain Definition

Each domain for J computation consists of the following information:

1. The alphanumeric name (id) of the domain as specified in a *domain* command.
2. Components of a unit vector normal to the crack plane.
3. A symmetry flag, if applicable. J–values are then doubled prior to printing.
4. A list of nodes defining a portion of the crack front under consideration and the order of geometry approximation along the crack front, e.g., linear, quadratic.
5. q–values at nodes along the portion of the crack front under consideration and over the desired volume of domain integration. Two methods to specify nodal values of q are available: user–defined and automatic.
    a. User defined—users specify actual nodal values for q and the list of elements over which the domain integration is desired. A single J–value is printed.
    b. Automatic—users specify the number of concentric *rings* of elements enclosing the tip over which J is evaluated at the crack front position. The q–values and lists of elements are generated automatically by WARP domain processors. A J–value is printed for each *ring* of elements requested.

6. Printing options. By default the total DM_i values are printed for the domain (each ring if automatic); individual element contributions are not printed. Users may request printing of individual element values as well.
7. Order of Gauss quadrature for element volume integrals. The default integration order is that used for element stiffness computation. A one–point rule is an optional order.
8. Crack face loadings option. By default, contributions to J from elements with detectable crack face loading are included. An option is available to neglect crack face loading contributions. This option is needed for crack growth analyses in which the crack closing forces are slowly relaxed to zero behind the extending front. These forces are interpreted by the domain processors as equivalent loads for crack face loading.
9. Debug output options. Two levels of debugging information may be requested.
10. Verification of domain input. A "dump" option prints the definition of domain parameters from internal storage.

4.4.4 Initiating a Domain Definition

The command to initiate a new domain has the form

domain < name: label >

where the domain name appears as a descriptor in printed output.

4.4.5 Crack Plane Orientation

The orientation of the local crack front system, X_1–X_2–X_3, shown in Fig. 4.3 must be specified. The user defines components of a unit vector normal to the crack plane (X_2–X_3) aligned in the positive direction of X_2. WARP then determines the direction of X_3 using the list of crack front nodes (the positive direction of X_3 is in the direction from the first node to the second node in the list). The direction X_1 is found from the cross product $X_2 \otimes X_3$.

The command to define crack plane normals has the form

normal (plane) [{ nx / ny / nz } < direction cosine: number >]

where *nx*, for example, defines the projection of the crack plane (unit) normal onto the global X axis. If the global Z axis is normal to the crack plane, for example, use the command

```
normal plane nz 1.0
```

The direction cosines provided in the command must define a vector of unit length ($nx^2 + ny^2 + nz^2 \equiv 1$).

When the J–values are negative but have the correct absolute value, reverse the sense of the crack plane normal vector.

This combined procedure in which the user specifies the X_2 direction and the domain processors use the front node list to compute directions for X_1–X_3 naturally fits the pointwise computation of J along a curved crack front. Similarly, a thickness–average J–value for a slightly curved or straight crack front in a through crack configuration is easily obtained with the automatic method of q specification. Note, however, that the X_3 direction for the domain is defined by the first two nodes given in the front node list.

4.4.6 Symmetric Option

The *symmetric* option is provided as a convenience since many finite element models are defined for symmetric geometries, loading and constraints. When this keyword is specified,

all J–values are double prior to printing. An output message signals when J–values are doubled as well.

The command to request doubling of J–values for symmetry has the form

symmetric

4.4.7 Crack Front Nodes

The command to define nodes on the crack front for the domain has the form

$$\underline{\text{front}}\ (\text{nodes})\ <\text{integerlist}>\ \left\{\begin{array}{c}\underline{\text{linear}}\\ \underline{\text{quadr}}\text{atic}\end{array}\right\}\ (\underline{\text{verify}})$$

where the ordering of front nodes in the list must follow increasing X_3. The specified interpolation order defines the variation of q *and* the shape along the crack front for computation of A_q. The default order is *linear*. For a *quadratic* order, the number of front nodes listed must follow be odd (3, 5, 7, ...). The interpolation order indicated with this command is *not* checked for compatibility with the interpolation order of the adjacent crack–tip elements. For example, only the *linear* option should be used for a mesh of *l3disop* elements.

When the crack front is modeled with collapsed elements, there are multiple coincident nodes at locations along the front. Only one of the coincident nodes should be specified at these locations in this command. The remaining coincident nodes are located automatically and included in subsequent processing. A list of the other nodes coincident with each front node specified in this command is printed if the keyword *verify* appears as the last item of the command.

To illustrate the use of this command, consider the curved crack front sketched in Fig. 4.4. Crack front elements are linear isoparametrics (*l3disop*). Let node 10 lie on a symmetry plane; node 22 lies on the outside (free) surface. To compute J at node 10 on the front, the crack front segment in the domain includes nodes 10 and 14. The input command is

```
front nodes 10 14 linear verify
```

To compute J at node 14, the crack front segment in the domain includes nodes 10, 14 and 18. The input command is

```
front nodes 10 14 18 linear verify
```

where the *linear* option is used to indicate that elements along the front have a linear displacement interpolation and that q should vary linearly (piecewise) along the front between nodes 10, 14 and 18 (q will be zero at 10 and 18 and > 0 at 14).

4.4.8 Specification of q–Values

Two methods for defining the q–values are available: automatic and fully user–specified. Each method is described in a section below. The automatic method will suffice for must applications.

Automatic q Definition

The automatic method supports J computation for the following situations:

1. Pointwise evaluation at a crack front location on a symmetry plane or on a free surface (there are elements only to one side of the crack front location).
2. Pointwise evaluation at an interior crack front location corresponding to a corner node (elements exists on both sides of the crack front location).

3. Average J–value for the complete crack front (straight or slightly curved fronts).

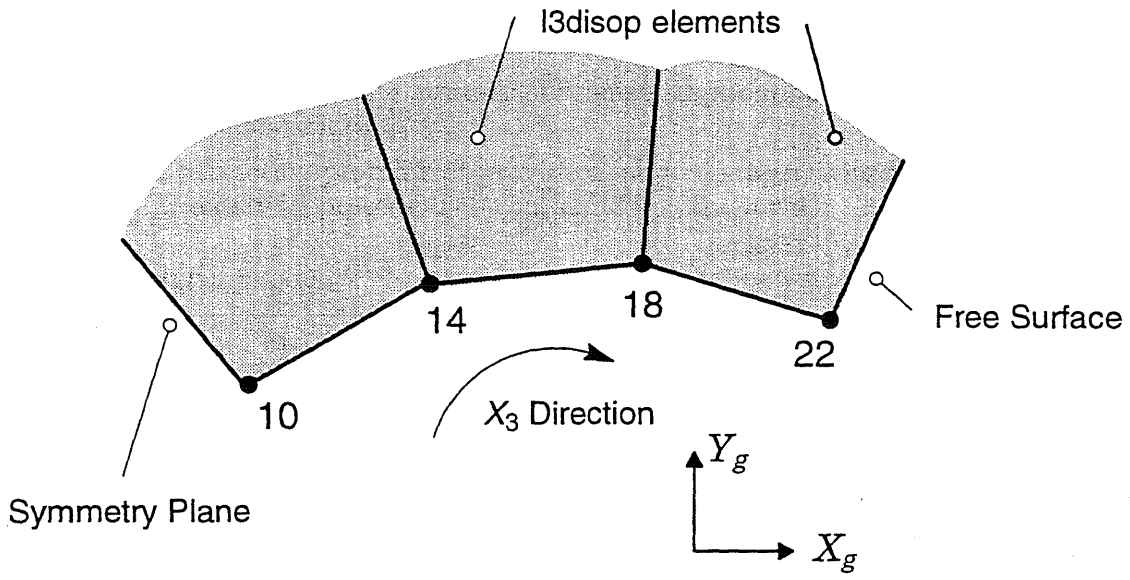


FIG. 4.4—*Example crack front to illustrate front nodes specification.*

At a crack front location, the automatic method constructs one or more domains for investigation of domain independence of computed J–values. The concept of a *ring* of elements is adopted to describe the domains generated at a crack front location (see Fig. 4.5). *Ring* 1 contains only elements incident on the nodes defined in the list of front nodes. *Ring* 2 contains the front elements plus the next ring of elements enclosing the tip, etc. J–values for *ring* 1 usually have the greatest error and should be avoided if possible. J–values for *rings* 2, 3, ... should be reasonably similar. For a nonlinear elastic (deformation plasticity) model, the values in *rings* 2, 3, ... often show less than 1% variation.

The command to specify automatic generation of q–values has the form

q(–values) automatic (rings) < integerlist >

and must be followed by the command

function (type) { a | b | c | d }

where a–d denotes the variation (function type) of q along the crack front. The four function types are illustrated in Fig. 4.6. Types a and c are used to evaluate J at end points of a crack front, e.g., at nodes 10 and 22 in Fig. 4.4. Type b is used to evaluate J at an interior node, e.g., nodes 14 and 18 in Fig. 4.4. Function type d is used to compute a "through–thickness" average J for a straight or slightly curved crack front. When higher–order elements are used along the crack front, the automatic method supports J computation only at the element corner nodes.

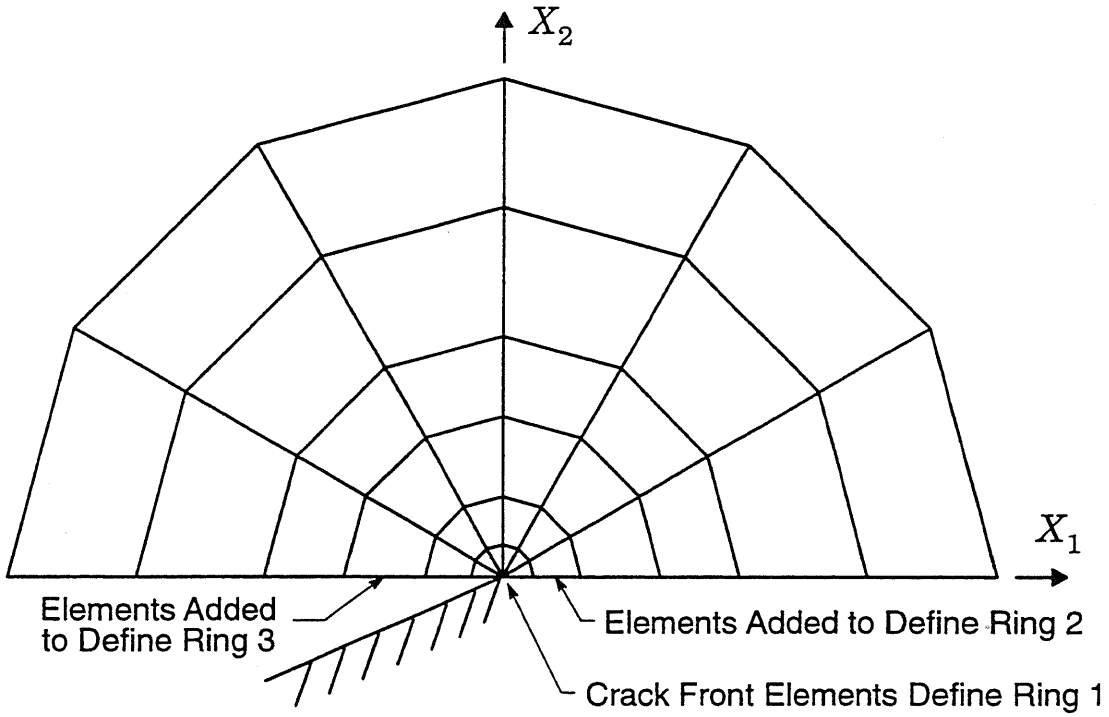


FIG. 4.5—*Concept of rings used in automatic domain generation.*

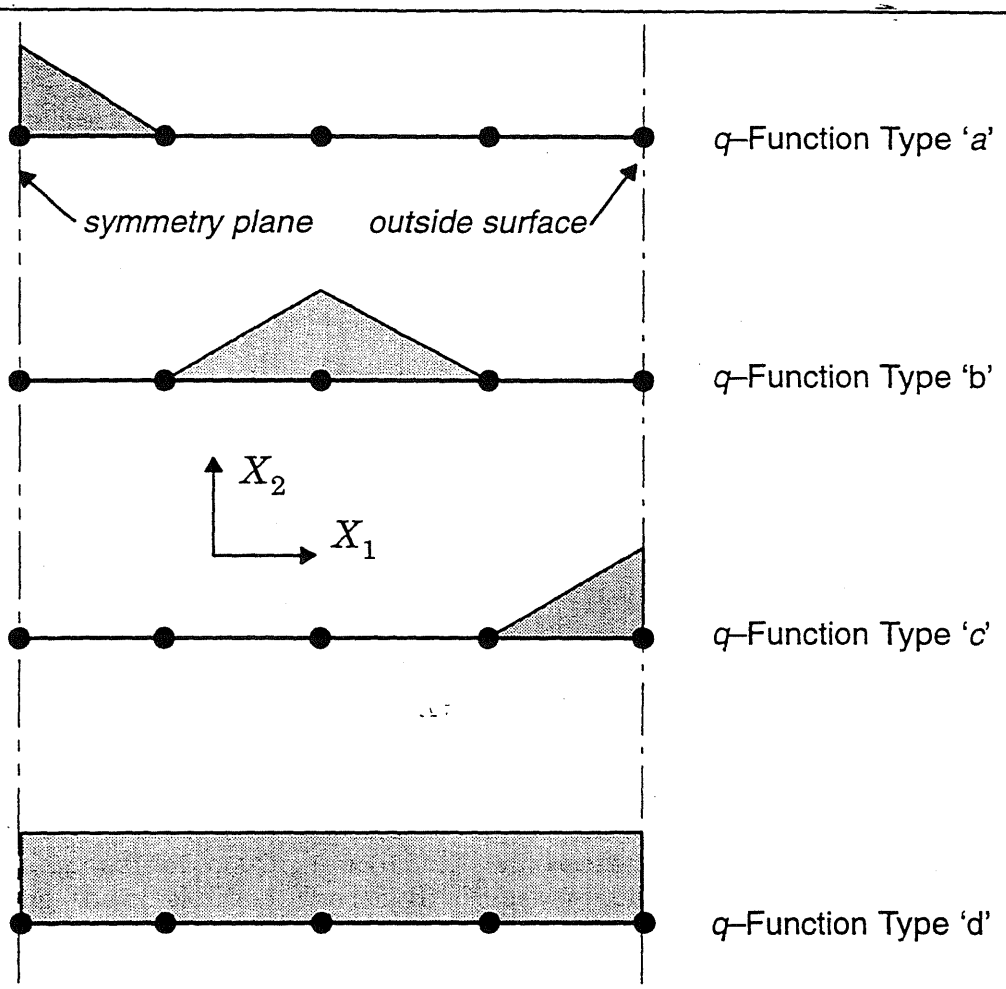
For function types *a*–*c*, the automatic algorithms construct nodal values for q which vary linearly in the X_3 direction. For function type *d*, q maintains a constant value in the X_3 direction along the front. Nodal values for q are generated automatically such that the following conditions hold:

Ring 1: q derivatives: $\partial q/\partial X_j$ = constant.

Ring i: for elements appearing in rings 1, 2, 3, ... i–1, the q derivatives: $\partial q/\partial X_1 = 0$, $\partial q/\partial X_2 = 0$ and $\partial q/\partial X_3 \neq 0$. For elements added to ring i–1 to define ring i, the q derivatives are $\partial q/\partial X_j$ = constant.

As a consequence of these q–derivative properties, element rings 1, 2, 3, ... i–1 have $DM_1 = DM_3 = 0$. These elements make a small contribution to DM_2 since the variation of u_3 with X_1 is non–singular. The acceleration forces which define DM_4 make significant contributions in near front rings since q, rather than q–derivatives, appear in the integral. For function type *d*, the terms DM_1, DM_2 and $DM_3 \equiv 0$ for elements in rings 1, 2, 3, ... i–1.

The automatic generation process creates one additional domain for each ring requested by the user. The J–value for each of these domains is printed and included in the average, minimum, maximum statistics. If the element printing option is also *on*, the contribution for each element to each domain is printed. The list of rings specified in the automatic domain method can be of the form *rings* 2 4 6 10 15 .... While the domains for all rings (through the maximum ring listed) are created internally, J is computed and printed only for the ring numbers in the list. In this way, for example, the user may request computation and output for a few rings far from the crack front, e.g., rings 10–15. The domain processors include the contributions of all elements in rings nearer the tip as required for each term of J, e.g., crack face loading and inertia terms which involve q and not q–derivatives.



Note: linear elements on front used for illustration

FIG. 4.6—*Types of q–functions available for automatic domain generation.*

Consider the following example of automatic domain generation (refer to Fig. 4.4). Let the crack plane be normal to the global Z–axis.

```
define domain symm_corner
   normal plane nz 1.0
   front nodes 10 14 linear verify
   q-values automatic rings 2-4
   function type a
   .
   .
   .
compute domain integral
```

Function type a is specified since node 10 is on the symmetry plane. Automatic domains are constructed for rings 1–4 but J is computed and printed only for rings 2–4 to omit ring 1 which usually has the most error.

To compute J at the front location of node 14 and 18, the following automatic domains and compute commands are used

```
define domain front_14
  normal plane nz 1.0
  front nodes 10 14 18 linear verify
  q-values automatic rings 2-4
  function type b
  .
  .
  .
compute domain integral

define domain front_18
  normal plane nz 1.0
  front nodes 14 18 22 linear verify
  q-values automatic rings 2-4
  function type b
  .
  .
  .
compute domain integral
```

At the intersection of the crack front with the outside free surface (at node 22), the following domain is specified

```
define domain outside_22
  normal plane nz 1.0
  front nodes 18 22 linear verify
  q-values automatic rings 2-4
  function type c
  .
  .
  .
compute domain integral
```

For a crack with the front curvature indicated in Fig. 4.4, a thickness–average J using function type d would seem to be of questionable value.

User Specified q–Values and Elements

All nodal values of q are zero by default. Non–zero nodal values of q over the domain are defined with the command

q(–values) < node list > < q: real >

where the nodal q–values must be of class <real> to be distinguished from the list of node numbers. This command may be repeated as needed to define all nodal values for q in the domain. q–values must be specified for all element corner nodes in the domain and for all nodes along the crack front segment under consideration. Computational routines for higher–order elements employ a linear variation of q between corner nodes.

The list of all elements to be included in the computations is defined with the command

elements <integerlist>

Elements that should be included are: (1) those over which q is not constant, (2) those with loaded crack faces and non–zero q–values, (3) those with inertia forces and non–zero q–values, (3) those with thermal loading and non–zero q–values.

The following example illustrates the definition of a domain to compute J at node 14 for the crack front illustrated in Fig. 4.4.

```
define domain outside
   normal plane nz 1.0
   front nodes 10 14 18 linear verify
   q-values 10 18 0.0
   q-values 18 1.0
   elements 10-14
   .
   .
compute domain integral
```

In this example, only the crack front elements incident on node 18 make contributions to J (this is *not* recommended!). q–values at nodes 10 and 18 default to 0.0 and can be omitted from the above commands (they are included for readability). The normal plane and front node specifications are identical to automatic domains. Only elements appearing in the specified list are evaluated during J computations.

4.4.9 Printing Options

By default, the total contributions (DM_1, DM_2 ...) and the sum of DM_1, DM_2 ... are printed for the domain (each ring of an automatic domain). The domain values are followed by the minimum J, maximum J and average J for the domains. When inertia effects are present, $DM_3, DM_4 \neq 0$, separate totals for static and dynamic terms are provided to make obvious the relative importance of these terms in the total J–value.

To explicitly request this level of output, use the command

print totals

More detailed output listing the contribution from each element is requested with the command

print element (values)

This option also provides the information of the *print totals* default.

4.4.10 Integration Order

The volume integrals contributing to J are evaluated using the same order of Gauss integration as is used for stiffness computation. For *l3disop* elements, J–values with a greater level of domain independence are often obtained by using one–point Gauss integration. This option is requested with the command

use 1 (point rule)

4.4.11 Face Loading

By default, crack face loadings if present are included in the domain integral computations. The crack face loadings may be omitted with the command

ignore (crack) (face) loading

As noted previously, this option should be invoked when crack growth is modeled by releasing the closing forces to zero over a number of load steps. Such forces are mistakenly interpreted as crack face tractions by the domain integral processors.

4.4.12 Domain Verification

The definition of a domain as stored in internal tables may be printed with the command *dump*. This command may be given at any time during the domain definition and as many times as desired.

4.4.13 Debugging Domain Computations

The actual domain computations may be traced with printed output detailing each step of the computations. This may prove convenient to more closely examine *J*–values. To trace the primary domain integral processor (but not element integration routines), use the command

debug driver

To debug element integration routines, use the command

debug elements

Both commands may be specified in the domain definition.

4.4.14 A Complete Example

The following is a complete example illustrating all commands for domain definition using automatic procedures.

```
define domain symm_corner
   symmetry
   normal plane nz 1.0
   front nodes 10 14 linear verify
   q-values automatic rings 2-10
   function type a
   print totals
   print element values
   use 1 point rule
   ignore crack face loading
   debug driver
   debug elements
   dump
compute domain integral
```

Chapter 5

Crack Growth Procedures

5.1 Introduction

Two types of procedures are provided to include the effects of discrete crack extension in WARP3D. In the first type of crack growth, termed *element_extinction*, complete elements in the model are deleted when a critical condition (damage) is reached under increased loading. The element stiffness is set to zero and the forces exerted by the element on adjacent nodes are released to zero gradually over a user–specified number of load steps. In this procedure the element is not topologically deleted from the model but it no longer contributes any resistance to loading. In other codes, this technique of element extinction is often referred to as an element "death" option.

In the second type of crack growth, termed *node_release*, an increment of crack extension along a symmetry plane of one element in length is achieved by the traditional node release procedure. The nodal constraint holding the crack closed is replaced by the corresponding reaction which is then released to zero gradually over a user–specified number of steps. The element is not deleted from the model and most often undergoes inelastic unloading and then re–yielding as the crack tip continues to extend further ahead. The criterion for crack extension is a critical crack–tip opening angle (CTOA) based on the opening mode displacement of the node nearest the current tip at each location along the crack front.

This chapter describes commands to invoke each of the two crack growth procedures and additional details of their implementation in WARP3D.

5.2 Element Extinction

In this procedure, elements are effectively deleted from the solution when a user–specified level of damage develops under increased loading. The presently available measure of damage is the average void fraction, f, in elements which have the Gurson–Tvergaard dilatant plasticity material model (type *gurson*). During subsequent load steps, the element stiffness is taken as zero and the nodal forces exerted by the element on adjacent nodes are relaxed to zero in a user–specified number of load steps.

The user actions required to invoke the element extinction option during an analysis are:

- specify the logical property *killable* in the definition of a material that invokes the *gurson* material model (in the same analysis, there can be other materials using the *gurson* model that do not have the *killable* property).
- following the procedures for other nonlinear analyses, define the finite element model, loading, constraints and nonlinear solution parameters.
- use the commands described subsequently in this section to define parameters controlling the crack growth procedures (critical porosity, number of release steps, printing options, etc.). These parameters are specified in a manner analogous to specification of the nonlinear solution parameters; some crack growth parameters may be altered during the analysis as noted in the command descriptions that follow.
- use various combinations of *compute* and *output* commands to control the nonlinear solution over load steps. The crack growth procedures are automatically invoked by solution management routines in WARP3D.
- the analysis restart features of WARP3D fully support crack growth modeling. Restart files contain the values of growth parameters and the solution state required to continue an analysis with crack growth.

5.2.1 Input Commands

The sequence of commands to initiate the definition of crack growth parameters is

crack (growth) (parameters)

type (of) (crack) (growth) { none | element_extinction }

where *none* turns off subsequent element extinction during the analysis. Once elements have been made extinct in an analysis and the option *none* is given, further crack growth cannot be re–invoked. To temporarily suppress further growth, the simplest (and recommended) procedure is to increase the critical porosity value.

The porosity value at which element extinction occurs is given be

critical (porosity) < porosity limit: value >

The average porosity at the Gauss points for each *killable* element with a *gurson* material model is compared with the specified critical value at the beginning of each load step. When the average value first exceeds the porosity limit, the element extinction process begins for that element. The default value for critical porosity is 0.20.

When the element porosity first exceeds the specified limit, the element "internal" forces are imposed on adjacent nodes in the model as though they are nodal forces. The value of these forces decreases linearly to zero over a number of sequential load steps (the ele-

ment stiffness is immediately set to zero and remains zero for all subsequent load steps) The command to specify the number of "release" steps has the form

release (steps) < integer >

The default value is 5 steps. The number of release steps *cannot* be altered once any elements have been made extinct.

The element extinction procedures provide a convenient *printing* option to simplify interpretation of the growth process. The command has the form

$$\text{print (status)} \left\{ \begin{matrix} \text{off} \\ \text{on} \end{matrix} \right\} \text{(order < element list: integerlist >)}$$

where the keyword *on* or *off* is required. An optional list of *killable* elements may be specified for processing. If no list is given, all elements having a *gurson* material model with the *killable* property are included in the list (in ascending numerical order). When the optional list is given, information is printed for elements in the order specified in the list. At the beginning of each load step when this printing option is *on*, the following data is tabulated for each element in the list: initial porosity (f_0), current (average) porosity (f), average plastic strain in the matrix ($\bar{\varepsilon}^p$) and average (Mises) equivalent stress in the matrix ($\bar{\sigma}$). To prevent excessive amounts of output, information is printed only for those elements with $f > f_0$.

By default, every element eligible to be made extinct is processed without regard to any specific topological order. In some cases, it may be desirable to force extinction of elements in prescribed topological order. To specify this feature, use the command

$$\text{sequential (extinction)} \left\{ \begin{matrix} \text{off} \\ \text{on} \end{matrix} \right\} \text{(order < element list: integerlist >)}$$

where the use of this feature is invoked/suppressed with the required *on / off* keyword. The optional list provides the topological sequencing of elements to be made extinct. For example, if the second element in the list reaches the critical porosity prior to the first element in the list, then both the first and second elements in the list are made extinct simultaneously. When the list is omitted, the topological ordering is taken to be ascending numerical sequence by element number for all elements in the model with the *killable* material property.

A complete example of crack growth commands is given below.

```
crack growth parameters
  type of growth element_extinction
  critical porosity 0.18
  release steps 10
  print status on order 20-80 by 2
  sequential extinction on order 20-80 by 2
```

5.2.2 Extinction Algorithm

At the beginning of each load step n (n>1), the average porosity is computed for each *killable* element in the model. When the element conditions are such to require extinction (achieved the critical porosity or the sequential ordering feature dictates extinction even when $f < f_{crit}$), the following actions are taken:

- Young's modulus and Poisson's ratio for the element are set to zero. The element history data is deleted (porosity, plastic strain, stresses, etc.).

- Element contributions to the global internal force vector are applied as nodal forces. All subsequent contributions of the element to global equilibrium are zero. The element internal force vector when extinction begins is gradually decreased in a linear fashion over the specified number of release steps. Because the element forces are converted into nodal forces and treated thereafter as ordinary (user–specified) forces, the adaptive step algorithm is unaffected by crack growth and often proves essential for obtaining converged solutions following a growth increment.
- All subsequent computations for the element stiffness (linear or tangent) resolve to a zero matrix.
- When all elements connected to a node are made extinct, the node has no stiffness and introduces a singularity into subsequent equation solving efforts. To prevent this, the element extinction procedures track the number of elements attached to model nodes at any time and automatically supply new constraints on "free" nodes to eliminate the singularity.
- The blocking requirements dictate that all elements in a block must be *killable*. When a new element is made extinct in a block, checks are made to determine if all elements in the block have been made extinct; computations on such blocks may be completely skipped in subsequent load step solutions.

5.3 Node Release

This feature not yet implemented.

Appendix A

Patran Results File Formats

Figures in this Appendix provide skeletal Fortran programs to read Patran nodal results files. They provide a starting point for development of more advanced programs.

Fig. A.1	read a binary file of nodal strain/stress results
Fig. A.2	read a binary file of nodal displacements, velocities, accelerations, internal forces
Fig. A.3	read an Ascii file of nodal strain/stress results
Fig. A.4	read an Ascii file of nodal displacements, velocities, accelerations, internal forces

```
c ***************************************************************************
c *                                                                         *
c *   read stress/strain binary patran file                                 *
c *                                                                         *
c ***************************************************************************
c
c
      implicit integer (a-z)
      parameter( maxnod=20000, maxcols=50 )
      double precision nodval(maxcols,maxnod)
      real rtemp, pvals(maxcols)
      dimension title(80)
      character * 80 binnam
c
      termin = 5
      termot = 6
      binfil = 10
c
      write(termot,*) ' '
      write(termot,*) ' '
      write(termot,*) '>> binary strain/stress processing program'
      write(termot,*) ' '
      write(termot,9400) ' > name of results file? '
      read(termin,9500) binnam
      open(unit=binfil,file=binnam,status='old',recl=3000,form='unformatted')
      write(termot,*) ' > file opened ok'
c
c             read the binary results file of nodal strains/stresses.
c             patran results are single precision. read as single and
c             store as double.
c
      read(binfil) title,nnode,ii,rtemp,ii,nvals
      read(binfil) title
      read(binfil) title
      write(termot,*) '>> number of nodes: ',nnode
      do node = 1, nnode
         if ( mod(node,200) .eq. 0 ) then
            write(termot,*) '   > processing node: ',node
         end if
         read(binfil) ii, (pvals(jj),jj=1,nvals)
         nodval(1:nvals,node) = pvals(1:nvals)
      end do
      close(unit=binfil)
c
c            call a routine to do something with the nodal values
c            of stress/strain.
c
      call process( nodval, nnode, maxcols, termot )
c
      write(termot,*) '>> processing completed'
      write(termot,*) '>> normal termination'
c
      call exit
c
 9400 format(a,$)
 9500 format(a80)
      end
c
      subroutine process ( values, nnode, nrow, termot )
      implicit integer (a-z)
      double precision values(nrow,nnode)
      return
      end
```

FIG. A.1—*Fortran program to read Patran binary file of nodal strain or stress results.*

```
c ***************************************************************************
c *                                                                         *
c *   read displ, vel, accel, inter. forces binary patran file              *
c *                                                                         *
c ***************************************************************************
c
c
      implicit integer (a-z)
      parameter( maxnod=20000 )
      double precision x(maxnod), y(maxnod), z(maxnod)
      real xval, yval, zval
      dimension title(80)
      character * 80 binnam
c
      termin = 5
      termot = 6
      binfil = 10
c
      write(termot,*) ' '
      write(termot,*) ' '
      write(termot,*) '>> binary node value pocessing program'
      write(termot,*) ' '
      write(termot,9400) ' > name of results file? '
      read(termin,9500) binnam
      open(unit=binfil,file=binnam,status='old',recl=3000,form='unformatted')
      write(termot,*) ' > file opened ok'
c
c             read the binary results file of nodal values.
c             read x, y, z components. patran results are single
c             precision. read as single and store as double.
c
      read(binfil) title,nnode,ii,rtemp,ii,nvals
      read(binfil) title
      read(binfil) title
      write(termot,*) '>> number of nodes: ',nnode
      do node = 1, nnode
         read(binfil) ii, xval, yval, zval
         x(node) = xval
         y(node) = yval
         z(node) = zval
      end do
      close(unit=binfil)
c
c            call a routine to do something with the nodal values
c
      call process( x, y, z, nnode, termot )
c
      write(termot,*) '>> processing completed'
      write(termot,*) '>> normal termination'
c
      call exit
c
 9400 format(a,$)
 9500 format(a80)
      end
c
      subroutine process ( x, y, z, nnode, termot )
      implicit integer (a-z)
      double precision x(*), y(*), z(*)
      return
      end
```

FIG. A.2—*Fortran program to read Patran binary file of nodal displacements, velocities, accelerations or internal forces.*

```
c ***************************************************************************
c *                                                                         *
c *   read ascii stress/strain patran file                                  *
c *                                                                         *
c ***************************************************************************
c
c
      implicit integer (a-z)
      parameter( maxnod=20000, maxcols=50 )
      double precision nodval(maxcols,maxnod), val3
      character * 80 asciinam, line
c
c
      termin = 5
      termot = 6
      asciifil = 10
c
      write(termot,*) ' '
      write(termot,*) ' '
      write(termot,*) '>> ascii strain/stress processing program'
      write(termot,*) ' '
      write(termot,9400) ' > name of results file? '
      read(termin,9500) asciinam
      open(unit=asciifil,file=asciinam,status='old')
      write(termot,*) ' > file opened ok'
c
c             skip past the header lines of neutral file.
c             get number of nodes and number of result values
c             for each node.
c
      read(asciifil,9500) line
      read(asciifil,9600) nnode, ival2, val3, ival4, nvals
      read(asciifil,8900) line
      read(asciifil,8900) line
c
c             read values for each node into a double array.
c
      write(termot,*) ' > reading nodal results file..'
      do node = 1, nnode
        read(asciifil,9000) ii, (nodval(jj,node),jj=1,nvals)
      end do
      close(unit=asciifil)
      write(termot,*) '>> nodal results file read'
c
c            call a routine to do something with the nodal values
c            of stress/strain.
c
      call process( nodval, nnode, maxcols, termot )
c
      write(termot,*) '>> processing completed'
      write(termot,*) '>> normal termination'
c
      call exit
c
 8900 format(a1)
 9000 format(i8,(5e13.7))
 9400 format(a,$)
 9500 format(a80)
 9600 format(2i5,e15.6,2i6)
      end
c
      subroutine process ( values, nnode, nrow, termot )
      implicit integer (a-z)
      double precision values(nrow,nnode)
      return
      end
```

FIG. A.3—*Fortran program to read Patran ASCII file of nodal strain or stress results.*

```
c  ***************************************************************************
c  *                                                                         *
c  *   read displ, vel, accel, inter. forces ascii patran file               *
c  *                                                                         *
c  ***************************************************************************
c
c
      implicit integer (a-z)
      parameter( maxnod=20000 )
      double precision x(maxnod), y(maxnod), z(maxnod)
      character * 80 asciinam, line
c
c
      termin = 5
      termot = 6
      asciifil = 10
c
      write(termot,*) ' '
      write(termot,*) ' '
      write(termot,*) '>> ascii node value pocessing program'
      write(termot,*) ' '
      write(termot,9400) ' > name of results file? '
      read(termin,9500) asciinam
      open(unit=asciifil,file=asciinam,status='old')
      write(termot,*) ' > file opened ok'
c
c             skip past the header lines of neutral file.
c             get number of nodes and number of result values
c             for each node.
c
      read(asciifil,9500) line
      read(asciifil,9600) nnode, ival2, val3, ival4, nvals
      read(asciifil,8900) line
      read(asciifil,8900) line
c
c            read values for each node into a double array.
c
      write(termot,*) ' > reading nodal results file..'
      do node = 1, nnode
        read(asciifil,9000) ii, x(ii), y(ii), z(ii)
      end do
      close(unit=asciifil)
      write(termot,*) '>> nodal results file read'
c
c            call a routine to do something with the nodal values
c
      call process( x, y, z, nnode, termot )
c
      write(termot,*) '>> processing completed'
      write(termot,*) '>> normal termination'
c
      call exit
c
 8900 format(a1)
 9000 format(i8,(5e13.7))
 9400 format(a,$)
 9500 format(a80)
 9600 format(2i5,e15.6,2i6)
      end
c
c
      subroutine process ( x, y, z, nnode, termot )
      implicit integer (a-z)
      double precision x(*), y(*), z(*)
      return
      end
```

FIG. A.4—*Fortran program to read Patran ASCII file of nodal displacements, velocities, accelerations, or internal forces.*

Appendix B

References

[1] Amestoy, H.D., Bui,, and Labbens, "On the Definition of Local Path Independent Integrals in *3–D* Crack Problems," *Mechanics Research Communications,* Vol. 8, 1981, pp. 231–236.†

[2] Aravas, N., "On the Numerical Integration of a Class of Pressure–Dependent Plasticity Models," *International Journal for Numerical Methods in Engineering,* Vol. 24, 1987, pp. 1395–1416.†

[3] Asaro, R. J., "Crystal Plasticity," *Journal of Applied Mechanics,* Vol. 50, 1984, pp. 1–12.†

[4] Atluri, S. N. "On Constitutive Relations at Finite Strain: Hypo–Elasticity and Elasto–Plasticity with Isotropic and Kinematic Hardening" *Computer Methods in Applied Mechanics and Engineering,* Vol. 43, 1984, pp. 137–171.†

[5] Bakker, A., "The Three–Dimensional *J*–Integral: An Investigation into Its Use for Post–Yield Fracture Assessment," WTHD No. 167, Laboratory for Thermal Power Engineering, Delft University of Technology, Mekelweg 2, 2628 CD, Delft., 1984.§

[6] Bathe, K. J. *Finite Element Procedures in Engineering Analysis.* Prentice–Hall, Inc. Englewood–Cliffs, N.J., 1982.†

[7] Biffle J.H., "Indirect Solution of Static Problems Using Concurrent Vector Processing Computers," *Parallel Computations and their Impact on Mechanics*, ed. by A.K. Noor, American Society of Mechanical Engineers, New York, 1987, pp. 317–330.†

[8] Biffle, J. H., and M. L. Blanford, "JAC2D– A Two–Dimensional Finite Element Computer Program for the Nonlinear Quasi–Static Response of Solids with the Conjugate Gradient Method," *SAND93–1891, Sandia National Laboratories,* Albuquerque, NM., 1994.§

[9] Biffle, J. H., and M. L. Blanford, "JAC3D– A Three–Dimensional Finite Element Computer Program for the Nonlinear Quasi–Static Response of Solids with the Conjugate Gradient Method," *SAND87–1305, Sandia National Laboratories,* Albuquerque, NM., 1993.§

[10] Budiansky, B., and Rice, J., "Conservation Laws and Energy Release Rates," *Journal of Applied Mechanics*, Vol. 40, 1973, pp. 201–203.†

[11] Carey G.F., and Jiang B., "Element–By–Element Linear and Nonlinear Solution Schemes," *Communications in Applied Numerical Methods*, Vol. 2, No. 2, March–April 1986, pp. 145–153.†

[12] Carpenter, W.C., Read, D.T., and Dodds, R.H., "Comparison of Several Path Independent Integrals Including Plasticity Effects," *International Journal of Fracture,* Vol. 31, 1986, pp. 303–323.†

[13] Cherepanov, G.P., "The Propagation of Cracks in a Continuous Medium," *Journal of Applied Mathematics and Mechanics,* Vol. 31, 1967, pp 503–512.†

[14] Chu, C. C. and Needleman, A., "Void Nucleation Effects in Biaxially Stretched Sheets," *Journal of Engineering Materials and Technology,* Vol. 102, 1980, pp. 249–256.†

[15] Concus P., Golub G.H., and O'Leary D.P., "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equation," *Sparse Matrix Computations,* ed. J.R. Bunch and D.J.Rose, Academic Press, New York, 1965, pp. 307–322.†

[16] Cook R. D., Malkus, D. S., and Plesha, M. E., "Concepts and Applications of Finite Element Analysis," Third Ed., John Wiley & Sons, Inc., New York NY, 1989.†

[17] Crisfield, M.A., *Nonlinear Finite Element Analysis of Solids and Structures – Volume 1:Essentials*, John Wiley & Sons Ltd., 1991.†

[18] Cuitino, A. and Ortiz, M., "A Material–Independent Method for Extending Stress Update Algorithms from Small–Strain Plasticity to Finite Plasticity with Multiplicative Kinematics," *Engineering Computations,* Vol. 9, 1992, pp. 437–451.†

[19] de Lorenzi, H.G., "On the Energy Release Rate and the *J*–integral for *3–D," International Journal of Fracture,* Vol. 19, 1982, pp. 183–193.†

[20] Dienes, J. K., "On the Analysis of Rotation and Stress Rate in Deforming Bodies," *Acta Mechanica*, Vol. 32, 1979, pp. 217–232.†

[21] Dodds, R., "Numerical Techniques for Plasticity Computations in Finite Element Analysis," *Computers and Structures,* Vol. 26, No. 5, 1987, pp. 767–779.†

[22] Eshelby, J.D., "Energy Relations and the Energy Momentum Tensor in Continuum Mechanics," in *Inelastic Behavior of Solids*, M.F. Kanninen, et al. (eds), Mc Graw–Hill, NY, 1970.†

[23] Flanagan, D. P. and Taylor, L. M., "An Accurate Numerical Algorithm for Stress Integration with Finite Rotations," *Computer Methods in Applied Mechanics and Engineering,* Vol. 62, 1987, pp. 305–320.†

[24] Flanagan D.P., and Taylor L.M., "Structuring Data for Concurrent Vectorized Processing in a Transient Dynamics Finite Element Program," *Parallel Computations and their Impact on Mechanics*, ed. by A.K. Noor, American Society of Mechanical Engineers, New York, 1987, pp. 291–299.†

[25] Golub G.H., and Van Loan C.F., "Matrix Computations," The Johns Hopkins University Press, Baltimore Maryland, 1983.§

[26] Goudreau, G. L. and Hallquist, J. O., "Recent Developments in Large–Scale Lagrangian Hydrocode Technology," *Computer Methods in Applied Mechanics and Engineering,* Vol. 33, 1982, pp. 725–757.†

[27] Green, A. E. and Naghdi, P. M., "A General Theory of an Elastic–plastic Continuum. *Archives of Rational Mechanics Analysis,* Vol. 18, 1965, pp. 251–281.†

[28] Gurson, A. L., "Continuum Theory of Ductile Rupture by Void Nucleation and Growth: Part I– Yield Criteria and Flow Rules for Porous Ductile Media," *Journal of Engineering Materials and Technology,* Vol. 99, 1977, pp. 2–15.†

[29] Hallquist, J. O., "NIKE 2–D – A Vectorized, Implicit, Finite Deformation, Finite–Element Code for Analyzing the Static and Dynamic Response of 2–D Solids," *Lawrence Livermore Laboratory Report UCRL–52678*, 1984.§

[30] Hallquist, J. O., "NIKE 3–D – A Vectorized, Implicit, Finite Deformation, Finite–Element Code for Analyzing the Static and Dynamic Response of 3–D Solids." *Lawrence Livermore Laboratory Report UCID–18822*, 1984.§

[31] Healy, B., Pecknold, D. A. and Dodds, R., "Applications of Parallel and Vector Algorithms in Nonlinear Structural Dynamics Using the Finite Element Method," *Civil Engineering Studies,* SRS No. 571, UILU–ENG–92–2011, University of Illinois, Urbana, Illinois, 1992.§

[32] Hellen, T.K., "On the Method of Virtual Crack Extension," *International Journal for Numerical Methods in Engineering,* Vol. 9, 1975, pp. 187–207.†

[33] Hibbitt, Karlsson & Sorensen, Inc., *ABAQUS* Theory Manual, Version 5.3, Providence R.I., 1993.§

[34] Hibbitt, Karlsson & Sorensen, Inc., *ABAQUS–Explicit* Theory Manual, Version 5.2, Providence R.I., 1992.§

[35] Hinton, E, Rock, T., and Zienkiewicz, O. C., "A Note on Mass Lumping and Related Processes in the Finite Element Method," *Earthquake Engineering and Structural Dynamics,* Vol. 4, No. 3, 1976, pp 245–249.†

[36] Hoger, A. and Carlson, D. E., "Determination of the Stretch and Rotation in the Polar Decomposition of the Deformation Gradient," *Quarterly of Applied mathematics,* Vol. 42, 1984, pp. 113–117.†

[37] Hughes, T.J.R., "Stability, Convergence, and Growth and Decay of Energy of the Average Acceleration Method in Nonlinear Structural Dynamics," *Computers and Structures,* Vol. 6, 1976, pp. 313–324.†

[38] Hughes, T. J. "Generalization of Selective Integration Procedures to Anisotropic and Nonlinear Media," *International Journal for Numerical Methods in Engineering*, Vol. 15, 1980, pp. 1413–1418.†

[39] Hughes, T. J. and Winget, J., "Finite Rotation Effects in Numerical Integration of Rate Constitutive Equations Arising in Large–Deformation Analysis," *International Journal for Numerical Methods in Engineering*, Vol. 15, 1980, pp. 1862–1867.†

[40] Hughes, T. J., Levit, I., and Winget, J. M., "An Element–By–Element Solution Algorithm for Problems of Structural and Solid Mechanics," *Computer Methods in Applied Mechanics and Engineering,* Vol. 36, 1983, pp. 241–254.†

[41] Hughes T.J.R., Ferencz R.M., and Hallquist J.O., "Large–scale Vectorized Implicit Calculations in Solid, Mechanics on a Cray X–MP/48 Utilizing EBE Preconditioned Conjugate Gradients," *Computer Methods in Applied Mechanics and Engineering,* Vol. 61, 1987, pp. 215–248.†

[42] Hughes T.J.R., *The Finite Element Method.* Prentice–Hall, Englewood–Cliffs, New Jersey, 1987.†

[43] Hutchinson, J., "Singular Behavior at the End of a Tensile Crack in a Hardening Material," *Journal of the Mechanics and Physics of Solids,* Vol. 16, 1968, pp. 13–31.†

[44] Jaumann, G. "Geschlossenes System Physikalischer Und Chemisher Differentialgesefze," *Sitz Zer. Akad. Wiss. Wein,* (IIa) 120, 1911, pp. 385.†

[45] Johnson, G. C. and Bammann, D. J., "A Discussion of Stress Rates in Finite Deformation Problems," *International Journal for Solids and Structures*, Vol. 20, 1984, pp. 725–737.†

[46] Keppel, M. and Dodds, R. H., "Improved Numerical Techniques for Plasticity Computations in Finite–Element Analysis," *Computers and Structures*, Vol. 36, No. 1, 1990, pp. 183–185.†

[47] Key, S. W. and Krieg, R. D., "On the Numerical Implementation of Inelastic Time Dependent and Time Independent, Finite Strain Constitutive Equations in Structural Mechanics," *Computer Methods in Applied Mechanics and Engineering,* Vol. 33, 1982, pp. 439–452.†

[48] Kishimoto, K., Aoki, S., and Sakata, M., "On the Path Independent $\hat{J}$ –Integral," *Engineering Fracture Mechanics,* Vol. 13, 1980, pp. 841–850.†

[49] Kojic, M. and Bathe, K. J., "Studies of Finite–element Procedures: Stress Solution of a Closed Elastic Strain Path with Stretching and Shearing Using the Updated Lagrangian Jaumann Formulation," *Computers and Structures*, Vol. 26, 1987, pp. 175–179.†

[50] Lee, E. H., "Elastic–Plastic Deformation at Finite Strain," *Journal of Applied Mechanics*, Vol. 36, 1969, pp. 1–6.†

[51] Knowles, J., and Sternberg, E., "On a Class of Conservation Integrals," *Archives of Rational Mechanics Analysis,* Vol. 44, 1972, pp. 187–211.†

[52] Krieg, R. D. and Key, S. W., "Implementation of a Time Independent Plasticity Theory into Structural Computer Programs," In *Constitutive Equations in Viscoplasticity: Computational and Engineering Aspects, AMD–20* (Edited by J. A. Stricklin and K. J. Saczalski), ASME, New York, 1976, pp. 125–138.†

[53] Li, F.Z., Shih, C.F., and Needleman, A., "A Comparison of Methods for Calculating Energy Release Rates," *Engineering Fracture Mechanics,* Vol. 21, (1985), 405–421.†

[54] Malvern, L., *An Introduction to the Mechanics of Continuous Media.* Prentice–Hall, Englewood–Cliffs, New Jersey, 1969.†

[55] Marsden, J. E.. and Hughes, T. J. R., *Mathematical Foundations of Elasticity.* Prentice–Hall, Englewood–Cliffs, New Jersey, 1983.†

[56] Moran, B., and Shih, C.F., "A General Treatment of Crack Tip Contour Integrals," *International Journal of Fracture,* Vol. 35, 1987, pp. 295–310.†

[57] Moran, B., and Shih, C.F., "Crack Tip and Associated Domain Integrals from Momentum and Energy Balance," *Engineering Fracture Mechanics,* Vol. 27, 1987, pp. 615–642.†

[58] Moran, B., Ortiz, M., and Shih, C. F., "Formulation of Implicit Finite Element Methods for Multiplicative Finite Deformation Plasticity," *International Journal for Numerical Methods in Engineering,* Vol. 29, 1990, pp. 483–514.†

[59] Nagtegaal, J. C. Parks, D. M., and Rice, J. R., "On Numerically Accurate Finite Element Solutions in the Fully Plastic Range," *Computer Methods in Applied Mechanics and Engineering,* Vol. 4, 1974, pp. 153–178.†

[60] Nagtegaal, J. C. and de Jong, J. E., "Some Computational Aspects of Elastic–Plastic, Large Strain Analysis," *International Journal for Numerical Methods in Engineering,* Vol. 12, 1981, pp. 15–41.†

[61] Nagtegaal, J. C., "On the Implementation of Inelastic Constitutive Equations with Special Reference to Large Deformations," *Computer Methods in Applied Mechanics and Engineering,* Vol. 33, 1982, pp. 221–245.†

[62] Nagtegaal, J. C. and Veldpaus, F. E., "On the Implementation of Finite Strain Plasticity Equations in a Numerical Model," In *Numerical Analysis of Forming Processes* (edited by J.F. Pittman, O. C. Ziekiewicz, R. D. Wood and J. M. Alexander), p. 351. John Wiley and Sons, New York, 1984.†

[63] Needleman, A. and Tvergaard, V., "An Analysis of Ductile Rupture Modes at a Crack Tip," *Journal of Mechanics and Physics of Solids,* Vol. 35, 1987, pp. 151–183.†

[64] Newmark N.M., "A Method of Computation for Structural Dynamics," *Journal of the Engineering Mechanics Division,* ASCE, Vol. 32, No. EM3, 1959, pp. 67–94.†

[65] Nikishkov, G.P. and Atluri, S.N., "Calculation of Fracture Mechanics Parameters for an Arbitrary Three–Dimensional Crack, by the 'Equivalent Domain Integral' Method," *International Journal for Numerical Methods in Engineering,* Vol. 24, 1987, pp. 1801–1827.†

[66] Parks, D.M., "The Virtual Crack Extension Method for Nonlinear Material Behavior," *Computer Methods in Applied Mechanics and Engineering,* Vol. 12, 1977, pp. 353–364.†

[67] Pinsky, P. M., Ortiz, M. and Pister, K. S. "Numerical Integration of Rate Constitutive Equations in Finite Deformation Analysis," *Computer Methods in Applied Mechanics and Engineering*, Vol. 40, 1983, pp. 137–158.†

[68] Rice, J., and Rosengren, G.F., "Plane Strain Deformation Near a Crack Tip in a Power Law Hardening Material," *Journal of Mechanics and Physics of Solids,* Vol. 16, 1968, pp. 1–12.†

[69] Rice, J. "A Path Independent Integral and the Approximate Analysis of Strain Concentration by Notches and Cracks," *Journal of Applied Mechanics,* Vol. 35, 1968, pp. 379–386. †

[70] Roy, S., Fossum, A. F., and Dexter, R. J., "On the Use of Polar Decomposition in the Integration of Hypo–Elastic Constitutive Laws," *International Journal of Engineering Science,* Vol. 30, 1992, pp. 119–133.†

[71] Rubinstein, R. and Atluri, S. N., "Objectivity of Incremental Constitutive Equations Over Finite Time Steps in Computational Finite Deformation Analysis," *Computer Methods in Applied Mechanics and Engineering,* Vol. 36, 1983, pp. 277–290.†

[72] Schoeberle, D.F., and Belytschko, T., "On the Unconditional Stability of an Implicit Algorithm for Nonlinear Structural Dynamics," *Journal of Applied Mechanics,* Vol. 42, 1975, pp. 865–869.†

[73] Schreyer H.L., Kulak R.F., and Kramer J.M., "Accurate Numerical Solutions for Elastic–Plastic Models," *Journal of Pressure Vessel Technology,* Vol. 101, 1979, pp. 226–234.†

[74] Shih, C.F., Moran, B., and Nakamura, T. "Energy Release Rate Along a Three–Dimensional Crack Front in a Thermally Stressed Body," *International Journal of Fracture,* Vol. 30, 1986, pp. 79–102.†

[75] Simo, J.C. and Taylor, R.L., "Consistent Tangent Operators for Rate Independent Elastoplasticity," *Computer Methods in Applied Mechanics and Engineering,* Vol. 35, 1985, pp. 101–118.†

[76] Simo, J. C. and Ortiz, M., "A Unified Approach to Finite Deformation Elasto–Plastic Analysis Based on the Use of Hyper–elastic Constitutive Equations," *Computer Methods in Applied Mechanics and Engineering,* Vol. 49, 1985, pp. 221–245.†

[77] Simo, J. C. and Hughes, T. J. R. *Elastoplasticity and Viscoplasticity: Computational Aspects.* Stanford University, 1988.§

[78] Taylor, L. M. and Flanagan, D. P., "PRONTO 2D, A Two–Dimensional Transient Solid Dynamics Program," *SAND86–0594, Sandia National Laboratories*, Albuquerque, NM., 1987.§

[79] Taylor, L. M. and Flanagan, D. P., "PRONTO 3D, A Three–Dimensional Transient Solid Dynamics Program," *SAND87–1912, Sandia National Laboratories*, Albuquerque, NM., 1989.§

[80] Tvergaard, V., "Influence of Void Nucleation on Ductile Shear Fracture at a Free Surface," *Journal of Mechanics and Physics of Solids,* Vol. 30, 1982, pp. 399–425.†

[81] Tvergaard, V., "Material Failure by Void Growth to Coalescence," *Advances in Applied Mechanics,* Vol. 27, 1990, pp. 83–151.†

[82] Wang, Y., "A Two–Parameter Characterization of Elastic–Plastic Crack Tip Fields and Applications to Cleavage Fracture," Ph. D. Dissertation, Dept. of Mechanical Engineering, MIT, 1991.§

[83] Zienkiewicz, O. C. and Taylor, R. L. *The Finite Element Method.* Fourth Edition, Volume 1, *Basic Formulation and Linear Problems.* McGraw–Hill, London, 1990.†

[84] Zienkiewicz, O. C. and Taylor, R. L. *The Finite Element Method.* Fourth Edition, Volume 2, *Solid and Fluid Mechanics, Dynam,ics and Nonlinearity.* McGraw–Hill, London, 1991.†

---

* Available for purchase from national Technical Information Service, Springfield, VA 22161.

† Available from public technical libraries.

‡ Copies are available from U.S. Government Printing Office, Washington, D.C. 20402. ATTN: Regulatory Guide Account.

§ Available for purchase from vendor.